

PostgreSQL 14.

Оптимизация,

Kubernetes,

кластера,

облака.

УДК 004.4/.6
ББК 32.97
А81

А81 Аристов Евгений. PostgreSQL 14. Оптимизация, Kubernetes, кластера, облака. - М.: ООО «Сам Полиграфист», 2022. - 576 с.

ISBN 978-5-00166-569-4

В данной книге подробно изучается внутреннее устройство PostgreSQL 14, начиная с вариантов установки, физического устройства этой системы управления базами данных, заканчивая бэкапами и репликацией, а также разбираются кластерные и облачные решения на основе PostgreSQL.

Книга написана понятным языком с использованием подробных примеров. Исходные коды и ссылки, используемые в книге, выложены на github.

Главы логически выстроены по пути усложнения материала и позволяют разобраться, как правильно развернуть кластер PostgreSQL с нуля на отдельных серверах, в виртуальном и облачном исполнении. Подробно рассматриваются настройки, отвечающие за производительность, безопасность и варианты их тюнинга.

Отдельно выделены несколько глав, где рассматриваются варианты оптимизации с использованием различных механизмов: индексы, секционирование, джойны и другое.

Книга будет интересна и полезна широкой аудитории: разработчикам, администраторам баз данных, архитекторам программного обеспечения, в том числе микросервисной архитектуры.

Тираж 100 экз. Заказ № 106201.

Отпечатано в типографии «OneBook.ru» ООО «Сам Полиграфист»
129090 г. Москва, Протопоповский пер., 6
www.onebook.ru

© Аристов Е.Н., 2022.

ISBN 978-5-00166-569-4



Оглавление

Об авторе	4
1. PostgreSQL 14. Установка, новые возможности	6
2. Физический уровень	28
3. Работа с консольной утилитой <code>psql</code>	57
4. ACID && MVCC. Vacuum и autovacuum	63
5. Уровни изоляции транзакций	79
6. Логический уровень	89
7. Работа с правами пользователя	105
8. Журналы	122
9. Блокировки	155
10. Настройка PostgreSQL	183
11. Работа с большим объёмом реальных данных	206
12. Виды индексов. Работа с индексами и оптимизация запросов	224
13. Различные виды join'ов. Применение и оптимизация	252
14. Сбор и использование статистики	279
15. Оптимизация производительности. Профилирование. Мониторинг	291
16. Секционирование	309
17. Резервное копирование и восстановление	330
18. Виды и устройство репликации в PostgreSQL	374
19. PostgreSQL и Google Kubernetes Engine	404
20. Кластеры высокой доступности для PostgreSQL	441
21. Кластеры для горизонтального масштабирования PostgreSQL	476
22. PostgreSQL и Google Cloud Platform	511
23. PostgreSQL и AWS	532
24. PostgreSQL и Azure	550
25. PostgreSQL и Яндекс Облако	564
Послесловие от автора	576

Об авторе

Для начала, хотелось бы поблагодарить за то, что вы выбрали эту книгу. Я в IT уже больше 20 лет. Начинал, как и многие когда-то с ZX Spectrum. Написал свою первую игру наподобие текстового Dictator. Тогда все хранилось на простых аудиокассетах, а язык был Basic... Потом IBM 8086. Хотя кто уже помнит те времена... Дальше DOS, первый Windows 3.11, - профильный университет, базы данных большие и маленькие, сотни проектов, - виртуализация, кластеризация и highload. В какой-то момент понял, что хочу учить людей, нести светлое и доброе в наш мир. И вот я больше года читаю свои авторские курсы по Постгресу в «OTUS Онлайн-образование»¹ и «УЦ Специалист» при МГТУ им Н.Э. Баумана².

За это время понял, насколько не хватает хорошей книги по лучшим методикам и возможностям самой популярной СУБД³ - PostgreSQL. В этой книге будут описаны практические рекомендации, проведён глубокий анализ внутреннего устройства для понимания принципов работы этой СУБД и возможностей её тонкой настройки. Ну и куда без самого популярного оркестратора контейнеризации K8s⁴ и облаков. Конечно, рассмотрим и различные виды кластеризации PostgreSQL.

Эта книга актуализирована под новую 14 версию Постгреса. Добавлены новые примеры, рассмотрено обновление кластера с 13 на 14 версию и тонкости этого процесса в связи с новой системой шифрования паролей. Разобраны популярные инструменты, не вошедшие в книгу по 13 Постгресу: современные бэкапы с помощью pg_probackup и Wal-G, утилита pg_rewind для восстановления кластера после сбоя.

Ссылки из данной книги будут расположены на github⁵ для удобства перехода. В репозитории ссылки будут разбиты по номерам тем, номер ссылки будет в квадратных скобках [1]. Я всегда готов к сотрудничеству - об этом можно посмотреть на моём личном сайте: <https://aristov.tech>. Также на сайте можно заказать оригинальную электронную версию в PDF или бумажную версию с автографом <https://aristov.tech/#orderbook>.

¹ OTUS URL : <https://otus.ru/> (дата обращения 15.10.2021)

² УЦ Специалист URL : <https://www.specialist.ru/> (дата обращения 15.10.2021)

³ Система Управления Базами Данных

⁴ Kubernetes

⁵ Мой github URL : <https://github.com/aeuge/Postgres14book> (дата обращения 04.10.2021)

Спасибо за поддержку Илье @JorianR, который и положил начало моей преподавательской карьеры. Отличный комьюнити-менеджер. Спасибо Алексею Цыкунову @erlong15. Мой наставник, Профессионал с большой буквы, опытнейший DBA⁶ и просто замечательный человек. Спасибо любимой жене Светлане @asveta за поддержку. Спасибо редакторам книги Павлу Андрееву @shaadowsky и Андрею Alandre @alandreei, которые заставили меня плакать, переделывая финальный вариант раз десять. Спасибо Сергею Краснову @SerjReds - приложил руку к моему долгому пути к преподаванию. Спасибо Дмитрию Кириллову @esteps_kirillov за техническую редактуру книги. Ну, и конечно же, любимой дочке Анастасии, это её рисунок вы видите на обложке книги.

Просьба оставить отзыв о книге <https://aristov.tech/postgres-feedback> на моём сайте. Любые мнения, пожелания, предложения по доработке приветствуются.

Желаю вам получить удовольствие от прочтения книги и удачи в карьере.

⁶ DataBase Architector/Administrator

1. PostgreSQL 14. Установка, новые возможности

Давайте рассмотрим основные варианты установки СУБД. Их можно разделить как по видам ОС⁷, так и по способу установки: на конкретной ОС, docker⁸, docker-compose⁹, в K8s различными способами: через манифесты или пакетный менеджер helm¹⁰. Также в этой главе мы рассмотрим вариант использования Постгреса как DBaaS¹¹ в облаке Google. Более подробно облако Google и остальные облака рассмотрим в соответствующих главах.

Начнём с самого общепринятого - установка СУБД на конкретной ОС. В текущих реалиях есть три стандартных ОС: Windows, Linux, MacOS X. Рекомендую использовать Linux, но в принципе работать будет на любой ОС. Здесь и далее будут ссылки на сайт с официальной документацией PostgreSQL Global Development Group <https://www.postgresql.org/>. Ссылка на страницу с дистрибутивами¹².

Итак, **Windows**. Постгрес выпускается для следующих 64-битных версий Windows:

- Windows 8.1, 10, 11
- Windows Server 2008 R2 и новее

Есть два варианта установки:

- в графическом интерактивном режиме
- в режиме командной строки

В первом варианте всё просто, скачиваем установочный дистрибутив и, ответив на несколько несложных вопросов и нажимая кнопку “далее”, установим СУБД на компьютер. Ссылка на дистрибутив для Windows¹³.

В режиме командной строки можно сразу задать ряд параметров, таких как: путь установки, каталог с данными, пароль суперпользователя и так далее. Для замены стандартных параметров вы можете использовать INI-файл и указать там один или несколько параметров.

Но в целом, установка Постгреса на Windows в продакшн-системах не рекомендуется, так как есть проблемы с производительностью файловой системы и распараллеливанием процессов. Поэтому развёрнутого описания в книге нет. В тренировочных целях вы можете сделать это самостоятельно.

⁷ Операционная система

⁸ Docker URL: <https://www.docker.com/> (дата обращения 15.10.2021) [1]

⁹ Docker Compose URL: <https://docs.docker.com/compose/> (дата обращения 15.10.2021) [2]

¹⁰ Helm URL: <https://helm.sh/> (дата обращения 15.10.2021) [3]

¹¹ Database As a Service

¹² Страница установки URL:

<https://www.postgresql.org/download/> (дата обращения 15.09.2021) [4]

¹³ Дистрибутив Постгрес для Windows URL:

<https://www.postgresql.org/download/windows/> (дата обращения 15.10.2021) [5]

MacOS X

Ссылка на дистрибутив¹⁴. Опять же варианты установки через GUI¹⁵ или, например, homebrew¹⁶.

Linux

Переходим к самой рекомендованной ОС и самым популярным вариантам установки. В книге будет рассматриваться работа СУБД на Ubuntu. На других других дистрибутивах Линукс все команды будут также работать, хотя возможны незначительные изменения, всё это есть в документации. Список актуальных версий всегда можно посмотреть на официальном сайте.

Для Ubuntu список выглядит так:

- impish (21.10)
- hirsute (21.04)
- focal (20.04)
- bionic (18.04)

Здесь и далее, в основном, будем рассматривать дистрибутив **Ubuntu 2021.04 Hirsute Hippo**¹⁷. Если будет использоваться другой дистрибутив Ubuntu, об этом я прямо напишу в тексте. Вообще, общая рекомендация - на продакшн-системах использовать только **LTS** (long time support) версии Ubuntu - пятилетняя поддержка (выпуск патчей, обновлений безопасности). Остальные версии поддерживаются максимум два года. В нашем случае книга предназначена для обучения, поэтому используем почти последнюю версию, заодно покажу, как решать возникающие при этом проблемы.

Вариант GUI не рассматриваем, так как в реальных проектах сервера используются без графической оболочки. Если всё же хотите посмотреть, как это выглядит и работает с GUI, устанавливаем или через установку приложений (магазин типа Ubuntu Software или дистрибутив с сайта), или через командную строку и работаем из GUI клиента, например, DBeaver¹⁸.

Остаётся самый используемый вариант - командная строка, и тут есть три варианта.

¹⁴ Дистрибутив Постгрес под MacOS URL :

<https://www.postgresql.org/download/macosx/> (дата обращения 15.10.2021) [6]

¹⁵ GUI (Graphical User Interface) или ГИП (графический интерфейс пользователя)

¹⁶ Homebrew URL : <https://brew.sh/> (дата обращения 15.10.2021) [7]

¹⁷ Ubuntu 2020.10 URL: <https://releases.ubuntu.com/21.04/> (дата обращения 15.10.2021) [8]

¹⁸ DBeaver URL: <https://dbeaver.io/> (дата обращения 15.10.2021) [9]

1. Простая установка - на версиях Ubuntu 2020.04 и ниже. Установится дефолтная версия для конкретной ОС. Например, в Ubuntu 2020.04 это будет PostgreSQL 12 (здесь и далее выполняемые нами команды будут помечены **полужирным курсивом**. **Без курсива** будет предложен подходящий вариант, но выполнять его не будем):

```
sudo apt install postgresql
```

В более современной **Ubuntu 2021.10 Groovy Gorilla** так просто установить Постгрес не получится:

```
E: Failed to fetch http://security.ubuntu.com/ubuntu/pool/main/p/postgresql-12/libpq5_12.5-0ubuntu0.20.10.1_amd64.deb 404 Not Found [IP: 35.184.213.5 80]
E: Failed to fetch http://security.ubuntu.com/ubuntu/pool/main/p/postgresql-12/postgresql-client-12_12.5-0ubuntu0.20.10.1_amd64.deb 404 Not Found [IP: 35.184.213.5 80]
E: Failed to fetch http://security.ubuntu.com/ubuntu/pool/main/p/postgresql-12/postgresql-12_12.5-0ubuntu0.20.10.1_amd64.deb 404 Not Found [IP: 35.184.213.5 80]
E: Unable to fetch some archives, maybe run apt-get update or try with --fix-missing?
```

Для решения проблемы обновим список пакетов:

```
sudo apt update
```

И установим сами пакеты:

```
sudo apt upgrade -y
```

Установим Постгрес:

```
sudo apt install postgresql
```

И посмотрим на статус кластеров:

```
pg_lsclusters
```

```
aeugene@postgres13:~$ pg_lsclusters
Ver Cluster Port Status Owner    Data directory          Log file
12  main     5432 online postgres /var/lib/postgresql/12/main /var/log/postgresql/postgresql-12-main.log
```

Получилась 12 версия Постгрес (действительно на 15.10.2021). Нам же нужна 14.

Если вы всё-таки установили Постгрес этим способом, его можно удалить:

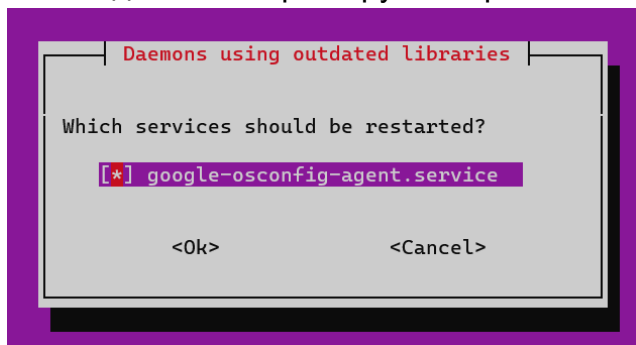
```
sudo apt remove postgresql
```


2. Расширенная установка, где можно выбрать версию дистрибутива, что конкретно устанавливать (сервер, клиент или всё вместе, а также другие расширения). Рекомендованный вариант. Команды установки приложены на [github](#)¹⁹.

Обновим список пакетов и сами пакеты:

```
sudo apt update && sudo apt upgrade -y
```

При обновлении пакетов можем получить предупреждающее окно о необходимости перезагрузки сервиса:



Чтобы сервис автоматически перезапускался добавим переменную окружения в команду `DEBIAN_FRONTEND=noninteractive`:

```
sudo apt update && sudo DEBIAN_FRONTEND=noninteractive apt upgrade -y
```

Также возможно сообщение о доступности новой версии ядра Линукса:

```
Pending kernel upgrade!
Running kernel version:
 5.11.0-1020-gcp
Diagnostics:
 The currently running kernel version is not the expected kernel version 5.11.0-1022-gcp.
Restarting the system to load the new kernel will not be handled automatically, so you should consider rebooting.
[Return]
```

Просто нажимаем ENTER и все. Если мы хотим автоматизировать отказ от обновления ядра, то можно добавить hold на этот пакет:

```
sudo apt-mark hold linux-image-5.11.0-1022-gcp
```

Добавим репозиторий с последней версией Постгреса:

```
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'
```

¹⁹ Установка PostgreSQL 14 [URL](https://github.com/aeuge/Postgres14book/blob/main/scripts/01/install.txt):

<https://github.com/aeuge/Postgres14book/blob/main/scripts/01/install.txt> (дата обращения 07.10.2021) [10]

Добавим ключ репозитория:

```
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc |  
sudo apt-key add -
```

Обновим список пакетов и установим Постгрес. Здесь важный момент какую версию мы хотим, если версию 13, то просто указываем имя пакета postgresql, если конкретную, то еще и имя через дефис:

```
sudo apt-get update && sudo DEBIAN_FRONTEND=noninteractive apt-  
get -y install postgresql-14
```

Для удобства все вышеперечисленные команды соберём в одну, используя символы **&&**:

```
aeugene@postgres14:~$ sudo apt update && sudo DEBIAN_FRONTEND=noninteractive apt upgrade -y && sudo sh -c 'echo "deb htt  
p://apt.postgresql.org/pub/repos/apt $(lsb_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list' && wget --quiet  
-O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add - && sudo apt-get update && sudo DEBIAN_FRONT  
END=noninteractive apt -y install postgresql-14
```

```
aeugene@postgres14:~$ pg_lsclusters  
Ver Cluster Port Status Owner    Data directory          Log file  
14  main     5432  online postgres /var/lib/postgresql/14/main /var/log/postgresql/postgresql-14-main.log
```

Кластер Постгреса успешно установлен и настроен по умолчанию. Также при установке кластер автоматически запускается. Что значат конкретные значения 14, main и т.д. мы разберём в следующей главе “Физический уровень”.

3. Сборка PostgreSQL из пакетов (вариант для гурманов). В основном это предназначено для применения кастомных патчей - они модифицируют исходный код на С для внесения изменений в ядро СУБД. Позволяет значительно расширить функционал или кастомизировать поведение. Единственный минус этого варианта, кроме сложности развертывания, это обновление версии Постгреса - так как исходники, естественно, меняются от версии к версии и нужно заново изменять патч, а если это патч от сторонней организации...

Вот один из примеров кастомного патча, с помощью которого можно закрыть изменение данных в таблице напрямую даже от суперпользователя БД²⁰ - <https://github.com/AbdulYadi/postgresql-private>²¹.

²⁰ База данных

²¹ Github AbdulYadi URL: <https://github.com/AbdulYadi/postgresql-private> (дата обращения 15.10.2021) [11]

Для сборки PostgreSQL из пакетов мы должны выполнить следующие шаги:

Скачиваем нужную версию Постгреса с FTP²². В данном случае нам нужна версия 12.1²³:

wget <https://ftp.postgresql.org/pub/source/v12.1/postgresql-12.1.tar.gz>

Скачаем патч из репозитория:

wget <https://github.com/AbdulYadi/postgresql-private>

Применим патч:

run patch -p0 < path-to-downloaded-patch-file

Соберём Постгрес из исходников, следуя командам из файла INSTALL в каталоге Постгреса:

```
./configure
make
su
make install
adduser postgres
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data
su - postgres
/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
/usr/local/pgsql/bin/pg_ctl -D /usr/local/pgsql/data -l logfile start
```

Как видим, всё не так просто.

Теперь при попытке модифицировать табличку с приватным модификатором получим следующую ошибку:

INSERT INTO public.test VALUES (1, 'abc');

ERROR: do not modify table with "private modify" option outside SQL, PLPGSQL or other SPI-based function

Модифицировать данные сможем только через хранимые функции с нужными правами.

Все предыдущие варианты установки на ОС Linux можно выполнить как на отдельной физической машине, так и на виртуальной. Сейчас и в дальнейшем мы будем использовать VM²⁴ в облаке Google²⁵. Варианты в облаках AWS, Azure и Яндекс Облако будут в дальнейших главах.

²² File Transfer Protocol URL: https://en.wikipedia.org/wiki/File_Transfer_Protocol (дата обращения 07.10.2021) [12]

²³ PostgreSQL 12.1 URL: <https://ftp.postgresql.org/pub/source/v12.1/postgresql-12.1.tar.gz> (дата обращения 07.10.2021) [13]

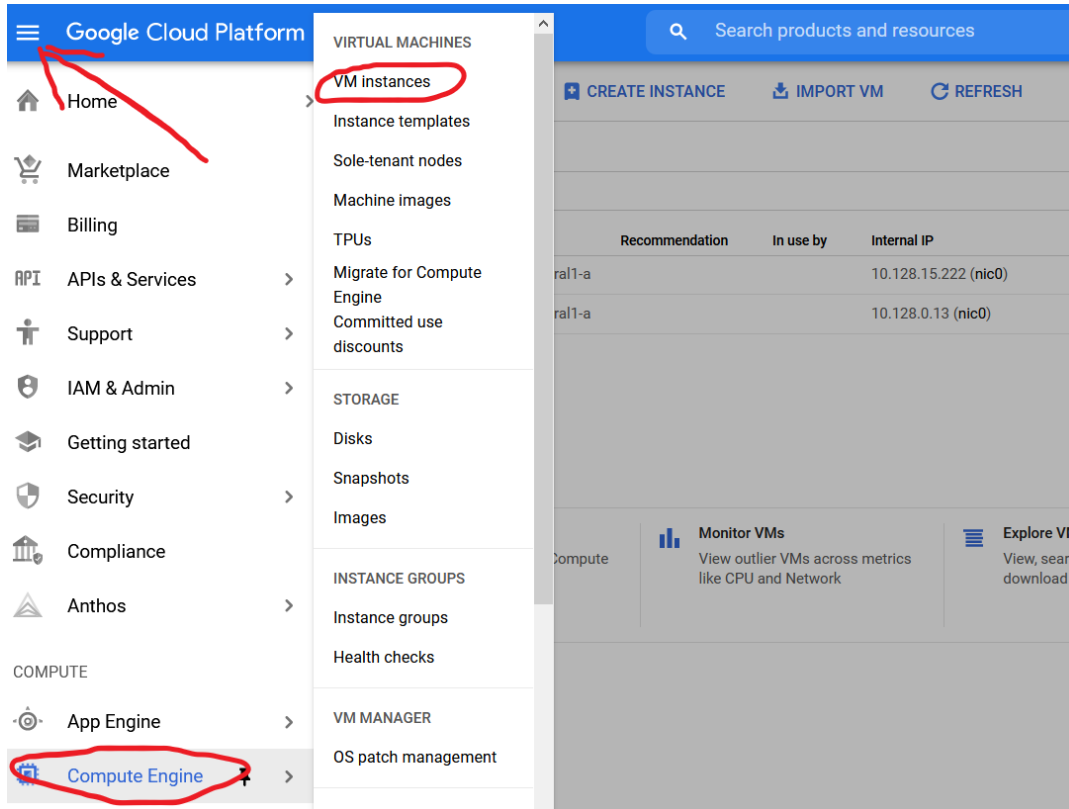
²⁴ Виртуальная машина

²⁵ Google Cloud Platform URL: <https://cloud.google.com/> (дата обращения 07.10.2021) [14]

VM будем разворачивать в пространстве GCE²⁶.

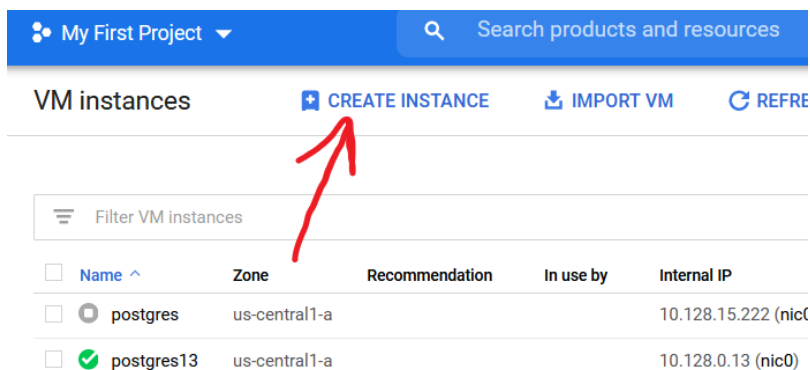
Есть два простых варианта, как развернуть VM:

Первый вариант - прямо через GUI в браузере.



В левом верхнем углу нажимаем на кнопку выбора продукта, затем выбираем Compute Engine и VM Instances.

Далее нажимаем кнопку для добавления VM - **Create Instance**:



²⁶ Google Compute Engine

После этого выбираем тип VM и её характеристики. Чем мощнее VM, тем больше в итоге мы заплатим. Оплата начисляется в зависимости от времени работы VM, так что общая рекомендация - выключать VM, когда не используете.

Name [?]
Name is permanent
Postgres
Name must be lowercase letters, numbers, and hyphens

Labels [?] (Optional)
+ Add label

Region [?]
Region is permanent
us-central1 (low cost)

Zone [?]
Zone is permanent
us-central1-a


Machine configuration

Machine family
General-purpose | Compute-optimized | Memory-optimized | GPU
Machine types for common workloads, optimized for cost and flexibility

Series
E2

CPU platform selection based on availability

Machine type
e2-medium (2 vCPU, 4 GB memory)

	vCPU	Memory	GPUs
	1 shared core	4 GB	-

∨ CPU platform and GPU

\$25.46 monthly estimate
That's about \$0.035 hourly
Pay for what you use: No upfront costs and per second billing
∨ Details

Несколько замечаний: имя инстанса может состоять только из латинских букв в нижнем регистре, цифр и дефисов. Оно должно быть уникально в рамках вашего проекта.

Здесь же можем выбрать регион, а в нём зону, где будет физически расположена VM - в зависимости от этого будет рассчитана стоимость владения. Чем ближе к вам физически будет VM, тем меньше ping²⁷ и выше скорость передачи.

Также можно выбрать количество виртуальных ядер и памяти. По умолчанию предлагается тип VM e2-medium (2 ядра и 4 гигабайта памяти). Его нам хватит для небольшого инстанса Постгреса.

Справа вверху будет указана стоимость за месяц, если VM будет включена 24/7. Если VM выключена, с нас будут списываться деньги только за использование диска.

²⁷ Ping URL: <https://en.wikipedia.org/wiki/Ping> (дата обращения 07.10.2021) [15]

Далее выбираем ОС, размер и тип жёсткого диска нашей VM:

The screenshot shows the 'Deploy Container Engine VM Instance' dialog in the Google Cloud console. The 'Boot disk' section is highlighted with a red circle. The 'Operating system' is set to 'Ubuntu', 'Version' to 'Ubuntu 21.04', 'Boot disk type' to 'SSD persistent disk', and 'Size (GB)' to '10'. The 'SELECT' button is highlighted.

В данном случае мы выбрали Ubuntu 21.04 и SSD²⁸-диск на 10 Гб. Соответственно, чем больше и быстрее выбираем диск, тем больше будет стоимость месячного владения. После выбора можем проверить, насколько увеличилась цифра оплаты за месяц. Остальные настройки VM мы будем рассматривать в дальнейших главах по мере необходимости.

После этого нажимаем кнопку «**Create**» и через пару секунд VM будет создана.

²⁸ SSD URL: https://en.wikipedia.org/wiki/Solid-state_drive (дата обращения 07.10.2021) [16]

Второй вариант: в самом низу, рядом с кнопкой создания ВМ, есть кнопка «**equivalent command line**», нажав которую получим код для выполнения утилитой `cli`²⁹ `gcloud`³⁰.

✓ NETWORKING, DISKS, SECURITY, MANAGEMENT, SOLE-TENANCY

Your free trial credit will be used for this VM instance. [GCP Free Tier](#)

CREATE

CANCEL

EQUIVALENT COMMAND LINE



gcloud command line

This is the gcloud command line with the parameters you have selected. [gcloud reference](#)

```
gcloud compute instances create postgres-14 --project=celtic-house-266612 --zone=us-central1-a --
machine-type=e2-medium --network-interface=network-tier=PREMIUM,subnet=default --maintenance-poli
cy=MIGRATE --service-account=933982307116-compute@developer.gserviceaccount.com --scopes=https://
www.googleapis.com/auth/devstorage.read_only,https://www.googleapis.com/auth/logging.write,http
s://www.googleapis.com/auth/monitoring.write,https://www.googleapis.com/auth/servicecontrol,http
s://www.googleapis.com/auth/service.management.readonly,https://www.googleapis.com/auth/trace.app
end --create-disk=auto-delete=yes,boot=yes,device-name=instance-1,image=projects/ubuntu-os-cloud/
global/images/ubuntu-2104-hirsute-v20210928,mode=rw,size=10,type=projects/celtic-house-266612/zon
es/us-central1-a/diskTypes/pd-ssd --no-shielded-secure-boot --shielded-vtpm --shielded-integrity-
monitoring --reservation-affinity=any
```

Line wrapping



COPY TO CLIPBOARD

RUN IN CLOUD SHELL

CLOSE

И, опять же, есть два варианта:

- нажать **run in cloud shell** - прямо в браузере откроется эмуляция терминала, можно оттуда выполнить команду развёртывания ВМ и в дальнейшем работать через браузерный терминал
- **скопировать команду** и использовать собственный терминал (будем использовать `cli gcloud` в дальнейшем)

²⁹ Command line interface

³⁰ Gcloud URL: <https://cloud.google.com/sdk/gcloud> (дата обращения 07.10.2021) [17]

В нашем случае развернём VM с 2 ядрами и 4 Гб памяти на ОС Ubuntu Hirsute с жёстким диском SSD на 10 Гб:

```
aeugene@Aeuge:~/mnt/c/Users/arist$ gcloud beta compute --project=celtic-house-266612 instances create postgres14 --zone=us-central1-a --machine-type=e2-medium --subnet=default --network-tier=PREMIUM --maintenance-policy=MIGRATE --service-account=933982307116-compute@developer.gserviceaccount.com --scopes=https://www.googleapis.com/auth/devstorage.read_only,https://www.googleapis.com/auth/logging.write,https://www.googleapis.com/auth/monitoring.write,https://www.googleapis.com/auth/servicecontrol,https://www.googleapis.com/auth/service.management.readonly,https://www.googleapis.com/auth/trace.append --image=ubuntu-2104-hirsute-v20210928 --image-project=subuntu-os-cloud --boot-disk-size=10GB --boot-disk-type=pd-ssd --boot-disk-device-name=postgres14 --no-shielded-secure-boot --shielded-vtpm --shielded-integrity-monitoring --reservation-affinity=any
Created [https://www.googleapis.com/compute/beta/projects/celtic-house-266612/zones/us-central1-a/instances/postgres14].
NAME      ZONE      MACHINE_TYPE  PREEMPTIBLE  INTERNAL_IP  EXTERNAL_IP  STATUS
postgres14  us-central1-a  e2-medium      10.128.0.7    34.134.57.202  RUNNING
```

Через несколько секунд получим VM с двумя IP-адресами³¹: внутренним, доступным только в кластере 10.128.* и внешним, доступным из интернета 34.134.57.202. По умолчанию, к только что развёрнутому Постгресу, по сети доступа не имеем, а как открыть доступ обсудим в следующей главе.

Соответственно, чтобы получить **доступ к VM**, есть три пути:

- через браузер и виртуальный терминал
- используя клиент **ssh**³² - доступ по защищённому шифрованному протоколу
- используя утилиту **gcloud + ssh**

Настройка ssh.

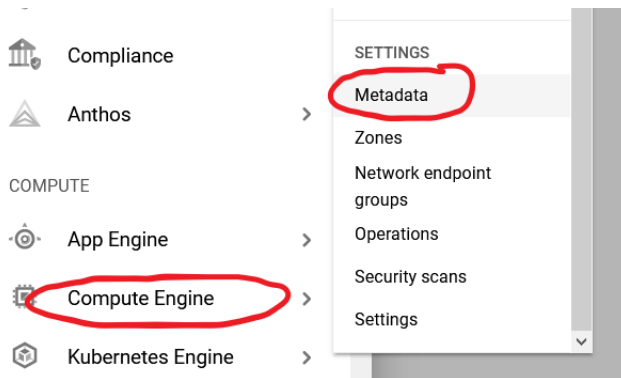
В данном случае настраиваем в Linux. Для Windows и MacOS есть свои ssh-клиенты и настройка примерно такая же:

- на своей машине создаём каталог: **mkdir \$HOME\ssh**
- даём права на него: **chmod 700 \$HOME\ssh**
- генерируем ssh-ключ: **ssh-keygen имя**
- добавляем его: **ssh-add имя**
- выводим на экран и копируем ключ: **cat имя.pub**
- добавляем свой ssh-ключ в GCE metadata

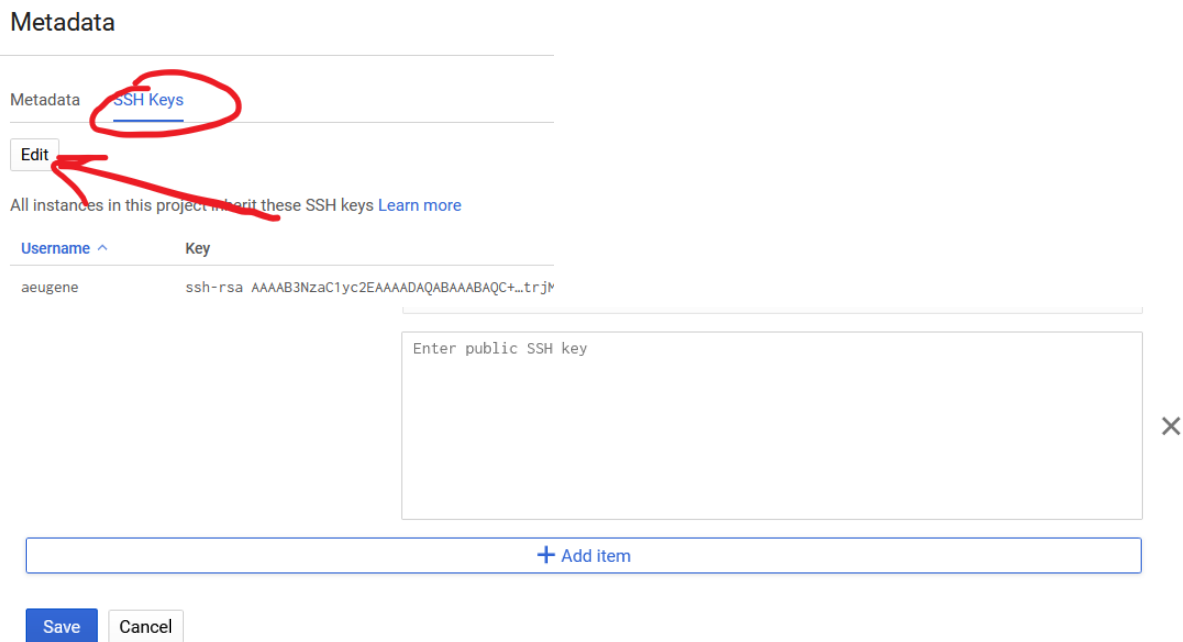
³¹ IP URL: https://en.wikipedia.org/wiki/IP_address (дата обращения 07.10.2021) [18]

³² SSH URL: [https://en.wikipedia.org/wiki/SSH_\(Secure_Shell\)](https://en.wikipedia.org/wiki/SSH_(Secure_Shell)) (дата обращения 07.10.2021) [19]

Для добавления ssh-ключа в GCE необходимо выбирать пункты **Compute Engine -> Metadata**.



Далее вкладка SSH keys, кнопка edit, добавляем свой public ssh-ключ, нажав кнопку add item, и вставляем из буфера обмена в окно наш ключ.



Теперь зайти со своего хоста очень просто:
ssh имя@ip созданной VM

В дальнейших главах будем использовать **вариант** подключения по **ssh**, используя утилиту **gcloud** и имя инстанса:

gcloud compute ssh postgres14

```
aeugene@Aeuge:/mnt/c/Users/arist$ gcloud compute ssh postgres14
No zone specified. Using zone [us-central1-a] for instance: [postgres14].
Welcome to Ubuntu 21.04 (GNU/Linux 5.11.0-1020-gcp x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:       https://ubuntu.com/advantage

System information as of Mon Oct 11 10:28:17 UTC 2021

System load:  0.0          Processes:            113
Usage of /:   20.7% of 9.52GB   Users logged in:    1
Memory usage: 6%           IPv4 address for ens4: 10.128.0.7
Swap usage:   0%

0 updates can be applied immediately.

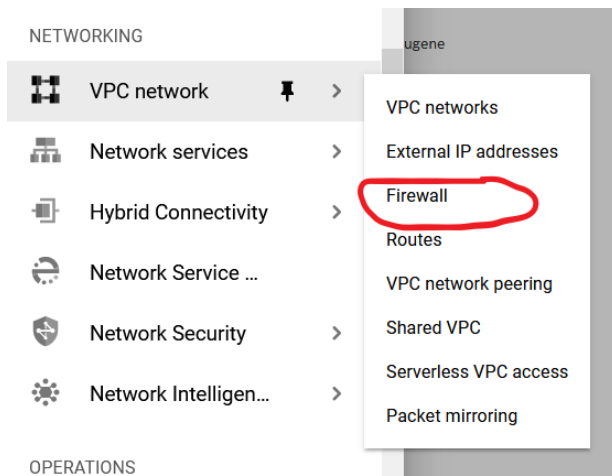
Last login: Mon Oct 11 10:07:43 2021 from 109.252.75.76
aeugene@postgres14:~$
```

Далее просто устанавливаем Постгрес 14, как было описано в главе ранее:

```
sudo apt update && sudo DEBIAN_FRONTEND=noninteractive apt upgrade -y && sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list' && wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add - && sudo apt-get update && sudo DEBIAN_FRONTEND=noninteractive apt -y install postgresql-14
```

После установки Постгреса нам также будет необходимо настроить фаервол в GCP для доступа извне по порту 5432.

Настройка фаервола в Google. Выбираем пункт VPC network -> Firewall:



Далее нажимаем **Create Firewall Rule** и, после задания имени правилу, прописываем, с каких адресов разрешено подключение, для каких экземпляров ВМ и по каким портам³³:

← Create a firewall rule

Deny

Targets
Specified target tags

Target tags *
At least one tag is required

Source filter
IP ranges

Source IP ranges *

Second source filter
None

Protocols and ports ?

Allow all

Specified protocols and ports

tcp : 5432

udp : all

Для начала, нужно или ввести тег машин, для которых открываем доступ, или выбрать пункт - все экземпляры (в нашем случае).

³³ Port URL: [https://en.wikipedia.org/wiki/Port_\(computer_networking\)](https://en.wikipedia.org/wiki/Port_(computer_networking)) (дата обращения 07.10.2021) [20]

Порт Постгреса по умолчанию 5432.

Также вводим маску подсети, откуда разрешён доступ: 0.0.0.0/0.

Нажимаем кнопку “Сохранить”, и через несколько минут доступ извне будет открыт, но при этом у Постгреса есть ещё встроенный фаервол.

Как настроить доступ изнутри Постгреса узнаем в следующей главе.

Предложенные настройки для широкого доступа не рекомендованы на продакшн-системах, мы их используем только для простоты демонстрации наших тестов.

Открытие портов Постгреса в интернет я не рекомендую, максимум, во внутреннюю сеть вашего гугл-проекта.

Использование **PostgreSQL 14** в докере.

Здесь всё просто. Установим докер на машину по инструкции³⁴. Создаём контейнер с 14 Постгресом, указываем пароль, открываем порт 5432 для доступа в контейнер и указываем, куда внутрь контейнера примонтировать³⁵ локальную директорию:

```
sudo docker run --name pg-server -e POSTGRES_PASSWORD=postgres -d -p 5432:5432 -v /var/lib/postgres:/var/lib/postgresql/data postgres:14
```

Проверим подключение используя встроенную утилиту для работы с Постгресом **psql** (более подробно будем изучать в 3 главе), укажем хост и порт для подключения. Ключ **-W** нужен для указания пароля:

```
psql -h localhost -U postgres -W
```

```
aeugene@Aeuge:/mnt/c/Users/arist$ sudo docker run --name pg-server -e POSTGRES_PASSWORD=postgres -d -p 5432:5432 -v /var/lib/postgres:/var/lib/postgresql/data postgres:14
2c442a1e381df1178bc938966b52e0083bda1fe5df1a98fd64781efe8051cd09
aeugene@Aeuge:/mnt/c/Users/arist$ psql -h localhost -U postgres -W
Password:
psql (14.0 (Ubuntu 14.0-1.pgdg20.04+1))
Type "help" for help.
postgres=# |
```

В данном случае мы запустили контейнер с 14 Постгресом и подключились к нему с localhost под пользователем postgres.

Не забываем сопоставить директорию с данными БД в контейнере и локальный каталог, так как все внутренние данные в контейнере будут удалены после рестарта контейнера.

³⁴ Установка docker URL: <https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04-ru> (дата обращения 23.09.2021) [21]

³⁵ Mount URL: [https://en.wikipedia.org/wiki/Mount_\(Unix\)](https://en.wikipedia.org/wiki/Mount_(Unix)) (дата обращения 23.09.2021) [22]

Использовать **docker-compose** для создания кластера Постгреса тоже просто. Устанавливаем docker compose по инструкции³⁶. Создаём файл **docker-compose.yml** со следующим содержимым³⁷:

```
version: '3.1'
volumes:
  pg_project:
services:
  pg_db:
    image: postgres:14
    restart: always
    environment:
      - POSTGRES_PASSWORD=secret
      - POSTGRES_USER=postgres
      - POSTGRES_DB=stage
    volumes:
      - pg_project:/var/lib/postgresql/data
    ports:
      - ${POSTGRES_PORT:-5432}:5432
```

Поднимаем Постгрес, используя docker compose:

docker-compose up -d

- ключ **up** - запустить конфигурацию, описанную в файле docker-compose.yml в текущей директории
- ключ **-d** - запустить в качестве демона (фоновый процесс), а не интерактивно

Обратите внимание на указание версии контейнера с Постгресом:

image: postgres:14

Рекомендую всегда (!!!) указывать версию контейнера, чтобы не было сюрпризов при обновлении версии.

³⁶ Установка docker compose URL: <https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-compose-on-ubuntu-20-04-ru> (дата обращения 23.09.2021) [23]

³⁷ docker-compose.yml URL: <https://github.com/aeuge/Postgres14book/blob/main/scripts/01/docker-compose.yml> (дата обращения 07.10.2021) [24]

Проверим подключение:

psql -h localhost -U postgres -W

```
aeugene@Aeuge:/mnt/c/download$ docker-compose up -d
Creating network "download_default" with the default driver
Creating volume "download_pg_project" with default driver
Creating download_pg_db_1 ... done
aeugene@Aeuge:/mnt/c/download$ psql -h localhost -U postgres -W
Password:
psql (14.0 (Ubuntu 14.0-1.pgdg20.04+1))
Type "help" for help.

postgres=# |
```

Чтобы остановить **docker compose**, выполним:

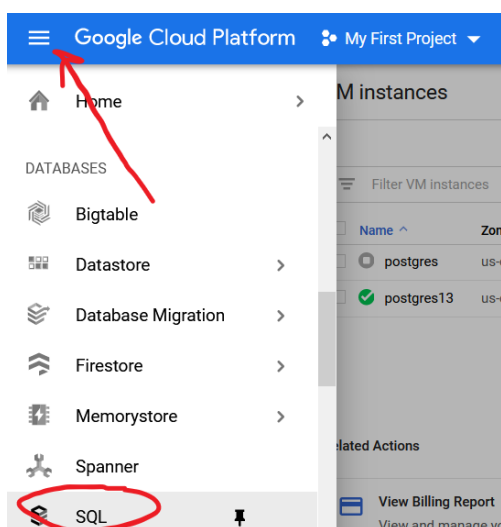
docker-compose down

Про использование **Kubernetes (K8s)** для развёртывания Постгреса у нас будет целая глава далее, так что здесь это рассматривать не будем.

Остаётся вариант **использования Постгреса как DBaaS-решения**. Тема использования БД как сервиса всё больше приобретает популярность. Теперь нам не надо поддерживать ОС, версию Постгреса и другие настройки, нужно только настроить доступ по порту. От нас требуется только схема данных и сами данные. Можем на ходу добавить мощности для БД, увеличить место для хранения или создать реплики для чтения. Более подробно рассмотрим настройки Cloud SQL - версия DBaaS от Google - в соответствующей главе.

Сейчас настроим самый простой инстанс Постгреса как сервиса и протестируем к нему доступ.

Выбираем сервис Cloud SQL:



Далее выбираем **Create Instance**:

MySQL Versions: 5.6, 5.7, 8.0 → Choose MySQL	PostgreSQL Versions: 9.6, 10, 11, 12, 13 → Choose PostgreSQL	SQL Server Versions: 2017 → Choose SQL Server
--	--	---

К сожалению, пока 14 версия недоступна, поэтому выбираем 13 Постгрес:

← Create a PostgreSQL instance

Instance info

Instance ID *
postgres

Use lowercase letters, numbers, and hyphens. Start with a letter.

Password *
●●●●●●

GENERATE

Database version *
PostgreSQL 13

Choose region and zonal availability

For better performance, keep your data close to the services that need it. Region is permanent, while zone can be changed any time.

Region
us-central1 (Iowa)

Single zone
In case of outage, no failover. Not recommended for production.

Multiple zones (Highly available)
Automatic failover to another zone within your selected region. Recommended for production instances. Increases cost.

Summary

Region	us-central1 (Iowa)
DB Version	PostgreSQL 13
vCPUs	4 vCPU
Memory	26 GB
Storage	100 GB
Network throughput	1,000 of 2,000
Disk throughput (MB/s)	Read: 48.0 of 240.0 Write: 48.0 of 240.0
IOPS	Read: 3,000 of 15,000 Write: 3,000 of 15,000
Connections	Public IP
Backup	Automated
Availability	Multiple zones (Highly available)
Point-in-time recovery	Enabled

Обратите внимание на следующие моменты:

- выбирайте регион, где будет расположен инстанс, на котором будет расположен ваш Постгрес, исходя из следующих вопросов:
 - расстояние до региона - чем ближе, тем быстрее
 - стоимость инстанса в данном регионе
- внутри это тоже ВМ, только у вас не будет туда доступа, следовательно, управляться она будет автоматически
- обратите внимание на включённую по умолчанию возможность мультизоны для высокой доступности (high availability). Это автоматически увеличивает стоимость владения в два раза минимум. Рекомендовано для продакшн систем, обеспечивает отказоустойчивость при отказе (failure) primary инстанса и автоматически переключает на secondary. Но при этом наша запасная нода не участвует в обработке запросов. Более тонкая настройка будет показана в соответствующей главе

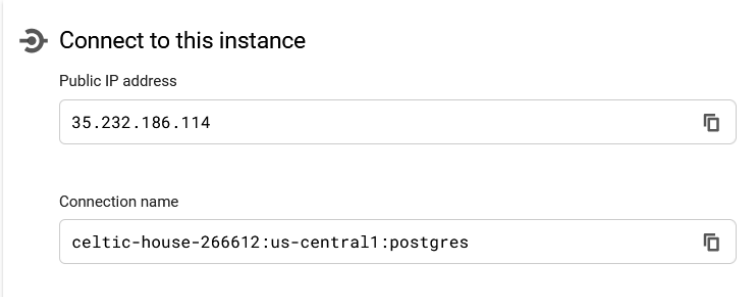
Итого, если обратить внимание на обведённый круг справа (Summary), имеем инстанс со следующими характеристиками:

- 4 ядра
- 26 Гб памяти
- 100 Гб диск
- гарантированные минимальные скорости сети/диска
- внешний IP-адрес
- автоматический бэкап (не бесплатно)
- мультizonную доступность (не бесплатно)
- возможность восстановления во времени PITR³⁸

Сколько же это стоит, спросите вы?

А вот сюрприз - узнаем уже после создания, посмотрев финансовый отчёт!!!

После создания инстанса мы получаем внешний IP. Но чтобы получить к нему доступ, нужно настроить брандмауэр (он же файрвол, он же межсетевой экран)³⁹:



Connect to this instance

Public IP address

35.232.186.114

Connection name

celtic-house-266612:us-central1:postgres

Есть ещё вариант подключения используя **Cloud Proxy**, но его мы будем рассматривать в главе про Google Cloud.

Настроим фаервол.

³⁸ Point in time recovery - восстановление на определенную точку времени

³⁹ Firewall URL: [https://en.wikipedia.org/wiki/Firewall_\(computing\)](https://en.wikipedia.org/wiki/Firewall_(computing)) (дата обращения 07.10.2021) [25]

SQL

Connections

PRIMARY INSTANCE

- Overview
- Query Insights
- Connections**
- Users
- Databases
- Backups
- Replicas
- Operations

All instances > postgres

postgres
PostgreSQL 13

Networking

Choose a network path for connecting to this instance. For extra security, consider using the Cloud SQL proxy. [Learn more](#)

Private IP
Requires additional APIs and permissions, which may require your system admin. Can't be disabled once enabled. [Learn more](#)

Public IP
Authorize a network or use [Cloud SQL Proxy](#) to connect to this instance. [Learn more](#)

i You have not authorized any external networks to connect to your Cloud SQL instance. External applications can still connect to the instance through the Cloud SQL Proxy. [Learn more](#)

Authorized networks

[ADD NETWORK](#)

[SAVE](#) [DISCARD CHANGES](#)

Выберем Connections в меню слева. И видим, что, несмотря на галочку Public IP, публичного доступа у нас нет. Нам **нужно добавить IP-адрес/подсеть**⁴⁰, с которой нам будет разрешён доступ.

Нажмём кнопку **Add Network**:

Public IP
Authorize a network or use [Cloud SQL Proxy](#) to connect to this instance. [Learn more](#)

⚠ You have added 0.0.0.0/0 as an allowed network. This prefix will allow any IPv4 client to pass the network firewall and make login attempts to your instance, including clients you did not intend to allow. Clients still need valid credentials to successfully log in to your instance.

Authorized networks

New network

Name
postgres

Use [CIDR notation](#) [↗](#)

Network *
0.0.0.0/0
Example: 199.27.25.0/24

[CANCEL](#) [DONE](#)

⁴⁰ Маска подсети URL: <https://en.wikipedia.org/wiki/Subnetwork> (дата обращения 07.10.2021) [26]

В данном случае мы разрешили доступ с любого ip 0.0.0.0/0. Повторяю, что так делать на рабочих системах однозначно не стоит. Да и вообще доступ из интернета к БД не рекомендован. Используйте вместо этого доступ из интернета через **backend**⁴¹ по **REST**⁴².

После нажатия кнопок **Done** и **Save**, через пару минут доступ по IP окажется открыт. Давайте проверим:

psql -h 35.232.186.114 -U postgres -W

```
aeugene@Aeuge:/mnt/c/Users/arist$ psql -h 35.232.186.114 -U postgres -W
Password:
psql (12.4 (Ubuntu 12.4-1.pgdg18.04+1), server 13.1)
WARNING: psql major version 12, server major version 13.
Some psql features might not work.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.

postgres=> |
```

Таким образом, мы с вами рассмотрели основные варианты развёртывания Постгреса.

Но у нас остались ещё две затронутые темы по настройке доступа, они будут рассмотрены в следующих главах:

- **настройка ssh** - для обеспечения защищённого подключения
- **настройка firewall в Постгресе** - для доступа к инстансу из интернета

Посмотрим на основные фишки 14 Постгреса⁴³:

- Добавлена поддержка доступа⁴⁴ к данным JSON при помощи выражений, напоминающий работу с массивами:
SELECT ('{ "postgres": { "release": 14 } }::jsonb')['postgres']['release'];
SELECT * FROM test WHERE details['attributes']['size'] = "medium";
- Повышена эффективность работы индексов B-tree и решена проблема с разрастанием индексов при частом обновлении таблиц.
- Внесены оптимизации для повышения производительности высоконагруженных систем, обрабатывающих большое число

⁴¹ Backend URL: https://en.wikipedia.org/wiki/Front_end_and_back_end (дата обращения 07.10.2021) [27]

⁴² REST URL: https://en.wikipedia.org/wiki/Representational_state_transfer (дата обращения 07.10.2021) [28]

⁴³ New features of PostgreSQL 14 URL: <https://www.opennet.ru/opennews/art.shtml?num=55891> (дата обращения 14.10.2021) [29]

⁴⁴ JSONB subscripting URL: <https://www.postgresql.org/docs/14/datatype-json.html#JSONB-SUBSCRIPTING> (дата обращения 14.10.2021) [30]

соединений. В некоторых тестах наблюдается двукратный прирост производительности

- Добавлена поддержка работающего на стороне клиента (реализован на уровне libpq) режима конвейерной (pipeline) передачи запросов, позволяющего значительно ускорить сценарии работы с БД, связанные с выполнением большого числа мелких операций записи (INSERT/UPDATE/DELETE) за счёт отправки следующего запроса, не дожидаясь результата предыдущего. Режим также помогает ускорить работу при соединениях с большими задержками доставки пакетов.
- Расширены возможности для распределённых конфигураций, включающих несколько серверов PostgreSQL. В реализации логической репликации появилась возможность отправки в потоковом режиме транзакций, находящихся в процессе выполнения, что позволяет значительно повысить производительность репликации крупных транзакций. Кроме того, оптимизировано логическое декодирование данных, поступающих в процессе логической репликации.
- В механизме подключения внешних таблиц Foreign Data Wrapper (postgres_fdw) добавлена поддержка параллельной обработки запросов, которая пока применима только при подключении к другим серверам PostgreSQL. В postgres_fdw также добавлена поддержка добавления данных во внешние таблицы в пакетном режиме и возможность импорта секционированных таблиц через указание директивы "IMPORT FOREIGN SCHEMA".

Остальные нововведения можно прочитать в статье по ссылке [28] выше.

Итого, в этой главе мы рассмотрели различные варианты установки Постгреса и в самом конце настроили доступ к нашему инстансу извне, прописав ssh-ключ и создав правило в фаерволе. В следующей главе мы рассмотрим, как открыть доступ к БД изнутри Постгреса.

Все команды, рассмотренные в этой главе, вы можете найти на гитхабе в файле <https://github.com/aeuge/Postgres14book/blob/main/scripts/01/install.txt>

2. Физический уровень

В предыдущей главе мы рассмотрели варианты установки 14 Постгреса. Далее будем рассматривать вариант установки Постгреса на VM под ОС Ubuntu Hirsute.

Все команды из этой главы доступны в файле на гитхабе https://github.com/aeuge/Postgres14book/blob/main/scripts/02/physical_level.txt. Для каждой последующей главы есть свой соответствующий файл.

После установки Постгреса доступ к нему открыт только с локальной машины и только из под линукс-пользователя **root**⁴⁵. Через утилиту `gcloud` посмотрим список VM и заодно узнаем внешний адрес для подключения:

`gcloud compute instances list`

```
aeugene@Aeuge:/mnt/c/download$ gcloud compute instances list
NAME          ZONE          MACHINE_TYPE  PREEMPTIBLE  INTERNAL_IP  EXTERNAL_IP  STATUS
postgres14    us-central1-a  e2-medium                  10.128.0.7   34.134.57.202  RUNNING
```

Можно напрямую подключиться к VM через **ssh** с указанием IP адреса, но для упрощения доступа, чтобы не смотреть каждый раз нужный адрес, мы будем использовать здесь и в дальнейшем команду `gcloud`:

`gcloud compute ssh postgres14`

После подключения посмотрим на список кластеров Постгреса через утилиту **`pg_lsclusters`** (в Постгресе принята такая терминология, что каждый инстанс называется кластером):

`pg_lsclusters`

```
aeugene@postgres14:~$ pg_lsclusters
Ver Cluster Port Status Owner    Data directory          Log file
14  main    5432  online postgres /var/lib/postgresql/14/main /var/log/postgresql/postgresql-14-main.log
```

Видим, что у нас пока развёрнут один кластер **14** версии **Постгреса** по имени **main** на порту **5432**.

Кластер находится онлайн.

Владелец кластера линукс-пользователь **postgres**.

*Обратите внимание, что, несмотря на идентичное имя **postgres** у суперпользователя Постгреса, это другой тип - пользователь ОС Линукс.*

⁴⁵ Пользователь `root` - суперпользователь линукс с максимальными правами.

Также видим, в каком каталоге находятся файлы с базой данных и файлы с логами. Параметры **cluster**, **port**, **data directory** и **log file** должны быть уникальны для одной ВМ.

На самом деле утилит Постгреса **pg_*** в Ubuntu много. Их список можно получить, используя функцию автодописывания команды, набрав **pg_** и нажав кнопку **tab** два раза:

```
aeugene@postgres14:~$ pg_  
pg_archivecleanup  pg_config          pg_dropcluster  pg_lsclusters   pg_renamecluster pg_upgradecluster  
pg_backupcluster  pg_conftool       pg_dump         pg_receivewal   pg_restore        pg_virtualenv  
pg_basebackup     pg_createcluster  pg_dumpall     pg_receivexlog  pg_restorecluster  
pg_buildext       pg_ctlcluster    pg_isready     pg_recvlogical  pg_updatedicts
```

Рассмотрим основные из них:

- **pg_createcluster**⁴⁶ - создать кластер с нужными параметрами
- **pg_renamecluster**⁴⁷ - переименовать кластер
- **pg_dropcluster**⁴⁸ - удалить кластер
- **pg_ctlcluster**⁴⁹ - утилита для запуска и остановки кластера
- **pg_config** - утилита изменения параметров кластера без ручной правки конфиг файлов
- **pg_basebackup**, **pg_dump**, **pg_dumpall**, **pg_receivewal**, **pg_restore** будут рассмотрены в главах по бэкапам и репликации
- **pg_upgradecluster**⁵⁰ - обновить кластер на другую основную версию. На практике посмотрим в конце главы.

⁴⁶ pg_createcluster URL: https://helpmanual.io/help/pg_createcluster/ (дата обращения 10.10.2021) [1]

⁴⁷ pg_renamecluster URL: <https://www.onworks.net/programs/pg-renamecluster-online> (дата обращения 10.10.2021) [2]

⁴⁸ pg_dropcluster URL: http://manpages.ubuntu.com/manpages/trusty/man8/pg_dropcluster.8.html (дата обращения 10.10.2021) [3]

⁴⁹ pg_ctlcluster URL: http://manpages.ubuntu.com/manpages/hirsute/en/man1/pg_ctlcluster.1.html (дата обращения 10.10.2021) [4]

⁵⁰ Pg_upgradecluster URL: https://manpages.ubuntu.com/manpages/trusty/man8/pg_upgradecluster.8.html (дата обращения 10.10.2021) [5]

Например, добавим ещё один кластер:

`pg_createcluster 14 main2`

```
aeugene@postgres14:~$ pg_createcluster 14 main2
install: cannot change permissions of '/etc/postgresql/14/main2': No such file or directory
Error: could not create configuration directory; you might need to run this program with root privileges
```

Обратите внимание на следующую особенность - под нашим простым линукс-пользователем кластер создать не удалось, так как нет доступа к каталогу **`/etc/postgresql/14`**.

Тут есть три варианта:

- переключиться на линукс-пользователя postgres командой **`sudo su postgres`** и из-под него создать кластер
- перейти под линукс-суперпользователя **`root`** командой **`sudo su`**
- использовать утилиту **`sudo`** для поднятия своих прав до прав суперпользователя

Добавим ещё один кластер используя вариант с `sudo`:

`sudo pg_createcluster 14 main2`

```
aeugene@postgres14:~$ sudo pg_createcluster 14 main2
Creating new PostgreSQL cluster 14/main2 ...
/usr/lib/postgresql/14/bin/initdb -D /var/lib/postgresql/14/main2 --auth-local peer --auth-host scram-sha-256 --no-instr
uctions
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "C.UTF-8".
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".

Data page checksums are disabled.

fixing permissions on existing directory /var/lib/postgresql/14/main2 ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ... Etc/UTC
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
syncing data to disk ... ok
Ver Cluster Port Status Owner    Data directory                   Log file
14  main2  5433  down  postgres /var/lib/postgresql/14/main2 /var/log/postgresql/postgresql-14-main2.log
```

Кластер создался, но находится в остановленном состоянии

Важно отметить, что именно на этом этапе задаются дефолтные настройки для кодировок и правил сортировки - локали. В данном случае это UTF8 и C.UTF-8. Порт **5433** задан автоматически. Если нам нужен другой порт - нужно было указать при создании кластера. Теперь для его изменения нужно остановить кластер и поменять параметр **`port`** в файле **`postgresql.conf`**. Например, используя редактор **`nano`**, следующей командой: **`sudo nano /etc/postgresql/14/main/postgresql.conf`**. После внесения изменений нажмите

Ctrl+X и подтвердите изменения в файле, нажав кнопки **Y** и потом **ENTER**. Запустите кластер - кластер будет уже запущен на новой порту.

Посмотрим на физическую структуру каталога с данными в Постгресе. Для этого **зайдём под линукс-пользователем postgres**:

```
sudo su postgres
```

Перейдём в этот каталог и выполним команду:

```
cd /var/lib/postgresql/14/main
```

```
ls -la
```

```
aeugene@postgres14:~$ sudo su postgres
postgres@postgres14:/home/aeugene$ cd /var/lib/postgresql/14/main
postgres@postgres14:~/14/main$ ls -la
total 92
drwx----- 19 postgres postgres 4096 Oct 11 10:14 .
drwxr-xr-x  4 postgres postgres 4096 Oct 11 11:06 ..
-rw-----  1 postgres postgres   3 Oct 11 10:14 PG_VERSION
drwx-----  5 postgres postgres 4096 Oct 11 10:14 base
drwx-----  2 postgres postgres 4096 Oct 11 10:14 global
drwx-----  2 postgres postgres 4096 Oct 11 10:14 pg_commit_ts
drwx-----  2 postgres postgres 4096 Oct 11 10:14 pg_dynshmem
drwx-----  4 postgres postgres 4096 Oct 11 10:19 pg_logical
drwx-----  4 postgres postgres 4096 Oct 11 10:14 pg_multixact
drwx-----  2 postgres postgres 4096 Oct 11 10:14 pg_notify
drwx-----  2 postgres postgres 4096 Oct 11 10:14 pg_replslot
drwx-----  2 postgres postgres 4096 Oct 11 10:14 pg_serial
drwx-----  2 postgres postgres 4096 Oct 11 10:14 pg_snapshots
drwx-----  2 postgres postgres 4096 Oct 11 10:14 pg_stat
drwx-----  2 postgres postgres 4096 Oct 11 10:14 pg_stat_tmp
drwx-----  2 postgres postgres 4096 Oct 11 10:14 pg_subtrans
drwx-----  2 postgres postgres 4096 Oct 11 10:14 pg_tblspc
drwx-----  2 postgres postgres 4096 Oct 11 10:14 pg_twophase
drwx-----  3 postgres postgres 4096 Oct 11 10:14 pg_wal
drwx-----  2 postgres postgres 4096 Oct 11 10:14 pg_xact
-rw-----  1 postgres postgres   88 Oct 11 10:14 postgresql.auto.conf
-rw-----  1 postgres postgres  130 Oct 11 10:14 postmaster.opts
-rw-----  1 postgres postgres  108 Oct 11 10:14 postmaster.pid
postgres@postgres14:~/14/main$ |
```

Здесь нас будет интересовать файл **postgresql.auto.conf**, в котором лежат настройки кластера, которые перезаписывают дефолтные значения. То есть мы во время работы БД изменяем какой-нибудь системный параметр - значение попадает в этот файл и будет применено в последнюю очередь при рестарте инстанса, и перезастрёт значение из файла **postgresql.conf**. Подробнее про эти файлы поговорим в главе про настройки Постгреса.

В каталоге **global** находятся глобальные/системные объекты, которые присутствуют во всех БД кластера. В каталоге **base** расположены файлы с нашими базами данных. Но используются не имена баз данных, а их **OID - object id**:

ls -l base

```
postgres@postgres14:~/14/main$ ls -l base
total 12
drwx----- 2 postgres postgres 4096 Oct 11 10:14 1
drwx----- 2 postgres postgres 4096 Oct 11 10:14 13756
drwx----- 2 postgres postgres 4096 Oct 11 10:14 13757
```

Узнать, какой номер какой БД соответствует, можно, выполнив запрос в **psql**:

psql

SELECT oid, datname, dattablespace FROM pg_database;

```
postgres=# select oid, datname, dattablespace from pg_database;
 oid | datname | dattablespace
-----+-----+-----
 13445 | postgres |          1663
    1 | template1 |          1663
 13444 | template0 |          1663
(3 rows)
```

Здесь видим кроме непонятных баз данных ещё и такую штуку как **dattablespace** - табличное пространство. Давайте разбираться в этом по порядку.

PostgreSQL кластер это несколько баз данных под управлением одного сервера. По умолчанию присутствуют:

- template0
- template1
- postgres

template0 используется:

- для восстановления из резервной копии
- по умолчанию даже нет прав на connect
- не рекомендовано вносить изменения - лучше всего не создавать в ней никаких объектов

template1 используется:

- как шаблон для создания новых баз данных
- в нём имеет смысл делать некие действия, которые не хочется делать каждый раз при создании новых баз данных - создать хранимые функции/процедуры, создать справочники, включить расширения и т.п.

postgres используется:

- первая база данных для регулярной работы
- создаётся по умолчанию
- хорошая практика - также не использовать
- но и не удалять - иногда нужна для различных утилит

Для своих целей создаём свою БД используя **DDL**⁵¹ **create database**⁵²:
CREATE DATABASE book;

Если теперь посмотреть **содержимое** каталога **base**, увидим ещё один новый каталог с нашей базой данных. Теперь поговорим о табличных пространствах.

Табличное пространство это:

- отдельный каталог с точки зрения файловой системы
- лучше делать отдельную файловую систему
- одно табличное пространство может использоваться несколькими базами данных
- каждой базе данных можно назначить как табличное пространство по умолчанию, так и в каждом конкретном случае указывать в каком табличном пространстве будет расположен конкретный объект внутри базы данных

По умолчанию есть два табличных пространства. Давайте на них посмотрим:

SELECT * FROM pg_tablespace;

```
postgres=# select * from pg_tablespace;
 oid | spcname | spcowner | spcacl | spcoptions
-----+-----+-----+-----+-----
 1663 | pg_default |      10 |      | 
 1664 | pg_global  |      10 |      | 
(2 rows)
```

Новые табличные пространства по умолчанию создаются в каталоге **\$PGDATA/pg_tblspc**. Также можем создать свой каталог и создать своё табличное пространство, указав путь к нашему каталогу. Попробуем на практике.

⁵¹ DDL URL: https://en.wikipedia.org/wiki/Data_definition_language (дата обращения 10.10.2021) [6]

⁵² Create database URL: <https://www.postgresql.org/docs/14/sql-createdatabase.html> (дата обращения 10.10.2021) [7]

Создадим каталог из-под суперпользователя линукс **root**, для этого выйдем из-под пользователя **postgres** и зайдём под пользователем **root**. Поменяем его владельца на пользователя линукс **postgres**:

```
exit  
sudo su  
sudo mkdir /home/postgres  
sudo chown postgres /home/postgres
```

Переключимся на пользователя линукс **postgres** и в новом месте создадим каталог для нашего табличного пространства:

```
sudo su postgres  
cd /home/postgres  
mkdir tmptblspc
```

Теперь в утилите **psql** создадим само табличное пространство, где укажем путь ко вновь созданному каталогу:

```
psql  
CREATE TABLESPACE 53 ts location '/home/postgres/tmptblspc';
```

Посмотрим на список табличных пространств:
\db

```
postgres=# \db  
List of tablespaces  
Name | Owner | Location  
-----+-----+-----  
pg_default | postgres |  
pg_global | postgres |  
ts | postgres | /home/postgres/tmptblspc  
(3 rows)
```

Создадим базу данных с настройкой нового табличного пространства по умолчанию:

```
CREATE DATABASE app TABLESPACE ts;
```

Подключимся к созданной БД:
\c app

⁵³ Create tablespace URL: <https://www.postgresql.org/docs/current/sql-createtablespace.html> (дата обращения 10.10.2021) [8]

Чтобы посмотреть дефолтный tablespace, выполним:

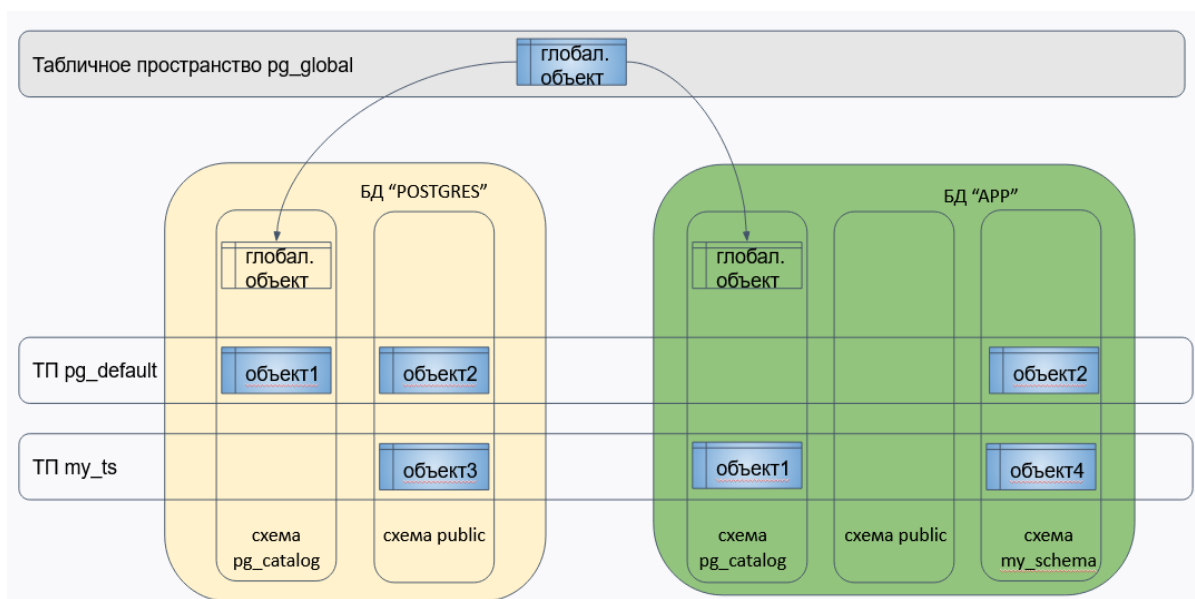
V+

```
List of databases
Name | Owner | Encoding | Collate | Ctype | Access privileges | Size | Tablespace | Description
-----+-----+-----+-----+-----+-----+-----+-----+-----
app | postgres | UTF8 | C.UTF-8 | C.UTF-8 | | 7901 kB | ts |
book | postgres | UTF8 | C.UTF-8 | C.UTF-8 | | 7901 kB | pg_default |
postgres | postgres | UTF8 | C.UTF-8 | C.UTF-8 | | 7909 kB | pg_default | default administrative connection database
template0 | postgres | UTF8 | C.UTF-8 | C.UTF-8 | =c/postgres | 7753 kB | pg_default | unmodifiable empty database
template1 | postgres | UTF8 | C.UTF-8 | C.UTF-8 | postgres=Ctc/postgres | 7901 kB | pg_default | default template for new databases
(5 rows)
(END)
```

Чтобы выйти из полноэкранный режим, нужно нажать кнопку q.

Без конкретного указания табличного пространства при создании таблицы она автоматически будет расположена в tablespace **ts**. Конечно, можно точно указать, где конкретно хранить таблицу.

Вернёмся немного к теории, а именно как хранятся таблицы физически с учётом табличных пространств (ТП) и баз данных:

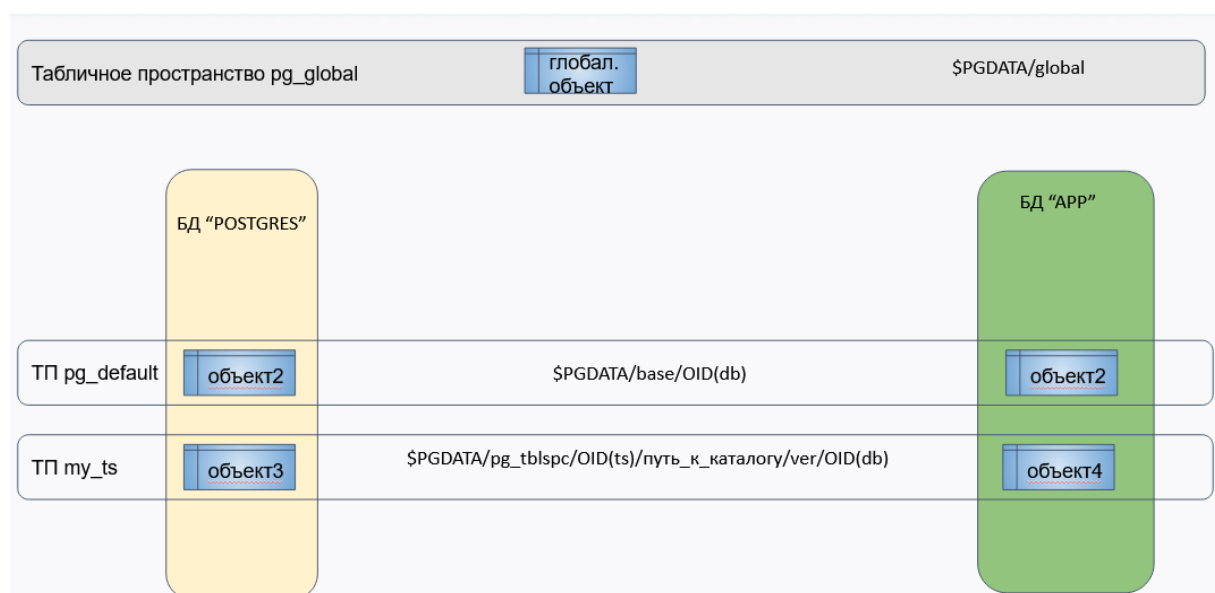


Видим, что объект может принадлежать только к одному табличному пространству (**pg_default** или другое), одной базе данных (**app** или **postgres**) и одной схеме (у каждой БД свой набор схем, про них будем говорить в теме про логическое устройство Постгреса). И только глобальные/системные объекты хранятся отдельно в табличном пространстве **pg_global** и присутствуют логически в каждой БД. При этом, одному объекту в базе данных может

соответствовать от одного и более файлов в зависимости от типа объекта и его размера.

Конечно, можно перемещать объект между табличными пространствами, но нужно понимать, что это будет физическое перемещение файлов между каталогами, и на это время доступа к объекту не будет.

Также внутри каталога с табличным пространством расположены каталоги с базами данных. При этом используются также OID, только уже для БД. Полный путь к объекту выглядит следующим образом:



Перечислю варианты оптимизации производительности с использованием табличных пространств:

- самые часто используемые данные на самые быстрые носители
- редко используемые архивы на медленные
- распараллелить нагрузку по разным рейд массивам
- локальный ssd диск для материальных представлений

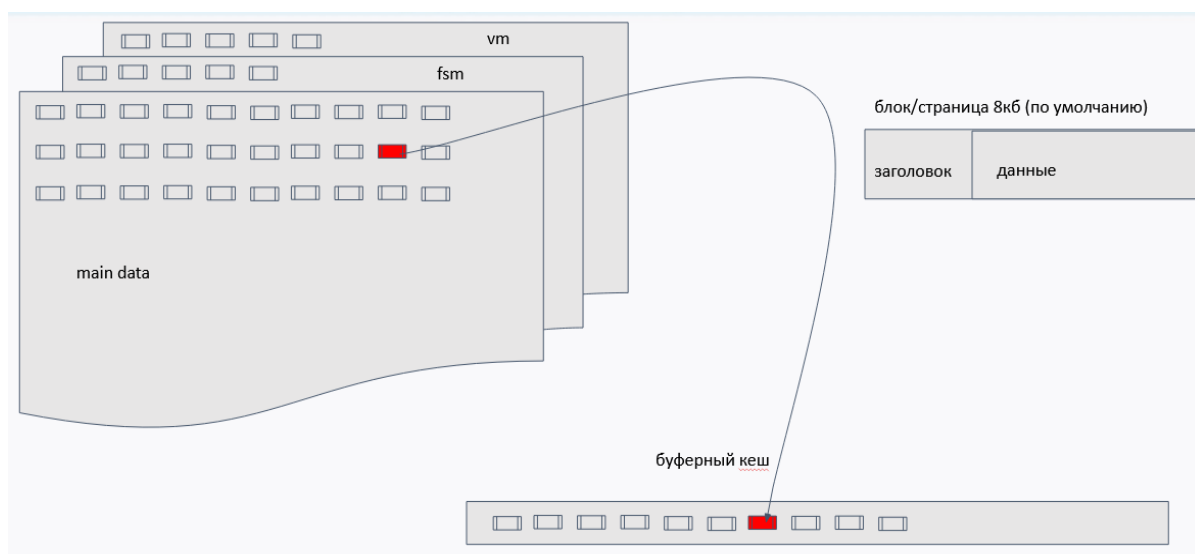
Теперь рассмотрим, как физически хранятся сами таблицы.

Внутри каталога с БД уже расположены непосредственно файлы объектов. Для каждой таблицы создаётся до трёх файлов (если размер сегмента больше 1 Гб, то будут созданы аналогичные файлы с добавлением номера сегмента .1 .2 и т.д.):

- файл с данными - OID таблицы
- файл со свободными блоками - OID_fsm (**free space map**)
 - отмечает свободное пространство в страницах после очистки
 - используется при вставке новых версий строк

- существует для всех объектов
- файл с таблицей видимости - `OID_vm` (**visibility map**)
 - отмечает страницы, на которых все версии строк видны во всех снимках
 - используется для оптимизации работы процесса очистки и ускорения индексного доступа
 - существует только для таблиц
 - иными словами, это страницы, которые давно не изменялись и успели полностью очиститься от неактуальных версий

Схематично можно представить себе это так:



Обратим внимание, что работа с данными идёт не побайтно, а блоками/страницами по 8 Кб⁵⁴ (размер задаётся при инициализации кластера, обычно никто не меняет). Нужные данные извлекаются из файла и загружаются в буферный кэш постранично (более подробно, как в Постгресе организована работа со слоями и кэшем, будем разбирать в следующих темах).

⁵⁴ Кб - килобайты

Необходимо заметить, что вся строка нашей таблицы должна помещаться как раз в 8 Кб, и если она превышает этот размер, то используется специальная TOAST-таблица для хранения больших строк. Тут есть свои тонкости:

- используется схема **pg_toast**
- поддерживается собственным индексом
- читается только при обращении к «длинному» атрибуту
- **стоит задуматься, когда пишем `select * from ...`**
- собственная версионность (если при обновлении toast-часть не меняется, то и не будет создана новая версия toast-части)
- работает прозрачно для приложения

Собственно, в файловой системе созданные каталоги выглядят следующим образом:

```
cd tmptblspc/  
ls  
cd PG_14_202107181/  
ls -la
```

```
postgres@postgres14:/home/postgres$ cd tmptblspc/  
postgres@postgres14:/home/postgres/tmptblspc$ ls  
PG_14_202107181  
postgres@postgres14:/home/postgres/tmptblspc$ cd PG_14_202107181/  
postgres@postgres14:/home/postgres/tmptblspc/PG_14_202107181$ ls  
16385  
postgres@postgres14:/home/postgres/tmptblspc/PG_14_202107181$ ls -la  
total 12  
drwx----- 3 postgres postgres 4096 Oct 11 11:20 .  
drwx----- 3 postgres postgres 4096 Oct 11 11:20 ..  
drwx----- 2 postgres postgres 4096 Oct 11 11:20 16385
```

Видим каталог с БД с OID 16385.

Зайдем в него и посмотрим, как выглядят таблицы в файловой системе:

```
postgres@postgres14:/home/postgres/tmptblspc/PG_14_202107181$ cd 16385/  
postgres@postgres14:/home/postgres/tmptblspc/PG_14_202107181/16385$ ls -l  
total 8564  
-rw----- 1 postgres postgres 8192 Oct 11 11:20 112  
-rw----- 1 postgres postgres 8192 Oct 11 11:20 113  
-rw----- 1 postgres postgres 114688 Oct 11 11:20 1247  
-rw----- 1 postgres postgres 24576 Oct 11 11:20 1247_fsm  
-rw----- 1 postgres postgres 8192 Oct 11 11:20 1247_vm  
-rw----- 1 postgres postgres 450560 Oct 11 11:20 1249  
-rw----- 1 postgres postgres 24576 Oct 11 11:20 1249_fsm  
-rw----- 1 postgres postgres 8192 Oct 11 11:20 1249_vm
```

Видим, что размеры файлов кратны 8 Кб.

Давайте создадим свои таблицы в нашем новом табличном пространстве, попробуем потом на существующей таблице его изменить.

Создадим таблицу в дефолтном табличном пространстве ts:

```
CREATE TABLE test (i int);
```

Создадим таблицу в конкретном табличном пространстве:

```
CREATE TABLE test2 (i int) TABLESPACE pg_default;
```

Посмотрим, кто где создан:

```
SELECT tablename, tablespace FROM pg_tables WHERE schemaname = 'public';
```

```
app=# select tablename, tablespace from pg_tables where schemaname = 'public';
tablename | tablespace
-----+-----
test      |
test2     | pg_default
(2 rows)
```

Необходимо отметить, что для таблицы test в строке tablespace видим пустое значение - означает, что таблица создана в дефолтном для этой базы данных табличном пространстве.

Изменим табличное пространство для таблицы test:

```
ALTER TABLE test SET tablespace pg_default;
```

```
app=# alter table test set tablespace pg_default;
ALTER TABLE
app=# select tablename, tablespace from pg_tables where schemaname = 'public';
tablename | tablespace
-----+-----
test2     | pg_default
test      | pg_default
(2 rows)
```

Также можем посмотреть, где лежит таблица:

```
SELECT pg_relation_filepath('test2');
```

```
app=# SELECT pg_relation_filepath('test2');
pg_relation_filepath
-----
base/16385/16389
(1 row)
```

Давайте теперь посмотрим на размеры таблиц, баз данных и табличных пространств изнутри Постгреса.

Узнать размер, занимаемый базой данных и объектами в ней, можно с помощью ряда функций:

```
SELECT pg_database_size('app');
```

```
app=# SELECT pg_database_size('app');
pg_database_size
-----
            8090159
(1 row)
```

Для упрощения восприятия возможно вывести число в отформатированном виде:

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

```
app=# SELECT pg_size_pretty(pg_database_size('app'));
pg_size_pretty
-----
            7901 kB
(1 row)
```

Полный размер таблицы (вместе со всеми индексами):

```
SELECT pg_size_pretty(pg_total_relation_size('test2'));
```

```
app=# SELECT pg_size_pretty(pg_total_relation_size('test2'));
pg_size_pretty
-----
            0 bytes
(1 row)
```

Остальные по аналогии без скриншотов - отдельно размер таблицы:

```
SELECT pg_size_pretty(pg_table_size('test2'));
```

И индексов:

```
SELECT pg_size_pretty(pg_indexes_size('test2'));
```

При желании можно узнать и размер отдельных слоёв таблицы, например:

```
SELECT pg_size_pretty(pg_relation_size('test2','vm'));
```

Размер табличного пространства показывает другая функция:

```
SELECT pg_size_pretty(pg_tablespace_size('ts'));
```


Общие рекомендации по использованию табличных пространств:

- не хранить данные в корневой файловой системе
- отдельная файловая система для каждого табличного пространства
- разделяя БД на табличные пространства - мы распараллеливаем файловую обработку и ускоряем БД в целом
- в случае внешнего файлового хранилища - отдельный каталог для каждого табличного пространства
- файловые системы EXT3/4 и XFS наиболее популярны

Воспроизведем на практике ситуацию, когда умирает физический носитель с табличным пространством.

Здесь есть два варианта:

- табличное пространство (ТП) было задано для базы данных по умолчанию
- отдельные объекты были размещены в таком ТП

Воспользуемся уже существующей базой данных App.

Создадим еще одно табличное пространство:

exit

cd /home/postgres

mkdir tmptblspc2

psql

CREATE tablespace ts2 location '/home/postgres/tmptblspc2';

```
postgres@postgres14:/home/aeugene$ cd /home/postgres
postgres@postgres14:/home/postgres$ mkdir tmptblspc2
postgres@postgres14:/home/postgres$ psql
psql (14.0 (Ubuntu 14.0-1.pgdg21.04+1))
Type "help" for help.

postgres=# CREATE tablespace ts2 location '/home/postgres/tmptblspc2';
CREATE TABLESPACE
```

Переместим таблицу test во вновь созданное ТП:

`\c app`

`ALTER TABLE test SET TABLESPACE ts2;`

`SELECT tablename, tablespace FROM pg_tables WHERE schemaname = 'public';`

```
postgres=# \c app
You are now connected to database "app" as user "postgres".
app=# ALTER TABLE test SET TABLESPACE ts2;
ALTER TABLE
app=# SELECT tablename, tablespace FROM pg_tables WHERE schemaname = 'public';
tablename | tablespace
-----+-----
test2     | pg_default
test      | ts2
(2 rows)
```

Выйдем из psql и удалим каталог с новым ТП:

`exit`

`rm -rf tmptblspc2`

Зайдем в psql и проверим доступ к БД и список таблиц:

`psql`

`\c app`

`\dt`

```
postgres=# \c app
You are now connected to database "app" as user "postgres".
app=# \dt
      List of relations
Schema | Name | Type | Owner
-----+-----+-----+-----
public | test | table | postgres
public | test2 | table | postgres
(2 rows)
```

Вроде таблица на месте...

Но при попытке получить к ней доступ получим следующую ошибку:

`SELECT * FROM test;`

```
app=# SELECT * FROM test;
ERROR: could not open file "pg_tblspc/16447/PG_14_202107181/16385/16448": No such file or directory
```

Логично, ведь физически каталога теперь не существует.

Значительно более печальная ситуация произойдет, если удалить дефолтное ТП для БД.

Давайте удалим ТП ts и попытаемся зайти в БД app:

```
exit  
rm -rf tmpblspc  
psql  
\c app
```

```
postgres@postgres14:/home/postgres$ rm -rf tmpblspc  
postgres@postgres14:/home/postgres$ psql  
psql (14.0 (Ubuntu 14.0-1.pgdg21.04+1))  
Type "help" for help.  
  
postgres=# \c app  
connection to server on socket "/var/run/postgresql/.s.PGSQL.5432" failed: FATAL: database "app" does not exist  
DETAIL: The database subdirectory "pg_tblspc/16384/Pg_14_202107181/16385" is missing.  
Previous connection kept
```

Вообще не можем зайти в БД, так как у нас также оказались недоступные системные объекты, которые должны присутствовать в каждой БД.

Обратите внимание, что мы также потеряли доступ к таблице **test2**, которая находится в схеме **pg_default**. Чтобы достать оттуда данные придется изрядно постараться.

В любом случае стандартным механизмом восстановления информации является или созданная заранее реплика, откуда мы эти данные сможем достать или вовремя созданный бэкап. Более подробно это будет разбираться в соответствующих главах.

А с поломанной БД остается один вариант - удалить БД:

```
DROP DATABASE app;
```

*Обратите внимание, что для удаления БД необходимо будет еще перезапустить кластер (**sudo pg_ctlcluster 14 main restart**), так как остаются открытые файлы к несуществующему каталогу.*

Мы рассмотрели, как физически расположены и хранятся конфиги и файлы БД.

Давайте теперь посмотрим, как Постгрес выглядит в процессах.

Первый процесс Постгреса - **postgres server process**:

- называется **postgres**
- запускается при старте сервиса
- порождает все остальные процессы используя клон процесса - **fork**
- создаёт **shared memory**
- слушает **TCP** и **Unix socket**

Остальные порождаемые им процессы это:

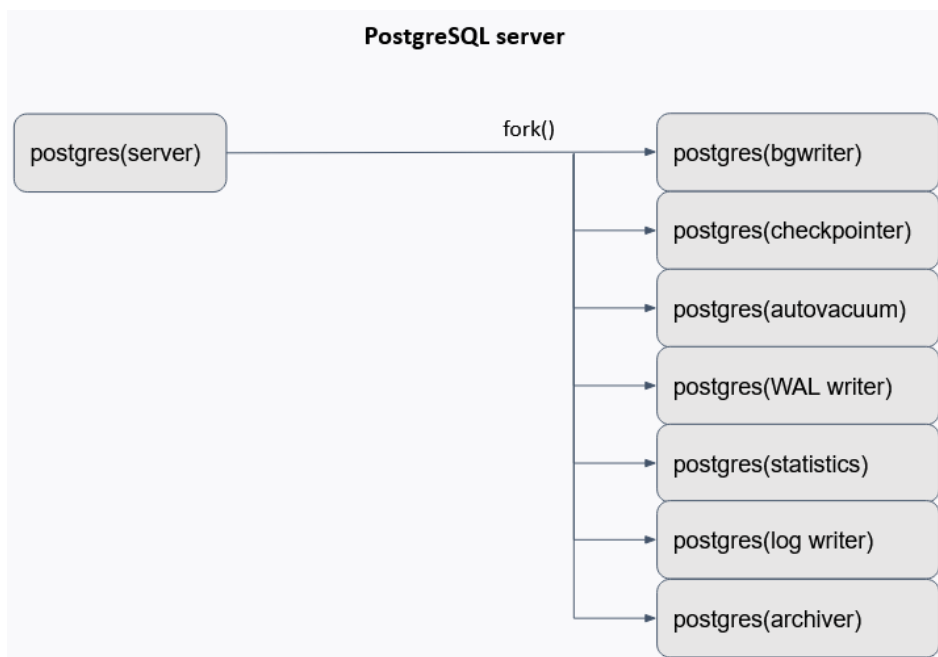
backend processes

- сейчас тоже называется **postgres**
- запускается основным процессом Постгреса
- обслуживает сессию
- работает, пока сессия активна
- максимальное количество определяется параметром **max_connections** (по умолчанию 100)

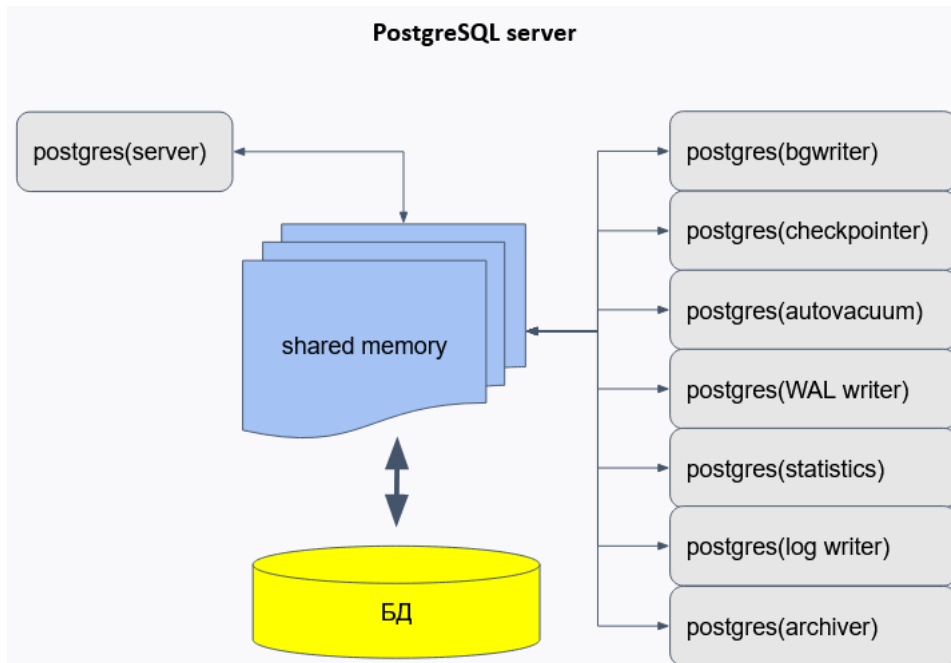
background processes:

- запускаются основным процессом Постгреса при старте сервиса
- выделенная роль у каждого процесса (будем рассматривать в дальнейших главах)
 - **logger** - запись сообщений в лог файл
 - **checkpointer** - запись грязных страниц из buffer cache на диск при наступлении checkpoint
 - **bgwriter** - проактивная запись грязных страниц из buffer cache на диск
 - **walwriter** - запись WAL buffer в WAL file
 - **autovacuum** - периодический запуск autovacuum
 - **archiver** - архивация и репликация WAL
 - **statscollector** - сбор статистики использования по сессиям и таблицам

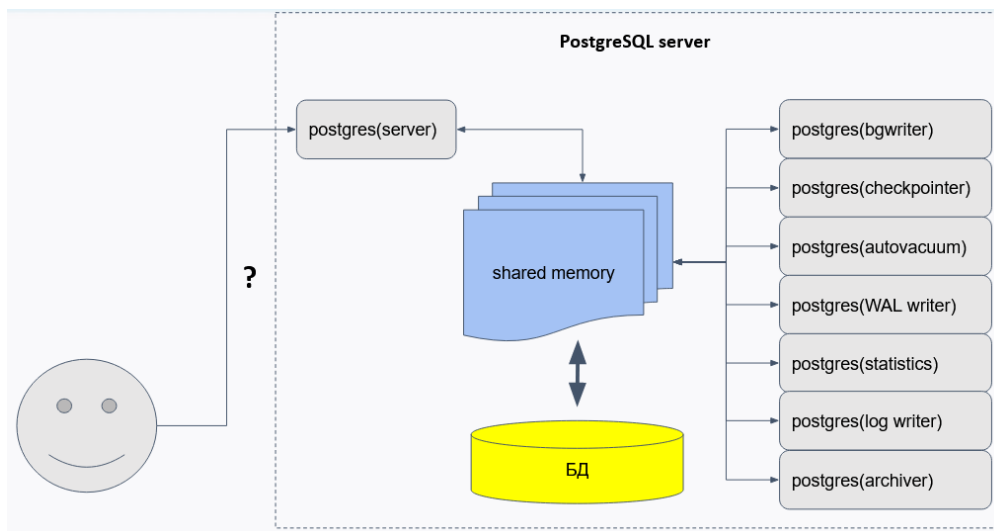
А теперь подробно, что происходит, схематично:



После того, как мы клонировали процессы, они начинают выполнять каждый свою задачу. Далее создаётся общая разделяемая память, которую все эти процессы совместно используют и через неё уже идёт общение с файлами на дисках:

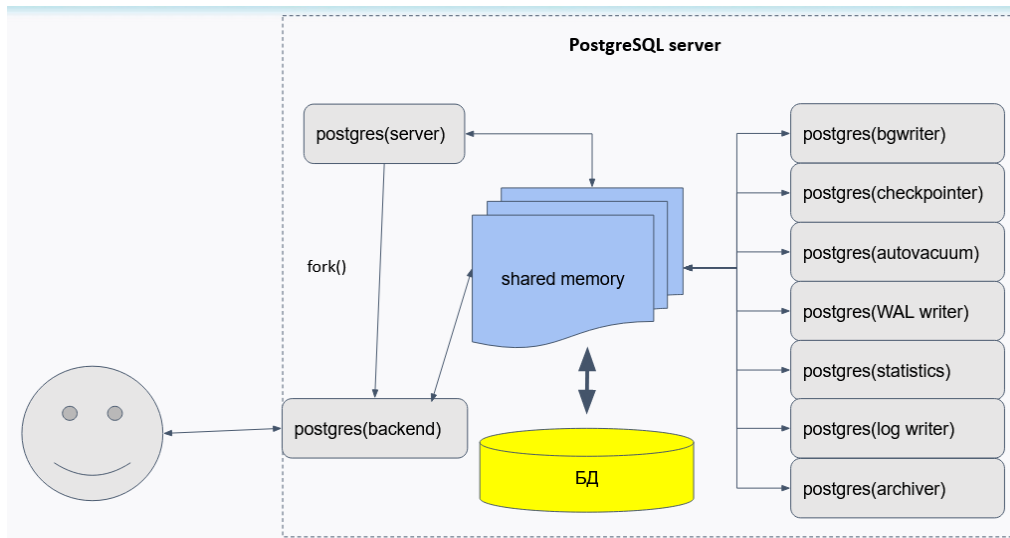


База данных готова к приёму подключений. Пользователь подключается к БД:



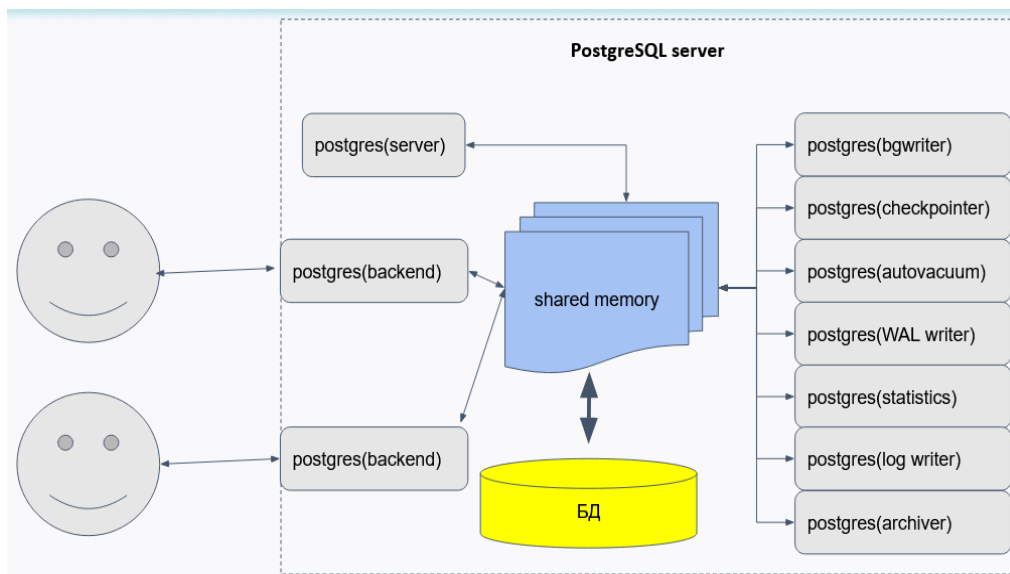
И что, как вы думаете, происходит?

Происходит **fork** процесса postgres, и дальше этот процесс **postgres backend** уже напрямую сам общается с **shared memory**:

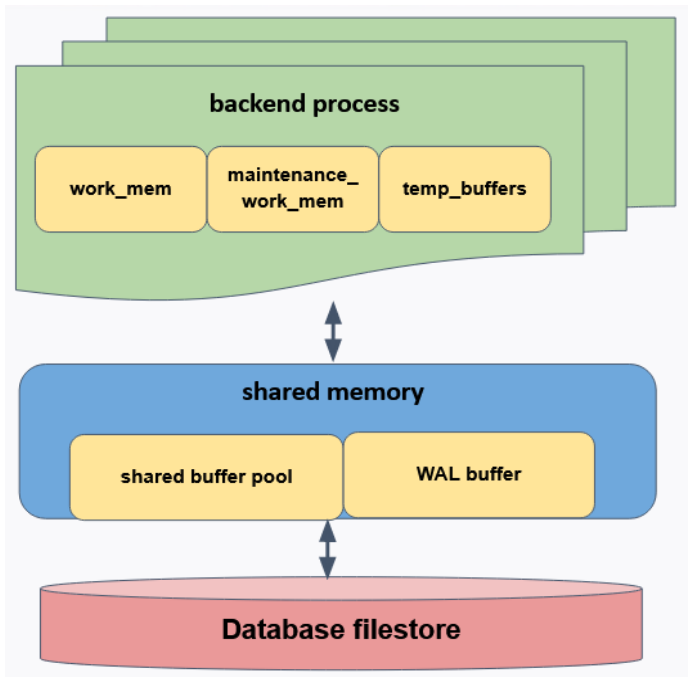


И вот приходит ещё один пользователь. И что теперь?

Снова клон процесса. На каждого пользователя свой бэкэнд-процесс:



Для каждого бэкенд-процесса выделяется своя память:



work_mem (4MB)

- эта память используется на этапе выполнения запроса для сортировок строк, например ORDER BY и DISTINCT

maintenance_work_mem (64MB)

- используется служебными операциями типа VACUUM и REINDEX
- выделяется только при наличии таких команд в сессии

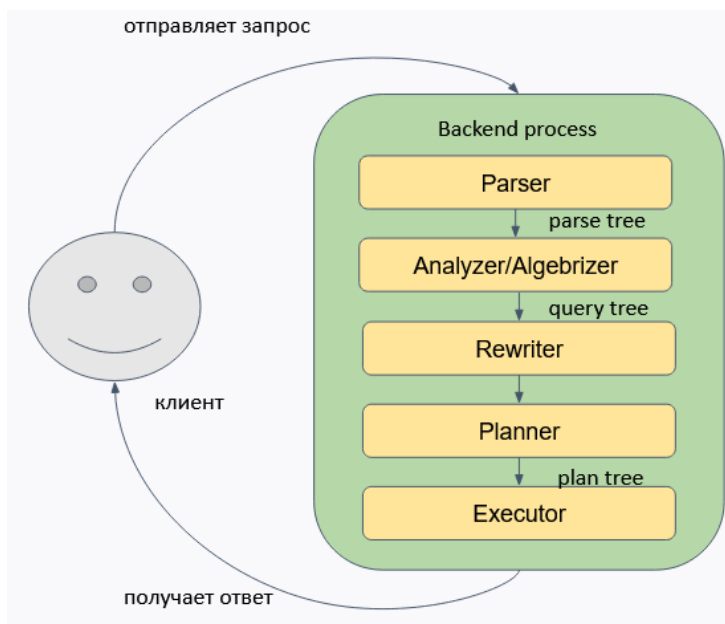
temp_buffers (8MB)

- используется на этапе выполнения для хранения временных таблиц

Все эти параметры настраиваются в зависимости от потребностей.

Shared buffer pool и **WAL buffer** будем рассматривать в дальнейших главах.

Рассмотрим, что происходит внутри сессии при запросе:



При выполнении запроса происходят следующие этапы:

- **Parser** - строится дерево парсинга
- **Analyser/Algebrizer** - строится дерево запросов
- **Rewriter** - переписываем исходный запрос
- **Planner** - строим план выполнения
- **Executor** - и только здесь выполняем запрос

Таким образом вы теперь понимаете, что процедура установки соединения в Постгресе очень дорогая. Именно поэтому рекомендуется использовать пулы подключений, например, **pgpool**⁵⁵.

Теперь, раз заговорили про подключения, настроим доступ к нашему кластеру извне.

⁵⁵ Pgpool URL: https://www.pgpool.net/mediawiki/index.php/Main_Page (дата обращения 10.10.2021) [9]

Так как по умолчанию доступ к Постгресу открыт только с локалхоста⁵⁶, при попытке подключения к ВМ postgres14 по внешнему IP-адресу, например, с локальной машины, используя командную утилиту **psql** (её рассмотрим подробно в следующей главе):

```
aeugene@Aeuge:/mnt/c/download$ gcloud compute instances list
NAME          ZONE          MACHINE_TYPE  PREEMPTIBLE  INTERNAL_IP  EXTERNAL_IP  STATUS
postgres14   us-central1-a e2-medium     10.128.0.7   34.134.57.202 RUNNING
aeugene@Aeuge:/mnt/c/download$ psql -h 34.134.57.202 -U postgres -W
Password:
psql: error: connection to server at "34.134.57.202", port 5432 failed: Connection refused
Is the server running on that host and accepting TCP/IP connections?
```

Ожидаемо, результата не получили, так как по умолчанию пароль не задан и порт не открыт.

Посмотрим на открытые соединения из-под нашего линукс пользователя **aeugene** (у вас, конечно, будет свой пользователь):

netstat -a | grep postgresql

```
aeugene@postgres14:~$ netstat -a | grep postgresql
tcp        0      0 localhost:postgresql  0.0.0.0:*          LISTEN
unix       2      0 [ ACC ] STREAM LISTENING  30195      /var/run/postgresql/.s.PGSQL.5432
```

Видим, что слушаем (LISTENING) мы **только localhost и unix socket**.

Настроим инстанс для доступа извне.

Для открытия доступа к кластеру нужно сделать ряд изменений в настройках Постгреса:

1. Включаем **listener** в **postgresql.conf**, раскомментируем соответствующую строчку, убрав символ **#** в текстовом редакторе **nano**⁵⁷:

sudo nano /etc/postgresql/14/main/postgresql.conf

listen_addresses = '' # IP адреса, на которых Постгрес принимает сетевые подключения, например, localhost, 10.*.*.**

*Установив значение * мы откроем доступ на всех сетевых адаптерах, но так делать на рабочих проектах не стоит.*

*Выход из nano - **Ctrl+X**, потом **Y**, если хотим сохранить изменения и **N**, если отменить, и **ENTER**.*

⁵⁶ localhost или 127.0.0.1 URL: <https://en.wikipedia.org/wiki/Localhost> (дата обращения 10.10.2021) [10]

⁵⁷ nano URL: https://en.wikipedia.org/wiki/GNU_nano (дата обращения 10.10.2021) [11]

Второй вариант через утилиту **psql**, при этом изменение попадёт в файл **postgresql.auto.conf**:

```
SHOW listen_addresses;  
ALTER SYSTEM SET listen_addresses = '*';
```

Третий вариант рассмотрим ближе к концу книги, когда уже будем профессионалами.

2. Указываем вход по паролю в **pg_hba.conf** и меняем маску подсети, откуда будет разрешён доступ к нашему кластеру:

```
sudo nano /etc/postgresql/14/main/pg_hba.conf  
host all all 0.0.0.0/0 scram-sha-256
```

*Здесь мы открыли доступ везде, установив маску 0.0.0.0/0, так делать не стоит и желательно указывать маску максимально узко. Также мы разрешили доступ от имени всех пользователей **all** - желательно тоже подходить к этому вопросу более тщательно.*

Обязательно указываем метод шифрования пароля:

- **scram-sha-256** - современный метод шифрования, доступен с 14 версии
- **md5** - устарел с выходом 14 версии. Передается в зашифрованном виде. Используется для совместимости со старыми клиентами.
- **password** - пароль будет передан в открытом виде и может быть перехвачен злоумышленником. Не рекомендовано!

3. Не забываем добавлять порт в **Google VPC** (рассмотрено в прошлой главе).
4. Задаём пароль пользователю СУБД **postgres** через утилиту **psql**:
ALTER USER postgres PASSWORD 'Postgres123#';

Второй вариант используя встроенную команду **psql**:
\password

5. Перегружаем сервер через утилиту **pg_ctlcluster** из под пользователя **aeugene**:
sudo pg_ctlcluster 14 main restart

Протестируем доступ с ноутбука/компьютера извне GCP:

```
psql -h 34.66.131.159 -U postgres -W
```

```
aeugene@Aeuge:/mnt/c/download$ psql -h 34.134.57.202 -U postgres -W
Password:
psql (14.0 (Ubuntu 14.0-1.pgdg20.04+1))
Type "help" for help.

postgres=# |
```

Вуаля, доступ получен.

Обновление основной версии Постгреса.

Используется, когда мы решаем обновить устаревшую версию на более новую, например, повысить основную версию Постгреса с 13 до 14, используя утилиту **pg_upgradecluster**. Самым огромным недостатком данного метода является значительный даунтайм (время неработоспособности кластера), так как операция выполняется на остановленном кластере.

Посмотрим на практике.

Для того, чтобы создать новый кластер 13 версии Постгреса, необходимо установить эту его версию:

```
sudo DEBIAN_FRONTEND=noninteractive apt -y install postgresql-13
```

Посмотрим список кластеров:

```
pg_lsclusters
```

```
aeugene@postgres14:~$ pg_lsclusters
Ver Cluster Port Status Owner   Data directory          Log file
13  main    5434  online postgres /var/lib/postgresql/13/main /var/log/postgresql/postgresql-13-main.log
14  main    5432  online postgres /var/lib/postgresql/14/main /var/log/postgresql/postgresql-14-main.log
14  main2   5433  online postgres /var/lib/postgresql/14/main2 /var/log/postgresql/postgresql-14-main2.log
```

Видим наш новый кластер main 13 версии.

Попробуем обновить кластер:

```
pg_upgradecluster 13 main
```

```
aeugene@postgres14:~$ pg_upgradecluster 13 main
Error: target cluster 14/main already exists
```

Ожидаемо, ничего не получилось, так как кластер по имени main уже существует для 14 версии Постгреса на данном инстансе.

Посмотрим помощь по команде и обдумаем варианты:
help pg_upgradecluster

```
PG_UPGRADECLUSTER(1)                Debian PostgreSQL infrastructure                PG_UPGRADECLUSTER(1)
NAME
  pg_upgradecluster - upgrade an existing PostgreSQL cluster to a new major version.
SYNOPSIS
  pg_upgradecluster [-v newversion] oldversion name [newdatadir]
DESCRIPTION
  pg_upgradecluster upgrades an existing PostgreSQL server cluster (i. e. a collection of databases served by a postgres instance) to a new version specified by newversion (default: latest available version). The configuration files of the old version are copied to the new cluster and adjusted for the new version. The new cluster is set up to use data page checksums if the old cluster uses them.

  The cluster of the old version will be configured to use a previously unused port since the upgraded one will use the original port. The old cluster is not automatically removed. After upgrading, please verify that the new cluster indeed works as expected; if so, you should remove the old cluster with pg_dropcluster(8). Please note that the old cluster is set to "manual" startup mode, in order to avoid inadvertently changing it; this means that it will not be started automatically on system boot, and you have to use pg_ctlcluster(8) to start/stop it. See section "STARTUP CONTROL" in pg_createcluster(8) for details.
```

Выберем новый каталог для старой версии:
sudo pg_upgradecluster 13 main upgrade13

```
aeugene@postgres14:~$ sudo pg_upgradecluster 13 main upgrade13
Error: target cluster 14/main already exists
```

Ожидаемо, вариант не сработал, так как несмотря на то, что указали другой каталог - имя всё равно должно быть уникально.

Переименуем старую версию и посмотрим, что получилось:
sudo pg_renamecluster 13 main main13
pg_lsclusters

```
aeugene@postgres14:~$ sudo pg_renamecluster 13 main main13
Stopping cluster 13 main ...
Starting cluster 13 main13 ...
aeugene@postgres14:~$ pg_lsclusters
Ver Cluster Port Status Owner    Data directory                   Log file
13  main13  5434  online postgres /var/lib/postgresql/13/main13 /var/log/postgresql/postgresql-13-main13.log
14  main    5432  online postgres /var/lib/postgresql/14/main    /var/log/postgresql/postgresql-14-main.log
14  main2   5433  online postgres /var/lib/postgresql/14/main2   /var/log/postgresql/postgresql-14-main2.log
```

Всё готово для миграции.

Но перед этим проведём ещё один эксперимент - создадим пользователя в 13 версии кластера с шифрованием пароля методом md5 и посмотрим, сможет ли этот пользователь залогиниться в 14 версии, где метод шифрования пароля по умолчанию scram-sha-256.

Создадим пользователя:

```
CREATE ROLE testpass PASSWORD 'testpass' LOGIN;
```

Проверим, что он может подключиться:

```
sudo -u postgres psql -p 5434 -U testpass -h localhost -d postgres -W
```

```
aeugene@postgres14:~$ sudo -u postgres psql -p 5434 -U testpass -W -h localhost -d postgres
could not change directory to "/home/aeugene": Permission denied
Password:
psql (14.0 (Ubuntu 14.0-1.pgdg21.04+1), server 13.4 (Ubuntu 13.4-4.pgdg21.04+1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.

postgres=> \l

                          List of databases
  Name  | Owner  | Encoding | Collate | Ctype  | Access privileges
-----+-----+-----+-----+-----+-----
 postgres | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres
 template0 | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres
         |         |         |         |         | postgres=Ctc/postgres
 template1 | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres
         |         |         |         |         | postgres=Ctc/postgres
(3 rows)
```

Ну и, собственно, обновим кластер до 14 версии:

```
sudo pg_upgradecluster 13 main13
```

```
aeugene@postgres14:~$ sudo pg_upgradecluster 13 main13
Stopping old cluster...
Restarting old cluster with restricted connections...
Notice: extra pg_ctl/postgres options given, bypassing systemctl for start operation
Creating new PostgreSQL cluster 14/main13 ...
/usr/lib/postgresql/14/bin/initdb -D /var/lib/postgresql/14/main13 --auth-local peer --auth-host scram-sha-256 --no-instructions --en
coding UTF8 --lc-collate C.UTF-8 --lc-ctype C.UTF-8
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.
```

Посмотрим, что в итоге получилось:

```
pg_lsclusters
```

```
aeugene@postgres14:~$ pg_lsclusters
Ver Cluster Port Status Owner  Data directory                               Log file
13 main13  5435 down  postgres /var/lib/postgresql/13/main13 /var/log/postgresql/postgresql-13-main13.log
14 main   5432 online postgres /var/lib/postgresql/14/main /var/log/postgresql/postgresql-14-main.log
14 main13 5434 online postgres /var/lib/postgresql/14/main13 /var/log/postgresql/postgresql-14-main13.log
14 main2  5433 online postgres /var/lib/postgresql/14/main2 /var/log/postgresql/postgresql-14-main2.log
```

Всё! Теперь у нас новый кластер на 14 версии, старый 13 версии можно удалять:

```
sudo pg_dropcluster 13 main13
```

Посмотрим на настройки доступа к БД в новой версии кластера:

```
sudo cat /etc/postgresql/14/main13/pg_hba.conf
```

```
# IPv4 local connections:
host    all             all             127.0.0.1/32      md5
# IPv6 local connections:
host    all             all             ::1/128           md5
```

Первый сюрприз - для совместимости оставлен старый метод шифрования.

Поменяем метод шифрования на принятый в 14 Постгресе scram-sha-256:

```
sudo nano /etc/postgresql/14/main13/pg_hba.conf
```

```
# IPv4 local connections:
host    all             all             127.0.0.1/32      scram-sha-256
```

Перезагрузим сервер и проверим доступ:

```
sudo pg_ctlcluster 14 main restart
```

```
sudo -u postgres psql -p 5434 -U testpass -h localhost -d postgres -W
```

```
aeugene@postgres14:~$ sudo pg_ctlcluster 14 main13 restart
aeugene@postgres14:~$ sudo -u postgres psql -p 5434 -U testpass -h localhost -d postgres -W
could not change directory to "/home/aeugene": Permission denied
Password:
psql: error: connection to server at "localhost" (127.0.0.1), port 5434 failed: FATAL: password authentication failed for user "testpass"
connection to server at "localhost" (127.0.0.1), port 5434 failed: FATAL: password authentication failed for user "testpass"
```

Ожидаемо, доступ не получили.

Шифрование md5 и scram-sha256 не совместимы при апгрейде версии!

Если зайдём под пользователем **postgres**, поменяем пароль для **testpass** и он уже будет зашифрован новым алгоритмом, то спокойно будем работать:

```
sudo -u postgres psql -p 5434
```

```
ALTER USER testpass PASSWORD 'testpass';
```

```
exit
```

```
sudo -u postgres psql -p 5434 -U testpass -h localhost -d postgres -W
```

```

aeugene@postgres14:~$ sudo -u postgres psql -p 5434
could not change directory to "/home/aeugene": Permission denied
psql (14.0 (Ubuntu 14.0-1.pgdg21.04+1))
Type "help" for help.

postgres=# alter user testpass password 'testpass';
ALTER ROLE
postgres=# exit
aeugene@postgres14:~$ sudo -u postgres psql -p 5434 -U testpass -h localhost -d postgres -W
could not change directory to "/home/aeugene": Permission denied
Password:
psql (14.0 (Ubuntu 14.0-1.pgdg21.04+1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.

postgres=> |

```

При этом, если вернуть настройку **md5** обратно в **pg_hba.conf**, то всё продолжит работать прекрасно! То есть у нас есть обратная совместимость.

Посмотрим на практике:

sudo nano /etc/postgresql/14/main13/pg_hba.conf

sudo pg_ctlcluster 14 main13 restart

sudo -u postgres psql -p 5434 -U testpass -h localhost -d postgres -W

```

aeugene@postgres14:~$ sudo nano /etc/postgresql/14/main13/pg_hba.conf
aeugene@postgres14:~$ sudo pg_ctlcluster 14 main13 restart
aeugene@postgres14:~$ sudo -u postgres psql -p 5434 -U testpass -h localhost -d postgres -W
could not change directory to "/home/aeugene": Permission denied
Password:
psql (14.0 (Ubuntu 14.0-1.pgdg21.04+1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.

postgres=> |

```

Всё работает.

Для удаления тестового кластера мы должны сначала его остановить, а потом уже удалять:

sudo pg_ctlcluster 14 main13 stop

sudo pg_dropcluster 14 main13

В этой главе мы рассмотрели физический уровень - как и в каких файлах хранятся данные, как устроен Постгрес в физических процессах и как организовать физический доступ к кластеру Постгреса извне. Увидели, как можно оптимизировать работу, используя отдельные табличные пространства. Обновили наш кластер до новой версии. В следующей главе рассмотрим интерактивную утилиту **psql**, с помощью которой можно полноценно работать с Постгресом.

** Задача на самостоятельное закрепление материала:
Памяти у инстанса 4 Gb (периодически приходил OOM killer⁵⁸)*

```
max_connections = 1000          # (change requires restart)
shared_buffers = 6GB           # min 128kB
work_mem = 16MB                # min 64kB
maintenance_work_mem = 256MB   # min 1MB
```

Что не так с параметрами?

Все команды, рассмотренные в этой главе, вы можете найти на гитхабе в файле https://github.com/aeuge/Postgres14book/blob/main/scripts/02/physical_level.txt

⁵⁸ Out of Memory Killer URL: <https://habr.com/ru/company/southbridge/blog/464245/> (дата обращения 10.10.2021) [11]

3. Работа с консольной утилитой psql

psql⁵⁹ - интерактивный терминал Постгрес. Это так называемая **cli - command line interface**. Позволяет полноценно общаться с кластером Постгреса.

Полный синтаксис вызова psql смотреть не будем, так как если посмотреть помощь - **info psql**, то там больше 2000 строк.

Рассмотрим основные опции:

- h - хост, к которому подключаемся
- U - пользователь, под которым заходим
- W - интерактивный ввод пароля
- p - порт кластера Постгреса
- d - база данных для подключения

После подключения к БД видим справочную информацию:

```
aeugene@Aeuge:/mnt/c/download$ psql -h 34.134.57.202 -U postgres -W
Password:
psql (14.0 (Ubuntu 14.0-1.pgdg20.04+1))
Type "help" for help.

postgres=# |
```

- версию сервера 14.0
- версию клиента для подключения 14.0
- версию ОС клиента - Ubuntu 20.04

Обратите внимание, что наше подключение использует зашифрованное соединение **SSL**⁶⁰ по протоколу **TLS**⁶¹ (до 14 версии эта информация выводилась в консоль при подключении).

Далее видим приглашение для ввода **#**. Можно использовать различные варианты команд, как встроенных в psql, так и сами команды SQL:

- \?** список команд psql
- \? variables** переменные psql
- \h[elp]** список команд SQL
- \h команда** синтаксис команды SQL
- \q** выход (до 11 версии), сейчас можно использовать команду **exit**

⁵⁹ psql URL: <https://www.postgresql.org/docs/14/app-psql.html> (дата обращения 10.10.2021) [1]

⁶⁰ SSL URL: https://en.wikipedia.org/wiki/Secure_Sockets_Layer (дата обращения 10.10.2021) [2]

⁶¹ TLS URL: https://en.wikipedia.org/wiki/Transport_Layer_Security (дата обращения 10.10.2021) [3]

Приведу здесь небольшой список наиболее полезных и часто встречающихся команд:

- `\l` - список баз данных
- `\dp` (или `\z`) - список таблиц, представлений, последовательностей, прав доступа к ним
- `\di` - список индексов
- `\dt` - список таблиц
- `\dt+` - список всех таблиц с описанием
- `\dt *s*` - список всех таблиц, содержащих `s` в имени
- `\d+` - описание таблицы
- `\d "table_name"` - описание таблицы
- `\du` - список пользователей
- `\set` - устанавливает значение переменной среды. Без параметров выводит список текущих переменных (`\unset` - удаляет)
- `\i` - запуск команды из внешнего файла, например `\i /home/test/my.sql`

Давайте более подробно остановимся на запуске команд из внешнего файла. Для этого сначала зайдём в Линукс под пользователем postgres:

`sudo su postgres`

Зайдем в консольную утилиту для работы с Постгресом:

`psql`

Подключимся к созданной нами базе данных `app`:

`\c app`

Посмотрим, какие таблицы в ней существуют:

`\dt`

Выполним простейшую SQL-команду:

`SELECT * FROM test;`

```
book=# \c app
Password:
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
You are now connected to database "app" as user "postgres".
app=# \dt
          List of relations
 Schema | Name  | Type  | Owner
-----+-----+-----+-----
 public | test  | table | postgres
 public | test2 | table | postgres
(2 rows)

app=# select * from test;
 i
---
(0 rows)

app=#
```

Затем выйдем из **psql**, запишем эти команды в файл через редактор **nano**. При этом **файлы будем создавать в каталоге, куда есть доступ к линукс-пользователя postgres**, так как **psql запускаем** под этим пользователем и, соответственно, нам нужен доступ к файловой системе именно под линукс-пользователем **postgres**.

Выходим из psql:
exit

После этого переходим в домашний каталог линукс-пользователя postgres:

cd \$HOME

Посмотрим, где он расположен:
pwd

Видим, что он как раз расположен по адресу **/var/lib/postgresql**, где и находятся файлы нашего кластера по умолчанию. Посмотрим содержимое каталога:

ls -l

```
aeugene@postgres14:~$ sudo su postgres
postgres@postgres14:/home/aeugene$ cd $HOME
postgres@postgres14:~$ pwd
/var/lib/postgresql
postgres@postgres14:~$ ls -l
total 4
drwxr-xr-x 4 postgres postgres 4096 Oct 11 11:06 14
postgres@postgres14:~$ ls -l 14/
total 8
drwx----- 19 postgres postgres 4096 Oct 11 11:53 main
drwx----- 19 postgres postgres 4096 Oct 11 11:06 main2
```

Видим каталог **14**, в котором, как помним, есть подкаталоги **main** и **main2** с данными кластеров.

Создадим каталог со скриптами:
mkdir scripts

Перейдём в него:
cd scripts

Создадим файл select.txt:
nano select.sql

Запишем туда наши три команды.

Зайдём обратно в psql и выполним скрипт, используя команду **psql \i**:
\i /var/lib/postgresql/scripts/select.sql

```
postgres=# \i /var/lib/postgresql/scripts/select.sql
You are now connected to database "app" as user "postgres".
      List of relations
Schema | Name  | Type  | Owner
-----+-----+-----+-----
public | test  | table | postgres
public | test2 | table | postgres
(2 rows)

i
---
(0 rows)

app=# |
```

Видим, что наш скрипт прекрасно отработал. Таким образом можем заранее заготавливать нужные нам скрипты или подготавливать миграции данных.

При **наборе команд в psql** нужно обращать внимание на правильность ввода команд. Об этом свидетельствует знак “=” после имени БД, к которой мы подключены, и перед “#”.

Например, если забыли добавить символ “;” в конец SQL-команды, то оболочка **psql** будет ждать окончания ввода команды для дальнейшего запуска парсера и т.д. Увидеть это можно, обратив внимание на изменение знака “=” на знак “-”:

```
app=# select * from test
app-#
app-#
app-# ;
i
---
(0 rows)

app=#
```

В данном случае **psql** ждал окончания ввода команды - нужно было или добавить точку с запятой или **превать ввод команды нажав Control + C**.

Важный момент, что при запуске **psql** выполняются команды, записанные в двух файлах - общесистемном и пользовательском. Общий системный файл называется **psqlrc** и располагается в каталоге по умолчанию: **/etc/postgresql-common** и **/usr/local/pgsql/etc** при обычной сборке из исходных кодов.

Расположение этого каталога можно узнать командой:

`pg_config --sysconfdir`

```
postgres@postgres13:~$ pg_config --sysconfdir
/etc/postgresql-common
postgres@postgres13:~$
```

Обратите внимание на **\$** в приглашении командной строки - командная строка Линукс. Внутри **psql** приглашение в виде **#** для пользователя root и **>** для обычного пользователя.

Пользовательский файл находится в домашнем каталоге пользователя ОС и называется **.psqlrc**. Его расположение можно изменить, задав переменную окружения PSQLRC.

В эти файлы можно записать команды, настраивающие **psql**. Например, изменить приглашение, включить вывод времени выполнения команд и т.п.

История вводимых команд сохраняется в файле **.psql_history** в домашнем каталоге пользователя. По умолчанию хранится 500 последних команд, это число можно изменить переменной `psql HISTSIZE`.

Также зачастую в ситуации, когда много колонок в нашей таблице, вывод на экран очень неинформативен:

`SELECT * FROM pg_stat_activity;`

```
datid | datname | pid | leader_pid | usesysid | username | application_name | client_addr | client_hostname | client_
port |      backend_start      |      xact_start      |      query_start      |      state_cha
nge |      wait_event_type     |      wait_event      |      state            |      backend_xid | backend_xmin |      query
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
      |      | 684 |      |      |      |      |      |      |      |
2021-03-12 12:12:45.457754+00 |      |      |      |      |      |      |      |      |
      | Activity | AutoVacuumMain |      |      |      |      |      |      |
autovacuum launcher |      |      |      |      |      |      |      |      |
      | 686 |      |      |      |      |      |      |      |
2021-03-12 12:12:45.458728+00 |      |      |      |      |      |      |      |      |
      | Activity | LogicalLauncherMain |      |      |      |      |      |
logical replication launcher |      |      |      |      |      |      |      |      |
13445 | postgres | 2939 |      |      |      | psql |      |      |      |
-1 | 2021-03-12 13:16:54.634896+00 | 2021-03-12 13:16:56.471017+00 | 2021-03-12 13:16:56.471017+00 | 2021-03-12 13:16:
56.471021+00 |      |      |      |      |      |      |      |      |      |
498 | select * from pg_stat_activ
ity; | client backend
```

Можем включить расширенный вывод информации - вертикальный вывод колонок:

`\x`

`SELECT * FROM pg_stat_activity;`

```
-[ RECORD 1 ]-----+-----
datid
datname
pid           684
leader_pid
usesysid
username
application_name
client_addr
client_hostname
client_port
backend_start 2021-03-12 12:12:45.457754+00
xact_start
query_start
state_change
wait_event_type | Activity
wait_event      | AutoVacuumMain
state
backend_xid
backend_xmin
query
backend_type   | autovacuum launcher
-[ RECORD 2 ]-----+-----
datid
```

Согласитесь, намного более наглядный вывод.

Повторный ввод команды `\x` отменит расширенный вывод:

`\x`

Также есть вариант указания `\gx` в конце строки вместо точки с запятой - получим расширенный вывод для этой конкретной команды.

В этой главе мы рассмотрели основы использования консольной утилиты `psql`. Переходим к изучению ACID, MVCC и всё, что с этим связано.

Все команды, рассмотренные в этой главе, вы можете найти на гитхабе в файле <https://github.com/aeuge/Postgres14book/blob/main/scripts/03/psql.txt>

4. ACID & MVCC. Vacuum и autovacuum

Реляционная теория и SQL позволяет абстрагироваться от конкретной реализации СУБД, но есть одна непростая проблема: **как обеспечить параллельную работу множества сессий (concurrency), которые модифицируют данные, так, чтобы они не мешали друг другу ни с точки зрения чтения, ни с точки зрения записи и обеспечивали целостность данных (consistency) и их надежность (durability)?**

Ответ - транзакционные системы **OLTP**⁶² - Online Transaction Processing. Они отвечают следующему принципу **ACID**⁶³:

- **Atomicity** - атомарность
- **Consistency** - согласованность
- **Isolation** - изолированность
- **Durability** - долговечность

Соответственно, транзакция (**transaction**) это:

- множество операций, выполняемое приложением
- которое переводит базу данных из одного корректного состояния в другое корректное состояние (согласованность)
- при условии, что транзакция выполнена полностью (атомарность)
- и без помех со стороны других транзакций (изолированность)

Всё было бы хорошо, но что делать с блокировками и сбоями?

Для этого придуманы следующие принципы:

ARIES⁶⁴ - Algorithms for Recovery and Isolation Exploiting Semantics. Алгоритмы эффективного восстановления после сбоев.

Использует следующие механизмы:

- logging - логирование
- undo - сегменты отката транзакций
- redo - сегменты проигрывания транзакций после сбоя
- checkpoints - система контрольных точек

⁶² OLTP URL: https://en.wikipedia.org/wiki/Online_transaction_processing (дата обращения 13.10.2021) [1]

⁶³ ACID URL: <https://en.wikipedia.org/wiki/ACID> (дата обращения 13.10.2021) [2]

⁶⁴ ARIES URL: https://en.wikipedia.org/wiki/Algorithms_for_Recovery_and_Isolation_Exploiting_Semantics (дата обращения 13.10.2021) [3]

MVCC⁶⁵ - MultiVersion Concurrency Control. Конкурентный контроль мультиверсионности. Использует следующие механизмы:

- сору-он-write - создаётся копия старых данных при записи или модификации
- каждый пользователь работает со снимком БД
- вносимые пользователем изменения не видны другим до фиксации транзакции
- практически не использует блокировок (только одна - писатель блокирует писателя, если они пытаются работать с одной записью)
- или ручная блокировка при вызове команды select for update

Эти все механизмы рассмотрим в этой и последующих главах.

Механизм реализации MVCC в PostgreSQL.

Данные хранятся поблочно и посмотрим, из чего состоит один блок:

HeapTupleHeader data									
xmin	xmax	cmin	cmax	t_cid	t_ctid	t_infomask	t_infomask2	Null bitmap	Data

В самом начале идёт служебный блок, состоящий из:

- пары идентификаторов транзакций по 4 байта
- ряда информационных байтов
- пары информационных масок
- пустой битовой карты для будущего использования в следующих версиях Постгреса
- данных (более подробно будем рассматривать в следующих темах).

Если рассматривать подробно, то заголовок состоит из:

- **xmin** - идентификатор транзакции, которая создала данную версию записи
- **xmax** - идентификатор транзакции, которая удалила данную версию записи
- **cmin** - порядковый номер команды в транзакции, добавившей запись
- **cmax** - номер команды в транзакции, удалившей запись

⁶⁵ MVCC URL: https://en.wikipedia.org/wiki/Multiversion_concurrency_control (дата обращения 13.10.2021) [4]

Соответственно, когда мы выполняем операции вставки, обновления или удаления записи в Постгресе, происходят следующие изменения этих значений:

- **Insert** - добавляется новая запись с **xmin = txid_current()** и **xmax = 0**
- **Update** - в старой версии записи **xmax = txid_current()**, то есть делается **delete**, добавляется новая запись с **xmin = txid_current()** и **xmax = 0**, то есть делается **insert**
- **Delete** - в старой версии записи **xmax = txid_current()**

Следовательно, при добавлении записи у нас происходит две операции: старая запись помечается как удалённая (проставляется **xmax**), и происходит добавление новой записи. При удалении проставляется значение **xmax**, при этом физическое удаление данных не производится.

Для фиксации или отмены транзакции нам нужно рассмотреть ещё дополнительные атрибуты строки - **infomask** содержит ряд битов, определяющих свойства данной версии:

- **xmin_committed, xmin_aborted** - xmin подтверждён или отменён
- **xmax_committed, xmax_aborted** - xmax подтверждён или отменён

Важно отметить поле **ctid** - ссылка на следующую, более новую, версию той же строки. У самой новой, актуальной, версии строки **ctid** ссылается на саму эту версию. Номера **ctid** имеют вид (x,y): здесь x - номер страницы, y - порядковый номер указателя в массиве

Каждая транзакция может завершиться:

- **успешно** - после команды **commit**, для этого, в зависимости от типа операции, проставляются биты **xmin_committed, xmax_committed**
- **неуспешно** из-за, например, ошибки доступа к данным или вызове команды **rollback**.

При неуспешном завершении нам нужно откатить все изменения, выполненные этой транзакцией. При этом в случае с Постгресом нам практически не нужно ничего делать - будут установлены биты подсказки **xmin_aborted, xmax_aborted**. Сам номер **xmax** при этом остаётся в строке, но смотреть на него уже никто не будет.

Так что операции как коммита, так и роллбэка транзакции в Постгресе одинаково быстры.

Посмотрим на практике, как выглядит изнутри служебная информация.

Номер текущей транзакции можно узнать командой:

```
SELECT txid_current();
```

```
postgres=# select txid_current();
 txid_current
-----
          499
(1 row)

postgres=# \c app
You are now connected to database "app" as user "postgres".
app=# select txid_current();
 txid_current
-----
          500
(1 row)

app=#
```

При этом обратите внимание, что при смене БД у нас старая транзакция завершается отменой (**rollback**) и начинается новая транзакция.

Добавим в нашу ранее созданную в предыдущей главе таблицу **test** три новых значения:

```
INSERT INTO test VALUES (10), (20), (30);
```

Посмотрим статистику по “живым” (актуальным) и “мёртвым” (старым версиям в результате **update** или **delete**) туплам (записям):

```
SELECT relname, n_live_tup, n_dead_tup, trunc(100*n_dead_tup/  
(n_live_tup+1))::float "ratio%", FROM pg_stat_user_tables WHERE relname =  
'test';
```

```
app=# SELECT relname, n_live_tup, n_dead_tup, trunc(100*n_dead_tup/(n_live_tup+1))::float "ratio%" FROM pg_stat_user_tables WHERE relname = 'test';
 relname | n_live_tup | n_dead_tup | ratio%
-----+-----+-----+-----
 test   |          3 |           0 |      0
(1 row)
```

Теперь, если обновить одну строчку из нашего набора и посмотрим предыдущий запрос:

```
UPDATE test SET i = 40 WHERE i = 30;
```

```
app=# update test set i = 40 where i = 30;
UPDATE 1
app=# SELECT relname, n_live_tup, n_dead_tup, trunc(100*n_dead_tup/(n_live_tup+1))::float "ratio%" FROM pg_stat_user_tables WHERE relname = 'test';
 relname | n_live_tup | n_dead_tup | ratio%
-----+-----+-----+-----
 test   |          3 |           1 |     25
(1 row)
```

Видим, что теперь запрос выдаст другую информацию.

Итого, имеем три актуальных записи и одну мёртвую, содержащую старую информацию в строке со значением 30.

Напрямую можно посмотреть служебную информацию только по пяти полям и только у актуальных записей:

```
SELECT xmin,xmax,cmin,cmax,ctid FROM test;
```

```
app=# select xmin,xmax,cmin,cmax,ctid from test;
 xmin | xmax | cmin | cmax | ctid
-----+-----+-----+-----+-----
   501 |    0 |    0 |    0 | (0,1)
   501 |    0 |    0 |    0 | (0,2)
   502 |    0 |    0 |    0 | (0,4)
(3 rows)
```

Здесь видим из колонки ctid, что у нас не видно запись со страницы под номером 3.

Для более подробного изучения нужно установить расширение Постгреса pageinspect:

```
CREATE EXTENSION pageinspect;
```

Посмотрим, какие функции оно содержит (ограничимся первыми на скриншоте):

```
\dx+
```

```
app=# CREATE EXTENSION pageinspect;
CREATE EXTENSION
app=# \dx+
                Objects in extension "pageinspect"
                Object description
-----
function brin_metapage_info(bytea)
function brin_page_items(bytea,regclass)
function brin_page_type(bytea)
function brin_revmmap_data(bytea)
function bt_metap(text)
function bt_page_items(bytea)
function bt_page_items(text,integer)
function bt_page_stats(text,integer)
function fsm_page_contents(bytea)
function get_raw_page(text,integer)
function get_raw_page(text,text,integer)
function gin_leafpage_items(bytea)
function gin_metapage_info(bytea)
function gin_page_opaque_info(bytea)
function hash_bitmap_info(regclass,bigint)
function hash_metapage_info(bytea)
function hash_page_items(bytea)
function hash_page_stats(bytea)
function hash_page_type(bytea)
function heap_page_item_attrs(bytea,regclass)
function heap_page_item_attrs(bytea,regclass,boolean)
function heap_page_items(bytea)
function heap_tuple_infomask_flags(integer,integer)
function page_checksum(bytea,integer)
```

Давайте для примера воспользуемся `heap_page_items` и `get_raw_page`:

```
SELECT lp as tuple, t_xmin, t_xmax, t_field3 as t_cid, t_ctid FROM heap_page_items(get_raw_page('test',0));
```

```
app=# select lp as tuple, t_xmin, t_xmax, t_field3 as t_cid, t_ctid from heap_page_items(get_raw_page('test',0));
 tuple | t_xmin | t_xmax | t_cid | t_ctid
-----+-----+-----+-----+-----
      1 |    501 |      0 |      0 | (0,1)
      2 |    501 |      0 |      0 | (0,2)
      3 |    501 |    502 |      0 | (0,4)
      4 |    502 |      0 |      0 | (0,4)
(4 rows)
```

Здесь получили доступ к сырой версии таблицы и видим третью строчку. А в ней видим проставленный `t_xmax` и ссылку на новую версию строки `t_ctid`.

Также можно посмотреть всё содержимое служебных полей командой:

```
SELECT * FROM heap_page_items(get_raw_page('test',0)) \gx
```

```
-[ RECORD 1 ]-----
lp           | 1
lp_off      | 8160
lp_flags    | 1
lp_len      | 28
t_xmin      | 501
t_xmax      | 0
t_field3    | 0
t_ctid      | (0,1)
t_infomask2 | 1
t_infomask  | 2304
t_hoff      | 24
t_bits      |
t_oid       |
t_data      | \x0a000000
-[ RECORD 2 ]-----
lp           | 2
lp_off      | 8128
lp_flags    | 1
lp_len      | 28
t_xmin      | 501
t_xmax      | 0
t_field3    | 0
t_ctid      | (0,2)
t_infomask2 | 1
t_infomask  | 2304
t_hoff      | 24
t_bits      |
t_oid       |
t_data      | \x14000000
```

Более подробную информацию смотреть уже труднее. Воспользуемся запросом:

```
SELECT '(0,||lp||)' AS ctid,  
CASE lp_flags  
  WHEN 0 THEN 'unused'  
  WHEN 1 THEN 'normal'  
  WHEN 2 THEN 'redirect to '||lp_off  
  WHEN 3 THEN 'dead'  
END AS state,  
t_xmin as xmin,  
t_xmax as xmax,  
(t_infomask & 256) > 0 AS xmin_committed,  
(t_infomask & 512) > 0 AS xmin_aborted,  
(t_infomask & 1024) > 0 AS xmax_committed,  
(t_infomask & 2048) > 0 AS xmax_aborted,  
t_ctid  
FROM heap_page_items(get_raw_page('test',0)) lgx
```

```
-[ RECORD 2 ]-+-----  
ctid          | (0,2)  
state         | normal  
xmin          | 501  
xmax          | 0  
xmin_committed | t  
xmin_aborted  | f  
xmax_committed | f  
xmax_aborted  | t  
t_ctid        | (0,2)  
-[ RECORD 3 ]-+-----  
ctid          | (0,3)  
state         | normal  
xmin          | 501  
xmax          | 502  
xmin_committed | t  
xmin_aborted  | f  
xmax_committed | t  
xmax_aborted  | f  
t_ctid        | (0,4)
```

Видим, что возле удалённой третьей записи стоит флаг **xmax_committed**, а у второй актуальной записи, наоборот, стоит флаг **xmax_aborted**. Соответственно, чтобы удалять мёртвые записи нужен какой-то механизм. И он называется вакуум (VACUUM). О нём сейчас и поговорим.

Vacuum⁶⁶.

VACUUM высвобождает пространство, занимаемое мёртвыми строками. Соответственно, чем больше таких операций, тем больше места занимают таблицы и индексы. Таким образом, периодически необходимо выполнять VACUUM, особенно для часто изменяемых таблиц.

Оператор VACUUM специфичен именно для MVCC в Постгресе.

Без списка *таблиц и столбцов* команда VACUUM обрабатывает все таблицы и материализованные представления в текущей базе данных, на очистку которых текущий пользователь имеет право.

Команда VACUUM только делает пространство доступным для повторного использования. Эта форма команды может работать параллельно с обычными операциями чтения и записи строк таблицы, так она **не требует исключительной блокировки**.

Посмотрим на самые важные параметры:

VERBOSE - выводит на экран отчёт об очистке для каждой таблицы.

ANALYZE - выполняет очистку (VACUUM), а затем анализ (ANALYZE) всех указанных таблиц. Это удобная комбинация для регулярного обслуживания БД. Команда `analyze` и принцип её работы у нас будут рассмотрены в дальнейших главах.

FULL - выбирает режим полной очистки, который еще и дефрагментирует пространство - соответственно выполняется гораздо дольше и **запрашивает исключительную блокировку таблицы**. Так как он записывает новую копию таблицы и не освобождает старую до завершения операции, то требует дополнительное место на диске.

SKIP_LOCKED - указывает, что команда VACUUM пропускает все строки с конфликтующими блокировками.

Соответственно, наблюдать за процессом очистки можно или в командной строке при выполнении **VACUUM VERBOSE**, или через системное представление **`SELECT * FROM pg_stat_progress_vacuum;`**

Пример использования:

Очистка одной таблицы `test`, проведение её анализа для оптимизатора и печать подробного отчёта о действиях операции очистки:

`VACUUM (VERBOSE, ANALYZE) test;`

⁶⁶ Vacuum URL: <https://www.postgresql.org/docs/14/sql-vacuum.html> (дата обращения 13.10.2021) [5]

Посмотрим, что теперь осталось в таблице:

```
SELECT lp as tuple, t_xmin, t_xmax, t_field3 as t_cid, t_ctid FROM heap_page_items(get_raw_page('test',0));
```

```
app=# VACUUM (VERBOSE, ANALYZE) test;
INFO:  vacuuming "public.test"
INFO:  "test": found 0 removable, 3 nonremovable row versions in 1 out of 1 pages
DETAIL:  0 dead row versions cannot be removed yet, oldest xmin: 504
There were 0 unused item identifiers.
Skipped 0 pages due to buffer pins, 0 frozen pages.
0 pages are entirely empty.
CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s.
INFO:  analyzing "public.test"
INFO:  "test": scanned 1 of 1 pages, containing 3 live rows and 0 dead rows; 3 rows in sample, 3 estimated total rows
VACUUM
app=# select lp as tuple, t_xmin, t_xmax, t_field3 as t_cid, t_ctid from heap_page_items(get_raw_page('test',0));
 tuple | t_xmin | t_xmax | t_cid | t_ctid
-----+-----+-----+-----+-----
  1    |   501  |      0 |      0 | (0,1)
  2    |   501  |      0 |      0 | (0,2)
  3    |        |        |        |
  4    |   502  |      0 |      0 | (0,4)
(4 rows)
```

Видим, что строка под номером 3 стала пустой, теперь используя карту видимости (VM) Постгрес знает, что место свободно и запишет туда другую запись по мере необходимости.

Очистка - отличный механизм, вопрос в том, как часто её вызывать.

Если очищать изменяющуюся таблицу слишком редко, она вырастет в размерах больше, чем хотелось бы. Кроме того, для очередной очистки может потребоваться несколько проходов по индексам, если изменений накопилось слишком много.

Если очищать таблицу слишком часто, то вместо полезной работы сервер будет постоянно заниматься обслуживанием - тоже нехорошо.

Также важно понимать, что запуск обычной очистки по расписанию никак не решает проблему, потому что нагрузка может изменяться со временем. Если таблица стала обновляться активней, то и очищать её надо чаще.

Более подробно про процесс Vacuum можно почитать в статье⁶⁷ на Хабр⁶⁸.

Автоматическая очистка - как раз тот самый механизм, который позволяет запускать очистку в зависимости от активности изменений в таблицах.

⁶⁷ MVCC-6. Очистка URL: <https://habr.com/ru/company/postgrespro/blog/452320/> (дата обращения 23.09.2021) [6]

⁶⁸ Хабр URL: <https://habr.com> (дата обращения 23.09.2021) [7]

Autovacuum⁶⁹.

Автовакуум управляется соответствующим **фоновым процессом Autovacuum launcher**. Т.е. он не работает всё время, а инициирует вызов вакуума в зависимости от настроек различных порогов срабатывания. Общая рекомендация от меня и специалистов настаивать максимально агрессивно.

Посмотрим на важные настройки⁷⁰:

log_autovacuum_min_duration - время выполнения автоочистки в мс, при превышении которого информация об этом действии записывается в журнал. Default -1, что отключает журналирование действий автоочистки. При нулевом значении в журнале фиксируются все действия автоочистки.

autovacuum_max_workers - максимальное число процессов автоочистки (не считая процесса autovacuum_launcher), которые могут выполняться одновременно. Default 3.

autovacuum_naptime - минимальная задержка в секундах между двумя запусками автоочистки. Если это значение задаётся без единиц измерения, оно считается заданным в секундах. Default 1min.

Другие параметры влияют на то, как часто будет автовакуум заходить в таблицы для очистки. Про них можно почитать по ссылке выше.

Практически все эти параметры задаются только в postgresql.conf или в командной строке при запуске сервера. По разнице в настройке отдельных параметров будет целая тема чуть позже. Например, часть параметров можно мягко перезагрузить:

```
sudo pg_ctlcluster 14 main reload
```

⁶⁹ Autovacuum URL: <https://www.postgresql.org/docs/14/routine-vacuuming.html#AUTOVACUUM> (дата обращения 13.10.2021) [8]

⁷⁰ Параметры автовакуума URL: <https://www.postgresql.org/docs/14/runtime-config-autovacuum.html> (дата обращения 23.10.2021) [9]

Посмотрим на текущие настройки кластера:

```
SELECT name, setting, context, short_desc FROM pg_settings WHERE category LIKE '%Autovacuum%';
```

name	setting	context	short_desc
autovacuum	on	sighup	Starts the autovacuum subprocess.
autovacuum_analyze_scale_factor	0.1	sighup	Number of tuple inserts, updates, or deletes prior to analyze as a fraction of reltuples.
autovacuum_analyze_threshold	50	sighup	Minimum number of tuple inserts, updates, or deletes prior to analyze.
autovacuum_freeze_max_age	200000000	postmaster	Age at which to autovacuum a table to prevent transaction ID wraparound.
autovacuum_max_workers	3	postmaster	Sets the maximum number of simultaneously running autovacuum worker processes.
autovacuum_multixact_freeze_max_age	400000000	postmaster	Multixact age at which to autovacuum a table to prevent multixact wraparound.
autovacuum_naptime	60	sighup	Time to sleep between autovacuum runs.
autovacuum_vacuum_cost_delay	2	sighup	Vacuum cost delay in milliseconds, for autovacuum.
autovacuum_vacuum_cost_limit	-1	sighup	Vacuum cost amount available before napping, for autovacuum.
autovacuum_vacuum_insert_scale_factor	0.2	sighup	Number of tuple inserts prior to vacuum as a fraction of reltuples.
autovacuum_vacuum_insert_threshold	1000	sighup	Minimum number of tuple inserts prior to vacuum, or -1 to disable insert vacuums.
autovacuum_vacuum_scale_factor	0.2	sighup	Number of tuple updates or deletes prior to vacuum as a fraction of reltuples.
autovacuum_vacuum_threshold	50	sighup	Minimum number of tuple updates or deletes prior to vacuum.

(13 rows)

Здесь видим такие настройки, как **autovacuum_freeze_max_age** и **autovacuum_multixact_freeze_max_age**.

Это ещё один аспект работы вакуума и автовакуума - они “замораживают” старые записи. Для чего это нужно сейчас и посмотрим.

Более подробно как работает Autovacuum можно посмотреть в статье⁷¹ на habr.com.

⁷¹ MVCC в PostgreSQL-8. Заморозка URL: <https://habr.com/ru/company/postgrespro/blog/452762/> (дата обращения 23.09.2021) [10]

Заморозка⁷².

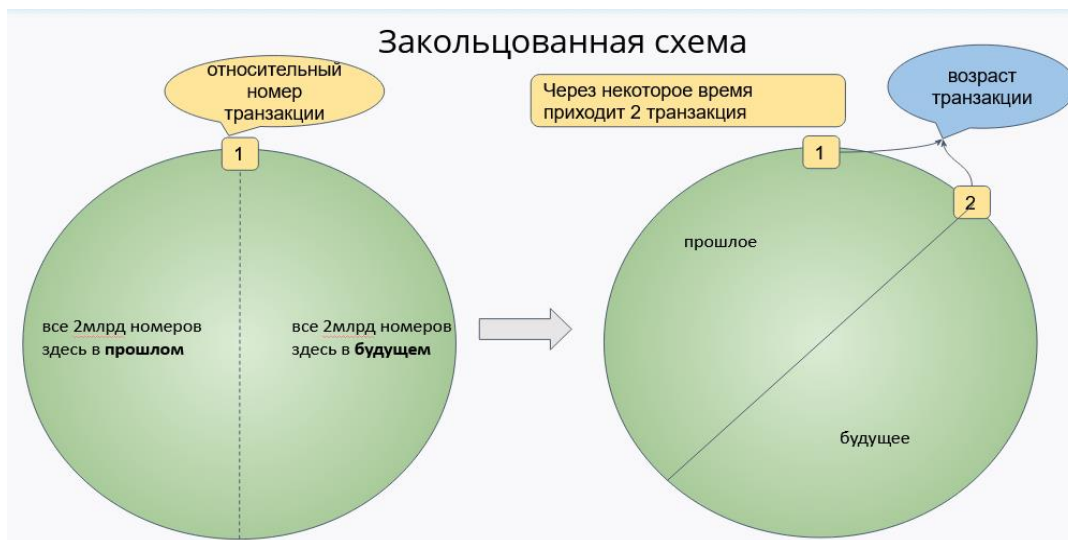
В Постгресе семантика транзакций зависит от возможности сравнения номеров идентификаторов транзакций (XID).

Версия строки, у которой XID добавившей её транзакции больше, чем XID текущей транзакции, относится «к будущему» и не должна быть видна в текущей транзакции.

Однако поскольку идентификаторы транзакций имеют ограниченный размер (32 бита), кластер, работающий долгое время (более 4 миллиардов транзакций) столкнётся с *зацикливанием идентификаторов транзакций*: счётчик XID дойдёт до максимального значения и прокрутится до нуля - внезапно транзакции, которые относились к прошлому, окажутся в будущем - это означает, что их результаты станут невидимыми. Для того, чтобы этого избежать, необходимо выполнять очистку для каждой таблицы в каждой базе данных как минимум единожды на два миллиарда транзакций.

Периодическое выполнение очистки решает эту проблему, потому что процедура **VACUUM помечает строки как замороженные**, указывая, что они были вставлены транзакцией, **зафиксированной максимально в прошлом**, так что эти данные с точки зрения MVCC определённо будут видны во всех текущих и будущих транзакциях.

Давайте для понимания схематично рассмотрим эту ситуацию (здесь использованы материалы статьи⁷³ на Хабр):

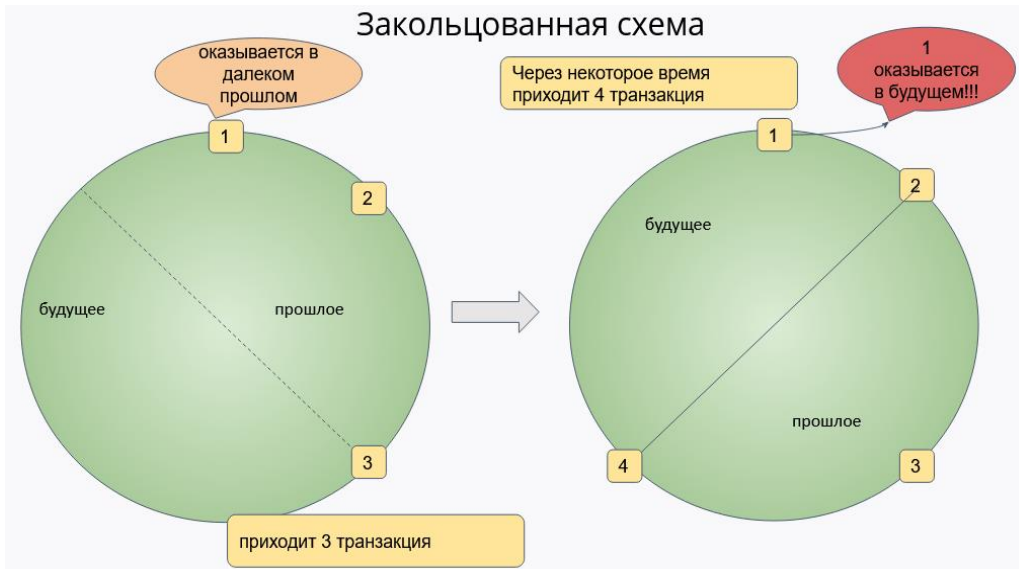


Итого, первая транзакция оказывается в прошлом.

⁷² Freeze URL: <https://www.postgresql.org/docs/14/routine-vacuuming.html#VACUUM-FOR-WRAPAROUND> (дата обращения 13.10.2021) [11]

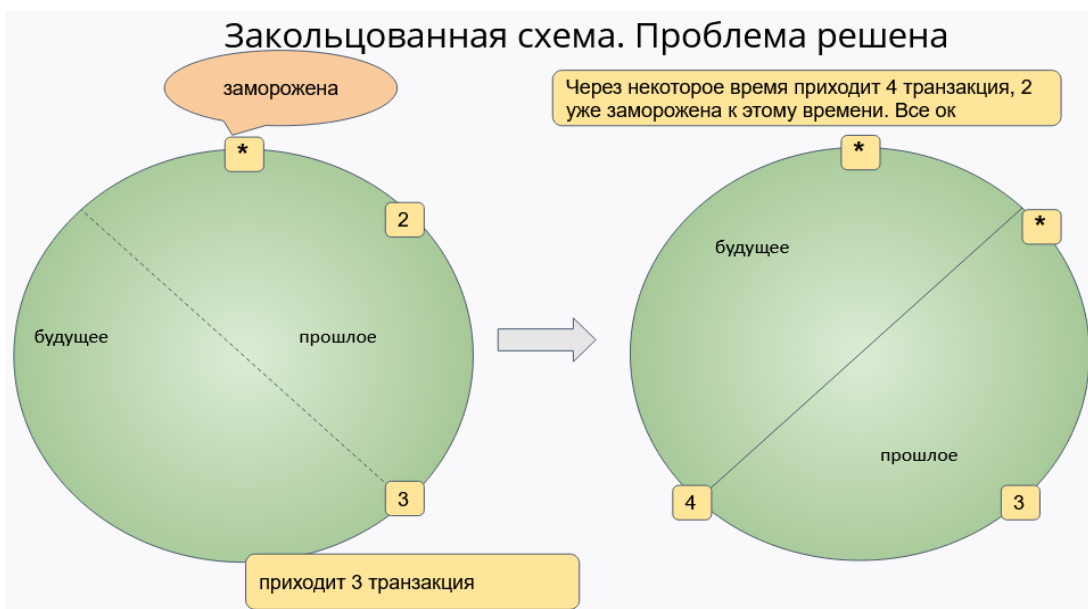
⁷³ MVCC в PostgreSQL-8. Заморозка URL: <https://habr.com/ru/company/postgrespro/blog/455590/> (дата обращения 13.10.2021) [12]

Затем у нас приходят 3 и 4 транзакции, и получается интересная ситуация:



Если транзакция в данном случае оказывается в будущем, то сервер аварийно завершит работу до устранения проблемы. Именно поэтому придуман механизм заморозки - при определённом пороге отставания от № текущей транзакции прошлые XID у записей замораживаются - т.е. у них проставляются биты `xmin_committed` и `xmin_aborted`.

Таким образом помечается, что эта запись считается очень старой и видной во всех снимках:



Теперь вернёмся к нашим параметрам вакуума и автовакуума (измеряются в количестве транзакций):

vacuum_freeze_min_age - определяет, насколько старым должен стать XID, чтобы строки с таким XID были заморожены. Уменьшение приводит к увеличению количества транзакций, которые могут выполняться, прежде чем потребуется очередная очистка таблицы, а увеличение его значения помогает избежать ненужной работы, если строки, которые могли бы быть заморожены в ближайшее время, будут изменены ещё раз. Всё сугубо индивидуально на каждом проекте.

Обычная команда VACUUM не будет всегда замораживать все версии строк, имеющиеся в таблице, так как пропускает страницы, в которых нет мёртвых версий строк, даже если в них могут быть версии строк со старыми XID.

Периодически VACUUM будет также производить *агрессивную очистку*, пропуская только те страницы, которые не содержат ни мёртвых строк, ни незамороженных значений XID. Когда VACUUM будет делать это, зависит от параметра:

vacuum_freeze_table_age - полностью видимые, но не полностью замороженные, страницы будут сканироваться, если число транзакций, прошедших со времени последнего такого сканирования, оказывается больше, чем **vacuum_freeze_table_age** минус **vacuum_freeze_min_age**. Если **vacuum_freeze_table_age** равно 0, VACUUM будет применять эту более агрессивную стратегию при каждом сканировании.

Максимальное время, в течение которого таблица может обходиться без очистки, составляет два миллиарда транзакций минус значение **vacuum_freeze_min_age** с момента последней агрессивной очистки.

Чтобы гарантировать, что это не произойдёт, для любой таблицы, которая может содержать значения XID старше, чем возраст, указанный в конфигурационном параметре **autovacuum_freeze_max_age**, вызывается автоочистка (Это случится, даже если автоочистка отключена).

Соответственно **autovacuum_freeze_max_age** - автоочистка будет вызываться приблизительно через каждые **autovacuum_freeze_max_age** минус **vacuum_freeze_min_age** транзакций. Работает только для таблиц, НЕ очищаемых регулярно.

Давайте на практике посмотрим на возраст самых старых данных в таблицах:

```
SELECT c.oid::regclass as table_name,  
       greatest(age(c.relFrozenxid),age(t.relFrozenxid)) as age  
FROM pg_class c  
LEFT JOIN pg_class t ON c.reltoastrelid = t.oid  
WHERE c.relkind IN ('r', 'm');
```

table_name	age
test2	10
test	11
pg_statistic	27
pg_type	27
pg_foreign_table	27
pg_authid	27
pg_statistic_ext_data	27
pg_largeobject	27
pg_user_mapping	27
pg_subscription	27
pg_attribute	27
pg_proc	27
pg_class	27
pg_attrdef	27
pg_constraint	27
pg_inherits	27
pg_index	27
pg_operator	27
pg_opfamily	27
pg_opclass	27
pg_am	27
pg_amop	27
pg_amproc	27

Кластер практически пустой и возраст самых старых транзакций довольно молодой.

В этой главе мы с вами рассмотрели проблематику ACID и параллельных транзакций. Рассмотрели MVCC, как это реализовано в Постгресе и как за собой удалять мёртвые строки.

По умолчанию параметры автовакуума настроены достаточно хорошо. Менять их рекомендую при наличии каких-либо проблем, либо чётко понимая, зачем вы это делаете. Одна из самых распространённых рекомендаций от других специалистов - настраивать максимально агрессивно - имеет тоже место быть.

Таким образом мы закончили блок физического представления данных в СУБД. Переходим к уровням изоляции транзакций, логическому устройству Постгреса и внутренним механизмам реализации отказоустойчивости и реализации блокировок.

В конце главы, интересный кейс: как у Амазона была проблема с работой автовакуума и как они её решили⁷⁴.

Ссылка *bonus* в соответствующем файле на гитхабе <https://github.com/aeuge/Postgres14book/blob/main/chapters/CHAPTER04.md>

Все команды, рассмотренные в этой главе, вы можете найти на гитхабе в файле <https://github.com/aeuge/Postgres14book/blob/main/scripts/04/mvcc.txt>

⁷⁴ Тюнинг автовакуума в Amazon URL: <https://aws.amazon.com/ru/blogs/database/a-case-study-of-tuning-autovacuum-in-amazon-rds-for-postgresql/> (дата обращения 13.10.2021) [bonus]