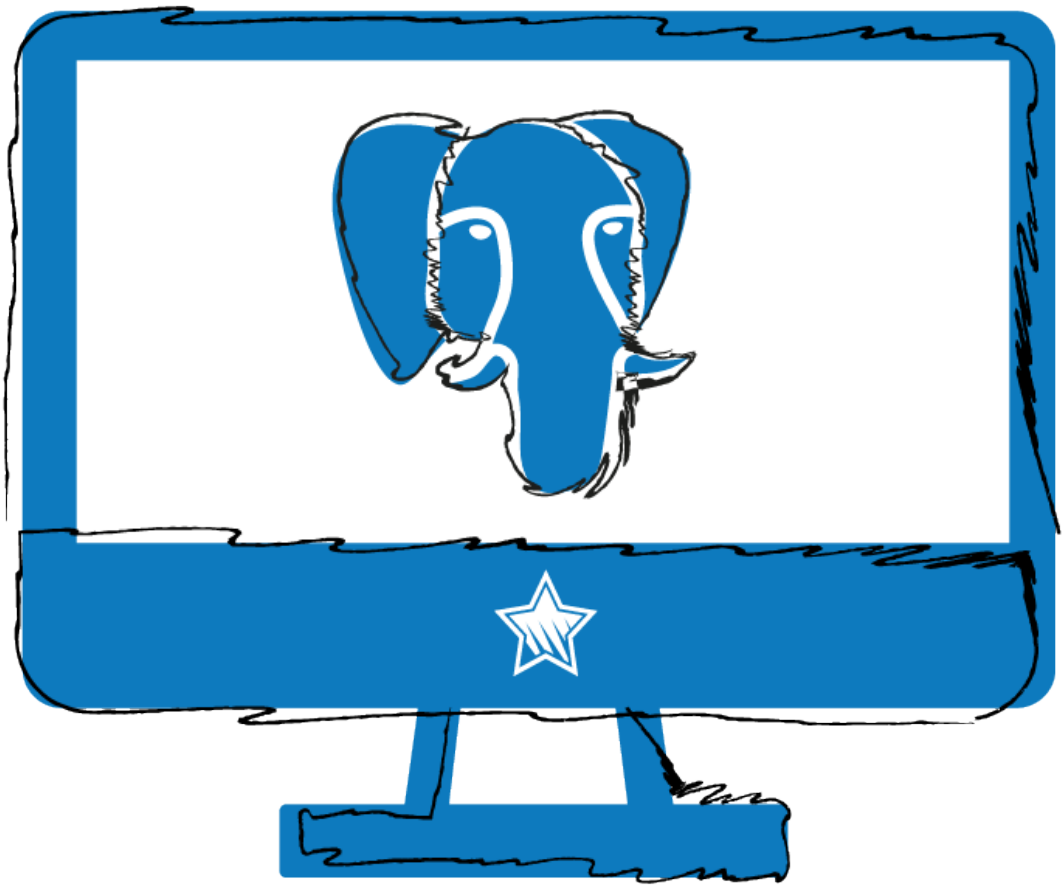
Курс AL-1707

СУБД PostgreSQL в Astra Linux Special Edition 1.7: установка, администрирование и мониторинг



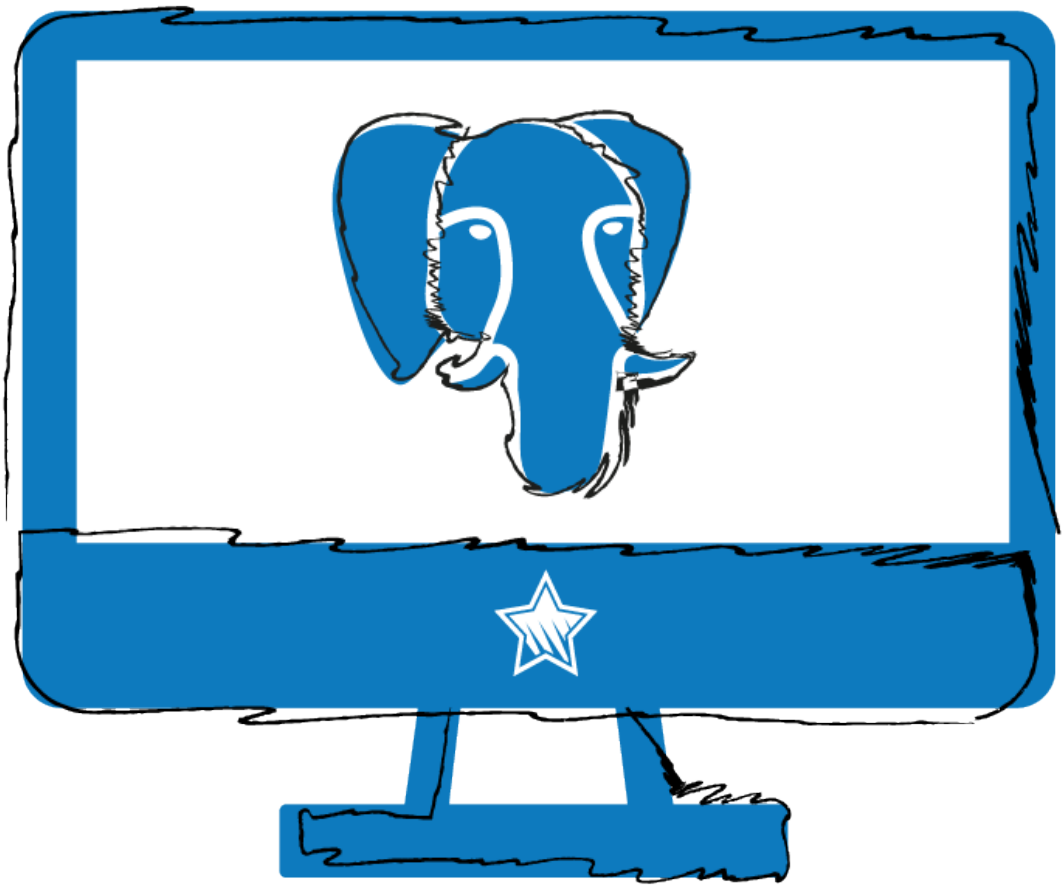
ASTRA LINUX®

Оглавление

Модуль 1. Установка и настройка СУБД PostgreSQL....4
Модуль 2. Создание баз данных....40
Модуль 3. Подключение и аутентификация....66
Модуль 4. Роли в СУБД PostgreSQL....94
Модуль 5. Дискреционное управление доступом в СУБД PostgreSQL в Astra Linux Special Edition....107
Модуль 6. Мандатное управление доступом в СУБД PostgreSQL в Astra Linux Special Edition....135
Модуль 7. Журналирование и аудит доступа к данным....189
Модуль 8. Логическое резервное копирование и восстановление....231
Модуль 9. Физическое резервное копирование и восстановление....271
Модуль 10. Регламентные работы....309
Модуль 11. Автоматизация обслуживания....332

Модуль 1

Установка и настройка СУБД PostgreSQL



ASTRA LINUX®

Модуль 1. Установка и настройка СУБД PostgreSQL

Краткое содержание:

- Терминология.
- Установка СУБД PostgreSQL из дистрибутива Astra Linux Special Edition.
- Управление экземпляром сервера.
- Управление кластером баз данных.
- Использования psql.
- Установка pgAdmin.
- Использование pgAdmin.
- Настройка параметров конфигурации на различных уровнях.

Основные знания и умения:

- понимать терминологию;
- уметь устанавливать СУБД PostgreSQL из дистрибутива Astra Linux Special Edition;
- уметь управлять экземпляром сервера;
- уметь управление кластером БД;
- уметь использовать psql;
- уметь устанавливать pgAdmin из дистрибутива Astra Linux Special Edition;
- уметь использовать pgAdmin;
- уметь настраивать параметры конфигурации на различных уровнях.

Оценка знаний и умений:

- Практическая работа, контрольные вопросы.

Дополнительная литература:

- Отсутствует.

Цели модуля 1. Установка и настройка СУБД PostgreSQL

- Терминология
- Установка СУБД
- Управление экземпляром сервера
- Управление кластером баз данных
- Использования psql
- Установка pgAdmin
- Использование pgAdmin
- Настройка параметров конфигурации на различных уровнях

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Терминология

- *СУБД* — комплекс программно-языковых средств, позволяющих создавать базы данных и управлять данными
- *Экземпляр сервера* — набор процессов, которые обслуживают один кластер баз данных
- *Кластер баз данных* — это коллекция баз данных, хранящихся в файловой системе в определённом размещении

1 *Система управления базами данных (СУБД)* — набор программ, позволяющий организовывать, контролировать и администрировать базы данных.

2 Основные функции СУБД:
- управление данными во внешней памяти;
- управление данными в оперативной памяти;
- журналирование изменений;
- поддержка языков СУБД.

3 Один экземпляр postgres всегда управляет данными ровно одного кластера баз данных.

4 Набор баз данных, управляемый одним экземпляром PostgreSQL, называется *кластером* баз данных.

При установке PostgreSQL с параметрами по умолчанию создается кластер с именем *main*, которым управляет экземпляр сервиса postgresql.

5 Данные PostgreSQL хранит на дисках в файлах операционной системы.

СУБД PostgreSQL в Astra Linux Special Edition

- В качестве защищенной СУБД используется *PostgreSQL*, доработанная в части мандатного управления доступом
- СУБД PostgreSQL в составе Astra Linux Special Edition поддерживает реализацию дискреционного, мандатного и ролевого управления доступом
- При работе с файловой системой (при создании резервных копий, при восстановлении, при экспорте и импорте данных) нужно учитывать, что в файловой системе применяется мандатное управление доступом
- Утилиты и команды для резервного копирования, входящие в состав дистрибутива, доработаны с учетом использования мандатного управления доступом

1 СУБД PostgreSQL в Astra Linux Special Edition содержит ряд параметров безопасности, которые тоже являются доработкой и присутствуют только в СУБД PostgreSQL из дистрибутива Astra Linux Special Edition.

2 Примеры доработки в части мандатного управления доступом:

- команда `COPY` может выгружать данные с их классификационными метками (параметр `WITH MAC`);
- при создании таблицы можно указать, что записи таблицы будут защищены классификационными метками:

```
CREATE TABLE <имя_таблицы> (...--<список_столбцов>)
WITH (MACS = true, ... );
```

- утилита pg_dump выгружает данные с их классификационными метками, но можно выгрузить данные и без указания классификационных меток, с помощью параметра `--disable-macs`.

Установка СУБД в Astra Linux

- Установка СУБД PostgreSQL:
  ```
  sudo apt install postgresql
  ```
- В результате установки будет создан кластер PostgreSQL с именем main
- Версия PostgreSQL:
  ```
  sudo -u postgres psql
  SHOW SERVER_VERSION;
  ```

```
postgres=# SHOW SERVER_VERSION;
          server_version
---------------------------------
 11.17 (Debian 11.17-astra.se1+b1)
```

1 Установить СУБД PostgreSQL в Astra Linux можно с помощью команды
```
sudo apt install postgresql
```
В результате установки будет создан кластер PostgreSQL с именем main. Он будет поддерживаться (обслуживаться) экземпляром сервиса postgresql.

2 В процессе установки PostgreSQL в операционной системе создается пользователь postgres.

В любом кластере по умолчанию создается роль (пользователь) postgres, которая обладает правами суперпользователя в кластере и используется для первичного подключения к кластеру и его настройки.

3 Для подключения к кластеру можно использовать команду
```
sudo -u postgres psql
```
Подключение будет выполнено с помощью терминального клиента psql.

При использовании такого синтаксиса для подключения через ключ `-u` задается пользователь ОС postgres. Его имя сопоставляется с именем роли postgres в кластере БД.

По умолчанию подключение будет выполнено к БД postgres, созданной при создании кластера.

4 Просмотреть версию PostgreSQL можно, например, с помощью выполнения запроса SQL в утилите psql:
```
SHOW SERVER_VERSION;
```

Проверка работы кластера PostgreSQL

- Проверка статуса кластера PostgreSQL:
  `sudo pg_ctlcluster 11 main status`
- Просмотр портов, которые прослушиваются:
  `sudo ss -tlp | grep postgres`
- Имя кластера, который создается при установке серверного пакета по умолчанию – *main*
- По умолчанию PostgreSQL прослушивает TCP-порт *5432*
- Просмотр состояния всех кластеров БД в ОС:
  `sudo pg_lsclusters`

1 После установки можно проверить статус экземпляра PostgreSQL (*main* – имя кластера по умолчанию):

`sudo pg_ctlcluster 11 main status`

```
sa@astra:~$ sudo pg_ctlcluster 11 main status
pg_ctl: сервер работает (PID: 661)
/usr/lib/postgresql/11/bin/postgres "-D" "/var/lib/postgresql/11/main" "-c" "config_fil
e=/etc/postgresql/11/main/postgresql.conf"
```

Вывод команды показывает, что идентификатор основного процесса экземпляра postgresql для кластера main (PID) равен *661*.

2 По умолчанию PostgreSQL прослушивает TCP-порт *5432*.

3 Для просмотра всех TCP-сокетов (IPv4 и IPv6), которые прослушиваются, можно использовать команду:

`sudo ss -tlp | grep postgres`

```
sa@astra:~$ sudo ss -tlp | grep postgres
LISTEN 0      1024         0.0.0.0:postgresql      0.0.0.0:*    users:(("postgres",pid=661,fd=3))
LISTEN 0      1024            [::]:postgresql         [::]:*    users:(("postgres",pid=661,fd=5))
```

4 Для отображения всех TCP-сокетов, которые прослушиваются, без разрешения имен (вместо имени сервиса будет выведен номер порта), можно использовать команду:

`sudo ss -tlpn | grep postgres`

Установка нескольких экземпляров PostgreSQL в одной ОС

- В одной ОС можно установить несколько экземпляров PostgreSQL, обеспечивающих работу нескольких кластеров одновременно, при условии, что они используют различные каталоги для хранения данных и порты
- Каждый кластер уникально идентифицируется по своей версии и названию
- Установка кластеров:

  `pg_createcluster [options] version name [-- initdb options]`
- Например,

  `sudo pg_createcluster 11 sales`

  - создание нового кластера версии PostgreSQL 11 с именем *sales*

1 В операционной системе можно установить несколько экземпляров PostgreSQL, обеспечивающих работу нескольких кластеров одновременно, при условии, что они используют различные каталоги для хранения данных и порты.

2 Каждый кластер уникально идентифицируется по своей версии и названию. Название может быть произвольным.

3 После создания кластера main можно создавать другие кластеры.

4 Для работы экземпляра, обслуживающего кластер sales, будет использоваться первый свободный порт после *5432*. В данном случае, 5433. Конфигурационные файлы будут размещены в каталоге */etc/postgresql/11/sales*.
Файлы данных будут размещены в каталоге */var/lib/postgresql/11/sales*.

5 Рекомендуется выполнить команду `sudo systemctl daemon-reload` для обеспечения правильной регистрации службы в systemd.

6 После запуска кластера sales при просмотре TCP-сокетов (TCPv4), которые прослушиваются, отобразятся 2 сокета, использующие порты 5432 и 5433:
`sudo ss -t4pln | grep postgres`

```
sa@astra:~$ sudo ss -t4pln | grep postgres
LISTEN 0      1024         0.0.0.0:5432       0.0.0.0:*    users:(("postgres",pid=693,fd=3))
LISTEN 0      1024         0.0.0.0:5433       0.0.0.0:*    users:(("postgres",pid=17538,fd=3))
```

7 Такой результат получен, поскольку одновременно работают 2 экземпляра службы postgresql (postgresql@11-main и postgresql@11-sales), которые поддерживают 2 кластера (main и sales).

8 Родительский процесс postgresql порождает еще ряд процессов, обслуживающих кластер.

9 Отобразить все процессы, запущенные родительским процессом postgres, можно с помощью команды:

```
sudo ps -o pid,command  --ppid
<идентификатор родительского процесса>
```

В команде `ps` через ключ `ppid` указан PID, который был выведен командой `ss`.

10 Для отображения процессов также можно воспользоваться командой:

```
ps aux | grep 11/main
```

```
sa@astra:~$ ps aux | grep 11/main
postgres  4481  0.0  2.4 250760 49992 ?        S    авг13   0:04 /usr/lib/postgresql/11/bin/postgres -D /var/lib/postgresql/11
/main -c config_file=/etc/postgresql/11/main/postgresql.conf
postgres  4482  0.0  0.2  68304  4404 ?        Ss   авг13   0:00 postgres: 11/main: logger
postgres  4484  0.0  0.4 251020  9284 ?        Ss   авг13   0:00 postgres: 11/main: checkpointer
postgres  4485  0.0  0.2 250892  5812 ?        Ss   авг13   0:03 postgres: 11/main: background writer
postgres  4486  0.0  0.4 250760  9964 ?        Ss   авг13   0:03 postgres: 11/main: walwriter
postgres  4487  0.0  0.4 251636  8396 ?        Ss   авг13   0:03 postgres: 11/main: autovacuum launcher
postgres  4488  0.0  0.2  68528  4436 ?        Ss   авг13   0:03 postgres: 11/main: stats collector
postgres  4489  0.0  0.4 251636  8112 ?        Ss   авг13   0:00 postgres: 11/main: logical replication launcher
sa       25299  0.0  0.0   6228   876 pts/0    S+   08:06   0:00 grep 11/main
```

При вызове команды используются ключи:

- a — показывать процессы для всех пользователей;
- u — отображать пользователя/владельца процесса;
- x — также показывать процессы, не подключенные к терминалу.

Основные параметры утилиты pg_createcluster

- `-u (--user)` – имя или идентификатор пользователя, который станет владельцем кластера БД, по умолчанию - *postgres*
- `-d (--datadir)` – каталог для размещения файлов кластера БД
- `-- locale` – установка локали сервера, по умолчанию используется локаль среды установки
- `-p (--port)` – порт, который будет прослушивать экземпляр сервера нового кластера БД, по умолчанию следующий свободный после *5432*
- Например,

```
sudo -u postgres pg_createcluster 11 mycluster
--datadir=/my/specific/data/directory
```

1 Пользователь – владелец кластера БД становится *суперпользователем*.

2 Другие параметры:

- `-l path, --logfile=path` – позволяет явно указать путь к файлу журнала сервера postgres. По умолчанию используется значение /var/log/postgresql/postgresql-version-cluster.log.
- `-e encoding, --encoding=encoding` – позволяет указать кодировку БД шаблона.
- `--start` - немедленно запускает сервер для кластера БД после его создания. По умолчанию сервис не запускается.

3 Также можно отдельно указать:

- `--lc-collate=locale` (задает кодировку, правило сортировки для строковых величин);
- `--lc-ctype=locale` (отвечает за классификацию символов);
- `--lc-messages=locale` (задаёт язык сообщений, который будет использоваться для этого кластера);
- `--lc-monetary=locale` (задает формат для денежных типов);
- `--lc-numeric=locale` (задает формат для числовых выражений);
- `--lc-time=locale` (задает формат для времени).

4 Например, можно создать кластер с применением русской локали, но классификация символов будет выполнена согласно локали `C`, что соответствует англоязычной локали с базовым набором символов:
```
pg_createcluster 11 main --start --locale ru_RU.UTF-8
--lc-ctype=C`
```

5 Для ознакомления с полным списком параметров используйте `man pg_createcluster`.

Размещение файлов PostgreSQL в файловой системе

- /usr/lib/postgresql/версия-СУБД/ — разделяемые файлы и консольные утилиты
- /usr/share/postgresql/версия-СУБД/ — загружаемые модули и иные файлы
- /etc/postgresql/версия-кластера/имя-кластера — конфигурационные файлы конкретного кластера
- /var/lib/postgresql/версия-кластера/имя-кластера — каталог данных конкретного кластера

1 На слайде приведены значения, которые используются по умолчанию.

2 Некоторые значения по умолчанию, используемые pg_createcluster, могут быть изменены в /etc/postgresql-common/createcluster.conf

```
root@astra:/etc/postgresql-common# cat createcluster.conf
# Default values for pg_createcluster(8)
# Occurrences of '%v' are replaced by the major version number,
# and '%c' by the cluster name. Use '%%' for a literal '%'.

# Create a "main" cluster when a new postgresql-x.y server package is installed
#create_main_cluster = true

# Default start.conf value, must be one of "auto", "manual", and "disabled".
# See pg_createcluster(8) for more documentation.
#start_conf = 'auto'

# Default data directory.
#data_directory = '/var/lib/postgresql/%v/%c'

# Default directory for transaction logs
# Unset by default, i.e. transaction logs remain in the data directory.
#waldir = '/var/lib/postgresql/wal/%v/%c/pg_wal'

# Options to pass to initdb.
#initdb_options = ''

# The following options are copied into the new cluster's postgresql.conf:

# Enable SSL by default (using the "snakeoil" certificates installed by the
# ssl-cert package, unless configured otherwise here)
ssl = on

# Show cluster name in process title
cluster_name = '%v/%c'
```

Управление экземпляром сервера

- Управление экземпляром выполняется при помощи утилиты *pg_ctlcluster*
- Синтаксис:
  `pg_ctlcluster <версия> <имя_кластера> <операция>`
- Операции управления экземпляром сервера включают:
  - **start** – запустить
  - **stop** – остановить
  - **restart** – перезапустить
  - **status** – отобразить статус
  - **reload** – обновить конфигурацию

1 Управление экземпляром сервера выполняется при помощи утилиты *pg_ctlcluster*.

2 Получить справку по утилите можно с помощью команды `man pg_ctlcluster`.

3 При остановке экземпляра можно указать режим остановки с помощью параметра `-m (--mode)`:
`pg_ctlcluster 11 main stop -m| --mode [smart|fast| immediate]`.

4 Возможные значения параметра `m`:
- `smart` — PostgreSQL ожидает окончания всех сеансов и сохраняет изменения на жесткий диск;
- `fast` — PostgreSQL принудительно завершает все сеансы и сохраняет изменения на жесткий диск;
- `immediate` — PostgreSQL принудительно завершает все сеансы и останавливает экземпляр сервера без сохранения изменений.

5 Для управления кластером можно использовать команду
`sudo systemctl <операция> postgresql@11-<имя_кластера>`.

6 Например,
`sudo systemctl <операция> postgresql@11-sales`.

7 Настоятельно рекомендуется использовать для управления экземпляром утилиту, pg_ctlcluster, поставляемую с ОС:

```
pg_ctlcluster 11 <имя кластера> <операция>
```

8 Например,

```
sudo pg_ctlcluster 11 sales status
```

– вывод в консоль информации о состоянии кластера sales.

9 Утилита pg_ctlcluster входит в состав пакета postgresql-common и представляют собой скрипты, осуществляющие вызов штатных утилит PostgreSQL, но с учетом специфики ОС и возможности одновременного управления кластерами разных версий СУБД.

Удаление кластера БД

- Удаление кластера баз данных:

```
sudo pg_dropcluster [--stop] <версия-кластера>
<имя-кластера>
```

- Указание опции `--stop` приводит к остановке сервера СУБД перед удалением файлов кластера
- Например, для удаления кластера main для СУБД PostgreSQL версии 11 необходимо выполнить команду:

```
sudo pg_dropcluster 11 main
```

1 При выполнении команды удалятся все файлы данных, журнала предзаписи, файлы журнала, конфигурационные файлы.

2 После удаления файлов удаляется директория расположения кластера баз данных, и освобождается порт прослушивания.

Обновление кластера pg_upgradecluster

- Обновления кластера баз данных:
  `/usr/bin/pg_upgradecluster [опции] <старая-версия> <имя-кластера> [новый-каталог-данных]`
- Новая версия задается через параметр `-v newversion`
- По умолчанию используется последняя доступная версия
- Конфигурационные файлы копируются со старого кластера на новый
- При обновлении можно изменить ряд опций

1 Для ознакомления со всеми параметрами используйте команду `man pg_upgradecluster` (в каталоге /usr/bin/).

Утилита psql

- `psql` — терминальный клиент для работы с PostgreSQL
- Синтаксис утилиты psql:
  `psql [параметр...] [имя_бд [имя_пользователя]]`
- Для подключения к базе данных требуется указать:
  - имя базы данных (ключ `-d`)
  - имя сервера (ключ `-h`)
  - номер порта сервера (ключ `-p`)
  - имя пользователя (ключ `-U`)
- Например: `psql -h localhost -d db_name -U user_name`

1 Утилита *psql:*

- это терминальный клиент для работы с PostgreSQL;
- позволяет интерактивно вводить запросы, передавать их в PostgreSQL и видеть результаты. Также запросы могут быть получены из файла или из аргументов командной строки;
- предоставляет ряд метакоманд и различные возможности, подобные тем, что имеются у командных оболочек, для облегчения написания скриптов и автоматизации широкого спектра задач.

2 Синтаксис psql:
`psql [параметр...] [имя_бд [имя_пользователя]]`

3 Для подключения к базе данных требуется: имя базы данных, имя сервера, номер порта сервера и имя пользователя, под которым будет проходить подключение.

Особенности работы psql

- Приглашение psql состоит из имени базы данных, к которой psql в данный момент подключён, а затем строки => (или =#) при условии:
  - => – если подключение к сеансу под рядовым пользователем
  - =# – если подключение к сеансу под суперпользователем
- В командной строке psql пользователь может вводить команды SQL
- Введённые строки отправляются на сервер, когда встречается точка с запятой, завершающая команду
- Если команда была отправлена и выполнена без ошибок, то результат команды выводится на экран

1 Приглашение psql состоит из имени базы данных, к которой утилита psql в данный момент подключёна, а затем строки =>.

```
sa@astra:~$ psql -h localhost -d mac_test_db -U user_ordinary
psql (11.17 (Debian 11.17-astra.se1+b1))
SSL-соединение (протокол: TLSv1.3, шифр: TLS_AES_256_GCM_SHA384, бит: 256, сжатие: выкл
.)
Введите "help", чтобы получить справку.

mac_test_db=>
```

2 Если пользователь текущего сеанса является суперпользователем базы данных, то выводится приглашение =# .

3 В командной строке psql пользователь может вводить команды SQL.

Введённые строки отправляются на сервер, когда встречается точка с запятой(;), завершающая команду.

Конец строки или нажатие ввода не завершает команду. Это позволяет разбивать команды на несколько строк.

Если команда была отправлена и выполнена без ошибок, то результат команды выводится на экран.

Пример вставки данных в таблицу:

```
mac_test_db=> INSERT INTO mactable (id, secret_level_comment) VALUES (11,'не секретно');
INSERT 0 1
```

Основные команды для работы с psql

- `sudo -u user psql db` — подключение пользователем user к БД db
- `psql -U user -d db -h hostname` — подключение пользователем user к БД db
- `sudo -u postgres psql -l` — вывод списка существующих БД без входа в консоль psql
- `sudo -u postgres psql -c '\du'` — список ролей без входа в консоль psql
- `psql --help` — справочная информация
- `man psql` — справочная информация

1 Пароль для суперпользователя по умолчанию не задан.

2 При подключении через команду `sudo -u postgres psql` через ключ -u указывается пользователь postgres операционной системы.
Он сопоставляется с одноименной ролью (пользователем) в кластере PostgreSQL.

Таким образом, при использовании указанного синтаксиса, подключение через psql будет выполнено от имени пользователя postgres кластера main.

3 Примеры команд:

- вывод списка существующих БД в консоль

  `sudo -u postgres psql -l`

  Подключение выполняется от имени пользователя postgres, интерактивный вход в psql не выполняется.

```
sa@astra:/$ sudo -u postgres psql -l
                                 Список баз данных
    Имя    | Владелец | Кодировка | LC_COLLATE  |  LC_CTYPE   |     Права доступа
-----------+----------+-----------+-------------+-------------+-----------------------
 postgres  | postgres | UTF8      | ru_RU.UTF-8 | ru_RU.UTF-8 |
 template0 | postgres | UTF8      | ru_RU.UTF-8 | ru_RU.UTF-8 | =c/postgres          +
           |          |           |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8      | ru_RU.UTF-8 | ru_RU.UTF-8 | =c/postgres          +
           |          |           |             |             | postgres=CTc/postgres
(3 строки)
```

- вывод списка пользователей (ролей) без входа в консоль:

  `sudo -u postgres psql -c '\du'`

4 В psql справку можно получить, например, следующим образом:

- `\?` — список команд psql;
- `\? variables` — переменные psql;
- `\h` — список команд SQL;
- `\h команда` — синтаксис команды SQL.

Основные команды при работе с БД в psql

- `\conninfo` — вывести информацию о подключении
- `\c[onnect]` — выполнить новое подключение
- `\q[uit]` — выход
- `\du` — вывести список пользователей (ролей)
- `\l` — вывести список баз данных
- `\dt` — вывести список таблиц
- При вводе команд psql (не запросов на SQL) указывать точку с запятой в конце команды не нужно

1 Команда `\c` или `\connect [ -reuse-previous=on|off ] [ имя_бд [ имя_пользователя ] [ компьютер ] [ порт ] | строка_подключения ]`

устанавливает новое подключение к серверу PostgreSQL.

Параметры подключения можно указывать как позиционно (один или несколько по списку: база данных, пользователь, компьютер и порт), так и передавая аргумент `строка_подключения`.

Если аргументы отсутствуют, новое подключение устанавливается с теми же параметрами, что и предыдущее.

2 По умолчанию параметры используются повторно при позиционной записи (не в случае использования строки подключения). Это поведение может переопределяться первым аргументом, `-reuse-previous=on` или `-reuse-previous=off`. По умолчанию `reuse-previous=on`.

3 В случае повторного использования параметров, любой параметр, явно не заданный в виде позиционного или в виде строки подключения, заимствуется из параметров текущего подключения.

Например (команды указываются после входа в утилиту psql),

```
\c sales iivanov localhost 5432
```

– подключение к БД salesdb от имени роли iivanov на хосте localhost по порту 5432;

- `\connect sales`

  – подключение к БД sales, остальные параметры остаются прежними;
- `\connect sales iivanov`

  – подключение к БД sales от имени iivanov (после подключения, например, под пользователем postgres).

Установка pgAdmin

- *pgadmin3* – графическая утилита для работы с PostgreSQL
- Команда для установки pgadmin3:
  `sudo apt install pgadmin3`
- После инсталляции утилита pgadmin3 доступна из меню:
  Пуск → Разработка → pgAdmin III
- Утилита, входящая в дистрибутив, доработана для отображения классификационных меток и других параметров, связанных с мандатным управлением доступа

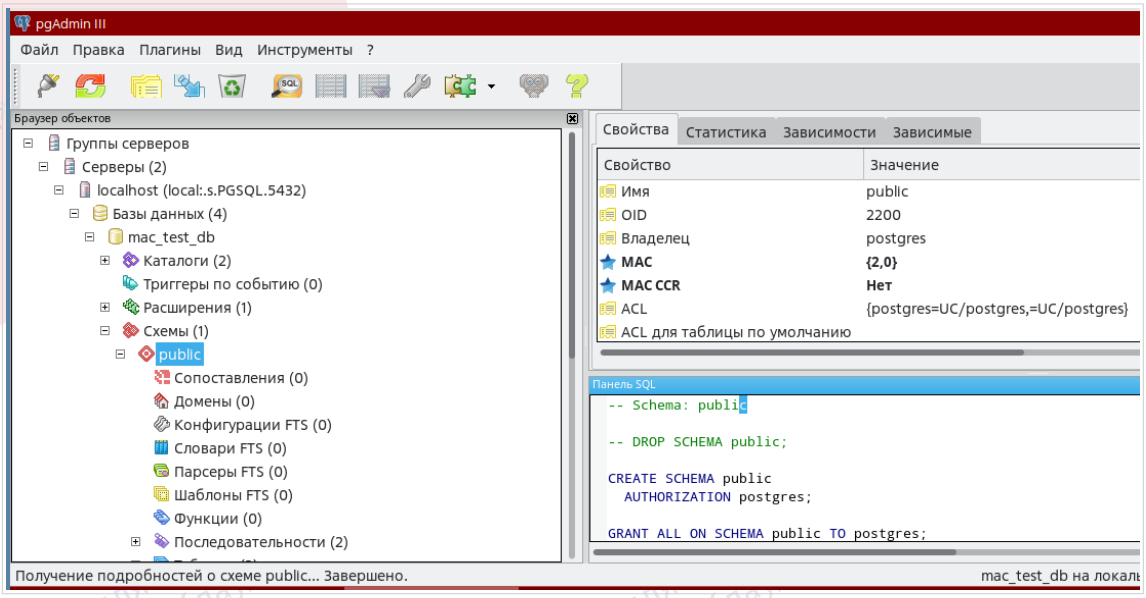
1 Графическая утилита pgadmin3 разработана сторонними разработчиками.

2 В ОС Linux Astra SE `pgadmin3` по умолчанию не установлена.

3 Для установки pgAdmin в ОС Linux Astra SE требуется выполнить команду:
`sudo apt install pgadmin3`

4 После инсталляции утилита доступна из меню **Пуск –> Разработка –> pgAdmin III**.

5 Интерфейс pgAdmin III:



6 Сайт разработчика утилиты pgadmin:
https://www.pgadmin.org

7 Утилита pgadmin3 для работы использует сетевое подключение.

8 При настройках по умолчанию у пользователя postgres не установлен пароль. Без изменения настроек подключиться через утилиту pgadmin3 не получится.

9 Для подключения через утилиту pgadmin3 следует создать роль суперпользователя, имя которого не является общеизвестным.

10 Например,

```
CREATE ROLE iivanov WITH LOGIN SUPERUSER PASSWORD
'P@ssw0rd';
```

(создание ролей будет подробно рассмотрено в модуле 4).

Для того, чтобы при подключении к PostgreSQL для пользователя была определена классификационная метка, нужно создать одноименного пользователя в ОС, назначить ему пароль и мандатную метку (подробнее в модулях 2, 6).

11 Далее можно подключиться к экземпляру PostgreSQL, используя параметры: хост: localhost, порт: 5432, обслуживание DB: postgres, имя пользователя: iivanov, пароль: P@ssw0rd.

Параметры конфигурации

- Параметры настраиваются на разных уровнях:
  - На уровне сервера
  - На уровне базы данных
  - На уровне отдельной роли
  - На уровне сеанса
  - На уровне транзакции
- Просмотр параметров:
  - SHOW
  - pg_settings

1 Поведение сервера PostgreSQL можно изменять с помощью параметров. Настройку можно выполнять на разных уровнях.

2 Параметры представляют собой пару ключ=значение (ключ – имя параметра).

3 Для просмотра параметров можно воспользоваться:
   - командой `psql SHOW`;
   - представлением `pg_settings`.

4 Большинство параметров можно настроить в своей сессии, но есть ряд параметров, которые доступны только для суперпользователя или требуют перезагрузки сервера.

5 При использования представления pg_settings столбец context определяет условия, необходимые для применения параметра:
   - internal – параметр можно изменить только с помощью изменения кода сервера. Пользователи не могут изменять параметры этого значения;
   - postmaster – параметр можно изменить только во время запуска сервера PostgreSQL. В файле postgresql.conf или в командной строке при запуске сервера;

- sighup – параметр можно изменить в файле postgresql.conf или с помощью команды SET в текущей сессии. Но изменения также вступают в силу после перезагрузки конфигураций командой `pg_ctlcluster reload` или отправкой сигнала SIGHUP процессу сервера;
- backend – параметр можно изменить во время начала пользовательской сессии. Если параметр изменен во время сессии, то это влияет только на текущую сессию;
- superuser – только суперпользователь может изменить этот параметр в файле postgresql.conf или командой SQL SET в текущей сессии;
- user – параметр можно изменить в SQL-команде SET в текущей сессии.

`SELECT DISTINCT context FROM pg_settings;`

```
postgres=# SELECT DISTINCT context FROM pg_settings;
      context
-------------------
 postmaster
 superuser-backend
 user
 internal
 backend
 sighup
 superuser
(7 строк)
```

6 Используя запрос

`SELECT name, setting, source FROM pg_settings;`,

можно определить источник текущего значения параметра (приведены не все результаты выборки):

```
                name                |                  setting                  |        source
------------------------------------+-------------------------------------------+----------------------
 ac_audit_destination               | all                                       | configuration file
 ac_audit_file                      | /var/lib/postgresql/11/reserve/pg_audit.conf | override
 ac_audit_log_only_failures         | off                                       | configuration file
 ac_audit_mode                      | internal                                  | configuration file
 ac_auto_adjust_macs                | off                                       | configuration file
 ac_capable_chmac                   | да                                        | default
 ac_capable_ignmaccat               | да                                        | default
```

Параметры конфигурации на уровне сервера

- Файлы конфигурации:
  - postgresql.conf
  - postgresql.auto.conf
- Параметры в postgresql.conf задаются в текстовом редакторе
- Параметры в postgresql.auto.conf задаются SQL-командой ALTER SYSTEM
- Для применения параметров
  - pg_ctlcluster 11 <имя_кластера> reload
  - pg_ctlcluster 11 <имя_кластера> restart
  - pg_reload_conf()

1 Основным способом настройки параметров на уровне сервера является использование файлов конфигурации:
- postgresql.conf (основной файл);
- postgresql.auto.conf (хранится в каталоге данных PostgreSQL).

2 Файл postgresql.conf может быть изменен при помощи текстового редактора.

3 Расположение файла postgresql.conf можно определить при помощи параметра config_file:

```
SHOW config_file;
```

```
postgres=# SHOW config_file;
               config_file
-----------------------------------------
 /etc/postgresql/11/reserve/postgresql.conf
(1 строка)
```

4 Для просмотра содержимого файла postgresql.conf также можно воспользоваться функцией pg_read_file.

5 Чтобы просмотреть все не закомментированные строки файла postgresql.conf или проверить правильность внесенных изменений, можно воспользоваться представлением pg_file_settings.

6 Если в файле postgresql.conf один и тот же параметр встречается несколько раз, всегда будет использоваться последнее значение.

7 Для применения параметров:

- следует перечитать файл конфигурации терминальной командой pg_ctlcluster 11 <имя_кластера> reload
- или использовать SQL-функцию pg_reload_conf().

8 Для использования функции нужно в клиентском приложении написать запрос, который эту функцию запускает:

```
SELECT pg_reload_conf();
```

```
postgres=# SELECT pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 строка)
```

9 *Примечание:* в некоторых случаях потребуется перезагрузить сервер.

Требуется ли повторное считывание параметра или перезапуск службы, чтобы изменения вступили в силу, указывается в описании параметра.

10 Перезапуск сервера:

```
sudo pg_ctlcluster 11 <имя_кластера> restart
```

11 Просмотреть текущее состояние параметра можно с помощью запроса:

```
SHOW имя_параметра;
```

Например,

```
SHOW ac_enable_grant_options;
```

Файл postgresql.auto.conf

- Расположение по умолчанию:
  /var/lib/postgresql/11/<имя кластера>/postgresql.auto.conf
- Имеет тот же формат, что и postgresql.conf
- Параметры задаются командой ALTER SYSTEM
- Считывается после postgresql.conf
- Синтаксис ALTER SYSTEM:

```
ALTER SYSTEM SET configuration_parameter { TO | = }
{ value | 'value' |DEFAULT };
```

- `ALTER SYSTEM RESET configuration_parameter;`
  – сброс значения параметра в значение по умолчанию

1 Для того, чтобы выполнить конфигурацию параметров, указанных в файле postgresql.conf, не имея доступа к консоли операционной системы, можно использовать файл /var/lib/postgresql/11/<имя кластера>/postgresql.auto.conf

2 *Примечание:*
не рекомендуется изменять параметры в файле postgresql.auto.conf в текстовом редакторе.

3 Значения параметров, указанные в postgresql.auto.conf, являются приоритетными.

4 Сброс всех параметров в значение по умолчанию выполняется командой:

```
ALTER SYSTEM RESET ALL;
```

5 Примеры использования:

```
ALTER SYSTEM SET ac_enable_truncate = false;
ALTER SYSTEM SET ac_enable_grant_options = true;
```

6 После выполнения команды изменения параметра нужно повторно загрузить конфигурацию.

Сделать это из psql можно с помощью запроса

```
SELECT pg_reload_conf();
```

После выполнения этого запроса изменения вступят в силу.

Параметры конфигурации на уровне БД

- Изменение параметров через ALTER DATABASE:
  ```
  ALTER DATABASE sales SET timezone TO 'Europe/Moscow';
  ```
- Примеры параметров:
  - DATESTYLE – определяет вывод даты и времени
  - TIMEZONE – определяет часовой пояс для вывода времени
  - LC_COLLATE – управляет упорядочиванием символов
  - LC_CTYPE – контролирует классификацию символов для БД
  - allow_connections – определяет, будет ли разрешено подключение к этой БД

1 В PostgreSQL многие параметры конфигурации могут быть установлены на уровне базы данных.

2 Подробнее: https://www.postgresql.org/docs/11/config-setting.html

Параметры конфигурации на уровне роли пользователя

- `ALTER ROLE { указание_роли | ALL } [ IN DATABASE имя_бд ] SET параметр_конфигурации { TO | = } { значение | DEFAULT }`
- `ALTER ROLE { указание_роли | ALL } [ IN DATABASE имя_бд ] SET параметр_конфигурации FROM CURRENT`
- `ALTER ROLE { указание_роли | ALL } [ IN DATABASE имя_бд ] RESET параметр_конфигурации`
- `ALTER ROLE { указание_роли | ALL } [ IN DATABASE имя_бд ] RESET ALL`
- Пример:
  `ALTER ROLE ppetrov SET timezone TO 'Europe/Moscow';`

1 Тот же набор параметров можно изменять для определенной роли пользователя и применительно к отдельной БД.

2 Параметр конфигурации принимает заданное значение по умолчанию в сеансах роли.

3 В новых сеансах роль будет получать системное значение параметра:

- если значение параметра задано как DEFAULT
- после применения операции RESET, так как переопределение этого параметра для роли удаляется.

4 Для очистки значений всех параметров, связанных с ролью, применяется RESET ALL. SET FROM CURRENT сохраняет текущее значение параметра в активном сеансе в качестве значения для данной роли.

5 Если указано IN DATABASE, параметр конфигурации настраивается или удаляется только для данной роли и указанной базы данных.

6 Подробнее роли будут рассматриваться в модуле 4.

7 Подробнее о назначении параметров конфигурации для роли:
https://www.postgresql.org/docs/current/sql-alterrole.html

Параметры конфигурации на уровне сеанса

- Контролируют поведение текущего сеанса
- Примеры параметров:
  - `DateStyle` – определяет формат выводимых дат и их разбора при вводе
  - `TimeZone` – устанавливает часовой пояс для отображения времени
  - `Idle_in_transaction_session_timeout` – определяет максимальное время простоя в транзакции, после которого сеанс будет завершен
- Установка параметров выполняется через команду SET или `set_config`
- Например,

```
SET TimeZone = 'Europe/Moscow';
```

1 В PostgreSQL некоторые параметры могут быть установлены на уровне сеанса пользователя. Эти параметры контролируют поведение текущего сеанса и становятся активными только для этого конкретного сеанса.

2 Эти параметры используются только для текущего сеанса.

3 Если пользователь завершит сеанс, а затем установит новое подключение, параметры будут сброшены на предопределенные значения.

4 Функция set_config устанавливает новое значение параметра и возвращает его.

5 Синтаксис:

`set_config(setting_name, new_value, is_local)`,

где

`setting_name` — имя параметра,

`new_value` — новое значение параметра.

6 Если параметр с именем setting_name не найден, то функция current_setting выдаёт ошибку, если только дополнительно не передан параметр missing_ok со значением true.

7 Если параметр `is_local` равен `true`, новое значение будет действовать только в рамках текущей транзакции. Чтобы это значение действовало на протяжении текущего сеанса, ему нужно присвоить `false`.

8 Таким образом, для установки значения параметра для сеанса, следует в качестве значения третьего параметра указывать `false`.

9 Пример использования функции set_config для настройки параметра сеанса:

```
SELECT set_config('log_statement_stats', 'off', false);
```

```
inventory=# SELECT set_config('log_statement_stats', 'off', false);
 set_config
------------
 off
(1 строка)
```

Параметры конфигурации на уровне транзакции

- Устанавливаются с помощью функции `set_config`
- В качестве значения третьего параметра `is_local` следует использовать `true`
- Пример:

```
SELECT current_setting('TimeZone');
BEGIN;
   SELECT set_config('TimeZone', 'Asia/Vladivostok',
   true);
   SELECT current_setting('TimeZone');
   -- здесь SQL запросы
COMMIT;
```

1 Для просмотра текущего значения параметра можно использовать функцию `current_setting(setting_name [, missing_ok ])`.

2 Если параметра с именем setting_name нет, функция current_setting выдаёт ошибку, если только дополнительно не передан параметр missing_ok со значением true.

3 Например,

```
SELECT current_setting('datestyle');
```

4 Подробнее: https://www.postgresql.org/docs/11/functions-admin.html

Вопросы для проверки

1 Набор процессов, которые обслуживают один кластер БД — это ...

2 Коллекция баз данных, хранящихся в файловой системе в определённом размещении — это ...

3 Имя кластера, который создается при установке серверного пакета по умолчанию — это ...

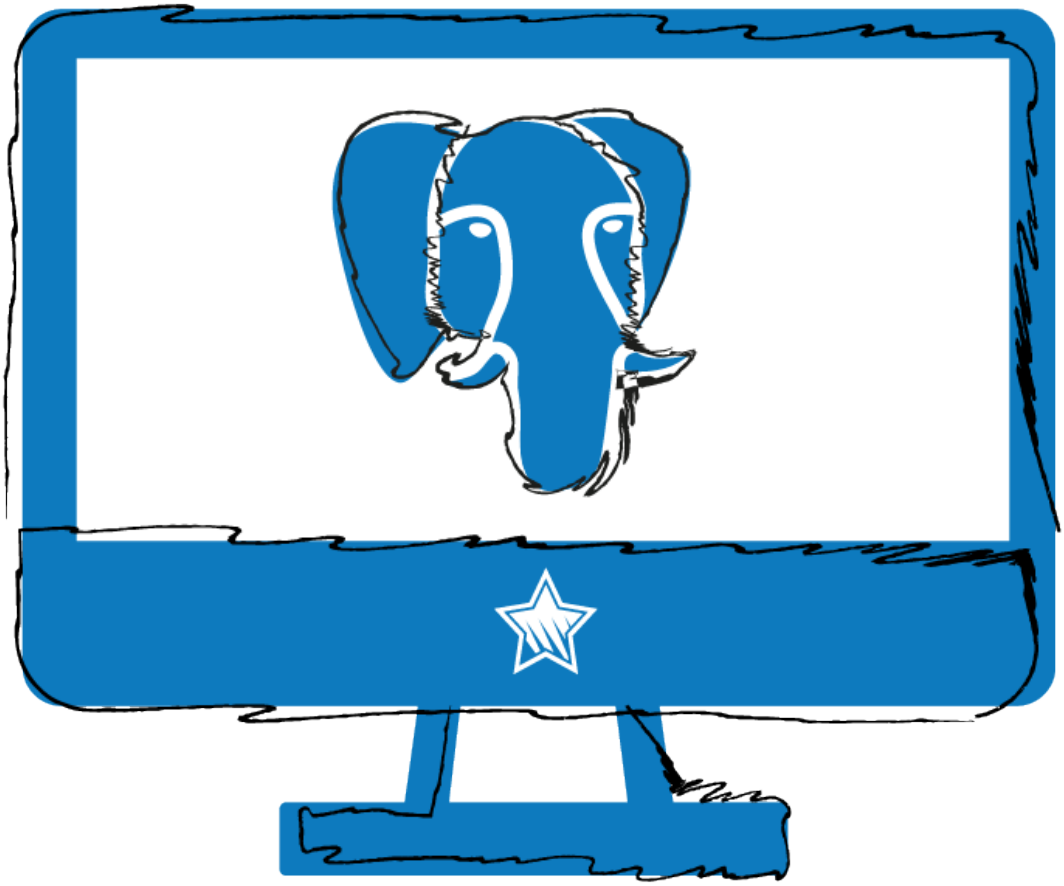
Практическая работа

Оценка времени выполнения: 40 мин.

- Выполните задания модуля 1 из Сборника практических заданий.

Модуль 2

Создание баз данных



ASTRA LINUX®

Модуль 2.
Создание баз данных

Краткое содержание:

- Шаблоны баз данных.
- Создание БД.
- Управление БД.
- Схемы в БД.
- Работа со схемами.
- Каталог PGDATA.
- Табличные пространства (ТП).
- Управление ТП.
- Перемещение объектов между ТП.

Основные знания и умения:

- знать шаблоны БД и понимать как их использовать;
- создавать БД;
- управлять БД (переименовывать, изменять параметры, определять размер);
- понимать, как использовать схемы БД;
- создавать схемы;
- перемещать объекты между схемами;
- создавать и удалять табличные пространства.

Оценка знаний и умений:

- Практическая работа, контрольные вопросы.

Дополнительная литература:

- Отсутствует.

Цели модуля 2. Создание баз данных

- Шаблоны баз данных
- Создание БД
- Управление БД
- Схемы в БД
- Работа со схемами
- Табличные пространства (ТП)
- Управление ТП
- Перемещение объектов между ТП

Astra Linux для АНО УЦ ДПО « Айти Клауд»

БД кластера PostgreSQL после создания

- Экземпляр PostgreSQL управляет несколькими БД — кластером БД
- При инициализации кластера создаются три одинаковые базы данных (postgres, template0, template1)
- База данных postgres используется при подключении по умолчанию пользователем postgres
- Шаблон *template1* используется по умолчанию для создания новых БД
- Шаблон *template0* не должен изменяться
- Остальные БД, создаваемые пользователем, клонируются из какой-либо существующей

1 Шаблон template0 нужен для восстановления БД из резервной копии, выполненной с помощью pg_dump, и при создании новой БД с кодировкой, отличной от указанной при инициализации кластера.

Просмотр списка БД

- команда `psql \l`

```
postgres=# \l
                                  Список баз данных
    Имя    | Владелец | Кодировка | LC_COLLATE  |  LC_CTYPE   |     Права доступа
-----------+----------+-----------+-------------+-------------+-----------------------
 postgres  | postgres | UTF8      | ru_RU.UTF-8 | ru_RU.UTF-8 |
 template0 | postgres | UTF8      | ru_RU.UTF-8 | ru_RU.UTF-8 | =c/postgres          +
           |          |           |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8      | ru_RU.UTF-8 | ru_RU.UTF-8 | =c/postgres          +
           |          |           |             |             | postgres=CTc/postgres
(4 строки)
```

- с помощью запроса к системному каталогу

```
SELECT datname, datistemplate, datallowconn FROM pg_database;
```

1 В таблице системного каталога pg_database хранится следующая информация:

- в столбце `datname` – имя БД;
- в столбце `datistemplate` – является ли база данных шаблоном;
- в столбце `datallowconn` – разрешены ли соединения с базой данных.

```
postgres=# SELECT datname, datistemplate, datallowconn FROM pg_database;
  datname  | datistemplate | datallowconn
-----------+---------------+--------------
 postgres  | f             | t
 template1 | t             | t
 template0 | t             | f
 sales     | f             | t
(4 строки)
```

2 С полным списком столбцов каталога pg_database можно ознакомиться в документации Postgres:

https://www.postgresql.org/docs/11/catalog-pg-database.html

Создание БД из шаблона

- По умолчанию новая БД создается на основе шаблона template1
- Можно изменить шаблон template1, создав в нем новые объекты, установив расширения
- После создания БД в ней будут присутствовать объекты, созданные в шаблоне template1, и установленные для шаблона template1 расширения
- Пример установки расширения:

```
CREATE EXTENSION pgagent;
```

1 В PostgreSQL существует большое количество расширений, которые созданы для простого решения сложных и нестандартных задач.

С их помощью можно увеличить возможности своей системы управления базами данных.

2 Посмотреть расширения, доступные для установки, версию по умолчанию, версию установленного приложения и небольшое описание выполняемых функций можно с помощью запроса:

```
SELECT * FROM pg_available_extensions;
```

3 Установить расширение можно используя синтаксис:

```
CREATE EXTENSION [ IF NOT EXISTS ] имя_расширения
[ WITH ] [ SCHEMA имя_схемы ]
[ VERSION версия ]
[ FROM старая_версия ]
[ CASCADE ]
```

4 Подробно о синтаксисе создания расширений: https://www.postgresql.org/docs/11/sql-createextension.html

Управление БД

- Переименование:
  ```
  ALTER DATABASE db RENAME TO sales;
  ```
- Изменение параметров БД:
  ```
  ALTER DATABASE sales CONNECTION LIMIT 50;
  ```
- Определение размера БД:
  ```
  SELECT pg_database_size('sales');
  ```

1 Для переименования БД к ней не должно быть подключений.

2 Функция `pg_size_pretty` преобразует размер в байтах, представленный 64-битным целым числом, в понятный человеку формат с единицами измерения.

3 Пример использования функции pg_size_pretty:

```
SELECT pg_size_pretty(pg_database_size('sales'));
```

```
sales=# SELECT pg_size_pretty(pg_database_size('sales'));
 pg_size_pretty
----------------
 10 MB
(1 row)
```

4 Размер пустой БД составляет около 10 МБ.

Схемы в PostgreSQL

- *Схемы* – пространства имен для объектов БД, объекты безопасности
- Схемы позволяют:
  - разделить объекты на логические группы для управления
  - предотвратить конфликты имен при работе нескольких пользователей или при установке приложений
- Схемы содержат таблицы и другие виды именованных объектов
- Синтаксис обращения к объекту: `схема.имя_объекта`
- Если схема не указана, осуществляется поиск объекта в одной из схем, указанных в пути поиска *search_path*

1 Схемы создаются внутри БД и представляют собой пространства имен для объектов БД. Также они являются объектами безопасности.

2 Каждый объект, существующий в базе данных, принадлежит какой-либо схеме.

3 Схемы позволяют разделить объекты на логические группы для управления ими, предотвратить конфликты имен при работе нескольких пользователей или при установке приложений.

4 Схемы содержат таблицы и другие виды именованных объектов, включая типы данных, функции и операторы.

5 Чтобы создать объекты в схеме или обратиться к ним, нужно указать полное имя, состоящее из имени схемы и объекта, разделённых точкой:

- Синтаксис:

  `схема.имя_объекта`

Если схема в имени объекта не указана, то PostgreSQL пытается найти имя в одной из схем, перечисленных в пути поиска, который определяется конфигурационным параметром search_path.

Конфигурационный параметр search_path

- Определяет путь для поиска схемы, если она не указана явно
- Просмотр:
  `SHOW search_path;`
- Значение по умолчанию:"$user", public
  - "$user" – ссылка на схему с именем текущего пользователя
  - public – ссылка на схему public
- Схема public создается по умолчанию и существует во всех БД PostgreSQL
- Изменение значения параметра search_path
  `SET search_path TO <новый список схем>;`

1 Просмотреть текущее значение параметра search_path можно с помощью запроса:

`SHOW search_path;`

2 В конфигурации по умолчанию параметр search_path имеет значение `"$user", public.`

Первый элемент ссылается на схему с именем текущего пользователя. Если такой схемы не существует, ссылка на неё игнорируется. Второй элемент ссылается на схему public.

Схема public создается по умолчанию и существует во всех БД PostgreSQL.

3 Первая существующая схема в пути поиска также считается схемой по умолчанию для новых объектов. Если для пользователя не создана схема, совпадающая с его именем, обращение происходит к схеме public.

4 При указании неполного имени объекта в запросе система просматривает путь поиска, пока не найдёт соответствующий объект.

5 Можно изменить значение параметра search_path с помощью следующего запроса:

`SET search_path TO <новый список схем>;`

Например,

`SET search_path TO iivanov, public;`

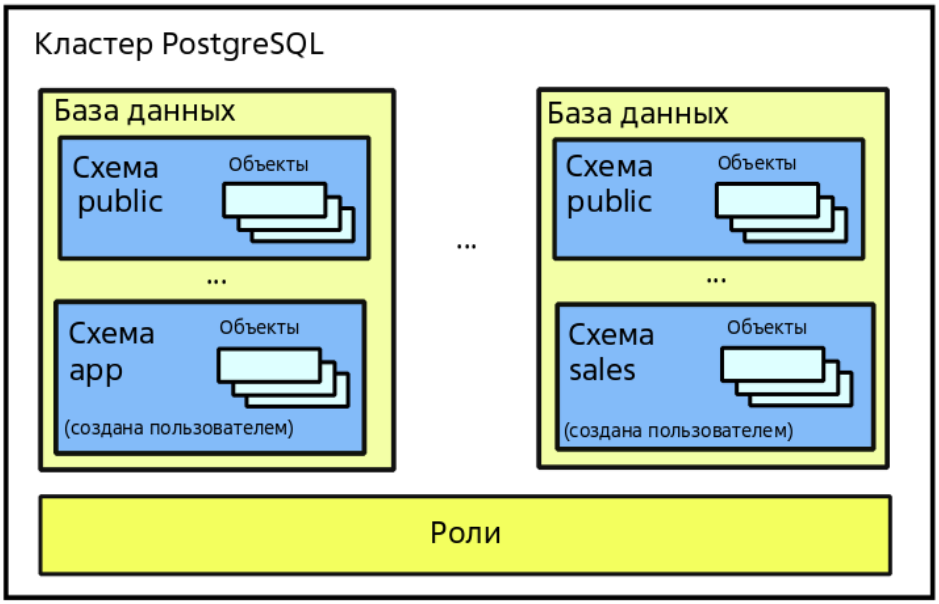
(схема iivanov должна быть создана **заранее**)

6 Вывести имена схем в пути поиска, включая схемы, добавляемые в него неявно, можно с помощью функции `current_schemas(boolean)`:

```
SELECT current_schemas(true);
```

```
postgres=# SELECT current_schemas(true);
   current_schemas
---------------------
 {pg_catalog,public}
(1 строка)
```

Схемы и объекты в БД PostgreSQL



1 На слайде представлено соотношение объектов кластера:

- экземпляр PostgreSQL управляет несколькими БД;
- БД может содержать несколько схем;
- в каждой БД по умолчанию создается схема public;
- объекты (таблицы, представления, функции и т.д.) привязываются к схемам;
- роли – объекты уровня кластера.

Работа со схемами

- Просмотр списка схем:

  `\dn` — команда psql
- Создание схемы:

  ```
  CREATE SCHEMA person;
  ```
- Создание таблицы в схеме:

  ```
  CREATE TABLE person.address(city character varying(30) NOT NULL, ...);
  ```
- Привязка объекта к другой схеме:

  ```
  ALTER TABLE person.address SET SCHEMA employee;
  ```

1 При перемещении объекта между схемами изменения касаются только логической организации данных. Изменение происходит только в системном каталоге. Сами данные физически остаются на месте.

2 Полный синтаксис CREATE SCHEMA:
https://www.postgresql.org/docs/11/sql-createschema.html

Специальные схемы

- *Схема public* – используется по умолчанию для хранения объектов (если не создавалась схема, имя которой совпадает с именем пользователя)
- *Схема pg_catalog* – хранит объекты системного каталога
- *Схема information_schema* – альтернативное представление системного каталога (определенное в стандарте SQL)
- *Схема pg_temp_N* – автоматически создается для временных таблиц

1 Схема pg_catalog содержит объекты системного каталога.

2 Системные каталоги — это место, где система управления реляционной базой данных хранит метаданные схемы, в частности, информацию о таблицах и столбцах, а также служебные сведения.

3 Альтернативное представление системного каталога (определенное в стандарте SQL) дает схема information_schema.

4 В схеме pg_catalog находятся объекты системного каталога (таблицы *pg_ **).

5 Временные таблицы в PostgreSQL предназначены для хранения данных, которые должны быть доступны только текущему сеансу (и только на время его жизни, или на время текущей транзакции).

6 Временные таблицы являются нежурналируемыми. В случае сбоя таблица не может быть восстановлена с помощью журнала. Вместо этого она будет очищена.

7 Страницы временных таблиц не попадают в общий буферный кеш. Работа с ними ведется во внутренней памяти обслуживающего процесса. Поэтому работа с временной таблицей происходит несколько более эффективно, чем с обычной.

8 Временные таблицы организованы с помощью схем.

Для сеанса создается временная схема с именем *pg_temp_N* (*pg_temp_1, pg_temp_2* и т. п.).

9 Обращаться к ней нужно по имени pg_temp (без номера) — для каждого сеанса это имя ссылается на конкретную временную схему.

10 Если pg_temp нет в пути, она неявно подразумевается самой первой.

11 По окончании сеанса все объекты временной схемы удаляются, а сама схема остается и повторно используется для других сеансов.

Каталог PGDATA

- *PGDATA* – это переменная, которая используется PostgreSQL для определения местоположения каталога данных кластера БД
- При установке PostgreSQL через менеджер пакетов, PGDATA обычно находится в */var/lib/postgresql/[версия]/main*
- На одном компьютере может существовать несколько кластеров, управляемых разными экземплярами сервера
- Для каждого кластера БД будет свое значение PGDATA
- Просмотреть значение PGDATA для кластера, к которому установлено подключение, можно, используя команду

```
SHOW data_directory;
```

1 В каталоге PGDATA хранятся следующие данные:
   - базы данных;
   - статистика;
   - информация для восстановления;
   - конфигурационные файлы.

2 Каталог PGDATA содержит несколько подкаталогов и управляющих файлов.

Табличные пространства

- *Табличные пространства* (ТП) в PostgreSQL служат для организации физического хранения данных
- Позволяют администраторам баз данных определять, где в файловой системе могут храниться файлы, представляющие объекты базы данных
- Используются для:
  - управления расположением данных на диске
  - для оптимизации производительности
- ТП указывает на каталог, в котором располагаются данные

1 Примеры использования ТП:

- если в разделе или томе, на котором был инициализирован кластер, не хватает места, и он не может быть расширен, ТП может быть создано в другом разделе и использоваться до тех пор, пока система не будет перенастроена;
- табличное пространство для архивных данных на дисках с низкой скоростью работы;
- индекс, который используется очень интенсивно, может быть помещен на очень быстрый и высокодоступный диск, такой как дорогое твердотельное устройство;
- таблица, хранящая архивные данные, которые редко используются или не критичны к производительности, могла бы храниться в менее дорогом и медленном хранилище.

Служебные ТП pg_default и pg_global

- ТП pg_default и pg_global создаются при инициализации кластера
- В табличном пространстве pg_global хранятся те объекты системного каталога, которые являются общими для кластера
- Для каждой БД указывается табличное пространство, которое используется как табличное пространство по умолчанию
- В ТП по умолчанию создаются все объекты, если явно не указано, что для хранения объекта нужно использовать другое ТП
- Табличное пространство pg_default указано как табличное пространство по умолчанию для шаблонов template1 и template0

1 Стандартные ТП pg_global и pg_default всегда находятся в PGDATA/global/ и PGDATA/base/ соответственно.

2 При создании пользовательского ТП указывается произвольный каталог. PostgreSQL создает на него символьную ссылку в каталоге PGDATA/pg_tblspc/.

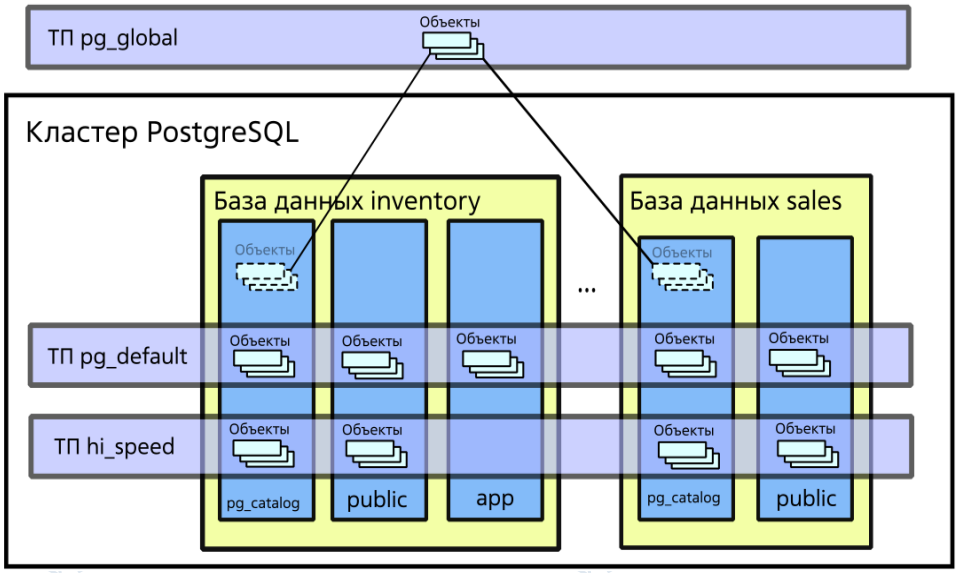
3 Внутри каталога PGDATA/base/ данные дополнительно разложены по подкаталогам баз данных (для PGDATA/global/ это не требуется, так как данные в нем относятся к кластеру в целом).

4 Внутри каталога пользовательского ТП появляется еще один уровень вложенности: версия сервера PostgreSQL. Это сделано для удобства обновления сервера на другую версию. Объекты хранятся в файлах внутри этих каталогов — каждый объект в отдельных файлах.

5 Получить информацию о табличных пространствах можно с помощью таблицы системного каталога pg_tablespace:

```
SELECT * FROM pg_tablespace;
```

Использование табличных пространств



1 Одно и то же ТП может использоваться разными базами данных.

2 Одна база данных может хранить данные в нескольких ТП.

3 У каждой БД есть настройка — ТП, которое она использует по умолчанию. В этом ТП создаются все объекты, если ТП при создании объекта не указано явным образом.

Например, при создании таблицы ТП можно указать явным образом:

```
CREATE TABLE foo(i int) TABLESPACE space1;
```

4 В ТП, указанном как ТП по умолчанию, хранятся и объекты системного каталога.

5 Изначально в качестве ТП по умолчанию указано ТП **pg_default**, но можно указать другое ТП.

6 Если БД создается без указания ТП для ее размещения, то она использует то же ТП, что и БД-шаблон, из которой она скопирована.

Создание пользовательских ТП

- Для создания ТП используется команда CREATE TABLESPACE:

```
CREATE TABLESPACE <имя_ТП> LOCATION <путь_к_каталогу>;
```

- Например:

```
CREATE TABLESPACE fastspace LOCATION
'/ssd1/postgresql/data';
```

- Получить список ТП можно:
  - через запрос к системному каталогу pg_tablespace

    ```
    SELECT spcname FROM pg_tablespace;
    ```

  - через метакоманду утилиты psql

    ```
    \db
    ```

1 Каталог, в котором создается ТП:
   - должен существовать, быть пустым и принадлежать пользователю ОС, под которым запущен PostgreSQL;
   - не должен размещаться на съёмных или устройствах временного хранения, так как кластер может перестать функционировать из-за потери этого пространства.

2 Все созданные впоследствии объекты, принадлежащие целевому табличному пространству, будут храниться в файлах, расположенных в этом каталоге.

3 ТП должно создаваться от имени суперпользователя кластера БД.

4 После создания ТП можно разрешить обычным пользователям базы данных использовать ТП.

   Для этого нужно предоставить пользователям привилегию CREATE на это ТП.

Указание ТП при создании объектов

- Таблицы, индексы и БД могут храниться в отдельных табличных пространствах
- При создании объекта пользователь с правом CREATE на ТП должен указать имя ТП в качестве параметра команды создания объекта
- Например, при создании таблицы:

```
CREATE TABLE foo(i int) TABLESPACE space1;
```

- С использованием парамера default_tablespace:

```
SET default_tablespace = space1;
CREATE TABLE foo(i int);
```

1 Если параметр default_tablespace имеет не пустое значение, он будет использоваться неявно в качестве значения параметра TABLESPACE в командах CREATE TABLE и CREATE INDEX, если в самой команде не задано предложение TABLESPACE.

Перемещение объектов между ТП

- Для перемещения объектов (например, таблиц) между табличными пространствами в PostgreSQL можно использовать команду ALTER
- Пример перемещения таблицы:

  ```
  ALTER TABLE your_table SET TABLESPACE new_tablespace;
  ```

  где,
  - your_table – это имя таблицы,
  - new_tablespace – это имя ТП, в которое нужно переместить эту таблицу
- Перемещение объекта в другое ТП — физическая операция, связанная с копированием файлов данных из каталога в каталог

1 Можно переместить все объекты из одного ТП в другое.

Например, для переноса всех таблиц можно использовать запрос:

```
ALTER TABLE ALL IN TABLESPACE pg_default SET TABLESPACE
new_tablespace;.
```

Просмотр размера табличного пространства

- Через функцию pg_tablespace_size:

  `SELECT pg_size_pretty( pg_tablespace_size(<имя_ТП>));`

  Например,
  `SELECT pg_size_pretty(pg_tablespace_size('pg_default'));`

  Функция `pg_size_pretty` преобразует размер в байтах в более удобный для восприятия человеком формат с использованием единиц измерения (байты, кБ, МБ, ГБ, ТБ или ПБ, в зависимости от размера)
- Через метакоманду утилиты psql:

  `\db+`

1 Пример выполнения запроса SELECT `pg_size_pretty( pg_tablespace_size(<имя_ТП>));`

```
inventory=# SELECT pg_size_pretty(pg_tablespace_size('pg_default'));
 pg_size_pretty
----------------
 114 MB
(1 строка)

inventory=# SELECT pg_size_pretty( pg_tablespace_size('fastspace'));
 pg_size_pretty
----------------
 4096 bytes
(1 строка)
```

2 Пример выполнение команды `psql \db+`:

```
inventory=# \db+
                                                  Список табличных пространств
    Имя     | Владелец |          Расположение           | Права доступа | Параметры |   Размер   | Описание
------------+----------+---------------------------------+---------------+-----------+------------+----------
 fastspace  | iivanov  | /var/lib/postgresql/11/fastspace |               |           | 4096 bytes |
 pg_default | postgres |                                 |               |           | 114 MB     |
 pg_global  | postgres |                                 |               |           | 623 kB     |
(3 строки)
```

Просмотр ТП, в котором размещен объект

- Для таблицы:

```
SELECT tablename, tablespace FROM pg_tables WHERE
tablename = <имя_таблицы>;
```

- Для индекса:

```
SELECT i.indexname, t.spcname AS tablespace
FROM pg_indexes AS i JOIN pg_class AS c ON c.relname =
i.indexname
LEFT JOIN pg_tablespace AS t ON t.oid = c.reltablespace
WHERE c.relkind = 'i' AND  i.indexname =
'your_index_name';
```

1 Пример выполнения запроса для определения ТП, в котором размещена таблица:

```
SELECT tablename, tablespace FROM pg_tables WHERE tablename
= 'equipment';
```

```
inventory=# SELECT tablespace
            FROM pg_tables
            WHERE tablename = 'equipment';
 tablespace
------------
 fastspace
(1 строка)
```

Удаление ТП

- ТП не может быть удалено до тех пор, пока не будут удалены все объекты во всех базах данных, использующих это ТП
- Удалить пустое табличное пространство можно с помощью команды

```
DROP TABLESPACE <имя_ТП>;
```

1. При попытке удалить непустое табличное пространство возникнет ошибка.
2. Для определения, в каких базах есть зависимые объекты следует:
   - определить идентификатор табличного пространства:

```
SELECT oid FROM pg_tablespace WHERE spcname = <имя_ТП>;
```

   - получить список баз данных, в которых есть объекты из удаляемого пространства:

```
SELECT datname FROM pg_database
WHERE oid IN (SELECT pg_tablespace_databases(<oid_ТП>));
```

   - в каждой из найденных БД получить список отношений, размещенных в ТП:

```
SELECT relnamespace::regnamespace, relname, relkind
FROM pg_class
WHERE reltablespace = <oid_ТП>;
```

3. Удалить найденные объекты, в случае их ненадобности или перенести их в другое ТП.
4. После этого ТП можно удалить.

Вопросы для проверки

1 Какое значение имеет конфигурационный параметр *search_path* по умолчанию?

2 Какой объект БД определяет физическое размещение объектов БД в файловой системе?

3 Что нужно сделать перед удалением табличного пространства?

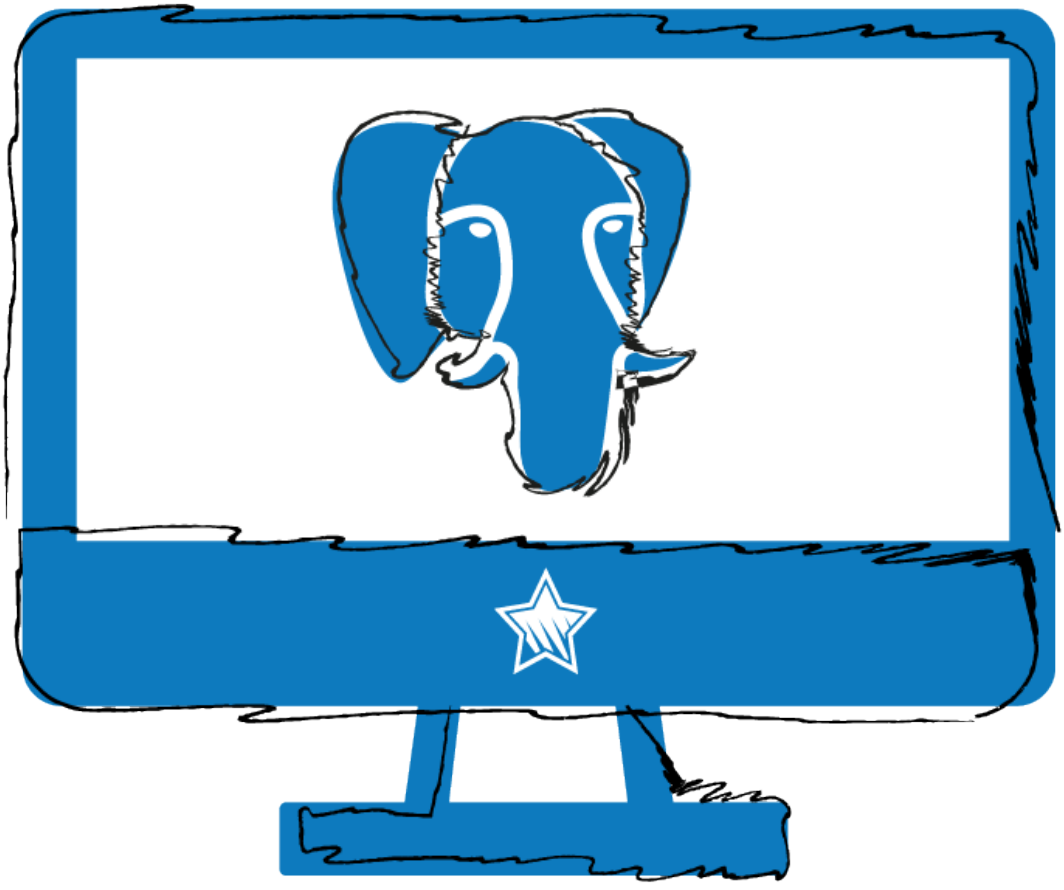
Практическая работа

Оценка времени выполнения: 40 мин.

- Выполните задания модуля 2 из Сборника практических заданий.

Модуль 3

Подключение и аутентификация



ASTRA LINUX®

Модуль 3.
Подключение и аутентификация

Краткое содержание:

- Этапы подключения к СУБД.
- Аутентификация в СУБД PostgreSQL из дистрибутива Astra Linux Special Edition.
- Обеспечение доступа PostgreSQL к данным о мандатных метках пользователей.
- Конфигурационный файл pg_hba.conf: порядок обработки, структура записей, возможные значения полей.
- Методы аутентификации в PostgreSQL.
- Сопоставление имен.

Основные знания и умения:

- понимать механизм идентификации, аутентификации и авторизации;
- понимать особенности аутентификация в СУБД PostgreSQL из дистрибутива Astra Linux Special Edition;
- уметь работать с конфигурационным файлом pg_hba.conf;
- знать различные методы аутентификации в PostgreSQL;
- уметь настраивать сопоставление имен.

Оценка знаний и умений:

- Практическая работа, контрольные вопросы.

Дополнительная литература:

- Отсутствует.

Цели модуля 3. Подключение и аутентификация

- Этапы подключения к СУБД
- Аутентификация в СУБД PostgreSQL из дистрибутива Astra Linux Special Edition
- Обеспечение доступа PostgreSQL к данным о мандатных метках пользователей
- Конфигурационный файл pg_hba.conf: порядок обработки, структура записей, возможные значения полей
- Методы аутентификации в PostgreSQL
- Сопоставление имен

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Этапы подключения к СУБД

- Идентификация:
  - определение имени пользователя, сопоставление его с ролью кластера БД
- Аутентификация:
  - действительно ли пользователь тот, за кого себя выдает?
  - обычно требуется подтверждение (например, проверка пароля)
- Авторизация:
  - имеет ли право данный пользователь подключаться к кластеру БД?
  - в СУБД активное имя пользователя определяет права на объекты БД

1 В защищенной ОС любой пользователь (субъект доступа) перед тем, как начать работу с системой, должен пройти идентификацию, аутентификацию и авторизацию.

2 На первом этапе, при попытке подключения клиента, проходит идентификация пользователя.

3 При попытке соединения с сервером СУБД клиентское приложение указывает пользователя СУБД PostgreSQL от имени которого осуществляется подключение.

4 Аутентифицировать пользователя – значит проверить, что он действительно является тем, за кого себя выдает. Например, выполнить проверку пароля.

5 PostgreSQL предлагает несколько различных методов аутентификации клиента.

6 Метод, используемый для аутентификации конкретного клиентского соединения, может быть выбран на основе адреса узла сети клиента, БД и пользователя.

7 Авторизовать пользователя – определить, имеет ли он право на подключение к кластеру БД, на подключение к БД, на доступ к определенным объектам внутри БД.

8 В пределах окружения SQL активное имя пользователя СУБД определяет права на объекты БД.

- Для подключения требуется сопоставление пользователей СУБД пользователям ОС
- Следует использовать только методы аутентификации, в которых осуществляется сопоставление пользователей СУБД пользователям ОС
- Для пользователей ОС должны быть явно заданы диапазоны его мандатных уровней и категорий

1 Несмотря на то, что имена пользователей СУБД PostgreSQL логически отделены от имен пользователей ОС, в которой запущен сервер, для подключения к СУБД PostgreSQL из дистрибутива Astra Linux Special Edition требуется сопоставление пользователей СУБД пользователям ОС.

2 Если пользователю требуется получать доступ к информации с уровнем конфиденциальности выше 0 или присвоенной категорией конфиденциальности, то при настройке аутентификации в СУБД следует использовать только методы аутентификации, в которых осуществляется такое сопоставление.

Для других пользователей осуществляется доступ только к данным с уровнем конфиденциальности 0, без категории конфиденциальности информации.

Внимание!

Для обеспечения нормальной работы пользователя с сетевыми сервисами должны быть явно заданы диапазоны его мандатных уровней и категорий с помощью соответствующих утилит, даже если ему не доступны уровни и категории выше 0.

Мандатные атрибуты пользователя в ОС

- Мандатные атрибуты, назначенные конкретным локальным учётным записям пользователей, хранятся в каталоге */etc/parsec/macdb*
- Для каждой учётной записи локального пользователя, для которой явно заданы мандатные атрибуты, создается текстовый файл, имя файла совпадает с UID учётной записи пользователя
- Если мандатные атрибуты для пользователя не заданы явным образом, то их значение зависит от настройки параметра `zero_if_notfound` в файле */etc/parsec/mswitch.conf*

```
# Return zero data instead of ENOENT/ENODATA in the absence of record
zero_if_notfound: no
```

1 Дополнительный параметр `zero_if_notfound` в конфигурационном файле /etc/parsec/mswitch определяет поведение системы в случае, если информация о мандатных атрибутах пользователя отсутствует.

2 Если значение параметра `zero_if_notfound` в конфигурационном файле /etc/parsec/mswitch установлено в **no**, то, если мандатные атрибуты пользователя не заданы, при попытке обратится к PostgreSQL будет возникать ошибка:

```
sa@astra:~$ psql -h localhost -d mac_test_db -U user_test
Пароль пользователя user_test:
psql: ВАЖНО:  error obtaining MAC configuration for user "user_test"
```

3 Также ошибка при попытке подключения к PostgreSQL будет возникать, если одноименный пользователь не создан в ОС.

4 Если значение параметра `zero_if_notfound` в конфигурационном файле /etc/parsec/mswitch.conf установлено в **yes**, и мандатные атрибуты пользователя не заданы, то, для обращения к БД, они будут сформированы и установлены в минимальные (нулевые) значения.

При этом значении параметра ошибка при подключении не будет возникать и в случае, когда одноименный пользователь ОС не создан, и в случае, если он создан, но для него не заданы мандатные атрибуты.

5 Команда `pdp-id` отображает мандатные атрибуты сеанса пользователя.

Обеспечение доступа PostgreSQL к данным о мандатных метках пользователей и их привилегиях

- Если для подключения к PostgreSQL используются локальные учетные записи пользователей, то пользователю, в контексте безопасности которого работает СУБД PostgreSQL (по умолчанию — *postgres*), необходимо выдать права на чтение информации:
  - о мандатных атрибутах, назначенных локальным пользователям — права на каталог */etc/parsec/macdb*
  - о привилегиях — права на каталог */etc/parsec/capdb*

1 Для корректного функционирования авторизации через PAM (подгружаемый аутентификационный модуль), пользователю, в контексте безопасности которого работает СУБД PostgreSQL (по умолчанию — postgres), необходимо выдать права на чтение информации из БД пользователей и сведений о мандатных метках и привилегиях.

2 Для выдачи соответствующих разрешений следует выполнить следующие команды:

```
setfacl -d -m u:postgres:r /etc/parsec/macdb
setfacl -R -m u:postgres:r /etc/parsec/macdb
setfacl -m u:postgres:rx /etc/parsec/macdb
setfacl -d -m u:postgres:r /etc/parsec/capdb
setfacl -R -m u:postgres:r /etc/parsec/capdb
setfacl -m u:postgres:rx /etc/parsec/capdb
```

3 Утилита `setfacl` устанавливает списки контроля доступа к файлам и каталогам.

4 Параметры имеют следующие значения:

- `-d` — указывает на то, что команда будет работать со списком контроля доступа, указанным по умолчанию;
- `-m` — позволяет изменить список контроля доступа;
- `-R` — применять команду рекурсивно к подкаталогам.

Конфигурационный файл pg_hba.conf

- Хранит настройки аутентификации
- Размещается в каталоге:

  /etc/postgresql/11/<имя экземпляра>
- Для применения изменений нужно повторно считать параметры конфигурации:

```
sudo pg_ctlcluster 11 main reload
```

1 Настройки аутентификации хранятся в конфигурационном файле *pg_hba.conf* (host-based authentication — аутентификации по имени узла).

2 Расположение определяется параметром hba_file. По умолчанию он расположен в каталоге */etc/postgresql/версия_postgresql/имя_кластера*.

Структура записей файла pg_hba.conf

```
# Database administrative login by Unix domain socket
local   all             postgres                                peer

# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     peer
# IPv4 local connections:
#host    all             all             127.0.0.1/32            md5
host    all             all             0.0.0.0/0            md5
# IPv6 local connections:
host    all             all             ::1/128                 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     trust
host    replication     all             127.0.0.1/32            md5
host    replication     all             ::1/128                 md5
```

1 Файл pg_hba.conf содержит строки, каждая из которых считается отдельной записью.

2 Пустые строки и комментарии (все, после символа #) игнорируются.

Строка состоит из полей, разделенных пробелами или табуляциями.

Записи не продолжаются на следующей строке.

Количество полей может различаться в зависимости от их содержимого.

3 Поля записей:

- тип подключения (TYPE);
- имя базы данных (DATABASE);
- имя пользователя (USER);
- адрес узла (ADDRESS);
- метод аутентификации (METHOD).

4 При установке СУБД создается файл pg_hba.conf, содержащий следующие записи:

- первая запись разрешает пользователю postgres подключаться к СУБД, пройдя аутентификацию в ОС;
- вторая запись относится к локальным не-TCP подключениям (local, unix domain socket) для любых баз (all) и пользователей (all);
- третья запись относится к удаленным подключениям по TCP (host);
- четвертая — удаленные подключения по IPv6.

Далее следуют записи, относящиеся к настройке репликации.

Порядок обработки файла pg_hba.conf

- Записи в файле просматриваются сверху вниз
- Первая запись с соответствующим типом соединения, адресом клиента, указанной базой данных и именем пользователя применяется для аутентификации
- Если запись выбрана, и аутентификация не прошла, последующие записи не рассматриваются
- Если же ни одна из записей не подошла, в доступе будет отказано
- Записи нужно упорядочивать так, чтобы сначала были указаны записи, содержащие более частные условия

1 Для каждой записи в файле проверяется, подходит ли она к запрашиваемому клиентом подключению (соответствует ли тип подключения, имя БД, имя пользователя и IP-адрес условиям, указанным в записи).

Возможные значения в поле «Тип подключения» (TYPE)

- local – управляет подключениями через Unix-сокеты (unix-domain socket)
- host – управляет подключениями, устанавливаемыми по TCP/IP
- hostssl – управляет подключениями, устанавливаемыми по TCP/IP с применением шифрования SSL
- hostnossl – подключения по TCP/IP без шифрования SSL

1 Записи host соответствуют подключениям с SSL и без SSL, а также подключениям, защищённым механизмами GSSAPI, и не защищённым GSSAPI.

2 Сервер принимает подключения только с адресов, определенных через параметр listen_addresses, который указывается в конфигурационном файле postgresql.conf.

По умолчанию значение параметра listen_addresses равно «*», т. е. подключения принимаются с любых IPv4 адресов.

3 Для использования подключений по TCP/IP с применением шифрования SSL механизм SSL должен быть включён параметром конфигурации ssl (в файле postgresql.conf). В PostgreSQL, из дистрибутива Astra Linux Special Edition, этот параметр ssl включен по умолчанию.

4 Подробнее о типах подключения:

https://www.postgresql.org/docs/11/auth-pg-hba-conf.html

Возможные значения в поле «Имя базы данных» (DATABASE)

- `all` – определяет, что для этой записи подходят все базы данных
- `sameuser` – определяет, что данная запись соответствует только если имя запрашиваемой базы данных совпадает с именем запрашиваемого пользователя
- `samerole` – определяет, что запрашиваемый пользователь должен быть членом роли с таким же именем, как и у запрашиваемой БД
- `replication` – показывает, что запись соответствует, если запрашивается подключение для физической репликации
- Любое другое значение воспринимается как имя определённой базы PostgreSQL

1 Поле «Имя базы данных» (DATABASE) определяет, каким именам баз данных соответствует эта запись.

2 *samegroup* — это устаревший, но допустимый вариант значения *samerole*.

3 Для значения replication не выбирается какая-то конкретная база данных.

4 Файл, содержащий имена баз данных, можно указать, поставив знак @ в начале его имени.

5 Несколько имён баз данных можно указать, разделяя их запятыми.

Поле «Имя пользователя» (USER)

- Поле USER указывает, какому имени (или именам) пользователя базы данных соответствует запись
- Значение `all` указывает, что запись соответствует всем пользователям
- Любое другое значение задаёт либо имя конкретного пользователя базы данных, либо имя роли (если это значение начинается с «+»)
- Если перед именем роли стоит знак **+**, то имя соответствует любому пользователю (или роли), входящему в указанную роль
- Например, *+backup_operators* — любой пользователь, входящий в роль backup_operators
- Несколько имён пользователей (ролей) можно указать, разделяя их запятыми

1 В PostgreSQL нет никакой разницы между пользователем и группой, знак + означает «совпадение любых ролей, которые прямо или косвенно являются членами роли».

2 Имя без знака + является подходящим только для этой конкретной роли.

3 Суперпользователь рассматривается как член роли, только если он явно является членом этой роли, прямо или косвенно, а не только потому, что он является суперпользователем.

4 Файл, содержащий имена пользователей, можно указать, поставив знак @ в начале его имени.

Возможные значения в поле «адрес узла» (ADDRESS)

- Может содержать: имя компьютера, диапазон IP-адресов, ключевое слово
- Примеры диапазонов:
  - 172.20.143.89/32 – для одного компьютера
  - 172.20.143.0/24 – для сети
- Ключевые слова:
  - `all` – любой IP-адрес,
  - `samehost` – любые IP-адреса данного сервера
  - `samenet` – любой адрес любой подсети, к которой сервер подключён напрямую

1 В поле «адрес узла» (ADDRESS) указывается адрес (или адреса) клиентской машины, которым соответствует данная запись. Это поле может содержать или имя компьютера, или диапазон IP-адресов, ключевое слово.

2 Диапазон IP-адресов указывается в виде начального адреса диапазона, дополненного косой чертой (/) и длиной маски CIDR (Classless Inter-Domain Routing – бесклассовая междоменная маршрутизация).

Длина маски задаёт количество старших битов клиентского IP-адреса, которые должны совпадать с битами IP-адреса диапазона. Биты, находящиеся правее, в указанном IP-адресе должны быть *нулевыми*. Между IP-адресом, знаком (/) и длиной маски CIDR *не должно* быть пробелов.

3 Указание диапазонов адресов IPv6:

- `::1/128` для одного компьютера (адрес замыкания IPv6);
- `fe80::7a31:c1ff:0000:0000/96` (сайт – локальный адрес). Локальные адреса используются внутри малых или средних сетей, где нет необходимости во внешнем подключении к интернету.

4 Запись `0.0.0.0/0` представляет все адреса IPv4, запись `::0/0` — все адреса IPv6.

5 Чтобы указать один компьютер, следует использовать длину маски **32** для IPv4 или **128** для IPv6.

Опускать замыкающие нули в сетевом адресе нельзя.

6 Запись, сделанная в формате IPv4, подходит только для подключений по IPv4, а запись в формате IPv6 подойдёт только для подключений по IPv6, даже если представленный адрес находится в диапазоне IPv4-в-IPv6.

Записи в формате IPv6 не будут приниматься, если системная библиотека C не поддерживает адреса IPv6.

7 Если определено имя компьютера (всё, что не является диапазоном IP-адресов или специальным ключевым словом, воспринимается как имя компьютера), то оно сравнивается с результатом обратного преобразования IP-адреса клиента (например, обратного DNS-запроса, если используется DNS).

При сравнении имён компьютеров регистр не учитывается.

8 Если имена совпали, выполняется прямое преобразование имени (например, прямой DNS-запрос) для проверки, относится ли клиентский IP-адрес к адресам, соответствующим имени. Если двусторонняя проверка пройдена, запись считается соответствующей компьютеру.

9 В качестве имени узла в файле pg_hba.conf должно указываться то, что возвращается при преобразовании IP-адреса клиента в имя, иначе строка не будет соответствовать узлу.

10 Некоторые базы данных имён позволяют связать с одним IP-адресом несколько имён узлов, но операционная система при попытке разрешить IP-адрес возвращает только **одно** имя.

11 Указание имени, начинающееся с точки (.), соответствует суффиксу актуального имени узла. Например, .example.com будет соответствовать *foo.example.com* (а не только example.com).

12 Когда в pg_hba.conf указываются имена узлов, следует добиться, чтобы разрешение имён выполнялось достаточно быстро. Для этого может быть полезен локальный кеш разрешения имён, например, nscd.

13 Также, можно включить конфигурационный параметр log_hostname, чтобы видеть в журналах имя компьютера клиента вместо IP-адреса.

14 Эти поля не применимы к записям local.

15 Также обратный запрос необходим для того, чтобы реализовать возможность соответствия суффиксов, поскольку для сопоставления с шаблоном требуется знать фактическое имя компьютера клиента.

16 Подробнее: https://www.postgresql.org/docs/11/auth-pg-hba-conf.html

Поля IP-адрес и IP-маска

- Поля IP-адрес и IP-маска могут быть использованы как альтернатива записи IP-адрес/длина-маски
- Вместо того, чтобы указывать длину маски, в отдельном столбце указывается сама маска
- Например:
  - 255.0.0.0 – маска CIDR для IPv4 длиной 8 бит
  - 255.255.255.255 – маска CIDR длиной 32 бита

1 Эти поля не применимы к записям local.

Некоторые возможные значения в поле «метод-аутентификации» (auth-method)

- `trust` – разрешает безусловное подключение
- `reject` – отклоняет подключение безусловно
- `md5` – проверяет пароль пользователя, производя аутентификацию SCRAM-SHA-256 или MD5
- `gss` – для аутентификации пользователя используется GSSAPI
- `peer` – получает имя пользователя операционной системы клиента из операционной системы и проверяет, соответствует ли оно имени пользователя запрашиваемой базы данных

1 В поле «метод-аутентификации» (auth-method) указывается метод аутентификации, когда подключение соответствует этой записи.

Подробнее: https://www.postgresql.org/docs/11/auth-methods.html

2 Все значения методов аутентификации воспринимаются с учётом регистра и должны быть записаны в нижнем регистре, даже аббревиатуры типа ldap.

3 Метод *trust* позволяет тому, кто может подключиться к серверу с базой данных PostgreSQL, войти под любым желаемым пользователем PostgreSQL без введения пароля и без какой-либо другой аутентификации.

Подробнее: https://www.postgresql.org/docs/11/auth-trust.html

4 Метод *reject* полезен для «фильтрации» некоторых серверов группы, например, строка reject может отклонить попытку подключения одного компьютера, при этом следующая строка позволяет подключиться остальным компьютерам в той же сети.

5 Метод *gss* доступен только для подключений по TCP/IP. Он может применяться в сочетании с шифрованием GSSAPI.

Подробнее: https://www.postgresql.org/docs/11/gssapi-auth.html

6 Метод *peer* доступен только для локальных подключений.

Подробнее: https://www.postgresql.org/docs/11/auth-peer.html

Методы аутентификации с использованием пароля

- При парольной аутентификации сервер PostgreSQL запрашивает у пользователя пароль и проверяет его на соответствие паролю, который может храниться:
  - в СУБД
  - во внешней службе
- При использовании методов: password, md5, scram-sha-256 пароль хранится в СУБД
- При использовании метода md5 пароль передается как MD5-хеш
- При использовании метода scram-sha-256 используется протокол SCRAM

1 Методы аутентификации, при которых пароль хранится в СУБД:

- *password* – требуется для аутентификации введения клиентом незашифрованного пароля.
- *md5* – сравнивает MD5-дайджест пароля с MD5-дайджестом, хранящимся в базе.

  При запросе сервер отправляет клиенту так называемую «соль», клиент вычисляет MD5-хеш пароля, добавляет «соль», еще раз вычисляет MD5-хеш и отправляет его на сервер для сравнения с сохраненным хешем.

  Благодаря «соли» хеши одинаковых паролей получаются разными. Однако в настоящее время алгоритм MD5 считается недостаточно криптостойким.
- *scram-sha-256* – проверяет пароль пользователя, производя аутентификацию SCRAM-SHA-256.

2 Поскольку при использовании метода password пароль посылается простым текстом через сеть, такой способ не рекомендуется использовать.

Подробнее: https://www.postgresql.org/docs/11/auth-password.html

3 Наиболее безопасным метод методом с использованием пароля считается scram-sha-256.

Он использует при аутентификации протокол SCRAM и задействует криптостойкий алгоритм SHA-256.

Пароли в СУБД

- При использовании метода аутентификации по паролю, пользователям с незаданным паролем будет отказано в доступе
- Пароли в СУБД хранятся в системном каталоге *pg_authid*
- Пароль хранится в зашифрованном виде, алгоритм шифрования (MD5, SCRAM-SHA-256) определяется параметром *password_encryption*

1 Можно указать время действия пароля.

2 Команда установки пароля

```
[ CREATE | ALTER ] ROLE ...  PASSWORD 'пароль'
[ VALID UNTIL дата_время ];
```

3 Если сохраненный пароль был зашифрован алгоритмом SCRAM-SHA-256, и установлен метод аутентификации MD5, то при обмене сообщениями будет использован более надежный метод SCRAM-SHA-256.

4 Просмотреть значение параметра password_encryption можно с помощью команды SHOW.

```
postgres=# SHOW password_encryption;
 password_encryption
---------------------
 md5
(1 строка)
```

5 Установить метод шифрования можно с помощью команды

```
SET password_encryption='метод';
```

Например,

```
SET password_encryption='scram-sha-256';
```

Хранение пароля во внешний службе

- Методы аутентификации, при которых пароль хранится вне СУБД:
  - `ldap [параметры]` – пароль хранится на сервере LDAP (Lightweight Directory Access Protocol)
  - `radius [параметры]` – пароль хранится на сервере RADIUS (Remote Authentication in Dial-In User Server)
  - `pam [параметры]` – пароль хранится в подключаемом модуле PAM (Pluggable Authentication Module)

1 Подробнее о методе аутентификации LDAP: https://www.postgresql.org/docs/11/auth-ldap.html

2 Подробнее о методе аутентификации radius: https://www.postgresql.org/docs/11/auth-radius.html

3 Подробнее о методе аутентификации pam: https://postgrespro.ru/docs/postgresql/13/auth-pam

Методы ввода пароля

- Вручную
- Задать значение переменной окружения PGPASSWORD
- Создать файл с паролями ~/.pgpass на ОС пользователя
- Формат файла ~/.pgpass:
  - ряд строк формата
    `hostname:port:database_name:role_name:password`
  - строки просматриваются сверху вниз, выбирается первая подходящая
  - файл должен быть доступен только пользователю, который владеет файлом (`rw- - - - - - -`)

1 Можно задать пароль в переменной окружения PGPASSWORD (на клиенте).

Это решение:

- неудобно при подключении к нескольким серверам СУБД,
- не рекомендуется из соображений безопасности.

2 Файл ~/.pgpass является специальным файлом, используемым PostgreSQL для хранения паролей. Он позволяет пользователям автоматизировать процесс аутентификации, когда требуется пароль для входа в базу данных PostgreSQL.

Файл ~/.pgpass создается на хосте пользователя.

3 Формат файла ~/.pgpass представляет собой ряд строк, каждая из которых содержит следующую информацию, разделенную символом « : » :

`hostname:port:database:username:password`

где:

- `hostname` – сервер, на котором работает PostgreSQL,
- `port` – порт, на котором сервер PostgreSQL слушает подключения,
- `database` – имя базы данных, к которой у пользователя есть доступ,
- `username` – имя роли для подключения к СУБД,
- `password` – пароль роли в СУБД.

4 Если есть необходимость использовать символ «:» в своем пароле в файле .pgpass, то нужно его экранировать с помощью символа «\».

5 После создания файла ~/.pgpass необходимо установить правильные права доступа к файлу.

Файл должен быть доступен только пользователю, который владеет файлом, иначе PostgreSQL проигнорирует его.

6 Для предоставления прав пользователю следует использовать команду:

```
chmod 600 ~/.pgpass
```

7 Пример файла ~/.pgpass:

```
localhost:5432:database_name:role_name:password
```

8 Расположение файла ~/.pgpass можно изменить, задав переменную окружения PGPASSFILE.

Внешняя аутентификация

- *peer* – запрос имени пользователя у ядра ОС (для локальных подключений)
- *cert* – аутентификация с использованием клиентского SSL-сертификата
- *gss* – аутентификация Kerberos по протоколу GSSAPI

1 При использовании перечисленных на слайде методов идентификация и аутентификация происходит вне СУБД.

2 Если аутентификация проходит успешно, PostgreSQL получает:
   - имя, указанное при подключении (внутреннее имя СУБД);
   - имя, идентифицированное внешней системой (внешнее имя);
   - если внешняя служба поддерживает мандатное управление доступом, то мандатные атрибуты пользователя.

3 Метод аутентификаци*и peer* работает, получая имя пользователя операционной системы клиента из ядра и используя его в качестве разрешённого имени пользователя базы данных (с возможностью сопоставления имён пользователя).

4 Этот метод поддерживается только для локальных подключений.

5 Для аутентификации по методу *cert* используется клиентский сертификат SSL, поэтому данный способ применим только для SSL-подключений.

6 Когда используется метод cert:
   - сервер потребует от клиента предъявления действительного и доверенного сертификата;
   - пароль у клиента не запрашивается;
   - атрибут *cn* (обычное имя) сертификата сравнивается с запрашиваемым именем пользователя БД, и если они соответствуют, вход разрешается.

- Если cn отличается от имени пользователя базы данных, то может быть использовано сопоставление имён пользователей.

7 *GSSAPI* (Generic Security Service Application Program Interface - общий программный интерфейс сервисов безопасности) — стандартизированный протокол для безопасной аутентификации.

8 GSSAPI обеспечивает автоматическую проверку подлинности (единый вход) для систем, которые её поддерживают.

9 Проверка подлинности реализуется безопасно, но данные, передаваемые через подключение к БД, будут защищены при условии использования шифрования GSSAPI или SSL.

10 Подробнее о методе gss: https://www.postgresql.org/docs/11/gssapi-auth.html

Сопоставление имен

- Чтобы задействовать, нужно указать `map=имя-сопоставления` в поле параметров в pg_hba.conf
- По умолчанию сопоставления имён пользователя определяются в файле pg_ident.conf
- Файл сопоставления ident содержит строки общей формы:

```
map-name system-username database-username
```

- Пример записи в файле pg_ident.conf

```
# MAPNAME       SYSTEM-USERNAME         PG-USERNAME
# на этих хостах у пользователя bob имя robert
omicron         robert                  bob
```

1 Когда используется внешняя система аутентификации, имя пользователя операционной системы, устанавливающего подключение, может не совпадать с именем целевого пользователя (роли) базы данных.

2 Параметр `map` поддерживается для всех методов аутентификации, которые принимают внешние имена пользователей. Так как для разных подключений могут требоваться разные сопоставления, сопоставление определяется параметром `имя-сопоставления` в pg_hba.conf для каждого отдельного подключения.

3 Сопоставления имён пользователя определяются в файле сопоставления *ident*, который по умолчанию называется *pg_ident.conf*.

4 В Astra Linux этот файл по умолчанию хранится в каталоге */etc/postgresql/11/имя_кластера*.

5 Файл сопоставления может быть помещён и в другое место. Настройка выполняется через параметр *ident_file* в файле *postgresql.conf*.

6 Комментарии и пробелы в файле pg_ident.conf применяются так же, как и в файле pg_hba.conf.

7 Имя сопоставления (map-name) является произвольным именем, на которое будет ссылаться файл сопоставления файла pg_hba.conf.

8 Два других поля указывают имя пользователя операционной системы и соответствующее имя пользователя базы данных.

9 Имя map-name может быть использовано неоднократно, чтобы указывать множественные сопоставления пользовательских имён в рамках одного файла сопоставления.

10 Записи в файле подразумевают, что «этот аутентифицированный пользователь может подключиться как пользователь этой базы данных», но не показывают, что эти имена пользователей эквивалентны.

11 Подключение разрешается, если существует запись в файле сопоставления, соединяющая имя, полученное от внешней системы аутентификации, с именем пользователя базы данных, под которым пользователь хочет подключиться.

12 Подробнее: https://www.postgresql.org/docs/11/auth-username-maps.html

Вопросы для проверки

1 Что нужно сделать для того, чтобы к PostgreSQL из дистрибутива Astra Linux Special Edition могли подключаться пользователи, у которых не заданы мандатные атрибуты?

2 В каком конфигурационном файле хранятся настройки аутентификации?

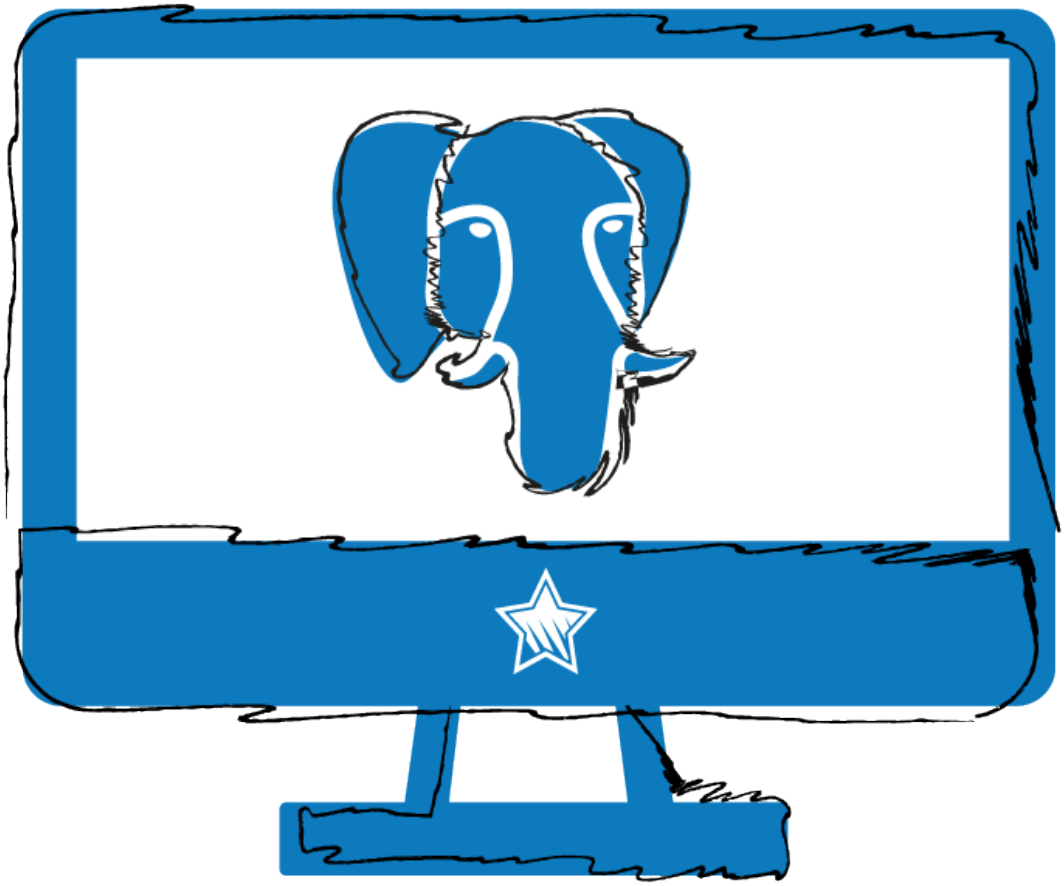
3 Какой конфигурационный файл позволяет указать сопоставления имён пользователей ОС и ролей БД?

Практическая работа

Оценка времени выполнения: 50 мин.

- Выполните задания модуля 3 из Сборника практических заданий.

Модуль 4
Роли в СУБД PostgreSQL



ASTRA LINUX®

Модуль 4.
Роли в СУБД PostgreSQL

Краткое содержание:

- Роли в СУБД PostgreSQL.
- Владельцы объектов БД.
- Членство в роли.
- Использование прав групповой роли.
- Предопределённые роли.

Основные знания и умения:

- уметь создавать и использовать роли в СУБД PostgreSQL;
- понимать механизм владения объектами в СУБД PostgreSQL;
- уметь включать роли в другие роли (и исключать из ролей);
- понимать механизм использования групповой роли;
- знать существующие предопределенные роли и их привилегии.

Оценка знаний и умений:

- Практическая работа, контрольные вопросы.

Дополнительная литература:

- Отсутствует.

Цели модуля 4. Роли в PostgreSQL

- Роли в СУБД PostgreSQL
- Владельцы объектов БД
- Членство в роли
- Использование прав групповой роли
- Предопределённые роли

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Роли в СУБД PostgreSQL

- Роли:
  - могут представлять пользователей и/или группы
  - это объекты кластера
- Одна роль может подключаться к разным базам данных
- Одна роль может быть владельцем объектов в разных БД
- Роль может быть включена в другую роль
- Возможно появление цепочек включений. Циклы не допускаются

1 Субъектами безопасности в PostgreSQL являются *роли*.

2 PostgreSQL использует концепцию ролей для аутентификации и авторизации. Они подобны аккаунтам Linux, но для PostgreSQL нет разницы между пользователями и группами: и те, и другие называются более гибким термином «роль».

3 Роли создаются на уровне кластера (объекты кластера). Т.е. для одного кластера существует один набор ролей.

4 Одна роль может подключаться к разным базам данных и быть владельцем объектов в разных БД.

5 Роль может рассматриваться как «пользователь», «группа» или и то, и другое, в зависимости от того, как она используется.

6 По умолчанию в PostgreSQL создана одна роль суперпользователя с именем *postgres*.

7 Роль может быть включена в другую роль подобно тому, как это происходит с группами в ОС. При этом возможно появление цепочек включений. Циклы не допускаются.

Атрибуты роли

- Атрибуты определяют свойства роли:

```
CREATE ROLE роль [WITH] атрибут [атрибут …];
```

- Примеры атрибутов:
  - `LOGIN` – возможность подключения к кластеру
  - `SUPERUSER` – суперпользователь
  - `CREATEDB` – возможность создавать базы данных
  - `CREATEROLE` – возможность создавать роли
  - `PASSWORD` – задает пароль
- По умолчанию используются ограничивающие доступ атрибуты, например, `NOCREATEDB`

1 Роль базы данных может иметь атрибуты, определяющие её полномочия и взаимодействие с системой аутентификации клиентов.

2 Обычно атрибуты имеют два варианта, например, CREATEDB (дает право на создание БД) и NOCREATEDB (не дает такого права). По умолчанию выбирается ограничивающий вариант.

3 С полным списком атрибутов роли можно ознакомиться в документации PostgreSQL:

https://www.postgresql.org/docs/11/sql-createrole.html

Владельцы объектов БД

- Когда роль создает в базе данных какие-либо объекты, она становится их владельцем
- Владельцами считаются также и роли, включенные в создавшую объект роль
- При необходимости владельца объекта можно сменить
- Для смены владельца используется команда ALTER для соответствующего объекта с предложением OWNER TO:

```
ALTER TABLE имя_таблицы OWNER TO новый_владелец;
```

1 При необходимости владельца объекта можно сменить. Для этого используется команда ALTER для соответствующего объекта с предложением OWNER TO.

Например,

```
ALTER TABLE имя_таблицы OWNER TO новый_владелец;
```

Членство в роли

- В PostgreSQL есть возможность сгруппировать пользователей для упрощения управления правами
- Создаётся роль, представляющая группу, а затем членство в этой группе выдаётся ролям индивидуальных пользователей
- Добавление и удаление членов роли осуществляется через команды GRANT и REVOKE:
  - `GRANT групповая_роль TO роль1, ... ;`
  - `REVOKE групповая_роль FROM роль1, ... ;`
- Членом роли может быть другая групповая роль
- Не допускается управление членством роли PUBLIC

1 В PostgreSQL есть возможность сгруппировать пользователей для упрощения управления правами: права можно выдавать для всей группы (роли) и забирать их у всей группы.

2 Для этого создаётся роль, представляющая группу, а затем членство в этой группе выдаётся ролям индивидуальных пользователей.

3 Обычно групповая роль не имеет атрибута LOGIN, но возможность установить этот атрибут есть.

4 Членом роли может быть другая групповая роль. Не допускается замыкания членства по кругу.

Также не допускается управление членством роли PUBLIC в других ролях.

Использование прав групповой роли

- Выполнение команды SET ROLE во время сеанса позволяет изменить роль, например, на групповую
- Роли, имеющие атрибут INHERIT, автоматически используют права всех ролей, членами которых они являются
- Атрибут INHERIT применяется при создании роли по умолчанию
- Команду GRANT можно вызвать с опцией WITH ADMIN OPTION
- Пользователь (роль), получивший членство в роли с указанием WITH ADMIN OPTION, сможет включать в члены этой роли другие роли (или исключать их)

1 Члены групповой роли могут использовать её права двумя способами:
   - каждый член группы может явно выполнить SET ROLE, чтобы временно «стать» групповой ролью. В этом состоянии сеанс базы данных использует полномочия групповой роли вместо оригинальной роли, под которой был выполнен вход в систему.
   - При этом для всех создаваемых объектов базы данных владельцем считается групповая, а не оригинальная роль;
   - роли, имеющие атрибут INHERIT, автоматически используют права всех ролей, членами которых они являются, в том числе и унаследованные этими ролями права.

2 Вызывать команду GRANT можно с опцией WITH ADMIN OPTION:

```
GRANT имя_роли [, ...] TO указание_роли [, ...]
    [ WITH ADMIN OPTION ]
```

   Пользователь (роль), получивший членство в роли с указанием WITH ADMIN OPTION, сможет включать в члены этой роли другие роли (или исключать их).

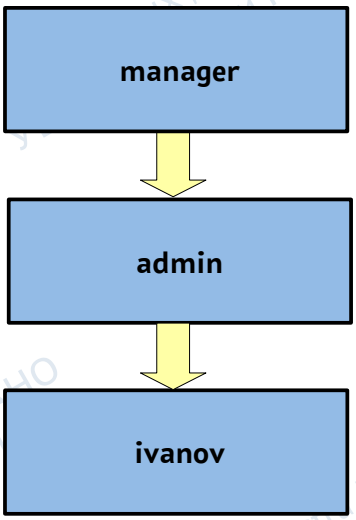
   Роль не имеет права WITH ADMIN OPTION для самой себя.

3 Суперпользователи баз данных могут включать или исключать любые роли из любых ролей.

4 Роли с привилегией CREATEROLE могут управлять членством в любых ролях, кроме ролей суперпользователей.

Пример использования групповой роли

- ```
  CREATE ROLE ivanov LOGIN INHERIT;
  CREATE ROLE admin NOINHERIT;
  CREATE ROLE manager NOINHERIT;
  GRANT admin TO ivanov;
  GRANT manager TO admin;
  ```
- `SET ROLE manager;` – переключение на роль manager
- `SET ROLE ivanov;` – возврат к роли ivanov
- `SET ROLE NONE;` – сброс роли
- `RESET ROLE;` – сброс роли



1 После подключения с ролью ivanov сеанс базы данных будет использовать права, выданные напрямую ivanov, и права, выданные роли admin, так как ivanov наследует права admin (роль ivanov создавалась с атрибутом INHERIT).

2 Права, выданные manager, не будут доступны, потому что, хотя ivanov неявно и является членом manager, это членство получено через роль admin, которая имеет атрибут NOINHERIT.

3 После выполнения команды:

```
SET ROLE admin;
```

сеанс будет использовать только права, назначенные admin, а права, назначенные роли ivanov, не будут доступны.

4 После выполнения команды

```
SET ROLE manager;
```

сеанс будет использовать только права, выданные manager, а права ivanov и admin не будут доступны.

5 Начальный набор прав можно получить с помощью любой из приведенных ниже команд (вернуться к набору привилегий, которые были при подключении к сеансу):

```
SET ROLE ivanov;
SET ROLE NONE;
RESET ROLE;
```
```

Предопределённые роли

- *pg_read_all_settings* — может читать все конфигурационные переменные
- *pg_read_all_stats* — может читать все представления pg_stat_* и использовать различные расширения, связанные со статистикой, даже те, что обычно видны только суперпользователям
- *pg_monitor* — может читать/выполнять различные представления и функции для мониторинга.

  Эта роль включена в роли pg_read_all_settings, pg_read_all_stats и pg_stat_scan_tables

1 В PostgreSQL есть набор предопределённых ролей, которые дают доступ к некоторым часто востребованным, но не общедоступным функциям и данным.

2 Администраторы могут назначать (GRANT) эти роли пользователям и/или ролям в своей среде, таким образом открывая этим пользователям доступ к указанной функциональности и информации.

3 Полный список всех ролей, включая предопределенные системные роли, можно посмотреть в psql командой \duS.

4 Члены роли pg_read_all_settings могут читать даже те конфигурационные переменные, которые обычно видны только суперпользователям.

5 Члены роли pg_stat_scan_tables могут выполнять функции мониторинга, которые могут устанавливать блокировки ACCESS SHARE в таблицах (возможно, на длительное время).

6 Другие предопределенные роли:

- *pg_execute_server_program* — членство в роли разрешает выполнение программ на сервере базы данных от имени пользователя, под которым работает база данных;
- *pg_read_server_files* — членство в роли разрешает чтение файлов из любого места на сервере, к которому может получить доступ база данных, с помощью COPY и других функций доступа к файлам;

- *pg_write_server_files* — членство в роли разрешает запись в файлы, размещенные в любом места на сервере, к которому может получить доступ база данных, с помощью COPY и других функций доступа к файлам;
- *pg_signal_backend* — может передавать другим обслуживающим процессам сигналы для отмены запроса или завершения сеанса.

7 Конкретные разрешения для каждой из предопределённых ролей могут изменяться по мере добавления дополнительной функциональности.

8 Администраторы должны следить за этими изменениями, просматривая замечания к выпускам.

9 Подробнее о преднастроенных ролях: https://www.postgresql.org/docs/current/predefined-roles.html

Вопросы для проверки

1 С помощью какой команды можно включить одну роль в состав другой?

2 Какая роль в PostgreSQL доступна сразу после инсталляции PostgreSQL и обладает правами суперпользователя?

3 Какая команда позволяет изменить роль на групповую во время сеанса?

Практическая работа

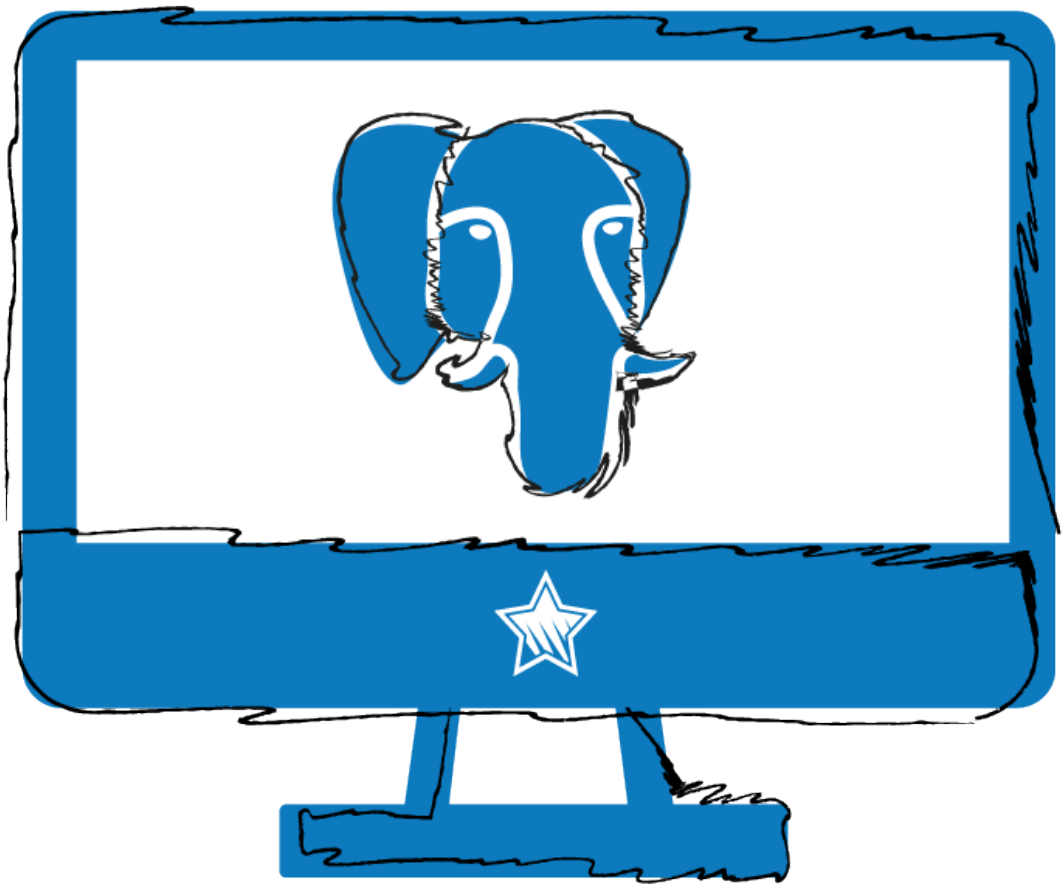
Оценка времени выполнения: 15 мин.

- Выполните задания модуля 4 из Сборника практических заданий.

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Модуль 5

Дискреционное управление доступом в СУБД PostgreSQL в Astra Linux Special Edition



ASTRA LINUX®

Модуль 5.
Дискреционное управление доступом в СУБД PostgreSQL в Astra Linux Special Edition

Краткое содержание:

- Автоматизированное тестирование функциональных возможностей PostgreSQL по разграничению доступа.
- Дискреционное управление доступом в СУБД PostgreSQL.
- Привилегии для работы с объектами СУБД.
- Просмотр списков управления доступом.
- Средства управления дискреционными правами доступа.
- Защита на уровне строк.
- Конфигурационные параметры для настройки дискреционного доступа.

Основные знания и умения:

- выполнять автоматизированное тестирование функциональных возможностей PostgreSQL по разграничению доступа, оценивать результаты;
- понимать механизм дискреционного управления доступом к защищаемым ресурсам БД;
- знать привилегии для работы с объектами СУБД;
- знать средства управления дискреционным доступом к объектам БД;
- уметь настраивать защиту на уровне строк;
- знать конфигурационные параметры дискреционного управления доступом в PostgreSQL из дистрибутива Astra Linux Special Edition;
- знать средства управления дискреционным доступом к объектам БД.

Оценка знаний и умений:

- Практическая работа, контрольные вопросы.

Дополнительная литература:

- Отсутствует.

Цели модуля 5. Дискреционное управление доступом в СУБД PostgreSQL в Astra Linux Special Edition

- Автоматизированное тестирование функциональных возможностей PostgreSQL по разграничению доступа
- Дискреционное управление доступом в СУБД PostgreSQL
- Привилегии для работы с объектами СУБД
- Просмотр списков управления доступом
- Средства управления дискреционными правами доступа
- Защита на уровне строк
- Конфигурационные параметры для настройки дискреционного доступа

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Автоматизированное тестирование функциональных возможностей PostgreSQL по разграничению доступа

- Пакет postgresql-se-test-x содержит тесты дополнительных функциональных возможностей по разграничению доступа
- Команда для установки пакета:
  ```
  sudo apt install postgresql-se-test-11
  ```
- В каталоге /usr/share/postgresql/11/test/pgacext находятся необходимые SQL-скрипты, вспомогательные скрипты и эталоны результатов для выполнения тестов:

```
sa@astra:/usr/share/postgresql/11/test/pgacext$ /bin/ls
expected  runsetests  runtests  sql  support  tests.lst
```

1 В состав КСЗ PostgreSQL входит пакет *postgresql-se-test-x* (где в качестве *x* используется конкретная версия СУБД), содержащий тесты дополнительных функциональных возможностей по разграничению доступа.

2 Тестирование предполагает, что используется режим защищенности ОС Astra Linux Special Edition: *максимальный* («Смоленск»), с включенными мандатным управлением доступа (МРД) и мандатным контролем целостности (МКЦ).

3 Тестирование осуществляется путем создания тестового кластера БД и выполнения SQL-скриптов с запросами, относящимся к тестируемой части функциональности, утилитой командной строки *psql,* предоставляющей доступ к БД.

4 Результат выполнения сохраняется в выходном файле и затем сравнивается с эталонным файлом результатов выполнения.

5 Решение об успешности прохождения теста принимается по результату сравнения.

6 Тест считается выполненным успешно при отсутствии расхождения результатов с эталоном.

7 Каталог *pgacext* имеет структуру:
- *expected* — каталог, содержащий файлы эталонов результатов теста;
- *sql* — каталог, содержащий SQL-скрипты тестов;
- *support* — каталог, содержащий вспомогательные SQL-скрипты и bash-скрипты общих частей тестов, таких как: создание тестовой БД, инициализация объектов, настройка параметров, завершение работы, создание пользователей;

- *runtests* — вспомогательный скрипт для запуска процесса тестирования;
- *tests.lst* — полный список тестов, доступных для текущей версии СУБД. Список тестов может быть изменен для проведения выборочного тестирования;
- *runsetests* — вспомогательный скрипт для запуска процесса тестирования дополнительных функциональных возможностей.

Выполнение тестирования

- Команда для запуска тестирования
  `sudo ./runtests -all`
- Во время тестирования в консоль выводится информация об успешности проведения тестов:

```
+------------------------------------------------------------------------------
| Подготовка к выполнению тестов
| Результаты выполнения тестов будут сохранены в /home/sa/pg-11-se-test
+------------------------------------------------------------------------------
| Запуск тестов
+------------------------------------------------------------------------------
 acl-column             - ДПРД для столбцов.......................... успех
 acl-database           - ДПРД для баз данных........................ успех
```

- Результаты тестирования сохраняются в домашнем каталоге пользователя, запустившего тестирование: */home/имя пользователя/pg-11-se-test/results*

1 В ходе тестов проверяются следующие механизмы:
- дискреционное управление доступом к объектам БД;
- мандатное управление доступом к объектам и данным БД;
- надежное восстановление.

2 Функции защиты, такие как взаимодействие пользователя с КСЗ, идентификация и аутентификация проверяются косвенным образом, т.к. в каждом тесте осуществляется доступ разных пользователей, используется их взаимодействие с КСЗ, и регистрируются все попытки доступа к защищаемым объектам, создания и удаления объектов и действия по изменению правил разграничения доступа.

3 Также проверяются системные функции СУБД, такие как индексирование, резервное копирование/восстановление, работа процедур и триггеров.

4 По завершению на экран выводится сообщение, содержащее общую статистическую информацию о том, как прошло тестирование:

```
+------------------------------------------------------------------------------
| Всего = 69, запущено = 69, ошибочных = 0
+------------------------------------------------------------------------------
Остановка кластеров ...
Удаление тестовых пользователей ...
Удаление тестового табличного пространства в ФС ...
Удаление кластеров ...
```

5 Результаты тестирования сохраняются в домашнем каталоге пользователя, запустившего тестирование.

6 Например, если тестирование проводилось под учетной записью sa, то результаты будут сохранены в каталоге **/home/sa/pg-11-se-test/results**.

```
sa@astra:~$ cd /home/sa/pg-11-se-test/results
sa@astra:~/pg-11-se-test/results$ ls
acl-column.out              mac-dblink.out              mac-triggers-perlu.out
acl-database.out            mac-declarative-part.out    mac-triggers-pgsql.out
acl-dblink.out              mac-delete.out              mac-triggers-pythonu.out
acl-foreign.out             mac-dp.out                  mac-triggers-tcl.out
acl-function.out            mac-files.out               mac-triggers-tclu.out
acl-language.out            mac-foreign.out             mac-update.out
acl-largeobject.out         mac-indexes.out             misc-altertable.out
acl-role.out                mac-inherit.out             misc-audit.out
acl-schema.out              mac-insert.out              misc-cluster.out
acl-sequence.out            mac-joins.out               misc-config.out
acl-table.out               mac-largeobject.out         misc-dump-restore.out
acl-tablespace.out          mac-materializedview.out    misc-dynamic-queries.out
acl-type.out                mac-plperl.out              misc-maclabel.out
acl-view.out                mac-plperlu.out             misc-memory-wiping.out
mac-altermac.out            mac-plpgsql.out             misc-notify.out
```

7 В файлах результатов приведены запросы SQL, которые выполнялись в процессе тестирования, а также приведены фактические и ожидаемые результаты.

Дискреционное управление доступом в СУБД PostgreSQL

- С каждым типом объектов БД ассоциируется набор типов доступа (привилегий)
- Для каждого объекта явно задается список разрешенных для каждого из поименованных субъектов БД (ролей) типов доступа
- *ACL (access control list)* – список управления доступом
- После создания объекта только владелец и администраторы СУБД могут использовать этот объект
- Перед выполнением запроса к БД осуществляется проверка возможности предоставления доступа субъекта к объектам, к которым необходимо обратиться в ходе выполнения запроса

1 В PostgreSQL объектами дискреционного управления доступом могут быть столбцы таблицы, поскольку они однозначно идентифицируются по составному имени таблицы и столбца, т.к. имя столбца внутри таблицы является уникальным.

2 После создания объекта только владелец и администраторы СУБД могут использовать этот объект.

Для того, чтобы с объектом могли работать другие пользователи, владелец объекта или администратор СУБД должен добавить данным пользователям соответствующие дискреционные права доступа.

Право владения объектом предоставляет владельцу объекта возможность модифицировать и удалять этот объект.

Привилегии для работы с объектами СУБД

- БД. Привилегии: CREATE, CONNECT, TEMPORARY/TEMP
- Схема. Привилегии: CREATE, USAGE
- Таблица. Привилегии: SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER
- Столбец таблицы. Привилегии: SELECT, INSERT, UPDATE, REFERENCES
- Табличное пространство. Привилегия: CREATE
- Представление. Привилегии: SELECT, INSERT, DELETE, UPDATE
- Последовательность. Привилегии: SELECT, UPDATE, USAGE
- Функция. Привилегия: EXECUTE

1 Для гибкой работы с данными в СУБД введены объекты и соответствующие привилегии для работы с ними:

- *БД* — способ организации области данных, содержащих остальные объекты СУБД;
- *схема* — способ организации объектов в пределах отдельной БД;
- *табличное пространство* — способ организации БД в ФС ОС;
- *представление* — способ организации предварительно подготовленных запросов;
- *последовательность* — способ получения уникальных значений (счетчик);
- *функция* — программный код манипулирования данными на сервере;
- в БД могут присутствовать дополнительные объекты.

2 Значение привилегий, перечисленных на слайде:

- SELECT — чтение данных из таблицы или представления, чтение значения счетчика (для последовательностей), чтение бинарного объекта;
- INSERT — вставка новых данных;
- DELETE — удаление некоторых/всех данных;
- UPDATE — изменение данных, установка значения счетчика (для последовательностей);

Astra Linux для АНО УЦ ДПО « Айти Клауд»

- USAGE — выполнение функций манипулирования счетчиком (для последовательностей) и использование объектов указанной схемы (для схемы);
- CREATE — создание БД (для БД), объектов БД;
- CONNECT — разрешение на установку соединения с БД;
- TEMPORARY/TEMP — создание временных объектов БД;
- EXECUTE — выполнение функции,
- TRUNCATE — быстрое удаление всех строк из таблицы или набора таблиц,
- REFERENCES – создание ограничения внешнего ключа, ссылающегося на определённую таблицу либо на определённые столбцы таблицы,
- TRIGGER – позволяет создавать триггеры в заданной таблице.

3 Предоставлять права доступа к объектам, по умолчанию, разрешено только администратору БД, но может быть предоставлено владельцу объекта.

4 Кроме рассмотренных прав доступа, существуют права, которые всегда принадлежат владельцам объектов и администраторам СУБД:

- удаление (DROP) и модификация объектов (ALTER);
- назначение пользователям делегируемых прав доступа к сущностям.

Данные права не могут быть предоставлены или отменены средствами СУБД.

5 При выполнении любого запроса пользователя (субъекта БД) к защищаемому ресурсу (объекту БД) выполняется дискреционное управление доступом на основе установленных пользователю прав.

6 Для каждой выполняемой операции производится проверка наличия права у пользователя на выполнение данной конкретной операции.

7 Ознакомиться более подробно со списком привилегий можно на сайте postgresql.org:

https://www.postgresql.org/docs/11/ddl-priv.html

Просмотр списков управления доступом через psql

- `\dp <имя объекта>` – в psql выводит список управления доступом к объекту
- Например, команда `\dp inventory` выведет список прав доступа к таблице inventory

```
inventory=# \dp supp_dept.equipment
                                  Права доступа
   Схема   |    Имя    |   Тип   |      Права доступа       | Права для столбцов | Политики
-----------+-----------+---------+--------------------------+--------------------+----------
 supp_dept | equipment | таблица | iivanov=arwdDxt/iivanov +|                    |
           |           |         | tech_support=arw/iivanov |                    |
(1 строка)
```

1 Ниже приведен список сокращений, обозначающих привилегии для таблиц:
   - `a = insert`
   - `r = select`
   - `w = update`
   - `d = delete`
   - `D = truncate`
   - `x = reference`
   - `t = trigger`

2 С полным списком сокращений, обозначающих тип доступа, можно ознакомиться на сайте https://www.postgresql.org/docs/11/ddl-priv.html

Просмотр списков управления доступом через запрос к системному каталогу

- Вывод списка разрешений на объект (например, на таблицу) через SQL-запрос к системному каталогу:

```
select relacl from pg_catalog.pg_class where
relname='<TABLENAME>';
```

- Пример:

```
select relacl from pg_catalog.pg_class where
relname='mactable';
```

```
mac_test_db=# select relacl from pg_catalog.pg_class where relname='mactable';
                                    relacl
-------------------------------------------------------------------------------
 {postgres=arwdDxt/postgres,user_ordinary=arw/postgres,user_secret=arw/postgres}
(1 строка)
```

1 Для означения привилегий, выводимых через запрос SQL, применяются те же сокращения, что и при выводе списка привилегий через команду `psql \ dp <имя объекта>`.

Средства управления дискреционными правами доступа

- Команда GRANT — используется для предоставления дискреционных прав доступа к объектам
- Команда REVOKE — используется для отмены (отзыва) дискреционных прав доступа к объектам
- Следует учитывать иерархию объектов
- Примеры:
  - `GRANT UPDATE ON "Счета" TO ivanov;`
  - `GRANT ALL ON "Счета" TO PUBLIC;`
  - `GRANT UPDATE ON "Счета" TO ivanov WITH GRANT OPTION;`

1 Синтаксис команды GRANT зависит от того, к какому объекту предоставляется привилегия на доступ. Подробнее с синтаксисом можно ознакомиться на сайте postgresql.org:

https://www.postgresql.org/docs/11/sql-grant.html

2 С синтаксисом команды REVOKE можно ознакомиться на сайте postgresql.org: https://www.postgresql.org/docs/11/sql-revoke.html

3 Следует учитывать иерархию объектов при предоставлении разрешений.

Например, для работы с таблицей, кроме разрешений на таблицу, должно быть предоставлено разрешение USAGE на схему, в которой создана эта таблица.

4 Примеры на слайде:

- `GRANT UPDATE ON "Счета" TO ivanov`
  Предоставление пользователю ivanov права на изменение данных в таблице «Счета»;
- `GRANT ALL ON "Счета" TO PUBLIC`
  Предоставление всем пользователям полного доступа к таблице «Счета» (для предоставления прав доступа к объекту сразу всем пользователям, существует специальное «имя пользователя» PUBLIC, для предоставления всех прав — специальное «право» ALL).

- `GRANT UPDATE ON "Счета" TO ivanov WITH GRANT OPTION`
  Владелец объекта может отменить собственные делегируемые права, например, переведя объект в режим «только для чтения» для себя, так же как и для всех остальных пользователей.
- Право доступа может быть предоставлено с возможностью делегирования данного права другим ролям. Для этого используется ключевая фраза `WITH GRANT OPTION`.

Доступ к объектам по умолчанию

- Суперпользователи имеют полный доступ ко всем объектам
- Проверки на наличие доступа не выполняются
- Владельцы объектов изначально получают полный набор прав доступа к объекту
- Могут удалять объекты, которыми владеют, изменять их, выдавать и отзывать привилегии на их использование
- Остальным ролям привилегии на работу с объектами нужно выдавать
- Для назначения права всем ролям в системе можно использовать специальное имя роли: PUBLIC. PostgreSQL по умолчанию назначает роли PUBLIC права для некоторых типов объектов после их создания

1 Для таблиц, столбцов, последовательностей, обёрток сторонних данных, сторонних серверов, больших объектов, схем и табличных пространств PUBLIC по умолчанию никаких прав не получает.

2 Подробнее с правами, предоставляемыми роли PUBLIC по умолчанию, можно ознакомиться на сайте postgresql.org: https://www.postgresql.org/docs/current/ddl-priv.html *Таблица 5.2. Summary of Access Privileges*

Защита на уровне строк

- Защита на уровне строк *RLS (Row-Level Security)*:
  - позволяет управлять доступом к таблице на уровне строк
  - настраивается через политики, установленные на уровне таблиц
  - ограничивает для пользователей наборы строк, которые могут быть возвращены обычными запросами, добавлены, изменены и удалены командами, изменяющими данные
- Защита на уровне строк – дополнительный механизм
- Для получения доступа роль должна иметь привилегии на уровне дискреционного доступа

1 Другое название инструмента *RLS – Fine Grained Access Control*, детальный контроль доступа.

2 По умолчанию таблицы не имеют политик, так что если система прав SQL разрешает пользователю доступ к таблице, все строки в ней одинаково доступны для чтения или изменения.

3 Подробно о политике защиты строк:

https://www.postgresql.org/docs/11/ddl-rowsecurity.html

Включение защиты строк

- Включение RLS выполняется с помощью команды
  `ALTER TABLE ... ENABLE ROW LEVEL SECURITY`
- После включения все обычные запросы к таблице на выборку или модификацию строк должны разрешаться политикой защиты строк
- Если политика для таблицы не определена, применяется политика запрета по умолчанию, так что никакие строки в этой таблице нельзя увидеть или модифицировать
- Политики не действуют:
  - на владельца таблицы
  - на суперпользователей и роли с атрибутом BYPASSRLS

1 На операции с таблицей в целом, такие как TRUNCATE и REFERENCES, защита строк не распространяется.

2 На владельца таблицы защита строк не действует, но он может включить её для себя принудительно, выполнив команду:
`ALTER TABLE ... FORCE ROW LEVEL SECURITY.`

3 Право включать или отключать защиту строк, а также определять политики для таблицы, имеет только её владелец.

4 Для отключения RLS выполняется команда:
`ALTER TABLE ... DISABLE ROW LEVEL SECURITY;.`

Управление политиками

- Политики защиты строк могут применяться к определённым командам и/или ролям
- Политику можно определить как применяемую к командам ALL (всем), либо `SELECT`, `INSERT`, `UPDATE` и `DELETE`
- Команды для управления политиками:
  - `CREATE POLICY` — создание
  - `ALTER POLICY` — изменение
  - `DROP POLICY` — удаление

1. Каждой политике назначается имя, при этом для одной таблицы можно определить несколько политик.
2. Так как политики привязаны к таблицам, каждая политика для таблицы должна иметь уникальное имя.
3. В разных таблицах политики могут иметь одинаковые имена.

Создание и принцип работы политики

- Синтаксис:
  `CREATE POLICY имя_политики ON имя_таблицы TO список_ролей [AS RESTRICTIVE] USING (предикат);`
- Чтобы определить, какие строки будут видимыми или могут изменяться в таблице, для политики задаётся выражение, возвращающее логический результат (предикат)
- Это выражение будет вычисляться для каждой строки перед другими условиями
- Строки, для которых это выражение возвращает не true, обрабатываться не будут (попадать в выборку, обновляться и т.д.)

1. Политику можно связать с несколькими ролями, при этом действуют обычные правила членства и наследования.
2. Выражения политик обрабатываются в составе запроса с правами исполняющего его пользователя, но для обращения к данным, недоступным этому пользователю, в этих выражениях могут применяться функции, определяющие контекст безопасности.

Комбинирование политик

- Если политики *разрешительные* (permissive policies) – они объединяются логическим сложением (OR)
- Если политики *ограничительные* (restricted policies) – они объединяются логическим умножением (AND)
- По умолчанию создаются разрешительные политики
- Чтобы строка была доступна, достаточно, чтобы хотя бы один из предикатов этих политик был истинен
- Доступ должны предоставить хотя бы одна разрешительная и все ограничительные политики

1. Если на таблице включена защита строк, и при этом не определено ни одной разрешительной политики, не будет доступна ни одна строка.
2. Пример создание ограничительной политики:

```
CREATE POLICY user_orders_only
ON orders
AS RESTRICTIVE
USING (user_id = current_user_id);
```

3. Ограничительные политики применяются до разрешительных политик, и все ограничительные политики должны быть истинными, чтобы строка была доступна.
4. Если какая-либо из ограничительных политик отсутствует, строка не будет доступна, даже если все разрешительные политики истинны.

Пример 1. Создания простой политики для роли managers

- Политика для таблицы account, позволяющая только членам роли managers обращаться к строкам таблицы, причем только к тем, которые они сами внесли в таблицу:

```
CREATE TABLE accounts (manager text DEFAULT CURRENT_USER,
company text, contact_email text);
ALTER TABLE accounts ENABLE ROW LEVEL SECURITY;
CREATE POLICY account_managers ON accounts TO managers
    USING (manager = current_user);
```

1 При применении этой политики:
   - один менеджер не может выполнить операции SELECT, UPDATE или DELETE для существующих строк, принадлежащих другому менеджеру;
   - командами INSERT и UPDATE нельзя создать строки, принадлежащие другому менеджеру.

2 Для заполнения значениями столбца manager, например, можно использовать значение по умолчанию (DEFAULT) CURRENT_USER.

Пример 2. Создания простой политики для всех пользователей

- Чтобы все пользователи могли обратиться только к собственной строке в таблице *users*, можно использовать политику:

```
CREATE POLICY user_policy ON users
    USING (user_name = current_user);
```

1 Если роль не задана, либо задано специальное имя пользователя PUBLIC, политика применяется ко *всем* пользователям в данной системе.

Пример 3. Комбинирование политик

- Пара политик позволяет всем пользователям видеть все строки в таблице *users*, но изменять только свою собственную:

```
CREATE POLICY user_sel_policy ON users
    FOR SELECT
    USING (true);
CREATE POLICY user_mod_policy ON users
    USING (user_name = current_user);
```

Параметры конфигурации дискреционного контроля доступа

- Параметры содержатся в конфигурационном файле *postgresql.conf*
- Параметры, относящиеся к дискреционному доступу:
  - `ac_enable_trusted_owner (по умолчанию true)`
  - `ac_enable_grant_options (по умолчанию false)`
  - `ac_enable_admin_options (по умолчанию false)`
  - `ac_enable_truncate (по умолчанию true)`
- Параметры могут принимать значения *false*, *true*, например:

```
ac_enable_grant_options = false
```

1 С помощью параметров конфигурации можно изменить поведение СУБД касательно дискреционного контроля доступа.

Изменение параметров производится в конфигурационном файле *postgresql.conf*.

2 Параметр `ac_enable_trusted_owner` разрешает администратору запретить владельцам объектов передавать права на доступ к ним другим ролям СУБД. Если значение этой переменной FALSE — распределение прав доступа к объектам БД разрешено только администраторам СУБД.

3 Параметр `ac_enable_grant_options` позволяет администратору запретить передачу имеющихся прав доступа другим ролям.

Если `ac_enable_grant_options = false`, то запрещается использовать команду GRANT с привилегией WITH GRANT OPTIONS.

Если у роли уже есть привилегия GRANT OPTIONS и `ac_enable_grant_options = false`, то передача прав доступа другим ролям также запрещается.

Изъятие (REVOKE) привилегии GRANT OPTIONS разрешается всегда.

4 Параметр `ac_enable_admin_options` позволяет администратору запретить передачу прав членства роли другим ролям.

Если `ac_enable_admin_options = false`, то запрещается использовать GRANT с привилегией WITH ADMIN OPTION.

Если у роли есть привилегия ADMIN OPTIONS и `ac_enable_admin_options = false`, то передача прав членства другим ролям также запрещается.

Изъятие (REVOKE) привилегии ADMIN OPTIONS разрешается всегда.

5 Параметр `ac_enable_truncate` позволяет администратору запретить владельцам объектов и любым пользователям, обладающим правом TRUNCATE, выполнять удаление записей из таблиц.

6 Операция TRUNCATE быстро удаляет все строки из набора таблиц. Удаление строк через команду TRUNCATE имеет тот же эффект, что и работа через команду DELETE, но при выполнении команды TRUNCATE не выполняется сканирование таблиц и операция проходит быстрее. Кроме того, операция TRUNCATE немедленно освобождает место на диске, вместо того чтобы требовать последующей операции очистки (vacuum). Это наиболее полезно для больших таблиц.

Если `ac_enable_truncate = false` — выполнение команды TRUNCATE запрещено всем пользователям.

7 Параметры конфигурационного файла, заданные по умолчанию:

```
sa@astra:/etc/postgresql/11/main$ grep ac_ postgresql.conf
ac_enable_maclabels_on_files    = false
ac_ignore_maclabel              = false
ac_ignore_socket_maclabel       = true
ac_ignore_server_maclabel       = true
ac_enable_trusted_owner         = true
ac_enable_truncate              = true
ac_enable_sequence_ccr          = true
ac_enable_copy_to_file          = true
ac_auto_adjust_macs             = false
ac_enable_dblink_mac            = false
ac_enable_grant_options         = false
ac_enable_admin_options         = false
ac_caps_ttl                     = 10            # rbt capabilities Time to live, in seconds
ac_debug_print          = false
ac_audit_mode           = 'internal'            # rbt audit mode. Possible values are
#ac_audit_file          = 'ConfigDir/pg_audit.conf'
ac_audit_destination    = 'all'                 # rbt Audit destination. Possible values are
ac_audit_log_only_failures = false
ac_log_stmt_length              = 100           # rbt audit stmt length, 0 - unlimited
ac_wipe_buffers         = false
ac_wipe_pages           = false
ac_wipe_files           = false
```

Вопросы для проверки

1 Какие привилегии можно выдать на схему в PosgreSQL?

2 С помощью какой команды psql можно вывести список управления доступом к объекту?

3 На уровне какого объекта включается защита строк?

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Практическая работа

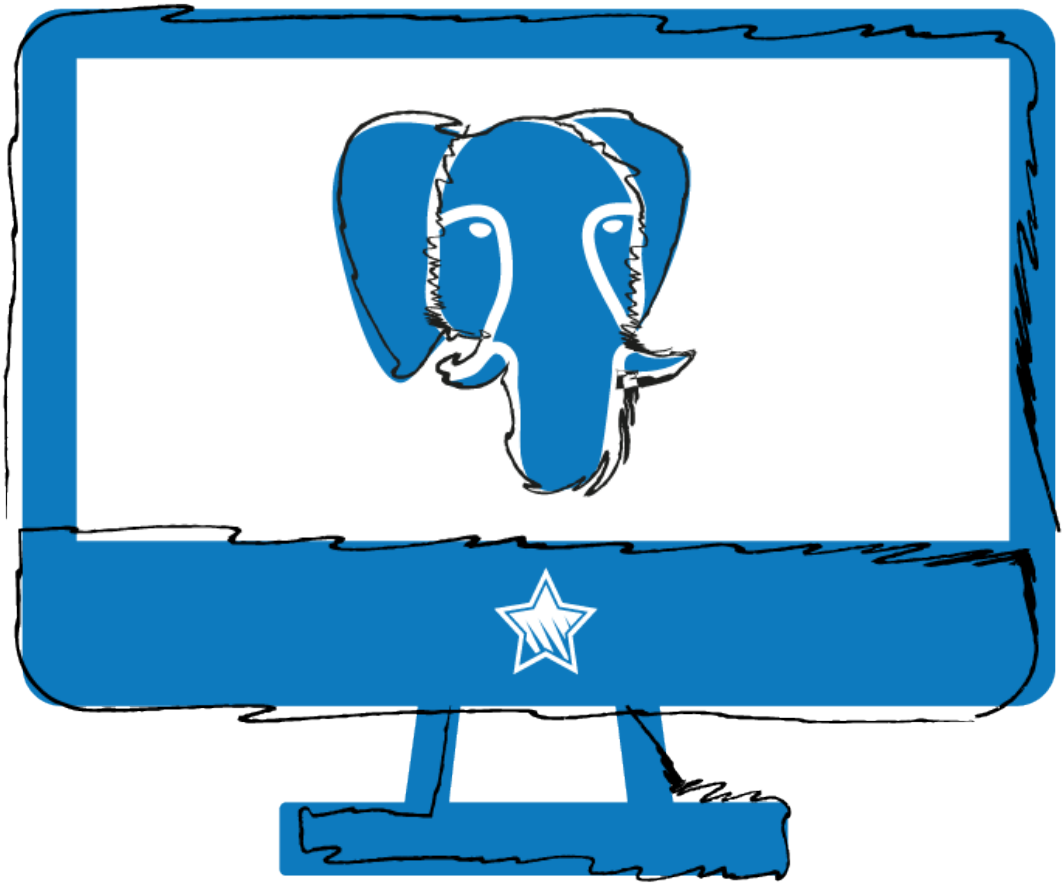
Оценка времени выполнения: 50 мин.

- Выполните задания модуля 5 из Сборника практических заданий.

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Модуль 6

Мандатное управление доступом в СУБД PostgreSQL в Astra Linux Special Edition



ASTRA LINUX®

Модуль 6.
Мандатное управление доступом в СУБД PostgreSQL в Astra Linux Special Edition

Краткое содержание:

- Мандатное управление доступом в Astra Linux.
- Мандатные атрибуты, присваиваемые сущностям и субъектам в ОС.
- Принципы обеспечения защиты от НСД при использовании мандатного управления доступом.
- Средства управления мандатными правилами разрешения доступа.
- Мандатное управление доступом в СУБД PostgreSQL.
- Мандатные атрибуты сеанса пользователя в СУБД PostgreSQL.
- Применение мандатного управления доступом.
- Средства управления мандатным доступом к объектам БД.

Основные знания и умения:

- понимать принципы мандатного управления доступом в Astra Linux;
- знать мандатные атрибуты, присваиваемые сущностям и субъектам в ОС;
- уметь использовать средства управления мандатными правилами разрешения доступа;
- понимать средства управления мандатным доступом к объектам БД;
- понимать целостность классификационных меток кластера БД;
- знать дополнительные опции настройки дискреционного и мандатного доступа с помощью конфигурационных файлов.

Оценка знаний и умений:

- Практическая работа, контрольные вопросы.

Дополнительная литература:

- Отсутствует.

Цели модуля 6. Мандатное управление доступом в СУБД PostgreSQL в Astra Linux Special Edition

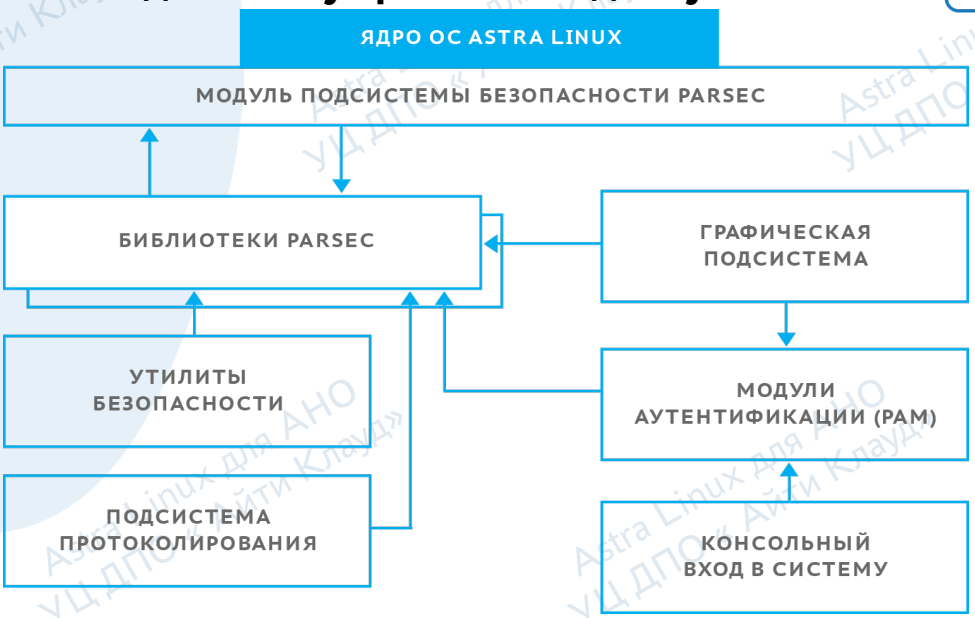
- Мандатное управление доступом в Astra Linux
- Мандатные атрибуты, присваиваемые сущностям и субъектам в ОС
- Принципы обеспечения защиты от НСД при использовании мандатного управления доступом
- Средства управления мандатными правилами разрешения доступа
- Мандатное управление доступом в СУБД PostgreSQL
- Мандатные атрибуты сеанса пользователя в СУБД PostgreSQL
- Применение мандатного управления доступом
- Средства управления мандатным доступом к объектам БД

Astra Linux для АНО УЦ ДПО « Айти Клауд»

- *Мандатная модель управления доступом* (Mandatory Access Control, MAC) — способ разграничения доступа с фиксированным набором полномочий
- MAC используется в системах с повышенным требованиями к безопасности и стоит на службе всевозможных силовых ведомств и организаций, связанных с государственной или служебной тайной
- При реализации политики мандатного управления доступом субъектам и сущностям задаются классификационные метки (уровни конфиденциальности и категории конфиденциальности)

1. При реализации политики мандатного управления доступом субъектам и сущностям задаются *классификационные метки* (уровни конфиденциальности и категории конфиденциальности).
2. Также сущностям могут быть присвоены дополнительные мандатные атрибуты управления доступом.
3. Для администрирования подсистем мандатного управления доступом и мандатного контроля целостности множество Linux-привилегий расширено специальными привилегиями.

Реализация мандатного управления доступом



1 Реализация мандатного управления доступом в ОС Astra Linux Special Edition основана на подсистеме безопасности PARSEC, включающей соответствующий программный интерфейс и модуль расширения ядра ОС, поддерживающий виртуальную файловую систему /parsecfs, и набор системных вызовов, позволяющих уполномоченным пользователям управлять политикой безопасности ОС.

2 Помимо мандатного управления доступом, подсистема безопасности PARSEC также реализует мандатный контроль целостности и дополнительные функции аудита. На рисунке показано место подсистемы безопасности PARSEC в архитектуре ОС Astra Linux и порядок её взаимодействия с другими компонентами ОС.

Основные термины используемые в механизме мандатного управления доступом

- *Субъект мандатного доступа* — тот, кто выполняет операции, подлежащие мандатному контролю (пользователь или процесс)
- *Сущность (объект) мандатного доступа* — то, с чем выполняются операции, подлежащие мандатному контролю (файл, каталог и т.д.)
- *Контейнер* — структурированная сущность доступа, т.е. сущность (каталог ФС), которая может содержать другие сущности доступа (каталоги или файлы)

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Мандатные атрибуты, присваиваемые сущностям и субъектам

- Сущностям и субъектам присваиваются следующие мандатные атрибуты:
  - *Иерархический* уровень конфиденциальности (уровень конфиденциальности)
  - *Неиерархическая* категория конфиденциальности (категории конфиденциальности)
  - *Уровень целостности* (неиерархический уровень целостности и иерархический (линейный) уровень целостности)
  - *Дополнительные* мандатные атрибуты управления доступом

1 *Иерархический* уровень конфиденциальности (уровень конфиденциальности) — определяет степень секретности документа (сущности) и уровень доступа к этому документу, назначенный персоналу (субъекту).

2 Субъекту с определенным уровнем конфиденциальности разрешено читать документы с таким же уровнем конфиденциальности или ниже и запрещено читать документы с более высоким уровнем. Персоналу с более высоким уровнем конфиденциальности запрещено передавать документы высокого уровня конфиденциальности персоналу с более низким уровнем.

3 *Неиерархическая* категория конфиденциальности (категории конфиденциальности) — разделение по категориям конфиденциальности.

4 Персонал, работающий с первой категорией конфиденциальности, имеет соответствующую категорию конфиденциальности. Не имея вторую категорию конфиденциальности, персонал не может иметь доступ к материалам второй категории, а также не может передавать материалы первой категории конфиденциальности персоналу, не имеющему первую категорию. Доступ может быть предоставлен одновременно к нескольким категориям конфиденциальности.

5 *Уровень целостности* (неиерархический уровень целостности и иерархический (линейный) уровень целостности) — субъект, работающий на некотором уровне целостности, может записывать (изменять) только сущности своего или более низкого уровня целостности. Иерархический уровень целостности в ОС зарезервирован, и на уровне пользователя не поддерживается его использование.

6 *Дополнительные* мандатные атрибуты управления доступом — являются необязательными атрибутами и позволяют уточнять или изменять правила мандатного доступа для отдельных контейнеров, субъектов или сущностей.

7 Мандатные атрибуты субъекта/сущности объединяются в *мандатный контекст* этого субъекта/сущности.

Уровень конфиденциальности

- Уровень конфиденциальности представляет собой числовое значение от 0 до 255 (включительно)
- Каждой классификационной метке в каждый момент времени может быть назначен только *один* уровень конфиденциальности
- Субъект с определенным уровнем конфиденциальности может получить доступ на чтение к сущности, если его уровень конфиденциальности не ниже уровня конфиденциальности сущности
- Субъект с определенным уровнем конфиденциальности может получить доступ на запись к сущности, если его уровень конфиденциальности совпадает с уровнем конфиденциальности сущности

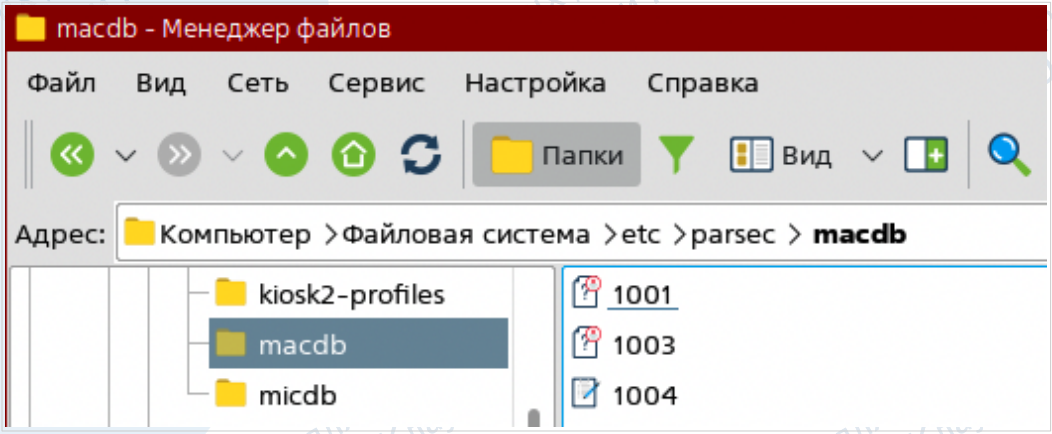
1 Пользователь может читать файлы, уровень конфиденциальности которых не превосходит его уровня доступа, и передавать данные только на одном уровне конфиденциальности. Это происходит при условии, что пользователю не присвоены специальные привилегии, или если файлам не заданы дополнительные мандатные атрибуты доступа.

Категория конфиденциальности

- Множество категорий конфиденциальности определяет набор категорий, доступных субъекту или назначенных сущности
- В ОС можно использовать до 64 категорий конфиденциальности
- Каждой классификационной метке в каждый момент времени может быть присвоен единственный набор категорий конфиденциальности
- Категории конфиденциальности могут быть несравнимы между собой

1. Категории конфиденциальности представляют собой *битовую маску,* состоящую из набора категорий конфиденциальности, каждая из категорий представлена отдельным разрядом битовой маски. Если значение разряда битовой маски равно 1, то у сущности (субъекта) есть категория конфиденциальности, если 0 — категория отсутствует.
2. Субъект с определенным набором категорий конфиденциальности может получить доступ на чтение к сущности, если в его наборе категорий конфиденциальности присутствуют все категории, заданные данной сущности.
3. Субъект с определенным набором категорий конфиденциальности получит доступ на запись к сущности, если его набор категорий конфиденциальности соответствует набору категорий, заданных данной сущности.
4. Пользователь может читать файлы, если набор категорий конфиденциальности доступен пользователю, и передавать данные, если эти наборы совпадают.

Мандатные атрибуты




```
sa@astra:~$ sudo cat /etc/parsec/macdb/1001
student:0:0:0:0
```

1 Мандатные атрибуты, назначенные конкретным учётным записям пользователей, хранятся в каталоге /etc/parsec/macdb.

2 Для каждой учётной записи пользователя, для которой указан ненулевой мандатный уровень или не пустой перечень неиерархических категорий, создается текстовый файл, имя файла совпадает с UID учётной записи пользователя. Файл включает в себя строку вида:

`<username>:<min_level>:<min_categories>:<max_level>:<max_categories>`

где:

- `username` — имя учётной записи пользователя;
- `min_level` — минимальный мандатный уровень, доступный процессам от имени учётной записи пользователя;
- `min_categories` — числовое значение битовой маски, задающее минимальный набор неиерархических категорий, установленных для данной учётной записи пользователя;
- `max_level` — максимальный мандатный уровень, доступный процессам от имени учётной записи пользователя;
- `max_categories` — числовое значение битовой маски, задающей максимальный набор неиерархических категорий, установленный для данной учётной записи пользователя.

При использовании уровней и категорий обеспечивается защита от НСД

- Невозможность прочитать информацию, к которой нет доступа:
  - нижним уровням конфиденциальности запрещено читать информацию с верхних уровней конфиденциальности
  - всем запрещено читать информацию, на которую нет разрешенной категории конфиденциальности
- Невозможность передать информацию:
  - верхним уровням конфиденциальности запрещено записывать свою информацию на нижние уровни конфиденциальности
  - всем запрещено передавать информацию тем, у кого нет соответствующей категории конфиденциальности

1 Принятие решения о запрете или разрешении доступа субъекта к сущности принимается на основе типа операции (чтение/запись/исполнение), мандатного контекста безопасности субъекта и мандатного контекста безопасности сущности.

2 Субъекты или сущности, которым явно не задан мандатный контекст, считаются имеющими *минимальный* (нулевой) мандатный контекст, т.е. мандатный контекст, в котором метка безопасности и все мандатные атрибуты имеют минимальные допустимые значения (например, равные нулю).

Средства управления мандатными правилами разрешения доступа в режиме командной строки

- `pdpl-file` — управление мандатными атрибутами (МА) файлов
- `pdp-id` — отображение мандатных атрибутов сеанса пользователя
- `pdp-init-fs` — скрипт инициализации мандатных атрибутов ФС
- `pdp-ls` — вывод информации о файлах с отображением МА
- `pdpl-ps` — управление мандатными атрибутами процессов
- `pdpl-user` — управление допустимыми мандатными уровнями (МУ) и мандатными категориями (МК) пользователей ОС
- `sumac` — запуск процесса с заданными МУ и категорией
- `userlev` — изменение БД МУ
- `usercat` — изменение БД МК

Метка безопасности

- *Метка безопасности* состоит из:
  - Классификационной метки, которая определяется:
    - Иерархическим уровнем конфиденциальности
    - Неиерархической категорией конфиденциальности
  - Метки целостности — определяется уровнем целостности
- Метка безопасности контейнера определяет максимальную метку безопасности вложенных сущностей
- *Уровень конфиденциальности* представляет собой единичное (скалярное) числовое значение от 0 до 255 (включительно)

1 Каждой классификационной метке в каждый момент времени может быть назначен только один уровень конфиденциальности.

2 Числовые значения уровня конфиденциальности сравнимы между собой и технически реализованы как 8-битная беззнаковая величина (uint8_t). В пользовательских интерфейсах представляется десятичным значением или наименованием единичного уровня конфиденциальности.

3 Категория конфиденциальности представляет собой маску, состоящую из набора единичных значений категорий конфиденциальности. В ОС реализовано использование до 64 единичных категорий конфиденциальности.

4 Каждой классификационной метке в каждый момент времени могут быть назначены одновременно до 64 категорий конфиденциальности.

5 Единичные категории конфиденциальности несравнимы между собой.

6 Числовые значения категории конфиденциальности частично сравнимы между собой и определяются как суммы значений назначенных единичных категорий конфиденциальности, могут принимать значения от 0 до 0xFFFF FFFF FFFF FFFF (включительно), технически реализованы как 64-битная маска, беззнаковая величина (unsigned long long).

7 В пользовательских интерфейсах представляется шестнадцатеричным значением или списком наименований единичных категорий конфиденциальности.

Мандатный контекст безопасности

Метка уровня конфиденциальности (по умолчанию 4: от 0 до 3)

Уровни целостности (по умолчанию 0 и 63)

Категории (по умолчанию 2)

Специальные атрибуты (ccnr, ehole, whole)

```
# pdp-ls -M /var/log/auth.log
-rw-r--r----- 1 root adm Уровень_0:Низкий:Нет:0x0 /var/log/auth.log
# pdpl-user student
минимальная метка: Уровень_0:Низкий:Нет:0x0
0:0:0x0:0x0
максимальная метка: Уровень_0:Высокий:Нет:0x0
0:63:0x0:0x0
```

Вывод мандатного контекста безопасности. Команда pdp-ls

- *pdp-ls* — вывод мандатного контекста безопасности
- Синтаксис: *pdp-ls [параметры] [имя файла]*

```
sa@astra:/home/backups/basebackup$ pdp-ls -lM
итого 92
-rw----------   1 root root Уровень_0:Низкий:Нет:0x0 backup_label
drwx---------- 11 root root Уровень_0:Низкий:Нет:0x0 base
-rw----------   1 root root Уровень_0:Низкий:Нет:0x0 current_logfiles
drwx---------   2 root root Уровень_0:Низкий:Нет:0x0 global
-rw----------   1 root root Уровень_0:Низкий:Нет:0x0 pg_audit.conf
drwx---------   2 root root Уровень_0:Низкий:Нет:0x0 pg_commit_ts
```

1 Команда `pdp-ls` выводит мандатный контекст безопасности файла, является аналогом команды `ls` с расширенными функциями отображения меток конфиденциальности файлов и каталогов.

2 Использование данной утилиты аналогично использованию команды `ls`, за исключением следующих особенностей:

- если на файле установлены списки контроля доступа, то к списку прав доступа добавляется символ «+»;
- если на файле установлена не нулевая мандатная метка, то к списку прав доступа добавляется символ «m»;
- если на файле установлены роли, то к списку прав доступа добавляется символ «r»;
- если на файле установлены списки протоколирования, то к списку прав доступа добавляется символ «a».

Изменение мандатных атрибутов файлов

- Команда `pdpl-file` изменяет мандатные атрибуты файлов ОС, которые включают мандатную метку и специальные мандатные атрибуты файла.
- Синтаксис: `pdpl-file [ОПЦИИ]... [УРОВЕНЬ] [:УРОВЕНЬ_ЦЕЛОСТНОСТИ[:КАТЕГОРИЯ[:ФЛАГИ]]] ФАЙЛ...`
- Например,
  `pdpl-file 3:0:3:CCNRA logical_backups`

```
sa@astra:/home$ sudo pdpl-file 3:0:3:CCNRA logical_backups
[sudo] пароль для sa:
sa@astra:/home$ pdp-ls -M
итого 68
drwxr-xr-xm--   3 root       root       Уровень_3:Низкий:Категория_1,Категория_2:ccnr logical_backups
```

1 Синтаксис: `pdpl-file [ОПЦИИ]... [УРОВЕНЬ] [:УРОВЕНЬ_ЦЕЛОСТНОСТИ[:КАТЕГОРИЯ[:ФЛАГИ]]] ФАЙЛ...`

2 Ключи:

- `-f, --silent, --quiet` – не выводить сообщений об ошибках;
- `-v, --verbose` – выводить диагностические сообщения для каждого файла;
- `-c, --changes` – то же, что и `--verbose`, но сообщает только об изменениях;
- `-u, --unite` – объединить текущую метку файла с заданной в качестве аргумента;
- `-s, --subtract` – вычесть из текущей метки файла заданную в качестве аргумента;
- `-R, --recursive` – применить рекурсивно;
- `-r, --reverse` – сначала обрабатываются файлы в каталоге, затем – каталог;
- `-h, --help` – вывести эту справку и выйти;
- `--version` – вывести информацию о версии и выйти.

3 УРОВЕНЬ и УРОВЕНЬ_ЦЕЛОСТНОСТИ могут быть заданы именем или десятичным значением.

4 КАТЕГОРИЯ может быть задана именем или шестнадцатеричным значением.

5 ФЛАГИ могут быть заданы значением, или именами через запятую:

- для каталогов: ccnr, ccnri;
  Можно использовать псевдоним CCNRA (ccnr,ccnri);
- для файлов: ehole (с мин. меткой) ИЛИ whole (с макс. меткой).

Вывод мандатного контекста безопасности. Команда pdpl-user

- Команда `pdpl-user` отображает и устанавливает допустимые уровни конфиденциальности и категории пользователей ОС
- Синтаксис:
  `pdpl-user [-dzhv [-m минимальный:максимальный уровень конфиденциальности] [-i максимальный уровень целостности] [-c минимальная категория:максимальная категория]] <пользователь>`

```
sa@astra:~$ sudo pdpl-user sa
минимальная метка: Уровень_0:Низкий:Нет:0x0
   0:0:0x0:0x0
максимальная метка: Уровень_3:Высокий:Нет:0x0
   3:63:0x0:0x0
```

1 Основные опции команды `pdpl-user`:

- `-d, --delete` – удалить строку пользователя из файла;
- `-z, --zero` – обнулить значения уровней и категорий;
- `-l, --levels` – установить допустимые уровни конфиденциальности;
- `-i, --ilevel` – установить максимальный уровень целостности;
- `-m, --maclabels` – то же, что и `-l` (используется для совместимости);
- `-c, --category` – установить допустимые категории;
- `-h, --help` – вывести справку и выйти;
- `-v, --version` – вывести информацию о версии и выйти.

Изменение мандатного контекста безопасности. Команда pdpl-user

- Пример изменения мандатного контекста безопасности с использованием команды `pdpl-user`:

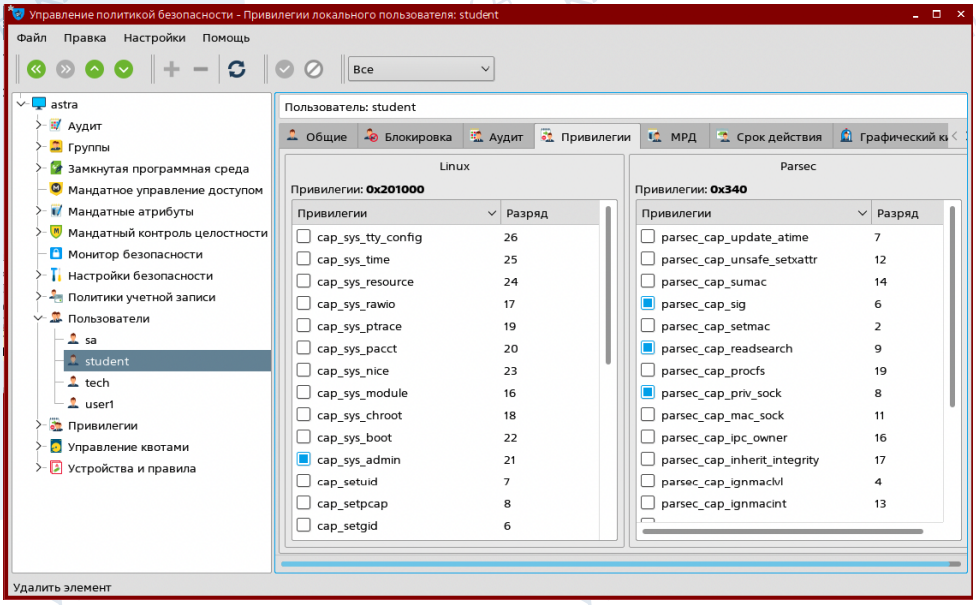
```
sudo pdpl-user -l 0:3 -c 0:1 ssidorov
```

```
sa@astra:~$ sudo pdpl-user -l 0:3 -c 0:1 ssidorov
минимальная метка: Уровень_0:Низкий:Нет:0x0
   0:0:0x0:0x0
максимальная метка: Уровень_3:Низкий:Категория_1:0x0
   3:0:0x1:0x0
```

1 Команда, показанная на слайде, для пользователя `ssidorov` установит:

- минимальный уровень — Уровень_0 (0);
- максимальный уровень — Уровень_3 (3);
- минимальную категорию — 0x0(0) (без категорий);
- максимальную категорию — 0x1(1) (Категория_1).

Привилегии



1 *Linux-привилегии* предназначены для передачи отдельным пользователям прав выполнения определенных административных действий и являются стандартными для системы Linux.

2 Linux-привилегии наследуются процессами от своих «родителей». Процессы, запущенные от имени суперпользователя, независимо от наличия у них привилегий, имеют возможность осуществлять все привилегированные действия.

3 Система привилегий ОС расширена привилегиями, относящимися к системе PARSEC. *PARSEC-привилегии* обеспечивают работу с механизмом мандатного управления доступом.

4 Все привилегии пользователя наследуются запущенными от имени его учетной записи процессами.

5 При запуске процесса с установленными привилегиями загрузчик динамических библиотек осуществляет сброс переменных среды окружения, позволяющих осуществлять загрузку динамических библиотек из нестандартных каталогов LD_LIBRARY_PATH и LD_PRELOAD.

6 Установка Linux-привилегий для пользователя может привести к невозможности запуска приложений, использующих динамическую загрузку библиотек из нестандартных каталогов (например, Firefox, Thunderbird, LibreOffice, fly-scan).

7 Для настройки КСЗ могут использоваться как Linux-, так и PARSEC-привилегии.

Средства управления привилегиями

- Графическая утилита для управления привилегиями пользователей и процессов: `fly-admin-smc`
- Инструмент командной строки: `usercaps`
- Управление привилегиями пользователей выполняется только суперпользователем с максимальным уровнем целостности, установленным в ОС

1 Команда `usercaps`:

- позволяет просматривать и устанавливать привилегии пользователей;
- должна использоваться только администратором через sudo;
- Синтаксис:

```
usercaps [-d, --delete] [-z, --zero] [-f, --full] [-l, --
linux] [-m, --parsec] [-l, --Linux] [-M, --PARSEC] [-n,
--numeric] [-h, --help] [-v, --version] пользователь
```

2 Опции команды `usercaps`:

- `-d, --delete` – удалить строку пользователя из файла привилегий;
- `-z, --zero` – сбросить все привилегии пользователя;
- `-f, --full` – присвоить все возможные привилегии пользователю;
- `-l, --linux` – изменить Linux-привилегии пользователя;
- `-m, --parsec` – изменить PARSEC-привилегии пользователя
- `-L, --Linux` – вывести список возможных Linux-привилегий;
- `-M, --PARSEC` – вывести список возможных PARSEC-привилегий
- `-n, --numeric` – вывести привилегии в шестнадцатеричном формате
- `-h, --help` – вывести справку и выйти;
- `-v, --version` – вывести информацию о версии и выйти.

3 Просмотр всех привилегий для пользователя student:

```
sa@astra:~$ sudo usercaps student
------------------
linux-привилегии:
12 cap_net_admin
21 cap_sys_admin
-----------------
PARSEC-привилегии:
 6 parsec cap sig
```

4 Просмотр списка возможных Linux-привилегий с сокращенными именами и числами:

```
usercaps -L
```

5 Просмотр списка возможных PARSEC-привилегий:

```
usercaps -M
```

6 Добавить Linux-привилегии под №12, 21:

```
sa@astra:~$ sudo usercaps -l +12:+21 student
-------------------
linux-привилегии:
12 cap_net_admin
21 cap_sys_admin
```

7 Добавить Linux-привилегии под №22, и удалить привилегию №21:

```
sa@astra:~$ sudo usercaps -l +22:-21 student
-------------------
linux-привилегии:
12 cap_net_admin
22 cap_sys_boot
```

8 Добавить PARSEC-привилегию под №6:

```
sa@astra:~$ sudo usercaps -m +6 student
-----------------
PARSEC-привилегии:
 6 parsec_cap_sig
```

9 Сбросить все привилегии можно с помощью команды `usercaps` с параметром `-z`:

```
sa@astra:~$ sudo usercaps -z student
-------------------
linux-привилегии:
 все флаги сброшены
-----------------
PARSEC-привилегии:
 все флаги сброшены
```

10 Установить максимальные привилегии:

```
sa@astra:~$ sudo usercaps -f student
------------------
linux-привилегии:
 0 cap_chown
 1 cap_dac_override
 2 cap_dac_read_search
 3 cap_fowner
 4 cap_fsetid
 5 cap_kill
 6 cap_setgid
 7 cap_setuid
 8 cap_setpcap
 9 cap_linux_immutable
```

Дополнительные мандатные атрибуты управления доступом

- Дополнительные мандатные атрибуты управления доступом позволяют уточнять или изменять правила мандатного управления доступом для тех или иных субъектов или сущностей:
  - `ccnr` — может присваиваться только контейнерам
  - `ehole` — может присваиваться сущностям (файлам), не являющимся контейнерами и имеющим минимальную классификационную метку и нулевую метку целостности
  - `whole` — присваивается сущностям (файлам), не являющимся контейнерами, с максимальной классификационной меткой

1 ***ccnr*** — определяет, что контейнер может содержать сущности с различными классификационными метками, но не большими, чем его собственная. Субъекту разрешается чтение содержимого контейнера вне зависимости от того, какую классификационную метку имеет субъект. Субъекту доступна информация только про сущности с классификационной меткой не большей, чем его собственная метка, или про сущности-контейнеры, которые имеют атрибут `ccnr`.

2 **`ehole`** — может присваиваться сущностям (файлам), не являющимся контейнерами и имеющим минимальную классификационную метку и нулевую метку целостности. Приводит к игнорированию мандатных правил управления доступом при получении доступа на запись к ним. Атрибут предназначен для сущностей, из которых субъект не может прочитать данные, записанные в них субъектами с более высокой классификационной меткой, чем его собственная (например, /dev/null).

3 **`whole`** — присваивается сущностям (файлам), не являющимся контейнерами, с максимальной классификационной меткой. Разрешает запись в них субъектам, имеющим более низкую классификационную метку (в обычном случае записывать «снизу вверх» запрещено).

Дополнительные мандатные атрибуты управления доступом. Файловая система

- *Каталог /home* – имеет наивысший мандатный уровень, ему присвоены все мандатные категории и атрибут CCNR
- Мандатные атрибуты домашних каталогов пользователей (/home/%username%) соответствуют мандатным атрибутам учётных записей пользователей, указанным при их регистрации
- Домашние каталоги пользователей подвергаются виртуализации. В каталоге */home/.pdp* создаются подкаталоги вида */home/.pdp/%username%*, в каждом из которых создаются подкаталоги вида: */home/.pdp/%username%/<level>:<int>:<category>:0*

  Например: */home/.pdp/iivanov/l2i0c0x0t0x0*

1 *Корневому каталогу /* назначается наивысший мандатный уровень, в битовой маске неиерархических категорий, устанавливаются все биты и атрибут CCNR. Данный набор настроек указывает, что в файловой системе могут храниться любые сущности с любыми мандатными метками, при этом мандатный уровень сущностей не может быть выше максимально возможного значения, назначенного /. Процесс с любым уровнем, даже минимальным, может обратиться внутрь корневого каталога /.

2 При инициализации сеанса работы пользователя все обращения к домашнему каталогу (т. е. имена файлов и каталогов, начинающиеся с */home/%username%*) преобразуются в обращения к тому подкаталогу каталога */home/.pdp/%username%*, который соответствует мандатным атрибутам текущего сеанса.

3 В результате пользователь во время каждого сеанса работы «видит» в своём домашнем каталоге только те файлы и подкаталоги, мандатные атрибуты которых соответствуют мандатным атрибутам текущего сеанса.

4 В записи */home/.pdp/%username% /<level>:< int>:< category>:0:*

- *<level>* — мандатный уровень сеанса работы пользователя;
- *<int>* — мандатный уровень целостности сеанса работы пользователя;
- *<category>* — числовое значение битовой маски неиерархических категорий, заданной для сеанса работы пользователя;
- *0* — зарезервирован для будущих версий и равен нулю.

Практическая работа

Оценка времени выполнения: 20 мин.

- Выполните задания модуля 6, часть 1 из Сборника практических заданий.

Мандатное управление доступом в СУБД PostgreSQL в Astra Linux SE

- СУБД PostgreSQL, входящая в состав дистрибутива Astra Linux SE, доработана в части мандатного управления доступом к информации
- PostgreSQL обеспечивает работу дискреционного, мандатного и ролевого управления доступом с учетом безопасности информационных потоков
- СУБД не имеет собственного механизма назначения, хранения и модификации меток пользователей, использует метки ОС
- Для работы с мандатным управлением доступа в PostgreSQL необходимо создавать одноименные ученые записи пользователей в ОС и роли (пользователей) в PostgreSQL

1 В основе механизма мандатного управления доступом лежит управление доступом к защищаемым ресурсам БД на основе иерархических и неиерархических меток доступа.

Это позволяет реализовать многоуровневую защиту с обеспечением разграничения доступа пользователей к защищаемым ресурсам БД и управление потоками информации.

2 СУБД не имеет собственного механизма назначения, хранения и модификации меток пользователей и использует для этого механизмы ОС.

В связи с этим, для работы мандатного управления доступом в PostgreSQL, необходимо создавать одноименные учетные записи пользователей в операционной системе и роли (пользователей) в PostgreSQL.

3 Мандатное управление доступом применяется ко всем объектам БД.

Мандатные атрибуты сеанса пользователя при подключении PostgreSQL в Astra Linux Special Edition

- В СУБД PostgreSQL каждому сеансу пользователя назначаются три атрибута, содержащие классификационные метки:
  - `ac_session_maclabel` — текущая классификационная метка сеанса пользователя
  - `ac_user_max_maclabel` — максимально возможное значение для ac_session_maclabel
  - `ac_user_min_maclabel` — минимально возможное значение для ac_session_maclabel

1 В ОС каждый пользователь может иметь множество классификационных меток, которое задается минимальной и максимальной классификационными метками диапазона.

2 Атрибут `ac_session_maclabel` — текущая классификационная метка сеанса пользователя СУБД.

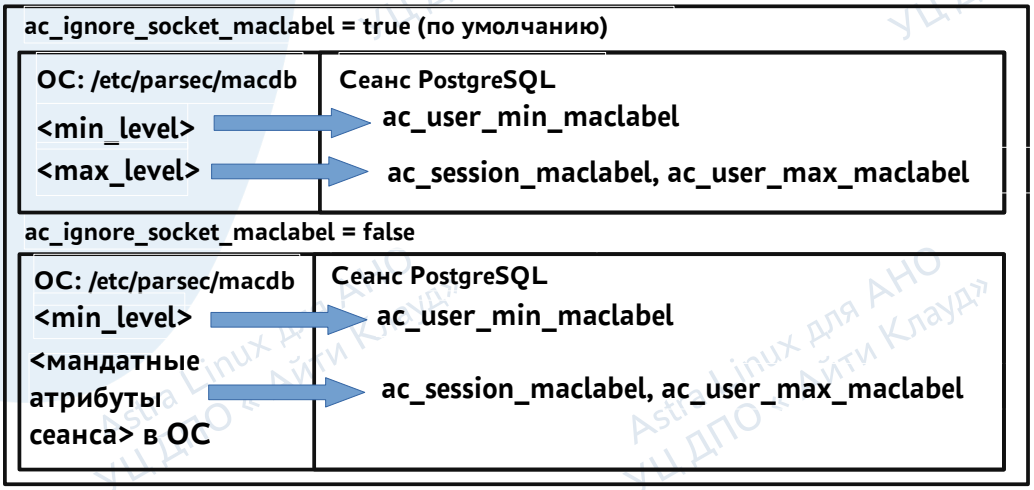
Эта классификационная метка определяет доступные пользователю объекты БД и является меткой безопасности по умолчанию для создаваемых пользователем объектов.

3 Если уровень конфиденциальности объекта выше уровня доступа пользователя — объект не будет доступен.

При соединении пользователя с СУБД значение этого атрибута устанавливается равным классификационной метке соединения или `ac_user_max_maclabel;`.

Формирование значений мандатных атрибутов сеанса

- Зависит от значения параметра ac_ignore_socket_maclabel (в postgresql.conf)



1 Для атрибута ac_user_min_maclabel устанавливается значение, равное *минимальной* классификационной метке пользователя в ОС (macdb).

2 Если параметр конфигурации *ac_ignore_socket_maclabel = true* (по умолчанию), то для атрибутов ac_user_max_maclabel и ac_session_maclabel устанавливаются значения, равные *максимальной* классификационной метке пользователя (уровень конфиденциальности равен максимальному уровню конфиденциальности, указанному в macdb).

3 Если параметр конфигурации *ac_ignore_socket_maclabel = false,* то классификационная метка сеанса в PostgreSQL (ac_session_maclabel) устанавливается *равной* классификационной метке входящего соединения.

4 Значение атрибута ac_user_max_maclabel (максимально возможное значение для ac_session_maclabel) также устанавливается *равным* классификационной метке входящего соединения.

Атрибуты пользователя (привилегии), связанные с мандатным контролем доступа

- Привилегии ОС, используемые в СУБД PostgreSQL, кроме атрибута ac_session_maclabel, не могут быть изменены с помощью средств СУБД ни пользователями, ни администраторами СУБД
- Администраторам СУБД дополнительно добавляются атрибуты:
  - *ac_capable_ignmaclvl* – позволяет пользователю игнорировать мандатное управление по уровням
  - *ac_capable_ignmaccat* – позволяет пользователю игнорировать мандатное управление по категориям
  - *ac_capable_chmac* – позволяет пользователю изменять метки объектов БД

1 Атрибут ac_capable_mac_readsearch позволяет пользователю *игнорировать* мандатное управление по уровням и категориям при чтении данных.

Просмотр и установка значений атрибута ac_session_maclabel

- Просмотр текущих значений привилегий (атрибутов пользователя): `SHOW attr_name;`
- *ac_capable_setmac* — позволяет пользователю изменять текущую классификационную метку своего сеанса в пределах, заданных ее минимальным и максимальным значением
- Установка нового значения атрибута `ac_session_maclabel`: `SET ac_session_maclabel=<новое_значение_классификационной_метки>;`

1 Привилегия `ac_capable_setmac` позволяет пользователю изменять текущую классификационную метку своего сеанса в пределах, заданных ее минимальным и максимальным значением, если в ОС для пользователя установлена привилегия parsec_cap_setmac.

Привилегия parsec_cap_setmac обычно не предоставляется рядовым пользователям.

2 Установить новое значение атрибута `ac_session_maclabel` можно с помощью команды:

- `SET ac_session_maclabel = <новое_значение_классификационной метки>;`

  В этой команде в качестве нового значения классификационной метки можно использовать только явно заданные значения метки.

Формат классификационной метки в PostgreSQL

- Классификационная метка указывается в формате `'{<уровень_конфиденциальности>, <категории>}'`
- `<уровень конфиденциальности>` – 8-битная беззнаковая величина, представляется десятичным числом
- `<категории>` – включает набор категорий конфиденциальности, 8-байтная беззнаковая величина

  Представляется шестнадцатеричным значением от 0 до 0xFFFF FFFF FFFF FFFF
- Например, '{2,0}' – уровень конфиденциальности 2, категория конфиденциальности отсутствует

Применение мандатных правил управления доступом

- Проверка мандатных прав доступа к объектам осуществляется одновременно с проверкой дискреционных прав доступа
- Доступ предоставляется только при одновременном разрешении доступа
- При превышении уровня конфиденциальности объекта уровня доступа пользователя — доступ будет запрещен. При этом может выводиться сообщение, что объекта не существует
- Применение мандатных правил осуществляется на уровне доступа к объектам БД и на уровне доступа непосредственно к данным (на уровне записей)

1 При обращении пользователя к БД определяются его допустимый диапазон меток и набор специальных мандатных атрибутов.

2 Для администратора БД предусмотрены системные привилегии игнорирования мандатного управления доступом для того, чтобы производить регламентные работы с БД (например, восстановление резервной копии).

Операции с данными в защищаемых объектах

- `INSERT` — доступ на запись
- `UPDATE, DELETE` — последовательное выполнение доступа на чтение и запись информации
- `SELECT` — доступ на чтение

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Работа с записями в таблицах

- Проверка мандатных прав доступа к записям таблиц осуществляется в процессе выполнения запроса при последовательном сканировании данных
- Все записи, помещаемые в таблицы, для которых установлена защита на уровне записей, наследуют текущую классификационную метку пользователя
- Обновляемые записи сохраняют свою классификационную метку при изменении
- Доступ к существующим записям и возможность их обновления и удаления определяются мандатными правилами

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Контейнеры

- *Сущности-контейнеры* — это сущности, которые могут содержать другие сущности
- Пример контейнеров в СУБД PostgreSQL: таблицы, схемы, табличные пространства, базы данных, кластер
- *Мандатный признак CCR* (Container Clearance Required) — используется для задания способа доступа к сущностям внутри контейнеров
- При установленном признаке CCR доступ к контейнеру и его содержимому определяется классификационной меткой контейнера
- Если признак CCR не установлен, доступ к содержимому разрешен без учета уровня конфиденциальности контейнера

1 *Системный каталог* (метаданные) рассматривается как самостоятельная БД, реализованная с помощью средств СУБД.

2 Все операции с системным каталогом осуществляются либо с помощью специальных конструкций языка запросов SQL, либо привилегированным пользователем в специальном режиме.

3 Метки безопасности системных объектов располагаются в записях таблиц системного каталога, непосредственно описывающих защищаемый объект.

4 Табличное пространство *pg_global* используется в качестве главного контейнера, создается одно на кластер базы данных.

5 При инициализации кластера автоматически создаются два табличных пространства: табличное пространство *pg_global* используется для общих системных каталогов, табличное пространство *pg_default* используется по умолчанию для баз данных template1 и template0.

Правила назначения меток безопасности

- Метка сущности не может превышать метку контейнера, в котором она содержится
- Нужно задать последовательно максимальные классификационные метки: кластера, базы данных, табличного пространства, схемы и таблицы
- Если метки безопасности определяются для таблиц или представлений, то они располагаются в записи системной таблицы, описывающей защищаемый объект
- Записи базы данных (кортежи) содержат в своем составе уровень конфиденциальности

1 Если метки безопасности определяются для таблиц или представлений, то они располагаются в записи системной таблицы, непосредственно описывающей защищаемый объект.

2 Записи базы данных (кортежи) содержат в своем составе уровень конфиденциальности.

Классификационная метка хранится в кортеже. Так классификационная метка будет неразрывно связана с данными, содержащимися в кортеже.

3 Кортежи рассматриваются в качестве объектов, а содержащие их таблицы — в качестве контейнеров.

Назначение меток безопасности

- При создании объекта БД его классификационная метка устанавливается равной текущей классификационной метке создавшего его пользователя
- Мандатный признак CCR при этом выставляется в значение ON
- Если пользователь имеет мандатный атрибут *ac_capable_chmac,* то он может менять классификационную метку принадлежащих ему объектов в пределах своего диапазона классификационных меток:

  ```
  MAC LABEL ON <тип_объекта имя_объекта> IS
  <новое_значение_классификационной_метки>;
  ```

- Изменение мандатного признака CCR для контейнеров:

  ```
  MAC CCR ON <тип_объекта> <имя_объекта> IS { ON | OFF };
  ```

1 Например, чтобы установить новое значение метки безопасности для таблицы mac_test_db в значение {3,0}, нужно выполнить запрос:

```
MAC LABEL ON DATABASE mac_test_db IS '{3,0}';
```

В конструкции **'{3,0}': 3** — это значение уровня конфиденциальности, **0** — категория конфиденциальности (не назначена).

2 В качестве типа объекта указывается тип объекта БД, например, DATABASE, TABLE, FUNCTION.

3 Для объектов-контейнеров может быть изменен мандатный признак CCR:

```
MAC CCR ON <тип_объекта> <имя_объекта> IS { ON | OFF };
```

Например, чтобы отключить (убрать) мандатный признак CCR с таблицы mac_test_db, нужно выполнить запрос:

```
MAC CCR ON DATABASE mac_test_db IS OFF;
```

Просмотр значений мандатного признака CCR и меток безопасности

- Значения классификационных меток объектов содержатся в полях *maclabel* таблиц системного каталога
- Значения мандатного признака CCR объектов-контейнеров содержатся:
  - для баз данных — в поле *datmacccr* системной таблицы pg_database
  - для схем — в поле *nspmacccr* системной таблицы pg_namespace
  - для табличных пространств — в поле *spcmacccr* системной таблицы pg_tablespace
  - для отношений — в поле *relmacccr* системной таблицы pg_class

Пример:

```
SELECT relname, relmacccr, maclabel FROM pg_class;
```

1 Для просмотра мандатного признака CCR кластера может быть использована следующая команда:

```
SELECT cluster_macccr;
```

2 Примеры запросов:

- `SELECT datname,datmacccr FROM pg_database;`
  – выводит список баз данных и их мандатный признак CCR.
- `SELECT relname, relmacccr, maclabel FROM pg_class;`
  – выводит список отношений (таблиц, индексов, последовательностей) и их мандатный признак CCR.

Установка классификационных меток на записи таблицы

- По умолчанию записи создаваемых таблиц не защищены классификационными метками
- Создание таблицы с записями, защищенными классификационными метками:

```
CREATE TABLE <имя_таблицы> (...--<список_столбцов>)
WITH (MACS = true, ... );
```

- Защита записей уже созданных таблиц без классификационных меток:

```
ALTER TABLE <имя_таблицы> SET WITH MACS;
```

1 Для того чтобы создать таблицы с записями, защищенными классификационными метками, следует использовать следующий вариант команды CREATE TABLE:

```
CREATE TABLE <имя_таблицы> (...--<список_столбцов>)
WITH (MACS = true, ... );
```

2 Для защиты записей уже созданных таблиц без классификационных меток следует использовать команду ALTER TABLE:

```
ALTER TABLE <имя_таблицы> SET WITH MACS;
```

После исполнения этой команды все записи таблицы автоматически получат текущую классификационную метку таблицы.

Установка мандатных меток записей

- Все вставляемые записи по умолчанию наследуют текущие классификационные метки создавших их пользователей
- Пользователи, имеющие установленный мандатный атрибут *ac_capable_chmac*, могут явно задать значение классификационной метки вставляемой записи
- Задаваемая классификационная метка должна быть в пределах диапазона классификационных меток пользователя:

```
INSERT INTO <имя_отношения>
(maclabel,...<список_столбцов>)
VALUES (<значение_классификационной_метки>, …
<значения_столбцов>);
```

1 Также задавать классификационную метку явным образом могут пользователи, имеющие привилегии игнорирования мандатного управления ac_capable_ignmaclvl и ac_capable_ignmaccat.

Просмотр классификационных меток записей в таблице

- Просмотреть значения классификационных меток доступных записей можно с помощью команды SELECT:

  ```
  SELECT maclabel FROM <имя_отношения>;
  ```

- *Совокупная* классификационная метка записей таблицы (максимальная по уровню конфиденциальности и наиболее полная по категориям конфиденциальности) может быть получена с помощью агрегирующей функции `supmaclabel`:

  ```
  SELECT supmaclabel (maclabel) FROM <имя_отношения>;
  ```

Целостность мандатных атрибутов кластера баз данных

- Классификационная метка объекта не может превышать классификационную метку контейнера, в котором он содержится
- Для вывода информации о соблюдении ограничений на выставление классификационных меток и признака CCR между объектами-контейнерами и находящимися в них объектами, реализована SQL-функция *check_mac_integrity*

```
SELECT check_mac_integrity();
```

- Если требование соблюдается для объекта и контейнера, то в результате проверки для этого объекта будет выведено значение **OK**

1 В СУБД PostgreSQL ДП-модель накладывает ограничение на классификационную метку объекта: классификационная метка объекта не может превышать классификационную метку контейнера, в котором он содержится.

Следует проверять данные на выполнение этого требования.

2 Для вывода информации о соблюдении ДП-модели между объектами-контейнерами и находящимися в них объектами реализована SQL-функция *check_mac_integrity*.

Пример работы функции:

```
SELECT check_mac_integrity();
```

```
mac_test_db=# SELECT check_mac_integrity();
    check_mac_integrity
---------------------------
 (24583,1259,24583,1259,OK)
 (24583,1259,2200,2615,OK)
 (24583,1259,1663,1213,OK)
 (2200,2615,16384,1262,OK)
 (16384,1262,1664,1213,OK)
(5 rows)
```

3 Функция check_mac_integrity выводит информацию в следующем виде:

- `objid` — идентификатор объекта;
- `classid` — идентификатор класса объекта;
- `cobjid` — идентификатор контейнера, содержащего объект;

- `cclassid` — идентификатор класса контейнера, содержащего объект;
- `status` — результат проверки. Может принимать следующие значения: OK (модель соблюдается для объекта и контейнера) и FAIL (модель не соблюдается для объекта и контейнера).

4 Информация о соблюдении модели выводится только для отношений (таблиц, представлений, последовательностей), схем, баз данных и табличных пространств.

Ссылочная целостность мандатных атрибутов

- Виды средств обеспечения целостности в реляционных БД:
  - для обеспечения целостности полей данных
  - декларативная ссылочная целостность
  - триггеры
- Событие изменения мандатных атрибутов *ON CHMAC:*

```
ALTER TABLE <имя_таблицы1> ADD CONSTRAINT
<имя_ограничения>
FOREIGN KEY (<список_полей1>) REFERENCES <имя_таблицы2>
(<список_полей2>)
ON CHMAC {NO ACTION | RESTRICT | CASCADE }
```

1 *Средства обеспечения целостности* в реляционных БД — это механизмы автоматической поддержки системы правил, которые определяют допустимость и корректность обрабатываемых данных и могут быть разбиты на несколько видов:

- обеспечение целостности полей данных:
  - задание разрешенных диапазонов значений,
  - запрет наличия неопределенных значений NULL,
  - задание значений по умолчанию,
  - уникальность каждой записи (для таблицы),
  - ограничение уникальности по выбранным столбцам,
  - вычисляемые значения столбцов и условия допустимых сочетаний значений в столбцах;
- *декларативная ссылочная целостность* (описание зависимостей между разными отношениями в БД – наличия в одном отношении внешнего, ссылающегося на первичный ключ в другом).

2 Подобные зависимости могут быть выявлены при процессе нормализации.

В этом случае одна сущность предметной области может состоять из набора отношений.

3 Ссылочная целостность реализуется путем описания ограничений на значения вторичных ключей в одном отношении и правил их обработки в случае изменения первичных ключей – в другом.

4 Существует частный случай ссылочной целостности, образованный между таблицами, являющимися частями одной сущности.

5 Для обеспечения целостности мандатных атрибутов список событий для ограничений целостности, наряду с существующими событиями ON SELECT, ON INSERT, ON UPDATE, ON DELETE, расширен событием изменения мандатных атрибутов ON CHMAC.

6 Механизм действия аналогичен механизму действия ограничения для события ON UPDATE для данных: при изменении мандатных атрибутов записи в одной таблице можно указать каскадное изменение мандатных атрибутов связанных записей второй таблицы, либо запрет на возможность такого изменения.

Целостность мандатных атрибутов: триггеры, правила (rules)

- Создание триггера на событие CHMAC:

```
CREATE TRIGGER <имя_триггера>
BEFORE CHMAC ON <имя_таблицы>
FOR EACH ROW
EXECUTE PROCEDURE <имя_процедуры>();
```

- Создание правил для представлений:

```
CREATE RULE <имя_правила>
AS ON CHMAC TO <имя_представления>
DO ALSO ...    -- возможно также DO INSTEAD
```

1 *Триггер* представляет собой исполняемый код, который может динамически проверить заданные условия корректности выполняемой операции и, при необходимости, внести изменения в другие таблицы.

2 Триггеры определяются на таблице (представлении) и назначаются для выполнения при реализации заданного вида доступа (например, при вставке, при изменении).

В приведенном определении отсутствуют мандатные атрибуты, так как в общем случае между классификационными метками записей разных таблиц может не быть зависимости.

3 Создание триггеров для указанного события изменения мандатных атрибутов (CHMAC):

```
CREATE TRIGGER <имя_триггера>
BEFORE CHMAC ON <имя_таблицы>
FOR EACH ROW
EXECUTE PROCEDURE <имя_процедуры>();
```

4 Создание правил для представлений:

```
CREATE RULE <имя_правила>
AS ON CHMAC TO <имя_представления>
DO ALSO ...
```

5 CREATE RULE <имя_правила>
AS ON CHMAC TO <имя_представления>
DO INSTEAD ...

6 В этом случае при изменении мандатных атрибутов записи будут вызываться соответствующие функции и правила.

Это дает возможность запрограммировать произвольную логику обработки события изменения и обеспечения целостности мандатных атрибутов внутри БД.

Параметры конфигурации кластера для настройки мандатного управления доступом

- Параметры указываются в файле `postgresql.conf`
- Важные параметры:
  - `ac_ignore_maclabel`
  - `ac_enable_sequence_ccr`
  - `ac_enable_dblink_mac`
  - `ac_auto_adjust_macs`

1 Для настройки работы СУБД с мандатным управлением доступом используются параметры конфигурации, которые влияют на его работу и поведение.

Параметры указываются в конфигурационном файле **postgresql.conf** кластера данных.

Параметр ac_ignore_maclabel

- По умолчанию имеет значение *false*
- Обеспечивает возможность работы с БД в режиме отключенного мандатного управления доступом
- Если `ac_ignore_maclabel = true`:
  - мандатные атрибуты пользователя не запрашиваются
  - классификационная метка сеанса устанавливается нулевой вне зависимости от значения ac_ignore_socket_maclabel
  - классификационная метка сервера также нулевая

Параметры ac_enable_sequence_ccr, ac_enable_dblink_mac, ac_auto_adjust_macs

- *ac_enable_sequence_ccr* (по умолчанию установлен в значение *true*)

  Если ac_enable_sequence_ccr = true, разрешается использование признака CCR для последовательностей аналогично таблицам и представлениям. Иначе признак CCR для последовательностей считается установленным

- *ac_enable_dblink_mac* (по умолчанию установлен в значение *false*)

  Если ac_enable_dblink_mac = true, то разрешается использование dblink и внешних таблиц (FOREIGN TABLE) в условиях МРД

- *ac_auto_adjust_macs* (по умолчанию установлен в значение *false*)

  Если ac_auto_adjust_macs = true, то разрешается автоматическая установка меток контейнеров. Применяется при восстановлении резервных копий

Вопросы для проверки

1 Какая утилита позволяет вывести информацию о файлах с отображением мандатных атрибутов?

2 Какой атрибут сеанса PostgreSQL содержит информацию текущей классификационной метке сеанса пользователя?

3 Какой мандатный признак используется для задания способа доступа к сущностям внутри контейнеров?

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Практическая работа

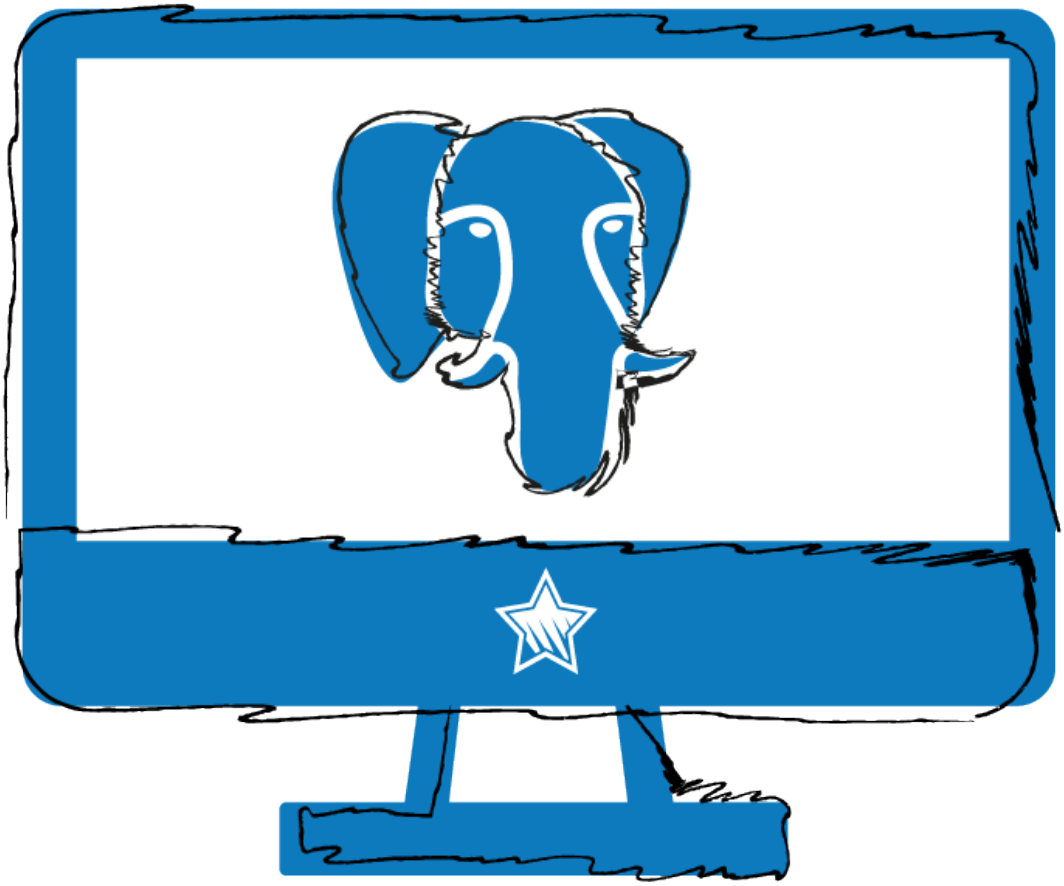
Оценка времени выполнения: 100 мин.

- Выполните задания модуля 6, часть 2 из Сборника практических заданий.

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Модуль 7

Журналирование и аудит доступа к данным



ASTRA LINUX®

Модуль 7.
Журналирование и аудит доступа к данным

Краткое содержание:

- Регистрация событий в СУБД PostgreSQL в журнале сообщений.
- Настройка журнала сообщений: размещение, формат, какие данные и когда вносить в журнал, ротация журналов.
- Анализ журнала сообщений.
- Работа с pgBadger для анализа журналов.
- Расширенные возможности аудита в СУБД PostgreSQL в дистрибутиве Astra Linux Special Edition.
- Генерация событий средствами PostgreSQL.

Основные знания и умения:

- настраивать журнал сообщений СУБД PostgreSQL;
- искать нужные данные в журнале сообщений;
- использовать pgBadger;
- настраивать расширенный аудит СУБД PostgreSQL в дистрибутиве Astra Linux Special Edition;
- искать нужные данные в журнале аудита;
- генерировать события средствами PostgreSQL.

Оценка знаний и умений:

- Практическая работа, контрольные вопросы.

Дополнительная литература:

- Отсутствует.

Цели модуля 7. Журналирование и аудит доступа к данным

- Регистрация событий в СУБД PostgreSQL в журнале сообщений
- Настройка журнала сообщений: размещение, формат, какие данные и когда вносить в журнал, ротация журналов
- Анализ журнала сообщений
- Работа с pgBadger для анализа журналов
- Расширенные возможности аудита в СУБД PostgreSQL в дистрибутиве Astra Linux Special Edition
- Генерация событий средствами PostgreSQL

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Регистрация событий в СУБД PostgreSQL в журнале сообщений

- Каталог с файлами журналов, где:

```
sa@astra:~$ sudo ls -l /var/lib/postgresql/11/main/pg_log
итого 4
-rw------- 1 postgres postgres 1270 мая 24 01:13 postgresql-Tue.log
```

  - 1 — версия СУБД PostgreSQL
  - 2 — название кластера
- Путь к журналам определяется настройкой *log_directory* в файле конфигурации postgresql.conf:

```
sa@astra:/etc/postgresql/11/main$ grep log_directory postgresql.conf
log_directory = 'pg_log'                # directory where log files are written,
```

1 Журналы СУБД PostgreSQL по умолчанию размещаются в каталоге `/var/lib/postgresql/{version}/{clustername}/pg_log/`,

где:

- `{version}` соответствует версии PostgreSQL. В Astra Linux Special Edition 1.7.4 это 11;
- `{clustername}` – имя кластера.

Например,

`sudo ls -l /var/lib/postgresql/11/main/pg_log/`

```
sa@astra:~$ sudo ls -l  /var/lib/postgresql/11/main/pg_log/
итого 736
-rw------- 1 postgres postgres  41410 окт 21 08:36 postgresql-Fri.log
-rw------- 1 postgres postgres   4970 окт 23 09:29 postgresql-Mon.log
-rw------- 1 postgres postgres 130924 окт 21 23:59 postgresql-Sat.log
-rw------- 1 postgres postgres 111470 окт 23 08:54 postgresql-Sun.log
-rw------- 1 postgres postgres  25938 окт 14 12:48 postgresql-Thu.log
-rw------- 1 postgres postgres 212862 окт 11 08:16 postgresql-Tue.log
-rw------- 1 postgres postgres 185054 окт 12 07:15 postgresql-Wed.log
```

2 Для того, чтобы события фиксировались в журнале, необходимо, чтобы параметр *logging_collector* был включен.

```
sa@astra:/etc/postgresql/11/main$ grep logging_collector postgresql.conf
                                  # requires logging_collector to be on.
logging_collector = on          # Enable capturing of stderr and csvlog
```

Параметры настройки журнала сообщений

- *log_destination* – указывает места, куда должны отправляться сообщения журнала, приемник сообщений:
  - возможные значения: stderr (по умолчанию), csvlog, syslog
  - может быть указано несколько мест (через запятую)
- *logging_collector* — включает сборщик сообщений (logging collector):
  - при значении *on* (по умолчанию), PostgreSQL будет использовать свои собственные процессы для перенаправления журналов в лог-файлы
- *log_directory (string)* – при включённом параметре logging_collector, определяет каталог, в котором создаются журнальные файлы

1 Возможные значения параметра log_destination:
- *stderr* (используется по умолчанию) – вывод в стандартный поток ошибок в текстовом виде;
- *csvlog* (Comma Separated Values Log) — данные журнала пишутся в CSV файлы. Это позволяет легко обрабатывать и анализировать данные журнала;
- *syslog* – направляет сообщения демону syslog.

2 Значение параметра log_destination можно задать только в конфигурационных файлах или в командной строке при запуске сервера.

3 Процесс logging collector - фоновый процесс, который собирает отправленные в stderr сообщения и перенаправляет их в файлы журналов.

4 Для того, чтобы события фиксировались в журнале, необходимо, чтобы параметр logging_collector в файле postgresql.conf был включен.

5 Такой подход более полезен, чем запись в syslog, поскольку некоторые сообщения в syslog могут не попасть (типичный пример с сообщениями об ошибках динамического связывания, другой пример — ошибки в скриптах типа archive_command.)

6 Для установки параметра требуется перезапуск сервера.

Данные в журнале: когда протоколировать?

- Определяется параметрами файла postgresql.conf
- Примеры параметров:
  - *log_min_messages* – управляет минимальным уровнем сообщений, записываемых в журнал (по умолчанию *warning*)
  - *log_min_error_statement* – управляет тем, какие SQL-операторы, завершившиеся ошибкой, записываются в журнал сервера (по умолчанию *error*)
  - *log_min_duration_statement* – записывает в журнал продолжительность выполнения всех команд, время работы которых не меньше указанного (по умолчанию *-1* – запись не ведется)

1 Через параметры файла *postgresql.conf* регулируется, какие данные будут заноситься в журнал.

По умолчанию большинство параметров отключено.

2 При включении дополнительных параметров нужно:
- обеспечить необходимое место на диске для хранения информации;
- оценить влияние записи журнала на общую производительность.

3 Параметр *log_min_error_statement* управляет тем, какие SQL-операторы, завершившиеся ошибкой, записываются в журнал сервера. SQL-оператор будет записан в журнал, если он завершится ошибкой с указанным уровнем важности или выше. По умолчанию имеет значение *error*.

4 С помощью параметра *log_min_duration_statement* можно выявить неоптимизированные запросы в приложениях.

Если значение этого параметра задаётся без единиц измерения, оно считается заданным в миллисекундах.

При нулевом значении записывается продолжительность выполнения всех команд.

5 С полным списком параметров можно ознакомиться по ссылке

https://www.postgresql.org/docs/11/runtime-config-logging.html#RUNTIME-CONFIG-LOGGING-WHEN

Данные в журнале: что протоколировать?

- Определяется параметрами
- Примеры параметров:
  - *log_checkpoints* – включает протоколирование выполнения контрольных точек и точек перезапуска сервера (по умолчанию выключен)
  - *log_connections* – включает протоколирование всех попыток подключения к серверу (по умолчанию выключен)
  - *log_duration* – записывает продолжительность каждой завершённой команды (по умолчанию выключен)

1 С полным списком параметров, регулирующих, какие события будут протоколироваться можно ознакомиться по ссылке: https://www.postgresql.org/docs/11/runtime-config-logging.html#RUNTIME-CONFIG-LOGGING-WHAT

Примечание:

Включение параметра *log_duration* не равнозначно установке нулевого значения для *log_min_duration_statement*.

Разница состоит в том, что при превышении значения *log_min_duration_statement* в журнал записывается текст запроса, а при включении данного параметра — нет.

Таким образом, при *log_duration* = *on* и положительном значении *log_min_duration_statement*, в журнал записывается длительность для всех команд, а текст запроса — только для команд с длительностью, превышающей предел.

Такая настойка может быть полезной при сборе статистики в условиях большой нагрузки.

Данные в журнале: содержание записи о событии

- Параметр *log_line_prefix* определяет содержание записи, которая выводится в начале каждой строки журнала сообщений
- По умолчанию этот параметр имеет значение `'%m [%p] %q%u@%d '`
- Примеры спецсимволов:
  - `%m` – штамп времени с миллисекундами
  - `%p` – идентификатор процесса
  - `%a` - имя приложения
  - `%u` – имя пользователя
  - `%r` – имя удалённого узла или IP-адрес, а также номер порта

1 Другие значения спецсимволов:
- `%q` – ничего не выводит. Непользовательские процессы останавливаются в этой точке. Игнорируется пользовательскими процессами;
- `%d` – имя БД;
- `%t` – штамп времени без миллисекунд.

2 С полным списком спецсимволов для параметра log_line_prefix можно ознакомиться по ссылке https://www.postgresql.org/docs/11/runtime-config-logging.html#RUNTIME-CONFIG-LOGGING-WHAT

3 Параметр log_error_verbosity управляет количеством детальной информации, записываемой в журнал сервера для каждого сообщения.

Допустимые значения: TERSE, DEFAULT и VERBOSE. Каждое последующее значение добавляет больше полей в выводимое сообщение.

4 Для TERSE из сообщения об ошибке исключаются поля DETAIL, HINT, QUERY и CONTEXT.

5 Для VERBOSE в сообщение включается код ошибки SQLSTATE, а также имя файла с исходным кодом, имя функции и номер строки сгенерировавшей ошибку.

6 По умолчанию параметр log_error_verbosity имеет значение DEFAULT.

7 Если включить параметр log_hostname, то в журнале будет фиксироваться имя сервера.

8 По умолчанию, сообщения журнала содержат только IP-адрес подключившегося клиента.

Примечание:

В зависимости от применяемого способа разрешения имён, включение параметра может отрицательно сказаться на производительности.

9 Задать параметр log_hostname можно только в postgresql.conf или в командной строке при запуске сервера.

Параметры для настройки ротации файлов журнала

- *log_filename* – задает имя журнального файла, может включать специальные символы для автоматического именования журналов в зависимости от даты и времени
- *log_rotation_age* – задает время переключения на следующий файл (по умолчанию 1 день)
- *log_rotation_size* — задает размер файла в килобайтах, при котором надо переключаться на следующий (по умолчанию 0)
- *log_truncate_on_rotation* – включение параметра позволяет перезаписывать уже существующие файлы (по умолчанию включено)

1 Коллектор сообщений имеет встроенные средства ротации, которые настраиваются с помощью параметров.

2 Параметры log_rotation_age и log_rotation_size определяют, когда следует начинать новый файл журнала, на основе возраста или размера текущего файла соответственно.

3 Для запрета создания нового файла по прошествии определённого времени, нужно установить значение параметра log_rotation_age в **0**.

4 Для запрета создания нового файла при превышении определённого размера, нужно установить значение параметра log_rotation_size в **0**.

5 Комбинируя маску для имени файла и время (размер) переключения, можно получать разные схемы ротации журналов.

6 Подробнее параметрах настройки журнала сообщений:

https://www.postgresql.org/docs/11/runtime-config-logging.html

Примеры сочетаний значений параметров

- Формируется 24 файла в течение суток, файлы перезаписываются:

```
log_filename = 'postgresql-%H.log'
log_rotation_age = '1h'
log_truncate_on_rotation = on
```

- Формируется 7 файлов в течение недели, файлы перезаписываются:

```
log_filename = 'postgresql-%a.log'
log_rotation_age = '1d'
log_truncate_on_rotation = on
```

1 Значение параметра log_filename трактуется как строка формата в функции strftime, поэтому в ней можно использовать спецификаторы % для включения в имена файлов информации о дате и времени.

2 С поддерживаемыми спецификациями можно познакомиться на сайте: https://pubs.opengroup.org/onlinepubs/009695399/functions/strftime.html

3 Например,

- %a – сокращенное названием дня недели в локали;
- %Y – год в виде десятичного числа (например, 2023);
- %m – месяц в виде десятичного числа [01,12];
- %d – день месяца в виде десятичного числа [01,31];
- %H – часы (в 24-часовом формате) в виде десятичного числа [00,23];
- %M – минуты в виде десятичного числа [00,59];
- %S – секунды в виде десятичного числа [00,59].

4 При наличии зависящих от часового пояса спецификаторов % будет использован пояс, заданный в параметре log_timezone.

5 Значение параметра log_filename по умолчанию *postgresql-%Y-%m-%d_%H%M%S.log*.

Анализ журнала сообщений PostgreSQL

- Анализ можно выполнять с помощью:
  - средств операционной системы (grep, awk и др.)
  - скриптов
  - специальных средства для анализа (pgBadger и др.)
- *awk* – мощный инструмент для обработки и фильтрации текста
- Поиск записи по вхождению слова:

```
sudo grep iivanov main/pg_log/postgresql-<день_недели>.log
```

– поиск в журнале информации, касающейся работы пользователя с данными кластера БД пользователя *iivanov*

1 Утилита awk читает документ по одной строке за раз, выполняет указанные действия и выводит результат на стандартный вывод.

2 Все параметры awk находятся в кавычках, а действие, которое надо выполнить – в фигурных скобках.

3 Синтаксис awk:

```
$ awk опции 'условие {действие}'
$ awk опции 'условие {действие} условие {действие}'
```

4 С помощью действия можно выполнять преобразования с обрабатываемой строкой.

5 Опции утилиты awk:

- `-F, --field-separator` – разделитель полей, используется для разбиения текста на колонки;
- `-f, --file` – прочитать данные не из стандартного вывода, а из файла;
- `-v, --assign` – присвоить значение переменной, например foo=bar;
- `-b, --characters-as-bytes` – считать все символы однобайтовыми;
- `-d, --dump-variables` – вывести значения всех переменных awk по умолчанию;

- `-D, --debug` – режим отладки, позволяет вводить команды интерактивно с клавиатуры;
- `-e, --source` – выполнить указанный код на языке awk;
- `-o, --pretty-print` – вывести результат работы программы в файл;
- `-V, --version` – вывести версию утилиты.

6 Функции-действия awk:

- `print(строка)` – вывод чего-либо в стандартный поток вывода;
- `printf(строка)` – форматированный вывод в стандартный поток вывода;
- `system(команда)` – выполняет команду в системе;
- `length(строка)` – возвращает длину строки;
- `substr(строка, старт, количество)` – обрезает строку и возвращает результат;
- `tolower(строка)` – переводит строку в нижний регистр;
- `toupper(строка)` – переводить строку в верхний регистр.

7 Переменные и операторы awk:

- `FNR` – номер обрабатываемой строки в файле;
- `FS` – разделитель полей;
- `NF` – количество колонок в данной строке;
- `NR` – общее количество строк в обрабатываемом тексте;
- `RS` – разделитель строк, по умолчанию символ новой строки;
- `$` – ссылка на колонку по номеру.

Утилита pgBadger

- *pgBadger*:
  - это инструмент для быстрого анализа журналов PostgresSQL
  - позволяет создавать подробные отчёты и графики
- В качестве файла журнала можно передать:
  - отдельный файл
  - список файлов
  - команду, выдающую список файлов
- Может обрабатывать файлы журналов больших размеров и сжатые файлы
- В журнале допускаются сообщения только на английском языке

1 Утилита pgBadger — авторский продукт Жиля Дарольда (Gilles Darold).
Найти пакет для установки можно по ссылке: https://pgbadger.darold.net/

2 Для изменения языка журнала сообщений в PostgreSQL нужно изменить значение параметра lc_messages в файле конфигурации postgresql.conf.
Нужно установить значение lc_messages 'en_US.UTF-8'.

3 Анализатор запросов pgbadger может обрабатывать огромные файлы журналов и сжатые файлы.

4 Он может автоматически определять формат файла журнала (syslog, stderr, csvlog), если файл достаточно большой.

5 Поддерживаемые форматы сжатия: gzip, bzip2, lz4, xz, zip и zstd.

6 Утилита pgBadger поддерживает любой формат префиксов строк журнала, который можно задать в параметре log_line_prefix в файле конфигурации postgresql.conf, при условии, что указаны как минимум %t и %p.

Возможности утилиты pgBadger

- Формирует отчёты в html-формате
- Формирование отчётов:
  - о запросах по типам, базам данных и пользователям
  - о запросах, например, самых медленных и наиболее частых, по пользователям, базам данных и т. д.
  - о сеансах
  - о блокировках
  - о контрольных точках
  - об автоочистке

1 С примерами отчетов можно ознакомиться по ссылке: https://pgbadger.darold.net/#reports

2 Если в формировании какого либо отчета нет необходимости, можно отключить создание отчета через параметр при запуске pgBadger.

Например,

- `--disable-type` – отключает формирование отчёта о запросах по типам, базам данных и пользователям;
- `--disable-query` – отключает формирование отчётов о запросах, например: самых медленных и наиболее частых, по пользователям, базам данных и т. д.;
- `--disable-session` – отключает формирование отчёта о сеансах;
- `--disable-connection` – отключает формирование отчёта о соединениях;
- `--disable-lock` – отключает формирование отчёта о блокировках;
- `--disable-checkpoint` – отключает формирование отчётов о контрольных точках/точках перезапуска.
- `--disable-autovacuum` – отключает формирование отчёта об автоочистке.

Синтаксис pgBadger

- pgbadger [параметр_подключения…] [параметр…] [файл_журнала…]
- Позволяет подключаться к журналу удаленно

  Параметры указываются в блоке **Параметры подключения**
- Примеры параметров:
  - `-a минуты (--average минуты)` – указывает длительность интервала (в минутах) для построения графика усреднённых значений для запросов и подключений. По умолчанию: 5.
  - `-b дата_время (--begin дата_время)` – указывает дату/время начала диапазона анализируемых данных в журналах

1 С полным списком параметров можно ознакомиться по ссылке: https://pgbadger.darold.net/documentation.html#INSTALLATION

2 Утилита pgbadger может анализировать удалённые файлы журналов, получаемые через SSH-соединение без пароля.

3 IP-адрес или имя целевого компьютера задаётся в параметре `-r/--remote-host`.

4 Другие свойства SSH-подключения определяются параметрами:
  - `--ssh-program ssh` – указывает путь к используемому SSH-клиенту. По умолчанию – **ssh**.
  - `--ssh-port` *порт* – указывает порт SSH для подключения. По умолчанию: 22.
  - `--ssh-user` *имя_пользователя* – указывает имя пользователя для подключения. По умолчанию используется имя пользователя, запускающего pgbadger.
  - `--ssh-identity имя_файла` – указывает путь к файлу идентификации.
  - `--ssh-timeout секунды` – задаёт тайм-аут в секундах на случай сбоя SSH-соединения. По умолчанию: 10.
  - `--ssh-option` *параметры* – задаёт список параметров для SSH-соединения.

- Следующие параметры используются **всегда**:

```
-o ConnectTimeout=$ssh_timeout -o
PreferredAuthentications=hostbased,publickey
```

5 Примеры параметров pgbadger:

- `-A минуты (--histo-average минуты)` – указывает длительность интервала (в минутах) для построения гистограмм для запросов. По умолчанию: 60;
- `-c хост (--dbclient хост)` — обрабатывает только записи, относящиеся к указанному клиентскому компьютеру;
- `-C (--nocomment)` – удаляет из запросов комментарии /* ... */;
- `-d имя (--dbname имя)` – обрабатывает только записи, относящиеся к указанной базе данных;
- `-D (--dns-resolv)` – заменяет IP-адреса клиентов их DNS-именами;
- `-e дата_время (--end дата_время)` – указывает дату/время конца диапазона анализируемых данных в журналах;
- `-E (--explode)` – строит отдельные отчёты для разных БД. Глобальная информация, не связанная с какой-либо базой данных, добавляется в отчёт по базе данных postgres;
- `-f тип_журнала (--format тип_журнала)` – указывает тип журнала.

  Возможные значения: syslog, syslog2, stderr, jsonlog, csv, pgbouncer, logplex, rds и redshift. Следует использовать, когда pgbadger не может определить формат журнала;
- `-G (--nograph)` – отключает графики в выводе HTML;
- `--help` – выводит подробную информацию о параметрах pbadger и завершиться;
- `-H путь (--html-outdir путь)` – указывает путь к каталогу, в который будет записываться HTML-отчёт в инкрементальном режиме.
- `-p строка (--prefix строка)` – указывает значение настраиваемого параметра log_line_prefix, заданное в файле postgresql.conf. Полезен в случаях, когда вы используете префикс строк, отличный от стандартных префиксов log_line_prefix, например, когда префикс включает дополнительные переменные, такие как IP-адрес клиента или имя приложения.

Примеры вызова pgBudger

- `pgbadger /var/lib/postgresql/11/main/pg_log/postgresql-Mon.log`
  – анализ журнала без дополнительных параметров
- `pgbadger /var/log/postgres.log.2.gz /var/log/postgres.log.1.gz /var/log/postgres.log`
  – анализ списка журналов
- `pgbadger --exclude-query="^(COPY|COMMIT)" /var/log/postgresql.log`
  – не учитывать в анализе запросы COPY, COMMIT
- `pgbadger -b "2012-06-25 10:56:11" -e "2012-06-25 10:59:11" /var/log/postgresql.log`
  – анализ журналов с указанием времени начала и завершения периода

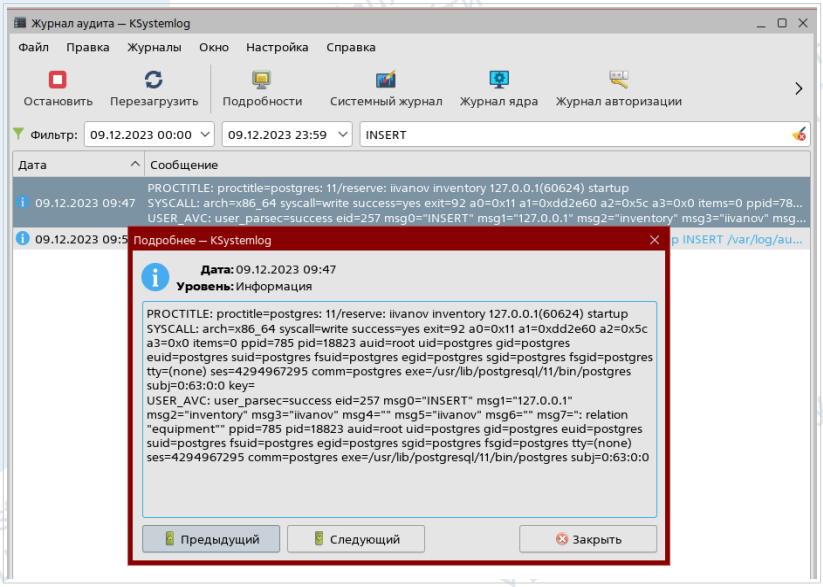
- При регистрации события фиксируются следующие параметры:
  - дата и время
  - субъект (имя или идентификатор)
  - объект
  - тип доступа
  - результат запроса на доступ
- Вывод сообщений производится в специальный журнал безопасности операционной системы */var/log/audit/audit.log*

1 Служба *auditd* регистрирует события безопасности в журнале аудита /var/log/audit.log.

```
sa@astra:~$ sudo grep INSERT /var/log/audit/audit.log
type=USER_AVC msg=audit(1702104472.086:43714): user_parsec=success eid=257 msg0="INSERT" msg1="127.0.0.1"
 msg2="inventory" msg3="iivanov" msg4="" msg5="iivanov" msg6="" msg7=": relation "equipment"" ppid=785 pi
d=18823 auid=4294967295 uid=115 gid=127 euid=115 suid=115 fsuid=115 egid=127 sgid=127 fsgid=127 tty=(none
) ses=4294967295 comm="postgres" exe="/usr/lib/postgresql/11/bin/postgres" subj=0:63:0:0
```

2 Для просмотра журнала аудита может быть используется графическая утилита *ksystemlog* («Системный журнал»).

Просмотр записей журнала через графическую утилиту ksystemlog («Системный журнал»)



1 Данные журнала размещены во вкладке **Журнал аудита**.

Просмотр записей журнала через команду ausearch

- Утилита `ausearch` - инструмент поиска по журналу аудита
- Просмотр записей журнала через терминал:
  ```
  sudo -x postgres -i | more
  ```
- Настройка генерации сообщений аудита PostgreSQL в интерактивном режиме:
  ```
  sudo watch -n 1 'ausearch -x postgres -i | tail'
  ```
  - выводит сообщения из журнала в терминал с интервалом равным одной секунде

1 Некоторые параметры утилиты ausearch:

- `-i,--interpret` – представлять результаты в удобном для человека формате;
- `-x, --executable программа` – искать события с заданным именем исполняемой программы;
- `-m, --message тип_сообщения | список_типов_сообщений` – искать события с заданным типом, при этом можно указать список значений, разделенных запятыми или несколько отдельных типов сообщений, каждый со своей опцией `-m`.
- `-n, --node` – искать события с определенного узла.
- `-ui, --uid идентификатор_пользователя` – искать события с заданным идентификатором пользователя.

2 Все условия, указанные в параметрах, объединяются логическим И.

Например, при указании `-m` и `-ui` в качестве параметров будут показаны события, соответствующие заданному типу и идентификатору пользователя. Исключением являются параметры `-m` и `-n` (в строке поиска допускается указание несколько типов записей и имен узлов, которые будут возвращать любой соответствующий узел и запись).

3 Детальное описание всех параметров утилиты можно получить через выполнение `man ausearch`.

Сообщение аудита, выданное СУБД PostgreSQL

```
PROCTITLE: proctitle=postgres: 11/reserve: iivanov inventory 127.0.0.1(60624)
startup
SYSCALL: arch=x86_64 syscall=write success=yes exit=92 a0=0x11 a1=0xdd2e60
a2=0x5c a3=0x0 items=0 ppid=785 pid=18823 auid=root uid=postgres gid=postgres
euid=postgres suid=postgres fsuid=postgres egid=postgres sgid=postgres
fsgid=postgres tty=(none) ses=4294967295 comm=postgres
exe=/usr/lib/postgresql/11/bin/postgres subj=0:63:0:0 key=
USER_AVC: user_parsec=success eid=257 msg0="INSERT" msg1="127.0.0.1"
msg2="inventory" msg3="iivanov" msg4="" msg5="iivanov" msg6="" msg7=":
relation "equipment"" ppid=785 pid=18823 auid=root uid=postgres gid=postgres
euid=postgres suid=postgres fsuid=postgres egid=postgres sgid=postgres
fsgid=postgres tty=(none) ses=4294967295 comm=postgres
exe=/usr/lib/postgresql/11/bin/postgres subj=0:63:0:0
```

1 Из записи можно получить следующую информацию:

- успешность осуществления события (`success=yes` или `success=no`);
- тип события (CONNECT, DISCONNECT, SUBJECT, RIGHTS и т. д.);
- хост, с которого отправлен клиентский запрос (в приведенном примере – 127.0.0.1);
- имя кластера, с которым работают (reserve);
- имя БД, с которой работают (inventory);
- имя авторизованного пользователя (iivanov).

2 Описание остальных полей в записи:

- `type` — тип записи;
- `msg=audit` — запись времени события и его уникальный идентификационный номер. Сообщения обычно начинаются с записи вида `msg=audit(1116360555.329:2401771)`. Идентификатор события — число после «:». Все события аудита, связанные с одним системным вызовом, имеют одинаковый идентификатор;
- `arch` — запись об архитектуре процессора;
- `syscall` — тип систем вызова;
- `success` — результат обработки вызова (успешно или нет);
- `exit` — значение выполнения, возвращенное системным вызовом;

- `a0, a1, a2, a3` — четыре аргумента, закодированные в шестнадцатеричный формат, зависят от системного вызова;
- `ppid` — идентификационный номер родительского процесса;
- `pid` — идентификационный номер процесса;
- `auid` — идентификационный номер пользователя аудита;
- `uid` — имя пользователя, который вызвал процесс;
- `gid` — группа пользователя, который вызвал процесс;
- `euid` — имя действующего пользователя, который вызвал процесс;
- `suid` — имя пользователя, установленного во время выполнения;
- `fsuid` — имя пользователя файловой системы;
- `egid` — имя действующей группы пользователя, который вызвал процесс;
- `sgid` — имя группы пользователя, установленного во время выполнения;
- `fsgid` — имя группы пользователя файловой системы;
- `tty` — номер терминала, с которого вызван анализируемый процесс;
- `ses` — идентификационный номер сессии, в которой вызван анализируемый процесс;
- `comm` — название команды, из которой был вызван процесс;
- `exe` — путь до исполняемого файла, который вызвал анализируемый процесс;
- `subj` — контекст безопасности анализируемого процесса.

Режимы регистрации событий

- Настройка — параметр `ac_audit_mode` в файле postgresql.conf
- Возможные значения параметра ac_audit_mode:
  - `internal` — настройки регистрации событий хранятся в таблице pg_db_role_settings, настройка через команды SQL (по умолчанию)
  - `external` — настройка регистрации событий для через файл pg_audit.conf
  - `external, internal` — смешанный режим
  - `internal, external` — смешанный режим
  - `none` — регистрация событий отключена

1 В СУБД PostgreSQL для настройки режима регистрации событий используется конфигурационный параметр `ac_audit_mode` файла postgresql.conf.

```
sa@astra:/etc/postgresql/11/main$ grep ac_audit_mode postgresql.conf
ac_audit_mode           = 'internal'            # rbt audit mode. Possible values are
```

Этот параметр может быть изменен только перезапуском сервера.

2 При использовании смешанного режима external, internal, настройки регистрации событий берутся сначала из внешнего файла pg_audit.conf, после чего дополняются настройками из таблицы pg_db_role_settings;

3 При использовании смешанного режима internal, external настройки регистрации событий берутся сначала из таблицы pg_db_role_settings, после чего дополняются настройками из внешнего файла pg_audit.conf.

4 По умолчанию файл pg_audit.conf размещается в каталоге */var/lib/postgresql/11/имя_кластера/*.

Настройка маски регистрации событий

- Какие события будут регистрироваться определяется через маску
- Маска определяется как значение атрибута ac_session_audit
- Маска регистрации событий имеет вид
  `{УСПЕХ:ОТКАЗ}`,
  где УСПЕХ — список успешных событий, ОТКАЗ — список неуспешных событий
- Маска может быть задана с помощью буквенных кодов или с помощью шестнадцатеричного числа. Маски выводится в текстовом виде
- Маска по умолчанию для роли postgres имеет значение
  `{SsRawdCTEMce:ce}`

1 В СУБД PostgreSQL версии 11 маска регистрации событий устанавливается в процессе авторизации пользователя, согласно выбранному режиму регистрации событий и находится в атрибуте сессии ac_session_audit.

2 При этом реализован следующий порядок применения настроек регистрации событий:
   - настройки для конкретной роли и конкретной базы данных;
   - настройки для конкретной роли;
   - настройки для конкретной базы данных;
   - для всех остальных.

3 Атрибут сессии ac_session_audit может быть изменен только администратором с помощью команды SET:
   `SET ac_session_audit TO 'новое_значение';`

4 Просмотреть значение атрибута сессии ac_session_audit можно с помощью команды:
   `SHOW ac_session_audit;`

```
postgres=# SHOW ac_session_audit;
 ac_session_audit
------------------
 {SsRawdCTEMce:ce}
(1 строка)
```

5 Для просмотра маски сессии используется команда:

```
SELECT session_audit;
```

```
postgres=# SELECT session_audit;
   session_audit
-------------------
 {SsRawdCTEMce:ce}
(1 строка)
```

6 Для просмотра текущей маски используется следующая команда:

```
SELECT current_audit;
```

```
postgres=# SELECT current_audit;
   current_audit
-------------------
 {SsRawdCTEMce:ce}
(1 строка)
```

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Примеры значений символов маски

- S (SUBJECT) – добавление/изменение/удаление пользователей и групп
- s (CONFIGURATION) – изменение конфигурации, влияющей на доступ к данным (запрос на изменение значения переменной ac_session_maclabel
- R (RIGHTS) – запрос на модификацию прав доступа к объектам БД
- a (INSERT) – добавление информации в БД
- w (UPDATE) – изменение информации в БД
- d (DELETE) – удаление информации из БД
- c (CONNECT) – запрос на начало сеанса
- e (DISCONNECT) – запрос на окончание сеанса

1 Для конвертации маски регистрируемых событий из текстового (буквенного) в шестнадцатеричное значение и из шестнадцатеричного в текстовое (буквенное) используются SQL-функции *text_to_auditmask(TEXT)* и *auditmask_to_text(TEXT)* соответственно.

2 Например,

- `SELECT text_to_auditmask('{ace:ce}');`

```
inventory=# SELECT text_to_auditmask('{ace:ce}');
    text_to_auditmask
-------------------------
 {0xC0000020:0xC0000000}
(1 строка)
```

- `SELECT auditmask_to_text('{0xC0000020:0xC0000000}');`

```
inventory=# SELECT auditmask_to_text('{0xC0000020:0xC0000000}');
 auditmask_to_text
-------------------
 {ace:ce}
(1 строка)
```

3 Полный список событий и их соответствие буквенным и шестнадцатеричным значениям маски регистрации событий приведен в таблице ниже:

Событие	Символ	Шестнадца-теричное	Описание значения
SUBJECT	S	1	Добавление/изменение/удаление пользователей и групп
CONFIGURATION	s	2	Изменение конфигурации, влияющей на доступ к данным (запрос на изменение значения переменной ac_session_maclabel
RIGHTS	R	4	Запрос на модификацию прав доступа к объектам БД
CHECK_RIGHTS	V	8	Проверка прав доступа
SELECT	r	10	Выборка информации из БД
INSERT	a	20	Добавление информации в БД
UPDATE	w	40	Изменение информации в БД
DELETE	d	80	Удаление информации из БД
TRUNCATE	D	100	Очистка данных
REFERENCES	x	400	Задание столбца таблицы в качестве внешнего ключа
TRIGGER	t	800	Добавление триггера к таблице
EXECUTE	X	1000	Запуск хранимой процедуры или триггера
USAGE	U	2000	Использование объекта БД
CREATE	C	10000	Создание объектов в БД
CREATE TEMP	T	20000	Создание временного объекта
DROP	E	40000	Удаление объектов БД
ALTER	M	80000	Изменение объекта БД
CONNECT	c	40000000	Запрос на начало сеанса
DISCONNECT	e	80000000	Запрос на окончание сеанса
Зарезервированный символ	*	C00F3DFF	Полная маска регистрации событий
Зарезервированный символ	O	0	Регистрация событий отключена

Назначение списков регистрации событий в режиме internal

- Назначения маски событий:

```
ALTER ROLE { ALL | имя_роли } [ IN DATABASE
имя_базы_данных ]
SET ac_session_audit TO новое_значение;
```

- Удаление списка регистрации событий:

```
ALTER ROLE { ALL | имя_роли } [ IN DATABASE
имя_базы_данных ] RESET ac_session_audit;
```

- Для выполнения приведенных команд требуются права администратора
- Маска по умолчанию для всех ролей (кроме postgres) имеет значение `{SsRDxCTEMce:SsRVrawdDxtXUCTEMce}`

1 При модификации маски происходит автоматическое обновление атрибута ac_session_audit.

Примечание:

При инициализации кластера баз данных автоматически добавляются следующие правила:

```
ALTER ROLE postgres SET ac_session_audit TO
'{SsRawdCTEMce:ce}';
ALTER ROLE ALL SET ac_session_audit TO
'{SsRDxCTEMce:SsRVrawdDxtXUCTEMce}';
```

Назначение списков регистрации событий в режиме external

- В файле pg_audit.conf маска указывается в шестнадцатеричном формате
- Пример заполнения файла pg_audit.conf:
  - `success events mask = F00E7 failure events mask = 0 user = postgres`

    – аудит действий администратора СУБД
  - `success events mask = 0 failure events mask = FFFFF user = имя_роли`

    – для роли *имя_роли* выполнять регистрацию только неуспешных действий

1 Для назначения маски событий в режиме external используется конфигурационный файл pg_audit.conf конкретного кластера данных, который имеет следующий формат:

```
success events mask = значение failure events mask = значение user = имя_пользователя database = имя_базы_данных
success events mask = значение failure events mask = значение user = имя_пользователя
success events mask = значение failure events mask = значение
```

2 Если нужно указать правило для всех пользователей, то оно указывается в конце конфигурационного файла и в записи user не указывается.

Например,

```
success events mask = F0707 failure events mask = FFFFF
```

– для всех остальных пользователей выполнять регистрацию всех неуспешных действий и всех успешных действий, кроме доступа к данным.

3 В конфигурационном файле задаются списки *успешных* (success events mask) и *неуспешных* (failure events mask) типов запросов на доступ, которые будут регистрироваться в журнале СУБД и журнале аудита ОС для отдельных пользователей и по умолчанию.

4 Списки типов запросов на доступ задаются в виде шестнадцатеричных чисел, в которых каждому типу запроса соответствует *установленный* (для регистрируемых запросов) или *сброшенный* (для не регистрируемых запросов) бит.

5 Типы запросов и их описание:

Событие	Бит	Шестнад-цатеричное	Описание значения
SUBJECT	0	1	Добавление/изменение/удаление пользователей и групп
CONFIGURATION	1	2	Изменение конфигурации, влияющей на доступ к данным (запрос на изменение значения переменной ac_session_maclabel
RIGHTS	2	4	Запрос на модификацию прав доступа к объектам БД
CHECK_RIGHTS	3	8	Проверка прав доступа
SELECT	4	10	Выборка информации из БД
INSERT	5	20	Добавление информации в БД
UPDATE	6	40	Изменение информации в БД
DELETE	7	80	Удаление информации из БД
TRUNCATE	8	100	Очистка данных
REFERENCES	10	400	Задание столбца таблицы в качестве внешнего ключа
TRIGGER	11	800	Добавление триггера к таблице
EXECUTE	12	1000	Запуск хранимой процедуры или триггера
USAGE	13	2000	Использование объекта БД
CREATE	16	10000	Создание объектов в БД
CREATE TEMP	17	20000	Создание временного объекта
DROP	18	40000	Удаление объектов БД
ALTER	19	80000	Изменение объекта БД
CONNECT	30	40000000	Запрос на начало сеанса
DISCONNECT	31	80000000	Запрос на окончание сеанса

6 Информация о соединении пользователей с БД (CONNECT) и разъединении с ней (DISCONNECT) регистрируется всегда, при условии, что список событий не установлен в 0.
Примечание:
Любые изменения файла pg_audit.conf будут применены только при перезапуске сервера.

Назначение списков регистрации событий в режимах external, internal и internal,external, режим none

- Загрузка маски регистрации событий в режиме *external,internal* двухэтапная:
  - загрузка маски регистрации событий из файла pg_audit.conf
  - дополнение настройками из таблицы pg_db_role_settings, если в pg_db_role_settings есть более точные настройки
- Для режима *internal,external*:
  - загрузка настроек из таблицы pg_db_role_settings
  - дополнение настройками из файла pg_audit.conf
- В режиме *none* регистрация событий отключена

1 В режиме none регистрация событий отключена, но администратор может изменять маску регистрации событий с помощью команд ALTER ROLE.

Генерация событий средствами PostgreSQL

- Цель - отслеживать изменение данных в таблицах БД, передавать данные во вне (например во внешний журнал событий)
- Можно использовать:
  - триггеры — для отслеживания изменений в данных
  - команду `NOTIFY` — для отправки сообщений в канал
  - функцию pg_notify
  - команду `LISTEN` - для получения сообщений из канала

1 Для отслеживания изменения данных в таблице можно использовать *триггеры*.

2 Когда происходит изменение, информацию о нем можно отправлять в канал.

3 Для получения оповещений нужно подключиться к базе данных и выполнить команду `LISTEN` с названием канала, на который отправляются оповещения.

4 Также можно использовать таблицу — *журнал изменений*. В ней возможно размещать изменения, например, в формате json.

5 Последовательность действий:

- Создать функцию, которую будет вызывать триггер.

  Функция должна содержать команду `NOTIFY`, которая отправляет оповещение в канал.

  Например,

```
CREATE FUNCTION notify_price_change()
RETURNS trigger AS $$
BEGIN
  NOTIFY sales_change;

END;
$$ LANGUAGE plpgsql;
```

- Создать триггер для определенной таблицы.
- Создать триггер, который будет срабатывать при изменении таблицы.

  Триггер создается с помощью команды `CREATE TRIGGER`.
- В команде нужно указать название триггера, название таблицы и условие, при выполнении которого будет генерироваться оповещение.

  Например, чтобы оповещать о каждом изменении записи в таблице стоимости, можно создать триггер следующим образом:

  ```
  CREATE TRIGGER price_change
  AFTER UPDATE OR INSERT OR DELETE ON price
  FOR EACH ROW
  EXECUTE PROCEDURE notify_order_change();
  ```
- Прослушивать канал оповещений.
- Для получения оповещений нужно подключиться к базе данных и выполнить команду `LISTEN` с названием канала, на который отправляются оповещения. Например,

  ```
  LISTEN price_change;
  ```
- После этого, при любом изменении таблицы заказов, будет генерироваться оповещение, которое можно перехватить с помощью приложения или скрипта.

Команды NOTIFY, LISTEN, функция pg_notify

- `NOTIFY` — сгенерировать сообщение (уведомление)

  Синтаксис: `NOTIFY канал [ , сообщение ]`

  - `канал` – имя канала для передачи уведомления (любой идентификатор)
  - `сообщение` – строка «сообщения», которая будет передана вместе с уведомлением
- `LISTEN` – ожидать уведомления

  Синтаксис: `LISTEN канал`
- Функция pg_notify(text, text) — сгенерировать сообщение, первый аргумент имя канала, второй – текст сообщения

1 Строка сообщений должна задаваться простой *текстовой* константой.

В стандартной конфигурации её длина должна быть меньше *8000 байт*.

2 Если требуется передать двоичные данные или большой объём информации, лучше поместить их в таблицу базы данных и передать ключ этой записи.

3 Команда `NOTIFY` отправляет сообщение всем клиентским приложениям, которые до этого выполнили в текущей базе данных команду `LISTEN` с указанным именем канала. Уведомления видны всем пользователям.

4 `NOTIFY` предоставляет простой механизм межпроцессного взаимодействия для множества процессов, работающих с одной базой данных PostgreSQL. Вместе с уведомлением может быть передана строка сообщения.

5 Передавая дополнительные данные через таблицы базы данных, можно создать более высокоуровневые механизмы обмена структурированными данными.

6 Информация, передаваемая клиенту с уведомлением, включает имя канала уведомлений, PID серверного процесса, управляющего сеансом, который выдал уведомление, и строку сообщения (она будет пустой, если сообщение не задано).

7 Команда `LISTEN` регистрирует текущий сеанс для получения уведомлений через канал с заданным именем (канал).

8 Если текущий сеанс уже зарегистрирован и ожидает уведомлений через этот канал, ничего не происходит.

9 Когда вызывается команда `NOTIFY` *имя_канала* (в текущем или другом сеансе, подключённом к той же базе данных), все сеансы, ожидающие уведомления через заданный канал, получают уведомление и каждый, в свою очередь, передаёт его подключённому клиентскому приложению.

10 Сеанс может отказаться от получения уведомлений через определённый канал с помощью команды `UNLISTEN`.

11 Подписка на любые уведомления автоматически отменяется при завершении сеанса.

12 Демонстрация процедуры ожидания/получения уведомления в psql:

```
LISTEN "virtual";
NOTIFY "virtual";
Получено асинхронное уведомление "virtual" от серверного процесса с PID 15.
```

13 Когда требуется работать с динамическими именами каналов и сообщениями, удобнее использовать функцию pg_notify(text, text).

Последовательность действий

- Создать функцию, которую будет вызывать триггер
- Создать триггер, который будет срабатывать при изменении таблицы
- Прослушивать канал оповещений
- После этого при любом изменении таблицы будет генерироваться оповещение, которое можно перехватить с помощью приложения или скрипта

1. Функция, которую вызывает триггер, должна содержать команду `NOTIFY`, которая отправляет оповещение в канал.
2. Создается триггер с помощью команды `CREATE TRIGGER`.

   В команде нужно указать название триггера, название таблицы и условие, при выполнении которого будет генерироваться оповещение.

   Например, чтобы оповещать о каждом изменении записи в таблице стоимости.
3. Для получения оповещений нужно подключиться к базе данных и выполнить команду `LISTEN` с названием канала, на который отправляются оповещения.

   Например:

   ```
   LISTEN "price_change";
   ```

Пример создания функции, которую будет вызывать триггер

```
ALTER FUNCTION notify_price_change()
  RETURNS trigger AS $$
  BEGIN
    NOTIFY "price_change";
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;
```

1 Строка `$$ LANGUAGE plpgsql;` означает, что следующий блок кода написан на языке PL/pgSQL. Эта строка обычно используется в PostgreSQL для определения функций или процедур на PL/pgSQL.

2 Язык PL/pgSQL служит для создания функциональности, которая не предоставляется стандартным SQL, например, управления транзакциями или средствами управления выполнением в виде условных операторов и циклов.

Пример создания триггера, который будет срабатывать при изменении таблицы

```
CREATE TRIGGER price_change
AFTER UPDATE OR INSERT OR DELETE ON price
FOR EACH ROW
EXECUTE PROCEDURE notify_price_change();
```

1 Триггер срабатывает после выполнения обновления данных таблицы, удаления или вставки.

2 Для каждой записи будет вызываться функция `notify_price_change();`.

Вопросы для проверки

1 В каком каталоге в Astra Linux Special Edition 1.7.4 размещаются журналы сообщений PostgreSQL?

2 Какой параметр нужно настроить, чтобы в журнале сообщений фиксировались все запросы, независимо от длительности их выполнения?

3 Сгенерировать сообщение (уведомление) средствами PostgreSQL можно с помощью команды ...

Практическая работа

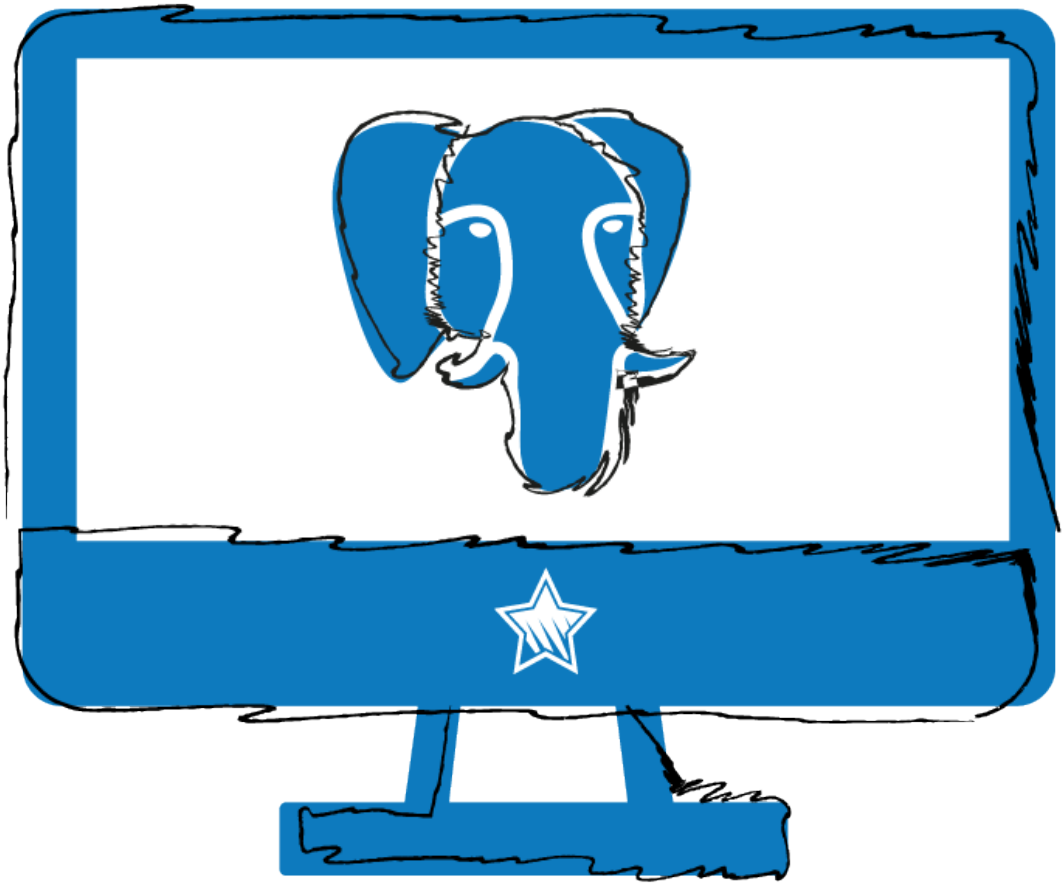
Оценка времени выполнения: 45 мин.

- Выполните задания модуля 7 из Сборника практических заданий.

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Модуль 8

Логическое резервное копирование и восстановление



ASTRA LINUX®

Модуль 8.
Логическое резервное копирование и восстановление

Краткое содержание:

- Логическое резервное копирование.
- Работа команд `COPY` и `\copy`.
- Работа с утилитами pg_dump и pg_restore в Astra Linux Special Edition.
- Работа с утилитой pg_dumpall.

Основные знания и умения:

- понимать возможности и недостатки логического резервного копирования;
- уметь использовать команды `COPY` и `\copy` для перемещения данных между таблицами PostgreSQL и файлами ФС;
- знать расширение синтаксиса команды COPY в Astra Linux Special Edition для работы с классификационными метками;
- уметь работать с утилитами pg_dump и pg_restore в Astra Linux Special Edition;
- уметь работать с утилитой pg_dumpall.

Оценка знаний и умений:

- Практическая работа, контрольные вопросы.

Дополнительная литература:

- Отсутствует.

Цели модуля 8. Логическое резервное копирование и восстановление

- Логическое резервное копирование
- Работа с командами `COPY` и `\copy`
- Работа с утилитами pg_dump и pg_restore в Astra Linux Special Edition
- Работа с утилитой pg_dumpall

Логическое резервное копирование

- *Логическое резервное копирование* – процесс сохранения данных в формате, который отражает логическую структуру БД
- При логическом резервном копировании создается SQL скрипт, который воспроизводит все операции создания и редактирования данных

1 Логическая резервная копия представляет собой обычный текстовый файл (скрипт с командами SQL). Этот файл можно отредактировать.

Например, можно:

- сделать копию только тех объектов, которые нужны;
- изменить имена или типы данных и т. д.

2 Такой скрипт с командами SQL можно выполнить на другой версии СУБД (если есть совместимость на уровне команд) или на другой архитектуре.

Возможности и недостатки логического резервного копирования

- Логическое резервное копирование позволяет:
  - восстановить кластер баз данных
  - сделать копию отдельного объекта или базы
  - восстановить объекты на кластере другой основной версии
  - восстановить объекты на другой архитектуре
- Недостатки:
  - при работе с большой БД выполнение команд займет много времени
  - восстановить объекты можно только в то состояние, в котором они были на момент создания резервной копии

1 Для создания логической резервной копии всего кластера применяется утилита *pg_dumpall*.
2 Для создания логической резервной копии отдельной БД применяется утилита *pg_dump*.
3 Выгружать данные из отдельной таблицы удобнее через команду `COPY`.

Команда COPY (серверный вариант)

- COPY — команда SQL
- В команде указывается путь к файлу на сервере БД
- Перемещает данные между таблицами PostgreSQL и файлами
- COPY TO копирует содержимое таблицы в файл
- Можно скопировать результаты запроса SELECT
- Позволяет вывести результаты: в файл, на консоль, передать на вход произвольной программе
- COPY FROM копирует данные из файла в таблицу (добавляя данные ко всему, что уже есть в таблице)

1 У пользователя, в контексте безопасности которого работает PostgreSQL (обычно postgres), должен быть доступ к файлу, который указывается в запросе.

2 Команда COPY работает быстрее, чем аналогичная команда INSERT.

3 При копировании данных из файла в таблицу таблица не очищается, данные из файла добавляются к существующим записям.

4 При выполнении команды COPY FROM:

- не применяются правила (rules).
- выполняются ограничения целостности, срабатывают триггеры.

Синтаксис команды COPY

- Копирование в файл

```
COPY { table_name [ ( column_name [, ...] ) ] | (
query ) }
TO { 'filename' | PROGRAM 'command' | STDOUT }
[ [ WITH ] ( option [, ...] ) ]
```

- Копирование из файла

```
COPY table_name [ ( column_name [, ...] ) ]
FROM { 'filename' | PROGRAM 'command' | STDIN }
[ [ WITH ] ( option [, ...] ) ]
```

1 Команда `COPY` перемещает данные между таблицами PostgreSQL и файлами стандартной файловой системы.

2 Команда `COPY TO` копирует содержимое таблицы в файл.

3 Если указан список столбцов, `COPY TO` копирует в файл только данные из указанных столбцов.

4 Команда `COPY FROM` копирует данные из файла в таблицу (добавляя данные ко всему, что уже есть в таблице).

5 Если в команде `COPY FROM` указан список столбцов, каждое поле в файле вставляется по порядку в указанный столбец. Столбцы таблицы, не указанные в списке COPY из столбца, получат значения по умолчанию.

6 С помощью команды `COPY TO` можно также скопировать результаты запроса SELECT.

7 Например, `COPY table TO` может использоваться для копирования тех же строк, что и `SELECT * FROM ONLY table`.

8 Синтаксис `COPY (SELECT * FROM table) TO ...` можно использовать для копирования всех строк в иерархии наследования, секционированной таблицы или представления.

9 Команду `COPY FROM` можно использовать с обычными, внешними или секционированными таблицами, а также с представлениями, которые имеют триггеры `INSTEAD OF INSERT`.

10 При выполнении команды COPY FROM будут вызваны все триггеры, и будут проверены ограничения в целевой таблице. Правила (rules) вызываться не будут.

11 Для столбцов идентификаторов команда COPY FROM всегда будет записывать значения столбцов, указанных во входных данных.

12 Входные данные интерпретируются в соответствии с параметром ENCODING или текущей клиентской кодировкой.

13 Выходные данные кодируются в соответствии с параметром ENCODING или текущей клиентской кодировкой, даже если данные не проходят через клиент, а считываются из файла или записываются в файл непосредственно сервером.

14 Команда COPY останавливает операцию при первой ошибке. Это не должно привести к проблемам в случае использования COPY TO, но при использовании COPY FROM в целевую таблицу уже будут записаны первые строки. Эти строки не будут видны или доступны, но они будут занимать место на диске. Это может привести к значительному и бессмысленному расходу дискового пространства, если сбой произошел во время операции копирования большого объема. Для освобождения дискового пространства следует выполнить VACUUM.

15 Если для таблицы включена защита на уровне строк, соответствующие политики SELECT будут применяться и к операторам COPY TO. Операторы COPY FROM для таблиц с защитой строк в настоящее время не поддерживаются. Вместо них следует использовать эквивалентные операторы INSERT.

Особенности работы команды COPY с представлениями

- При вызове команды COPY TO можно указывать имена таблиц, их столбцы или указывать запросы на выборку
- Нельзя напрямую ссылаться на представления по их именам
- Для копирования данных из представлений можно использовать запрос

  Например,
  ```
  COPY (SELECT * FROM имя_представления) TO
  '/backup_path/file.csv' WITH CSV HEADER;
  ```

1 В синтаксисе команды COPY указывается, что обращаться в команде можно только к таблицам или их столбцам.

2 Для обхода ограничения копирования из представлений можно использовать выборку с помощью запроса и обращаться к представлению уже в запросе.

Привилегии для работы с командой COPY в PostgreSQL

- Для выполнения команды должна быть привилегия:
  - SELECT – на таблицу, значения которой считываются путем копирования
  - INSERT – на таблицу, в которую значения вставляются путем копирования
- Достаточно иметь привилегии столбца для столбцов, перечисленных в команде
- COPY с указанием имен файлов разрешено супер-пользователям БД, членам ролей pg_read_server_files, pg_write_server_files

1 Использование команды COPY с указанием имен файлов или программ разрешено только суперпользователям базы данных или пользователям, которым предоставлена одна из ролей pg_read_server_files, pg_write_server_files или pg_execute_server_program, поскольку это позволяет читать или записывать любой файл, или запускать программу, к которой сервер имеет привилегии доступа.

Привилегии на файлы в ФС для работы с командой COPY

- Файлы, указанные в команде COPY, должны быть на компьютере-сервере
- Файлы должны быть доступны пользователю, в контексте безопасности которого работает PostgreSQL (идентификатор пользователя, от имени которого запускается сервер):
  - для чтения (для работы `COPY FROM`)
  - записи (для работы `COPY TO`)
- Разрешения при работе в Astra Linux Special Edition должны быть как на уровне дискреционного доступа, так и мандатного

1 Файлы, указанные в команде COPY, считываются или записываются непосредственно сервером, а не клиентским приложением.

Следовательно, они должны находиться на компьютере-сервере базы данных или быть доступными для него, а не для клиента.

2 Рекомендуется, чтобы имя файла, используемое в команде COPY, всегда указывалось как абсолютный путь.

Это требование принудительно выполняется сервером в случае использования команды `COPY TO`. Если при использовании команды `COPY FROM` указан относительный путь, он будет интерпретироваться относительно рабочего каталога серверного процесса (обычно каталога данных кластера), а не рабочего каталога клиента.

Некоторые параметры команды COPY

- `STDIN` – указывает, что данные будут поступать из стандартного потока ввода
- `STDOUT` – указывает, что данные будут выдаваться в стандартный поток вывода
- `FORMAT` – выбирает формат чтения или записи данных:
  - text (текстовый)
  - csv (значения, разделённые запятыми, Comma Separated Values)
  - binary (двоичный)

  По умолчанию выбирается формат *text*
- `DELIMITER` – задаёт символ, разделяющий столбцы в строках файла

1 Когда заданы значения STDIN или STDOUT, данные передаются по соединению между клиентом и сервером.

2 По умолчанию в качестве разделителя используется символ табуляции в текстовом формате и запятая в формате CSV.

Задаваемый символ должен быть *однобайтовым*.

Для формата binary этот параметр не допускается.

3 С полным синтаксисом команды COPY можно ознакомиться на сайте https://www.postgresql.org/docs/11/sql-copy.html

Примеры использования команды COPY

- `COPY country TO STDOUT (DELIMITER '|');`
  - передача таблицы в стандартный поток вывода с разделителем полей «вертикальная черта» (|)
- `COPY (SELECT * FROM country WHERE country_name LIKE 'A%') TO '/usr1/proj/bray/sql/a_list_countries.copy';`
  - копирование в файл данных тех стран, название которых начинается с «A»:
- `COPY country TO PROGRAM 'gzip > /usr1/proj/bray/sql/country_data.gz';`
  - копирование данных в сжатый файл через внешнюю программу сжатия

Расширение команды COPY в Astra Linux Special Edition для работы с классификационными метками

- Флаг MACS указывает на то, что записи должны выгружаться/загружаться вместе с их классификационными метками
- Пример синтаксиса:
  - `COPY <имя_отношения> TO stdout WITH MACS;`
  - `COPY <имя_отношения> FROM stdin WITH MACS;`
- Вывести доступные пользователю данные вместе с классификационными метками может любой пользователь
- Загрузить записи в БД вместе с их классификационными метками может пользователь с установленным мандатным атрибутом `ac_capable_chmac`

1 Для того, чтобы сохранить записи вместе с их классификационными метками в архиве и в дальнейшем загрузить их обратно, предусмотрен специальный флаг MACS команды COPY.

2 При загрузке данных в таблицу классификационные метки загружаемых записей должны находиться в пределах диапазона классификационных меток пользователя, который их загружает. Либо пользователь должен иметь атрибуты игнорирования мандатного управления *ac_capable_ignmaclvl* и *ac_capable_ignmaccat*.

3 Использовать команду COPY без указания меток может любой пользователь. Загруженные таким образом данные будут иметь классификационные метки, равные текущей классификационной метке сеанса пользователя.

Команда \copy в psql (клиентский вариант)

- `\copy` – команда psql
- В команде указывается путь к файлу на клиенте, обращение к файлу происходит на клиенте, на сервер передается только содержимое файла
- У пользователя, который запускает команду, должны быть привилегии на использование файла
- Синтаксис:

```
\copy { table [ ( column_list ) ] | ( query ) } { from |
to } { 'filename' | program 'command' | stdin | stdout |
pstdin | pstdout } [ [ with ] ( option [, ...] ) ]
```

1 Метакоманда `\copy` вызывает `COPY FROM STDIN` или `COPY TO STDOUT`, а затем работает с данными в файле, доступном клиенту psql.

2 Когда применяется команда `\copy`, доступность файла и права доступа зависят от клиента, а не от сервера.

Практическая работа

Оценка времени выполнения: 30 мин.

- Выполните задания модуля 8, часть 1 из Сборника практических заданий.

Утилита pg_dump

- Утилита pg_dump служит для создания резервной копии БД
- Результатом работы утилиты pg_dump может быть:
  - SQL-скрипт, содержащий команды, создающие выбранные объекты
  - файл в специальном формате с оглавлением
- Работа утилиты не препятствует доступу других пользователей к БД (ни для чтения, ни для записи)
- Для восстановления из скрипта его содержимое можно передать psql:

```
psql -U username -d database_name -a -f backup_script
```

1 Базу данных для восстановления надо создавать из шаблона *template0*. Иначе все изменения, внесенные в *template1*, также попадут в восстановленную БД.

2 Также заранее должны быть созданы необходимые роли и табличные пространства. Поскольку эти объекты относятся к кластеру БД, они не будут выгружены в резервную копию.

3 Созданный утилитой pg_dump архивный файл не содержит информации о статистике, которую использует оптимизатор запросов. Таким образом, рекомендуется выполнять команду ANALYZE после восстановления из резервной копии для достижения лучшей производительности.

Утилита pg_restore

- восстанавливает БД PostgreSQL из архивного файла (каталога), созданного с помощью утилиты pg_dump
- позволяет избирательно восстанавливать данные
- может подключиться к базе данных и восстановить содержимое архива непосредственно в базу данных
- может создать скрипт, содержащий SQL-команды, необходимые для перестройки базы данных, и записать их в файл или стандартный вывод
- некоторые параметры, управляющие выводом, аналогичны параметрам pg_dump

1 Утилита pg_restore формирует команды, необходимые для восстановления базы данных в том состоянии, в котором она находилась на момент сохранения.

Архивные файлы также позволяют pg_restore избирательно восстанавливать данные или даже изменять порядок, в котором восстанавливаются объекты.

2 Архивные файлы спроектированы таким образом, чтобы их можно было переносить на разные архитектуры.

3 Утилита pg_restore может работать в двух режимах:

- если при выполнении команды указано имя БД, pg_restore подключается к этой базе данных и восстанавливает содержимое архива непосредственно в базу данных;
- если имя БД не указано, создается скрипт, содержащий SQL-команды, необходимые для перестройки базы данных, и записывается в файл или стандартный вывод. Этот вывод скрипта эквивалентен формату вывода обычного текста pg_dump. Таким образом, некоторые параметры, управляющие выводом, аналогичны параметрам pg_dump.

Синтаксис и параметры подключения утилиты pg_dump

- `pg_dump [connection-option...] [option...] [dbname]`
- `dbname` – указывает имя базы данных, из которой будут выгружаться данные
- `-h сервер (--host=сервер)` – указывает имя компьютера, на котором работает сервер
- *`-p порт (--port=порт)`* – указывает TCP-порт или расширение файла локального Unix-сокета, через который сервер принимает подключения
- `-U имя_пользователя (--username=имя_пользователя)` – имя пользователя, под которым производится подключение

1 Если имя БД не задано, то используется значение переменной окружения PGDATABASE. Если и переменная не задана, то в качестве имени базы будет взято имя пользователя, под которым осуществляется подключение.

2 Если значение ключа `-h` начинается с косой черты, оно определяет каталог Unix-сокета.

Значение ключа `-h` по умолчанию берётся из переменной окружения PGHOST, если она установлена. В противном случае выполняется подключение к Unix-сокету.

3 Значение ключа `-p` по умолчанию определяется переменной окружения PGPORT(если она установлена), либо числом, заданным при компиляции.

4 Со всеми параметрами утилиты можно ознакомиться на сайте https://www.postgresql.org/docs/11/app-pgdump.html

Частичная выгрузка с помощью утилиты pg_dump

- При использовании следующих параметров выгружаться будут только указанные объекты:
  - `-n schema (--schema=`*`имя_схемы`*`)` – шаблон для имени схемы
  - `-t (--table=`*`имя_таблицы`*`)` – шаблон для имен таблиц
- При использовании следующих параметров выгружаться будут объекты, за исключением указанных:
  - `-N (--exclude-schema=`*`имя_схемы`*`)` — шаблон для имен схем
  - `-T (--exclude-table=`*`имя_таблицы`*`)` — шаблон для имен таблиц
- `-s (--schema-only)` — выгрузить только определения объектов без данных
- `-a (--data-only)` — выгрузить только данные, без создания объектов

1 При выгрузке схемы выбирается как сама схема, так и все содержащиеся в ней объекты. Если этот параметр не указан, все несистемные схемы в целевой базе данных будут удалены.

2 Несколько схем можно выбрать, записав несколько ключей `-n`.

3 Параметр schema интерпретируется как шаблон в соответствии с теми же правилами, которые используются командами `psql \d`, поэтому также можно выбрать несколько схем, введя в шаблон подстановочные знаки.

4 При указании параметра `-a, --data-only` выгружаются только данные, а не определения данных. Выгружаются табличные данные, большие объекты и значения последовательностей.

Параметры файла и формата для выгрузки

- `-f file (--file=file)` — указывает имя и путь к файлу для выгрузки
- `-F format (--format=format)` – указывает формат выходных данных
- Формат может быть одним из следующих:
  - `p (plain)` — вывод в текстовый файл SQL-скрипта (по умолчанию)
  - `c (custom)` — вывод в архив пользовательского формата, подходящий для ввода в pg_restore
  - `d (directory)` — вывод в архив в формате каталога, подходящий для ввода в pg_restore
  - `t (tar)` – вывод архива в формате tar, подходящий для ввода в pg_restore

1 Если параметр `-f file` не указан, то утилита pg_dump выведет результаты в консоль.

2 Для вывода данных в каталог, параметр `-f` должен быть указан, и в качестве его значения должен быть указан целевой каталог. В этом случае каталог создается с помощью pg_dump. Он не должен существовать на момент запуска команды.

3 При использовании параметра `-d (--directory)` создается каталог с одним файлом для каждой выгружаемой таблицы и большого двоичного объекта, а также так называемый *файл оглавления*, описывающий выгружаемые объекты в формате, который может прочитать pg_restore.

Архивом в формате каталога можно управлять с помощью стандартных инструментов Unix; например, файлы в несжатом архиве можно сжать с помощью инструмента *gzip*. Этот формат сжат по умолчанию и также поддерживает параллельное выполнение резервного копирования с использованием нескольких задач.

4 Формат *tar* совместим с форматом каталога: при извлечении архива в формате tar создается архив в формате каталога.

Формат tar не поддерживает сжатие. Кроме того, при использовании формата tar, относительный порядок элементов табличных данных не может быть изменен во время восстановления.

Формат custom (-F c, --format=custom)

- Резервная копия в специальном формате, содержит не только объекты, но и оглавление
- Это позволяет сначала сделать полную копию, а выбирать объекты при загрузке через pg_restore
- Файл формата custom по умолчанию сжат
- Перед восстановлением можно:
  - получить из оглавления список объектов, содержащихся в резервной копии (ключ `--list, -l`).
  - отредактировать список, удалив ненужное
  - передать измененный список на вход pg_restore (ключ `--use-list, -L`)

1 Формат *custom* позволяет сначала сделать полную копию, а выбирать объекты уже при загрузке.

2 Параметр *c (custom)* вместе с форматом вывода каталога - это наиболее гибкий формат вывода, поскольку он позволяет вручную выбирать и изменять порядок архивированных элементов во время восстановления.

3 Резервная копия формата custom хранится в двоичном виде.

4 Для просмотра оглавления резервной копии можно использовать ключ `--list`.

Например,

```
pg_restore --list sales.custom
```

5 Такой список можно записать в файл:

```
pg_restore --list sales.custom > sales_list.list
```

6 Список, выгруженный в файл, можно отредактировать. Строки в файле могут быть закомментированы, удалены. Можно изменять порядок строк и использовать его для восстановления с помощью ключа `--use-list`.

7 Например, после удаления таблицы customers можно выборочно восстановить именно ее.

```
pg_restore --table=customers -d
sales /home/backups/sales.custom
pg_restore --list /home/backups/sales.custom
```

8 Восстановление возможно в несколько потоков.

Пример резервного копирования и восстановления с использованием формата custom и оглавления

- `pg_dump --format=custom -d sales -f sales.custom`
  — резервное копирование БД sales в формате custom
- `pg_restore --list sales.custom > sales_list.list`
  – выгрузка оглавления резервной копии в файл
- `pg_restore -d sales --use-list sales_list.list sales.custom`
  – восстановление объектов в БД sales согласно оглавлению sales_list.list из резервной копии sales.custom

1 Пример восстановления отдельной таблицы из резервной копии, созданной с помощью утилиты pg_dump в формате custom:

`pg_restore --table=customers -d sales sales.custom`

– восстановление таблицы customers БД sales из резервной копии sales.custom.

Формат directory (-F d, --format=directory)

- При указании формата directory создается каталог, содержащий объекты и оглавление
- По умолчанию файлы внутри каталога будут сжаты
- Резервная копия может создаваться параллельно в несколько потоков (количество указывается в ключе `-j, --jobs`), что существенно сокращает время создания резервной копии
- При сохранении данных в этом формате при восстановлении данных также возможно использование нескольких потоков

1 Резервная копия будет содержать согласованные данные, даже если для ее создания будут использовать несколько потоков.

Это достигается за счет создания общего снимка данных для всех параллельно работающих процессов.

Пример резервного копирования и восстановления с использованием резервной копии в формате directory

- `sudo pg_dump -U iivanov -h localhost --format=directory --jobs=2 -d sales -f /home/backups/sales_backup.directory`
- `sudo pg_restore -U iivanov -h localhost --clean --jobs=2 -p 5433 -d sales /home/backups/sales_backup.directory`
  - где `--clean` – удалить (DROP) объекты базы данных, прежде чем пересоздавать их

1 Если при использовании ключа `--clean` не указывать дополнительно ключ `--if-exists`, то при попытке удаления объектов из целевой БД будут выдаваться ошибки, если таких объектов в БД не окажется.

Эти ошибки не повлияют на успешность выполнения других операций.

Сжатие резервной копии при работе pg_dump

- Параметр `--compress` утилиты pg_dump позволяет указать метод или степень сжатия:

  `--compress=уровень`

  `--compress=метод[:параметры_сжатия]`

- В качестве метода сжатия можно указывать значения: gzip, lz4, zstd или none (без сжатия)
- По умолчанию:
  - для формата plain сжатие не производится
  - для форматов custom и directory применяется умеренный уровень сжатия с использованием gzip

1 В качестве дополнительной информации можно передать параметры сжатия.

2 Если в строке информации передаётся целое число, оно задаёт уровень сжатия.

В противном случае она должна содержать список элементов, разделённых запятыми, в форме ключевое_слово или ключевое_слово=значение.

Поддерживаются ключевые слова level и long.

3 Если уровень сжатия не указан, будет выбран уровень сжатия по умолчанию.

4 Если указан только уровень сжатия, но не указан метод, будет применяться метод сжатия gzip, когда уровень больше 0, а когда уровень равен 0, сжатие не будет выполняться.

5 Для форматов custom и формата directory будут сжиматься файлы отдельных таблиц.

6 Сжатие с использованием zstd можно проводить с указанием long, что может улучшить коэффициент сжатия за счёт повышения нагрузки на память.

7 Формат tar в настоящий момент не поддерживает сжатие.

Сравнение форматов резервной копии pg_dump

	plain	custom	directory	tar
восстановление с помощью утилиты	psql	pg_restore		
использование сжатия	gzip, lz4, zstd			
поддержка выборочного восстановления		+	+	+
параллельные потоки при резервировании			+	
параллельные потоки при восстановлении		+	+	

1 От того, в каком формате создавалась резервная копия, будет зависеть метод восстановления.

Полезные параметры pg_dump

- `-O (--no-owner)` — указывает, что не нужно генерировать команды для установки владельца объектов
- `-x (--no-acl)` — указывает, что не нужно генерировать команды для установки привилегий
- `--no-comments` — указывает, что не нужно генерировать комментарии
- `-c (--clean)` — указывает, что нужно генерировать команды DROP для создаваемых объектов
- `-C (--create)` — указывает, что нужно генерировать команды создания БД и подключения к ней

1 Параметры `-O, --no-owner, -x, --no-acl` можно использовать, если восстановление резервной копии будет проходить на системе с другим набором ролей.

2 При использовании параметра `-O` не выводятся команды для установки владельцев объектов в соответствии с исходной базой данных.

По умолчанию pg_dump выдает инструкции ALTER OWNER или SET SESSION AUTHORIZATION для установки владельцев для объектов базы данных. Эти инструкции завершатся ошибкой при запуске скрипта, если только он не запущен суперпользователем (или тем же пользователем, которому принадлежат все объекты в скрипте).

Чтобы создать скрипт, который может быть восстановлен любым пользователем, но предоставит этому пользователю право собственности на все объекты, следует указать параметр `-O`.

Этот параметр игнорируется при создании архивного (нетекстового) выходного файла. Для архивных форматов вы можете указать опцию при вызове pg_restore.

3 Параметр `-c (--clean)` указывает, что нужно выводить команды для очистки (удаления) объектов БД перед выводом команд для их создания.

4 Если не указано значение `--if-exists`, восстановление может генерировать сообщения об ошибках, если какие-либо объекты отсутствовали в целевой базе данных. Эти ошибки не повлияют на успешность восстановления.

Этот параметр игнорируется при создании архивного (нетекстового) выходного файла. Для архивных форматов можно указать этот параметр при вызове pg_restore.

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Выгрузка данных с помощью утилиты pg_dump без указания классификационных меток объектов

- Параметр `--disable-macs` позволяет создавать резервные копии без указания классификационных меток объектов
- Синтаксис:

```
pg_dump --disable-macs <имя_базы_данных> --file
<имя_файла_резервной_копии>
```

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Работа pg_dump в условиях мандатного управления доступом

- В текущей реализации postgres при выгрузке дампа базы данных принят следующий порядок действий:
  - в дамп выгружаются все записи (включая записи, имеющие ненулевые классификационные метки)
  - при выгрузке аккумулируется максимальная классификационная метка найденных записей
  - после выгрузки производится попытка установить аккумулированную метку на файл дампа
  - если попытка установить метку не удалась, то файл удаляется

1 В случае создания резервной копии необходимо назначить максимальную метку на каталог, в который будет выгружена база данных (для назначения метки на файл копии).

2 Файл дампа по сути является контейнером. По правилам мандатного разграничения доступа контейнер не должен содержать сущности с меткой выше, чем у контейнера.

3 Если удаление файла запрещено, то в результате файл с выгруженными записями остается в системе и будет доступен для чтения.

4 Поэтому, перед выгрузкой данных, следует заранее установить корректную мандатную метку на каталог, в который будут выгружаться резервные копии.

Резервное копирование и восстановление с БД с ненулевыми классификационными метками

- Для создания и восстановления резервных копий баз данных с ненулевыми классификационными метками необходимо, чтобы пользователь имел привилегии:
  - PARSEC_CAP_SETMAC
  - PARSEC_CAP_CHMAC
- По умолчанию пользователи этими привилегиями не обладают

1 В случае восстановления, помимо указанных привилегий, требуются права администратора в базе данных (для создания объектов и назначения мандатных атрибутов).

2 Для восстановления копии базы данных с мандатными атрибутами в другой базе данных необходимо:
   - назначить максимальную метку на каталог, в который будет проводиться выгрузка резервной копии;
   - создать резервную копию исходной базы данных от имени пользователя с привилегиями PARSEC_CAP_SETMAC и PARSEC_CAP_CHMAC;
   - на целевом кластере назначить мандатные атрибуты на кластер;
   - восстановить резервную копию базы данных с помощью клиента psql (если резервная копия сделана в текстовом виде) или с помощью pg_restore (если резервная копия сделана в бинарном виде) от имени пользователя администратора базы данных с привилегиями PARSEC_CAP_SETMAC и PARSEC_CAP_CHMAC.

3 Утилита pg_dump, поставляемая вместе с СУБД версии 11, выгружает команды по установке комментариев, меток безопасности и назначению мандатных атрибутов в следующем виде:

```
COMMENT ON DATABASE CURRENT_DATABASE IS 'комментарий';
SECURITY LABEL ON DATABASE CURRENT_DATABASE IS '...';
MAC LABEL ON DATABASE CURRENT_DATABASE IS '...';
```

```
MAC CCR ON DATABASE CURRENT_DATABASE IS '...';
```

4 Такая резервная копия может быть восстановлена с установкой комментариев, меток безопасности и мандатных атрибутов в желаемую базу данных.

5 Для восстановления резервных копий баз данных, сделанных в предыдущих версиях, в СУБД версии 11 необходимо установить параметр

```
ac_auto_adjust_macs = true;
```

6 Для переноса кластера с предыдущих версий СУБД на версию 11 необходимо воспользоваться утилитами pg_upgradecluster или pg_upgrade.

Подготовка каталога для резервных копий в условиях мандатного управления доступом

- На каталог следует установить тот уровень конфиденциальности, который является максимальным для выгружаемого набора данных
- Категории, присутствующие в выгруженных данных, также должны быть указаны в метке безопасности каталога
- Утилита pdpl-file позволяет менять мандатную метку каталога:

  Синтаксис: `pdpl-file [ОПЦИИ]... [УРОВЕНЬ] [:УРОВЕНЬ_ЦЕЛОСТНОСТИ[:КАТЕГОРИЯ[:ФЛАГИ]]] ФАЙЛ…`

  Пример: sudo `pdpl-file 3::F:CCNR /home/backups`

1 Поэтому, перед выгрузкой данных, следует заранее установить на каталог для выгрузки мандатную метку, которая по уровню конфиденциальности будет соответствовать максимальному уровню конфиденциальности выгружаемых данных.

2 Категории, присутствующие в выгруженных данных, также должны быть указаны в метке безопасности каталога.

3 Изменить мандатную метку каталога можно с помощью утилиты pdpl-file.

Утилита pdpl-file позволяет изменить мандатные свойства файла (каталога): УРОВЕНЬ (конфиденциальности), УРОВЕНЬ_ЦЕЛОСТНОСТИ, КАТЕГОРИЮ и ФЛАГИ.

Ключи pg_restore для фильтрации

- `-I (--index)` — загрузить определенные индексы
- `-P (--function)` — загрузить определенные функции
- `-T ( --trigger)` — загрузить определенные триггеры
- Поддерживаются те же ключи для фильтрации, что и в pg_dump:
  -n schema (--schema=*имя_схемы*), -N (--exclude-schema=*имя_схемы)*,
  -t (--table=*имя_таблицы*), -T (--exclude-table=*имя_таблицы*),
  -s (--schema-only), -a (--data-only)

1 Параметр `-I` *имя_индекса* `(--index=`*имя_индекса*`)` позволяет восстановить из резервной копии определение индекса только с определенным именем. Определения нескольких индексов могут быть заданы с помощью нескольких параметров `-I`.

2 Параметр `-P` *имя_функции* `(argtype [, …])` `(--function=`*имя_функции*`(argtype [, …]))` позволяет восстанавливать только функции с определенными именами. Указывать имена функций и их аргументы нужно точно также, как они указаны в оглавлении файла резервной копии. С помощью нескольких ключей `-P` можно задать несколько функций.

3 Параметр `-T` *имя_триггера* `(--trigger=`*имя_триггера*`)` позволяет восстанавливать только триггер с определенным именем. С помощью нескольких параметров переключателей `-T` можно задать несколько триггеров.

4 Пример восстановления отдельной таблицы из резервной копии, созданной с помощью утилиты pg_dump в формате custom:

```
pg_restore --table=customers -d sales sales.custom
```

– восстановление таблицы customers БД sales из резервной копии sales.custom.

5 Подробнее о утилите pg_restore:
https://www.postgresql.org/docs/11/app-pgrestore.html

Подготовка к восстановлению данных через pg_restore в условиях мандатного управления доступом

- Установить классификационные метки на объектах выше по иерархии, чтобы не нарушалось правило: метка сущности не может превышать метку контейнера, в котором она содержится
- Выставить корректное значение мандатного признака CCR
- Убедиться, что есть доступ к файлам резервной копии на уровне ФС, в том числе на уровне мандатного разграничения доступа

1 Перед восстановлением необходимо удостовериться, что корректно выставлены классификационные метки на уровне кластера и табличного пространства.

2 Например, если на уровне кластера установлена классификационная метка {3,3}, признак CCR установлен в ON, не удастся восстановить БД с классификационной меткой {2,0} в такой кластер.

Для выполнения восстановления следует предварительно установить значение признака CCR в OFF.

Утилита pg_dumpall

- служит для создания резервной копии кластера, включая роли, табличные пространства и предоставление привилегий для параметров конфигурации
- в результате работы утилиты формируется SQL-скрипт, который может быть выведен на консоль или в файл
- не поддерживает параллельную выгрузку данных

1 Утилита создает резервную копию, вызывая pg_dump для каждой базы данных в кластере.

2 Поскольку pg_dumpall считывает таблицы из всех БД, для создания полной резервной копии для запуска утилиты нужно использовать учетную запись, имеющую доступ ко всем БД, возможно, учетную запись суперпользователя.

3 Результат работы утилиты – SQL-скрипт, будет записан в стандартный вывод. Для перенаправления его в файл следует использовать параметр `-f/--file` или операторы оболочки.

4 Утилите pg_dumpall необходимо несколько раз подключиться к серверу PostgreSQL (по одному разу для каждой базы данных).

При использовании аутентификации по паролю, она будет запрашивать пароль каждый раз. В таких случаях удобно иметь файл ~/.pgpass.

5 Это означает, что pg_dumpall не поддерживает параллельную выгрузку данных, что может оказаться проблемой при больших объемах данных. В таком случае можно воспользоваться ключом `--globals-only`, чтобы выгрузить только роли и табличные пространства, а сами базы данных выгрузить с помощью pg_dump.

6 Для восстановления данных из резервной копии, созданной с помощью pg_dumpall, нужно запустить созданный с помощью pg_dumpall сценарий, используя утилиту psql.

Влияние политик защиты строк на резервное копирование

- Если на таблицах определены политики защиты строк, то есть вероятность, что выгруженные данные окажутся неполными
- Во избежание этого нужно перед выполнением команды COPY установить параметр сеанса `row_security` в значение `off`
- При этом значении параметра применение политики RLS приведет к явной ошибке

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Вопросы для проверки

1 В SQL-команде COPY указывается путь к файлу, размещенному на ...

2 Результатом работы утилиты pg_dump может быть ...

3 Если при вызове утилиты pg_dump не указан формат, то будет использован формат ...

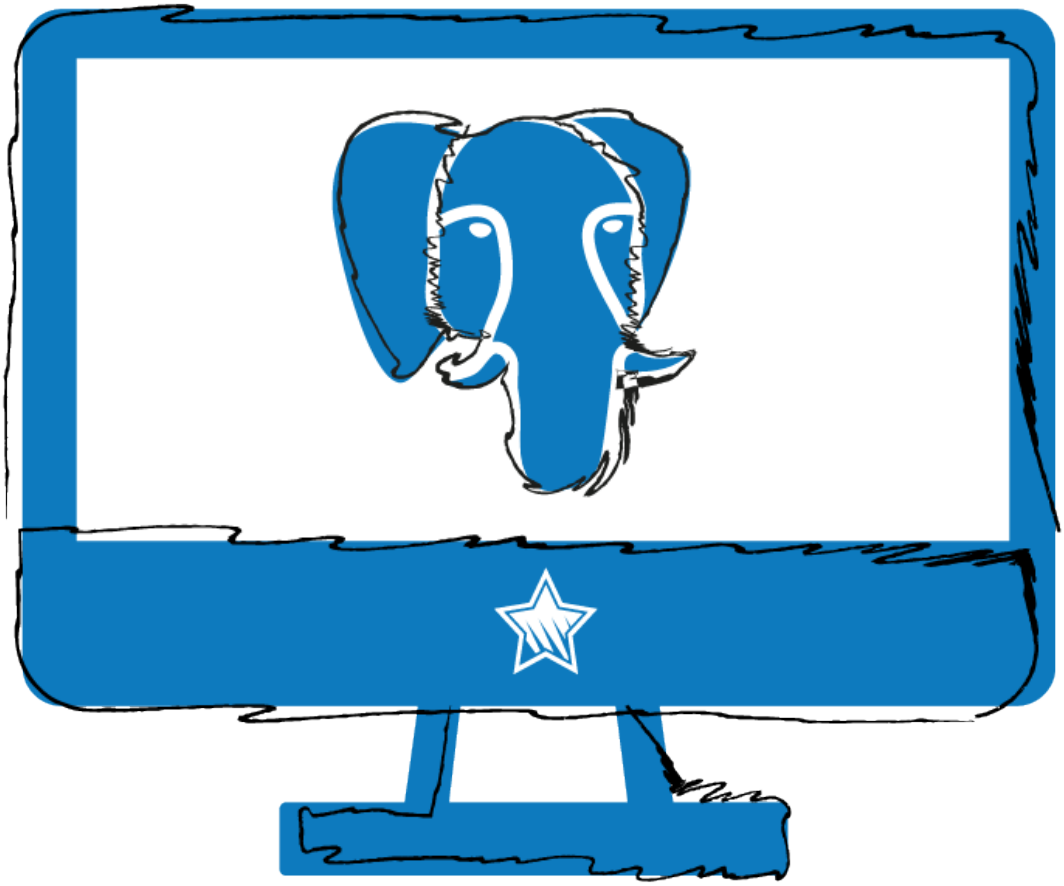
Практическая работа

Оценка времени выполнения: 50 мин.

- Выполните задания модуля 8, часть 2 из Сборника практических заданий.

Модуль 9

Физическое резервное копирование и восстановление



ASTRA LINUX®

Модуль 9.
Физическое резервное копирование и восстановление

Краткое содержание:

- Особенности физического резервного копирования.
- Холодное резервное копирование.
- Горячее резервное копирование.
- Создание автономной резервной копии с помощью утилиты pg_basebackup.
- Непрерывное архивирование журналов предзаписи.
- Создание резервной копии при настроенном непрерывном архивировании журналов предзаписи.
- Восстановление с использование резервных копий журналов предзаписи.
- Восстановление до определенной точки.

Основные знания и умения:

- понимать особенности физического резервного копирования;
- уметь выполнять холодное резервное копирование;
- уметь выполнять горячее резервное копирование;
- уметь создавать автономную резервную копию с помощью утилиты pg_basebackup;
- уметь настраивать непрерывное архивирование журналов предзаписи;
- уметь создавать резервные копии при настроенном непрерывном архивировании журналов предзаписи;
- уметь выполнять восстановление с использование резервных копий журналов предзаписи;
- уметь выполнять восстановление до определенной точки.

Оценка знаний и умений:

- Практическая работа, контрольные вопросы.

Дополнительная литература:

- Отсутствует.

Цели модуля 9. Физическое резервное копирование и восстановление

- Особенности физического резервного копирования
- Холодное резервное копирование
- Горячее резервное копирование
- Создание автономной резервной копии с помощью утилиты pg_basebackup
- Непрерывное архивирование журналов предзаписи
- Создание резервной копии при настроенном непрерывном архивировании журналов предзаписи
- Восстановление с использование резервных копий журналов предзаписи
- Восстановление до определенной точки

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Физическое резервное копирование

- *Физическое резервное копирование* (резервное копирование на блочном уровне) в PostgreSQL выполняется путем копирования физических файлов, из которых состоит база данных PostgreSQL, вместе с файлами журнала предзаписи (WAL)
- Восстановление из резервной копии проходит быстрее, чем при логическом копировании и восстановлении
- Статистика также восстанавливается из резервной копии
- Можно восстановить систему только на совместимой платформе
- Невозможно сделать физическую копию отдельных баз данных кластера (возможно копирование только всего кластера)

1 *Физическое резервирование* – копирование всех файлов, относящихся к кластеру БД.

2 При физическом резервном копировании создаются полные двоичные копии файлов.

3 Восстановление из физической резервной копии возможно только на совместимой платформе. То есть должны совпадать: ОС, включая разрядность, порядок байтов в представлении чисел и т. д, а также основная (мажорная) версия PostgreSQL.

4 Для восстановления данных при использовании физического резервного копирования требуются:
   - копия файлов кластера (базовая резервная копия);
   - копии журналов предзаписи, необходимых для восстановления согласованности.

5 Если копии файлов создавались при корректно остановленном сервере, то копии журналов предзаписи не требуются.

6 Наличие архивов журналов предзаписи позволяет получить состояние кластера на любой момент времени. Таким образом можно восстановить состояние кластера на момент времени незадолго перед сбоем.

7 Физическое резервное копирование позволяет делать резервную копию без выключения сервера и обеспечивает высокую скорость восстановления.

Холодное резервное копирование

- Создается копия файлов кластера
- При восстановлении файлы разворачиваются из резервной копии, после чего сервер можно запускать
- Для создания резервной копии с согласованными данными требуется остановить сервер (с выполнением контрольной точки)
- Создание резервной копии с несогласованными данными выполняют:
  - если сервер был экстренно выключен
  - из снимка файловой системы (для создания резервной копии не требуется останавливать кластер БД)

1 Время простоя сервера можно сократить за счет предварительного выполнения *rsync* (или другого похожего инструмента) при работающем сервере.

2 Утилита rsync используется для эффективной синхронизации файлов и каталогов между различными местами на основе бинарного протокола. В PostgreSQL, rsync используется для репликации или создания резервных копий данных PostgreSQL.

3 При использовании rsync после останова сервера rsync скопирует только изменения, произошедшие после копирования основной части данных. Таким образом можно сократить время простоя.

4 Пример использования:

```
sudo apt install rsync
sudo rsync -av /var/lib/postgresql/11/main/
/home/rsync_backup
```

где:

- параметр `-a, --archive` указывает на применение архивного режима, включает рекурсивное копирование и сохранение прав и владельца (опция включает в себя опции rlptgoD);
- параметр `-v, --verbose` – указывает, что будет выведена подробная информация о процессе.

5 В случае восстановления из копии с несогласованными данными восстановление проходит аналогично, но при запуске серверу потребуется выполнить восстановление до согласованного состояния. Эта процедура потребует дополнительного времени.

6 Подробнее: https://www.postgresql.org/docs/11/backup-file.html

Горячее резервное копирование

- Горячее резервное копирование выполняется при работающем сервисе PostgreSQL
- Данные при этом читаются из файлов (не через буферный кеш)
- Копирование файлов данных может занимать много времени
- Для горячего резервирования требуются специальные инструменты
- Копия файлов из ФС будет содержать несогласованные данные
- Для восстановления потребуются журналы предзаписи, сформированные за время копирования файлов (как минимум)

1 Причины, почему нельзя использовать копирование на уровне файлов при работающем сервере:
   - в копию попадут несогласованные данные;
   - содержимое файлов на диске изменяется во время копирования;
   - копирование файлов данных может быть длительным, и сервер, после выполнения контрольной точки, может удалить часть файлов журнала, которые уже не нужны для восстановления после сбоя, но нужны для резервной копии.

2 Если попытаться скопировать такие файлы в другой каталог, то полученная таким образом резервная копия будет непригодна для восстановления.

3 PostgreSQL предоставляет низкоуровневый интерфейс, используя который можно реализовать надежное копирование (с использованием функций pg_start_backup, pg_stop_backup).

4 Эти функции использует и утилита pg_basebackup, и другие сторонние средства резервного копирования.

Контрольная точка (checkpoint)

- *Контрольная точка* – это механизм, который гарантирует, что данные и журналы, которые накоплены в памяти (и кэшированы на диске), периодически записываются на диск
- Этот механизм помогает минимизировать потерю данных при сбое и уменьшает время восстановления после перезапуска
- Контрольные точки происходят автоматически на основе параметров конфигурации (checkpoint_timeout и max_wal_size)
- Контрольные точки можно инициировать вручную, используя команду `CHECKPOINT`

1 В PostgreSQL процесс контрольной точки представляет собой механизм, который гарантирует сохранность данных в системе.

2 Во время контрольной точки происходит:
   - синхронизация журнальных записей: все журнальные записи, которые до этого момента были сохранены в буфере, записываются (flush) на диск. Эта операция гарантирует, что все записи о транзакциях, произошедших до контрольной точки, будут на диске, даже если система внезапно упадет;
   - синхронизация данных буфера: все «грязные» страницы (т. е. страницы, которые были изменены, но еще не записаны на диск) из общего буфера записываются на диск;
   - обновление файла контрольной точки: файл контрольной точки обновляется так, чтобы указать, что все транзакции, произошедшие до данной контрольной точки, записаны на диске;
   - высвобождение старых журнальных файлов: после обновления файла контрольной точки старые файлы журнала, которые больше не нужны для восстановления, можно безопасно удалить.

3 Процесс контрольной точки важен для восстановления после сбоя, поскольку упрощает процедуру восстановления, сокращая объем данных, который должен быть просмотрен в журнальных файлах.

4 Файл контрольной точки хранится в подкаталоге *pg_wal*, внутри основного каталога данных PostgreSQL.

Сравнение горячего и холодного резервного копирования

Характеристики	Холодное резервное копирование		Горячее резервное копирование
	при корректно остановленном сервере	на сервере, остановленном в результате сбоя	
Работа сервиса PostgreSQL во время копирования файлов данных	остановлен		работает
Нужны ли для восстановления копии журналов предзаписи?	нет	да, нужны, с момента последней контрольной точки	да, нужны, за период копирования

1 Резервная копия, созданная при холодном резервном копировании, содержит согласованные данные:

- если сервер был остановлен корректно;
- если содержит все необходимые для восстановления журналы (например, если используется снимок данных средствами операционной системы).

2 Если копия создается при работающем сервере (просто так копировать файлы нельзя, требуются настройка и определенные действия), она называется «горячей».

3 Настройка для создания горячей резервной копии сложнее. При горячем резервировании копия ФС будет содержать несогласованные данные.

4 Для приведения данных в согласованное состояние требуются копии журналов предзаписи, сгенерированных за время копирования файлов.

Утилита pg_basebackup. Создание автономной резервной копии

- Используется для создания горячей резервной копии
- Создает копию, содержащую файлы данных, и WAL — автономную копию
- Процесс резервного копирования:
  - подключение к серверу по протоколу репликации
  - создание контрольной точки
  - копирование файлов из ФС в указанный каталог
  - копирование сегментов WAL, сгенерированных за время копирования ФС

1 Процедура создания копии не влияет на работу других клиентов БД.

2 Утилита pg_basebackup создаёт точную копию файлов кластера, автоматически включая режим резервного копирования и завершая его.

3 Копия создаётся через обычное подключение к PostgreSQL, при этом используется протокол репликации.

4 Соединение с сервером должен устанавливать пользователь с правом REPLICATION или суперпользователь.

5 В pg_hba.conf должно разрешаться подключение для репликации.

6 Копия созданная с помощью утилиты pg_basebackup называется автономной, поскольку содержит в себе всё необходимое для восстановления.

7 Для восстановления требуется развернуть резервную копию и запустить сервер.

При необходимости будет выполнено восстановление согласованности с помощью имеющихся файлов WAL.

8 Подробнее: https://www.postgresql.org/docs/11/app-pgbasebackup.html

Журнал предзаписи. Параметр wal_level (enum)

- Параметр *wal_level* определяет, как много информации записывается в WAL
- При значении *minimal* в журнал записывается только информация, необходимая для восстановления после сбоя или аварийного отключения
- Значение по умолчанию *replica* – добавляет в WAL то, что необходимо для архивирования WAL, а также информацию, необходимую для выполнения запросов на ведомом сервере в режиме «только чтение»
- Значение *logical* добавляет информацию, требуемую для поддержки логического декодирования

1 Каждый последующий уровень включает всю информацию, записываемую на всех предыдущих.

2 Задать этот параметр можно только при запуске сервера.

3 Подробнее о настройке WAL: https://www.postgresql.org/docs/11/wal-configuration.html

4 Подробнее параметре WAL level: https://www.postgresql.org/docs/11/runtime-config-wal.html

Протокол потоковой репликации

- Используется для репликации и резервного копирования, в том числе для получение потока журнальных записей
- Подключение по протоколу репликации обслуживается процессом *wal sender*
- Минимальное значение параметра *wal_level = replica*
- Для подключения по протоколу потоковой репликации необходимо:
  - роль с атрибутом REPLICATION или суперпользователь в PostgreSQL
  - разрешение на локальное подключение по протоколу репликации без прохождения аутентификации конфигурационном файле pg_hba.conf

1 Для сохранения файлов WAL, сгенерированных сервером за время копирования файлов, утилита *pg_basebackup* подключается к серверу по специальному протоколу репликации.

2 Данный протокол используется не только для репликации, но и для резервного копирования.

3 Протокол позволяет получать поток журнальных записей параллельно с копированием файлом.

4 Число одновременно работающих процессов wal_sender ограничено значением параметра max_wal_senders.

5 При уровне minimal такие команды, как `CREATE TABLE AS SELECT`, `CREATE INDEX`, `COPY FROM` не попадают в журнал. Данные о них сразу записываются на диск.

Этого недостаточно для восстановления из горячей копии.

6 Для того, чтобы подключение по протоколу репликации было возможно, необходимо следующее:

- роль, в контексте безопасности которой будет работать утилита pg_basebackup:
  - должна обладать атрибутом REPLICATION или обладать правами суперпользователя в PostgreSQL.

- ей должно быть выдано разрешение на локальное подключение по протоколу репликации без прохождения аутентификации в конфигурационном файле pg_hba.conf.

```
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     trust
host    replication     all             127.0.0.1/32            md5
host    replication     all             ::1/128                 md5
```

- Параметр max_wal_senders ограничивает число одновременно работающих процессов wal_sender, обслуживающих подключения по протоколу репликации. Его значение должно быть достаточно большим.

7 Значения, установленные по умолчанию в PostgreSQL 11 в Astra Linux позволяют использовать репликацию:

```
postgres=# SELECT name, setting
postgres-# FROM pg_settings
postgres-# WHERE name IN ('wal_level','max_wal_senders');
      name       | setting
-----------------+---------
 max_wal_senders | 10
 wal_level       | replica
(2 rows)
```

Подробнее о протоколе потоковой репликации:
https://www.postgresql.org/docs/11/protocol-replication.html

Слот репликации

- *Слот репликации* в PostgreSQL – это структура данных, которая сохраняет информацию о прогрессе репликации БД
- Каждому подключающемуся репликатору назначается свой слот репликации
- Репликационные слоты используются для определения, какие данные из журнала предзаписи еще необходимы для репликации и не должны быть удалены
- Наличие слота не позволит серверу удалять файл WAL до тех пор, пока клиент не получит все записи из файла — сегмента WAL

1 Использование слотов репликации позволяет PostgreSQL эффективно управлять своими ресурсами и гарантировать, что все работающие реплики получат необходимые для них данные, несмотря на различия в скорости работы или временном отставании.

2 Если поток журнальных записей идет через слот, то слот фиксирует, какие записи уже были переданы клиенту.

3 При использовании слота репликации, если клиент отключится, а затем через какое-то время снова подключится — он продолжит получать журнальные записи с того момента, на котором остановился.

4 При отключении клиента файлы журнала будут накапливаться на сервере и могут занять все свободное место.

5 Каждый созданный слот следует добавлять в мониторинг (отслеживать через представление pg_replication_slots) и своевременно удалять ненужные слоты.

6 Общее количество слотов, которые могут быть созданы, ограничено конфигурационным параметром max_replication_slots.

7 Подробнее о слотах репликации: https://www.postgresql.org/docs/11/warm-standby.html#STREAMING-REPLICATION-SLOTS

Особенности работы pg_basebackup

- pg_basebackup использует два подключения по протоколу репликации:
  - одно для передачи данных
  - одно для передачи потока журнальных записей, которые генерирует работающий сервер во время копирования
- Значение параметра max_wal_senders должно быть не менее, чем 2
- PostgreSQL по умолчанию использует *временный* слот репликации, который существует только на время соединения и удаляется при завершении работы pg_basebackup

1 В параметрах утилиты pg_basebackup можно указать и имя обычного слота, который должен существовать на момент запуска утилиты или создаваться ей.

2 Подробнее: https://www.postgresql.org/docs/11/app-pgbasebackup.html

Синтаксис pg_basebackup

- `pg_basebackup [параметр…]`
- Важные параметры:
  - `-D каталог (--pgdata=каталог)` – целевой каталог для записи данных
  - `-F формат (--format=формат)`:
    - `p (plain)` – записывает выводимые данные в файлы, сохраняя структуру каталогов данных и табличных пространств как на исходном сервере (используется по умолчанию)
    - `t (tar)` – записывает в целевой каталог файлы в формате tar

1 При использовании параметра `-D каталог (--pgdata=каталог)` целевой каталог для записи данных будет создан автоматически (при необходимости и родительские каталоги).

Каталог может быть создан заранее, но должен быть пустым. Если каталог не будет пуст, возникнет ошибка.

2 При использовании значения формата plain выводимые данные записываются в файлы, сохраняя структуру каталогов данных и табличных пространств как на исходном сервере.

Если в кластере нет дополнительных табличных пространств, вся база будет помещена в заданный каталог.

3 Использовать формат plain рационально, если цель создания резервной копии – немедленно развернуть из нее новый кластер.

4 Если в кластере БД созданы другие табличные пространства, то основной каталог хранения данных будет помещён в целевой каталог, а все остальные табличные пространства — в те же абсолютные пути, в которых они располагаются на исходном сервере.

Для изменения этих путей можно использовать параметр `--tablespace-mapping`.

5 Использовать формат tar рационально, если копия выполняется в рамках обычной политики резервного копирования.

6 При использовании значения формата tar содержимое основного каталога данных будет записано в файл base.tar, а каждое дополнительное табличное пространство — в отдельный файл, содержащий в имени OID этого пространства.

Если в качестве имени целевого каталога задано *«-» (минус)*, содержимое tar будет записано в устройство стандартного вывода, что позволяет, например, передать его программе gzip. Это возможно, только если в кластере нет дополнительных табличных пространств, и не используется трансляция WAL (передача данных журнала WAL на другой сервер в режиме реального времени).

7 С полным синтаксисом можно ознакомиться на сайте: https://www.postgresql.org/docs/11/app-pgbasebackup.html

Пример создания резервной копии с помощью pg_basebackup и подмены файлов целевого кластера

- `sudo pg_basebackup -U iivanov -h localhost -D /home/backups/basebackup`

  – создание базовой резервной копии в каталоге basebackup
- Восстановление будет проходить в кластер *replica*. Он должен быть остановлен в момент замещения его файлов данными из резервной копии
- ```
  sudo rm -rf /var/lib/postgresql/11/replica
  sudo mv /home/backups/basebackup/
  /var/lib/postgresql/11/replica
  ```

  – удаление файлов данных кластера БД replica и перенос в каталог данных содержимого резервной копии

1 Файлы кластера должны принадлежать пользователю *postgres*, поэтому после копирования данных в каталог /var/lib/postgresql/11/replica следует изменить владельца папки:

```
sudo chown -R postgres:postgres
/var/lib/postgresql/11/replica
```

2 Поскольку базовая резервная копия включает файлы WAL, созданные за период резервного копирования, после запуска кластера БД (в данном случае replica) данные будут приведены в согласованное состояние, соответствующее времени создания резервной копии.
```

Непрерывное архивирование журналов предзаписи

- *Файловый архив*
  - сегменты WAL копируются в архив после заполнения
  - реализуется через настройку параметров сервера
  - файл сегмента WAL попадает в архив только тогда, когда сервер переключается на запись в другой файл
- *Потоковый архив*
  - в архив постоянно записывается поток журнальных записей
  - требуются внешние средства (утилита pg_receivewal)

1 В процессе работы PostgreSQL ведёт *журнал предзаписи WAL* (write-ahead logging), который расположен в подкаталоге pg_wal/ каталога с данными кластера баз данных.

В этот журнал записываются все изменения, вносимые в файлы данных.

2 Журнал существует для безопасного восстановления после краха сервера: если происходит крах, целостность СУБД может быть восстановлена в результате «воспроизведения» записей, зафиксированных после последней контрольной точки.

3 При создании резервной копии можно сочетать резервное копирование на уровне файловой системы с копированием файлов WAL.

При восстановлении данных можно восстановить копию файлов, а затем воспроизвести журнал из скопированных файлов WAL. Таким образом можно восстановить кластер БД на состояние любого момента времени, для которого сохранены журналы.

4 Такое восстановление называется *восстановлением на момент времени* (Point-in-Time Recovery, PITR).

Создание файлового архива журналов предзаписи

- Архивирование реализуется фоновым процессом *archiver*
- Для включения нужно в postgresql.conf указать параметры:

```
archive_mode = on
archive_command = 'cp %p /path/to/archive/directory/%f'
```

- После заполнения сегмента WAL вызывается команда копирования
- Параметр *archive_timeout* позволяет указать максимальное время переключения на новый сегмент WAL
- Переключиться на новый сегмент можно с помощью функции *pg_switch_wal()*

1 Параметр archive_timeout позволяет указать максимальное время переключения на новый сегмент WAL.

2 Переключение на новый сегмент по значению параметра archive_timeout позволяет при невысокой активности сервера сохранять журналы с нужной периодичностью.

3 Обычно архив размещают на другом сервере.

4 Скорость архивирования должна быть не ниже средней скорости, с которой сервер генерирует записи WAL.

Обычно работа продолжается, даже если процесс архивирования немного отстаёт.

5 Если архивирование отстаёт значительно, это приводит к увеличению объёма данных, которые могут быть потеряны в случае аварии.

При этом каталог *pg_wal/* будет содержать большое количество ещё не заархивированных файлов-сегментов, которые в конце концов могут занять всё доступное дисковое пространство.

Поэтому рекомендуется контролировать процесс архивации и следить за тем, чтобы он выполнялся как задумано.

6 Подробнее: https://www.postgresql.org/docs/11/continuous-archiving.html

Команда архивирования archive_command

- `archive_command = 'cp %p /path/to/archive/directory/%f'`, где
  - `%p` – заменяется полным именем файла WAL
  - `%f` – заменяется именем файла WAL
- Должна скопировать указанный файл в архив
- Не должна перезаписывать существующие файлы
- Архив может быть организован любым образом
- Команда архивирования будет запущена от имени того же пользователя, от имени которого работает сервер PostgreSQL

1 После изменения конфигурации (указания команды) следует повторно считать параметры.
2 Архив может быть размещен как локально, так и удаленно.
3 Выполнение команды копирования должно завершаться с кодом 0, чтобы PostgreSQL рассматривал процедуру архивации как *успешную*. Если команда не завершается с exit code 0, сегмент может повторно добавляться в очередь для архивации.
4 В archive_command можно указать любую команду, но она должна завершаться со статусом 0 только в случае *успеха*.
5 Рекомендуется протестировать команду архивирования, чтобы убедиться, что она не перезаписывает существующие файлы и возвращает ненулевое состояние в этом случае.
6 Статус архивации можно отслеживать через представление *pg_stat_archiver*.
7 Сбор сообщений удобно вести с помощью процесса *logging_collector,* так как в этом случае сообщения об ошибках при выполнения archive_command будут попадать в журнал сервера.
8 Если требуется выполнять несколько команд, то их можно оформить как *сценарий* и указать *имя* сценария в качестве команды копирования.

Примеры команд архивирования

- С проверкой наличия файлов и сжатие архивируемых сегментов:

```
archive_command = 'test ! -f /home/WAL_file_arch/%f &&
gzip <%p >/home/WAL_file_arch/%f';
```

- С вызовом сценария:

```
archive_command = 'archive.sh "%f" "%p"';
```

1 Подробнее о команде архивирования: https://www.postgresql.org/docs/11/runtime-config-wal.html#GUC-ARCHIVE-COMMAND

Создание потокового архива журналов предзаписи

- Утилита *pg_receivewal*:
  - копирует данные из текущего сегмента WAL
  - используется для записи сегментов в архив по протоколу потоковой репликации
  - использует слот репликации, чтобы гарантированно не потерять записи
  - формирует файлы записывает их в указанный каталог
  - не запускается автоматически (как сервис)
  - требует дополнительного мониторинга
  - обычно запускается на отдельном сервере
  - будет получать записи бесконечно, если не указан ключ: `--endpos=lsn`

1 Сегменты, которые не заполнены до конца и еще находятся в процессе заполнения сегменты имеют префикс *.partial*.

2 Синхронизация с файловой системой по умолчанию происходит только при закрытии файла-сегмента.

3 Архивирование всегда начинается с начала сегмента, следующего за последним уже полностью заполненным сегментом, который присутствует в архиве.

4 Если архив пуст (первый запуск архивирования), архивирование начинается с начала текущего сегмента.

5 Если указан ключ `--endpos=lsn`, утилита pg_receivewal завершит работу по получении записи с заданным LSN.

6 Подключение обрабатывается отдельным процессом *wal sender*.

7 Необходимо, чтобы параметра max_wal_senders имел достаточно большое значение.

8 Утилита записывает данные, не дожидаясь получения всего сегмента.

9 Подробнее: https://www.postgresql.org/docs/11/app-pgreceivewal.html

Создание резервной копии при настроенном непрерывном архивировании журналов предзаписи

- При настроенном архивировании журнала предзаписи в базовой резервной копии файлы журнала архивировать не обязательно
- При восстановлении их можно получить из архива
- Утилиту pg_basebackup можно использовать с параметром `--wal-method=none`
- Например,

```
pg_basebackup -U ssidorov -h localhost -D
/home/basebackup_no_WAL --wal-method=none
```

1 Можно использовать утилиту pg_basebackup и без параметра — `wal-method=none`.

В этом случае файлы журнала будут занимать дополнительны объем дискового пространства при размещении базовой копии, но не будут препятствовать восстановлению.

2 Наличие журнальных файлов в базовой копии позволит выполнить восстановление, если архив журналов окажется недоступен.

Восстановление с использование резервных копий журналов предзаписи: параметры

- Процессом восстановления управляют:
  - файл *tablespace_map* – содержит пути для табличных пространств
  - файл метки *backup_label* — содержит информацию о резервной копии (в т.ч. название сегмента WAL и позицию в нем, с которой надо начинать восстановление)
  - файл *recovery.signal* – создается вручную
  - параметры

1 Файл tablespace_map генерируется при создании базовой резервной копии.

2 Если в резервной копии в каталоге pg_tblspc нет символических ссылок (в формате tar), то они будут созданы при запуске сервера на основании информации в tablespace_map.

3 Файл метки backup_label генерируется автоматически при создании базовой резервной копии и содержит название, время создания копии, номер линии времени, название сегмента WAL и позицию в нем, с которой надо начинать восстановление.

4 Наличие файла recovery.signal дает указание серверу перейти в режим восстановления.

Содержимое файла игнорируется.

5 Если recovery.signal отсутствует, то PostgreSQL считает, что он выполняет автоматическое восстановление после сбоя.

6 Если же файл есть, сервер понимает, что происходит управляемое пользователем восстановление из резервной копии.

Восстановление с использование резервных копий журналов предзаписи: шаги

- Шаги:
  - создать файл recovery.conf
  - указать в нем параметр restore_command, например,
    `restore_command = 'cp %p /path/to/archive/directory/%f'`
  - развернуть резервную копию
  - создать сигнальный файл recovery.signal
  - запустить кластер БД

1 Команда, указанная через параметр *restore_command* копирует файлы WAL из архива в каталог сервера.

2 При наличии сигнального файла *recovery.signal* сервер при старте входит в режим управляемого восстановления. Содержимое файла игнорируется.

3 Процесс восстановления:
  - Развернуть файлы из базовой резервной копии. Развертывание может выполняться на тот же сервер, в тот же экземпляр, в другой экземпляр, в экземпляр на другом хосте.
  - Создание файла recovery.signal. Наличие файла recovery.signal дает указание серверу перейти в режим восстановления. Содержимое файла игнорируется.
  - Запуск сервера.
  - Считывание сегментов WAL из архива с помощью команды restore_command.
  - Применение сегментов WAL из архива. Сегменты WAL, считываются из каталога pg_wal.

4 Примечание:

При файловой архивации последний сегмент WAL основного сервера не попадет в архив, поскольку при создании резервной копии в него еще идет запись.

Его можно вручную скопировать в каталог pg_wal на резервном сервере.

В случае сбоя архивирования таких сегментов может оказаться и несколько.

5 Если при запуске сервера оказывается, что какой-то из последовательности сегментов WAL отсутствует, сервер пытается его скопировать из архива.

6 В восстановленную копию не попадут данные только из заключительного сегмента (сегментов), если его не удалось скопировать.

7 По умолчанию применяются все доступные записи из журналов WAL.

8 Если требуется прервать восстановление на какой-то точке, чтобы изменения, внесенные в кластер позже, не применились к восстановленной копии, указывается целевая точка восстановления.

9 После восстановления резервный сервер переходит в обычный режим работы.

Восстановление до определенной точки (PITR - Point-in-time recovery)

- По умолчанию восстановление производится вплоть до окончания журнала WAL
- Есть возможность остановить восстановление в определенный момент, в более ранней точке
- Достигается за счет указания одного из параметров `recovery_target_*`
- В версии 11 PostgreSQL параметры восстановления указываются в отдельном файле recovery.conf

1 По умолчанию восстановление производится вплоть до окончания журнала WAL.

2 Если не задана цель восстановления (один из параметров `recovery_target_*`), базы данных будут восстановлены максимально близко к моменту сбоя. Однако процесс восстановления можно остановить в любой другой момент, указав целевую точку.

3 Чтобы остановить процесс восстановления, можно использовать один из следующих параметров:
   - `recovery_target`
   - `recovery_target_lsn`
   - `recovery_target_name`
   - `recovery_target_time`
   - `recovery_target_xid.`

4 Если в конфигурационном файле устанавливаются сразу несколько этих параметров, выдаётся ошибка.

Возможные параметры и значения

- `recovery_target = 'immediate'` — остановит восстановление, как только будет достигнуто целостное состояние
- `recovery_target_name = 'имя'` — восстановление до именованной точки, созданной с помощью функции pg_create_restore_point()
- `recovery_target_time = 'время'*` — до момента времени
- `recovery_target_xid = 'xid'*` — до указанного идентификатора транзакции
- `recovery_target_lsn = 'lsn'*` — до LSN позиции в журнале предзаписи

  * – окончательная точка, до которой будет выполняться восстановление определяется в зависимости от значения параметра recovery_target_inclusive
- `recovery_target_inclusive = on|off` — включать ли указанную точку

1 Параметр *recovery_target= 'immediate'* указывает, что процесс восстановления должен завершиться, как только будет достигнуто целостное состояние, т. е. как можно раньше.

Технически это строковый параметр, но значение 'immediate' — единственно допустимое в данный момент.

2 Параметр *recovery_target_name (string)* указывает именованную точку восстановления (созданную с помощью pg_create_restore_point()), до которой будет производиться восстановление.

Это полезная опция, если заранее известно, что может потребоваться восстановление.

3 Параметр *recovery_target_time (timestamp)* указывает точку времени, вплоть до которой будет производиться восстановление.

4 Параметр *recovery_target_xid (string)* указывает идентификатор транзакции, вплоть до которой необходимо произвести процедуру восстановления.

5 Параметр *recovery_target_lsn (pg_lsn)* - указывает LSN позиции в журнале предзаписи, до которой должно выполняться восстановление.

Этот параметр принимает значение системного типа данных pg_lsn.

6 Окончательно точка, до которой будет выполняться восстановление для приведенных выше трех параметров (recovery_target_time, recovery_target_xid, recovery_target_lsn) определяется в зависимости от значения параметра *recovery_target_inclusive*.

Этот параметр указывает на необходимость остановки восстановления сразу после (при значении *true* или *on*), либо до (при значении *false* или *off*) достижения целевой точки.

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Настройки восстановления

- Команда восстановления
  - `restore_command` — копирует файл %f из архива в %p
  - должна завершаться со статусом 0 только при успехе
- По окончании восстановления (если задана цель) действие задает параметр *recovery_target_action*:
  - *pause* (по умолчанию) — администратор может завершить восстановление через pg_wal_replay_resume() или задать новую цель и продолжить применение журнальных записей
  - *promote* — сервер начинает принимать подключения
  - *shutdown* — сервер останавливается

1 Для восстановления необходимо задать параметр *restore_command*, который определяет команду восстановления.

2 Команда должна скопировать файл из архива в каталог pg_wal и завершаться со статусом 0 только при *успехе*.

3 Если задана цель, параметр recovery_target_action позволяет выбрать состояние, в котором окажется сервер по окончании восстановления.

4 По умолчанию параметр имеет значение *pause*.

При этом значении:

- процедура восстановления приостанавливается;
- администратор может:
  - выполнить запросы к данным и решить, корректно ли была выбрана цель;
  - завершить восстановление, через вызов функции *pg_wal_replay_resume();*
  - задать новую цель, перезапустить сервер и продолжить применение журнальных записей.

5 При значении *promote*, сервер сразу сможет принимать запросы на подключения.

6 При значении *shutdown* сервер будет остановлен. При этом значении файл recovery.signal не удаляется.

Завершение восстановления

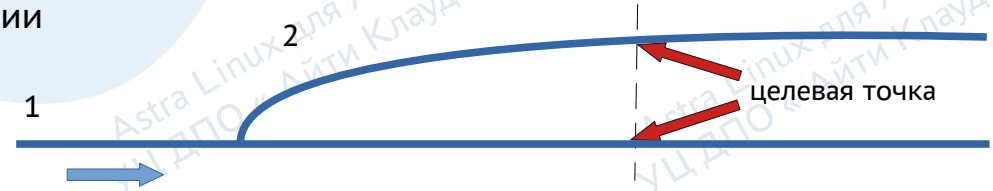
- Процесс *startup* (процесс запуска) завершается
- Процесс *postmaster* запускает остальные служебные процессы, необходимые для работы экземпляра
- Сервер начинает работать в обычном режиме
- Файл *recovery.signal* удаляется
- backup_label переименовывается в *backup_label.old*
- Если сервер не может запуститься, причину ошибки можно узнать из журнала сообщений сервера (например, в параметрах была допущена ошибка)
- После исправления причины ошибки сервер следует запустить еще раз

1 Если сервер не может запуститься:
   - в ОС будут отсутствовать процессы (и не будет файла postmaster.pid);
   - recovery.signal не будет удален.

2 *Примечание:*
если на сервере-источнике было настроено непрерывное архивирование, то на резервном сервере его надо либо отключить, либо перенаправить в другой архив.

Линии времени

- Когда завершается восстановление из архива, создаётся новая линия времени, позволяющая идентифицировать последовательность записей WAL, добавленных после этого восстановления
- `recovery_target_timeline = 'latest'` – восстановление по последней линии (по умолчанию)
- `recovery_target_timeline = 'current'` – восстановление по текущей линии
- `recovery_target_timeline = '1'` – восстановление по указанной линии



1 После восстановления на момент в прошлом сервер начинает генерировать новые сегменты WAL, которые будут пересекаться с сегментами сформированными до восстановления.

2 Чтобы не потерять сегменты и, вместе с ними, возможность восстановления на другой момент времени, PostgreSQL использует понятие линии времени.

3 Просмотреть значение линии времени можно в метке резервной копии:

```
root@astra:~# cat /home/basebackup_no_WAL/backup_label
START WAL LOCATION: 0/79000028 (file 000000010000000000000079)
CHECKPOINT LOCATION: 0/79000060
BACKUP METHOD: streamed
BACKUP FROM: master
START TIME: 2023-11-17 09:42:17 MSK
LABEL: pg_basebackup base backup
START TIMELINE: 1
```

4 Номер линии времени включается в имя файлов-сегментов WAL, так что файлы новой линии времени не перезаписывают файлы WAL, сгенерированные предыдущими линиями времени.

5 Посмотреть, какая линия времени указывается в именах-сегментов WAL можно с помощью запроса
`SELECT pg_walfile_name(pg_current_wal_lsn());`

6 Это позволяет архивировать много различных линий времени.

7 Когда неизвестно какую точку времени выбрать для восстановления, приходится проводить восстановление методом проб и ошибок, пока не будет найден лучший момент для ответвления от старой истории.

8 Благодаря линиям времени, можно вернуться к любому предыдущему состоянию, включая состояния в ветках линий времени, покинутых ранее.

9 Каждый раз, когда образуется новая линия времени, PostgreSQL создаёт файл «истории линии времени», показывающий, от какой линии времени ответвилась данная и когда.

Эти файлы истории нужны, чтобы система могла выбрать правильные файлы-сегменты WAL при восстановлении из архива, содержащего несколько линий времени.

10 Таким образом, они помещаются в область архивов WAL так же, как и файлы сегментов WAL.

11 *Файлы истории*:

- представляют собой небольшие текстовые файлы, так что они не занимают много места и их можно сохранять неограниченно долго (в отличие от файлов сегментов, имеющих большой размер);
- размещаются в каталоге pg_wal, именуются N.history.

12 В файл истории можно добавлять комментарии, собственные заметки о том, как и почему была создана эта конкретная линия времени.

13 По умолчанию восстанавливается та же линия времени, которая была текущей при создании базовой резервной копии.

14 Если требуется восстановить состояние на какой-либо из дочерней линии времени, (то есть, требуется вернуться к некоторому состоянию, которое тоже было получено в результате попытки восстановления), необходимо указать идентификатор целевой линии времени в файле recovery.conf, название параметра - *recovery_target_timeline*.

15 На слайде изображена схема линий времени для кластера БД, который был восстановлен из резервной копии. Причем восстановление было остановлено (или по моменту времени, или по номеру транзакции и т.п.):

- первая линия имеет номер 1;
- произошло восстановление на момент времени в прошлом. Это положило начало линии времени 2.

16 Если выполнить восстановление с указанием параметра *recovery_target_timeline = 'latest'*, на точке ветвления применение записей WAL продолжится по второй линии (т. к. она более поздняя).

17 Если же указать параметр *recovery_target_timeline = '1'*, то восстановление продолжится по первой (указанной) линии.

18 Если же указать параметр *recovery_target_timeline = '2'*, то восстановление продолжится по второй (указанной) линии.

19 Если указать *recovery_target_timeline = 'current'*, то восстановление пойдет по текущей линии (линии 1).

20 Подробнее: https://www.postgresql.org/docs/11/continuous-archiving.html#BACKUP-TIMELINES

Очистка архива

- Утилита `pg_archivecleanup`
- Синтаксис:

  ```
  pg_archivecleanup [параметр...] расположение_архива
  старейший_сохраняемый_файл
  ```

- В результате работы утилиты будут удалены все предшествующие сегменты

1 Архив со временем разрастается, поэтому требуется периодическая очистка от ненужных файлов.

2 Трудность заключается в том, чтобы определить позицию в архиве, начиная с которой сегменты WAL *можно* удалять.

3 Если архив используется только для восстановления одного резервного экземпляра, можно выполнять очистку сразу после восстановления.

4 Если архив должен поддерживать восстановление из нескольких базовых резервных копий, то для автоматизации очистки придется написать собственный скрипт.

5 Подробнее: https://www.postgresql.org/docs/11/pgarchivecleanup.html

Вопросы для проверки

1 При каком типе физического резервного копирования копирование файлов выполняется при остановленном экземпляре сервера?

2 Какая утилита используется для создания автономной резервной копии?

3 Какая утилита применяется для создания потокового архива журналов предзаписи?

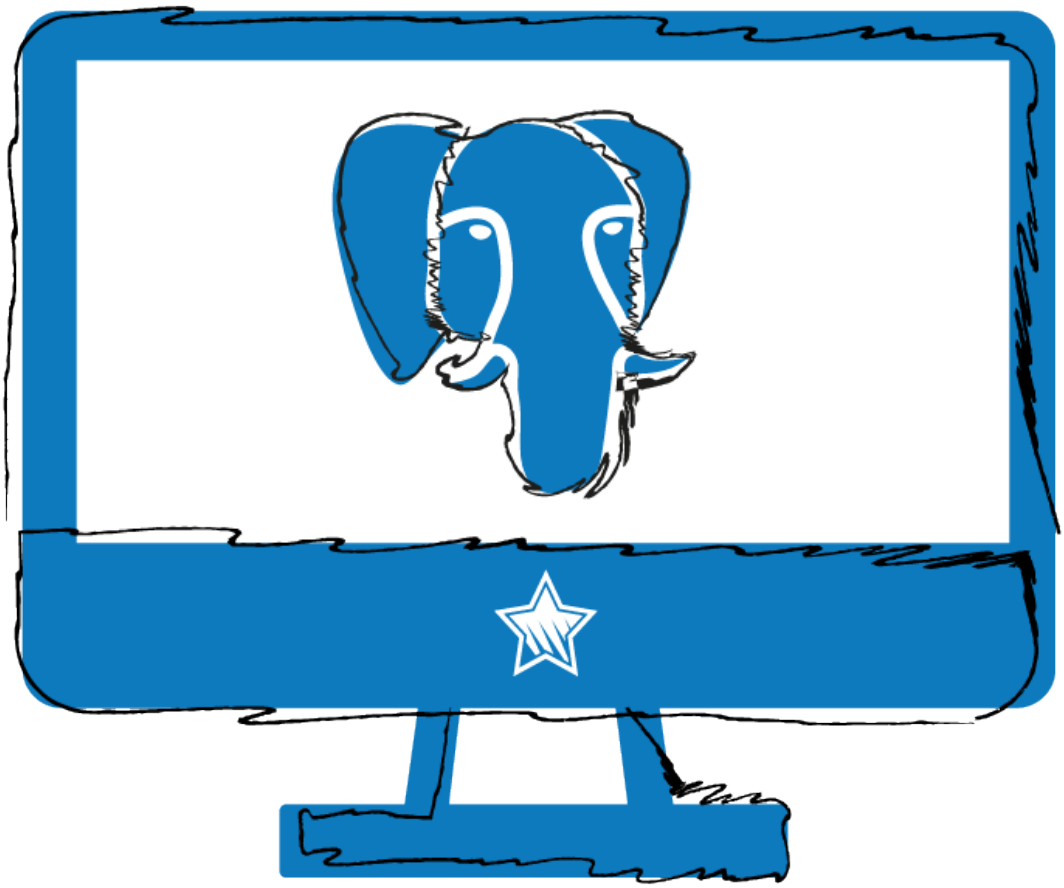
Практическая работа

Оценка времени выполнения: 120 мин.

- Выполните задания модуля 9 из Сборника практических заданий.

Модуль 10

Регламентные работы



ASTRA LINUX®

Модуль 10.
Регламентные работы

Краткое содержание:

- Возможности команды VACUUM.
- Регулирование нагрузки в процессе очистки.
- Параллельная очистка индексов.
- Команды для оптимизации данных: CLUSTER, REINDEX, TRUNCATE.
- Предотвращение переполнения счетчика транзакций.
- Сбор статистики по базе данных.

Основные знания и умения:

- знать возможности команды VACUUM;
- знать команды для оптимизации данных: CLUSTER, REINDEX, TRUNCATE;
- уметь собирать статистику.

Оценка знаний и умений:

- Практическая работа, контрольные вопросы.

Дополнительная литература:

- Отсутствует.

Цели модуля 10. Регламентные работы

- Возможности команды VACUUM: очистка, полная очистка, обновление статистики, удаление элементов индекса, указывающих на «мёртвые» кортежи
- Регулирование нагрузки в процессе очистки
- Параллельная очистка индексов
- Команды для оптимизации данных: `CLUSTER`, `REINDEX`, `TRUNCATE`
- Предотвращение переполнения счетчика транзакций
- Сбор статистики по базе данных

Регламентные работы: очистка

- Очистка обеспечивает:
  - высвобождение дискового пространства
  - обновление статистики планировщика
  - обновление карты видимости
  - предотвращение ошибок из-за зацикливания счётчика транзакций

1 Базы данных PostgreSQL требуют периодического проведения процедуры обслуживания, которая называется *очисткой*.

2 Во многих случаях очистку достаточно выполнять с помощью демона автоочистки.

Ненужные версии строк

- *MVCC (Multi-version Concurrency Control)* – управление параллельным доступом посредством многоверсионности
- Версии строк возникают вследствие работы этого механизма
- При внесении изменений в строку он создает новую «версию» строки
- Ненужные версии строк появляются при:
  - выполнении операций UPDATE, DELETE
  - при откате транзакций

1 В PostgreSQL «версия строки» относится к механизму, который называется MVCC (Multi-version Concurrency Control – управление параллельным доступом посредством многоверсионности).

Это система контроля параллельности, которая позволяет каждой транзакции работать со «снимком» базы данных в определенный момент времени.

2 При внесении изменений в строку, PostgreSQL не перезаписывает старые данные. Вместо этого он создает новую «версию» этой строки. Это помогает в предотвращении блокировок при одновременном доступе множества транзакций к одним и тем же данным.

3 Это может привести к накоплению «мертвых кортежей» или неиспользуемых версий строк, что, в конечном итоге, требует регулярной «очистки» с помощью процесса очистки (VACUUM).

4 *Ненужные версии строк* («мёртвые кортежи» или «dead tuples») появляются в PostgreSQL в следующих ситуациях:

- при выполнении операции UPDATE: при обновлении строки создаётся новая копия этой строки (версия), оставляя старую версию в таблице;
- при выполнении операции DELETE: когда строка удаляется, она помечается как ненужная, но остаётся в таблице;
- транзакции, которые не были успешно завершены (откаты).

Если транзакция обновляет данные и затем неудачно завершается, изменённые данные остаются как ненужные версии строк.

5 Накопление таких мёртвых кортежей может привести к замедлению работы базы данных и увеличению использования дискового пространства.

6 Для очистки этих ненужных версий строк используются процесс VACUUM (очистка) и AUTOVACUUM (автоочистка).

Карта видимости (visibility map)

- *Карта видимости* – это специальная структура данных, которая помогает СУБД определить, какие страницы содержат некоторые не видимые данным снимком строки
- В карте видимости отмечены страницы, содержащие только достаточно старые версии строк, которые гарантированно видимы во всех снимках
- Каждый бит в этой карте представляет собой страницу данных в таблице: если бит установлен в 1, это значит, что все строки на этой странице видимы всем транзакциям, и можно безопасно пропустить проверку видимости строк на этой странице при последующем чтении

1 Пропуск проверки видимости строк обеспечивает более быструю работу с индексами и увеличивает производительность выполнения запросов SELECT .

2 Карта видимости:

- ускоряет процесс очистки, позволяя ему пропускать страницы, которые не нуждаются в очистке;
- не позволяет определить, какие строки не видимы для данного снимка, а лишь даёт информацию, все ли строки на определённой странице видимы.

Карта свободного пространства (Free Space Map, FSM)

- *Карта свободного пространства* (Free Space Map, FSM) – это система, которая отслеживает доступное пространство в каждой отдельной таблице и индексе
- Используется для нахождения места при добавлении новой записи, чтобы избежать ненужного расширения файла данных
- FSM хранит в себе указатели на места в файлах данных таблиц, где есть свободное пространство для новых записей. Это позволяет PostgreSQL быстро находить место для записи новых данных, минимизируя ненужные операции записи на диск

1 FSM играет важную роль в поддержании производительности баз данных на высоком уровне. Однако, для больших баз данных, она может занимать значительное количество дискового пространства.

Очистка с помощью команды VACUUM

- Обрабатывает таблицу полностью, включая все созданные на ней индексы
- Освобождает место за счет удаления ненужных версий строк и указателей
- Не конфликтует с обычной активностью в системе
- Пропускает страницы, уже отмеченные в карте видимости
- Очищает индексные записи, ссылающиеся на очищенные версии строк
- Освобождает указатели
- Обновляет карту свободного пространства
- Обновляет карту видимости

1 Обработка с помощью команды VACUUM происходит в фоновом режиме, таблица при этом может использоваться для чтения и для изменения.

2 В процессе очистки нельзя выполнять команды `CREATE INDEX`, `ALTER TABLE` и др.

3 В процессе очистки из таблицы читаются только те страницы, в которых происходила какая-то активность. Для этого используется *карта видимости* (visibility map).

4 В процессе работы команды VACUUM обновляется и карта свободного пространства (free space map), в которой отражается наличие свободного места на страницах.

Синтаксис команды VACUUM

- `VACUUM [ ( параметр [, ...] ) ] [ таблица_и_столбцы [, ...] ]`
- Параметры:
  - `FULL` — полная очистка
  - `ANALYZE` — обновление статистики
  - `INDEX_CLEANUP` — удаление элементов индекса, указывающие на «мёртвые» кортежи
- Пример:
  `VACUUM (ANALYZE) sales;`
  - очистка одной таблицы sales, проведение её анализа для оптимизатора

1 При указании параметра FULL выполняется полная очистка. При выборе этого режима можно освободить больше пространства, но очистка выполняется гораздо дольше и запрашивает исключительную блокировку таблицы.

2 Этот режим также требует наличия дополнительного места на диске, так как он записывает новую копию таблицы и не освобождает старую до завершения операции. Обычно это следует использовать, только когда требуется высвободить значительный объём пространства, выделенного таблице.

3 При указании параметра ANALYZE:
- команда VACUUM обновляет статистику, которую использует планировщик для выбора наиболее эффективного способа выполнения запроса;
- обработка происходит в фоновом режиме и не мешает обычному использованию таблиц и индексов.

4 Параметр `INDEX_CLEANUP` указывает, что команда VACUUM должна удалить элементы индекса, указывающие на «мёртвые» кортежи

5 С полным синтаксисом команды VACUUM можно ознакомиться в документации PostgreSQL:

https://www.postgresql.org/docs/11/sql-vacuum.html

Регулирование нагрузки в процессе очистки

- Настройка выполняется через параметры в файле postgresql.conf
- Чередование работы и ожидания:
  - `vacuum_cost_limit` – количество условных единиц работы (credits)
  - `vacuum_cost_delay` – ожидание в мс
- Определение стоимости обработки страницы:
  - `vacuum_cost_page_hit = 1` – страница нашлась в кеше
  - `vacuum_cost_page_miss = 10` – страница прочитана с диска
  - `vacuum_cost_p age_dirty = 20` – чистая страница стала грязной

1 В процессе очистки идет интенсивная работа с таблицами и индексами.

Для сглаживания нагрузки на сервер можно в процессе очистки можно распределить действия по очистке во времени – выполнять очистку порциями, чередуя работу и ожидание.

2 По умолчанию эти параметры не используются.

3 Размер порции vacuum_cost_limit указывается в условных единицах (credits).

Может принимать значение от 1 до 10000.

4 Значение параметра vacuum_cost_delay может варьироваться от 0 до 100 мс.

5 Параметры vacuum_cost_page_hit, vacuum_cost_page_miss, vacuum_cost_page_dirty определяют стоимость обработки одной страницы в условных единицах (credits).

6 В лучшем случае стоимость будет равна vacuum_cost_hit.

7 *Чистые страницы* – это страницы в общих буферах базы данных (основной кеш PostgreSQL), которые точно соответствуют их дисковому представлению, т. е. они не были изменены с момента их считывания с диска или после последней записи на диск.

8 *Грязные страницы* – это страницы с изменениями (вставками/обновлениями/удалениями), которые не были записаны/сброшены на диск. Они представляют собой более новую версию данных по сравнению с тем, что сохранилось на диске.

9 Очень важно, чтобы грязные страницы периодически сбрасывались на диск (с контрольными точками), чтобы обеспечить долговечность данных и свести к минимуму вероятность потери данных в случае неожиданного завершения работы или сбоя.

Термин «грязный» означает необходимость «очистить» их путем синхронизации их содержимого с дисковым хранилищем.

10 Наличие грязных страниц - обычное явление при обработке транзакций, особенно в многопользовательской среде.

11 Основная причина существования «грязных» страниц заключается в том, чтобы ускорить процессы чтения и записи, отсрочив физическую запись изменений на диск.

Это также является важной составляющей механизма восстановления после сбоя, поскольку PostgreSQL сможет использовать журнал записи для восстановления не сохраненных на диск данных.

12 *Примечание:*
высокий уровень грязных страниц может указывать на потенциальные проблемы производительности или на нехватку ресурсов (например, проблемы с оперативной памятью или дисковым пространством).

13 Если страница нашлась в кеше и либо не изменилась, либо уже была грязной — стоимость работы с ней в процессе очистки составляет столько условных единиц, сколько указано в параметре vacuum_cost_page_hit.

14 Если страница была прочитана с диска и изменилась в результате очистки - стоимость работы с ней в процессе очистки составит vacuum_cost_page_miss + vacuum_cost_page_dirty условных единиц.

15 При настройке по умолчанию механизм регулирования нагрузки отключен.

Параллельная очистка индексов

- Каждый индекс таблицы, размер которого больше значения параметра min_parallel_index_scan_size (по умолчанию 512 Кбайт) во время очистки может обрабатывается только отдельным рабочим процессом
- Индексов должно быть больше одного
- Количество процессов:
  - не может превышать значение параметра max_parallel_maintenance_workers
  - может быть дополнительно ограничено явным указанием степени параллелизма при вызове команды VACUUM (PARALLEL n)

1 Очистка индексов может выполняться параллельно, при этом каждый индекс обрабатывается только одним рабочим процессом.

2 Если на таблице создано несколько достаточно больших индексов (с размером, превышающим значение параметра min_parallel_index_scan_size), для работы с каждым, соответствующим этому требованию индексом, запускается отдельный рабочий процесс.

Примечание:

Сканирование и очистка таблицы выполняются последовательно.

Полная очистка

- Запуск: VACUUM FULL
- Не совместима с другими операциями над таблицей (включая выборку данных)
- Полностью перестраивает таблицу и все ее индексы
- При работе требуется дополнительное место для новых файлов
- Блокирует всякую работу с таблицей
- Не предполагает регулярного использования

1 Обычная очистка не приводит к сокращению числа страниц данных (и к уменьшению файлов).

2 При обычной очистке:
   - внутри существующих страниц появятся свободные области, которые могут быть использованы для вставки новых строк или изменения существующих;
   - страница может быть освобождена, только если очищена полностью и расположена в конце файла. Такие страницы возвращаются операционной системе.

3 При выполнении полной очистки:
   - таблица и все ее индексы перестраиваются полностью с нуля (с учетом параметра fillfactor);
   - PostgreSQL последовательно перестраивает сначала таблицу, а затем и каждый из индексов;
   - для таблицы и индексов создаются новые файлы, а старые удаляются.

Команды для оптимизации данных

- Применяются к таблицам
- Блокируют работу с таблицей
- Удаляют старые файлы данных и создают новые
- Команда `CLUSTER`:
  - полностью перестраивает таблицу и ее индексы
  - физически упорядочивает версии строк в соответствии с одним из индексов
- `REINDEX` – полностью перестраивает отдельный индекс
- `TRUNCATE` — усекает, «опустошает» таблицу

1 Упорядочивание версий строк, которое выполняет команда `CLUSTER`, дает планировщику возможность более эффективно использовать индексный доступ в некоторых случаях.

При модификации данных таблицы физический порядок версий строк будет нарушаться. Для восстановления упорядоченной структуры нужно будет повторно выполнять команду `CLUSTER`.

2 Команда `TRUNCATE` логически работает так же, как и `DELETE` — удаляет все записи из таблицы.

3 Команда `TRUNCATE` создает новый чистый файл для хранения таблицы.

4 Команда `DELETE` – помечает версии строк как удаленные. Далее понадобится очистка.

5 Команда `TRUNCATE` работает быстрее, но блокирует работу с таблицей на все время до конца транзакции.

Предотвращение переполнения счетчика транзакций

- Под счетчик транзакций (XID – transaction ID) отведено 32 бита
- Пространство значений XID циклично и не имеет конечной точки
- Процедура VACUUM помечает строки как замороженные, указывая, что они были вставлены транзакцией, зафиксированной достаточно давно
- Признак «заморожено» говорит о том, что версия строки появилась так давно, что номер создавшей ее транзакции больше не имеет значения, и его можно использовать повторно
- Если не выполнять заморозку своевременно, сервер не сможет выделить очередной номер транзакции, а это аварийная ситуация: сервер остановится, все незавершенные транзакции оборвутся

1 PostgreSQL упорядочивает события с помощью *номеров транзакций* (XID - transaction ID).

Примечание:
*р*азмера счетчика в 32 бита достаточно для выполнения более 4 миллиардов транзакций.

2 Кластер, работающий долгое время, столкнётся с зацикливанием идентификаторов транзакций: счётчик транзакций прокрутится до нуля, и транзакции, которые относились к прошлому, окажутся в будущем — это означает, что их результаты станут невидимыми.

3 Для того, чтобы избежать этой ситуации, необходимо выполнять очистку для каждой таблицы в каждой базе данных как минимум **один** раз на **два** миллиарда транзакций.

4 Процедура VACUUM помечает строки как *замороженные*, указывая, что они были вставлены транзакцией, зафиксированной достаточно давно, так что эффект добавляющей транзакции с точки зрения MVCC будет виден во всех текущих и будущих транзакциях.

5 Обычные значения XID сравниваются по модулю с 2^32. Это означает, что для каждого обычного XID существуют два миллиарда значений XID, которые «старше» него, и два миллиарда значений, которые «младше» него.

6 Как только создаётся версия строки с обычным XID, для следующих двух миллиардов транзакций эта версия строки оказывается «в прошлом», неважно о каком значении обычного XID идет речь.

7 Если после двух миллиардов транзакций эта версия строки всё ещё существует, она внезапно окажется в будущем.

Для того чтобы это предотвратить, в какой-то момент значение XID для старых версий строк должно быть заменено на FrozenTransactionId (заморожено) до того, как будет достигнута граница в два миллиарда транзакций.

8 В карту видимости добавлен бит, отмечающий те страницы, на которых все версии строк уже заморожены.

9 Если произойдет переполнение счетчика транзакций администратор должен будет вручную стартовать сервер и выполнить заморозку.

10 Подробнее: https://www.postgresql.org/docs/11/routine-vacuuming.html

Запуск команд очистки и анализа

- Через клиентское приложение
  - VACUUM — очистка всех таблиц БД
  - VACUUM [таблица, ...] — очистка таблиц из указанного списка
  - ANALYZE — сбор статистики по базе данных
  - VACUUM ANALYZE — очистка и анализ
- Из ОС:
  - vacuumdb – обертка для команды VACUUM для запуска из ОС
  - vacuumdb --analyze-only – сбор статистики по БД
  - vacuumdb --analyze – сбор статистики по базе данных

1 Утилите *vacuumdb* – это утилита командной строки в PostgreSQL, которая позволяет выполнять операции очистки и анализа на базах данных PostgreSQL.

2 Пример:
```
vacuumdb --all --verbose
```
– выполнение операции VACUUM на всех базах данных сервера с выводом подробной информации о процессе.

3 Команды очистки и анализа можно запускать по расписанию:
- Слишком частый запуск по расписанию создаст ненужную нагрузку на систему.
- Если же запускать очистку редко, при большом объеме изменений файлы могут успеть существенно вырасти в размерах.

Примечание:

Команду VACUUM можно вызывать, указывая ANALYZE в скобках и без них:

- Команда VACUUM (ANALYZE) имя_таблицы; использует скобки для указания того, что параметр ANALYZE является опциональным. Он служит для отображения того факта, что можно использовать просто VACUUM имя_таблицы; , без формирования статистики, если необходимо только освободить пространство.
- Команда VACUUM ANALYZE имя_таблицы; напрямую указывает на выполнение обеих операций (VACUUM и ANALYZE) без скобок.

Наблюдение за прогрессом очистки

- Представление `pg_stat_progress_vacuum`
- Некоторые столбцы представления и их описание:
  - `pid` – идентификатор (PID) обслуживающего процесса
  - `datname` – имя БД, к которой подключён этот обслуживающий процесс
  - `phase` – текущая фаза очистки
  - `max_dead_tuples` – число «мёртвых» кортежей, которое можно сохранить, прежде чем потребуется выполнить цикл очистки индекса, в зависимости от maintenance_work_mem
  - `num_dead_tuples` – число «мёртвых» кортежей, собранных со времени последнего цикла очистки индекса

1 Во время выполнения VACUUM представление pg_stat_progress_vacuum будет содержать по одной строке для каждого обслуживающего процесса (включая рабочие процессы автоочистки), производящего очистку.

2 Подробнее: https://www.postgresql.org/docs/current/progress-reporting.html#VACUUM-PROGRESS-REPORTING

3 Параметр `maintenance_work_mem` (`integer`) задаёт максимальный объём памяти для операций обслуживания БД, в частности VACUUM, INDEX и др.

4 По умолчанию значение maintenance_work_mem — **64 мегабайта** (64MB).

5 В один момент времени в сеансе может выполняться только одна операция обслуживания БД. Обычно такие операции не запускаются параллельно, и значение параметра maintenance_work_mem вполне может быть гораздо больше значения параметра work_mem.

6 Параметр work_mem задаёт объём памяти, который будет использоваться для внутренних операций сортировки и хеш-таблиц, прежде чем будут задействованы временные файлы на диске. Значение параметра work_mem по умолчанию — 4 мегабайта (4MB). Увеличение значения параметра maintenance_work_mem может привести к ускорению операций очистки и восстановления БД из копии.

7 Когда выполняется автоочистка, объём памяти, указанный через параметр maintenance_work_mem может быть выделен *autovacuum_max_workers раз*, поэтому не нужно устанавливать значение по умолчанию слишком большим.

Автоматическая очистка

- *Автоочистка* (autovacuum) – автоматизирует выполнение команд VACUUME и ANALYZE
- Автоочистка реагирует на активность изменения данных
- Для осуществления автоочистки работают процессы:
  - *Autovacuum launcher* – фоновый процесс, который запускает рабочие процессы
  - *Autovacuum worker* – процесс, который очищает таблицы отдельной базы данных, требующие обработки
- Работа автоочистки настраивается через параметры

1 Когда автоочистка включена, она проверяет, в каких таблицах было вставлено, изменено или удалено много строк.

При этих проверках используются средства сбора статистики.

Чем активней изменяется таблица, тем чаще она будет обрабатываться с помощью автоочистки.

2 Автоочистка будет работать, только если параметр track_counts имеет значение *true*. Этот параметр можно переопределить в файле postgresql.conf.

В конфигурации по умолчанию автоочистка включена.

3 «Демон автоочистки» включает нескольких процессов. Существует постоянный фоновый процесс, называемый *процессом запуска автоочистки* (autovacuum launcher), который отвечает за запуск рабочих процессов автоочистки для всех баз данных (autovacuum worker).

4 Этот контролирующий процесс распределяет работу по времени, стараясь запускать рабочий процесс для каждой базы данных каждые *autovacuum_naptime секунд*. По умолчанию параметр autovacuum_naptime имеет значение **1 минута**.

5 Например, если всего имеется N баз данных, новый рабочий процесс будет запускаться каждые *autovacuum_naptime/N секунд*.

6 Одновременно могут выполняться несколько рабочих процессов. Максимальное количество ограничено парамером autovacuum_max_workers.

7 Если число баз данных, требующих обработки, превышает количество, указанное в параметре autovacuum_max_workers, обработка следующей базы начинается сразу по завершении первого рабочего процесса.

8 Каждый рабочий процесс проверяет все таблицы в своей базе данных и в случае необходимости выполняет VACUUM и/или ANALYZE.
Для отслеживания действий рабочих процессов можно установить параметр *log_autovacuum_min_duration*.

9 Очистка работает постранично, не приводит к блокировкам других транзакций.

10 Автоматическая очистка перестанет работать при отключении любого из двух параметров *autovacuum* или *track_counts*.

11 Отказ от очистки влечет за собой:

- неконтролируемое разрастание файлов,
- замедление запросов,
- риск аварийной остановки сервера.

Вопросы для проверки

1 Какие действия производятся при выполнении команды `VACUUM (ANALYZE) sales;`?

2 В каких единицах определяется стоимости обработки страницы?

3 С каким параметром требуется запустить очистку, чтобы сократить число страниц данных (и уменьшить размер файлов)?

Практическая работа

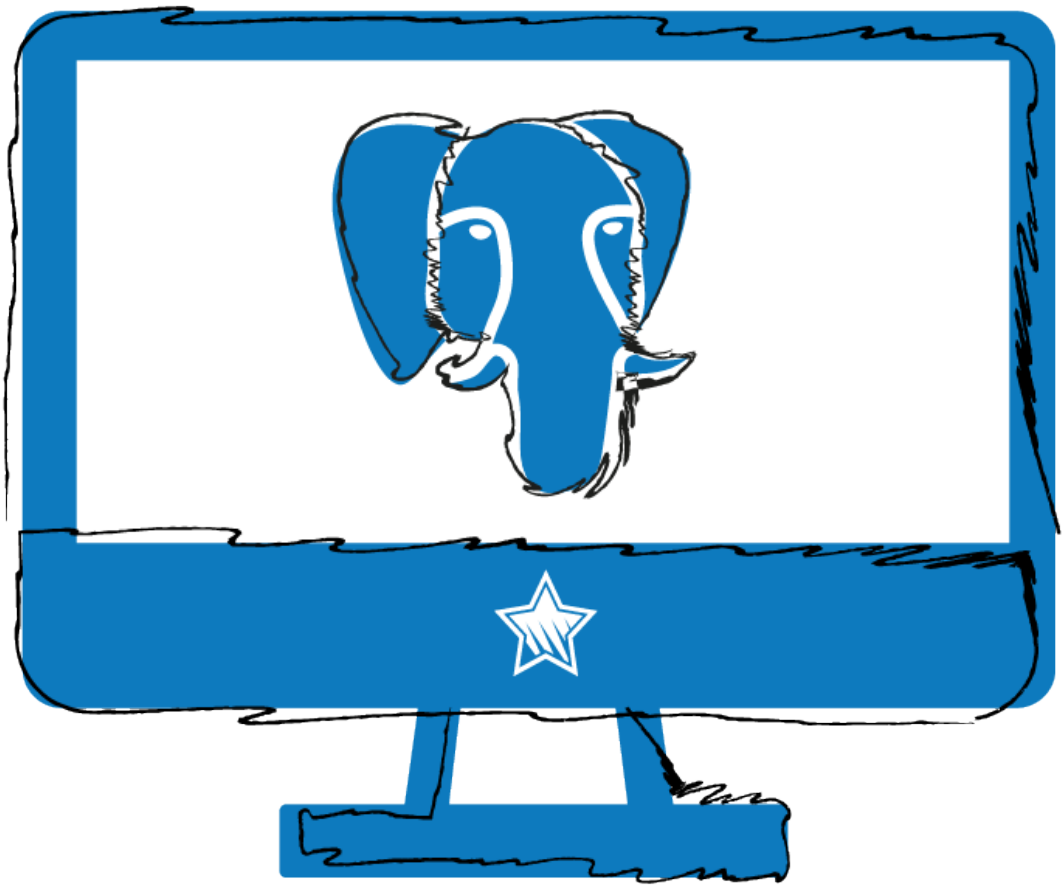
Оценка времени выполнения: 30 мин.

- Выполните задания модуля 10 из Сборника практических заданий.

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Модуль 11

Автоматизация обслуживания



ASTRA LINUX®

Модуль 11.
Автоматизация обслуживания

Краткое содержание:

- Планировщик заданий pgAgent.
- Создание заданий через графическую утилиту pgAdmin.
- Создание заданий через сценарии SQL.

Основные знания и умения:

- понимать механизм работы планировщика заданий pgAgent;
- уметь устанавливать и запускать pgAgent;
- уметь создавать задания через графическую утилиту pgAdmin;
- уметь создавать задания сценарии SQL.

Оценка знаний и умений:

- Практическая работа, контрольные вопросы.

Дополнительная литература:

- Отсутствует.

Цели модуля 11. Автоматизация обслуживания

- Планировщик заданий pgAgent
- Создание заданий через графическую утилиту pgAdmin
- Создание заданий через сценарии SQL

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Планировщик заданий pgAgent

- *pgAgent* – это агент планирования, который может:
  - запускать задания и управлять ими
  - запускать последовательности заданий, состоящих из SQL-скриптов (в т.ч. на разных БД) и/или shell/batch-скриптов
  - задавать нерабочие окна (например, НЕ совершать некоторое действие по выходным)
- Создание и управление заданиями возможно с использованием графического интерфейса через pgAdmin и через выполнение SQL-запросов

1 Устанавливается pgAgent в качестве отдельной службы (демона), которая постоянно запущена в режиме ожидания, готова выполнить любую зарегистрированную задачу.

2 Этот планировщик работает как расширение PostgreSQL, но исполняет задания не «внутри» СУБД, а создавая собственных «внешних» демонов.

3 Если запланировано одновременное выполнение двух или более заданий, pgAgent выполнит их *параллельно* (каждое в своем собственном потоке).

4 Шагом задания может быть серия инструкций SQL или пакетный сценарий операционной системы/оболочки.

5 Каждый шаг в данном задании выполняется после завершения предыдущего шага в алфавитно-цифровом порядке по названию.

6 Переключатели в диалоговом окне задания pgAgent (доступ к которому осуществляется через контекстное меню свойств) позволяют изменять задание, включая или отключая отдельные шаги по мере необходимости.

7 Планировщик pgAgent не настраивается автоматически при установке pgAdmin.

Установка демона pgagent

- Установка пакета: `sudo apt install pgagent`
- Установка расширения для PostgreSQL:
  - при подключении к БД postgres выполнит запрос:
    `CREATE EXTENSION pgagent;`
  - эта команда создаст несколько таблиц и других объектов в схеме под названием *pgagent*

1 Планировщик pgAgent сохраняет свою конфигурацию в базе данных postgres в кластере.

2 Чтобы установить демон pgAgent, понадобятся права суперпользователя для изменения сценариев запуска системы.

Параметры утилиты pgAgent

- `-f` — запускать процесс на переднем плане (не отсоединять от терминала)
- `-t` <частота опроса в секундах (по умолчанию 10)>
- `-r` <интервал повторной попытки подключения после прерывания соединения в секундах (>=10, по умолчанию 30)>
- `-s` <файл журнала (сообщения записываются в стандартный вывод, если не указано иное)>
- `-l` <подробность ведения журнала (ОШИБКА=0, ПРЕДУПРЕЖДЕНИЕ=1, ОТЛАДКА=2, по умолчанию 0)>

1 Утилита pgAgent использует несколько параметров командной строки, большинство из которых необходимы только для отладки или специализированных конфигураций.

Запуск pgAgent

- Запустить демон: `pgagent [параметры] <строка_подключения>`
- Пример запуска: `sudo /usr/bin/pgagent hostaddr=127.0.0.1 dbname=postgres user=iivanov`
- Пароль для подключения к PostgreSQL нельзя указывать в строке подключения pgAgent – он может быть виден всем пользователям в выводе ‘ps’
- Для подключения следует использовать файл *~/.pgpass*, который хранит пароли для каждой базы данных, к которой должен получить доступ pgAgent.

1 Для запуска используется стандартная строка подключения PostgreSQL *libpq*.

libpq – это набор программных библиотек, которые позволяют клиентским программам подключаться и работать с базой данных PostgreSQL. Эти библиотеки реализуют важные функции, такие как: подключение к серверу баз данных, передача команд от клиента к серверу, а также возврат результатов.

2 Например, следующая командная строка запустит pgAgent на сервере, прослушивающем 127.0.0.1, используя базу данных с именем *postgres*, подключающийся от имени пользователя *iivanov*:

```
/usr/bin/pgagent hostaddr=127.0.0.1 dbname=postgres user=iivanov
```

3 Чтобы убедиться, что процесс запустился, можно использовать команду:

```
ps -all | grep pgagent
```

```
sa@astra:~$ ps -all | grep pgagent
1 S     0  1732     1  0  80   0 -  6383 -      pts/1    00:00:03 pgagent
```

4 *Пример:*

вывод команды `ps`, когда для запуска pgagent используется строка подключения, в которой указан пароль:

```
/usr/bin/pgagent hostaddr=127.0.0.1 dbname=postgres user=iivanov password=P@ssw0rd
```

В этом случае команда ps отобразит следующую информацию:

```
sa@astra:~$ ps -ef | grep pgagent
root      1732     1  0 02:33 ?        00:00:04 /usr/bin/pgagent hostaddr=127.0.0.1 dbname=postgres user=iivanov password=P@ssw0rd
```

5 Для создания файла и настройки доступа используются команды:

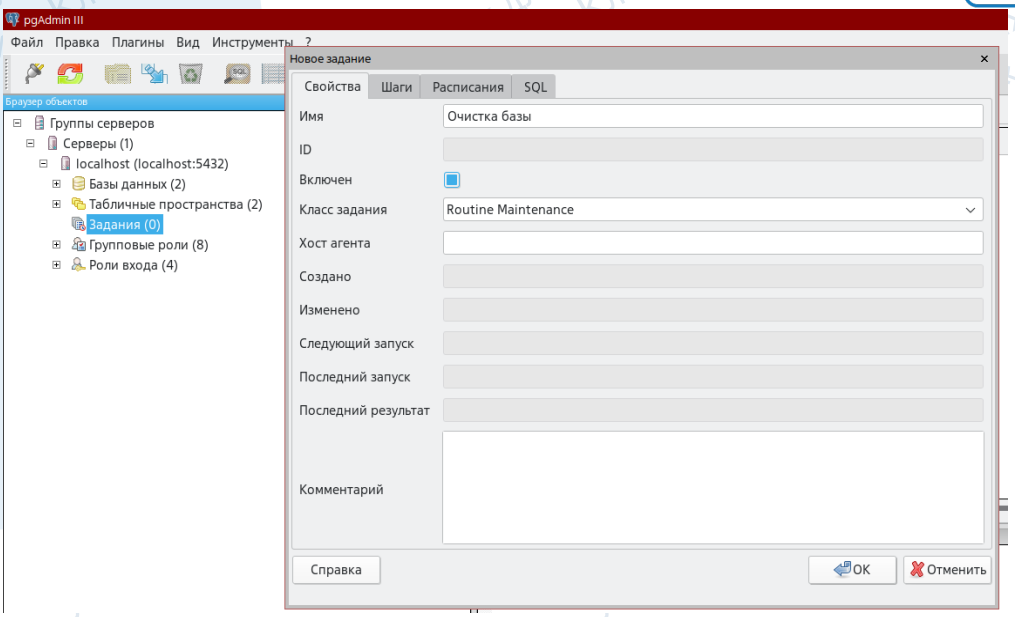
```
touch ~/.pgpass
chmod 600 ~/.pgpass
```

```
sa@astra:~$ touch ~/.pgpass
sa@astra:~$ sudo chmod 600 ~/.pgpass
sa@astra:~$ sudo cat ~/.pgpass
localhost:5432:*:iivanov:P@ssw0rd
```

6 Файл использует формат:

```
hostname:port:database:username:password
```

Задания pgAgent



1 Планировщик pgAgent запускает задания, каждое из которых включает описание шагов и расписание.

2 Настроить задания можно на том же сервере, на котором были созданы объекты базы данных pgAgent.

3 Настройку можно выполнить в графической утилите pgAdmin.

4 После установки расширения и запуска службы pgagent в pgAdmin отобразится узел заданий.

Новые задания могут быть созданы, а существующие задания изменены таким же образом, как и любой другой объект в pgAdmin.

5 При настройке задания нужно указать имя задания, независимо от того, включено оно или нет, класс задания или категорию.

6 Имя хоста конкретной машины, на которой запущен pgAgent, если это задание должно выполняться только на этом конкретном сервере, указывать необязательно.

Если оставить поле пустым, любой сервер может запустить задание. Обычно это не требуется для заданий, которые включают только шаги с выполнением SQL-запросов.

7 Любые задания с шагами пакетной обработки/оболочки, возможно, потребуют настройки на определенный сервер.

8 Также может быть включен комментарий.

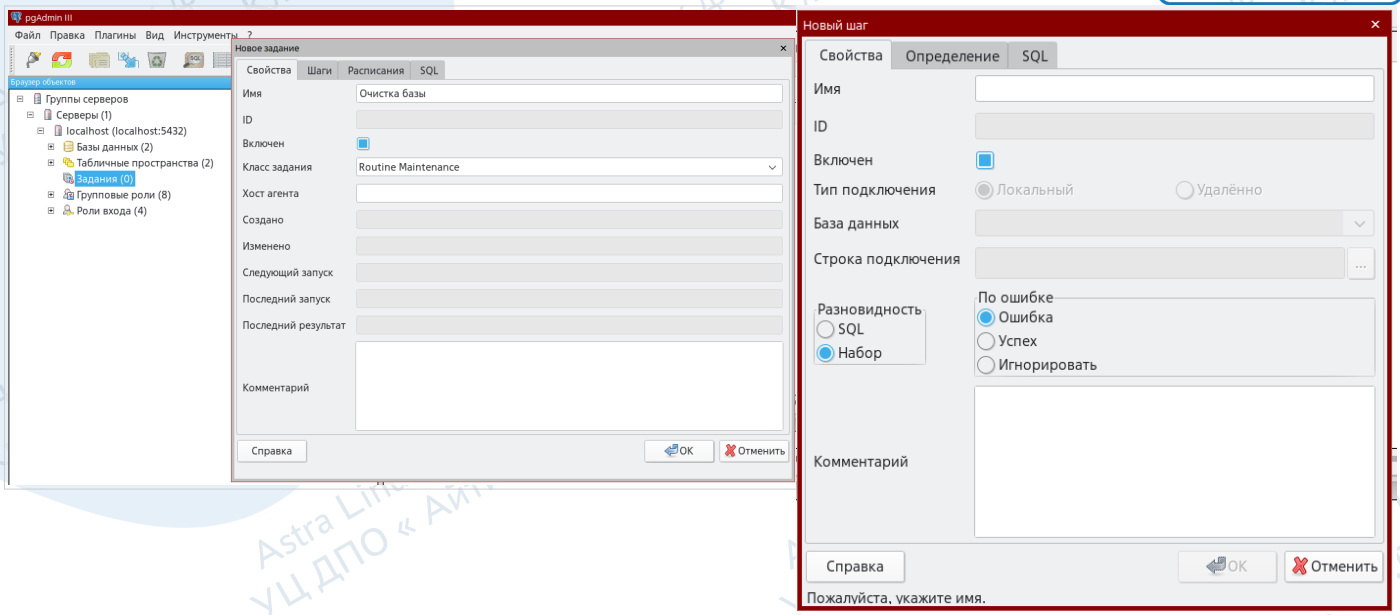
Примечание:

не всегда очевидно, какое значение следует указать для хост-агента, чтобы настроить цель шага задания для выполнения на конкретной машине.

9 Когда pgAgent запущен на требуемых компьютерах и подключен к базе данных планировщика, можно выполнить следующий запрос для просмотра имен хостов, сообщаемых каждым агентом:

```
SELECT jagstation FROM pgagent.pga_jobagent;.
```

Шаги pgAgent



1 Каждое задание состоит из нескольких шагов, каждый из которых может быть SQL-скриптом или пакетным сценарием операционной системы/оболочки.

2 Каждый шаг в данном задании выполняется по очереди, в алфавитно-цифровом порядке имен.

3 Шаги могут быть добавлены к заданию через диалог свойств задания или добавлены как подобъект.

4 На вкладке **Свойства** главного окна pgAdmin будут отображаться подробные сведения о выбранном шаге, а на вкладке **Статистика** будут отображаться подробные сведения о каждом запуске шага, включая выходные данные или ошибки скрипта.

5 Если при создании шага для параметра **Разновидность** задано значение SQL, то должна быть выбрана база данных, с которой будет выполняться скрипт.

6 Если установлено значение **Набор** (пакет), строка базы данных/подключения должна быть оставлена пустой.

7 Параметр **On Error** (По ошибке) определяет, как невыполнение этого шага повлияет на состояние всего задания.

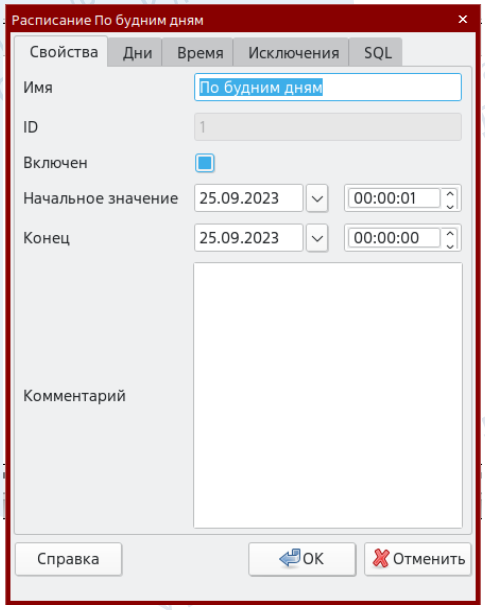
8 Вкладка **Определение** содержит единственное текстовое поле, в которое следует ввести пошаговый сценарий.

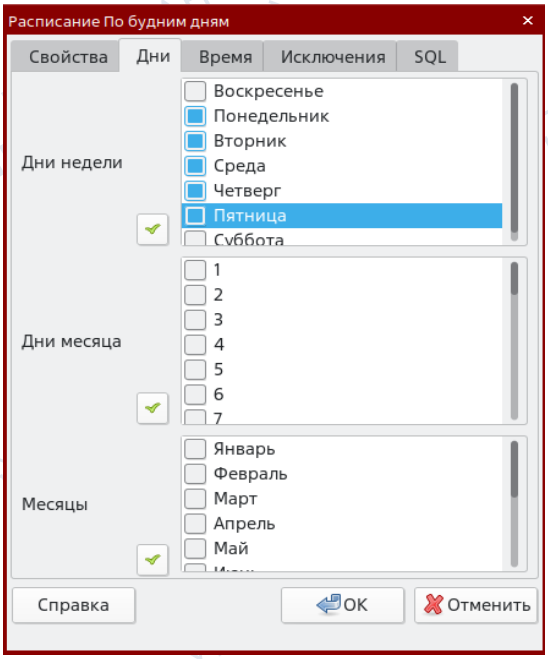
Для шагов SQL это должна быть серия из одной или нескольких инструкции SQL.

9 При запуске на сервере *nix может использоваться любой сценарий оболочки, при условии, что в первой строке указан подходящий интерпретатор (например, `*#!/bin/sh`).

Astra Linux для АНО УЦ ДПО « Айти Клауд»

Расписания pgAgent





1 Каждое задание выполняется в соответствии с одним или несколькими расписаниями.

Каждый раз, когда изменяется задание или любое из его расписаний, пересчитывается время следующего выполнения задания.

2 Каждый экземпляр pgAgent периодически опрашивает базу данных на наличие заданий со значением времени следующего выполнения, которое на данный момент уже прошло.

При опросе не реже одного раза в минуту все задания обычно запускаются в течение одной минуты после указанного времени начала выполнения.

3 Если в тот момент, когда должно выполниться задание, ни один экземпляр pgAgent не запущен, задание будет выполнено сразу же после следующего запуска pgAgent.

После этого запуск заданий будет происходить по обычному расписанию.

4 Расписания могут быть добавлены к заданию через диалоговое окно свойств задания или добавлены как подобъект.

5 На вкладке **Свойства** главного окна pgAdmin отобразятся подробные сведения о выбранном расписании.

Создание заданий через сценарий SQL

- Создавать задания можно без использования графического интерфейса
- Необходимо вставить строку в таблицу
- Пример сценария добавления задания:

```
INSERT INTO pgagent.pga_job (jobjclid, jobname, jobdesc,
jobenabled, jobhostagent)
SELECT jcl.jclid, 'Логическое РК БД Инвентаризация', '',
true, ''
FROM pgagent.pga_jobclass jcl WHERE jclname='Routine
Maintenance';
```

1 Для создания задания необходимо в таблицу pgagent.pga_job вставить строку, указав значения для следующих столбцов:

- `jobjclid` – идентификатор класса задания (в данном запросе для задания класса с названием Routine Maintenance);
- `jobname` – название задания;
- `jobdesc` – описание задания;
- `jobenabled` – включено ли задание;
- `jobhostagent` – на каком хосте должно выполняться.

Создание шагов заданий через сценарий SQL

- Необходимо вставить строку в таблицу *pgagent.pga_jobstep*
- Пример сценария добавления шага:

```
INSERT INTO pgagent.pga_jobstep (jstjobid, jstname,
jstdesc, jstenabled, jstkind, jstonerror, jstcode,
jstdbname, jstconnstr)
SELECT (SELECT jobid
        FROM pgagent.pga_job
        WHERE jobname = 'Логическое РК БД
Инвентаризация'), 'Создание логической копии', '', true,
'b', 'f', 'pg_dump -U zzakharov -h localhost -d backup_db
-t sidorov.test_copy -f /test_copy.sql --create', '', '';
```

1 Для создания задания необходимо в таблицу pgagent.pga_jobstep вставить строку, указав значения для следующих столбцов:

- `jstname` — название шага задания;
- `jstdesc` — описание шага задания;
- `jstenabled` — включен ли шаг задания;
- `jstkind` — разновидность шага задания;
- `jstonerror` — действие для всего задания при ошибке на данном шаге;
- `jstcode` — код сценария на шаге,
- `jstdbname` — имя БД (при SQL сценариях),
- `jstconnstr` — строка подключения.

Создание расписаний заданий через сценарий SQL

- Пример сценария добавления расписания (каждый день в 08:45):

```
INSERT INTO pgagent.pga_schedule (jscjobid, jscname,
jscdesc, jscminutes, jschours, jscweekdays, jscmonthdays,
jscmonths, jscenabled, jscstart, jscend)
VALUES((SELECT jobid FROM pgagent.pga_job
WHERE jobname = 'Логическое РК БД Инвентаризация'),
'Каждый день
08:45','','{f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,
f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,t,f,f,
f,f,f,f,f,f,f,f,f,f,f,f,f}','{f,f,f,f,f,f,f,f,t,f,f,f,f,f,f,
f,f,f,f,f,f,f,f,f,f,f}','{t,t,t,t,t,t,t}','{t,t,t,t,t,t,t,t,
t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t}','{t,t,t,
t,t,t,t,t,t,t,t,t,t}',true, '2023-07-16 00:00:00', NULL);
```

1 Для создания задания необходимо в таблицу *pgagent.pga_jobstep* вставить строку, указав значения для следующих столбцов:

- `jscjobid` — идентификатор расписания задания;
- `jscname` — название расписания;
- `jscdesc` — описание расписания;
- `jscminutes` – минуты запуска задания (60 значений, для тех минут, когда должно запускаться задание указывается t(true), для остальных f(false));
- `jschours` — часы запуска задания (24 значения t(true) или f(false));
- `jscweekdays` – дни недели запуска задания (7 значений t(true) или f(false));
- `jscmonthdays` – дни месяца запуска задания (31 значение t(true) или f(false));
- `jscmonths` – месяцы запуска задания (12 значений t(true) или f(false));
- `jscenabled` — включено ли расписание;
- `jscstart` — дата и время начала работы расписания;
- `jscend` – дата и время завершения работы расписания.

Вопросы для проверки

1 Где планировщик pgAgent сохраняет свою конфигурацию?

2 Какими способами можно создавать задания/расписания/шаги заданий pgAgent?

3 Где в утилите pgAdmin следует вводить код сценария shell/batch или SQL, который должен быть выполнен на шаге задания?

Практическая работа

Оценка времени выполнения: 45 мин.

- Выполните задания модуля 11 из Сборника практических заданий.

Astra Linux для АНО УЦ ДПО « Айти Клауд»