GeekBrains

PostgreSQL

Начало работы с PostgreSQL, базовые инструменты, сравнение с другими СУБД

Оглавление

История

В начале 70-х годов в рамках работы над проектом IBM System R IBM опубликовали статью, в которой впервые употребили термин «реляционная база данных».

Michael Stonebraker из University of California, Berkeley, вдохновившись этим whitepaper, начал работу над проектом **Ingres**. Ingres расшифровывается как INteractive Graphics REtrieval System. Такое название появилось из-за желания иметь не один большой компьютер на весь университет, а интерактивную систему.

Michael Stonebreaker попробовал делать проприетарную версию Ingres, но у него не получилось, и он продолжил работать над Ingres как над академическим проектом, попутно переименовав проект в Postgres (1986 год). В итоге многие идеи и большая часть кода Postgres дали жизнь проекту IBM DB2.

Сервер баз данных логически может делиться на множество баз данных. Именно отсюда и идёт название «Система управления базами данных» (СУБД). Michael Stonebreaker был первопроходцем на пути создания современных СУБД. Он заложил расширяемость как основную идею создания СУБД. Благодаря этому подходу современный Postgres стал самой мощной и многофункциональной СУБД в мире, а Michael Stonebreaker в 2014 году получил Turing Award.

Postgres95 стала первой версией, где поддерживался SQL синтаксис. На этой версии завершилась эпоха развития Postgres как академического проекта. С этого момента разработка ушла в сообщество и таковой остаётся до сих пор. Новую версию назвали **PostgreSQL** (1997 год), и с неё можно вести историю современного Postgres.

Общепринятый синоним Postgres можно использовать взаимозаменяемо с PostgreSQL.

CAP теорема

Что лучше? PostgreSQL, или MySQL, или MongoDB, или Cassandra, или Clickhouse, или ScyllaDB, или Tarantool, или Redis, или DynamoDB, или Google BigQuery, или <подставьте название любого продукта, который что-то хранит>?

В таком виде на этот вопрос ответа нет. Но есть разные способы сравнивать СУБД. И если мы научимся их сравнивать, то сможем выбирать наиболее подходящую для нашего случая. **CAP теорема** — это один из способов (*не лучший и не единственный*) говорить про различия в СУБД.

Когда объясняют, что такое CAP теорема, часто вспоминают выражение «*Быстро, дешево, качественно: выберите два*». В распределённых системах тоже есть три свойства, из которых придется выбрать только два:

- **Consistency** (*linearizability*) — свойство указывает, что если запрос на изменение данных завершился успешно, то можно ожидать, что запрос на получение данных вернёт последние записанные данные. В этом случае система обязана возвращать актуальные данные вне зависимости от того, к какому экземпляру СУБД подключен клиент;
- **Availability** — свойство указывает, что запрос к «живому» экземпляру СУБД гарантированно вернёт корректный ответ;
- **Partition tolerance** — свойство указывает, что система обязана работать корректно даже в условиях разрыва канала связи между экземплярами СУБД.

Но если мы говорим о **распределённых** системах, то можно считать, что одно из свойств мы уже выбрали, а именно **Partition tolerance**. Поэтому CAP теорема в случае сравнения СУБД звучит примерно так: В случае **partition** (*т.е., например, разрыва сети между экземплярами базы данных*) какие гарантии мы хотим иметь: **availability** — жизнеспособность нашей системы до починки аварии — или **consistency** *(linearizability)* — гарантии соответствия данных нашим ожиданиям.

Если у вас возник вопрос, почему мы говорим именно о распределённых системах, ответ — других в современном мире не осталось.

Принципы ACID

Большинство реляционных СУБД являются транзакционными, т.е. оперируют транзакциями. Транзакция — способ группировки нескольких операций чтения или записи в одну логическую единицу, в которой все операции выполняются как единое целое. Либо выполняются все, либо ни одна. Транзакции не могут быть выполнены частично.

Принципы ACID часто используют для категоризации СУБД. ACID расшифровывается как:

- **Atomicity** — если транзакция состояла из нескольких операций, то все операции одной транзакции должны быть успешно выполнены (или не выполнены, но тоже все вместе);
- **Consistency** (не путать с C в CAP Theorem) — если данные были записаны, то они однозначно удовлетворяют установленным ограничениям и связям;
- **Isolation** — транзакция не имеет возможности повлиять на результат работы другой транзакции;
- **Durability** — если СУБД ответила пользователю, что данные записаны, то данные не должны быть потеряны.

Если СУБД удовлетворяет всем принципам ACID, то мы говорим, что это ACID-compliant (ACID-совместимая) СУБД. Когда мы сравниваем СУБД, мы можем задать вопрос, является ли выбранная база ACID-compliant.

При этом существует множество СУБД, которые отказываются от поддержки некоторых принципов ACID в угоду производительности.

Место PostgreSQL в мире СУБД

Postgres — это, в классическом понимании, транзакционная, **ACID-compliant** (ACID-совместимая), реляционная база данных.

Всё управление разработкой Postgres находится в руках независимой Core Team, состоящей из разработчиков из разных компаний. Postgres — один из очень немногих продуктов, за которым не стоит компания-техгигант. Каждое из изменений, вносимых в основную кодовую базу, проходит большое количество обсуждений. Отсутствие давления со стороны бизнеса позволяет разработчикам Postgres принимать взвешенные решения, качественно тестировать каждую фичу и не пускать сырой код в продакшн.

The PostgreSQL Licence позволяет использовать, распространять и изменять Postgres где и как угодно. И это один из самых важных факторов, почему бизнес предпочитает использовать Postgres.

В сравнении с другими реляционными базами данных Postgres наиболее полно реализует стандарт SQL, при этом даёт дополнительные, не включенные в стандарт, возможности для удобной работы с данными.

Позже мы поговорим про реализацию работы с транзакциями в Postgres, но важно знать, что эффективная реализация Multiversion concurrency control (MCC or MVCC) в Postgres позволяет ему быть ACID-compliant, при этом обеспечивая высокую производительность.

Сегодня почти невозможно провести чёткую границу между базами данных.

Запуск PostgreSQL в Docker

Для работы с СУБД нам необходимо иметь работающий экземпляр сервера. Есть несколько вариантов начать работу с Postgres:

- установить локально — процесс установки зависит от используемой операционной системы;
- запустить виртуальную машину (например, используя VirtualBox);
- использовать облачного провайдера — в этом случае СУБД поднимается на удалённом сервере;
- использовать Docker — это контейнерная виртуализация, которая позволяет унифицировать процесс запуска на разных операционных системах.

Для быстрого старта мы будем запускать сервер Postgres, используя Docker.

Контейнер Docker по умолчанию не сохраняет данные при перезапуске. Чтобы данные не терялись, мы замонтируем локальную директорию в Docker-контейнер. Для этого запускаем контейнер с флагом `-v $(pwd)/mntdata:/var/lib/postgresql/data`. При этом ENV-переменная PGDATA явно указывает Postgres, что внутри контейнера файлы должны храниться в директории `/var/lib/postgresql/data`. Теперь все файлы экземпляра СУБД будут сохранены в директории `$(pwd)/mntdata` и не потеряются при перезапуске контейнера.

```
docker run \
    --rm -it \
    -p 5432:5432 \
    --name postgres \
    -e POSTGRES_PASSWORD=password \
    -e PGDATA=/var/lib/postgresql/data \
    -v $(pwd)/mntdata:/var/lib/postgresql/data \
    postgres:13.1
```

Должен запуститься контейнер с именем postgres с работающим PostgreSQL версии 13.1.

После запуска мы можем зайти в контейнер:

```
docker exec -it postgres bash
```

Расположение данных на файловой системе

Расположение файлов на файловой системе зависит от значения переменной окружения `$PGDATA`. В нашем контейнере `$PGDATA` указывает на директорию `/var/lib/postgresql/data/`. Переходим туда и смотрим, что есть в директории, используя команду `ls -al`:

```
root@hostname:/# echo $PGDATA
/var/lib/postgresql/data
root@hostname:/# cd $PGDATA
root@hostname:/var/lib/postgresql/data# ls -al
total 132
drwx------ 19 postgres postgres  4096 Nov 29 09:55 .
drwxr-xr-x  1 postgres postgres  4096 Nov 18 08:35 ..
drwx------  5 postgres postgres  4096 Nov 29 09:46 base
drwx------  2 postgres postgres  4096 Nov 29 09:46 global
drwx------  2 postgres postgres  4096 Nov 29 09:46 pg_commit_ts
drwx------  2 postgres postgres  4096 Nov 29 09:46 pg_dynshmem
-rw-------  1 postgres postgres  4782 Nov 29 09:46 pg_hba.conf
-rw-------  1 postgres postgres  1636 Nov 29 09:46 pg_ident.conf
drwx------  4 postgres postgres  4096 Nov 29 09:51 pg_logical
drwx------  4 postgres postgres  4096 Nov 29 09:46 pg_multixact
drwx------  2 postgres postgres  4096 Nov 29 09:46 pg_notify
drwx------  2 postgres postgres  4096 Nov 29 09:46 pg_replslot
drwx------  2 postgres postgres  4096 Nov 29 09:46 pg_serial
drwx------  2 postgres postgres  4096 Nov 29 09:46 pg_snapshots
drwx------  2 postgres postgres  4096 Nov 29 09:46 pg_stat
drwx------  2 postgres postgres  4096 Nov 29 10:19 pg_stat_tmp
drwx------  2 postgres postgres  4096 Nov 29 09:46 pg_subtrans
drwx------  2 postgres postgres  4096 Nov 29 09:46 pg_tblspc
drwx------  2 postgres postgres  4096 Nov 29 09:46 pg_twophase
-rw-------  1 postgres postgres     3 Nov 29 09:46 PG_VERSION
drwx------  3 postgres postgres  4096 Nov 29 09:46 pg_wal
drwx------  2 postgres postgres  4096 Nov 29 09:46 pg_xact
-rw-------  1 postgres postgres   107 Nov 29 09:55 postgresql.auto.conf
-rw-------  1 postgres postgres 28043 Nov 29 09:46 postgresql.conf
-rw-------  1 postgres postgres    36 Nov 29 09:46 postmaster.opts
-rw-------  1 postgres postgres    94 Nov 29 09:46 postmaster.pid
```

Назначение каждого из используемых файлов подробно освещено в официальной документации в разделе Database File Layout. Обратите внимание на файлы конфигураций postgresql.conf и pg_hba.conf.

Файлы конфигурации

Из **postgresql.conf** сервер Postgres при запуске читает параметры конфигурации. Документация покрывает все возможные опции конфигурации.

Не всегда стандартные настройки Postgres подходят под используемые сервера. Если важна производительность, то имеет смысл настроить параметры в postgresql.conf под ваше железо. В статьях Performance Tuning PostgreSQL и How to tune PostgreSQL for memory можно узнать, на какие параметры следует смотреть в postgresql.conf. Анализ эффективности и рекомендации по оптимизации SQL-запросов мы разберем в последующих уроках.

pg_hba.conf позволяет задавать настройки аутентификации для доступа к базе. Про формат файла подробно написано в официальной документации. Посмотрим содержимое pg_hba.conf в нашем контейнере:

```
$ cat pg_hba.conf
# TYPE  DATABASE        USER            ADDRESS                 METHOD

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

host    all             all             all                     md5
```

В первой строчке указано `local all all trust` — это значит, что если мы присоединяемся к базе данных с той же машины, где установлен сервер, то аутентифицировать не нужно, можно доверять.

При этом в последней строчке `host all all all md5` указано, что если соединение идет извне, то нужно требовать пароль.

Такими настройками обеспечивается удобство администрирования сервера, при этом гарантируется безопасность соединения.

Далее рассмотрим команды для подключения к БД. После подключения смотреть параметры конфигурации можно в таблице-представлении `pg_settings`:

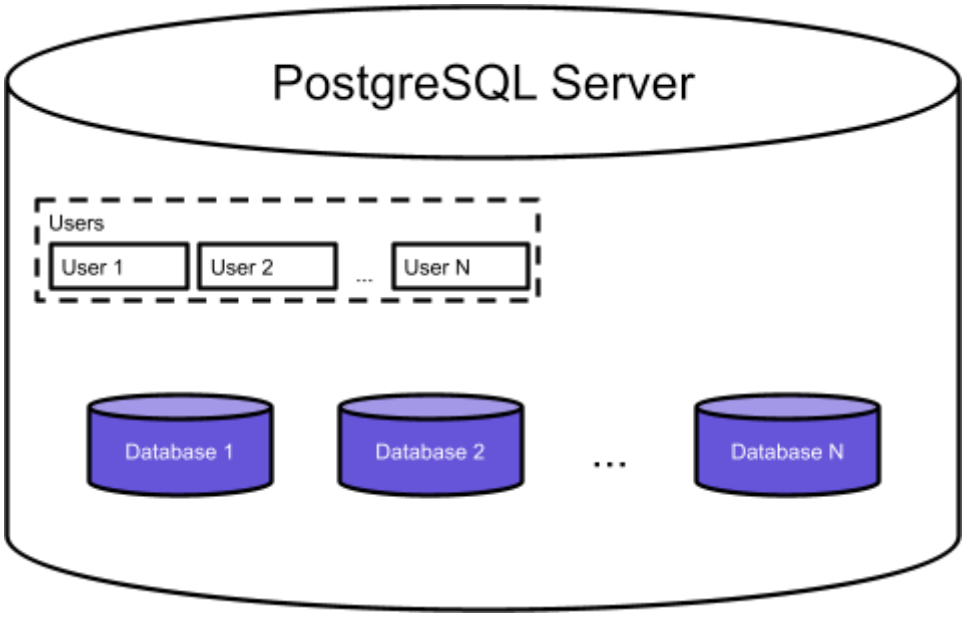
```
postgres=# select * from pg_settings where name = 'max_connections' \gx
-[ RECORD 1 ]---+----------------------------------------------------
name            | max_connections
setting         | 100
unit            |
category        | Connections and Authentication / Connection Settings
short_desc      | Sets the maximum number of concurrent connections.
extra_desc      |
context         | postmaster
vartype         | integer
source          | configuration file
min_val         | 1
max_val         | 262143
enumvals        |
boot_val        | 100
reset_val       | 100
sourcefile      | /var/lib/postgresql/data/postgresql.conf
sourceline      | 64
pending_restart | f
```

(\gx в конце SQL-запроса позволяет построчно вывести значения столбцов)

Часть конфигураций можно вносить без редактирования postgresql.conf, для этого используется команда alter system.

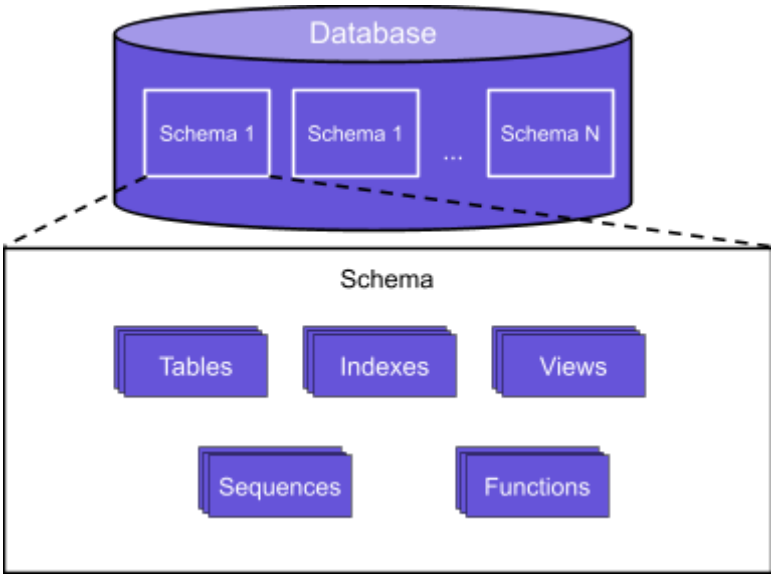
Логическая структура данных

Сервер PostgreSQL является системой управления базами данных (СУБД) и логически делится на множество баз данных. В рамках СУБД мы можем создавать пользователей, которым будем давать права доступа на одну или несколько баз данных.



Каждая из баз данных делится на несколько схем (schemas). По умолчанию используется схема `public`. Схемы могут использоваться для:

- разделения доступа к данным для разных пользователей;
- разделения данных на разные логические группы;
- предоставления возможности использовать одинаковые имена таблиц в одной общей базе данных (в рамках одной схемы каждая таблица обязана иметь уникальное имя).


Database
Schema 1
Schema 1
...
Schema N
Schema
Tables
Indexes
Views
Sequences
Functions

Типы данных

Postgres поддерживает большое множество различных типов данных как из стандарта SQL, так и специфичные именно для Postgres.

Помимо стандартных `integer, bigint, varchar, date, double` и т.д. стоит обратить внимание на:

- `json, jsonb` — Postgres умеет эффективно хранить, индексировать и удобно работать со структурами в формате JSON;
- `tsvector, tsquery` — Postgres поддерживает полнотекстовый поиск;
- `interval` — возможность хранить временные интервалы;
- `money` — тип данных для работы с деньгами;
- `point, polygon, path, line, circle, box` — хранение геометрических фигур;
- `inet, cidr, macaddr` — сетевые адреса.

Как и в языках со строгой типизацией, наличие типов данных позволяет нам допускать меньше ошибок и контролировать данные, которые мы отдаём на хранение базе.

Пример создания таблички с автоматически генерируемым идентификатором и обязательным (`not null`) полем типа `integer`:

```
create table tablename (
      id bigint generated always as identity,
      field1 integer not null
)
```

Postgres поддерживает приведение типов (types casting). Это позволяет конвертировать строку в число или в дату и выполнять операции, доступные над этими типами:

```
mydb=> select '2020-10-23'::date;
    date
------------
 2020-10-23
(1 row)

mydb=> select '23'::integer;
 int4
------
   23
(1 row)
```

В моменты, когда вам нужно выбрать тип данных для столбца, обращайтесь к официальной документации.

PSQL

Для подключения к серверу Postgres используется утилита-клиент `psql`.

Подключение к Postgres с использованием psql:

```
# psql -d database -U username -h hostname -p port
```

Соединяемся с сервером, используя команду `psql -U postgres`. Флагом `-U` мы указываем имя пользователя, которого используем для соединения с сервером. Пользователь с именем `postgres` доступен по умолчанию.

psql позволяет использовать мета-команды для работы с базой без необходимости явно делать SQL-запросы. Список всех доступных мета-команд доступен в официальной документации. Ниже представлены наиболее часто используемые:

- `\c [database name]` — выбор базы данных для использования;
- `\l` — вывести список всех баз данных;
- `\d` — вывести список таблиц (`tables`), представлений (`views`) и последовательностей (`sequences`);
    - `\dt` — вывести список таблиц (`tables`);
    - `\dv` — вывести список представлений (`views`);
    - `\dm` — вывести список материализованных представлений (`materialized views`);
    - `\di` — вывести список индексов (`indexes`);
    - `\dn` — вывести список схем (`schemas`);
    - `\dT` — вывести список доступных типов данных (`data types`);
    - `...`
- `\x` — включить/выключить расширенный режим просмотра. Полезно при выводе информации из таблиц с большим количеством столбцов;
- `\set` — вывести список всех локальных переменных `psql`;
    - `\set [Name] [Value]` — задать значение локальной переменной;
- `\unset [Name]` — удалить локальную переменную `psql`;
- `\cd` — сменить директорию, в рамках которой происходит выполнение `psql`;
- `\! [Command]` — выполнить shell команду
    - например, `\! ls` or `\! pwd`
- `\copy` — скопировать данные в файл (полезно при выгрузке данных из БД в файл);
- `\i [filename]` — выполнить команды из файла;
- `\o [file]` — переключить вывод результатов запросов на вывод в файл вместо вывода на экран;
- `\q` — закрыть `psql`.

Создание пользователя и базы данных

Чтобы начать работать с базой данных, нужно её создать. Создаём базу данных с именем `mydb`

```
postgres=# create database mydb;
CREATE DATABASE
```

Посмотрим список доступных баз данных, для этого вводим мета-команду `\l`

```
postgres=# \l
                                 List of databases
   Name    |  Owner   | Encoding |  Collate   |   Ctype    |   Access privileges
-----------+----------+----------+------------+------------+-----------------------
 mydb      | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 postgres  | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 template0 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
           |          |          |            |            | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
           |          |          |            |            | postgres=CTc/postgres
(4 rows)
```

Первой строкой показана созданная нами база данных mydb.

Рекомендуется создавать отдельного пользователя для каждой базы данных, чтобы явно ограничивать область доступа. Создаём нового пользователя и даём ему полные права на управление базой:

```
postgres=# create user myuser password 'secret';
CREATE ROLE
postgres=# grant all privileges on database mydb to myuser;
GRANT
```

Закрываем текущее соединение, используя мета-команду `\q`, и переподключаемся с новым пользователем к новой базе:

```
postgres=# \q
root@hostname:/var/lib/postgresql/data# psql -U myuser -d mydb
psql (13.1 (Debian 13.1-1.pgdg100+1))
Type "help" for help.

mydb=>
```

Мета-команда `\conninfo` покажет информацию о текущем подключении к базе данных:

```
mydb=> \conninfo
You are connected to database "mydb" as user "myuser" via socket in
"/var/run/postgresql" at port "5432".
```

Чтобы посмотреть список всех доступных мета-команд `psql`, используйте `\?`

Заключение

На этом уроке мы разобрались, что такое CAP теорема и как базы данных можно сравнивать, используя свойства Consistency, Availability и Partition tolerance.

PostgreSQL является реляционной СУБД. Для категоризации таких СУБД часто используют принципы ACID. PostgreSQL удовлетворяет всем принципам ACID, можно сказать, что он является ACID-compliant СУБД.

Расположение файлов на файловой системе зависит от значения переменной окружения $PGDATA. При создании контейнера с экземпляром Postgres мы явно указывали переменную окружения $PGDATA и монтировали директорию с локальной файловой системы, чтобы не потерять данные при перезапуске контейнера.

Каждый экземпляр Postgres может оперировать несколькими логическими базами данных. Каждая логическая база делится на схемы, а схемы делятся на таблицы.

Рекомендуется для работы с каждой логической базой данных создавать отдельного пользователя. Это позволяет разделять права доступа и уменьшить потенциальные риски безопасности.

Postgres поддерживает большинство типов данных из стандарта SQL, а также расширяет функциональность дополнительными типами данных, такими как tsvector, jsonb и т.п.

Практические задания

1. Развернуть сервер PostgreSQL в Docker.
2. Создать пользователя и базу данных.
3. В базе из пункта 2 создать таблицу: не менее трёх столбцов различных типов. SQL-запрос на создание таблицы добавить в текстовый файл `class1_hometask.txt`.
4. В таблицу из пункта 3 вставить не менее трёх строк. SQL-запрос на вставку добавить в текстовый файл `class1_hometask.txt.`
5. Используя мета-команды psql, вывести список всех сущностей в базе данных из пункта 2. Полученный список сущностей добавить в текстовый файл `class1_hometask.txt`.
6. Придумать проект, над которым вы будете работать в последующих уроках. Примером может быть любой сайт/приложение, которым вы пользуетесь (YouTube, Medium, AirBnB, Booking, Twitter, Instagram, Facebook, LinkedIn). Это может быть соц. сеть, блог, книга рецептов, база данных авиаперелетов, мессенджер, система бронирования бань и саун и т.п.
7. Кратко (не более 10 предложений) описать суть проекта и основной use-case в файле `schema.sql` (описывать как sql комментарий в начале файла).
8. Разработать структуру базы данных, которая будет фундаментом для выбранного проекта (не менее трёх таблиц, не более 10 таблиц). В структуре базы данных желательно иметь логические связи между сущностями (не менее одной связи). Команды на создание таблиц описать в файле `schema.sql`.

Глоссарий

CAP theorem — теорема, применяемая в распределённых системах, которая указывает, что система одновременно имеет не более двух свойств из следующих: Consistency, Availability, Partition tolerance.

Consistency (linearizability) — свойство указывает, что если запрос на изменение данных завершился успешно, то можно ожидать, что запрос на получение данных вернёт последние записанные данные. В этом случае система обязана возвращать актуальные данные вне зависимости от того, к какому экземпляру СУБД подключен клиент.

Availability — свойство указывает, что запрос к «живому» экземпляру СУБД гарантированно вернёт корректный ответ.

Partition tolerance — свойство указывает, что система обязана работать корректно даже в условиях разрыва канала связи между экземплярами СУБД.

Транзакция — способ группировки нескольких операций чтения или записи в одну логическую единицу, в которой все операции выполняются как единое целое. Либо выполняются все, либо ни одна. Транзакции не могут быть выполнены частично.

Принципы ACID — принципы, часто используемые для категоризации СУБД.

Atomicity — если транзакция состояла из нескольких операций, то все операции одной транзакции должны быть успешно выполнены (или не выполнены, но тоже все вместе).

Consistency (не путать с C в CAP Theorem) — если данные были записаны, то они однозначно удовлетворяют установленным ограничениям и связям.

Isolation — транзакция не имеет возможности повлиять на результат работы другой транзакции.

Durability — если СУБД ответила пользователю, что данные записаны, то данные не должны быть потеряны.

ACID-compliant СУБД — СУБД, которая удовлетворяет всем принципам ACID.

Список литературы

1. A Brief History of PostgreSQL: https://www.postgresql.org/docs/current/history.html
2. Looking Back at Postgres: https://arxiv.org/pdf/1901.01973.pdf
3. Database File Layout: https://www.postgresql.org/docs/current/storage-file-layout.html
4. PGConfRu2019 Олег Бартунов — «Профессиональный постгрес»: https://youtu.be/0vRN4KZY75o
5. All you didn't know about the CAP theorem: https://www.itechart.com/blog/all-you-didnt-know-about-cap-theorem/
6. What is the CAP Theorem? MongoDB vs Cassandra vs RDBMS, where do they stand in the CAP theorem? https://medium.com/@bikas.katwal10/mongodb-vs-cassandra-vs-rdbms-where-do-they-stand-in-the-cap-theorem-1bae779a7a15
7. Всё, что вы не знали о CAP теореме: https://habr.com/ru/post/328792/
8. The CAP theorem and physical limitations of replication: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781783550609/1/ch01lvl1sec08/the-cap-theorem-and-physical-limitations-of-replication
9. Please stop calling databases CP or AP: https://martin.kleppmann.com/2015/05/11/please-stop-calling-databases-cp-or-ap.html
10. You Can’t Sacrifice Partition Tolerance: https://codahale.com/you-cant-sacrifice-partition-tolerance/
11. Which parameter changes do require a restart of my PostgreSQL instance? https://blog.dbi-services.com/which-parameter-changes-do-require-a-restart-of-my-postgresql-instance/
12. psql: https://www.postgresql.org/docs/current/app-psql.html
13. Meta commands in PSQL: https://dataschool.com/learn-sql/meta-commands-in-psql/
14. Descriptive SQL Style Guide: https://github.com/pgulutzan/descriptive-sql-style-guide/blob/master/style.md