

УТВЕРЖДЕН

643.72410666.00067-07 98 01-ЛУ

**ЗАЩИЩЕННАЯ СИСТЕМА УПРАВЛЕНИЯ  
БАЗАМИ ДАННЫХ «ЈАТОВА»**

**Руководство по настройке. Часть 16.  
Обеспечение работы с СУБД Oracle**

**643.72410666.00067-07 98 01-16**

Листов 141

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

2024

Литера О<sub>1</sub>

## АННОТАЦИЯ

В документе приведены сведения, необходимые для установки и эксплуатации компонентов, обеспечивающих работу с СУБД «Oracle», таких как:

- компонент совместимости с СУБД Oracle (OraFCE);
- компонент совместимости с системой глобальных переменных СУБД Oracle (pg\_Variables);
- компонент для работы с таблицами Oracle как с внешними таблицами (Oracle\_FDW).

Настоящее руководство предназначено для администраторов СУБД.



Все примеры в данном документе приведены для СУБД «Jatoba» версии ядра 4.x, для других версий все шаги выполняются аналогично, разница состоит в именах директорий.

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

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

Для СУБД «Jatoba» версии ядра 4 используются следующие версии компонент: OraFCE – 3.22.0, pg\_Variables – 1.2.0, Oracle\_FDW – 1.2.0.

Для СУБД «Jatoba» версии ядра 5/6 используются следующие версии компонент: OraFCE – 4.3.0, pg\_Variables – 1.2.5, Oracle\_FDW – 1.2.0



### **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.....	8
1.1. Условия применения.....	9
2. Установка и настройка.....	10
2.1. Установка компонентов на ОС Windows .....	10
2.2. Установка компонентов ОС GNU/Linux .....	12
2.3. Установка расширений .....	14
2.3.1. Установка расширения «orafce».....	14
2.3.2. Установка расширения «pg_variables».....	14
2.3.1. Установка расширения «oracle_fdw».....	14
3. Функциональные возможности компонента «orafce» .....	15
3.1. Функции работы с датой.....	15
3.1.1. Функция «add_months».....	15
3.1.2. Функция «last_day» .....	15
3.1.3. Функция «next_day» .....	16
3.1.4. Функция «oracle.months_between» .....	18
3.1.5. Функция «oracle.to_date».....	18
3.1.6. Функция «oracle.sysdate».....	20
3.1.7. Функция «oracle.dbtimezone» .....	21
3.1.8. Функция «oracle.sessiontimezone» .....	21
3.1.9. Функция «oracle.to_char» .....	22
3.2. Операторы для работы с датой .....	23
3.2.1. Оператор «oracle.+» .....	23
3.2.2. Оператор «oracle.-».....	24
3.3. Функции модуля «PLVdate» .....	26
3.3.1. Функция «plvdate.default_holidays».....	26
3.3.2. Функция «plvdate.nearest_bizday» .....	27
3.3.3. Функция «plvdate.next_bizday» .....	27
3.3.4. Функция «plvdate.prev_bizday» .....	28
3.3.5. Функция «plvdate.add_bizdays».....	29
3.3.6. Функция «plvdate.bizdays_between».....	29
3.3.7. Функция «plvdate_isbizday» .....	30
3.3.8. Функция «plvdate.set_nonbizday» .....	31
3.3.9. Функция «plvdate.unset_nonbizday» .....	32
3.3.10. Функция «plvdate.set_nonbizday» .....	32
3.3.11. Функция «plvdate.unset_nonbizday» .....	33
3.3.12. Функция «plvdate.use_easter» .....	34
3.3.13. Функция «plvdate.unuse_easter» .....	35

3.3.14. Функция «plvdate.using_easter» .....	36
3.3.15. Функция «plvdate.use_great_friday» .....	36
3.3.16. Функция «plvdate.include_start» .....	37
3.3.17. Функция «plvdate.noinclude_start».....	38
3.3.18. Функция «plvdate.including_start» .....	38
3.4. Функции модуля «PLVstr» и «PLVchr» .....	39
3.4.1. Функция «plvstr.normalize» .....	39
3.4.2. Функция «plvstr.is_prefix».....	40
3.4.3. Функция «plvstr.substr» .....	41
3.4.4. Функция «plvstr.instr».....	42
3.4.5. Функция «plvstr.lpart».....	45
3.4.6. Функция «plvstr.rpart» .....	49
3.4.7. Функция «plvstr.rvrs».....	53
3.4.8. Функция «plvstr.left» .....	55
3.4.9. Функция «plvstr.right» .....	56
3.4.10. Функция «plvstr.swap».....	56
3.4.11. Функция «plvstr.betwn» .....	58
3.4.12. Функция «plvchr.nth».....	60
3.4.13. Функция «plvchr.first» .....	61
3.4.14. Функция «plvchr.last» .....	61
3.4.15. Функция «plvchr.is_blank».....	62
3.4.16. Функция «plvchr.is_digit» .....	62
3.4.17. Функция plvchr.is_other .....	63
3.4.18. Функция «plvchr.is_letter» .....	65
3.4.19. Функция «plvchr.char_name».....	66
3.4.20. Функция «plvchr.quoted1» .....	66
3.4.21. Функция «plvchr.quoted2» .....	67
3.4.22. Функция «plvchr.stripped» .....	68
3.5. Функции модуля «PLVsubst».....	68
3.5.1. Функция «plvsubst.string».....	69
3.5.2. Функция «plvsubst.setsubst» .....	71
3.5.3. Функция «plvsubst.subst».....	71
3.6. Функции модуля «DBMS_random» .....	72
3.6.1. Функция «dbms_random.initialize» .....	72
3.6.2. Функция «dbms_random.normal».....	72
3.6.3. Функция «dbms_random.random».....	73
3.6.4. Функция «dbms_random.seed».....	73
3.6.5. Функция «dbms_random.string».....	74
3.6.6. Функция dbms_random.value.....	75

3.7. Дополнительные функции .....	77
3.7.1. Функция «oracle.substr».....	77
3.7.2. Функция «oracle.lpad» .....	80
3.7.3. Функция «oracle.rpad» .....	81
3.7.4. Функция «oracle.ltrim».....	82
3.7.5. Функции «oracle.rtrim» .....	83
3.7.6. Функция «oracle.btrim».....	84
3.7.7. Функция «oracle.length» .....	85
3.7.8. Функция «oracle.to_number» .....	86
3.7.9. Функция «oracle.mod» .....	87
3.8. Механизм DBMS_PIPE .....	87
3.8.1. Пример реализации автономных транзакции (компонент Orafce, схема dbms_pipe).....	89
4. Функциональные возможности компонента pg_Variables.....	93
4.1. Создание и использование нетранзакционных переменных .....	93
4.2. Создание и использование транзакционных переменных .....	94
4.3. Создание и использование скалярной переменной.....	97
4.4. Создание и использование переменных типа запись.....	98
4.5. Создание и использование переменных типа массив .....	103
4.6. Восстановление удаленной транзакционной переменной .....	105
4.7. Вывод занимаемой памяти.....	107
4.8. Удаление переменной .....	108
4.9. Удаление всех пакетов и переменных .....	110
4.10. Проверка на существование пакета .....	112
5. Функциональные возможности компонента Oracle_FDW .....	115
5.1. Добавление внешней таблицы Oracle в СУБД «Jatoba».....	115
5.2. Использование компонента .....	120
5.2.1. Необходимые права в Oracle.....	120
5.2.2. Соединения.....	120
5.2.3. Таблицы .....	121
5.2.4. Типы данных.....	121
5.2.5. Операторы WHERE и ORDER BY .....	122
5.2.6. Использование оператора JOIN с внешними таблицами .....	123
5.2.7. Изменение данных внешней таблицы.....	124
5.2.8. Оператор EXPLAIN .....	125
5.2.9. Оператор ANALYZE .....	125
5.2.10. Поддержка операции IMPORT FOREIGN SCHEMA .....	125
6. Удаление компонентов .....	128
6.1. Удаление компонентов при отсутствии зависимых от него объектов .....	128
6.2. Удаление компонента при наличии зависимых от него объектов .....	129

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

Приложение 1 .....	130
Перечень сокращений.....	140

## 1. НАЗНАЧЕНИЕ КОМПОНЕНТА

Компонент «OraFCE» выполнен в виде расширения и имеет дополнительные функции и операторы для работы текстовыми и временными строками. Может применяться для миграции данных из Oracle в СУБД «Jatoba».

Все функции полностью совместимы с Oracle и учитывают все известные строки формата. Формат строк приведен в таблице 1.1.

Таблица 1.1 – Строки формата

Формат	Обозначение (eng)	Обозначение (ru)
Y, YY, YYY, YYYY, SYYY, SYEAR	year	год
I, IY, IYY, IYYY	iso year	система дат
Q	quarter	квартал
WW	week, day as first day of year	неделя, день как первый день года
IW	week, beginning Monday	неделя, начинающаяся с понедельника
W	week, day as first day of month	неделя, день как первый день месяца
DAY, DY, D	first day of week, sunday	первый день недели, воскресенье
MONTH, MON, MM, RM	Month	месяц
CC, SCC	Century	век
DDD, DD, J	day	день
HH, HH12, HH24	hour	час
MI	minute	минуты

Компонент «pg\_Variables» выполнен в виде расширения и предназначен для работы с переменными различных типов. Созданные переменные существуют только в рамках текущей пользовательской сессии.

Компонент «Oracle\_FDW» дает возможность создать обертку (Foreign-Data Wrapper, FDW) для доступа к базе данных Oracle.

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



## 1.1. Условия применения

Компонент «OraFCE» может использоваться совместно с СУБД «Jatoba» версий 1.x и выше, под управлением ОС GNU/Linux.

Встроенные функции Oracle date были протестированы на соответствие требованиям Oracle 10.

Диапазоны дат с 1960 по 2070 год работают корректно. Дата до 1100-03-01 не может быть проверена из-за ошибки в Oracle.

Компонент «pg\_Variables» может использоваться совместно с СУБД «Jatoba» версий 1.x и выше, под управлением ОС GNU/Linux.

Компонент «Oracle\_FDW» может использоваться совместно с СУБД «Jatoba» версий 1.x и выше, под управлением ОС GNU/Linux и ОС семейства Windows. Для работы с расширением необходимо установить:

- для ОС GNU/Linux — Oracle Instance Client версии 12.2 и выше;
- для ОС семейства Windows — Oracle Instance Client версии 12.2 и выше.



### Важная информация

В ОС семейства Windows директорию установки Oracle Instance Client необходимо добавить в переменную среды Path.

Для работы компонента «Oracle\_FDW» на ОС семейства Windows необходимо дополнительно установить распространяемый пакет Microsoft Visual C++ для Visual Studio 2013 (Microsoft Visual C++ 2013).

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 компонентов на ОС Windows

Компоненты устанавливаются из пакета JatobaInstaller-X.XX.X-XXXX 1 .msi, находящегося на дистрибутивном диске. При выборе полной установки он установится автоматически.

В случае выборочной установки потребуется:

а) в окне «Выбор типа установки» следует выбрать тип установки «Выборочная» (см. рис. 2.1);

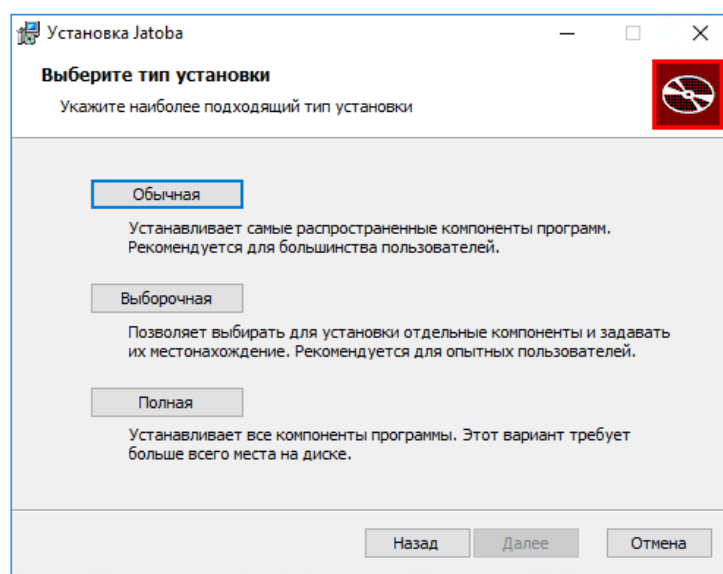


Рисунок 2.1 – Окно выбора типа установки

б) в окне «Выборочная установка», выбрать «Интеграция с СУБД Oracle (orafce и oracle\_fdw)» (см. рис. 2.2) и (или) «Поддержка переменных сессий (pg\_variables)»;

<sup>1</sup> Версия уточняется при поставке продукта

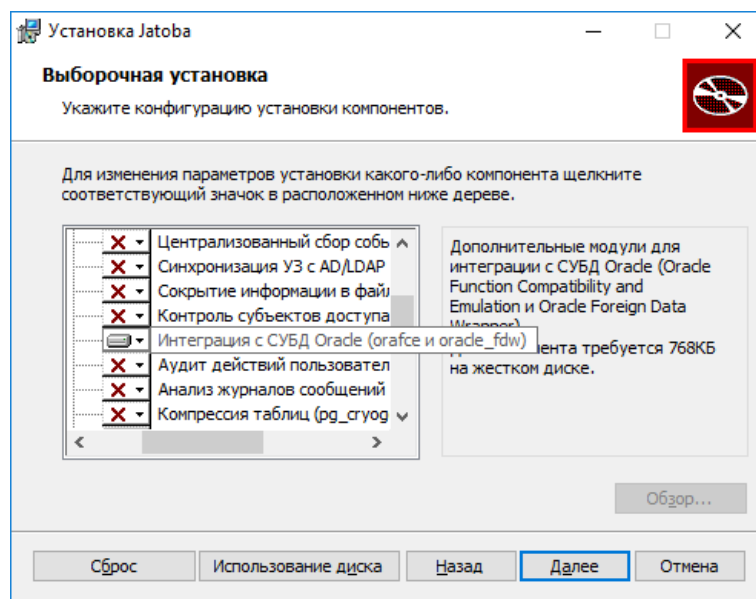


Рисунок 2.2 – Выбор устанавливаемых компонент

в) в открывшемся окне «Все готово к установке Jatoba» запустить процесс установки, нажав кнопку «Установить» (см. рис. 2.3);

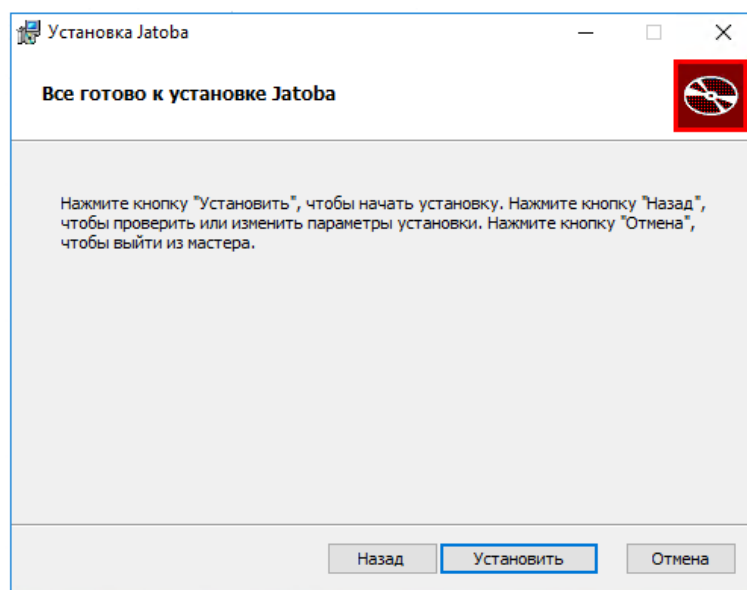


Рисунок 2.3 – Окно «Все готово к установке Jatoba»

## 2.2. Установка компонентов ОС GNU/Linux

Компоненты устанавливаются в составе СУБД «Jatoba». Их возможно установить при первичной установке, либо доустановить.

Установку компонент 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.

Компоненты выполнены в виде отдельного deb или rpm-пакета. Установка компонент 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 компонента «Oracle\_FDW» следующая:

```
apt-get install jatoba4-oracle-fdw
```

- команда установки компонента «OraFCE» следующая:

```
apt-get install jatoba4-orafce
```

- команда установки компонента «pg\_Variables» следующая:

```
apt-get install jatoba4-pg-variables
```

– 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 компонента «Oracle\_FDW» следующая:

```
yum install jatoba4-oracle_fdw
```

- команда установки компонента «OraFCE» следующая:

```
yum install jatoba4-orafce
```

- о команда установки компонента «pg\_Variables» следующая:

```
yum install jatoba4-pg_variables
```

Отдельного уточнения требуют операционные системы ALT Linux и openSUSE.

- ALT Linux использует пакетный менеджер APT, но распространяется в виде rpm-пакетов:

```
apt-get install jatoba4-oracle_fdw  
apt-get install jatoba4-orafce  
apt-get install jatoba4-pg_variables
```

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

```
zypper install jatoba4-oracle_fdw  
zypper install jatoba4-orafce  
zypper install jatoba4-pg_variables
```

Установка компонентов в составе других версий СУБД «Jatoba» осуществляется аналогично. Отличие будет только в номере версии СУБД, в составе которой он распространяется. Например, jatoba5-oracle-fdw и т.п.



Для корректной работы компонента oracle\_fdw требуется установка дополнительной библиотеки libnsl. Выполнить установку можно стандартными средствами операционной системы, используя пакетный менеджер.

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

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

## 2.3. Установка расширений

### 2.3.1. Установка расширения «orafce»

Расширение устанавливается от имени и с правами привилегированного пользователя при помощи SQL-команды:

```
CREATE EXTENSION orafce;
```

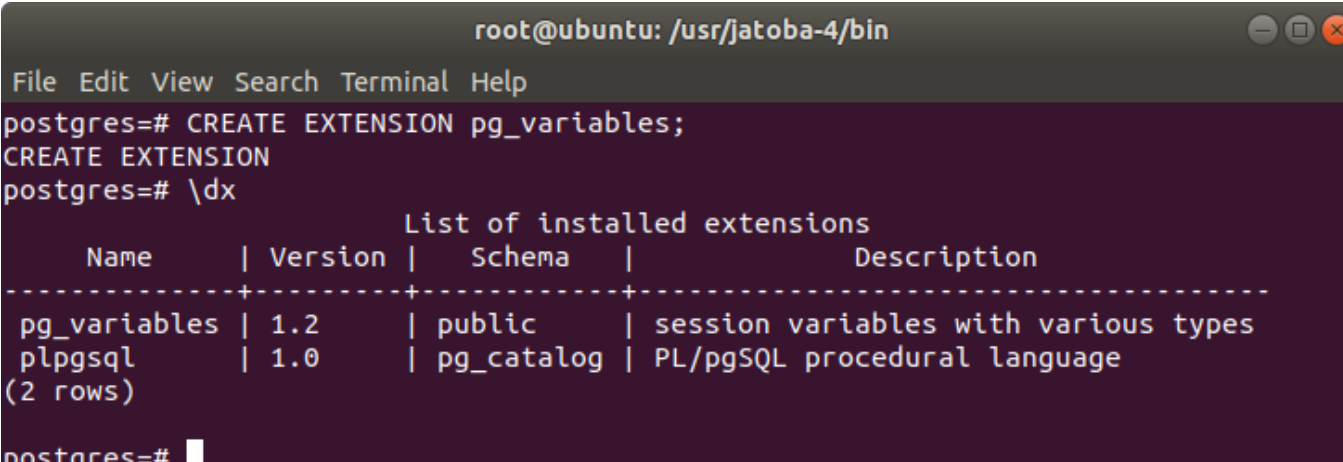
### 2.3.2. Установка расширения «pg\_variables»

Расширение устанавливается от имени и с правами привилегированного пользователя при помощи SQL-команды:

```
CREATE EXTENSION pg_variables;
```

Убедиться, что расширение установлено возможно SQL-командой:

```
\dx
```



```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# CREATE EXTENSION pg_variables;
CREATE EXTENSION
postgres=# \dx
                                List of installed extensions
   Name   | Version | Schema  | Description
-----+-----+-----+-----
pg_variables | 1.2    | public  | session variables with various types
plpgsql   | 1.0    | pg_catalog | PL/pgSQL procedural language
(2 rows)

postgres=#
```

Рисунок 2.4 – Установка расширения pg\_variables и вывод списка установленных расширений

### 2.3.1. Установка расширения «oracle\_fdw»

Расширение устанавливается от имени и с правами привилегированного пользователя при помощи SQL-команды:

```
CREATE EXTENSION oracle_fdw;
```

### 3. ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ КОМПОНЕНТА «OraFCE»

#### 3.1. Функции работы с датой

##### 3.1.1. Функция «add\_months»

Функция прибавляет определенное количество месяцев к заданной дате.

```
add_months(date, integer) date
```

Функция применяется с параметрами, приведенными в таблице 3.1.

Таблица 3.1 – Параметры функции «add\_months»

Параметр	Тип данных	Обозначение
#1	date	заданная дата
#2	integer	количество месяцев для добавления

Например, требуется прибавить к дате 1 месяц:

```
add_months(date '2022-06-19',1) -> 2022-07-19
```

SQL-запрос будет выглядеть следующим образом:

```
SELECT      oracle.add_months(event_time::date,1)      from  
security_event where id_event = 1;
```

```
postgresdb=# select oracle.add_months(event_time::date, 1) from security_event where id_event = 1;  
add_months  
-----  
2022-07-19  
(1 строка)
```

Рисунок 3.1 – Пример выполнения SQL-запроса с использованием функции «add\_months»

##### 3.1.2. Функция «last\_day»

Функция возвращает последний день месяца заданной даты.

```
last_day(date '') date
```

Функция применяется с параметрами, приведенными в таблице 3.2.

Таблица 3.2 – Параметры функции «last\_day»

Параметр	Тип данных	Обозначение
#1	date	заданная дата

Допустим, что текущая дата 2022-06-24 и требуется вычислить последний день месяца.

```
last_day(date '2022-06-24') -> 2022-06-30
```

В этом случае SQL-запрос будет следующим:

```
SELECT      oracle.last_months(event_time::date,1)      from
security_event where id_event = 1;
```

```
postgresdb=# select oracle.last_day(event_time::date) from security_event where id_event = 1;
 last_day
-----
2022-06-30
(1 строка)
```

Рисунок 3.2 – Пример выполнения SQL-запроса с использованием функции «last\_day»

### 3.1.3. Функция «next\_day»

Функция возвращает заданный день недели, следующий после заданной даты.

```
next_day(date, text) date
```

Функция применяется с параметрами, приведенными в таблице 3.3.

Таблица 3.3 – Параметры функции «next\_day(date, text) date»

Параметр	Тип данных	Обозначение
#1	date	заданная дата
#2	text	название дня недели

SQL-запрос будет следующим:

```
SELECT next_day(date '2005-05-24', 'monday');
```



Данный SQL-запрос можно интерпретировать как: «Верни мне дату, которая будет понедельником после указанной даты» или как «Верни мне дату дня недели заданного вторым параметром после даты заданной первым параметром»:

```
SELECT oracle.next_day(event_time::date, 'monday') from
security_event where id_event = 1;
```

```
postgresdb=# select oracle.next_day(event_time::date, 'monday') from security_event where id_event = 1;
 next_day
-----
2022-06-20
(1 строка)
```

Рисунок 3.3 – Пример выполнения SQL-запроса с использованием функции «next\_day(date, text) date»

Функция возвращает заданный день недели, следующий после заданной даты, используя номер дня недели.

```
next_day(date, integer) date
```

Функция применяется с параметрами, приведенными в таблице 3.4.

Таблица 3.4 – Параметры функции «next\_day(date, integer) date»

Параметр	Тип данных	Обозначение
#1	date	заданная дата
#2	integer	порядковый номер дня недели (начиная с воскресенья)

SQL-запрос будет следующим:

```
SELECT      oracle.next_day(event_time::date,      2)      from
security_event where id_event = 1;
```

Данный SQL-запрос можно интерпретировать как: «Верни мне дату, которая будет первым днем недели (понедельником), после указанной даты».

```
postgresdb=# select oracle.next_day(event_time::date, 2) from security_event where id_event = 1;
next_day
-----
2022-06-20
(1 строка)
```

Рисунок 3.4 – Пример выполнения SQL-запроса с использованием функции «next\_day(date, integer) date»

### 3.1.4. Функция «oracle.months\_between»

Функция возвращает количество календарных месяцев между двумя датами. В том случае, если между датами проходит не ровное количество месяцев, дробная часть возвращаемого числа будет вычислена на основе 31-дневного месяца.

```
oracle.months_between(timestamp with time zone, timestamp with
time zone)
```

Функция применяется с параметрами, приведенными в таблице 3.5.

Таблица 3.5 – Параметры функции «oracle.months\_between»

Параметр	Тип данных	Обозначение
#1	timestamp	первая заданная дата
#2	timestamp	вторая заданная дата

SQL-запрос будет следующим:

```
SELECT oracle.months_between(event_time::date '2022-01-
19'::date) from security_event where id_event =1;
```

```
postgresdb=# select oracle.months_between(event_time::date, '2022-01-19'::date) from
security_event where id_event = 1;
months_between
-----
5
(1 строка)
```

Рисунок 3.5 – Пример выполнения SQL-запроса с использованием функции «oracle.months\_between»

### 3.1.5. Функция «oracle.to\_date»

Функция позволяет преобразовывать:

- текстовую строку в тип данных «timestamp without time zone»;
- текстовую строку в тип данных «oracle.date».

### 3.1.5.1 Преобразование в тип данных «timestamp without time zone»

Функция преобразовывает текстовую строку в тип данных «timestamp without time zone». В качестве второго параметра указывается шаблон, по которому записано время в изначальной строке.

```
oracle.to_date(text, text)
```

Функция применяется с параметрами, приведенными в таблице 3.6.

Таблица 3.6 – Параметры функции «oracle.to\_date»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	формат, в котором записано время в изначальной строке

Допустим есть текстовая строка 07/12/22 04:12:12, с датой в формате месяц (ММ), день (DD), год (YY) и временем в формате час (НН24), минуты(ММ) и секунды(SS).

В SQL-запросе указываем:

- дату в текстовом формате;

```
'07/12/22 04:12:12'
```

- шаблон даты в текстовом формате;

```
'MM/DD/YY HH24:MI:SS'
```

SQL-запрос будет следующим:

```
SELECT oracle.to_date('07/12/22 04:12:12', 'MM/DD/YY  
HH24:MI:SS');
```

```
postgresdb=# select oracle.to_date('07/12/22 04:12:12', 'MM/DD/YY HH24:MI:SS');  
to_date  
-----  
2022-07-12 04:12:12  
(1 строка)
```

Рисунок 3.6 – Пример выполнения SQL-запроса с использованием функции «oracle.date»

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

### 3.1.5.2 Преобразование в тип данных «oracle.date»

Функция преобразовывает текстовую строку в тип данных «oracle.date». В данном случае изначальную строку необходимо вводить в определенном формате.

```
oracle.to_date(text)
```

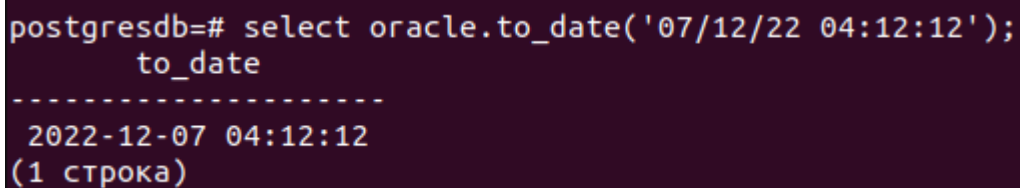
Функция применяется с параметрами, приведенными в таблице 3.7.

Таблица 3.7 – Параметры функции «oracle.to\_date»

Параметр	Тип данных	Обозначение
#1	text	заданная строка

SQL-запрос будет следующим:

```
SELECT oracle.to_date('07/12/22 04:12:12');
```



```
postgresdb=# select oracle.to_date('07/12/22 04:12:12');
to_date
-----
2022-12-07 04:12:12
(1 строка)
```

Рисунок 3.7 – Пример выполнения SQL-запроса с использованием функции «oracle.to\_date»

При выполнении SQL-запроса были введены текстовые данные:

```
'07/12/22 04:12:12'
```

и вывод данных в формате «oracle.date»:

```
2022-12-07 04:12:12'
```

### 3.1.6. Функция «oracle.sysdate»

Функция возвращает значение времени в часовом поясе сервера в формате «timestamp».

```
oracle.sysdate()
```

SQL-запрос будет следующим:

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

```
SELECT oracle.sysdate();
```

```
postgresdb=# select oracle.sysdate();
          sysdate
-----
2022-07-12 13:16:01
(1 строка)
```

Рисунок 3.8 – Пример выполнения SQL-запроса с использованием функции

### 3.1.7. Функция «oracle.dbtimezone»

Функция возвращает часовой пояс сервера.

```
oracle.dbtimezone()
```

SQL-запрос будет следующим:

```
SELECT oracle.dbtimezone();
```

```
postgresdb=# select oracle.dbtimezone();
          dbtimezone
-----
GMT
(1 строка)
```

Рисунок 3.9 – Пример выполнения SQL-запроса с использованием функции  
«oracle.dbtimezone»

### 3.1.8. Функция «oracle.sessiontimezone»

Функция возвращает часовой пояс сеанса, который указан как текущий часовой пояс СУБД.

```
oracle.sessiontimezone()
```

SQL-запрос будет следующим:

```
SELECT oracle.sessiontimezone();
```

```
postgresdb=# select oracle.sessiontimezone();
sessiontimezone
-----
Europe/Moscow
(1 строка)
```

Рисунок 3.10 – Пример выполнения SQL-запроса с использованием функции «oracle.sessiontimezone»

### 3.1.9. Функция «oracle.to\_char»

Функция возвращает значение заданного времени в формате «nls\_date\_format».

```
oracle.to_char(timestamp)
```

Функция применяется с параметрами, приведенными в таблице 3.8.

Таблица 3.8 – Параметры функции «oracle.to\_char»

Параметр	Тип данных	Обозначение
#1	timestamp	заданная дата

SQL-запрос будет следующим:

```
SELECT oracle.to_char(oracle.to_date('12072022 12:13:44+05:30',
'DDMMYYYY HH24:MI:SS'));
```

```
postgresdb=# select oracle.to_char(oracle.to_date('12072022 12:13:44+05:30',
'DDMMYYYY HH24:MI:SS'));
to_char
-----
22-Jul12 12:13:44
(1 строка)
```

Рисунок 3.11 – Пример выполнения SQL-запроса с использованием функции «oracle.to\_char»

При выполнении SQL-запроса были введены текстовые данные в формате «nls\_date\_format»:

```
'12072022 12:13:44+05:30'
```

и вывод данных в формате «to\_char»:

'2022-12-07 04:12:12'

## 3.2. Операторы для работы с датой

### 3.2.1. Оператор «oracle.+»

Оператор позволяет добавить к заданной дате некоторое количество дней.

```
oracle.+(oracle.date,smallint)
oracle.+(oracle.date,integer)
```

Оператор применяется с параметрами, приведенными в таблице 3.9.

Таблица 3.9 – Параметры оператора «oracle.+(oracle.date,smallint)»

Параметр	Тип данных	Обозначение
#1	date	заданная дата
#2	integer/ smallint	количество дней, которое необходимо добавить

SQL-запрос будет следующим:

```
SELECT (oracle.add_months(event_time::date, 1) + 1::smallint)
as add_day from security_event where id_event = 1;
```

```
postgresdb=# select (oracle.add_months(event_time::date, 1) + 1::smallint)
as add_day from security_event where id_event = 1;
 add_day
-----
2022-07-20
(1 строка)
```

Рисунок 3.12 – Пример выполнения SQL-запроса с использованием оператора «oracle.+(oracle.date,smallint)»

SQL-запрос будет следующим:

```
SELECT (oracle.add_months(event_time::date, 1) + 1::integer) as
add_day from security_event where id_event = 1;
```

```
postgresdb=# select (oracle.add_months(event_time::date, 1) + 1::integer)
as add_day from security_event where id_event = 1;
 add_day
-----
2022-07-20
(1 строка)
```

Рисунок 3.13 – Пример выполнения SQL-запроса с использованием оператора «oracle.+(oracle.date,integer)»

### 3.2.2. Оператор «oracle.-»

Оператор позволяет:

- отнять от заданной даты некоторое количество дней;
- вычесть из одной даты другую.

#### 3.2.2.1 Вычитание из даты дней

Оператор позволяет отнять от заданной даты некоторое количество дней.

```
oracle.-(oracle.date, smallint)
oracle.-(oracle.date, integer)
```

Оператор применяется с параметрами, приведенными в таблице 3.10.

Таблица 3.10 – Параметры функции «oracle.-»

Параметр	Тип данных	Обозначение
#1	date	заданная дата
#2	integer/ smallint	количество дней, которое необходимо вычесть

SQL-запрос для типа данных «smallint» будет следующим:

```
SELECT (oracle.add_months(event_time::date, 1) - 2::smallint)
as sub_day from security_event where id_event = 1;
```

```
postgresdb=# select (oracle.add_months(event_time::date, 1) - 2::smallint)
as sub_day from security_event where id_event = 1;
 sub_day
-----
2022-07-17
(1 строка)
```



Рисунок 3.14 – Пример выполнения SQL-запроса с использованием оператора «oracle.-»

SQL-запрос для типа данных «integer» будет следующим:

```
SELECT (oracle.add_months(event_time::date, 1) - 2::integer) as  
sub_day from security_event where id_event = 1;
```

```
postgresdb=# select (oracle.add_months(event_time::date, 1) - 2::integer)  
as sub_day from security_event where id_event = 1;  
sub_day  
-----  
2022-07-17  
(1 строка)
```

Рисунок 3.15 – Пример выполнения SQL-запроса с использованием оператора «oracle.-»

### 3.2.2.2 Вычитание дат

Оператор позволяет вычесть из одной даты другую. Возвращает число двойной точности.

```
oracle.- (oracle.date, oracle.date)
```

Оператор применяется с параметрами, приведенными в таблице 3.11.

Таблица 3.11 – Параметры оператора «oracle.-»

Параметр	Тип данных	Обозначение
#1	date	заданная дата
#2	date	дата, которую необходимо вычесть

SQL-запрос будет следующим:

```
SELECT (oracle.add_months(event_time::date, 1) -  
oracle.to_date('2021-06-11 10:00:00', yyyy-mm-dd hh24:mi:ss'))  
as sub_day from security_event where id_event = 1;
```

```
postgresdb=# select (oracle.add_months(event_time::date, 1) - oracle.to_date('2021-06-11
10:00:00', 'yyyy-mm-dd hh24:mi:ss')) as sub_day from security_event where id_event = 1;
sub_day
-----
402 days 14:00:00
(1 строка)
```

Рисунок 3.16 – Пример выполнения SQL-запроса с использованием оператора

«oracle.-»

### 3.3. Функции модуля «PLVdate»

Данный модуль предоставляет функции для работы с рабочими, выходными и праздничными днями. На данный момент существуют конфигурации для следующих стран: Чехия, Германия, Австрия, Польша, Словакия, Россия, Великобритания и США. Однако модуль позволяет и самостоятельную настройку рабочих или нерабочих дней.

#### 3.3.1. Функция «plvdate.default\_holidays»

Функция позволяет установить конфигурацию по умолчанию из существующего списка активных для выбора стран.

```
plvdate.default_holidays (varchar)
```

Функция применяется с параметрами, приведенными в таблице 3.12.

Таблица 3.12 – Параметры функции «plvdate.default\_holidays»

Параметр	Тип данных	Обозначение
#1	varchar	заданная конфигурация для модуля

SQL-запрос будет следующим:

```
SELECT plvdate.default_holiday('Russia');
```

```
postgresdb=# select plvdate.default_holidays('Russia');
default_holidays
-----
(1 строка)
```

Рисунок 3.17 – Пример выполнения SQL-запроса с использованием функции «plvdate.default\_holidays»

### 3.3.2. Функция «plvdate.nearest\_bizday»

Функция возвращает ближайшую рабочую дату к указанной дате. Под «ближайшей» понимается дата, с которой минимальная разница в днях относительно указанной. Таким образом, функция может вернуть как предыдущую, так и последующую дату.

```
plvdate.nearest_bizday(day date)
```

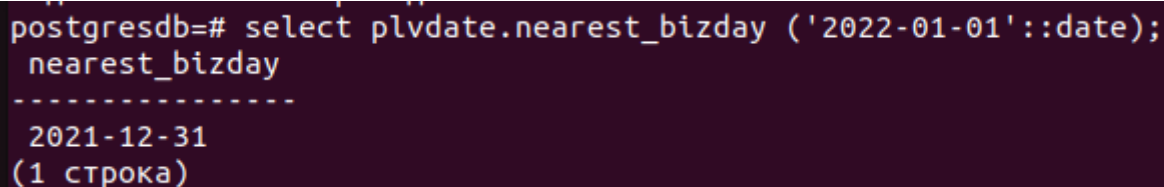
Функция применяется с параметрами, приведенными в таблице 3.13.

Таблица 3.13 – Параметры функции «plvdate.nearest\_bizday»

Параметр	Тип данных	Обозначение
#1	date	заданная дата

SQL-запрос будет следующим:

```
SELECT plvdate.nearest_bizday('2022-01-01'::date);
nearest_bizday
```



```
postgresdb=# select plvdate.nearest_bizday ('2022-01-01'::date);
nearest_bizday
-----
2021-12-31
(1 строка)
```

Рисунок 3.18 – Пример выполнения SQL-запроса с использованием функции «plvdate.nearest\_bizday»

### 3.3.3. Функция «plvdate.next\_bizday»

Функция возвращает следующую рабочую дату относительно указанной даты. В данном случае функция может вернуть только последующую дату.

```
plvdate.next_bizday(day date)
```

Функция применяется с параметрами, приведенными в таблице 3.14.

Таблица 3.14 – Параметры функции «plvdate.next\_bizday»

Параметр	Тип данных	Обозначение
#1	date	заданная дата

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

SQL-запрос будет следующим:

```
SELECT plvdate.next_bizday('2022-01-01'::date);  
next_bizday
```

```
postgresdb=# select plvdate.next_bizday ('2022-01-01'::date);  
next_bizday  
-----  
2022-01-06  
(1 строка)
```

Рисунок 3.19 – Пример выполнения SQL-запроса с использованием функции «plvdate.next\_bizday»

### 3.3.4. Функция «plvdate.prev\_bizday»

Функция возвращает предыдущую рабочую дату относительно указанной даты. В данном случае функция может вернуть только предшествующую дату.

```
plvdate.prev_bizday(day date)
```

Функция применяется с параметрами, приведенными в таблице 3.15.

Таблица 3.15 – Параметры функции «plvdate.prev\_bizday»

Параметр	Тип данных	Обозначение
#1	date	заданная дата

SQL-запрос будет следующим:

```
SELECT plvdate.prev_bizday('2022-01-03'::date);
```

```
postgresdb=# select plvdate.prev_bizday('2022-01-03'::date);  
prev_bizday  
-----  
2021-12-31  
(1 строка)
```

Рисунок 3.20 – Пример выполнения SQL-запроса с использованием функции «plvdate.prev\_bizday»

### 3.3.5. Функция «plvdate.add\_bizdays»

Функция возвращает дату, полученную путем прибавления к указанной дате некоторого количества рабочих дней.

```
plvdate.add_bizdays(day date, days int)
```

Функция применяется с параметрами, приведенными в таблице 3.16 .

Таблица 3.16 – Параметры функции «plvdate.add\_bizdays»

Параметр	Тип данных	Обозначение
#1	date	заданная дата
#2	integer	количество рабочих дней для добавления

SQL-запрос будет следующим:

```
SELECT plvdate.add_bizdays('2022-07-10'::date, 10);  
add_bizdays
```

```
postgresdb=# select plvdate.add_bizdays('2022-07-10'::date, 10);  
add_bizdays  
-----  
2022-07-22  
(1 строка)
```

Рисунок 3.21 – Пример выполнения SQL-запроса с использованием функции «plvdate.add\_bizdays»

### 3.3.6. Функция «plvdate.bizdays\_between»

Функция возвращает количество рабочих дней между двумя указанными датами.

```
plvdate.bizdays_between(day1 date, day2 date)
```

Функция применяется с параметрами, приведенными в таблице 3.17.

Таблица 3.17 – Параметры функции «plvdate.bizdays\_between»

Параметр	Тип данных	Обозначение
#1	date	начальная заданная дата

Параметр	Тип данных	Обозначение
#2	date	конечная заданная дата

SQL-запрос будет следующим:

```
SELECT plvdate.bizdays_between('2021-12-30'::date, '2022-01-10'::date);

bizdays_between
```

```
postgresdb=# select plvdate.bizdays_between('2021-12-30'::date, '2022-01-10'::date);
bizdays_between
-----
4
(1 строка)
```

Рисунок 3.22 – Пример выполнения SQL-запроса с использованием функции «plvdate.bizdays\_between»

### 3.3.7. Функция «plvdate\_isbizday»

Функция позволяет узнать является ли указанная дата рабочим днем. Если дата рабочая, будет возвращено значение «t», в ином случае будет возвращено «f».

```
plvdate_isbizday(date)
```

Функция применяется с параметрами, приведенными в таблице 3.18.

Таблица 3.18 – Параметры функции «plvdate\_isbizday»

Параметр	Тип данных	Обозначение
#1	date	заданная дата

SQL-запрос с выводом параметра рабочей даты будет следующим:

```
SELECT plvdate.isbizday ('2022-01-01'::date);
```

```
postgresdb=# select plvdate.isbizday ('2022-01-01'::date);
isbizday
-----
f
```

Рисунок 3.23 – Пример выполнения SQL-запроса с использованием функции «plvdate\_isbizday»

SQL-запрос с выводом параметра выходного дня будет следующим:

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

```
SELECT plvdate.isbizday ('2022-01-10'::date);
```

```
postgresdb=# select plvdate.isbizday ('2022-01-10'::date);
 isbizday
-----
 t
(1 строка)
```

Рисунок 3.24 – Пример выполнения SQL-запроса с использованием функции «plvdate\_isbizday»

### 3.3.8. Функция «plvdate.set\_nonbizday»

Функция позволяет установить нерабочим определенный день недели.

Выражение «DOW» можно интерпретировать, как «Day off week» и перевести «Выходной день в неделю»

```
plvdate.set_nonbizday(dow varchar)
```

Функция может быть полезной, если в отличии от производственного календаря требуется в календаре установить отметку о выходном дне.

Функция применяется с параметрами, приведенными в таблице 3.19.

Таблица 3.19 – Параметры функции «plvdate.set\_nonbizday»

Параметр	Тип данных	Обозначение
#1	varchar	заданный день недели

SQL-запрос будет следующим:

```
SELECT plvdate.set_nonbizday('saturday');
```

```
postgresdb=# select plvdate.set_nonbizday('saturday');
 set_nonbizday
-----
(1 строка)
```

Рисунок 3.25 – Пример выполнения SQL-запроса с использованием функции «plvdate.set\_nonbizday(dow varchar)»

### 3.3.9. Функция «plvdate.unset\_nonbizday»

Функция позволяет установить рабочим определенный день недели.

```
plvdate.unset_nonbizday(dow varchar)
```

Функция применяется с параметрами, приведенными в таблице 3.20.

Таблица 3.20 – Параметры функции «plvdate.unset\_nonbizday»

Параметр	Тип данных	Обозначение
#1	varchar	заданный день недели

SQL-запрос будет следующим:

```
SELECT plvdate.unset_nonbizday('saturday');
```

```
postgresdb=# select plvdate.unset_nonbizday('saturday');
unset_nonbizday
-----
(1 строка)
```

Рисунок 3.26 – Пример выполнения SQL-запроса с использованием функции

### 3.3.10. Функция «plvdate.set\_nonbizday»

Функция позволяет установить нерабочей определенную дату.

```
plvdate.set_nonbizday(day date)
plvdate.set_nonbizday(day date, repeat bool)
```

Функция применяется с параметрами, приведенными в таблице 3.21.

Таблица 3.21 – Параметры функции «plvdate.set\_nonbizday»

Параметр	Тип данных	Обозначение
plvdate.set_nonbizday(day date)		
#1	date	заданная дата
plvdate.set_nonbizday(day date, repeat bool)		
№ изменения: _____		Подпись отв. лица: _____
		Дата внесения изм: _____



Параметр	Тип данных	Обозначение
plvdate.set_nonbizday(day date)		
#1	date	заданная дата
#2	boolean	при указании этого параметра как true заданная дата станет рабочей каждый год

SQL-запросы будут следующими:

```
SELECT plvdate.set_nonbizday('2022-01-01'::date);
```

```
postgresdb=# select plvdate.set_nonbizday('2022-01-01'::date);
set_nonbizday
-----
(1 строка)
```

Рисунок 3.27 – Пример выполнения SQL-запроса с использованием функции «plvdate.set\_nonbizday(day date)»

```
SELECT plvdate.set_nonbizday('2022-11-04'::date, '1'::bool);
```

```
postgresdb=# select plvdate.set_nonbizday('2022-11-04'::date, '1'::bool);
set_nonbizday
-----
(1 строка)
```

Рисунок 3.28 – Пример выполнения SQL-запроса с использованием функции «plvdate.set\_nonbizday(day date, repeat bool)»

### 3.3.11. Функция «plvdate.unset\_nonbizday»

Функция позволяет установить рабочей определенную дату.

```
plvdate.unset_nonbizday(day date)
plvdate.unset_nonbizday(day date, repeat bool)
```

Функция применяется с параметрами, приведенными в таблице 3.22.

Таблица 3.22 – Параметры функции «plvdate.unset\_nonbizday»

Параметр	Тип данных	Обозначение
plvdate.unset_nonbizday(day date)		
#1	date	заданная дата
plvdate.unset_nonbizday(day date, repeat bool)		
#1	date	заданная дата
#2	boolean	при указании этого параметра как true заданная дата станет рабочей каждый год

SQL-запросы будут следующими:

```
SELECT plvdate.unset_nonbizday('2022-01-01'::date);
```

```
postgresdb=# select plvdate.unset_nonbizday('2022-01-01'::date);
unset_nonbizday
-----
(1 строка)
```

Рисунок 3.29 – Пример выполнения SQL-запроса с использованием функции

```
SELECT plvdate.unset_nonbizday('2022-11-04'::date, '1'::bool);
```

```
postgresdb=# select plvdate.unset_nonbizday('2022-11-04'::date, '1'::bool);
unset_nonbizday
-----
(1 строка)
```

Рисунок 3.30 – Пример выполнения SQL-запроса с использованием функции

### 3.3.12. Функция «plvdate.use\_easter»

Функция позволяет установить нерабочими днями пасхальное воскресенье и понедельник.

```
plvdate.use_easter()
plvdate.use_easter(useit boolean)
```

Функция применяется с параметрами, приведенными в таблице 3.23.

Таблица 3.23 – Параметры функции «plvdate.use\_easter»

Параметр	Тип данных	Обозначение
#1	boolean	включение/отключение режима

SQL-запросы будут следующими:

```
select plvdate.use_easter();
```

```
postgresdb=# select plvdate.use_easter();
use_easter
-----
(1 строка)
```

Рисунок 3.31 – Пример выполнения SQL-запроса с использованием функции «plvdate.use\_easter»

```
SELECT plvdate.use_easter('1'::bool);
```

```
postgresdb=# select plvdate.use_easter('1'::bool);
use_easter
-----
(1 строка)
```

Рисунок 3.32 – Пример выполнения SQL-запроса с использованием функции «plvdate.use\_easter»

### 3.3.13. Функция «plvdate.unuse\_easter»

Функция позволяет «отменить» установку пасхальных выходных.

```
plvdate.unuse_easter()
```

Пример запроса:

```
SELECT plvdate.unuse_easter();
```

```
postgresdb=# select plvdate.unuse_easter();
unuse_easter
-----
(1 строка)
```

Рисунок 3.33 – Пример выполнения SQL-запроса с использованием функции «plvdate.unuse\_easter»

### 3.3.14. Функция «plvdate.using\_easter»

Функция позволяет узнать используются ли в текущей конфигурации пасхальные выходные. Если пасхальные выходные используются будет возвращено значение «t», в ином случае будет возвращено «f».

```
plvdate.using_easter()
```

SQL-запрос будет следующим:

```
SELECT plvdate.using_easter();
```

```
postgresdb=# select plvdate.using_easter();
using_easter
-----
f
(1 строка)
```

Рисунок 3.34 – Пример выполнения SQL-запроса с использованием функции «plvdate.using\_easter»

### 3.3.15. Функция «plvdate.use\_great\_friday»

Функция позволяет установить предпасхальную пятницу нерабочим днем.

```
plvdate.use_great_friday()
```

SQL-запрос будет следующим:

```
SELECT plvdate.use_great_friday();
```

```
postgresdb=# select plvdate.use_great_friday();
use_great_friday
-----
(1 строка)
```

Рисунок 3.35 – Пример выполнения SQL-запроса с использованием функции «plvdate.use\_great\_friday»

### 3.3.16. Функция «plvdate.include\_start»

Функция позволяет включать начальную дату в расчеты при использовании функции «bizdays\_between».

```
plvdate.include_start()
plvdate.include_start(include boolean)
```

Функция применяется с параметрами, приведенными в таблице 3.24.

Таблица 3.24 – Параметры функции «plvdate.include\_start»

Параметр	Тип данных	Обозначение
#1	boolean	включение/отключение режима

SQL-запросы будут следующими:

```
SELECT plvdate.include_start();
```

```
postgresdb=# select plvdate.include_start();
include_start
-----
(1 строка)
```

Рисунок 3.36 – Пример выполнения SQL-запроса с использованием функции «plvdate.include\_start»

```
SELECT plvdate.include_start('1'::bool);
```

```
postgresdb=# select plvdate.include_start('1'::bool);
include_start
-----
(1 строка)
```

Рисунок 3.37 – Пример выполнения SQL-запроса с использованием функции «plvdate.include\_start»

### 3.3.17. Функция «plvdate.noinclude\_start»

Функция позволяет не включать начальную дату в расчеты при использовании функции «bizdays\_between».

```
plvdate.noinclude_start()
```

SQL-запрос будет следующим:

```
SELECT plvdate.noinclude_start();
```

```
postgresdb=# select plvdate.noinclude_start();
noinclude_start
-----
(1 строка)
```

Рисунок 3.38 – Пример выполнения SQL-запроса с использованием функции «plvdate.noinclude\_start»

### 3.3.18. Функция «plvdate.including\_start»

Функция позволяет узнать включается ли начальная дата в расчеты при использовании функции bizdays\_between при текущей конфигурации. Если начальная дата включается в расчеты будет возвращено значение «t», в ином случае будет возвращено «f».

```
plvdate.including_start()
```

SQL-запрос будет следующим:

```
SELECT plvdate.including_start();
```

```
postgresdb=# select plvdate.including_start();
including_start
-----
t
(1 строка)
```

Рисунок 3.39 – Пример выполнения SQL-запроса с использованием функции «plvdate.including\_start»

### 3.4. Функции модуля «PLVstr» и «PLVchr»

#### 3.4.1. Функция «plvstr.normalize»

Функция позволяет нормализовать строку, а именно заменить множественные пробелы, различные пробельные символы и знаки табуляции на одиночные пробелы.

```
plvstr.normalize(str text)
```

Функция применяется с параметрами, приведенными в таблице 3.25.

Таблица 3.25 – Параметры функции «plvstr.normalize»

Параметр	Тип данных	Обозначение
#1	text	заданная строка

SQL-запрос будет следующим:

```
select plvstr.normalize('some string    for    an    example
normalization    function ');

-----

some    stringfor an example normalization function
```

```
postgresdb=# select plvstr.normalize('some    string for an example normalization
function    ');
normalize
-----
some string for an example normalization function
(1 строка)
```

Рисунок 3.40 – Пример выполнения SQL-запроса с использованием функции «plvstr.normalize»

### 3.4.2. Функция «plvstr.is\_prefix»

Функция позволяет узнать является ли заданное значение префиксом заданной строки. Если строка содержит заданный префикс будет возвращено значение «t», т.е. «true», в ином случае будет возвращено «f», т.е. «false».

```
plvstr.is_prefix(str text, prefix text, cs bool)
plvstr.is_prefix(str text, prefix text)
plvstr.is_prefix(str int, prefix int)
plvstr.is_prefix(str bigint, prefix bigint)
```

Для SQL-выражения используются параметры, приведенные в таблице 3.26:

```
plvstr.is_prefix(str text, prefix text)
```

Таблица 3.26 – Параметры функции «plvstr.is\_prefix(str text, prefix text)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	префикс для поиска

Для SQL-выражения используются параметры, приведенные в таблице 3.27:

```
plvstr.is_prefix(str int, prefix int)
```

Таблица 3.27 – Параметры функции «plvstr.is\_prefix(str int, prefix int)»

Параметр	Тип данных	Обозначение
#1	integer	заданное число
#2	integer	префикс для поиска

Например, в строке требуется найти сочетание букв 'ambi'.

В этом случае SQL-запрос будет следующим:

```
SELECT plvstr.is_prefix('ambidextrous', 'ambi');
```



```
postgresdb=# select plvstr.is_prefix('ambidextrous', 'ambi');
is_prefix
-----
t
```

Рисунок 3.41 – Пример выполнения SQL-запроса с использованием функции «plvstr.is\_prefix»

Поскольку строка содержит заданный префикс 'ambi', будет возвращено значение «t», т.е. «true». Искомое значение найдено.

Аналогично строится поиск цифрового выражения «8921» в строке.

SQL-запрос строится следующим образом:

```
SELECT plvstr.is_prefix(89216538733, 8921);
```

```
postgresdb=# select plvstr.is_prefix(89216538733, 8921);
is_prefix
-----
t
(1 строка)
```

Рисунок 3.42 – Пример выполнения SQL-запроса с использованием функции «plvstr.is\_prefix»

Также, как и в предыдущем примере будет возвращено значение «t», т.е. «true». Следовательно, искомое значение найдено.

### 3.4.3. Функция «plvstr.substr»

Функция возвращает подстроку из указанной строки с определенной позиции.

```
plvstr.substr(str text, start int, len int)
plvstr.substr(str text, start int)
```

Для SQL-выражения используются параметры, приведенные в таблице 3.28:

```
plvstr.substr(str text, start int, len int)
```

Таблица 3.28 – Параметры функции «plvstr.substr(str text, start int, len int)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка

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

Параметр	Тип данных	Обозначение
#2	integer	начальная позиция
#3	integer	конечная позиция

Для SQL-выражения используются параметры, приведенные в таблице 3.29:

```
plvstr.substr(str text, start int)
```

Таблица 3.29 – Параметры функции «plvstr.substr(str text, start int)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	integer	начальная позиция

SQL-запросы будут следующими:

```
SELECT plvstr.substr('some string for an example substring  
function', 0, 11);
```

```
postgresdb=# select plvstr.substr('some string for an example substring function',
0, 11);
      substr
-----
some string
(1 строка)
```

Рисунок 3.43 – Пример выполнения SQL-запроса с использованием функции

```
SELECT plvstr.substr('some string for an example substring  
function', 28);
```

```
postgresdb=# select plvstr.substr('some string for an example substring function', 28);
      substr
-----
substring function
(1 строка)
```

Рисунок 3.44 – Пример выполнения SQL-запроса с использованием функции

#### 3.4.4. Функция «plvstr.instr»

Функция позволяет искать подстроку в заданной строке по определенным шаблонам. Возвращает номер позиции вхождения. Функция использует нижеперечисленный синтаксис SQL-запросов:

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

```
plvstr.instr(str text, patt text, start int, nth int)
plvstr.instr(str text, patt text, start int)
plvstr.instr(str text, patt text)
```

Возможно осуществлять:

- поиск подстроки в строке;
- поиск определенного вхождения подстроки в строке начиная с определенной позиции;
- поиск определенного вхождения подстроки в строке начиная с определенной позиции.

#### 3.4.4.1 Поиск подстроки в строке

Для поиска подстроки в строке, в SQL-запросе использует синтаксис команды:

```
plvstr.instr(str text, patt text)
```

Функция применяется с параметрами, приведенными в таблице 3.30.

Таблица 3.30 – Параметры функции «plvstr.instr(str text, patt text)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	подстрока для поиска

SQL-запрос будет следующим:

```
SELECT plvstr.instr('some string for an example instring
function', 'instring');
```

```
postgresdb=# select plvstr.instr('some string for an example instring function', 'instring');
instr
-----
      28
(1 строка)
```

Рисунок 3.45 – Пример выполнения SQL-запроса с использованием функции «plvstr.instr(str text, patt text)»

### 3.4.4.2 Поиск подстроки в строке, начиная с определенной позиции

Для поиска подстроки в строке начиная с определенной позиции, в SQL-запросе использует синтаксис команды:

```
plvstr.instr(str text, patt text, start int)
```

Функция применяется с параметрами, приведенными в таблице 3.31.

Таблица 3.31 – Параметры функции «plvstr.instr(str text, patt text, start int)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	подстрока для поиска
#3	integer	начальная позиция

SQL-запрос будет следующим:

```
SELECT plvstr.instr('some string for an example instring  
function', 'instring',30);
```

```
postgresdb=# select plvstr.instr('some string for an example instring function', 'instring',30)  
;  
instr  
-----  
      0  
(1 строка)
```

Рисунок 3.46 – Пример выполнения SQL-запроса с использованием функции «plvstr.instr(str text, patt text, start int)»

### 3.4.4.3 Поиск определенного вхождения подстроки в строке, начиная с определенной позиции

Для поиска определенного вхождения подстроки в строке, начиная с определенной позиции, в SQL-запросе использует синтаксис команды:

```
plvstr.instr(str text, patt text, start int, nth int)
```

Функция применяется с параметрами, приведенными в таблице 3.32.

Таблица 3.32 – Параметры функции «plvstr.instr(str text, patt text, start int, nth int)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	подстрока для поиска
#3	integer	начальная позиция
#4	integer	номер вхождения подстроки

SQL-запрос будет следующим:

```
SELECT plvstr.instr('some string for an example instring
function', 'i', 1, 3);
```

```
postgresdb=# select plvstr.instr('some string for an example instring function', 'i', 1, 3);
instr
-----
    33
(1 строка)
```

Рисунок 3.47 – Пример выполнения SQL-запроса с использованием функции «plvstr.instr(str text, patt text, start int, nth int)»

### 3.4.5. Функция «plvstr.lpart»

Функция возвращает левую часть заданной строки по определенным критериям и используется нижеперечисленный синтаксис SQL-запросов:

```
plvstr.lpart(str text, div text)
plvstr.lpart(str text, div text, start int, nth int,
all_if_notfound bool)
plvstr.lpart(str text, div text, start int, nth int)
plvstr.lpart(str text, div text, start int)
```

Функция возвращает левую часть заданной строки при:

- первом вхождении заданной подстроки (п. 3.4.5.1);
- первом вхождении заданной подстроки и поиском с заданной позиции (п. 3.4.5.2);
- определенном вхождении заданной подстроки и поиском с заданной позиции (п. 3.4.5.3);
- определенном вхождении заданной подстроки и поиском с заданной позиции, а также возвратом изначальной строки (п. 3.4.5.4).

### 3.4.5.1 Возврат левой части строки до первого вхождения заданной подстроки

Функция имеет функциональную возможность возврата левой части заданной строки до первого вхождения заданной подстроки. При этом синтаксис SQL запроса будет следующим:

```
plvstr.lpart(str text, div text)
```

Функция применяется с параметрами, приведенными в таблице 3.33.

Таблица 3.33 – Параметры функции «plvstr.lpart(str text, div text)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	подстрока для поиска

SQL-запрос будет следующим:

```
SELECT plvstr.lpart('some string for an example function',  
'an');
```

```
postgresdb=# select plvstr.lpart('some string for an example function', 'an');  
          lpart  
-----  
some string for  
(1 строка)
```

Рисунок 3.48 – Пример выполнения SQL-запроса с использованием функции «plvstr.lpart(str text, div text)»

### 3.4.5.2 Возврат левой части строки до первого вхождения заданной подстроки и поиском начинается с заданной позиции

Функция имеет функциональную возможность, возврата левой части заданной строки до первого вхождения заданной подстроки и поиском начинается с заданной позиции. При этом синтаксис SQL запроса будет следующим:

```
plvstr.lpart(str text, div text, start int)
```

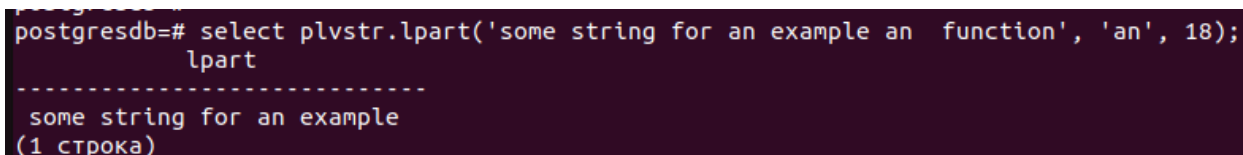
Функция применяется с параметрами, приведенными в таблице 3.34.

Таблица 3.34 – Параметры функции «plvstr.lpart(str text, div text, start int)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	подстрока для поиска
#3	integer	начальная позиция

SQL-запрос будет следующим:

```
SELECT plvstr.lpart('some string for an example an function',  
'an', 18);
```



```
postgres=# select plvstr.lpart('some string for an example an function', 'an', 18);
      lpart
-----
some string for an example
(1 строка)
```

Рисунок 3.49 – Пример выполнения SQL-запроса с использованием функции

### 3.4.5.3 Возврат левой часть строки до определенного вхождения заданной подстроки с поиском с заданной позиции

Функция имеет функциональную возможность, возврата левой части заданной строки до определенного вхождения заданной подстроки. При этом синтаксис SQL запроса будет следующим:

```
plvstr.lpart(str text, div text, start int, nth int)
```

Функция применяется с параметрами, приведенными в таблице 3.35.

Таблица 3.35 – Параметры функции «plvstr.lpart(str text, div text, start int, nth int)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	подстрока для поиска
#3	integer	начальная позиция
#4	integer	номер вхождения подстроки

SQL-запрос будет следующим:

```
SELECT plvstr.lpart('some string for an example an function',
'an', 1, 2);
```

```
postgresdb=# select plvstr.lpart('some string for an example an function', 'an', 1, 2);
          lpart
-----
some string for an example
(1 строка)
```

Рисунок 3.50 – Пример выполнения SQL-запроса с использованием функции

#### 3.4.5.4 Возврат левой части строки до определенного вхождения заданной подстроки и поиском с заданной позиции

Функция имеет функциональную возможность, возврата левой части заданной строки до определенного вхождения заданной подстроки. Если заданная подстрока не найдена будет возвращена изначальная строка (при значении последнего параметра true).

При этом синтаксис SQL-запроса будет следующим:

```
plvstr.lpart(str text, div text, start int, nth int,
all_if_notfound bool)
```

Функция применяется с параметрами, приведенными в таблице 3.36.

Таблица 3.36 – Параметры функции «plvstr.lpart(str text, div text, start int, nth int, all\_if\_notfound bool)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	подстрока для поиска

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



Параметр	Тип данных	Обозначение
#3	integer	начальная позиция
#4	integer	номер вхождения подстроки
#5	boolean	возвращать ли изначальную строку при ненахождении подстроки

SQL-запрос будет следующим:

```
SELECT plvstr.lpart('some string for an example an function',
'and', 1, 2, '1'::bool);
```

```
postgresdb=# select plvstr.lpart('some string for an example an function', 'and', 1, 2,
'1'::bool);
          lpart
-----
some string for an example an function
(1 строка)
```

Рисунок 3.51 – Пример выполнения SQL-запроса с использованием функции plvstr.lpart(str text, div text, start int, nth int, all\_if\_notfound bool)

### 3.4.6. Функция «plvstr.rpart»

Функция возвращает правую часть заданной строки по определенным критериям и использует нижеперечисленный синтаксис SQL-запросов:

```
plvstr.rpart(str text, div text, start int, nth int,
all_if_notfound bool)
plvstr.rpart(str text, div text, start int, nth int)
plvstr.rpart(str text, div text, start int)
plvstr.rpart(str text, div text)
```

Функция возвращает правую часть заданной строки при:

- первом вхождении заданной подстроки (п. 3.4.6.1);
- первом вхождении заданной подстроки и поиском с заданной позиции (п. 3.4.6.2);
- определении вхождения заданной подстроки и поиском с заданной позиции (п. 3.4.6.3);

- определенном вхождении заданной подстроки и поиском с заданной позиции, а также возвратом изначальной строки (п. 3.4.6.4).

#### 3.4.6.1 Возврат правой части строки до первого вхождения заданной подстроки

Функция имеет функциональную возможность возврата правой части заданной строки до первого вхождения заданной подстроки. При этом синтаксис SQL запроса будет следующим:

```
plvstr.rpart(str text, div text)
```

Функция применяется с параметрами, приведенными в таблице 3.37.

Таблица 3.37 – Параметры функции «plvstr.rpart(str text, div text)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	подстрока для поиска

SQL-запрос будет следующим:

```
SELECT plvstr.rpart('some string for an example function',  
'an');
```

```
postgresdb=# select plvstr.rpart('some string for an example function', 'an');  
          rpart  
-----  
n example function  
(1 строка)
```

Рисунок 3.52 – Пример выполнения SQL-запроса с использованием функции «plvstr.rpart(str text, div text)»

#### 3.4.6.2 Возврат правой части строки до первого вхождения заданной подстроки и поиском с заданной позиции

Функция имеет функциональную возможность возврата правой части заданной строки до первого вхождения заданной подстроки. Поиск начинается с заданной позиции. При этом синтаксис SQL запроса будет следующим:

```
plvstr.rpart(str text, div text, start int)
```

Функция применяется с параметрами, приведенными в таблице 3.38.

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

Таблица 3.38 – Параметры функции «plvstr.rpart(str text, div text, start int)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	подстрока для поиска
#3	integer	начальная позиция

SQL-запрос будет следующим:

```
SELECT plvstr.rpart('some string for an example an function',
'an', 20);
```

```
postgresdb=# select plvstr.rpart('some string for an example an function', 'an', 20);
rpart
-----
n function
(1 строка)
```

Рисунок 3.53 – Пример выполнения SQL-запроса с использованием функции «plvstr.rpart(str text, div text, start int)»

### 3.4.6.3 Возврат правой части строки до определенного вхождения заданной подстроки и поиском с заданной позиции

Функция имеет функциональную возможность, возврата правой части заданной строки до определенного вхождения заданной подстроки. Поиск начинается с заданной позиции. При этом синтаксис SQL запроса будет следующим:

```
plvstr.rpart(str text, div text, start int, nth int)
```

Функция применяется с параметрами, приведенными в таблице 3.39.

Таблица 3.39 – Параметры функции «plvstr.rpart(str text, div text, start int, nth int)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	подстрока для поиска
#3	integer	начальная позиция
#4	integer	номер вхождения подстроки

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

SQL-запрос будет следующим:

```
SELECT plvstr.rpart('some string for an example an function',
'an', 1, 2);
```

```
postgresdb=# select plvstr.rpart('some string for an example an function', 'an', 1, 2);
 rpart
-----
n function
(1 строка)
```

Рисунок 3.54 – Пример выполнения SQL-запроса с использованием функции «plvstr.rpart(str text, div text, start int, nth int)»

#### 3.4.6.4 Возврат правой части строки до определенного вхождения заданной подстроки и поиском с заданной позиции

Функция имеет функциональную возможность возврата правой части заданной строки до определенного вхождения заданной подстроки. Если заданная подстрока не найдена будет возвращена изначальная строка (при значении последнего параметра true).

При этом синтаксис SQL запроса будет следующим:

```
plvstr.rpart(str text, div text, start int, nth int,
all_if_notfound bool)
```

Функция применяется с параметрами, приведенными в таблице 3.40.

Таблица 3.40 – Параметры функции «plvstr.rpart(str text, div text, start int, nth int, all\_if\_notfound bool)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	подстрока для поиска
#3	integer	начальная позиция
#4	integer	номер вхождения подстроки
#5	boolean	возвращать ли изначальную строку при ненахождении подстроки

SQL-запрос будет следующим:

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

```
SELECT plvstr.rpart('some string for an example an function',
'and', 1, 2, '1'::bool);
```

```
postgresdb=# select plvstr.rpart('some string for an example an function', 'and', 1, 2,
'1'::bool);
          rpart
-----
some string for an example an function
(1 строка)
```

Рисунок 3.55 – Пример выполнения SQL-запроса с использованием функции «plvstr.rpart(str text, div text, start int, nth int, all\_if\_notfound bool)»

### 3.4.7. Функция «plvstr.rvrs»

Функция возвращает перевернутую заданную строку или часть заданной строки по определенным критериям и использует нижеперечисленный синтаксис SQL-запросов:

```
plvstr.rvrs(str text, start int, _end int)
plvstr.rvrs(str text, start int)
plvstr.rvrs(str text)
```

Функция «plvstr.rvrs» имеет функциональные возможности:

- возврата перевернутой строки (п. 3.4.7.1);
- возврата перевернутой части строки с определенной позиции (п. 3.4.7.2);
- возврат первой части строки, находящуюся между заданными позициями (п. 3.4.7.3).

#### 3.4.7.1 Возврат перевернутой строки

Функция возвращает перевернутую строку, используя синтаксис SQL-команды:

```
plvstr.rvrs(str text, start int, _end int)
```

Функция применяется с параметрами, приведенными в таблице 3.41.

Таблица 3.41 – Параметры функции «plvstr.rvrs(str text, start int, \_end int)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка

SQL-запрос будет следующим:

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

```
SELECT plvstr.rvrs('reverse string');
```

```
postgresdb=# select plvstr.rvrs('reverse string');
      rvrs
-----
gnirts esrever
(1 строка)
```

Рисунок 3.56 – Пример выполнения SQL-запроса с использованием функции «plvstr.rvrs(str text, start int, \_end int)»

### 3.4.7.2 Возврат перевернутой части строки с определенной позиции

Функция возвращает перевернутую часть строки, начиная с определенной позиции используя синтаксис SQL-команды:

```
plvstr.rvrs(str text, start int)
```

Функция применяется с параметрами, приведенными в таблице 3.42.

Таблица 3.42 – Параметры функции «plvstr.rvrs(str text, start int)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	integer	начальная позиция

SQL-запрос будет следующим:

```
SELECT plvstr.rvrs('reverse string', 8);
```

```
postgresdb=# select plvstr.rvrs('reverse string', 8);
      rvrs
-----
gnirts
(1 строка)
```

Рисунок 3.57 – Пример выполнения SQL-запроса с использованием функции «plvstr.rvrs(str text, start int)»

### 3.4.7.3 Возврат первой части строки, находящуюся между заданными позициями

Функция возвращает перевернутую часть строки, находящуюся между заданными позициями используя синтаксис SQL-команды:

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

```
plvstr.rvrs(str text, start int, _end int)
```

Функция применяется с параметрами, приведенными в таблице 3.43:

Таблица 3.43 – Параметры функции «plvstr.rvrs(str text, start int, \_end int)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	integer	начальная позиция
#3	integer	конечная позиция

SQL-запрос будет следующим:

```
SELECT plvstr.rvrs('reverse string', 3, 10);
```

```
postgresdb=# select plvstr.rvrs('reverse string', 3, 10);
rvrs
-----
ts esrev
(1 строка)
```

Рисунок 3.58 – Пример выполнения SQL-запроса с использованием функции «plvstr.rvrs(str text, start int, \_end int)»

### 3.4.8. Функция «plvstr.left»

Функция возвращает подстроку из заданной строки, находящуюся перед указанной позицией используя синтаксис SQL-команды:

```
plvstr.left(str text, n int)
```

Функция применяется с параметрами, приведенными в таблице 3.44.

Таблица 3.44 – Параметры функции «plvstr.left(str text, n int)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	integer	начальная позиция

SQL-запрос будет следующим:

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

```
SELECT plvstr.left('left string', 4);
```

```
postgresdb=# select plvstr.left('left string', 4);
left
-----
left
(1 строка)
```

Рисунок 3.59 – Пример выполнения SQL-запроса с использованием функции «plvstr.left(str text, n int)»

### 3.4.9. Функция «plvstr.right»

Функция возвращает подстроку из заданной строки, находящуюся после указанной позиции используя синтаксис SQL-команды:

```
plvstr.right(str text, n int)
```

Функция применяется с параметрами, приведенными в таблице 3.45.

Таблица 3.45 – Параметры функции «plvstr.right(str text, n int)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	integer	начальная позиция

SQL-запрос будет следующим:

```
SELECT plvstr.right('right string', 6);
```

```
postgresdb=# select plvstr.right('right string', 6);
right
-----
string
(1 строка)
```

Рисунок 3.60 – Пример выполнения SQL-запроса с использованием функции «plvstr.right(str text, n int)»

### 3.4.10. Функция «plvstr.swap»

Функция заменяет подстроку в строке на заданную подстроку по определенным критериям и использует нижеперечисленный синтаксис SQL-запросов:

```
plvstr.swap(str text, replace text, start int, length int)
```

Имя пользователя:	Имя сервера:	Дата выполнения:
-------------------	--------------	------------------



```
plvstr.swap(str text, replace text)
```

Функция «plvstr.swap» имеет функциональные возможности:

- замены на подстроку до первого пробела в строке (п. 3.4.10.1);
- замены на подстроку с определенной позиции и на определенную длину (п. 3.4.10.2).

### 3.4.10.1 Замена на подстроку до первого пробела в строке

Функция заменяет на подстроку при нахождении первого пробела в строке. При этом используется синтаксис SQL-запроса:

```
plvstr.swap(str text, replace text)
```

Функция применяется с параметрами, приведенными в таблице 3.46.

Таблица 3.46 – Параметры функции «plvstr.swap(str text, replace text)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	подстрока для замены

SQL-запрос будет следующим:

```
SELECT plvstr.swap('some string for an example function',  
'swap');
```

```
postgresdb=# select plvstr.swap('some string for an example function', 'swap');
          swap
-----
swap string for an example function
(1 строка)
```

Рисунок 3.61 – Пример выполнения SQL-запроса с использованием функции «plvstr.swap(str text, replace text)»

### 3.4.10.2 Замена на подстроку с определенной позиции и на определенную длину.

Функция заменяет на подстроку происходит с определенной позиции и на определенную длину. При этом используется синтаксис SQL-запроса:

```
plvstr.swap(str text, replace text, start int, length int)
```

Функция применяется с параметрами, приведенными в таблице 3.47.

Таблица 3.47 – Параметры функции «plvstr.swap(str text, replace text, start int, length int)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	подстрока для замены
#3	integer	начальная позиция
#4	integer	длина участка, который необходимо заменить

SQL-запрос будет следующим:

```
SELECT plvstr.swap('some string for an example function',
'swap', 6, 6);
```

```
postgresdb=# select plvstr.swap('some string for an example function', 'swap', 6, 6);
          swap
-----
some swap for an example function
(1 строка)
```

Рисунок 3.62 – Пример выполнения SQL-запроса с использованием функции «plvstr.swap(str text, replace text, start int, length int)»

### 3.4.11. Функция «plvstr.betwn»

Функция возвращает подстроку из заданной строки, находящуюся между определенными позициями или символами. Функция использует нижеперечисленный синтаксис SQL-запросов:

```
plvstr.betwn(str text, start int, _end int)
plvstr.betwn(str text, start text, _end text)
```

#### 3.4.11.1 Возврат подстроки из заданной строки находящейся между позициями

Функция возвращает подстроку из заданной строки, находящуюся между определенными позициями. Позиция определяется целыми числом.

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

```
plvstr.betwn(str text, start int, _end int)
```

Функция применяется с параметрами, приведенными в таблице 3.48.

Таблица 3.48 – Параметры функции «plvstr.betwn(str text, start int, \_end int)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	integer	начальная позиция
#3	integer	конечная позиция

SQL-запрос будет следующим:

```
SELECT plvstr.betwn('some string for an example function', 5,  
12);
```

```
postgresdb=# select plvstr.betwn('some string for an example function', 5, 12);  
betwn  
-----  
string  
(1 строка)
```

Рисунок 3.63 – Пример выполнения SQL-запроса с использованием функции «plvstr.betwn(str text, start int, \_end int)»

#### 3.4.11.2 Возврат подстроки из заданной строки находящейся между символами

Функция возвращает подстроку из заданной строки, находящуюся между определенными символами. Позиция определяется текстовыми символами.

```
plvstr.betwn(str text, start text, _end text)
```

Функция применяется с параметрами, приведенными в таблице 3.49.

Таблица 3.49 – Параметры функции «plvstr.betwn(str text, start text, \_end text)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	начальная позиция (текстом)
#3	text	конечная позиция (текстом)

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

SQL-запрос будет следующим:

```
SELECT plvstr.betwn('some string for an example function',
'some', 'for');
```

```
postgresdb=# select plvstr.betwn('some string for an example function', 'some', 'for');
      betwn
-----
some string for
(1 строка)
```

Рисунок 3.64 – Пример выполнения SQL-запроса с использованием функции «plvstr.betwn(str text, start text, \_end text)»

### 3.4.12. Функция «plvchr.nth»

Функция возвращает символ, находящийся на определенной позиции в заданной строке, используя синтаксис SQL-запроса:

```
plvchr.nth(str text, n int)
```

Функция применяется с параметрами, приведенными в таблице 3.50.

Таблица 3.50 – Параметры функции «plvchr.nth(str text, n int)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	integer	позиция в строке

SQL-запрос будет следующим:

```
SELECT plvchr.nth('some string', 2);
```

```
postgresdb=# select plvchr.nth('some string', 2);
      nth
-----
o
(1 строка)
```

Рисунок 3.65 – Пример выполнения SQL-запроса с использованием функции «plvchr.nth(str text, n int)»

### 3.4.13. Функция «plvchr.first»

Функция возвращает первый символ в заданной строке, используя синтаксис SQL-запроса:

```
plvchr.first(str text)
```

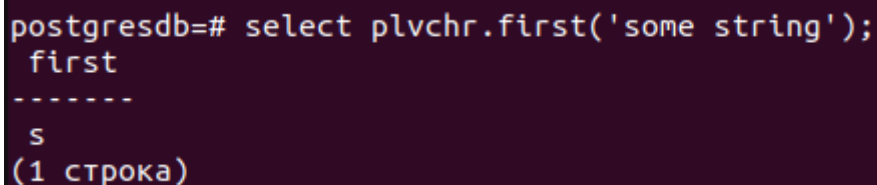
Функция применяется с параметрами, приведенными в таблице 3.51.

Таблица 3.51 – Параметры функции «plvchr.first(str text)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка

SQL-запрос будет следующим:

```
SELECT plvchr.first('some string');
```



```
postgresdb=# select plvchr.first('some string');
first
-----
s
(1 строка)
```

Рисунок 3.66 – Пример выполнения SQL-запроса с использованием функции «plvchr.first(str text)»

### 3.4.14. Функция «plvchr.last»

Функция возвращает последний символ в заданной строке.

```
plvchr.last(str text)
```

Функция применяется с параметрами, приведенными в таблице 3.52.

Таблица 3.52 – Параметры функции «plvchr.last(str text)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка

SQL-запрос будет следующим:

```
SELECT plvchr.last('some string');
```

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

```
postgresdb=# select plvchr.last('some string');
last
-----
g
(1 строка)
```

Рисунок 3.67 – Пример выполнения SQL-запроса с использованием функции «plvchr.last(str text)»

### 3.4.15. Функция «plvchr.is\_blank»

Функция определяет является ли заданная строка пустой. Если строка пустая будет возвращено значение «t» (true), в ином случае будет возвращено «f» (false). Функция использует нижеперечисленный синтаксис SQL-запросов:

```
plvchr.is_blank(c int)
plvchr.is_blank(c text)
```

Функция применяется с параметрами, приведенными в таблице 3.53.

Таблица 3.53 – Параметры функции «plvchr.is\_blank»

Параметр	Тип данных	Обозначение
#1	integer	заданное число
#2	text	заданная строка

SQL-запрос будет следующим:

```
SELECT plvchr.is_blank(' ');
```

```
postgresdb=# select plvchr.is_blank(' ');
is_blank
-----
t
```

Рисунок 3.68 – Пример выполнения SQL-запроса с использованием функции «plvchr.is\_blank»

### 3.4.16. Функция «plvchr.is\_digit»

Функция определяет является ли заданное значение символом числа. Если значение является символом числа будет возвращено значение «t» (true), в ином случае будет возвращено «f» (false). Функция использует нижеперечисленный синтаксис SQL-запросов:

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

```
plvchr.is_digit(c int)
plvchr.is_digit(c text)
```

Функция применяется с параметрами, приведенными в таблице 3.54.

Таблица 3.54 – Параметры функции «plvchr.is\_digit»

Параметр	Тип данных	Обозначение
#1	integer	заданное число
#2	text	заданный символ

SQL-запросы будут следующими:

```
SELECT plvchr.is_digit('4');
```

```
postgresdb=# select plvchr.is_digit('4');
 is_digit
-----
 t
(1 строка)
```

Рисунок 3.69 – Пример выполнения SQL-запроса с использованием функции «plvchr.is\_digit»

```
SELECT plvchr.is_digit(4);
```

```
postgresdb=# select plvchr.is_digit(4);
 is_digit
-----
 f
(1 строка)
```

Рисунок 3.70 – Пример выполнения SQL-запроса с использованием функции «plvchr.is\_digit»

### 3.4.17. Функция plvchr.is\_other

Функция определяет является ли заданное значение «иным» символом (не число и не буква). Если значение является «иным» символом будет возвращено значение «t», в ином случае будет возвращено «f». Функция использует нижеперечисленный синтаксис SQL-запросов:

```
plvchr.is_other(c int)
plvchr.is_other(c text)
```

Функция применяется с параметрами, приведенными в таблице 3.55.

Таблица 3.55 – Параметры функции «plvchr.is\_other»

Параметр	Тип данных	Обозначение
plvchr.is_other(c int)		
#1	integer	заданное число
plvchr.is_other(c text)		
#1	text	заданный символ

SQL-запросы будут следующими:

```
SELECT plvchr.is_other('*');
```

```
postgresdb=# select plvchr.is_other('*');
is_other
-----
t
(1 строка)
```

Рисунок 3.71 – Пример выполнения SQL-запроса с использованием функции «plvchr.is\_other»

```
SELECT plvchr.is_other(4);
```

```
postgresdb=# select plvchr.is_other(4);
is_other
-----
f
(1 строка)
```

Рисунок 3.72 – Пример выполнения SQL-запроса с использованием функции «plvchr.is\_other»



### 3.4.18. Функция «plvchr.is\_letter»

Функция определяет является ли заданное значение символом буквы. Если значение является символом буквы будет возвращено значение «t» (true), в ином случае будет возвращено «f» (false). Функция использует нижеперечисленный синтаксис SQL-запросов:

```
plvchr.is_letter(c int)
plvchr.is_letter(c text)
```

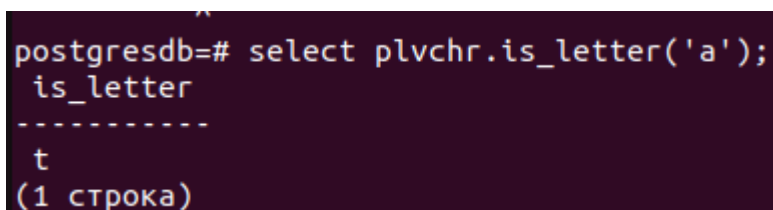
Функция применяется с параметрами, приведенными в таблице 3.56.

Таблица 3.56 – Параметры функции «plvchr.is\_letter»

Параметр	Тип данных	Обозначение
plvchr.is_letter(c int)		
#1	integer	заданное число
plvchr.is_letter(c text)		
#1	text	заданный символ

SQL-запросы будут следующими:

```
SELECT plvchr.is_letter('a');
```



```
postgresdb=# select plvchr.is_letter('a');
 is_letter 
-----
t
(1 строка)
```

Рисунок 3.73 – Пример выполнения SQL-запроса с использованием функции plvchr.is\_letter(c text)

```
SELECT plvchr.is_letter(2);
```

```
postgresdb=# select plvchr.is_letter(2);
is_letter
-----
f
(1 строка)
```

Рисунок 3.74 – Пример выполнения SQL-запроса с использованием функции `plvchr.is_letter(c int)`

### 3.4.19. Функция «`plvchr.char_name`»

Функция возвращает имя символа в коде «ASCII» как «VARCHAR». Для функции используется синтаксис SQL-команды:

```
plvchr.char_name(c text)
```

Функция применяется с параметрами, приведенными в таблице 3.57.

Таблица 3.57 – Параметры функции «`plvchr.char_name`»

Параметр	Тип данных	Обозначение
#1	text	заданный символ

SQL-запрос будет следующим:

```
SELECT plvchr.char_name('#');
```

```
postgresdb=# select plvchr.char_name('#');
char_name
-----
#
(1 строка)
```

Рисунок 3.75 – Пример выполнения SQL-запроса с использованием функции «`plvchr.char_name`»

### 3.4.20. Функция «`plvchr.quoted1`»

Функция возвращает заданную строку, оформленную в одинарные кавычки и использует синтаксис SQL-команды:

```
plvchr.quoted1(str text)
```

Функция применяется с параметрами, приведенными в таблице 3.58.

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

Таблица 3.58 – Параметры функции «plvchr.quoted1(str text)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка

SQL-запрос будет следующим:

```
SELECT plvchr.quoted1('some text');
```

```
postgresdb=# select plvchr.quoted1('some text');
        quoted1
-----
'some text'
(1 строка)
```

Рисунок 3.76 – Пример выполнения SQL-запроса с использованием функции plvchr.quoted1(str text)

### 3.4.21. Функция «plvchr.quoted2»

Функция возвращает заданную строку, оформленную в двойные кавычки, и использует синтаксис SQL-команды:

```
plvchr.quoted2(str text)
```

Функция применяется с параметрами, приведенными в таблице 3.59.

Таблица 3.59 – Параметры функции «plvchr.quoted2(str text)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка

SQL-запрос будет следующим:

```
SELECT plvchr.quoted2('some text');
```

```
postgresdb=# select plvchr.quoted2('some text');
      quoted2
-----
"some text"
(1 строка)
```

Рисунок 3.77 – Пример выполнения SQL-запроса с использованием функции «plvchr.quoted2(str text)»

### 3.4.22. Функция «plvchr.stripped»

Функция удаляет из заданной строки определенные символы.

```
plvchr.stripped(str text, char_in text)
```

Функция применяется с параметрами, приведенными в таблице 3.60.

Таблица 3.60 – Параметры функции «plvchr.stripped»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	символ, который будет удален

sql-запрос будет следующим:

```
SELECT plvchr.stripped('some text', 'e');
```

```
postgresdb=# select plvchr.stripped('some text', 'e');
      stripped
-----
som txt
(1 строка)
```

Рисунок 3.78 – Пример выполнения SQL-запроса с использованием функции «plvchr.stripped»

### 3.5. Функции модуля «PLVsubst»

Модуль «PLVsubst» выполняет замену в строках на основе ключевого слова замены.

Функция позволяет производить замену в строке по различным шаблонам.

### 3.5.1. Функция «plvsubst.string»

Функция заменяет символы замены (установленные по умолчанию) на значения из массива в последовательном порядке. При этом использует нижеперечисленный синтаксис SQL-запросов:

```
plvsubst.string(template_in text, vals_in text[])
plvsubst.string(template_in text, vals_in text, delim_in text)
```

Функция «plvsubst.string» имеет функциональную возможность:

- проверки строки на наличие всех вхождений ключевого слова и замена ее значением из списка значений подстановки (п. 3.5.1.1);
- проверки строки на наличие всех вхождений ключевого слова и замена ее значением из списка значений подстановки и установкой символа замены (п. 3.5.1.2).

#### 3.5.1.1 Проверка строки на наличие всех вхождений ключевого слова и замена ее значением из списка значений подстановки

Функция «plvsubst.string» выполняет проверку строки на наличие всех вхождений ключевого слова и заменяет ее значением из списка значений (массива) подстановки. Используя синтаксис SQL запроса:

```
plvsubst.string(template_in text, vals_in text[])
```

Функция применяется с параметрами, приведенными в таблице 3.61.

Таблица 3.61 – Параметры функции «plvsubst.string(template\_in text, vals\_in text[])»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	значения для замены

SQL-запрос будет следующим:

```
SELECT plvsubst.string('some %s for an %s',
ARRAY['string','example']);
```

```
postgresdb=# select plvsubst.string('some %s for an %s', ARRAY['string','example']);
          string
-----
some string for an example
(1 строка)
```

Рисунок 3.79 – Пример выполнения SQL-запроса с использованием функции

### 3.5.1.2 Проверка строки на наличие всех вхождений ключевого слова и замена ее значением из списка значений подстановки и установкой символа замены

Функция «plvsubst.string» выполняет проверку строки на наличие всех вхождений ключевого слова и заменяет ее значением из списка значений (массива) подстановки, в последовательном порядке и установкой символа замены.

Используя синтаксис SQL-запроса:

```
plvsubst.string(template_in text, vals_in text, delim_in text)
```

Функция применяется с параметрами, приведенными в таблице 3.62.

Таблица 3.62 – Параметры функции «plvsubst.string(template\_in text, vals\_in text, delim\_in text)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	значения для замены
#3	text	символ замены

SQL-запрос будет следующим:

```
SELECT plvsubst.string('some 1 for an 1',
ARRAY['string','example'], '1');
```

```
postgresdb=# select plvsubst.string('some 1 for an 1', ARRAY['string','example'], '1');
          string
-----
some string for an example
(1 строка)
```

Рисунок 3.80 – Пример выполнения SQL-запроса с использованием функции «plvsubst.string(template\_in text, vals\_in text, delim\_in text)»

### 3.5.2. Функция «plvsubst.setsubst»

Функция позволяет установить определенный символ как символ замены по умолчанию. Изначально установлено значение '%s'.

Используется синтаксис SQL-запроса:

```
plvsubst.setsubst(str text)
```

Функция применяется с параметрами, приведенными в таблице 3.63.

Таблица 3.63 – Параметры функции «plvsubst.setsubst»

Параметр	Тип данных	Обозначение
#1	text	символ замены

SQL-запрос будет следующим:

```
SELECT plvsubst.setsubst('1');
```

```
postgresdb=# select plvsubst.setsubst('1');
setsubst
-----
(1 строка)

postgresdb=# select plvsubst.string('some 1 for an 1', ARRAY['string','example']);
      string
-----
some string for an example
(1 строка)
```

Рисунок 3.81 – Пример выполнения SQL-запроса с использованием функции «plvsubst.setsubst»

### 3.5.3. Функция «plvsubst.subst»

Функция возвращает символ, который в данный момент установлен как символ замены.

```
plvsubst.subst()
```

SQL-запрос будет следующим:

```
SELECT plvsubst.subst();
```

```
postgresdb=# select plvsubst.subst();
subst
-----
1
(1 строка)
```

Рисунок 3.82 – Пример выполнения SQL-запроса с использованием функции «plvsubst.subst»

### 3.6. Функции модуля «DBMS\_random»

#### 3.6.1. Функция «dbms\_random.initialize»

Функция инициализирует пакет с начальным значением.

```
dbms_random.initialize(int)
```

Функция применяется с параметрами, приведенными в таблице 3.64.

Таблица 3.64 – Параметры функции «dbms\_random.initialize»

Параметр	Тип данных	Обозначение
#1	integer	начальное значение

Пример SQL-запроса:

```
SELECT dbms_random.initialize(10);
```

```
postgresdb=# select dbms_random.initialize(10);
initialize
-----
(1 строка)
```

Рисунок 3.83 – Пример выполнения SQL-запроса с использованием функции «dbms\_random.initialize»

#### 3.6.2. Функция «dbms\_random.normal»

Функция возвращает случайные числа в стандартном нормальном распределении.

```
dbms_random.normal()
```

Пример SQL-запроса:

```
SELECT dbms_random.normal();
```

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



```
postgresdb=# select dbms_random.normal();
normal
-----
0.7827029772285923
(1 строка)
```

Рисунок 3.84 – Пример выполнения SQL-запроса с использованием функции «dbms\_random.normal»

### 3.6.3. Функция «dbms\_random.random»

Функция возвращает случайное число из диапазона -231 .. 231.

```
dbms_random.random()
```

Пример SQL-запроса:

```
SELECT dbms_random.random();
```

```
postgresdb=# select dbms_random.random();
random
-----
-1210077376
(1 строка)
```

Рисунок 3.85 – Пример выполнения SQL-запроса с использованием функции «dbms\_random.random»

### 3.6.4. Функция «dbms\_random.seed»

Функция сбрасывает состояние генератора псевдослучайных цифр и использует следующий синтаксис SQL-команд:

```
dbms_random.seed(int)
dbms_random.seed(text)
```

Функция применяется с параметрами, приведенными в таблице 3.65.

Таблица 3.65 – Параметры функции «dbms\_random.seed»

Параметр	Тип данных	Обозначение
dbms_random.seed(int)		
#1	Integer	ключ генерации
dbms_random.seed(text)		
#1	text	ключ генерации

SQL-запросы будут следующими:

```
SELECT dbms_random.seed(1);
```

```
postgresdb=# select dbms_random.seed(1);
seed
-----
(1 строка)
```

Рисунок 3.86 – Пример выполнения SQL-запроса с использованием функции «dbms\_random.seed(int)»

```
SELECT dbms_random.seed('text');
```

```
postgresdb=# select dbms_random.seed('text');
seed
-----
(1 строка)
```

Рисунок 3.87 – Пример выполнения SQL-запроса с использованием функции «dbms\_random.seed(text)»

### 3.6.5. Функция «dbms\_random.string»

Функция генерирует случайную строку.

```
dbms_random.string(opt text(1), len int)
```

Функция применяется с параметрами, приведенными в таблице 3.66.

Таблица 3.66 – Параметры функции «dbms\_random.string»

Параметр	Тип данных	Обозначение
#1	text	ключ генерации
#2	integer	длина генерируемой строки

Пример запроса. Для получения случайной строки используем:

- случайный набор текстовых символов (словарь):

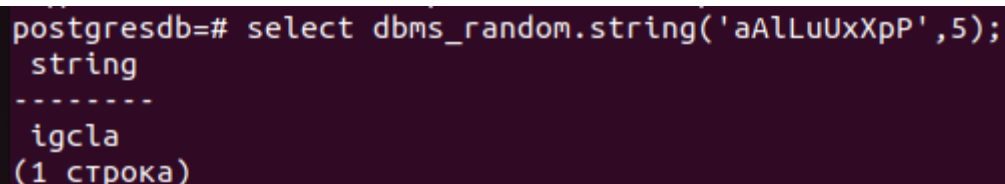
```
opt text - 'aAlLuUxXpP'
```

- указываем длину генерируемой строки:

```
len int - 5
```

В итоге сформируется SQL-запрос:

```
SELECT dbms_random.string('aAlLuUxXpP',5);
```



```
postgresdb=# select dbms_random.string('aAlLuUxXpP',5);
 string
-----
 igcla
(1 строка)
```

Рисунок 3.88 – Пример выполнения SQL-запроса с использованием функции «dbms\_random.string»

В итоге сформируется строка из случайных символов.

### 3.6.6. Функция dbms\_random.value

Функция возвращает случайное число из определенного диапазона. Функция возвращает случайное число из диапазона [0,0–1,0).

```
dbms_random.value()
```

```
dbms_random.value(low double precision, high double precision)
```

Функция «dbms\_random.value» выполняет функциональную возможность:

- возврата случайного числа (3.6.6.1);

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

- возврата числа из заданного диапазона (3.6.6.2).

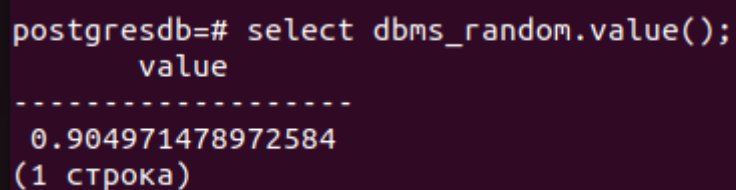
### 3.6.6.1 Возврат случайного числа

Для возврата случайного используется SQL-команда:

```
dbms_random.value()
```

Пример запроса:

```
SELECT dbms_random.value();
```



```
postgresdb=# select dbms_random.value();
           value
-----
 0.904971478972584
(1 строка)
```

Рисунок 3.89 – Пример выполнения SQL-запроса с использованием функции

### 3.6.6.2 Возврат числа из заданного диапазона

Функция возвращает случайное число из заданного диапазона, используя синтаксис SQL-команды:

```
dbms_random.value(low double precision, high double precision)
```

Функция применяется с параметрами, приведенными в таблице 3.67.

Таблица 3.67 – Параметры функции «dbms\_random.value(low double precision, high double precision)»

Параметр	Тип данных	Обозначение
#1	double	нижняя граница диапазона
#2	double	верхняя граница диапазона

SQL-запрос будет следующим:

```
SELECT dbms_random.value(1,10);
```

```
postgresdb=# select dbms_random.value(1,10);  
value  
-----  
6.8631771598011255  
(1 строка)
```

Рисунок 3.90 – Пример выполнения SQL-запроса с использованием функции  
В представленном примере указывается:

- нижняя граница диапазона:

```
low double precision - 1
```

- верхняя граница диапазона:

```
high double precision - 10
```

В результате будет выведено случайное число.

### 3.7. Дополнительные функции

#### 3.7.1. Функция «oracle.substr»

Функция обрезает заданное значение по определенным параметрам.

Функция имеет функциональную возможность:

- вырезки из строки подстроку совместимую с Oracle между заданными позициями (п. 3.7.1.1);
- вырезки из строки подстроку совместимую с Oracle начиная с заданной позиции (п. 3.7.1.2);
- вырезки из числа подстроку совместимую с Oracle начиная с заданной позиции (п. 3.7.1.3);
- вырезки из числа подстроку совместимую с Oracle между заданными позициями (п. 3.7.1.4);

##### 3.7.1.1 Вырезка из строки подстроку, совместимую с Oracle между заданными позициями

Функция вырезает из строки подстроку совместимую с Oracle между заданными позициями используя синтаксис SQL-команды:

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

```
oracle.substr(str text, start int, len int)
```

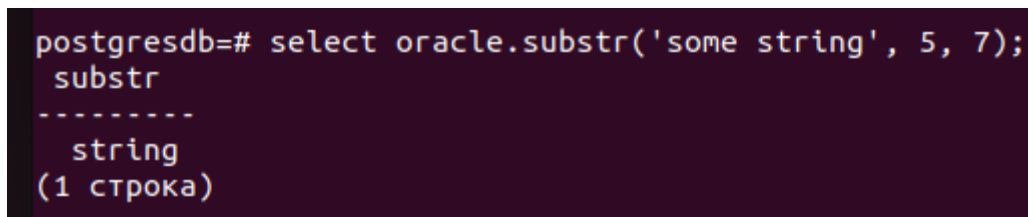
Функция применяется с параметрами, приведенными в таблице 3.68.

Таблица 3.68 – Параметры функции «oracle.substr(str text, start int, len int)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	numeric	начальная позиция
#3	numeric	количество вырезаемых символов

SQL-запрос будет следующим:

```
SELECT oracle.substr('some string', 5, 7);
```



```
postgresdb=# select oracle.substr('some string', 5, 7);
substr
-----
string
(1 строка)
```

Рисунок 3.91 – Пример выполнения SQL-запроса с использованием функции «oracle.substr(str text, start int, len int)»

### 3.7.1.2 Вырезка из строки подстроку совместимую с Oracle, начиная с заданной позиции

Функция вырезает из строки подстроку совместимую с Oracle, начиная с заданной позиции используя синтаксис SQL-команды:

```
oracle.substr(str text, start int)
```

Функция применяется с параметрами, приведенными в таблице 3.69.

Таблица 3.69 – Параметры функции «oracle.substr(str text, start int)»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	numeric	начальная позиция

SQL-запрос будет следующим:

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

```
SELECT oracle.substr('some string', 5);
```

```
postgresdb=# select oracle.substr('some string', 5);
 substr
-----
 string
(1 строка)
```

Рисунок 3.92 – Пример выполнения SQL-запроса с использованием функции «oracle.substr(str text, start int)»

### 3.7.1.3 Вырезка из числа подстроку совместимую с Oracle начиная с заданной позиции

Функция вырезает из числа подстроку совместимую с Oracle, начиная с заданной позиции.

```
oracle.substr(str numeric, start numeric)
```

Функция применяется с параметрами, приведенными в таблице 3.70.

Таблица 3.70 – Параметры функции «oracle.substr(str numeric, start numeric)»

Параметр	Тип данных	Обозначение
#1	numeric	заданное число
#2	numeric	начальная позиция

Пример запроса:

```
SELECT oracle.substr(987654, 4);
```

```
postgresdb=# select oracle.substr(987654, 4);
 substr
-----
 654
(1 строка)
```

Рисунок 3.93 – Пример выполнения SQL-запроса с использованием функции «oracle.substr(str numeric, start numeric)»

### 3.7.1.4 Вырезка из числа подстроку совместимую с Oracle между заданными позициями

Функция вырезает из числа подстроку совместимую с Oracle между заданными позициями.

```
oracle.substr(str numeric, start numeric, len numeric)
```

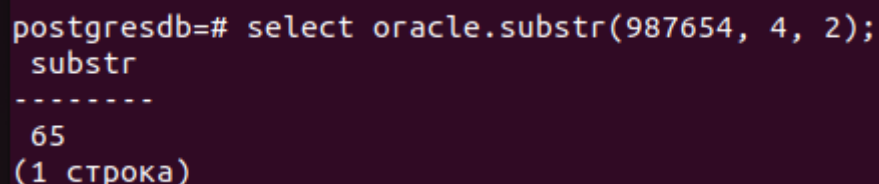
Функция применяется с параметрами, приведенными в таблице 3.71.

Таблица 3.71 – Параметры функции «oracle.substr(str numeric, start numeric, len numeric)»

Параметр	Тип данных	Обозначение
#1	numeric	заданное число
#2	numeric	начальная позиция
#3	numeric	количество вырезаемых символов

Пример запроса:

```
SELECT oracle.substr(987654, 4, 2);
```



```
postgresdb=# select oracle.substr(987654, 4, 2);
substr
-----
65
(1 строка)
```

Рисунок 3.94 – Пример выполнения SQL-запроса с использованием функции «oracle.substr(str numeric, start numeric, len numeric)»

### 3.7.2. Функция «oracle.lpad»

Функция добавляет в начало заданной строки определенный символ, увеличивая строку до определенной длины.

```
oracle.lpad(string, length [, fill])
```

Функция применяется с параметрами, приведенными в таблице 3.72.



Таблица 3.72 – Параметры функции «oracle.lpad»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	integer	итоговая длина строки
#3	text	символ для добавления

SQL-запрос будет следующим:

```
SELECT oracle.lpad('some string', 15, '!');
```

```
postgresdb=# select oracle.lpad('some string', 15, '!');
      lpad
-----
!!!!some string
(1 строка)
```

Рисунок 3.95 – Пример выполнения SQL-запроса с использованием функции «oracle.lpad»

### 3.7.3. Функция «oracle.rpad»

Функция добавляет в конец заданной строки определенный символ, увеличивая строку до определенной длины.

```
oracle.rpad(string, length [, fill])
```

Функция применяется с параметрами, приведенными в таблице 3.73.

Таблица 3.73 – Параметры функции «oracle.rpad»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	integer	итоговая длина строки
#3	text	символ для добавления

SQL-запрос будет следующим.

```
SELECT oracle.rpad('some string', 15, '!');
```

```
postgresdb=# select oracle.rpad('some string', 15, '!');
      rpad
-----
some string!!!!
(1 строка)
```

Рисунок 3.96 – Пример выполнения SQL-запроса с использованием функции «oracle.rpad»

### 3.7.4. Функция «oracle.ltrim»

Функция удаляет из начала заданной строки определенный символ, используя синтаксис SQL-команды:

```
oracle.ltrim(string text [, characters text])
```

Функция применяется с параметрами, приведенными в таблице 3.74.

Таблица 3.74 – Параметры функции «oracle.ltrim»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	символ для удаления

SQL-запрос будет следующим:

```
SELECT oracle.ltrim('111some exmaple string', '1');
```

```
postgresdb=# select oracle.ltrim('111some exmaple string', '1');
      ltrim
-----
some exmaple string
(1 строка)
```

Рисунок 3.97 – Пример выполнения SQL-запроса с использованием функции «oracle.ltrim»

В представленном примере имеется строка «string text»

```
'111some exmaple string'
```

Из которой требуется удалить символ удаления:

```
[, characters text]) - '1'
```

В результате функция удалит из начала строки, найденный символ '1' и выведет строку:

```
some exmaple string
```

### 3.7.5. Функции «oracle.rtrim»

Функция удаляет с конца заданной строки определенный символ, используя синтаксис SQL-команды:

```
oracle.rtrim(string text [, characters text])
```

Функция применяется с параметрами, приведенными в таблице 3.75.

Таблица 3.75 – Параметры функции «oracle.rtrim»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	символ для удаления

SQL-запрос будет следующим:

```
SELECT oracle.rtrim('some exmaple string111', '1');
```

```
postgresdb=# select oracle.rtrim('some exmaple string111', '1');
      rtrim
-----
some exmaple string
(1 строка)
```

Рисунок 3.98 – Пример выполнения SQL-запроса с использованием функции «oracle.rtrim»

В представленном примере имеется строка «string text»

```
'some exmaple string111'
```

Из которой требуется удалить символ удаления:

```
[, characters text]) - '1'
```

В результате функция удалит из конца строки, найденный символ '1' и выведет строку:

```
some exmaple string
```

### 3.7.6. Функция «oracle.btrim»

Функция удаляет из начала и с конца заданной строки определенный символ, используя синтаксис SQL-команды:

```
oracle.btrim(string text [, characters text])
```

Функция применяется с параметрами, приведенными в таблице 3.76.

Таблица 3.76 – Параметры функции «oracle.btrim»

Параметр	Тип данных	Обозначение
#1	text	заданная строка
#2	text	символ для удаления

SQL-запрос будет следующим:

```
SELECT oracle.btrim('111some exmaple string111', '1');
```

```
postgresdb=# select oracle.btrim('111some exmaple string111', '1');
      btrim
-----
some exmaple string
(1 строка)
```

Рисунок 3.99 – Пример выполнения SQL-запроса с использованием функции «oracle.btrim»

В представленном примере имеется строка «string text»

```
'111some exmaple string111'
```

Из которой требуется удалить символ удаления:

```
[, characters text]) - '1'
```

В результате функция удалит из строки, найденный символ '1' и выведет строку:

```
some exmaple string
```

### 3.7.7. Функция «oracle.length»

Функция возвращает длину заданной строки, используя синтаксис SQL-команды:

```
oracle.length(string char)
```

Функция применяется с параметрами, приведенными в таблице 3.77.

Таблица 3.77 – Параметры функции «oracle.length»

Параметр	Тип данных	Обозначение
#1	char	заданная строка

SQL-запрос будет следующим:

```
SELECT oracle.length('some string');
```

```
postgresdb=# select oracle.length('some string');
length
-----
      11
(1 строка)
```

Рисунок 3.100 – Пример выполнения SQL-запроса с использованием функции «oracle.length»

В представленном примере требуется подсчитать количество символов в строке:

```
some string
```

Функция «oracle.length» вывела подсчет результата в количестве 11 символов, что соответствует подсчету приведенному в таблице 3.78.

Таблица 3.78 – Таблица подсчета символов

Символ	s	o	m	e		s	t	r	i	n	g
Количество	1	2	3	4	5	6	7	8	9	10	11
Итого	11										

Пробел учитывается в качестве символа.

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

### 3.7.8. Функция «oracle.to\_number»

Функция преобразовывает заданное значение в число.

Практически функция переводит формат Oracle «number» в формат Postgres «numeric»

```
oracle.to_number(numeric)
```

Функция применяется с параметрами, приведенными в таблице 3.79.

Таблица 3.79 – Параметры функции «oracle.to\_number»

Параметр	Тип данных	Обозначение
oracle.to_number(numeric)		
#1	numeric	число для преобразования
oracle.to_number(text)		
#1	text	текст для преобразования

SQL-запросы будут следующими:

```
SELECT oracle.to_number('265');
```

```
postgresdb=# select oracle.to_number('265');
 to_number
-----
      265
(1 строка)
```

Рисунок 3.101 – Пример выполнения SQL-запроса с использованием функции «oracle.to\_number(numeric)»

```
SELECT oracle.to_number(265);
```

```
postgresdb=# select oracle.to_number(265);
 to_number
-----
      265
(1 строка)
```

Рисунок 3.102 – Пример выполнения SQL-запроса с использованием функции «oracle.to\_number(text)»

### 3.7.9. Функция «oracle.mod»

Функция возвращает остаток от деления заданного числа на заданное число.

```
oracle.mod(int, int)
```

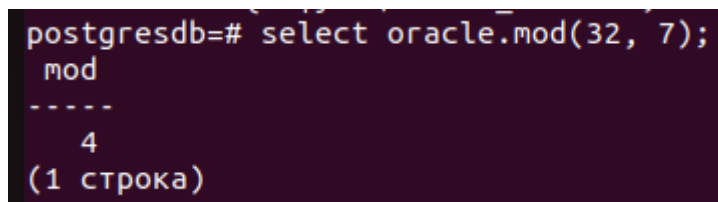
Функция применяется с параметрами, приведенными в таблице 3.80.

Таблица 3.80 – Параметры функции «oracle.mod»

Параметр	Тип данных	Обозначение
#1	integer	делимое
#2	integer	делитель

Пример запроса:

```
SELECT oracle.mod(32, 7);
```



```
postgresdb=# select oracle.mod(32, 7);
mod
----
  4
(1 строка)
```

Рисунок 3.103 – Пример выполнения SQL-запроса с использованием функции «oracle.mod»

В приведенном примере показана строка вычисления:

$$32 = 7 * 4 + 4$$

В результате будет выведен остаток равный «4».

### 3.8. Механизм DBMS\_PIPE

Пакет DBMS\_PIPE позволяет взаимодействовать между двумя и более сессиями подключения к psql. Pipe в данном контексте означает канал обмена сообщениями. Канал может быть частным и публичным: в первом случае из других сессий он доступен только автору, во втором – любым пользователям.

Канал можно создавать, перемещаться по его элементам, помещать в буфер и извлекать из него, отправлять и получать сообщения по мере наполнения буфера, чистить канал и сбрасывать счетчик элементов, получать сгенерированное имя для канала.

Каналы всегда нужно удалять по окончании использования во избежание переполнения доступной памяти.

Ниже представлены функции DBMS\_PIPE:

- DBMS\_PIPE.CREATE\_PIPE – при указании имени, размера и типа канала можно его создать;
- DBMS\_PIPE.NEXT\_ITEM\_TYPE – определить тип следующего элемента в буфере канала;
- DBMS\_PIPE.PACK\_MESSAGE – при указании элемента можно добавить элемент в буфер при формировании канала;
- DBMS\_PIPE.PURGE – при указании имени канала можно его очистить;
- DBMS\_PIPE.RECEIVE\_MESSAGE – при указании имени канала и таймаута ожидания можно выгрузить содержимое канала;
- DBMS\_PIPE.REMOVE\_PIPE – при указании имени удалить канал по окончании использования;
- DBMS\_PIPE.RESET\_BUFFER – сбросить счетчик элементов канала на 0;
- DBMS\_PIPE.SEND\_MESSAGE – при указании имени, таймаута и максимального размера буфера отправить сформированный буфер канала для последующего получения;
- DBMS\_PIPE.UNIQUE\_SESSION\_NAME – получить уникальное имя сессии для использования канала;
- UNPACK\_MESSAGE\_BYTEA - получить сообщение в локальном буфере с типом данных «BYTES»;
- UNPACK\_MESSAGE\_DATE - получить сообщение в локальном буфере с типом данных «DATE»;
- UNPACK\_MESSAGE\_NUMBER - получить сообщение в локальном буфере с типом данных «NUMERIC»;



- UNPACK\_MESSAGE\_RECORD - получить сообщение в локальном буфере с типом данных «RECORD»;
- UNPACK\_MESSAGE\_TEXT - получить сообщение в локальном буфере с типом данных «TEXT»;
- UNPACK\_MESSAGE\_TIMESTAMP - получить сообщение в локальном буфере с типом данных «TIMESTAMP».

Схема работы механизма DBMS\_PIPE представлена на рисунке 3.104

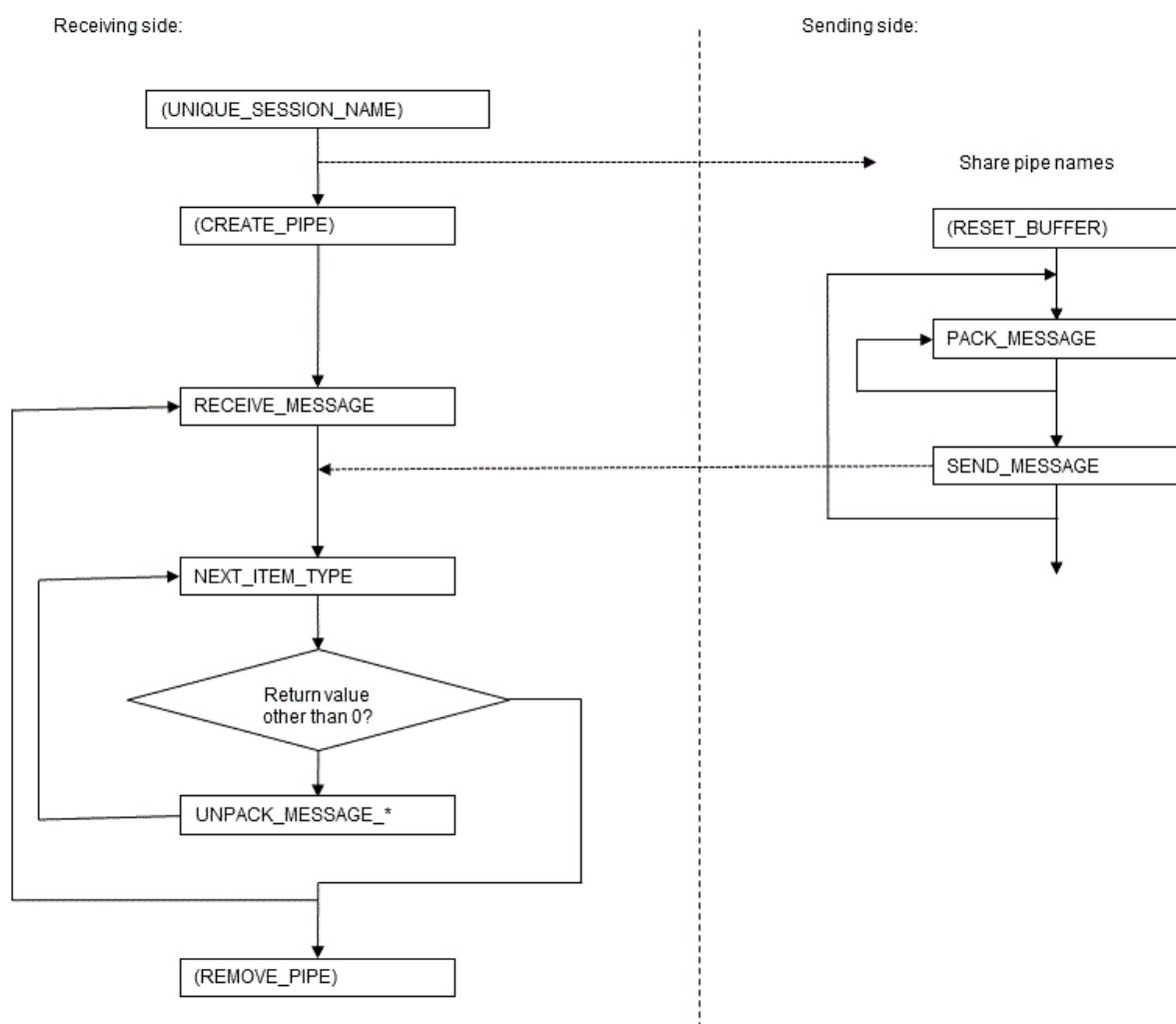


Рисунок 3.104 – Схема работы DBMS\_PIPE

### 3.8.1. Пример реализации автономных транзакции (компонент Orafce, схема dbms\_pipe)

#### Методика проверки

В СУБД «Jatoba» создается канал пересылки сообщений (pipe). Из одной сессии отправляются сообщения, во второй сессии они принимаются. Принимаемые значения записываются в таблицу «test2».

## Сценарий проверки

### Этап 1. Установка

Выполнить полную установку Jatoba, согласно «Руководства по установке».

Для ОС семейства Windows применяется полная или выборочная установка, в которой должны быть выбраны компоненты:

- oracle-fdw;
- orafce;
- pg-variables;

Для GNU Linux установить следующие пакеты:

- jatoba<версия>-oracle-fdw
- jatoba<версия>-orafce
- jatoba<версия>-pg-variables

Установка пакетов выполняется командами:

- для deb:

```
apt-get install jatoba<версия>-oracle-fdw jatoba<версия>-  
orafce jatoba<версия>-pg-variables
```

- для rpm:

```
yum install jatoba<версия>-oracle-fdw jatoba<версия>-  
orafce jatoba<версия>-pg_variables
```

Для GNU Linux помимо вышеописанных пакетов устанавливается пакет oracle-instantclient.

Установка выполняется командой:

```
yum install oracle-instantclient11.2-basic-11.2.0.4.0-1.x86_64.rpm
```

Установить путь к библиотеке Oracle:

```
sh -c "echo /usr/lib/oracle/11.2/client64/lib > /etc/ld.so.conf.d/oracle-instantclient.conf"
```

Выполнить:

```
ldconfig
```

## Этап 2. Функциональное тестирование

Авторизоваться в СУБД от имени и с правами привилегированного пользователя.

Установить расширения SQL-командами:

```
# CREATE EXTENSION orafce;  
# CREATE EXTENSION pg_variables;  
# CREATE EXTENSION oracle_fdw;
```

Открыть 3 сессии пользователем к postgres.

В первой сессии создать 2 таблицы SQL-командами:

```
# CREATE TABLE test1(a int);  
# CREATE TABLE test2(b int);
```

Создать канал пересылки сообщений (pipe):

```
SELECT dbms_pipe.create_pipe('my_pipe',10,true);
```

Запустить внешний цикл в транзакции SQL-командой:

```
CREATE OR REPLACE FUNCTION send_func()  
RETURNS VOID  
LANGUAGE PLPGSQL AS $send_func$  
BEGIN  
FOR i IN 0..1000 LOOP  
    INSERT INTO test1 (a) VALUES (i);  
    perform dbms_pipe.pack_message(i);  
    perform dbms_pipe.send_message('my_pipe',5,0);  
    RAISE NOTICE 'send message';  
END LOOP;
```

```
perform pg_sleep(2);  
END LOOP;  
END  
$send_func$;
```

**Во второй сессии** (должны быть установлены расширения) запустить цикл в транзакции:

```
CREATE OR REPLACE PROCEDURE recv_func()  
LANGUAGE PLPGSQL AS $recv_func$  
BEGIN  
FOR i IN 0..1000 LOOP  
perform dbms_pipe.receive_message('my_pipe',5);  
perform dbms_pipe.next_item_type();  
RAISE NOTICE 'receive message';  
INSERT INTO test2 (b) select dbms_pipe.unpack_message_number();  
COMMIT;  
END LOOP;  
END  
$recv_func$;
```

**Во второй сессии** вызвать процедуру:

```
CALL recv_func();
```

Выводятся сообщения RECEIVE MESSAGE.

**В первой сессии** вызвать функцию:

```
SELECT send_func();
```

Выводятся сообщения SEND MESSAGE.

**В третьей сессии** несколько раз выполнить SQL-команду:

```
SELECT count(*) from test2;
```

Количество записей увеличивается.

### Результат

Сообщения, отправленные в первой сессии psql, получены во второй сессии и записаны в таблицу test2.

## 4. ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ КОМПОНЕНТА PG\_VARIABLES

Компонент используется для работы с переменными различных типов, таких как:

- транзакционные переменные;
- нетранзакционные переменные;
- скалярные переменные;
- переменных-записей;
- переменных типа массив.

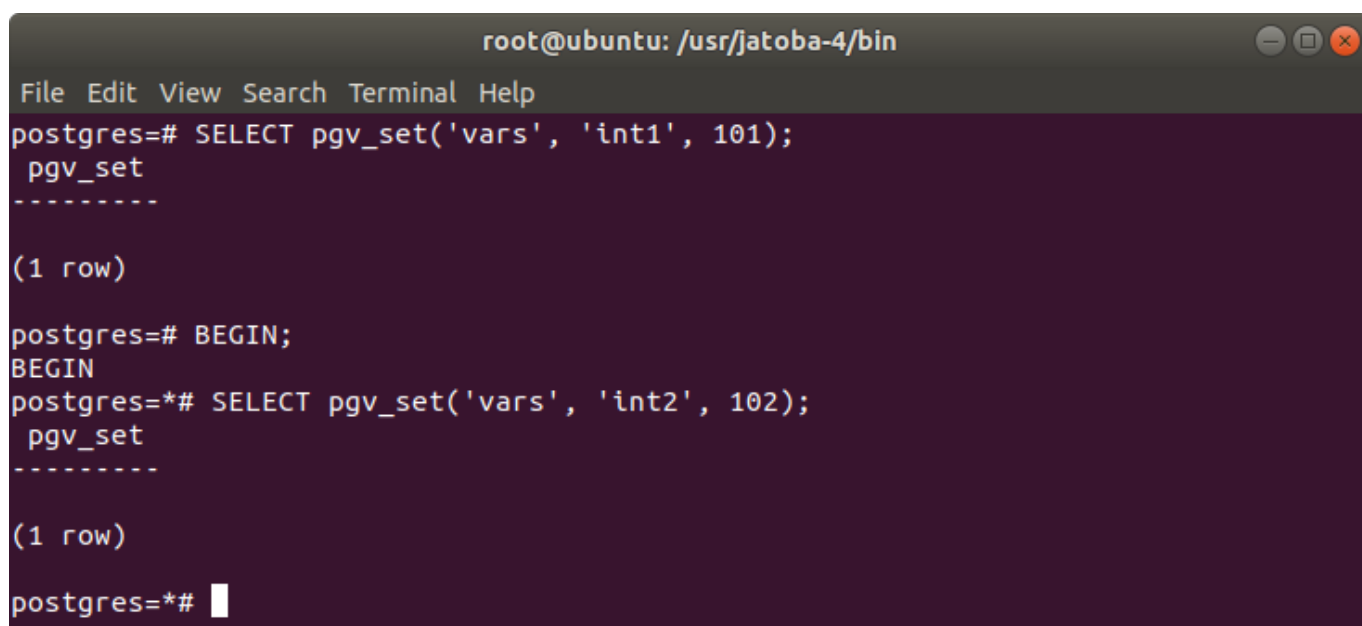
### 4.1. Создание и использование нетранзакционных переменных

Для использования нетранзакционных переменных используется функция «pgv\_set».

Например.

- Создать несколько нетранзакционных переменных и после первой начать транзакцию, выполнив SQL-команды:

```
# SELECT pgv_set('vars', 'int1', 101);  
# BEGIN;  
# SELECT pgv_set('vars', 'int2', 102);
```



The screenshot shows a terminal window titled 'root@ubuntu: /usr/jatoba-4/bin'. The terminal output is as follows:

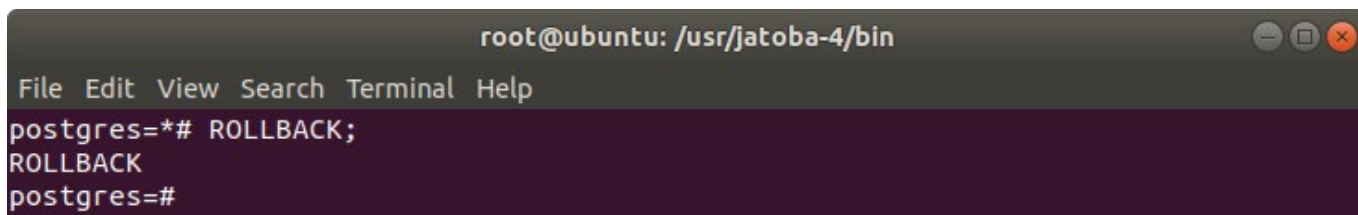
```
File Edit View Search Terminal Help  
postgres=# SELECT pgv_set('vars', 'int1', 101);  
pgv_set  
-----  
(1 row)  
  
postgres=# BEGIN;  
BEGIN  
postgres=# SELECT pgv_set('vars', 'int2', 102);  
pgv_set  
-----  
(1 row)  
  
postgres=#
```

Рисунок 4.1 – Создание нетранзакционных переменных и начало транзакции

- Выполнить откат транзакции:

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

```
ROLLBACK;
```

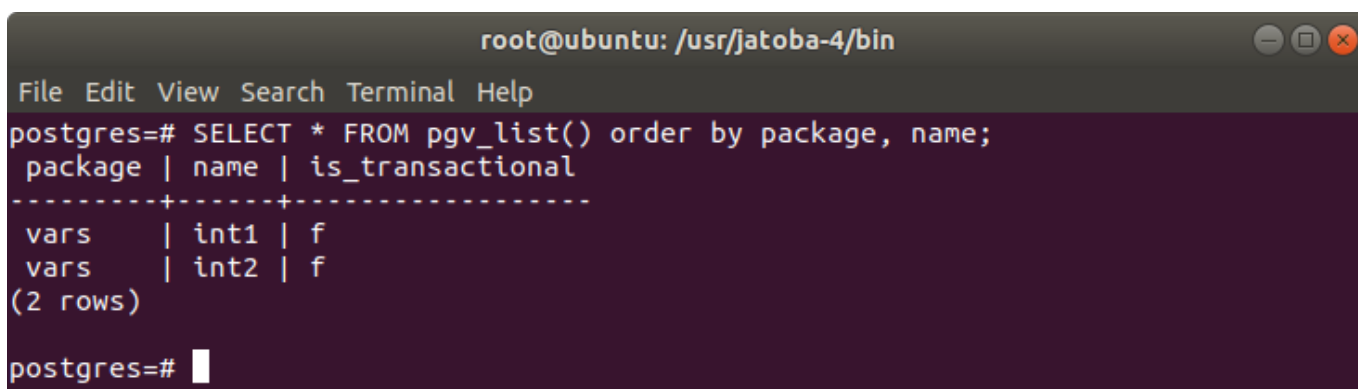


```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# ROLLBACK;
ROLLBACK
postgres=#
```

Рисунок 4.2 – Команда отката операции

- Вывести список переменных:

```
SELECT * FROM pgv_list() order by package, name;
```



```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT * FROM pgv_list() order by package, name;
 package | name | is_transactional
-----+-----+-----
 vars    | int1 | f
 vars    | int2 | f
(2 rows)

postgres=#
```

Рисунок 4.3 – Список переменных

Обе переменные существуют, несмотря на откат.

## 4.2. Создание и использование транзакционных переменных

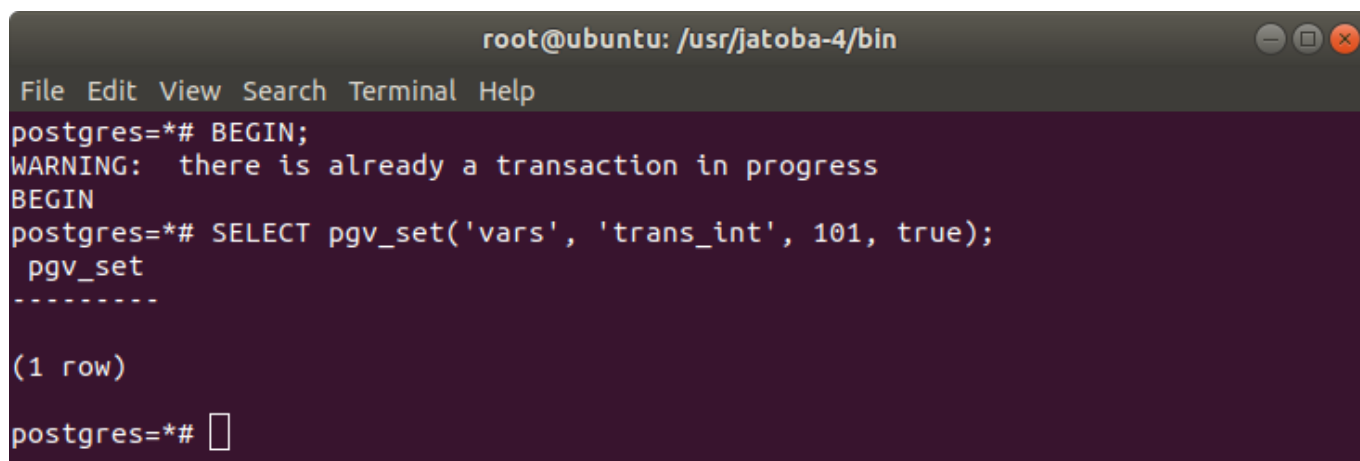
В СУБД по умолчанию переменные создаются как нетранзакционные. Успешно созданная переменная продолжает существовать на протяжении всего сеанса, вне зависимости от возможных откатов транзакций.

Например.

- Создать транзакционную переменную (начать транзакцию в самом начале)

SQL-командой:

```
# BEGIN;
# SELECT pgv_set('vars', 'trans_int', 101, true);
```

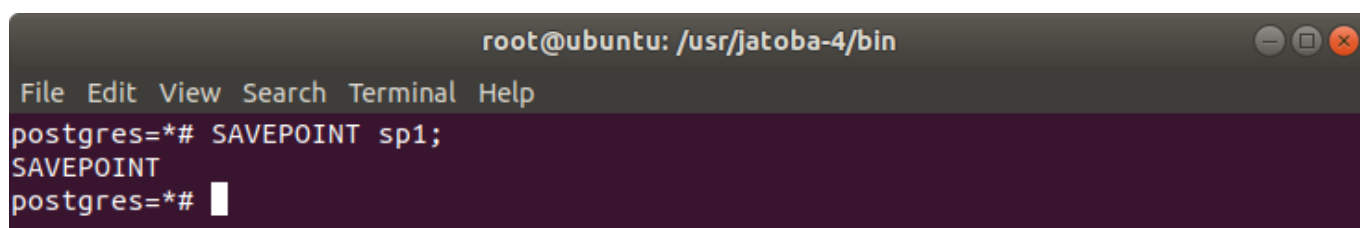


```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# BEGIN;
WARNING:  there is already a transaction in progress
BEGIN
postgres=# SELECT pgv_set('vars', 'trans_int', 101, true);
pgv_set
-----
(1 row)
postgres=#
```

Рисунок 4.4 – Создание транзакционной переменной

- Создать точку сохранения SQL-командой:

```
SAVEPOINT sp1;
```

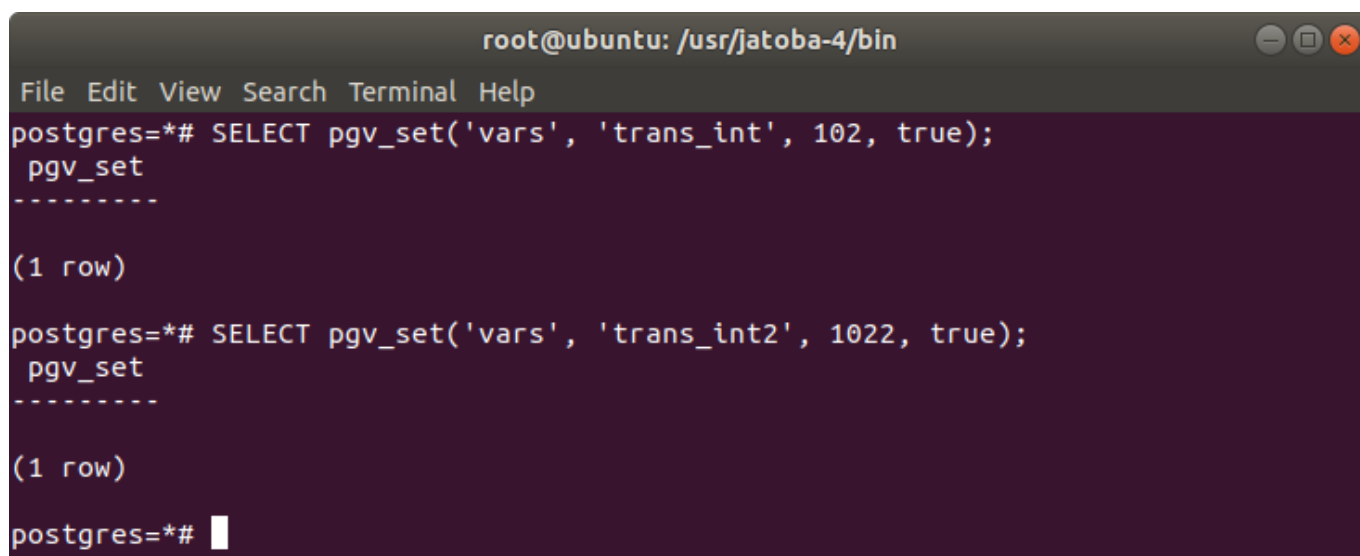


```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SAVEPOINT sp1;
SAVEPOINT
postgres=#
```

Рисунок 4.5 – Создание точки сохранения

- Перезаписать переменную и создать новую переменную SQL-командой:

```
# SELECT pgv_set('vars', 'trans_int', 102, true);
# SELECT pgv_set('vars', 'trans_int2', 1022, true);
```



```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT pgv_set('vars', 'trans_int', 102, true);
pgv_set
-----
(1 row)

postgres=# SELECT pgv_set('vars', 'trans_int2', 1022, true);
pgv_set
-----
(1 row)

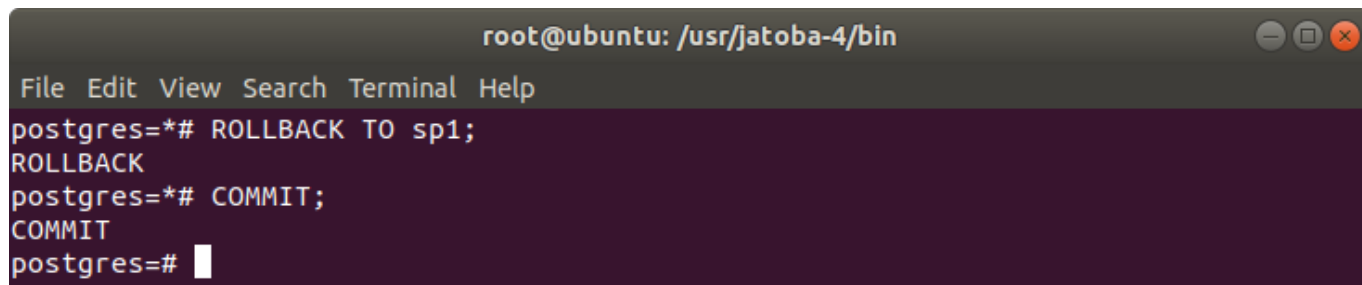
postgres=#
```

Рисунок 4.6 – Перезапись переменной и создание новой

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

- Выполнить откат на точку сохранения и закончить транзакцию SQL-командой:

```
# ROLLBACK TO sp1;  
# COMMIT;
```



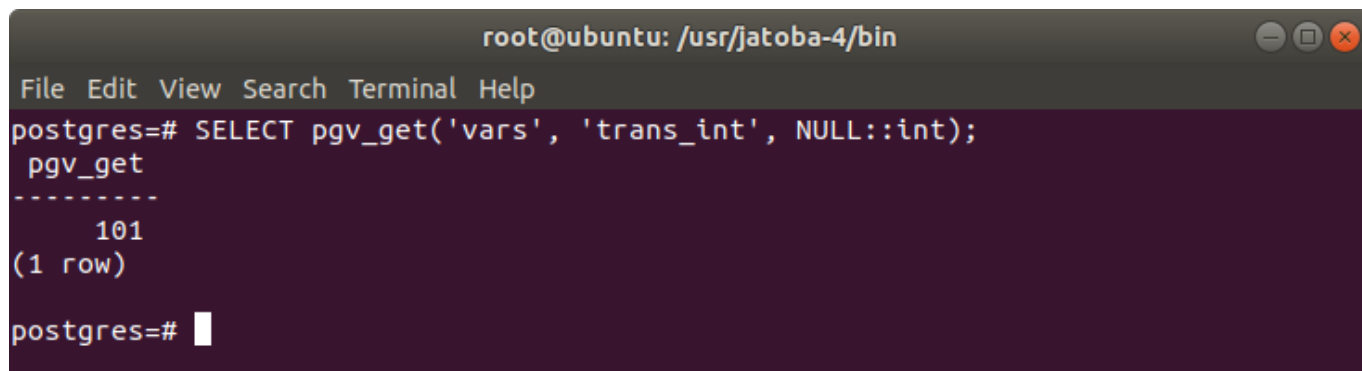
A terminal window titled 'root@ubuntu: /usr/jatoba-4/bin' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
postgres=# ROLLBACK TO sp1;  
ROLLBACK  
postgres=# COMMIT;  
COMMIT  
postgres=#
```

Рисунок 4.7 – Выполнение отката на точку сохранения

- Вывести значение переменной 'trans\_int' SQL-командой:

```
SELECT pgv_get('vars', 'trans_int', NULL::int);
```



A terminal window titled 'root@ubuntu: /usr/jatoba-4/bin' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following command and output:

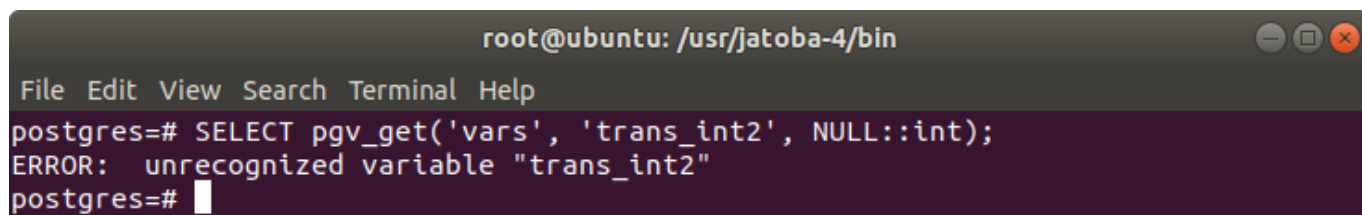
```
postgres=# SELECT pgv_get('vars', 'trans_int', NULL::int);  
pgv_get  
-----  
101  
(1 row)  
postgres=#
```

Рисунок 4.8 – Ввод значения переменной

Значение «trans\_int» соответствует первоначальному, следовательно - откат сработал.

- Вывести значение переменной 'trans\_int2' SQL-командой:

```
SELECT pgv_get('vars', 'trans_int2', NULL::int);
```



A terminal window titled 'root@ubuntu: /usr/jatoba-4/bin' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following command and output:

```
postgres=# SELECT pgv_get('vars', 'trans_int2', NULL::int);  
ERROR: unrecognized variable "trans_int2"  
postgres=#
```

Рисунок 4.9 – Вывод значения переменной 'trans\_int2'

Переменная «trans\_int2» не распознана, т.к. была создана после точки сохранения.



### 4.3. Создание и использование скалярной переменной

Функции «pgv\_set» и «pgv\_get» поддерживают скалярные переменные.

Функция «pgv\_set» выполняет создание пакета и переменной, имеет следующий синтаксис:

```
pgv_set(package text, name text, value anynonarray,  
is_transactional bool default false)
```

Функция «pgv\_get» выполняет проверку типа переменной и вывод значения переменной, имеет следующий синтаксис:

```
pgv_get(package text, name text, var_type anynonarray, strict  
bool default true)
```

Например.

- Выполнить функцию вывода значения переменной SQL-командой:

```
SELECT pgv_get('vars', 'int1', NULL::int);
```

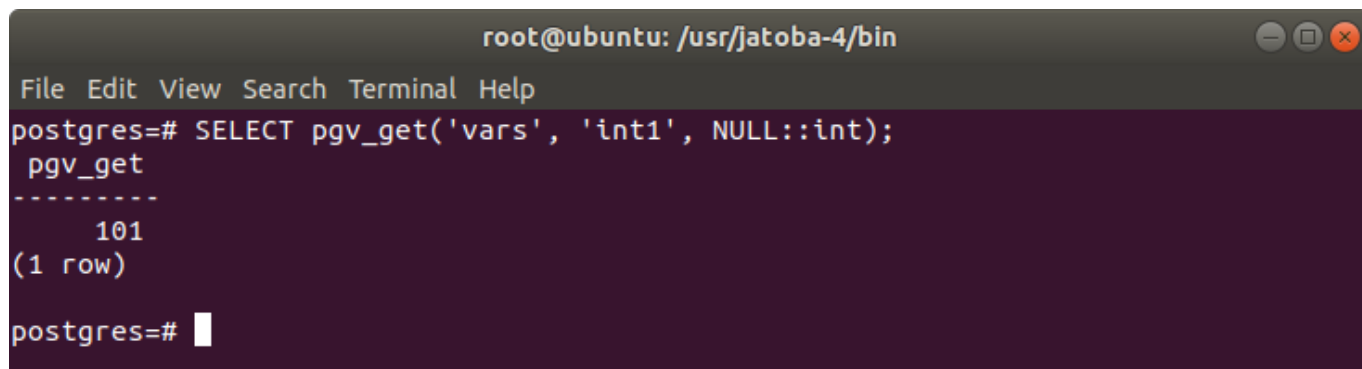
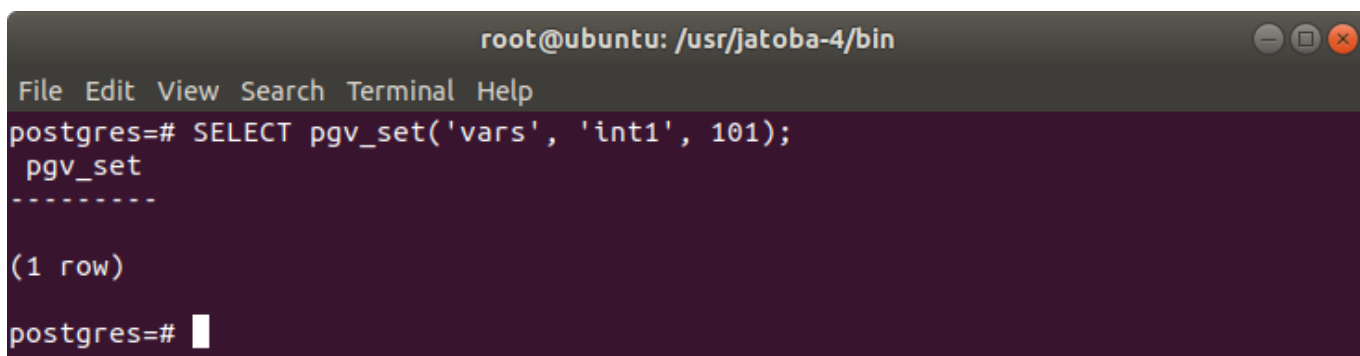


Рисунок 4.10 – Вывод значения переменной

Выводится сообщение об ошибке, т.к. нет пакета и переменной.

- Выполнить создание пакета и переменной SQL-командой:

```
SELECT pgv_set('vars', 'int1', 101);
```



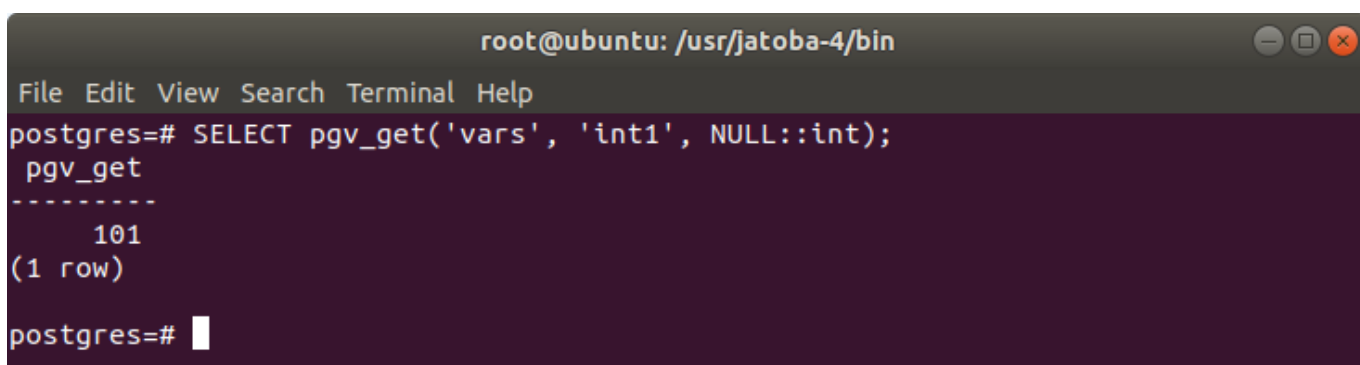
```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT pgv_set('vars', 'int1', 101);
pgv_set
-----
(1 row)
postgres=#
```

Рисунок 4.11 – Создание пакета и переменной

Пакет и переменная созданы.

- Еще раз выполнить функцию вывода значения переменной SQL-командой:

```
SELECT pgv_get('vars', 'int1', NULL::int);
```



```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT pgv_get('vars', 'int1', NULL::int);
pgv_get
-----
      101
(1 row)
postgres=#
```

Рисунок 4.12 – Вывод значения переменной

#### 4.4. Создание и использование переменных типа запись

Функция «pgv\_insert» вставляет запись в набор переменных для заданного пакета. Если пакет или переменная не существуют, они создаются автоматически. Первый столбец записи r — первичный ключ. Если запись с таким же первичным ключом уже существует или этот набор переменных имеет другую структуру, выдается ошибка. Возвращает тип данных void и имеет следующий синтаксис:

```
pgv_insert(package text, name text, r record, is_transactional
bool default false)
```

Функция «pgv\_update» изменяет запись с соответствующим первичным ключом (он задается в первом столбце r). Возвращает «true», если запись была найдена. Если этот набор переменных имеет другую структуру, выдается ошибка.

Функция «pgv\_update» возвращает тип данных «boolean» и имеет следующий синтаксис:

```
pgv_update(package text, name text, r record)
```

Функция «pgv\_delete» удаляет запись с соответствующим первичным ключом (он задается в первом столбце r). Возвращает «true», если запись была найдена.

Функция возвращает тип данных «boolean» и имеет следующий синтаксис:

```
pgv_delete(package text, name text, value anynonarray)
```

Функция «pgv\_select» возвращает:

- записи из набора переменных;
- записи из набора переменных с соответствующими первичными ключами (первичный ключ задается в первом столбце r).

Функция возвращает тип данных «set of records», «records» и имеет следующий синтаксис:

```
pgv_select(package text, name text)  
pgv_select(package text, name text, value anynonarray)
```

Например.

- Создать и наполнить таблицу SQL-командой:

```
# CREATE TABLE tab (id int, t varchar);  
# INSERT INTO tab VALUES (0, 'str00'), (1, 'str11');
```

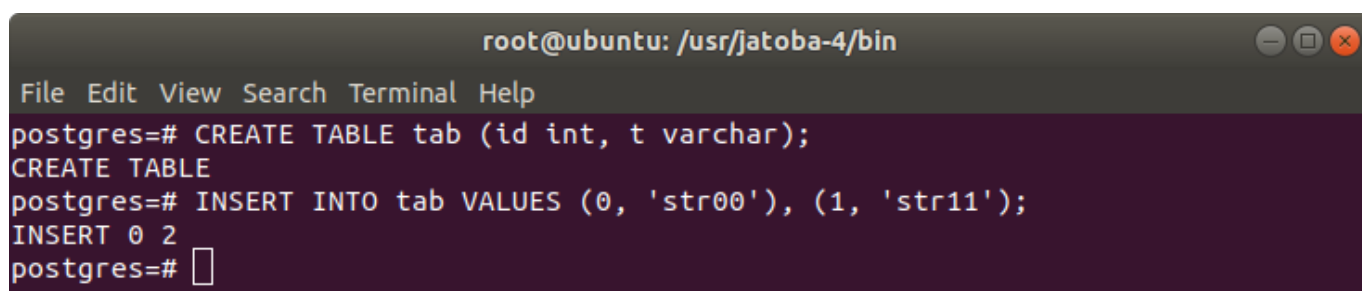


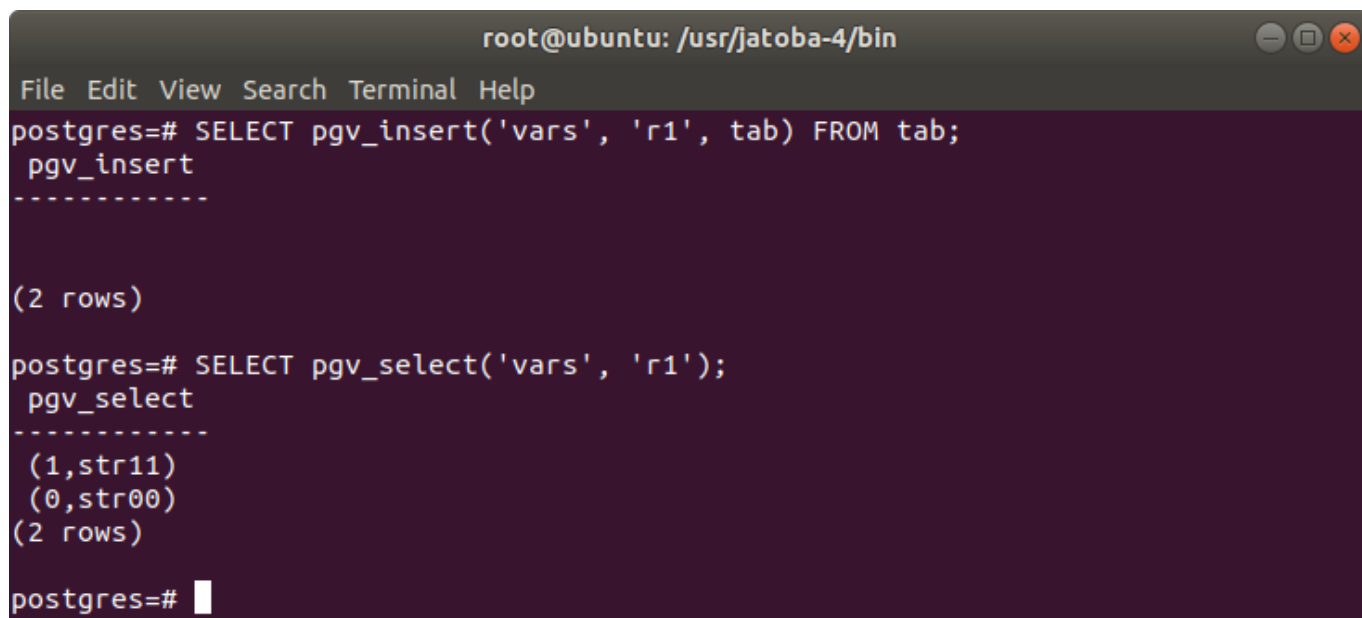
Рисунок 4.13 – Создание таблицы

Таблица успешно создана и наполнена данными.

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

- Выполнить функцию вставки «pgv\_insert» и функцию отображения записей «pgv\_select»:

```
# SELECT pgv_insert('vars', 'r1', tab) FROM tab;  
# SELECT pgv_select('vars', 'r1');
```



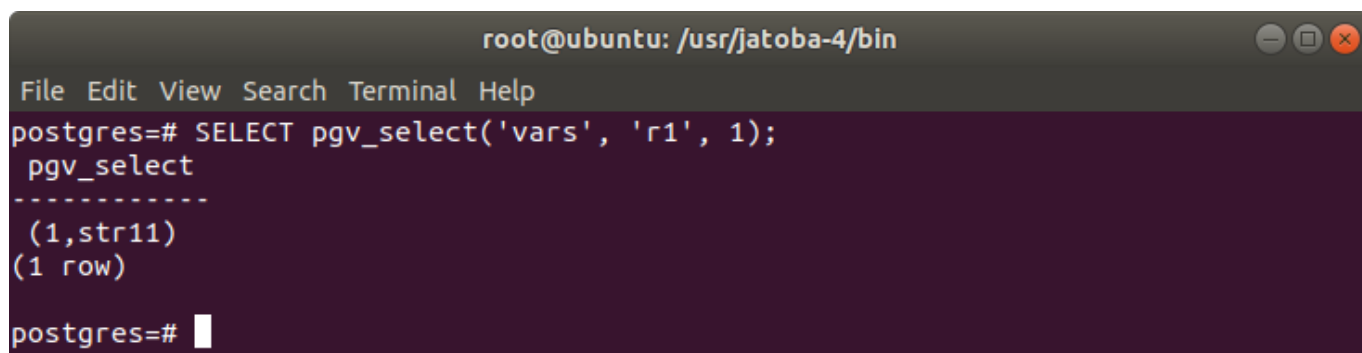
The screenshot shows a terminal window titled 'root@ubuntu: /usr/jatoba-4/bin'. The terminal contains the following commands and output:

```
File Edit View Search Terminal Help  
postgres=# SELECT pgv_insert('vars', 'r1', tab) FROM tab;  
pgv_insert  
-----  
  
(2 rows)  
  
postgres=# SELECT pgv_select('vars', 'r1');  
pgv_select  
-----  
(1,str11)  
(0,str00)  
(2 rows)  
  
postgres=#
```

Рисунок 4.14 – Выполнение функции и вывода записи

- Отобразить запись с идентификатором '1' из пакета 'vars':

```
SELECT pgv_select('vars', 'r1', 1);
```



The screenshot shows a terminal window titled 'root@ubuntu: /usr/jatoba-4/bin'. The terminal contains the following commands and output:

```
File Edit View Search Terminal Help  
postgres=# SELECT pgv_select('vars', 'r1', 1);  
pgv_select  
-----  
(1,str11)  
(1 row)  
  
postgres=#
```

Рисунок 4.15 – Отображение записи с индикатором '1'

- Отобразить запись с идентификатором '0' из пакета 'vars':

```
SELECT pgv_select('vars', 'r1', 0);
```

```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT pgv_select('vars', 'r1', 0);
pgv_select
-----
(0,str00)
(1 row)

postgres=#
```

Рисунок 4.16 - Отображение записи с индикатором '0'

- Отобразить записи из массива выбранных идентификаторов (1 и 0):

```
SELECT pgv_select('vars', 'r1', ARRAY[1, 0]);
```

```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT pgv_select('vars', 'r1', ARRAY[1, 0]);
pgv_select
-----
(1,str11)
(0,str00)
(2 rows)

postgres=#
```

Рисунок 4.17 – Отображение записи из массива

- Выполнить функцию удаления записи с идентификатором 1:

```
SELECT pgv_delete('vars', 'r1', 1);
```

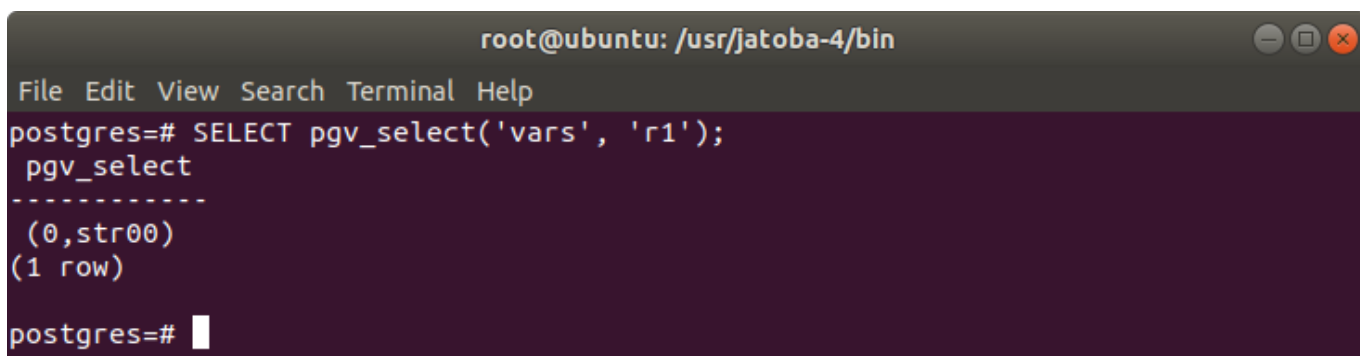
```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT pgv_delete('vars', 'r1', 1);
pgv_delete
-----
t
(1 row)

postgres=#
```

Рисунок 4.18 – Удаление записи

- Убедиться, что осталась одна запись:

```
SELECT pgv_select('vars', 'r1');
```



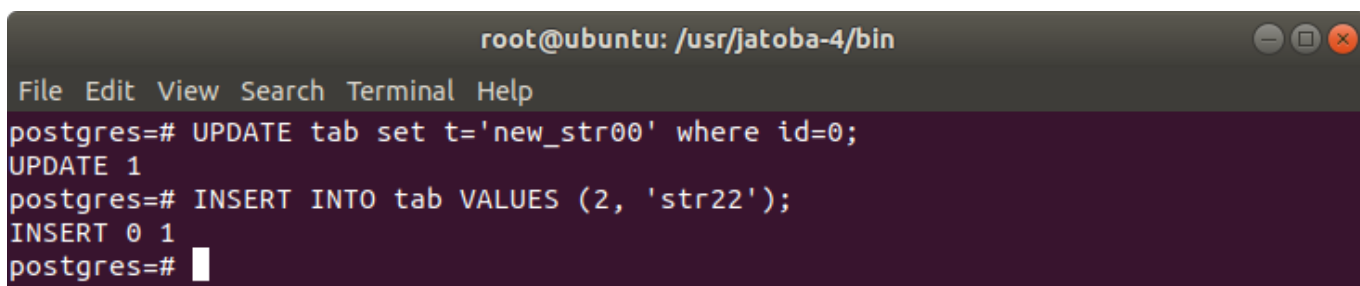
```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT pgv_select('vars', 'r1');
pgv_select
-----
(0,str00)
(1 row)

postgres=#
```

Рисунок 4.19 – Вывод списка записей

- Обновить старую запись и добавить новую:

```
# UPDATE tab set t='new_str00' where id=0;
# INSERT INTO tab VALUES (2, 'str22');
```



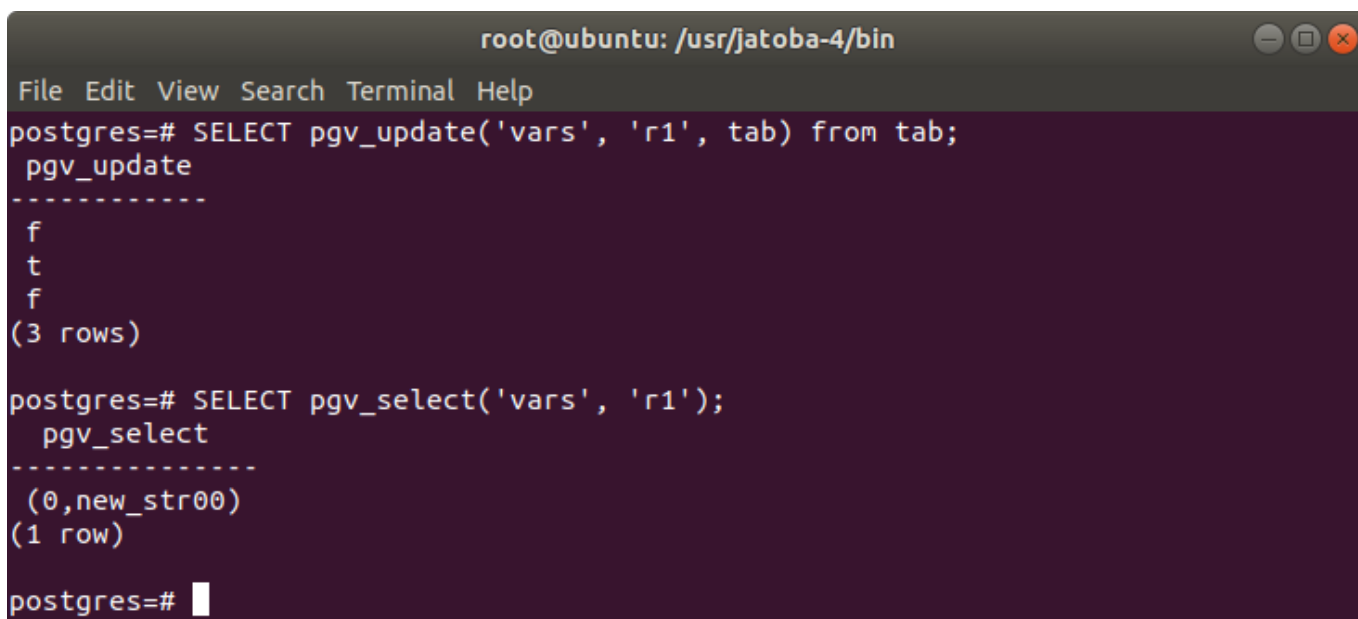
```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# UPDATE tab set t='new_str00' where id=0;
UPDATE 1
postgres=# INSERT INTO tab VALUES (2, 'str22');
INSERT 0 1
postgres=#
```

Рисунок 4.20 – Обновление записей

В таблице старая запись обновлена и добавлена новая.

- Выполнить функции обновления и отображения записей:

```
# SELECT pgv_update('vars', 'r1', tab) from tab;
# SELECT pgv_select('vars', 'r1');
```



```

root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT pgv_update('vars', 'r1', tab) from tab;
pgv_update
-----
f
t
f
(3 rows)

postgres=# SELECT pgv_select('vars', 'r1');
pgv_select
-----
(0,new_str00)
(1 row)

postgres=#

```

Рисунок 4.21 – Обновление и отображение записей

Функция «pgv\_update» только обновляет существующие записи, но не добавляет новые, поэтому будет отражена 1 запись.

#### 4.5. Создание и использование переменных типа массив

Функции «pgv\_set» и «pgv\_get» поддерживают переменный с типом «массив».

Функции «pgv\_set» использует для переменных типа массив следующий синтаксис:

```
pgv_set(package t ext, name text, value
anyarray, is_transactional bool default false)
```

и использует аргументы:

- package - имя пакета, оно должно быть создано если не существует;
- name - имя переменной, она будет создана, если не существует. pgv\_set терпит неудачу, если переменная уже существует и ее транзакционность не соответствует is\_transactional аргументу;
- value - новое значение переменной. pgv\_set терпит неудачу, если переменная уже существует и ее тип не соответствует типу нового значения;
- is\_transactional - транзакционность вновь созданной переменной, по умолчанию false.

Функции «pgv\_get» использует для переменных типа массив следующий синтаксис:

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

```
pgv_get(p ackage text, name text, var_type an yarray, strict  
bool default true)
```

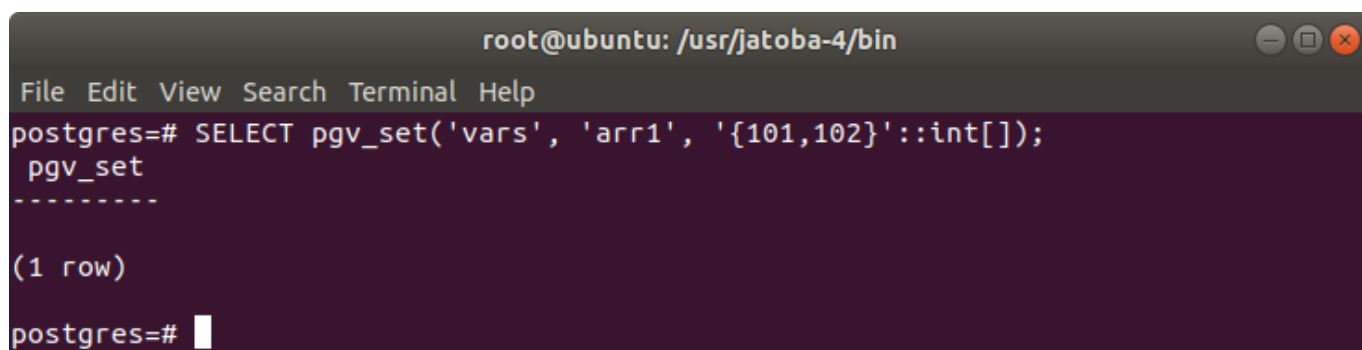
и использует аргументы:

- `package` – имя существующего пакета. Если пакет не существует, результат зависит от `strict` аргумента: если он ложный, то `pgv_get` возвращает `NULL`, в противном случае происходит сбой;
- `name` – имя существующей переменной. Если переменная не существует, результат зависит от `strict` аргумента: если он ложный, то `pgv_get` возвращает `NULL`, в противном случае происходит сбой;
- `var_type` – тип существующей переменной. Его необходимо передать, чтобы получить правильный тип возвращаемого значения;
- `strict` – передать `false`, если `pgv_get` не должно вызывать ошибку, если переменная или пакет не были созданы ранее, по умолчанию это правда.

Например.

- Выполнить функцию задания массива:

```
SELECT pgv_set('vars', 'arr1', '{101,102}'::int[]);
```



The screenshot shows a terminal window titled 'root@ubuntu: /usr/jatoba-4/bin'. The terminal has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The command 'postgres=# SELECT pgv\_set('vars', 'arr1', '{101,102}'::int[]);' has been entered. The output shows 'pgv\_set' followed by a separator line '-----', then '(1 row)', and finally 'postgres=#' with a cursor.

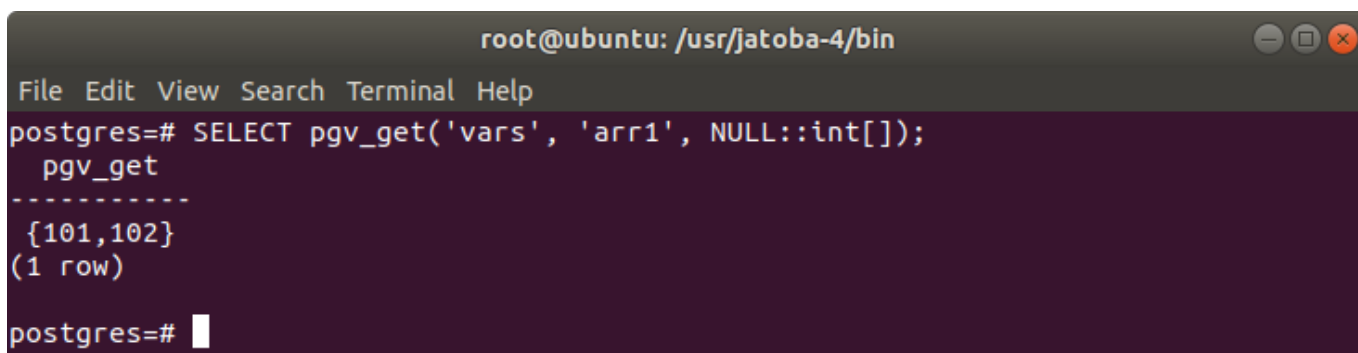
Рисунок 4.22 – Создание переменной типа массив

Функция успешно выполнена.

- Вывести созданный массив:

```
SELECT pgv_get('vars', 'arr1', NULL::int[]);
```





```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT pgv_get('vars', 'arr1', NULL::int[]);
pgv_get
-----
{101,102}
(1 row)

postgres=#
```

Рисунок 4.23 – Вывод массива

Массив создан.

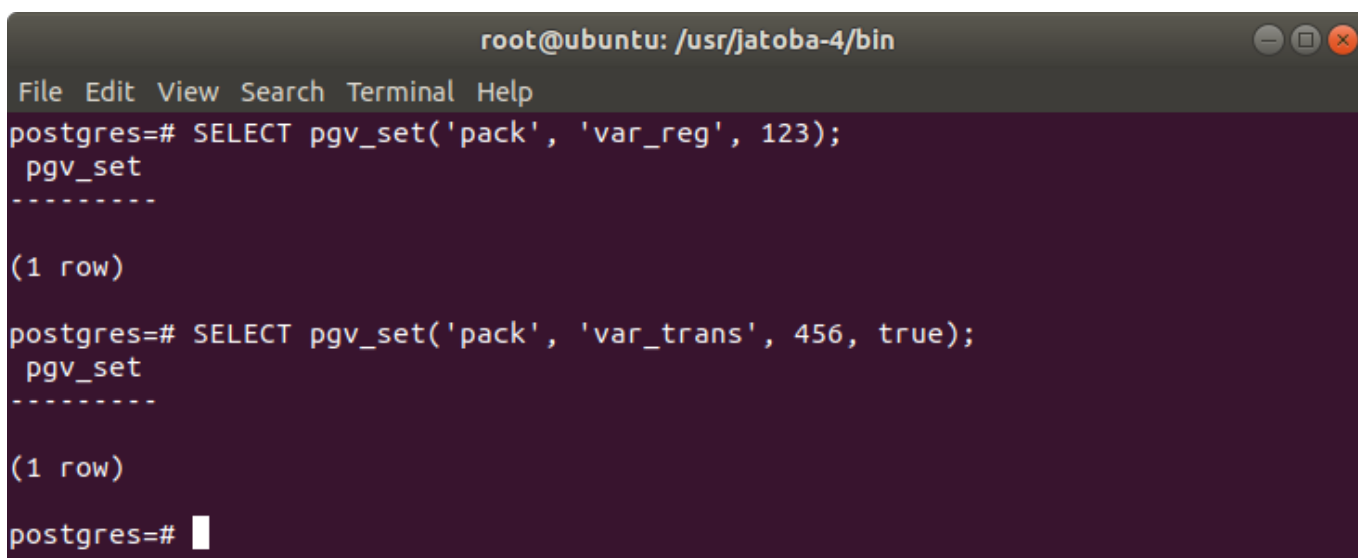
#### 4.6. Восстановление удаленной транзакционной переменной

Команда «ROLLBACK» позволяет восстановить удаленную транзакционную переменную.

Например.

- Создать пакет с нетранзакционной и транзакционной переменными:

```
# SELECT pgv_set('pack', 'var_reg', 123);
# SELECT pgv_set('pack', 'var_trans', 456, true);
```



```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT pgv_set('pack', 'var_reg', 123);
pgv_set
-----
(1 row)

postgres=# SELECT pgv_set('pack', 'var_trans', 456, true);
pgv_set
-----
(1 row)

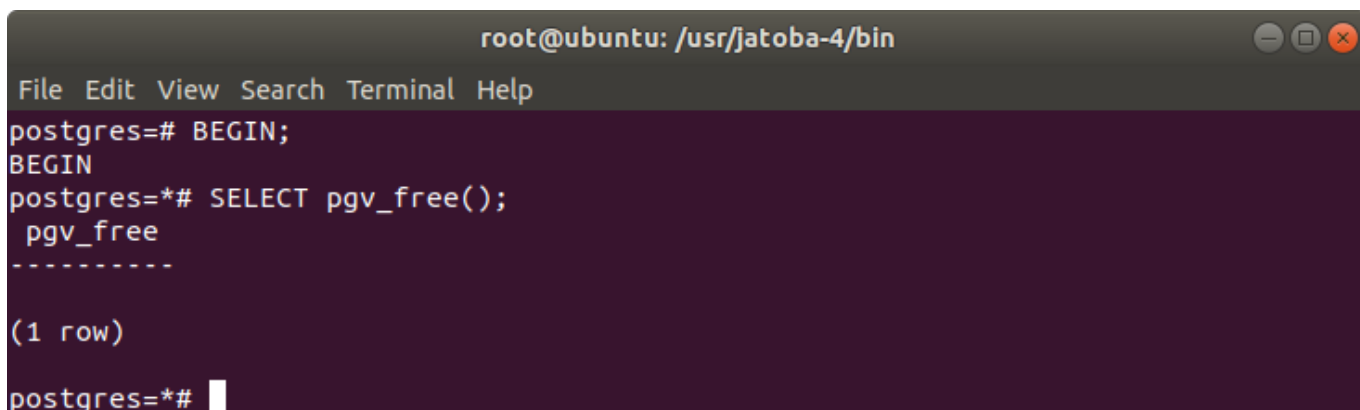
postgres=#
```

Рисунок 4.24 – Создание пакетов с различными переменными

Пакеты с переменными созданы.

- Начать транзакцию и удалить все пакеты с переменными:

```
# BEGIN;  
# SELECT pgv_free();
```



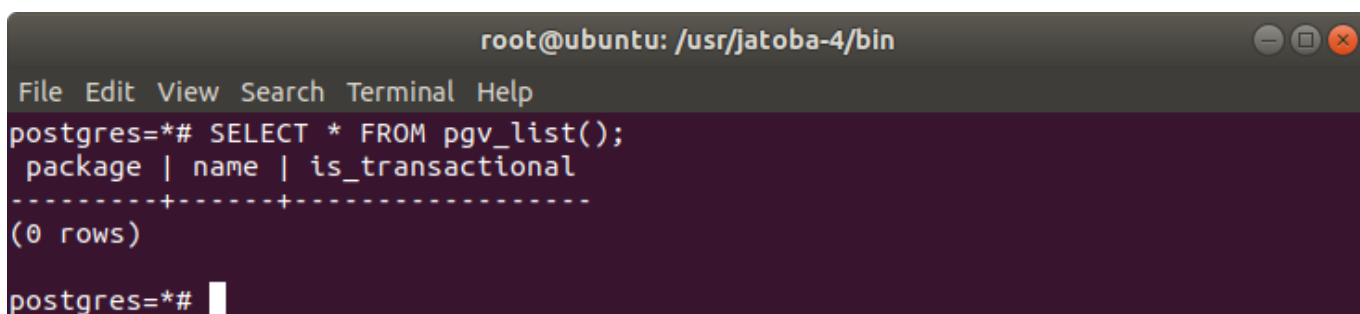
```
root@ubuntu: /usr/jatoba-4/bin  
File Edit View Search Terminal Help  
postgres=# BEGIN;  
BEGIN  
postgres=# SELECT pgv_free();  
pgv_free  
-----  
(1 row)  
postgres=#
```

Рисунок 4.25 – Выполнение транзакции

Пакеты с переменными удалены.

- Убедиться, что все пакеты удалены:

```
SELECT * FROM pgv_list();
```

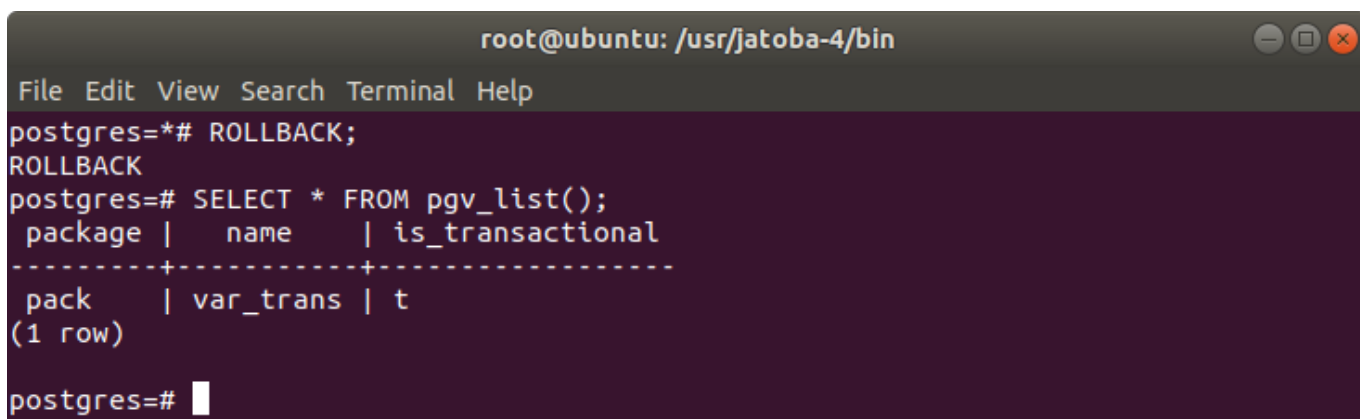


```
root@ubuntu: /usr/jatoba-4/bin  
File Edit View Search Terminal Help  
postgres=# SELECT * FROM pgv_list();  
package | name | is_transactional  
-----+-----+-----  
(0 rows)  
postgres=#
```

Рисунок 4.26 – Вывод списка пакетов

Выполнить откат и убедиться, что восстановлена только транзакционная переменная:

```
# ROLLBACK;  
# SELECT * FROM pgv_list();
```



```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# ROLLBACK;
ROLLBACK
postgres=# SELECT * FROM pgv_list();
 package |   name   | is_transactional
-----+-----+-----
 pack    | var_trans | t
(1 row)
postgres=#
```

Рисунок 4.27 - Откат удаленной транзакции

#### 4.7. Вывод занимаемой памяти

Функция «pgv\_stats» возвращает список созданных пакетов и объем памяти, используемый переменными, в байтах.

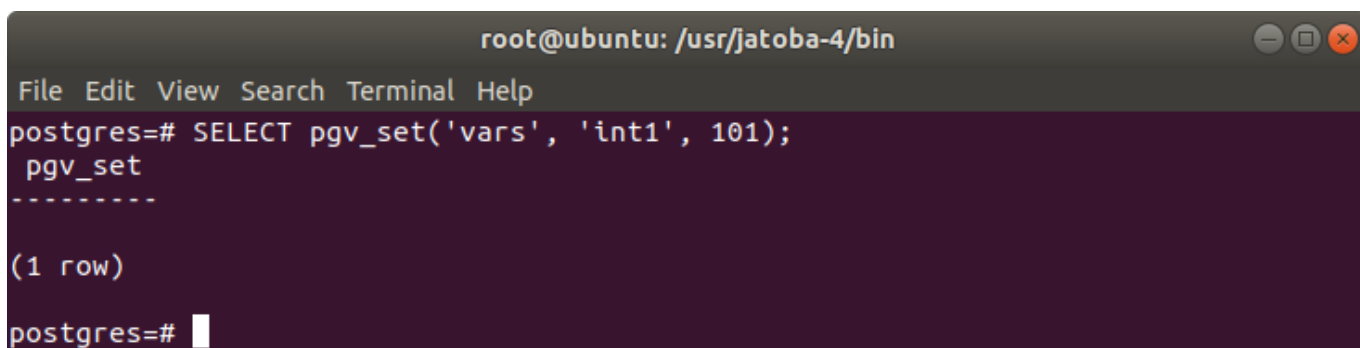
Функция имеет синтаксис:

```
pgv_stats()
```

Например.

- Создать пакет с переменной:

```
SELECT pgv_set('vars', 'int1', 101);
```



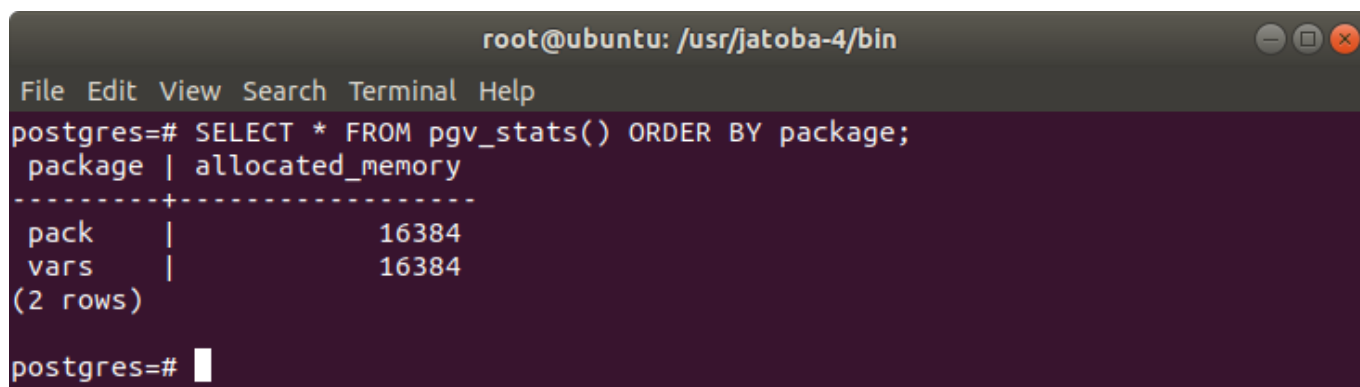
```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT pgv_set('vars', 'int1', 101);
pgv_set
-----
(1 row)
postgres=#
```

Рисунок 4.28 – Создание пакета с переменной

Пакет с переменной созданы.

- Вывести объем памяти в байтах, занятой переменными:

```
SELECT * FROM pgv_stats() ORDER BY package;
```



```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT * FROM pgv_stats() ORDER BY package;
 package | allocated_memory
-----+-----
 pack    |             16384
 vars    |             16384
(2 rows)

postgres=#
```

Рисунок 4.29 – Вывод объема памяти занятой переменной

Выведен объем занятой памяти по пакетам.

#### 4.8. Удаление переменной

Функция «pgv\_remove» удаляет заданный пакет и все его переменные. Указанный пакет должен существовать, иначе выдается ошибка.

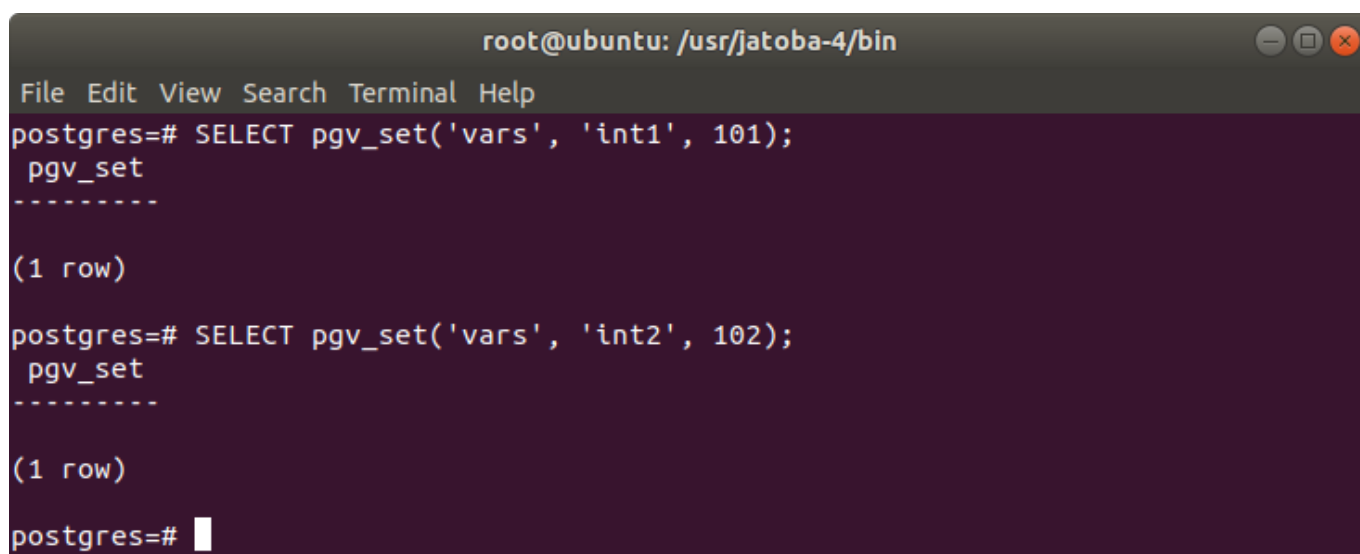
Функция «pgv\_remove» возвращает тип данных void и имеет следующий синтаксис:

```
pgv_remove(package text)
```

Например

- Создать пакет и несколько переменных SQL-командой:

```
# SELECT pgv_set('vars', 'int1', 101);
# SELECT pgv_set('vars', 'int2', 102);
```



```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT pgv_set('vars', 'int1', 101);
pgv_set
-----
(1 row)

postgres=# SELECT pgv_set('vars', 'int2', 102);
pgv_set
-----
(1 row)

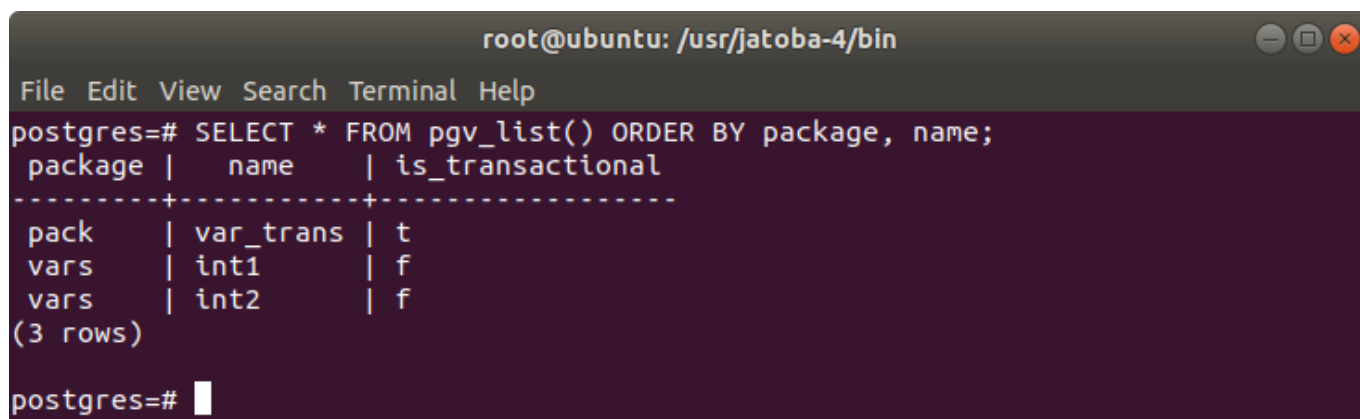
postgres=#
```

Рисунок 4.30 – Создание пакета и переменных

Пакет с переменными созданы.

- Вывести пакеты и переменные SQL-командой:

```
SELECT * FROM pgv_list() ORDER BY package, name;
```



```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT * FROM pgv_list() ORDER BY package, name;
 package |   name   | is_transactional
-----+-----+-----
 pack    | var_trans | t
 vars    | int1      | f
 vars    | int2      | f
(3 rows)

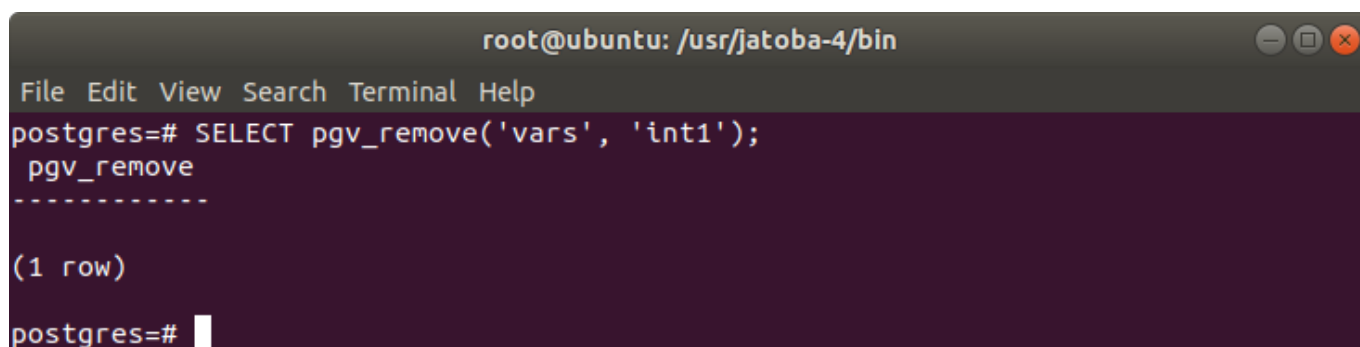
postgres=#
```

Рисунок 4.31 – Вывод пакета и переменных

Созданные пакет и переменные отображены.

- Удалить переменную «int1» SQL-командой:

```
SELECT pgv_remove('vars', 'int1');
```



```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT pgv_remove('vars', 'int1');
 pgv_remove
-----
(1 row)

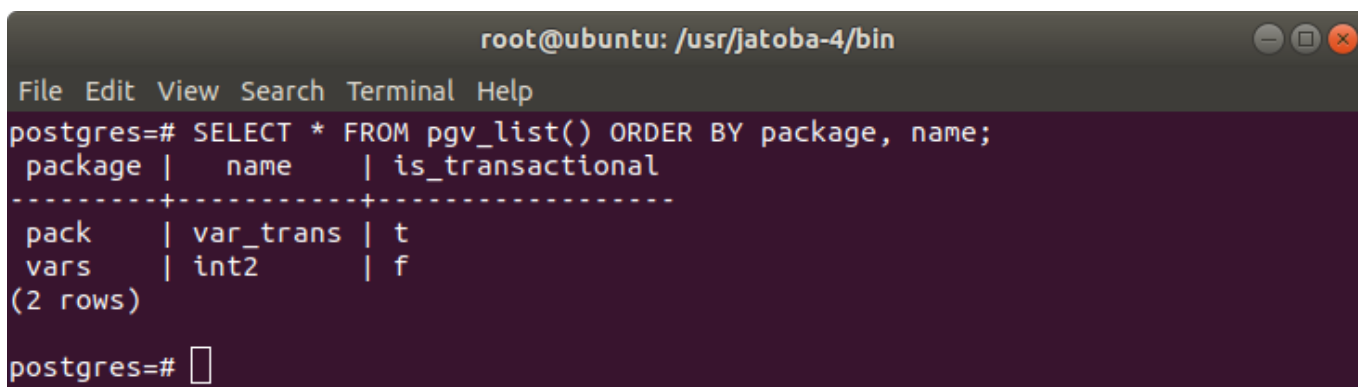
postgres=#
```

Рисунок 4.32 – Удаление переменной

Переменная удалена.

- Еще раз вывести пакеты и переменные SQL-командой:

```
SELECT * FROM pgv_list() ORDER BY package, name;
```



```

root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT * FROM pgv_list() ORDER BY package, name;
 package |   name   | is_transactional
-----+-----+-----
 pack    | var_trans | t
 vars    | int2      | f
(2 rows)

postgres=#

```

Рисунок 4.33 – Вывод пакета и переменных

Удаленная переменная в списке отсутствует.

#### 4.9. Удаление всех пакетов и переменных

Функция «pgv\_free» удаляет все пакеты и переменные. Возвращает тип данных void и имеет следующий синтаксис:

```
pgv_free()
```

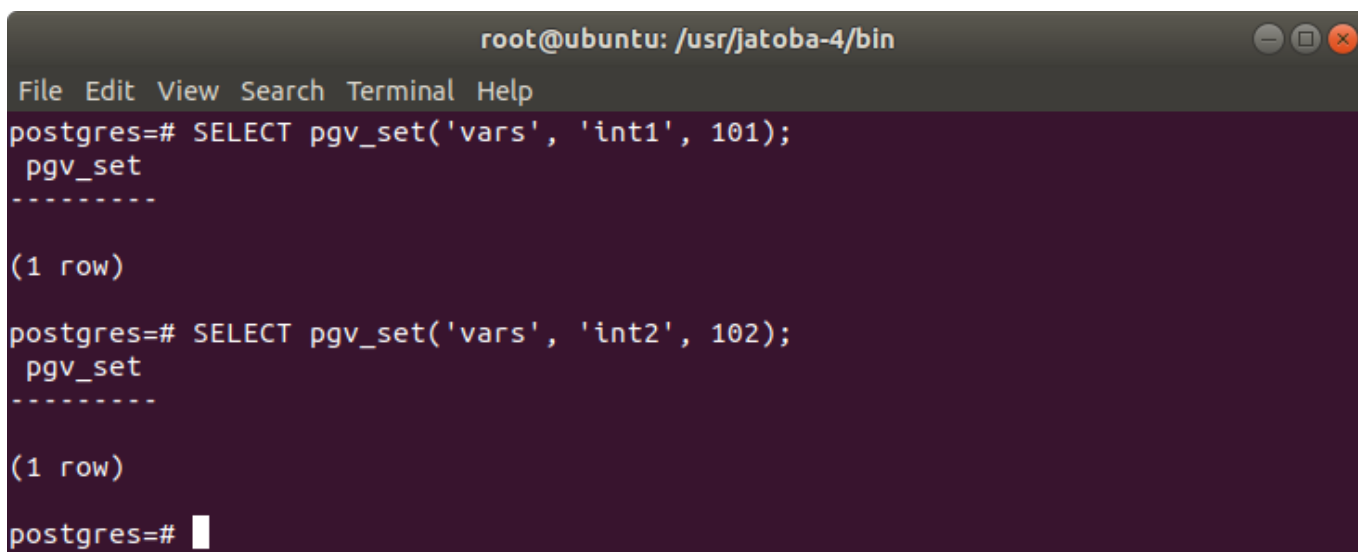
Например

- Создать 1-й пакет и несколько переменных:

```

# SELECT pgv_set('vars', 'int1', 101);
# SELECT pgv_set('vars', 'int2', 102);

```



```

root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT pgv_set('vars', 'int1', 101);
 pgv_set
-----
(1 row)

postgres=# SELECT pgv_set('vars', 'int2', 102);
 pgv_set
-----
(1 row)

postgres=#

```

Рисунок 4.34 – Создание первого пакета и переменных

Пакет с переменными созданы.

- Создать 2-й пакет и несколько переменных:

```
# SELECT pgv_set('vars2', 'text_a', 'text_a'::text);  
# SELECT pgv_set('vars2', 'text_b', 'text_b'::text);
```

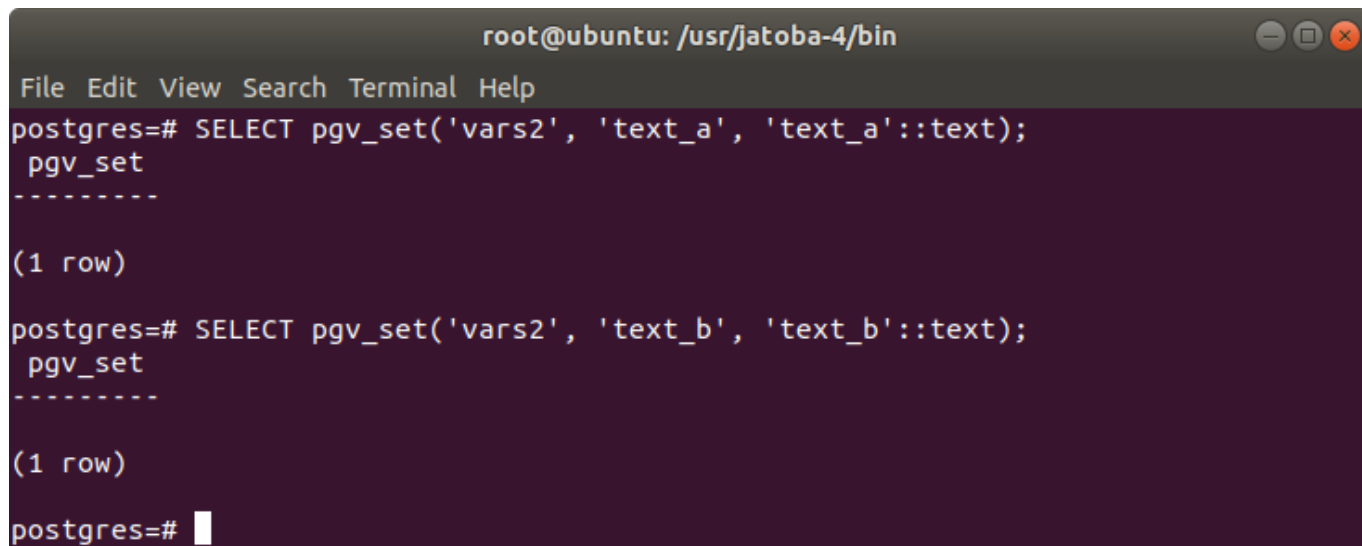


Рисунок 4.35 – Создание второго пакета и переменных

Пакет с переменными созданы.

- Удалить все пакеты и переменные:

```
SELECT pgv_free();
```

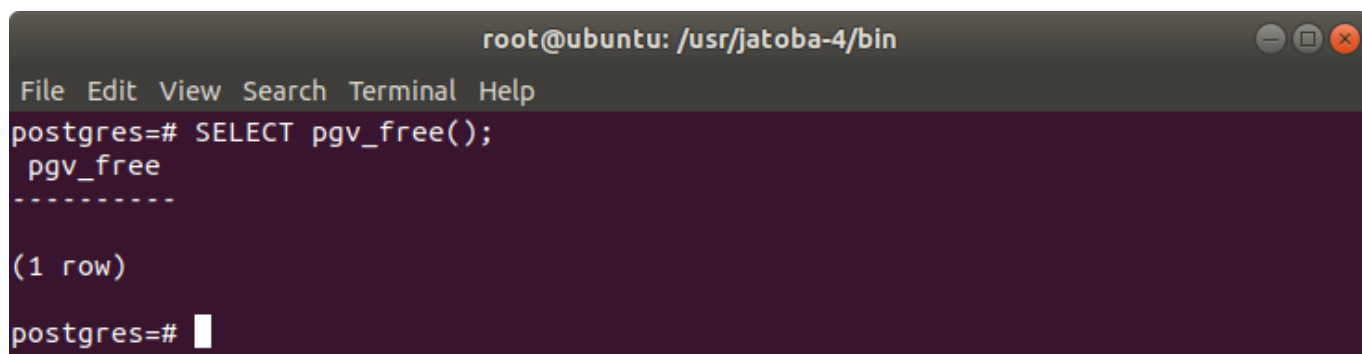
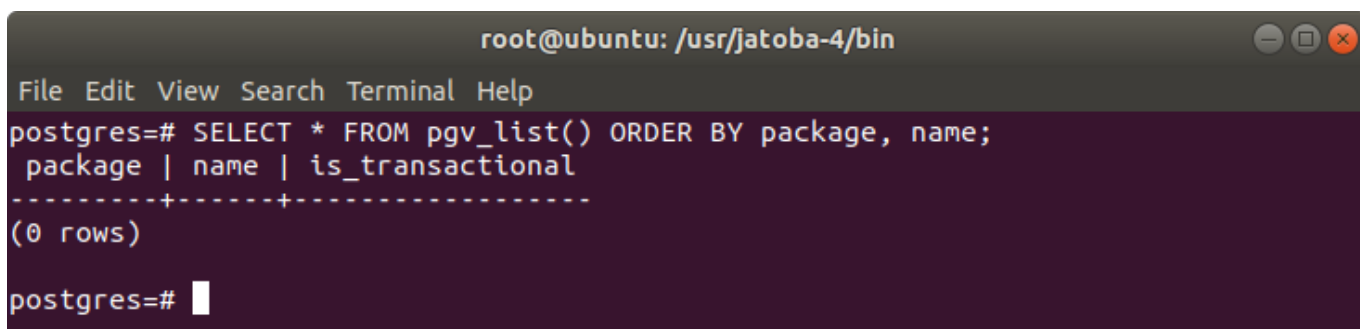


Рисунок 4.36 – Удаление пакетов и переменных

Функция удаления выполнена.

- Вывести все пакеты и переменные:

```
SELECT * FROM pgv_list() ORDER BY package, name;
```



```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT * FROM pgv_list() ORDER BY package, name;
 package | name | is_transactional
-----+-----+-----
(0 rows)

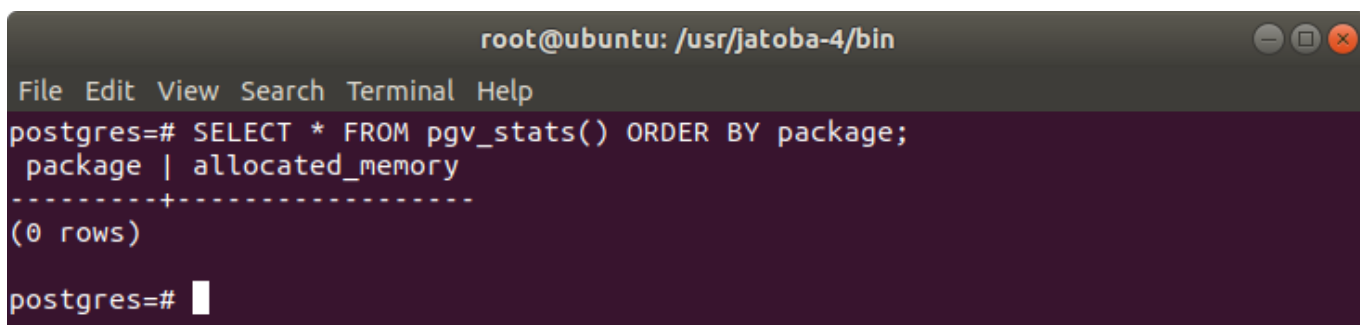
postgres=#
```

Рисунок 4.37 – Вывод списка пакетов и переменных

Список пуст.

- Убедиться, что память свободна:

```
SELECT * FROM pgv_stats() ORDER BY package;
```



```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT * FROM pgv_stats() ORDER BY package;
 package | allocated_memory
-----+-----
(0 rows)

postgres=#
```

Рисунок 4.38 – Вывод состояния памяти

#### 4.10. Проверка на существование пакета

Функция «pgv\_exists» обладает функциональными возможностями проверки на существование пакета и (или) переменной

Возвращает значение «true», если существует указанный пакет и (или) переменная существует.

Функция имеет следующий синтаксис:

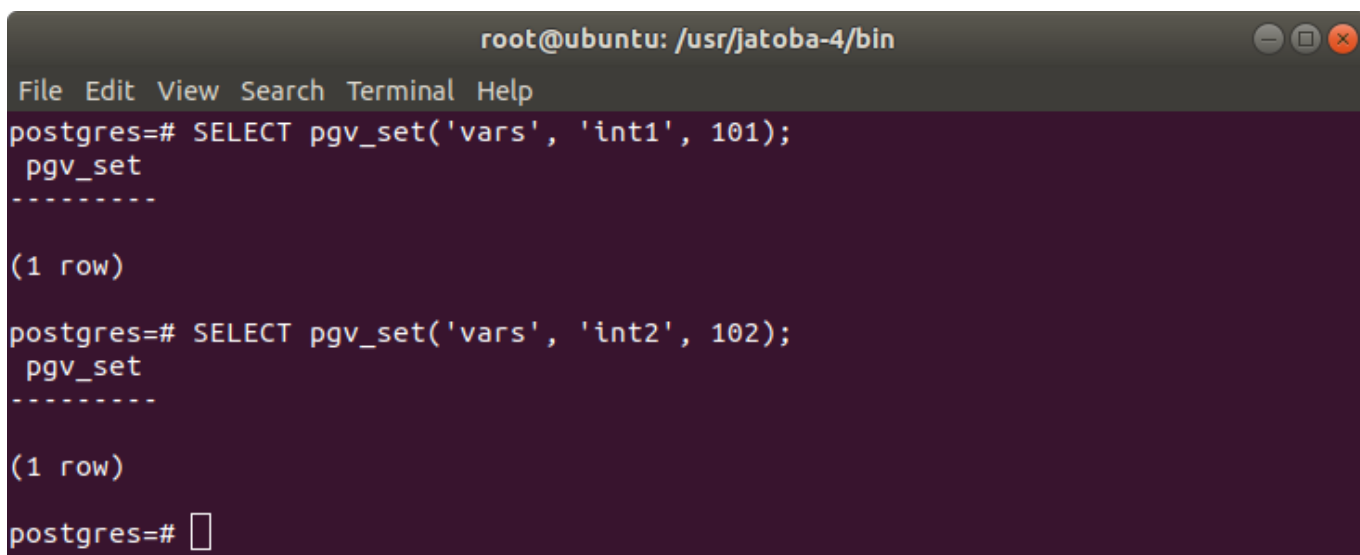
```
pgv_exists(package text)
```

Например.

- Создать пакет и несколько переменных:

```
# SELECT pgv_set('vars', 'int1', 101);
# SELECT pgv_set('vars', 'int2', 102);
```





```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT pgv_set('vars', 'int1', 101);
 pgv_set
-----
(1 row)

postgres=# SELECT pgv_set('vars', 'int2', 102);
 pgv_set
-----
(1 row)

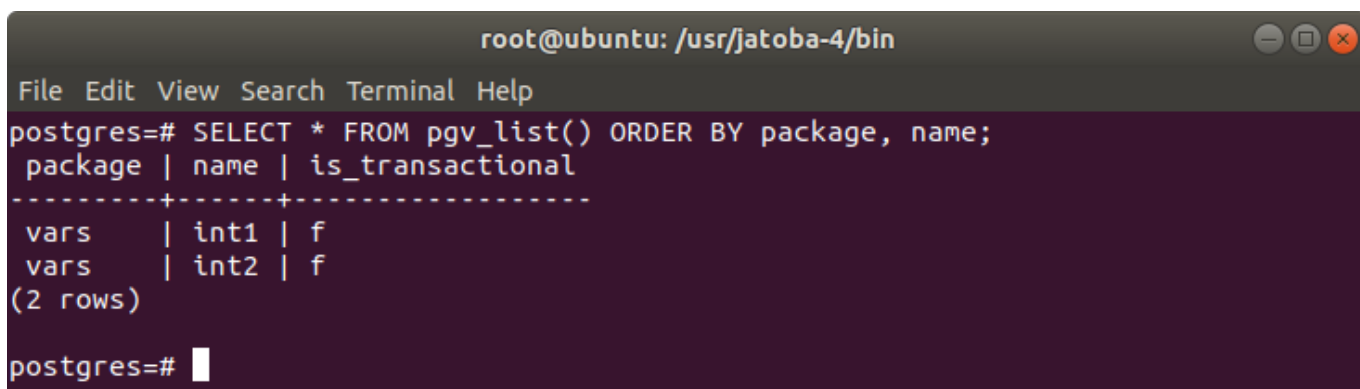
postgres=#
```

Рисунок 4.39 – Создание пакета и нескольких переменных

Пакет с переменными созданы.

- Вывести пакеты и переменные:

```
SELECT * FROM pgv_list() ORDER BY package, name;
```



```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# SELECT * FROM pgv_list() ORDER BY package, name;
 package | name | is_transactional
-----+-----+-----
 vars    | int1 | f
 vars    | int2 | f
(2 rows)

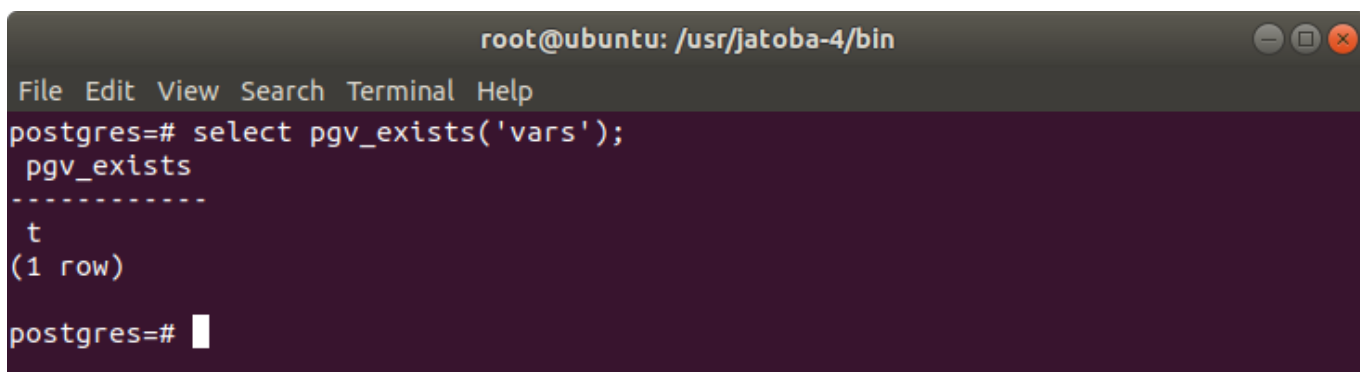
postgres=#
```

Рисунок 4.40 – Вывод состояния пакетов и переменных

Созданные пакет и переменные отображены.

- Выполнить функцию, проверяющую существование указанного пакета:

```
SELECT pgv_exists('vars');
```



```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# select pgv_exists('vars');
 pgv_exists
-----
t
(1 row)

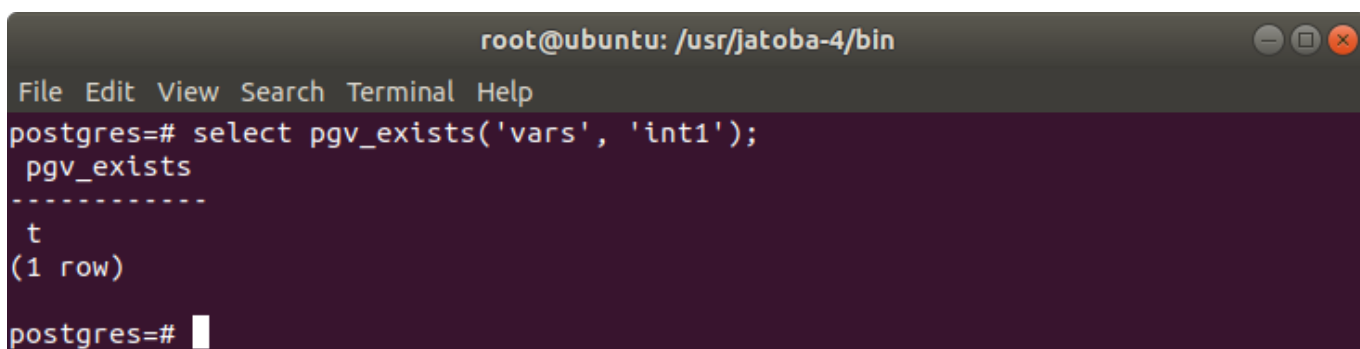
postgres=#
```

Рисунок 4.41 – Вывод состояния пакета

Указанный пакет существует, т.к. выведено значение «t» (true).

- Выполнить функцию, проверяющую существование указанного пакета и переменной:

```
SELECT pgv_exists('vars', 'int1');
```



```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
postgres=# select pgv_exists('vars', 'int1');
 pgv_exists
-----
t
(1 row)

postgres=#
```

Рисунок 4.42 – Вывод состояния пакета и переменной

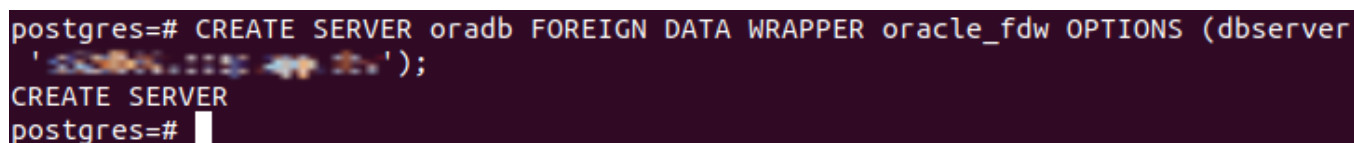
Указанный пакет и переменная существуют, т.к. выведено значение «t» (true).

## 5. ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ КОМПОНЕНТА ORACLE\_FDW

### 5.1. Добавление внешней таблицы Oracle в СУБД «Jatoba»

Чтобы подключиться к базе данных Oracle и в дальнейшем обращаться к ней под именем «oradb», необходимо от имени привилегированного пользователя выполнить команду:

```
CREATE SERVER oradb FOREIGN DATA WRAPPER oracle_fdw
        OPTIONS (dbserver '<строка подключения>');
```



```
postgres=# CREATE SERVER oradb FOREIGN DATA WRAPPER oracle_fdw OPTIONS (dbserver
'1521/ORDS101:1521/ORDS101');
CREATE SERVER
postgres=#
```

Рисунок 5.1 – Пример выполнения команды CREATE SERVER

Параметры для использования в OPTIONS указаны в таблице 5.1.

Таблица 5.1 – Параметры для команды CREATE SERVER

Название	Обязательный	Значение по умолчанию	Описание
dbserver	+		Строка подключения Oracle. Например: <code>'//dbserver.mydomain.com:1521/ORDADB'</code>
isolation_level	-	serializable	Уровень изолированности транзакций к БД Oracle. Значения параметра: <ul style="list-style-type: none"> <li>• serializable,</li> <li>• read_committed,</li> <li>• read_only.</li> </ul>
nchar	-	off	При значении параметра «on» преобразование символов на стороне Oracle будет более затратным, однако это необходимо в случае если в Oracle используется однобайтовая кодировка,

Название	Обязательный	Значение по умолчанию	Описание
			но в таблицах есть данные NCHAR или NVARCHAR2, содержащие символы, которые не могут быть отражены базой данных.
set_timezone	-	off	<p>При значении параметра «on» в БД Oracle на время сессии установится часовой пояс, выставленный в БД Jatoba. Установить параметр в «on» может быть полезно, если во внешней таблице Jatoba требуется использовать тип данных timestamp without time zone, а в Oracle у соответствующей колонки используется тип TIMESTAMP WITH LOCAL TIME ZONE.</p> <p>При использовании значения, которое не определено на сервере Oracle, возникнет ошибка:</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;">ORA-01882: timezone region not found</div>

Чтобы дать доступ к «oradb» другому (не привилегированному) пользователю Jatoba, выполнить команду:

```
GRANT USAGE ON FOREIGN SERVER oradb TO <имя пользователя Jatoba>;
```

Чтобы дать право пользователю Jatoba выполнять команды в БД Oracle, выполнить команду:

```
CREATE USER MAPPING FOR <имя пользователя Jatoba> SERVER oradb
```

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

```
OPTIONS (user '<имя пользователя БД Oracle>',  
password '<пароль пользователя БД Oracle>');
```

```
postgres=# CREATE USER MAPPING FOR postgres SERVER oradb OPTIONS (user 'oradb', p  
assword 'oradb');  
CREATE USER MAPPING  
postgres=#
```

Рисунок 5.2 – Пример выполнения команды CREATE USER MAPPING

Параметры для использования в OPTIONS указаны в таблице 5.2.

Таблица 5.2 – Параметры для команды CREATE USER MAPPING

Название параметра	Обязательный	Описание
user	+	Имя пользователя БД Oracle.
password	+	Пароль для пользователя БД Oracle.

Чтобы получить доступ к внешней таблице, выполнить команду CREATE FOREIGN TABLE, определив ее колонки. Например:

```
CREATE FOREIGN TABLE oratab (  
    id integer OPTIONS (key 'true') NOT NULL,  
    text character varying(30),  
    floating double precision NOT NULL  
    ) SERVER oradb OPTIONS (schema '<схема Oracle>',  
table '<название таблицы Oracle>');
```

```
postgres=# CREATE FOREIGN TABLE oratab (id integer OPTIONS (key 'true') NOT NULL  
,text character varying(30), floating double precision NOT NULL) SERVER oradb OP  
TIONS (schema 'oradb', table 'FDW_TEST');  
CREATE FOREIGN TABLE  
postgres=#
```

Рисунок 5.3 – Пример выполнения команды CREATE FOREIGN TABLE

Параметры внешней таблицы для использования в OPTIONS указаны в таблице 5.3.

Параметры колонок указаны в таблице 5.4.

Соответствие типов данных JatoBa и типов Oracle описано в подразделе 5.2.4.

Также можно добавить целиком схему Oracle (все таблицы). Инструкция находится в подразделе 5.2.10.

Таблица 5.3 – Параметры для команды CREATE FOREIGN TABLE

Название параметра	Обязательный	Значение по умолчанию	Описание
table	+		Название таблицы Oracle.
dblink	-		Oracle database link (ссылка на базу данных).
schema	-		Схема, к которой принадлежит таблица.
max_long	-	32767	<p>Максимальная длина значений типов LONG, LONG RAW и XMLTYPE. Значение параметра должно быть от 1 до 1073741823.</p> <p>Если в таблице есть значение больше, при запросе выведется ошибка:</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> ORA-01406: fetched column value was truncated </div>
readonly	-	false	Если установить параметр в true, то будет доступна запись во внешнюю таблицу (операции INSERT, UPDATE и DELETE).
sample_percent	-	100	Процент строк внешней таблицы, которые будут случайно выбраны для подсчета статистики (операция ANALYZE). Значение параметра должно быть от 0.000001 до 100.
prefetch	-	50	Количество строк, которые будут предзагружены из внешней таблицы за один цикл. Значение параметра должно быть от 1 до 1000. Высокие значения могут улучшить

Название параметра	Обязательный	Значение по умолчанию	Описание
			производительность, но будут занимать больше места на сервере. Если какая-то колонка внешней таблицы имеет тип MDSYS.SDO_GEOMETRY, предзагружаться таблица не будет.
lob_prefetch	-	1048576	Количество байтов, которые будут предзагружены для значений типов BLOB, CLOB и BFILE. Для загрузки значений, которые превышают значение этого параметра, потребуется дополнительный цикл обмена данными.

Таблица 5.4 – Параметры колонок

Название параметра	Обязательный	Значение по умолчанию	Описание
key	-	false	Если установлен в «true», соответствующая колонка внешней таблицы пометится как первичный ключ. Для проведения операций UPDATE и DELETE необходимо указывать через этот параметр корректные первичные ключи.
strip_zeros	-	false	Если установлен в «true», символы ASCII 0 будут удалены из всех строковых значений во время переноса внешней таблицы. Эти символы считаются валидными в Oracle, но не в СУБД Jatoba,

Название параметра	Обязательный	Значение по умолчанию	Описание
			поэтому их выгрузка может привести к ошибке. Параметр работает только со значениями типов char, varchar и text.

После указания внешней таблицы, с ней можно работать как с таблицей СУБД «Jatoba».

Например, выполнить команду SELECT:

```
postgres=# SELECT * FROM ORATAB;
WARNING: no Oracle character set for database encoding "SQL_ASCII"
DETAIL: All but ASCII characters will be lost.
HINT: You can set the option "nls_lang" on the foreign data wrapper to force an
Oracle character set.
 id | text | floating
-----+-----+-----
  1 | abc  |          3
(1 row)

postgres=#
```

Рисунок 5.4 – Пример выполнения команды SELECT

## 5.2. Использование компонента

### 5.2.1. Необходимые права в Oracle

Пользователь, через которого идет соединение к Oracle, должен иметь право создавать сессии (CREATE SESSION) и читать таблицу.

### 5.2.2. Соединения

Компонент кеширует соединения к Oracle. Все соединения будут автоматически завершены при завершении сессии Jatoba.

Чтобы вручную завершить все соединения Oracle, можно использовать функцию `oracle_close_connections()`. Например, ее можно вызывать при долгой сессии, во время которой редко происходят обращения ко внешним таблицам, чтобы не блокировать ресурсы, требующиеся для соединения с БД Oracle.

Это функцию невозможно вызвать внутри транзакции, которая меняет данные в БД Oracle.

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



### 5.2.3. Таблицы

Поля таблицы Oracle сопоставляются с колонками таблицы Jatoba в том порядке, в котором они определялись при выполнении команды CREATE FOREIGN TABLE.

При выполнении запроса к внешней таблице компонент будет обращаться только к тем колонкам таблицы Oracle, информация из которых необходима для выполнения запроса.

Во внешней таблице Jatoba может быть больше или меньше колонок, чем в таблице Oracle. Если в таблице Jatoba больше полей, запрос вернет значение NULL для соответствующих полей и покажет предупреждение.

Перед выполнением UPDATE или DELETE необходимо убедиться, что параметр «key» выставлен для всех колонок, которые являются первичными ключами таблицы. Если параметр не выставлен, запрос вернет ошибку.

### 5.2.4. Типы данных

Для колонок внешней таблицы Jatoba необходимо определить те типы данных, которые компонент может сопоставить с типами Oracle (см. таблицу 5.5). Если размер значения в таблице Oracle больше, чем размер поля в таблице Jatoba (например, если длина типа varchar или максимальное число integer в колонке Oracle больше, чем в Jatoba), в процессе выполнения запроса вернется ошибка.

Таблица 5.5 – Соответствие типов данных Oracle и Jatoba

Тип данных Oracle	Возможный тип данных Jatoba
CHAR	char, varchar, text
NCHAR	char, varchar, text
VARCHAR	char, varchar, text
VARCHAR2	char, varchar, text, json
NVARCHAR2	char, varchar, text
CLOB	char, varchar, text. json
LONG	char, varchar, text
RAW	uuid, bytea
BLOB	bytea
BFILE	bytea (read-only)
LONG RAW	bytea

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

Тип данных Oracle	Возможный тип данных Jatoba
NUMBER	numeric, float4, float8, char, varchar, text
NUMBER(n,m) при m<=0	numeric, float4, float8, int2, int4, int8, boolean, char, varchar, text
FLOAT	numeric, float4, float8, char, varchar, text
BINARY_FLOAT	numeric, float4, float8, char, varchar, text
BINARY_DOUBLE	numeric, float4, float8, char, varchar, text
DATE	date, timestamp, timestamptz, char, varchar, text
TIMESTAMP	date, timestamp, timestamptz, char, varchar, text
TIMESTAMP WITH TIME ZONE	date, timestamp, timestamptz, char, varchar, text
TIMESTAMP WITH LOCAL TIME ZONE	date, timestamp, timestamptz, char, varchar, text
INTERVAL YEAR TO MONTH	interval, char, varchar, text
INTERVAL DAY TO SECOND	interval, char, varchar, text
XMLTYPE	xml, char, varchar, text
MDSYS.SDO_GEOMETRY	geometry

При сопоставлении типа NUMBER (Oracle) и boolean (Jatoba) значение «0» будет конвертироваться в false, все остальное — в true.

Добавление или изменение XMLTYPE работает только со значениями, которые не превышают максимальное значение типа VARCHAR2 (4000 или 32767, в зависимости от значения параметра MAX\_STRING\_SIZE)

Тип NCLOB в текущее время не поддерживается, потому что Oracle не может автоматически его конвертировать.

При конвертации типа TIMESTAMP WITH LOCAL TIME ZONE (Oracle) в timestamp (Jatoba) необходимо настроить параметр set\_timezone в БД Oracle.

Если необходима специфическая конвертация, не указанная в таблице Таблица 5.5, можно определить подходящее представление в БД Oracle или БД Jatoba.

### 5.2.5. Операторы WHERE и ORDER BY

При отправлении запроса, содержащего условия WHERE, к внешней таблице компонент преобразует запрос к Oracle таким образом, чтобы он содержал те же условия

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

WHERE, если Oracle может их обработать. Это, во-первых, позволяет уменьшить количество строк, которые загружаются из Oracle, а во-вторых, позволяет Oracle выбрать оптимальный способ обращения к нужным таблицам.

Точно так же, ORDER BY условия будут отправлены в запросе к Oracle, если есть такая возможность. Однако условия ORDER BY, которые сортируют результат по строке символов, не будут отправлены к Oracle, потому что порядок сортировки в Jatoba и Oracle может отличаться.

Запросы, содержащие условия, будут отправлены на сервер Oracle целиком с большей вероятностью, если использовать простые условия и если при создании внешней таблицы в Jatoba использовать типы данных, соответствующие типам Oracle (см. таблицу Таблица 5.5).

Выражения now(), transaction\_timestamp(), current\_timestamp, current\_date и localtimestamp будут транслированы в Oracle корректно.

Чтобы посмотреть, какая часть запроса Jatoba к внешней таблице будет отправлена в Oracle, можно использовать команду EXPLAIN (см. подраздел 5.2.8).

#### **5.2.6. Использование оператора JOIN с внешними таблицами**

JOIN таблиц будет обрабатываться на стороне Oracle, если:

- обе таблицы определены на одном и том же внешнем сервере;
- запрос с JOIN содержит не более двух таблиц;
- запрос с JOIN является запросом SELECT;
- условия WHERE запроса могут быть обработаны на стороне Oracle (см. подраздел 5.2.5);
- если в запросе есть CROSS JOIN, он содержит условия.

Если в запросе есть и оператор JOIN, и оператор ORDER BY, то в случае, если JOIN может выполняться на стороне Oracle, ORDER BY выполняется на стороне Jatoba.

Важно чтобы статистика по обоим внешним таблицам через ANALYZE (см. подраздел 5.2.9) была собрана, чтобы СУБД «Jatoba» могла выбрать оптимальный способ сделать JOIN.

### 5.2.7. Изменение данных внешней таблицы

Oracle\_FDW поддерживает операции INSERT, UPDATE и DELETE над внешними таблицами. По умолчанию эти операции разрешены, но могут быть запрещены с помощью параметра «readonly» (см. таблицу Таблица 5.3).

Чтобы операции UPDATE и DELETE выполнялись, у колонок, соответствующих первичным ключам в таблице Oracle, должен быть указан параметр «key» (см. таблицу Таблица 5.4). Эти колонки используются для идентификации строк внешней таблицы.

Если при выполнении команды INSERT значение какой-то колонки внешней таблицы не было указано, будет использовано значение по умолчанию, указанное при создании внешней таблицы Jatoba, или NULL. Значение по умолчанию, указанное для соответствующей колонки в Oracle, будет использоваться только в случае, если эта колонка не была проинициализирована при создании внешней таблицы Jatoba.

Условие RETURNING для команд INSERT, UPDATE и DELETE будет работать, кроме случаев, когда в условии указаны колонки, которые в Oracle имеют тип LONG и LONG RAW (Oracle не поддерживает RETURNING для этих типов данных).

Триггеры для внешних таблиц поддерживаются. Триггеры, которые определяются через AFTER и FOR EACH ROW требуют, чтобы во внешних таблицах не было колонок, которые в Oracle имеют тип LONG и LONG RAW, поскольку эти триггеры используют условие RETURNING, об ограничениях которого упомянуто выше.

Производительность запросов, изменяющих данные во внешних таблицах, низкая, особенно когда меняется много строк, потому что строки обрабатываются по отдельности.

Операции BEGIN, COMMIT, ROLLBACK и SAVEPOINT компонентом поддерживаются.

Подготовленные запросы (prepared statements) ко внешним таблицам не поддерживаются.

По умолчанию oracle\_fdw использует уровень изоляции «serializable», поэтому запросы, меняющие данные во внешних таблицах, могут привести к ошибке:

```
ORA-08177: can't serialize access for this transaction
```

Такая ошибка может возникнуть при параллельном выполнении транзакций, особенно если они выполняются долго. Ошибки этого типа обозначены как SQLSTATE (40001). Приложение, которое использует Oracle\_FDW должно заново отправить транзакцию, если произошла ошибка этого типа.

Можно также поменять уровень изоляции транзакции (см. параметр «isolation\_level» в таблице 5.1).

### 5.2.8. Оператор EXPLAIN

Операция EXPLAIN к внешней таблице покажет запрос, который отправится к серверу Oracle. EXPLAIN VERBOSE покажет план исполнения запроса в Oracle (не будет работать в Oracle server 9i или старше).

### 5.2.9. Оператор ANALYZE

Оператор ANALYZE можно использовать на внешней таблице, чтобы собрать статистику.

Без сбора статистики Jatoba не сможет оценить количество строк во время запросов к внешней таблице, из-за чего БД может выбирать неоптимальный план выполнения запросов. Jatoba не будет автоматически собирать статистику для внешних таблиц с использованием автоочистки (autovacuum daemon), как она это делает для обычных таблиц, поэтому важно исполнять команду ANALYZE для внешней таблицы после ее создания и после существенных изменений в таблице.

Выполнение операции ANALYZE над внешней таблицей потребует полного последовательного сканирования таблицы. Чтобы ускорить выполнение операции, можно указать меньшее значение параметра «sample\_percent» (см. таблицу 5.3).

### 5.2.10. Поддержка операции IMPORT FOREIGN SCHEMA

Операция IMPORT FOREIGN SCHEMA позволяет импортировать схему (набор таблиц) из БД Oracle.

Пример команды:

```
IMPORT FOREIGN SCHEMA <имя схемы Oracle> FROM SERVER oradb INTO  
<локальная схема> OPTIONS (case 'lower')
```

Правила использования операции:

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

– **IMPORT FOREIGN SCHEMA** создаст внешние таблицы для всех объектов в **ALL\_TAB\_COLUMNS** (таблицы, представления, материализованные представления). Синонимы не будут перенесены из схемы Oracle.

– Название схемы в команде должно совпадать с названием схемы в БД Oracle. Регистр также должен совпадать (в Oracle чаще всего используется верхний регистр). Поскольку Jatoba автоматически меняет регистр имен на нижний, необходимо указывать имя схемы в двойных кавычках (например, «SCOTT»).

– При использовании команд **LIMIT TO** или **EXCEPT** имена таблиц необходимо указывать в нижнем регистре (вне зависимости от того, какой регистр использовался в Oracle).

Параметры для использования в **OPTIONS** указаны в таблице 5.6.

Таблица 5.6 – Параметры для команды **IMPORT FOREIGN SCHEMA**

Название	Значение по умолчанию	Описание
case	smart	Регистр, в котором переносятся таблицы и названия столбцов. Значения параметра: <ul style="list-style-type: none"> <li>• keep – оставить тот же регистр, который был в Oracle;</li> <li>• lower – привести к нижнему регистру;</li> <li>• smart – привести к нижнему регистру только те имена, все символы которых во верхнем регистре.</li> </ul>
collation	default	Правило сортировки. Необходимо использовать названия из системной таблицы Jatoba (колонка collname в таблице pg_collation).
dblink		Oracle database link (ссылка на базу данных).
readonly	false	Параметр readonly, использующийся для всех таблиц схемы. Подробнее о параметре в таблице Таблица 5.3.

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

Название	Значение по умолчанию	Описание
max_long	32767	Параметр max_long, использующийся для всех таблиц схемы. Подробнее о параметре в таблице Таблица 5.3.
sample_percent	100	Параметр sample_percent, использующийся для всех таблиц схемы. Подробнее о параметре в таблице Таблица 5.3.
prefetch	50	Параметр prefetch, использующийся для всех таблиц схемы. Подробнее о параметре в таблице Таблица 5.3.
lob_prefetch	1048576	Параметр lob_prefetch, использующийся для всех таблиц схемы. Подробнее о параметре в таблице Таблица 5.3.
nchar		Параметр nchar, использующийся для всех таблиц схемы. Подробнее о параметре в таблице Таблица 5.1.
set_timezone		Параметр set_timezone, использующийся для всех таблиц схемы. Подробнее о параметре в таблице Таблица 5.1.

## 6. УДАЛЕНИЕ КОМПОНЕНТОВ

### 6.1. Удаление компонентов при отсутствии зависимых от него объектов

Удаление компонентов осуществляется средствами пакетного менеджера ОС. При этом нужно использовать команду удаления, соответствующую пакетному менеджеру: remove, purge, erase и т.п.

Для удаления компонента «orafce» потребуется авторизоваться в СУБД и выполнить команду:

```
DROP EXTENSION orafce;
```

В ОС GNU/Linux требуется выйти из psql и удалить пакет расширения, выполнив команду:

```
apt-get remove jatoba4-orafce
```

Для удаления компонента «pg\_variables» потребуется авторизоваться в СУБД и выполнить команду:

```
DROP EXTENSION pg_variables;
```

В ОС GNU/Linux требуется выйти из psql и удалить пакет расширения, выполнив команду:

```
apt-get remove jatoba4-pg-variables
```

Для удаления компонента «oracle\_fdw» потребуется авторизоваться в СУБД и выполнить команду:

```
DROP EXTENSION oracle_fdw;
```

В ОС GNU/Linux требуется выйти из psql и удалить пакет расширения, выполнив команду:

```
apt-get remove jatoba4-oracle-fdw
```



## **6.2. Удаление компонента при наличии зависимых от него объектов**

Для удаления компонента вместе со всеми зависимыми от него объектами потребуется авторизоваться в СУБД и выполнить команду:

```
# DROP EXTENSION orafce cascade;  
# DROP EXTENSION pg_variables cascade;  
# DROP EXTENSION oracle_fdw cascade;
```

## ПРИЛОЖЕНИЕ 1

### Пример установки СУБД «Jatoba» из локального репозитория для ОС Ubuntu с компонентами обеспечения работы с СУБД Oracle

Установка СУБД «Jatoba» из локального репозитория для ОС Ubuntu проводится в следующем порядке:

- 1) В терминале войти в режим суперпользователя, выполнив команду:

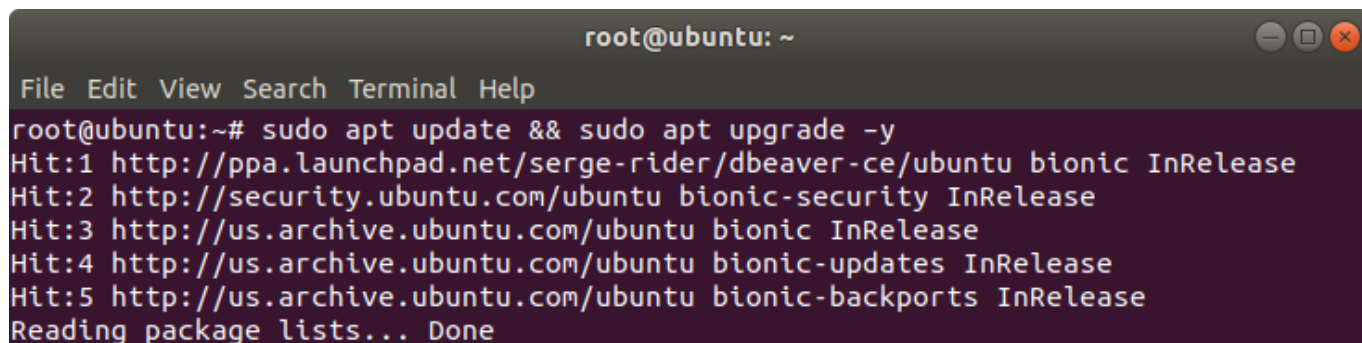
```
sudo su
```

- 2) Если команды sudo не существует – установить:

```
su -l  
apt-get install sudo -y
```

- 3) Выполнить обновление системы:

```
- sudo apt update && sudo apt upgrade -y  
- sudo apt -s dist-upgrade  
- sudo apt dist-upgrade
```



The screenshot shows a terminal window titled 'root@ubuntu: ~'. The terminal output displays the execution of 'sudo apt update && sudo apt upgrade -y'. It lists five 'Hit' messages for various Ubuntu repositories (ppa.launchpad.net, security.ubuntu.com, and us.archive.ubuntu.com) and ends with 'Reading package lists... Done'.

Рисунок П1.1 – Обновление системы

- 4) Создать папку localrepo в корневом каталоге:

```
mkdir /localrepo
```

- 5) В созданную папку скопировать:

- каталог <pool>;
- каталог <dist>;
- файл <DEB-GPG-KEY-Jatoba>

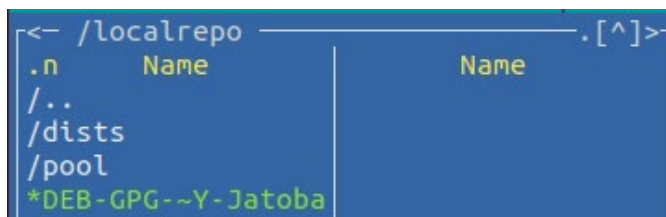


Рисунок П1.2 – Структура каталога «localrepo»

6) Установить открытый ключ репозитория:

```
apt-key add /localrepo/DEB-GPG-KEY-Jatoba
```

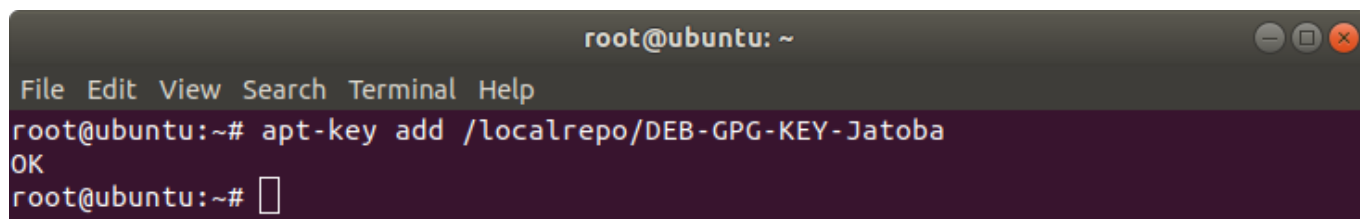


Рисунок П1.3 – Установка открытого ключа репозитория

7) Добавить описание локального репозитория в систему:

```
nano /etc/apt/sources.list.d/jatoba-4.list
```

8) Добавить в файл следующее содержимое и сохранить:

```
deb file:///localrepo stable non-free
```

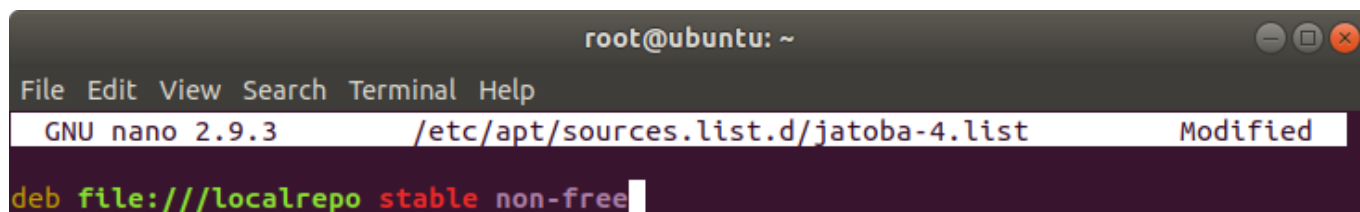
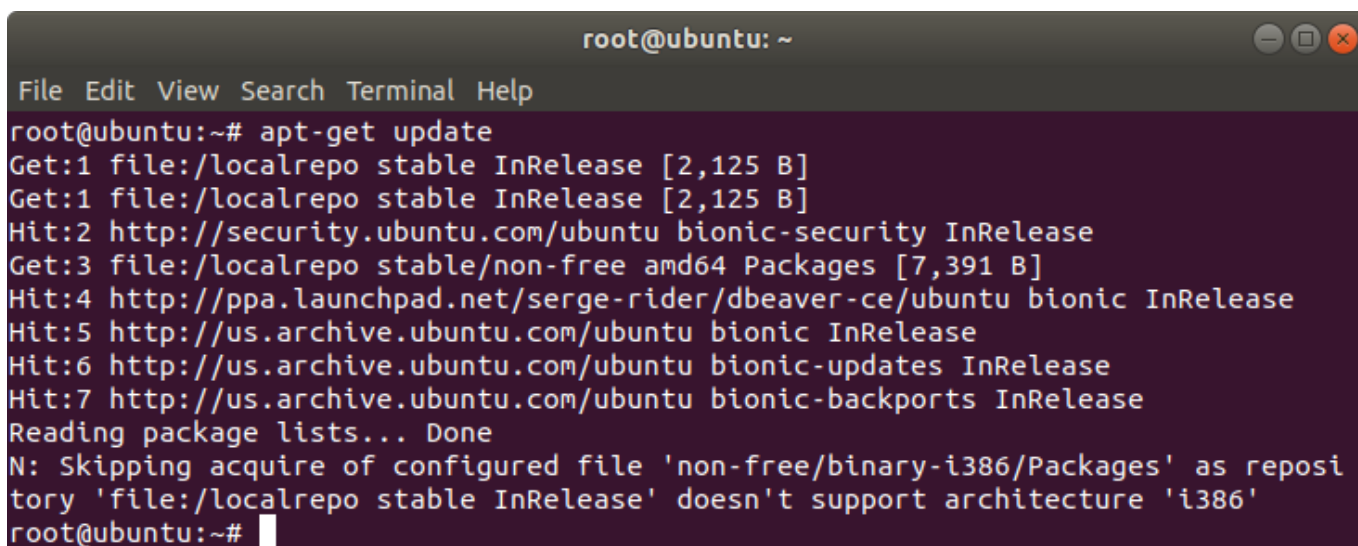


Рисунок П1.4 – Содержание файла «jatoba-4.list»

9) Проиндексировать обновленное состояние репозитория:

```
apt-get update
```

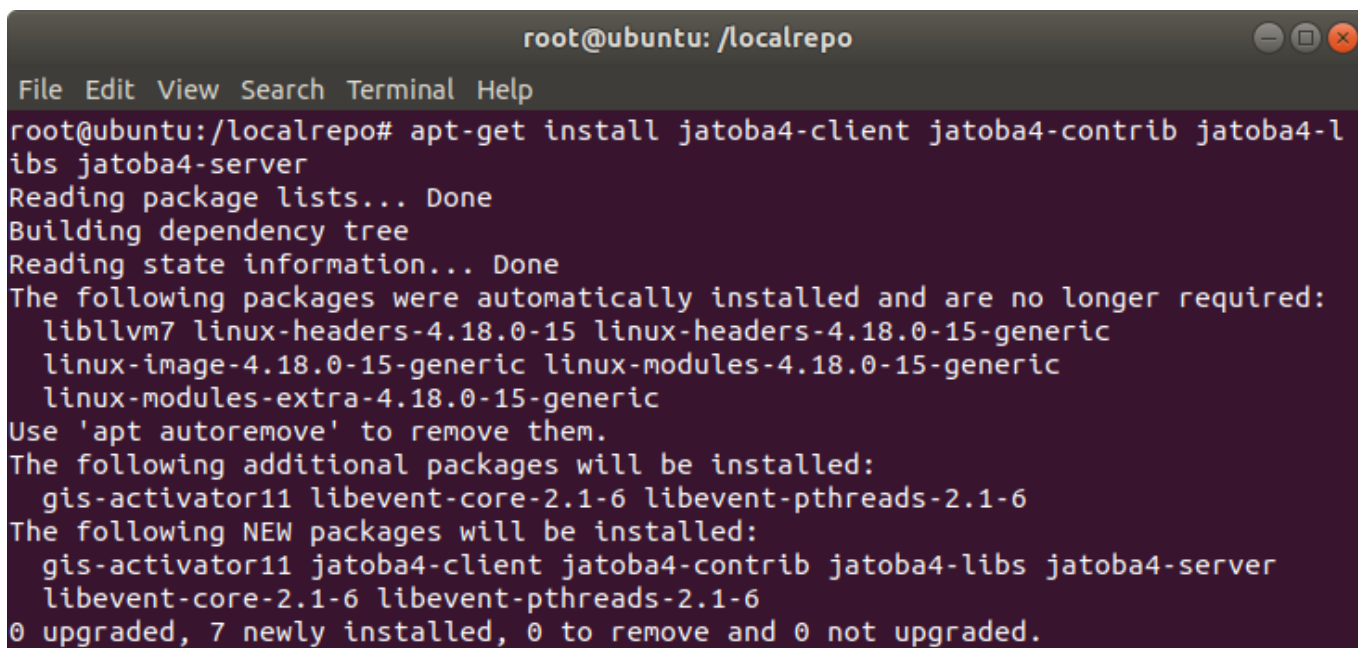


```
root@ubuntu: ~  
File Edit View Search Terminal Help  
root@ubuntu:~# apt-get update  
Get:1 file:/localrepo stable InRelease [2,125 B]  
Get:1 file:/localrepo stable InRelease [2,125 B]  
Hit:2 http://security.ubuntu.com/ubuntu bionic-security InRelease  
Get:3 file:/localrepo stable/non-free amd64 Packages [7,391 B]  
Hit:4 http://ppa.launchpad.net/serge-rider/dbeaver-ce/ubuntu bionic InRelease  
Hit:5 http://us.archive.ubuntu.com/ubuntu bionic InRelease  
Hit:6 http://us.archive.ubuntu.com/ubuntu bionic-updates InRelease  
Hit:7 http://us.archive.ubuntu.com/ubuntu bionic-backports InRelease  
Reading package lists... Done  
N: Skipping acquire of configured file 'non-free/binary-i386/Packages' as repository 'file:/localrepo stable InRelease' doesn't support architecture 'i386'  
root@ubuntu:~#
```

Рисунок П1.5 – Индексация репозитория

10) Установить основные пакеты СУБД «Jatoba»

```
apt-get install jatoba4-client jatoba4-contrib jatoba4-libs  
jatoba4-server
```



```
root@ubuntu: /localrepo  
File Edit View Search Terminal Help  
root@ubuntu:/localrepo# apt-get install jatoba4-client jatoba4-contrib jatoba4-libs jatoba4-server  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following packages were automatically installed and are no longer required:  
  libllvm7 linux-headers-4.18.0-15 linux-headers-4.18.0-15-generic  
  linux-image-4.18.0-15-generic linux-modules-4.18.0-15-generic  
  linux-modules-extra-4.18.0-15-generic  
Use 'apt autoremove' to remove them.  
The following additional packages will be installed:  
  gis-activator11 libevent-core-2.1-6 libevent-pthreads-2.1-6  
The following NEW packages will be installed:  
  gis-activator11 jatoba4-client jatoba4-contrib jatoba4-libs jatoba4-server  
  libevent-core-2.1-6 libevent-pthreads-2.1-6  
0 upgraded, 7 newly installed, 0 to remove and 0 not upgraded.
```

Рисунок П1.6 – Установка основных пакетов

11) Скачать внешний пакет для поддержки работы компонента oracle-fdw

Пакет oracle-instantclient11.2-basic-11.2.0.4.0-1.x86\_64 – Oracle® Instant Client версии 11.2 требуется для работы пакета jatoba4-oracle-fdw. Скачать его возможно по адресу:

<https://www.oracle.com/database/technologies/instant-client/linux-x86-64-downloads.html>

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

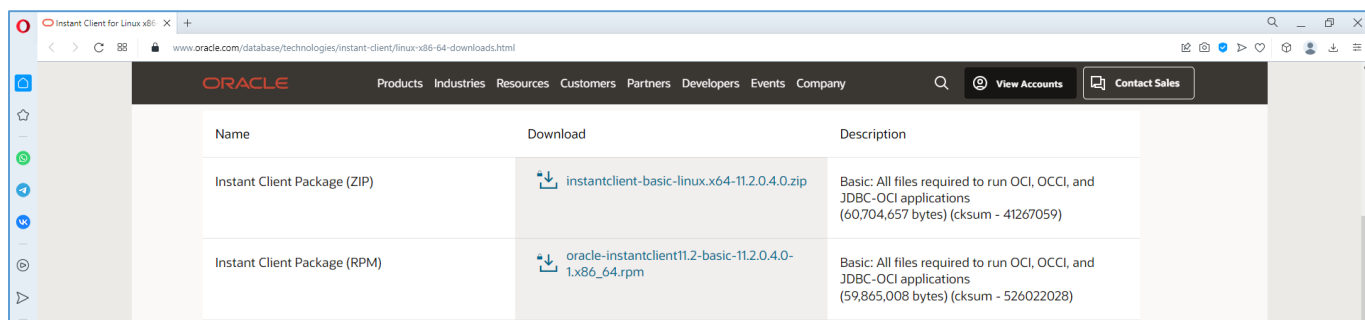


Рисунок П1.7 – Страница загрузки пакета

Скопировать пакет в директорию Download и перейти в нее.

12) Установить утилиту преобразования пакетных форматов Alien

```
sudo apt-get install alien
```

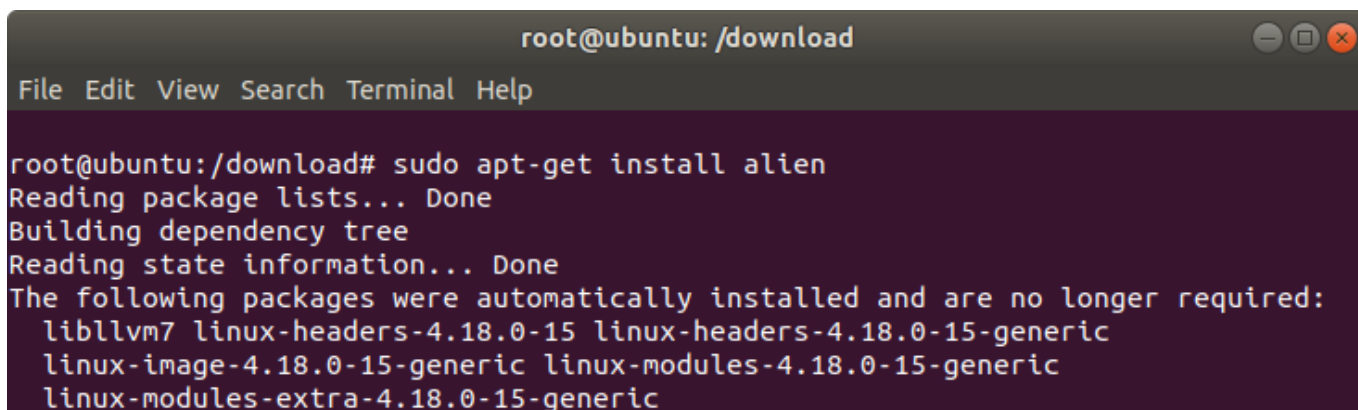


Рисунок П1.8 – Установка утилиты Alien

13) Установить пакет операционной системы libaio1

```
sudo apt-get install libaio1
```

```
root@ubuntu: ~
File Edit View Search Terminal Help
root@ubuntu:~# sudo apt-get install libaio1
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libllvm7 linux-headers-4.18.0-15 linux-headers-4.18.0-15-generic
  linux-image-4.18.0-15-generic linux-modules-4.18.0-15-generic
  linux-modules-extra-4.18.0-15-generic
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  libaio1
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 6,476 B of archives.
After this operation, 30.7 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu bionic-updates/main amd64 libaio1 amd6
4 0.3.110-5ubuntu0.1 [6,476 B]
Fetched 6,476 B in 1s (11.6 kB/s)
Selecting previously unselected package libaio1:amd64.
(Reading database ... 196037 files and directories currently installed.)
Preparing to unpack .../libaio1_0.3.110-5ubuntu0.1_amd64.deb ...
Unpacking libaio1:amd64 (0.3.110-5ubuntu0.1) ...
Setting up libaio1:amd64 (0.3.110-5ubuntu0.1) ...
Processing triggers for libc-bin (2.27-3ubuntu1.6) ...
root@ubuntu:~#
```

Рисунок П1.9 – Установка пакета операционной системы libaio1

14) Установить пакет клиентских библиотек СУБД Oracle oracle-instantclient11.2-basic-11.2.0.4.0-1.x86\_64.rpm

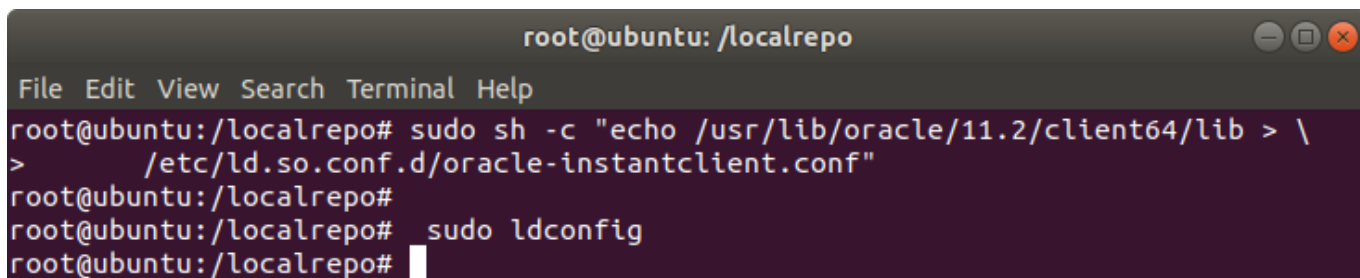
```
sudo alien -i oracle-instantclient11.2-basic-11.2.0.4.0-1.x86_64.rpm
```

```
root@ubuntu: /download
File Edit View Search Terminal Help
root@ubuntu:/download# sudo alien -i oracle-instantclient11.2-basic-11.2.0.4.0-1.x86_64.rpm
dpkg --no-force-overwrite -i oracle-instantclient11.2-basic_11.2.0.4.0-2_amd64.deb
Selecting previously unselected package oracle-instantclient11.2-basic.
(Reading database ... 196075 files and directories currently installed.)
Preparing to unpack oracle-instantclient11.2-basic_11.2.0.4.0-2_amd64.deb ...
Unpacking oracle-instantclient11.2-basic (11.2.0.4.0-2) ...
Setting up oracle-instantclient11.2-basic (11.2.0.4.0-2) ...
Processing triggers for libc-bin (2.27-3ubuntu1.6) ...
root@ubuntu:/download#
```

Рисунок П1.10 – Установка пакет клиентских библиотек СУБД Oracle

15) Выполнить конфигурирование установленной библиотеки

```
sudo sh -c "echo /usr/lib/oracle/11.2/client64/lib > \
/etc/ld.so.conf.d/oracle-instantclient.conf"
sudo ldconfig
```

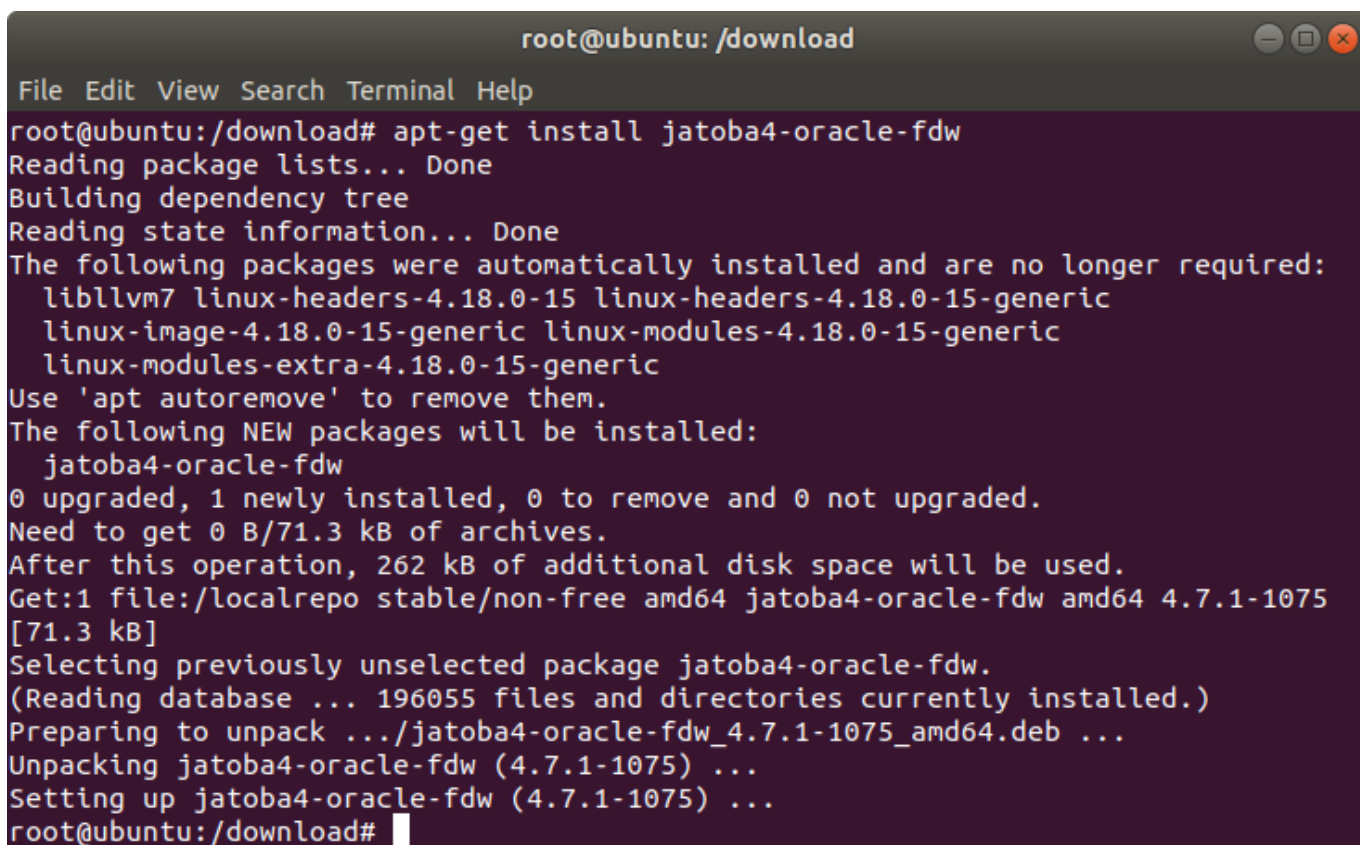
A terminal window titled 'root@ubuntu: /localrepo' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the execution of two commands: 'sudo sh -c "echo /usr/lib/oracle/11.2/client64/lib > \ /etc/ld.so.conf.d/oracle-instantclient.conf"' and 'sudo ldconfig'. The prompt returns to 'root@ubuntu: /localrepo#' after each command.

```
root@ubuntu: /localrepo
File Edit View Search Terminal Help
root@ubuntu: /localrepo# sudo sh -c "echo /usr/lib/oracle/11.2/client64/lib > \
> /etc/ld.so.conf.d/oracle-instantclient.conf"
root@ubuntu: /localrepo#
root@ubuntu: /localrepo# sudo ldconfig
root@ubuntu: /localrepo#
```

Рисунок П1.11 – Конфигурирование пакета

16) Установить компонент Oracle\_FDW

```
apt-get install jatoba4-oracle-fdw
```

A terminal window titled 'root@ubuntu: /download' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the execution of 'apt-get install jatoba4-oracle-fdw'. It displays the process of reading package lists, building a dependency tree, and identifying packages to be installed. It lists several packages that will be removed as they are no longer required. It then shows the details of the new package to be installed, including its size and disk space requirements. Finally, it shows the package being unpacked and set up.

```
root@ubuntu: /download
File Edit View Search Terminal Help
root@ubuntu: /download# apt-get install jatoba4-oracle-fdw
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libllvm7 linux-headers-4.18.0-15 linux-headers-4.18.0-15-generic
  linux-image-4.18.0-15-generic linux-modules-4.18.0-15-generic
  linux-modules-extra-4.18.0-15-generic
Use 'apt autoremove' to remove them.
The following NEW packages will be installed:
  jatoba4-oracle-fdw
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 0 B/71.3 kB of archives.
After this operation, 262 kB of additional disk space will be used.
Get:1 file:/localrepo stable/non-free amd64 jatoba4-oracle-fdw amd64 4.7.1-1075
[71.3 kB]
Selecting previously unselected package jatoba4-oracle-fdw.
(Reading database ... 196055 files and directories currently installed.)
Preparing to unpack .../jatoba4-oracle-fdw_4.7.1-1075_amd64.deb ...
Unpacking jatoba4-oracle-fdw (4.7.1-1075) ...
Setting up jatoba4-oracle-fdw (4.7.1-1075) ...
root@ubuntu: /download#
```

Рисунок П1.12 – Установка СУБД с компонентом Oracle\_FDW

17) Установить компонент OraFCE

```
apt-get install jatoba4-orafce
```



```
root@ubuntu: /localrepo
File Edit View Search Terminal Help
root@ubuntu:/localrepo# apt-get install jatoba4-orafce
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libllvm7 linux-headers-4.18.0-15 linux-headers-4.18.0-15-generic
  linux-image-4.18.0-15-generic linux-modules-4.18.0-15-generic
  linux-modules-extra-4.18.0-15-generic
Use 'apt autoremove' to remove them.
The following NEW packages will be installed:
  jatoba4-orafce
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 0 B/104 kB of archives.
After this operation, 586 kB of additional disk space will be used.
Get:1 file:/localrepo stable/non-free amd64 jatoba4-orafce amd64 4.7.1-1075 [104
kB]
Selecting previously unselected package jatoba4-orafce.
(Reading database ... 196061 files and directories currently installed.)
Preparing to unpack .../jatoba4-orafce_4.7.1-1075_amd64.deb ...
Unpacking jatoba4-orafce (4.7.1-1075) ...
Setting up jatoba4-orafce (4.7.1-1075) ...
root@ubuntu:/localrepo#
```

Рисунок П1.13 – Установка СУБД с компонентом OraFCE

18) Установить компонент pg\_Variables

```
apt-get install jatoba4-pg-variables
```

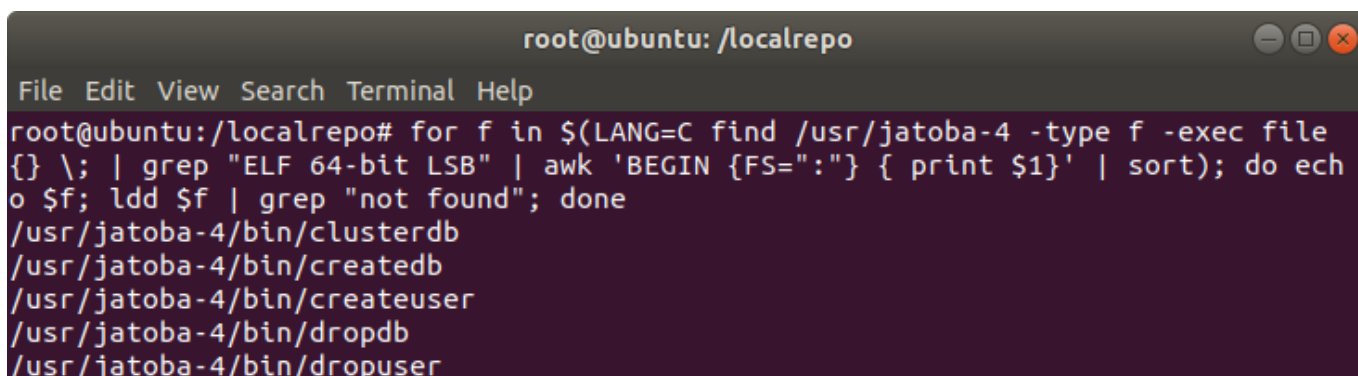
```
root@ubuntu: /localrepo
File Edit View Search Terminal Help
root@ubuntu:/localrepo# apt-get install jatoba4-pg-variables
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libllvm7 linux-headers-4.18.0-15 linux-headers-4.18.0-15-generic
  linux-image-4.18.0-15-generic linux-modules-4.18.0-15-generic
  linux-modules-extra-4.18.0-15-generic
```

Рисунок П1.14 – Установка СУБД с компонентом pg\_Variables

19) Убедиться в отсутствии ошибок зависимостей:

```
for f in $(LANG=C find /usr/jatoba-4 -type f -exec file {} \; |
grep "ELF 64-bit LSB" | awk 'BEGIN {FS=":"} { print $1}' |
sort); do echo $f; ldd $f | grep "not found"; done
```



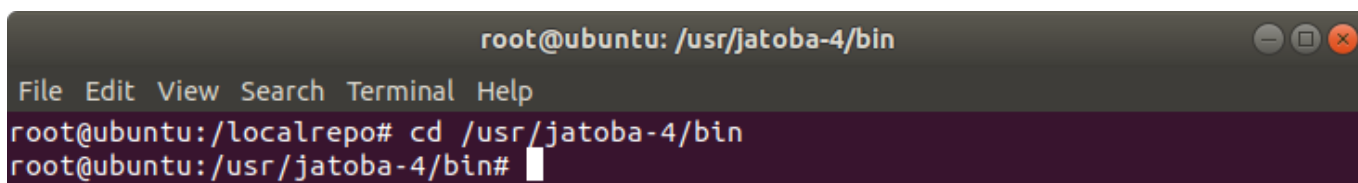


```
root@ubuntu: /localrepo
File Edit View Search Terminal Help
root@ubuntu:/localrepo# for f in $(LANG=C find /usr/jatoba-4 -type f -exec file {} \; | grep "ELF 64-bit LSB" | awk 'BEGIN {FS=":"} { print $1}' | sort); do echo $f; ldd $f | grep "not found"; done
/usr/jatoba-4/bin/clusterdb
/usr/jatoba-4/bin/createdb
/usr/jatoba-4/bin/createuser
/usr/jatoba-4/bin/dropdb
/usr/jatoba-4/bin/dropuser
```

Рисунок П1.15 – Проверка ошибок зависимостей

- 20) Перейти в директорию исполняемых файлов СУБД:

```
cd /usr/jatoba-4/bin
```

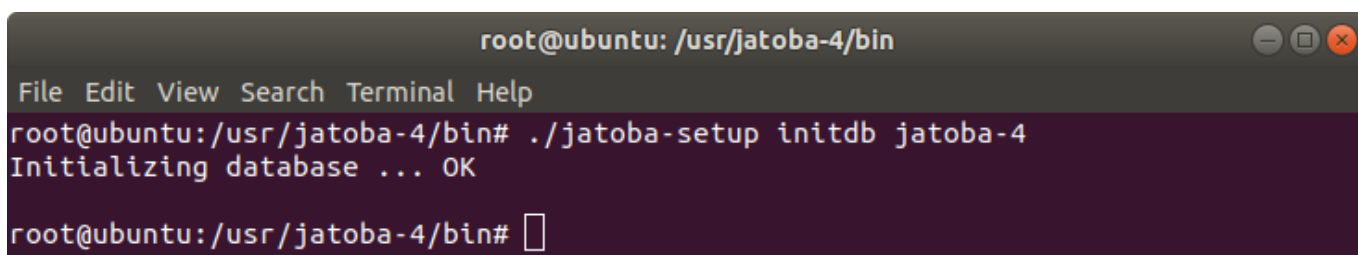


```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
root@ubuntu:/localrepo# cd /usr/jatoba-4/bin
root@ubuntu:/usr/jatoba-4/bin#
```

Рисунок П1.16 – Переход в каталог

- 21) Инициализировать каталог данных СУБД при помощи команды:

```
./jatoba-setup initdb jatoba-4
```

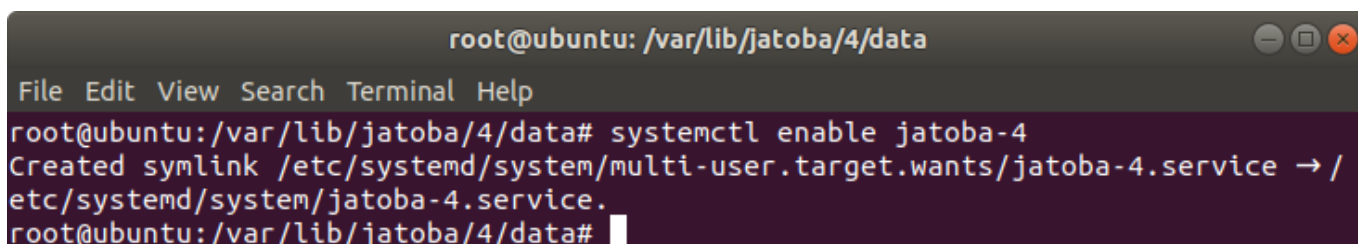


```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
root@ubuntu:/usr/jatoba-4/bin# ./jatoba-setup initdb jatoba-4
Initializing database ... OK
root@ubuntu:/usr/jatoba-4/bin#
```

Рисунок П1.17 – Инициализация СУБД

- 22) Добавить сервис в список автозапуска:

```
systemctl enable jatoba-4
```



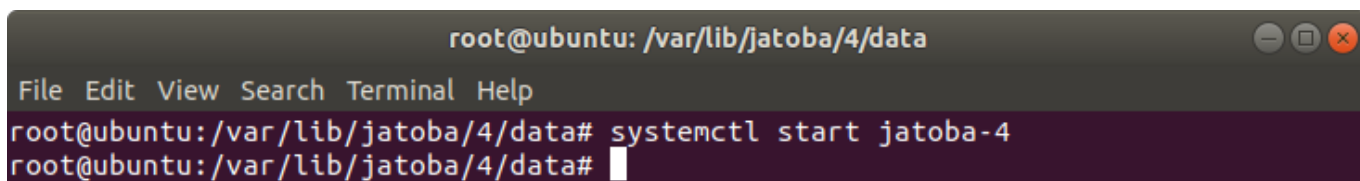
```
root@ubuntu: /var/lib/jatoba/4/data
File Edit View Search Terminal Help
root@ubuntu:/var/lib/jatoba/4/data# systemctl enable jatoba-4
Created symlink /etc/systemd/system/multi-user.target.wants/jatoba-4.service → /etc/systemd/system/jatoba-4.service.
root@ubuntu:/var/lib/jatoba/4/data#
```

Рисунок П1.18 – Добавление сервиса jatoba-4 а автозагрузку ОС

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

23) Запустить службу:

```
systemctl start jatoba-4
```

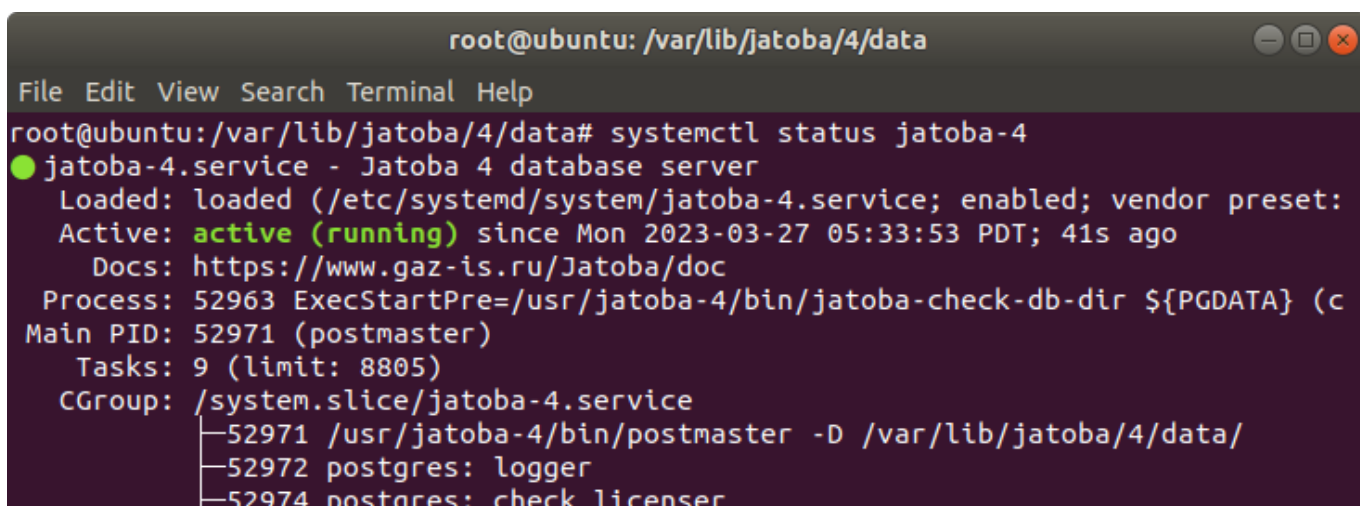


```
root@ubuntu: /var/lib/jatoba/4/data
File Edit View Search Terminal Help
root@ubuntu:/var/lib/jatoba/4/data# systemctl start jatoba-4
root@ubuntu:/var/lib/jatoba/4/data#
```

Рисунок П1.19 – Запуск службы jatoba-4

24) Проверить статус службы:

```
systemctl status jatoba-4
```



```
root@ubuntu: /var/lib/jatoba/4/data
File Edit View Search Terminal Help
root@ubuntu:/var/lib/jatoba/4/data# systemctl status jatoba-4
● jatoba-4.service - Jatoba 4 database server
   Loaded: loaded (/etc/systemd/system/jatoba-4.service; enabled; vendor preset:
   Active: active (running) since Mon 2023-03-27 05:33:53 PDT; 41s ago
     Docs: https://www.gaz-is.ru/Jatoba/doc
   Process: 52963 ExecStartPre=/usr/jatoba-4/bin/jatoba-check-db-dir ${PGDATA} (c
   Main PID: 52971 (postmaster)
     Tasks: 9 (limit: 8805)
    CGroup: /system.slice/jatoba-4.service
            └─52971 /usr/jatoba-4/bin/postmaster -D /var/lib/jatoba/4/data/
              └─52972 postgres: logger
                └─52974 postgres: check licenser
```

Рисунок П1.20 – Проверка статуса службы jatoba-4

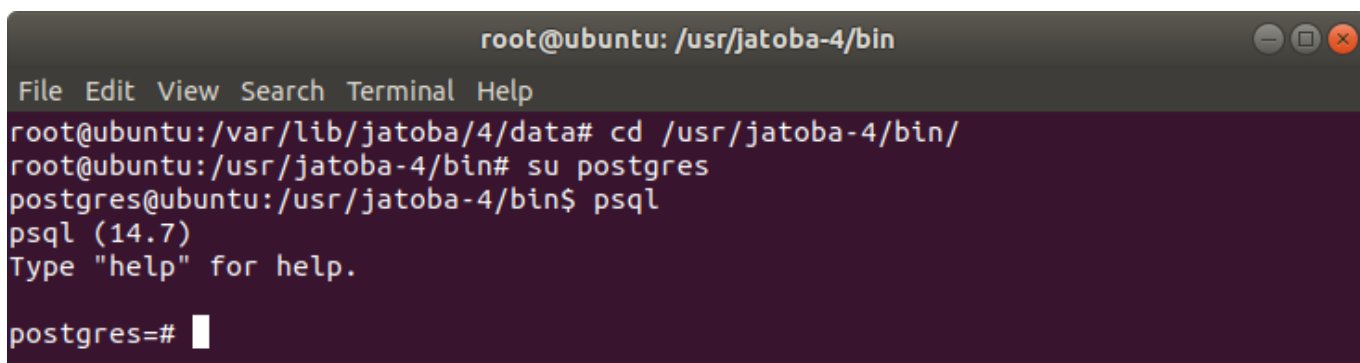
25) Установить пароль для пользователя СУБД postgres:

Необходимо выйти из сессии root:

```
su - postgres
```

и зайти в СУБД:

```
psql
```



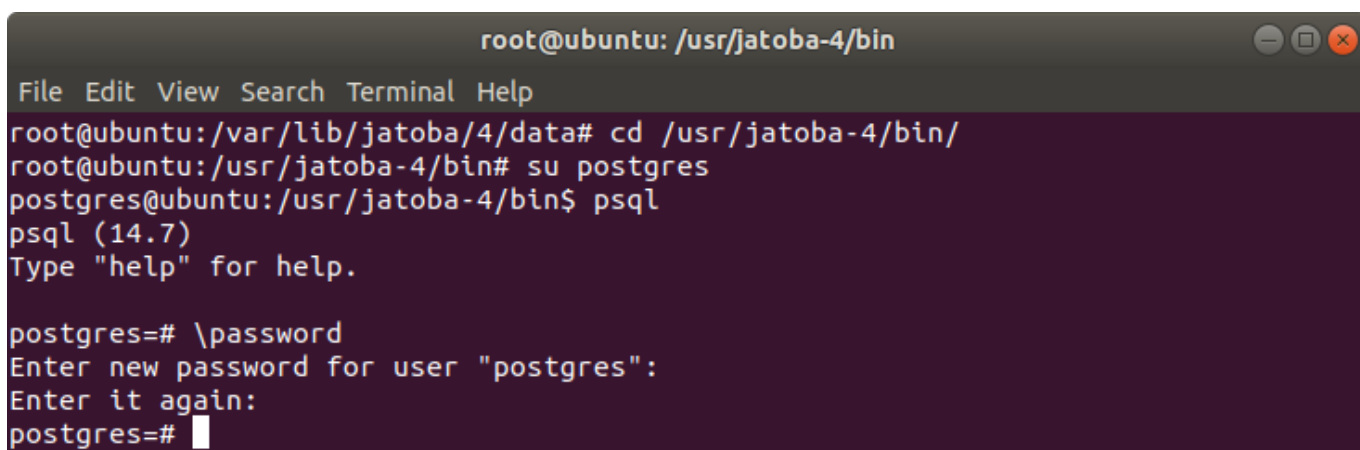
```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
root@ubuntu:/var/lib/jatoba/4/data# cd /usr/jatoba-4/bin/
root@ubuntu:/usr/jatoba-4/bin# su postgres
postgres@ubuntu:/usr/jatoba-4/bin$ psql
psql (14.7)
Type "help" for help.

postgres=#
```

Рисунок П1.21 – Вход в СУБД

Выполнить SQL-команду установки пароля:

```
\password
```



```
root@ubuntu: /usr/jatoba-4/bin
File Edit View Search Terminal Help
root@ubuntu:/var/lib/jatoba/4/data# cd /usr/jatoba-4/bin/
root@ubuntu:/usr/jatoba-4/bin# su postgres
postgres@ubuntu:/usr/jatoba-4/bin$ psql
psql (14.7)
Type "help" for help.

postgres=# \password
Enter new password for user "postgres":
Enter it again:
postgres=#
```

Рисунок П1.22 – Установка пароля для пользователя СУБД postgres

На этом этапе установка СУБД с компонентами обеспечения работы с СУБД Oracle и СУБД закончена.

## **ПЕРЕЧЕНЬ СОКРАЩЕНИЙ**

SQL	–	(Structured Query Language) — язык структурированных запросов
БД	–	База данных
ОС	–	Операционная система
СУБД	–	Система управления базами данных

[illegible]

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