

Командная строка. Резервные копии. Транзакции

Николай Хацанов
Full-stack Developer



Николай Хащанов

Full-stack Developer



План занятия

- ① Командная строка
- ② Интерактивный режим
- ③ Резервное копирование и восстановление
- ④ Create user / role
- ⑤ DCL
- ⑥ TCL
- ⑦ Транзакции. Блокировки.

Командная строка



1

Командная строка

Командная строка используется для ввода и выполнения команд, запуска приложений и взаимодействия с операционной системой

**Как вы думаете,
в каких случаях и почему нужно
использовать командную строку, а не
интерфейс?**

Командная строка / Интерфейс

- ① Интерфейсы могут быть уязвимы с точки зрения безопасности
- ② Размер импортируемого файла больше, чем поддерживает интерфейс
- ③ Универсальный интерфейс для взаимодействия с различными приложениями
- ④ Отключение проверки работы ограничений
- ⑤ Богатые возможности автоматизации через скриптовые ЯП
- ⑥ Возможность работать с удаленными серверами

Запуск командной строки

В Windows:

- Пуск -> В поиске набираете «cmd» -> выбираете «cmd.exe»
- Сочетание клавиш Win + R -> набираете «cmd» -> Enter

В Mac OS:

- Launchpad -> Другие -> Терминал
- Spotlight -> набираете «Терминал» -> выбираете «Терминал»

Команды командной строки

Для работы с приложениями `psql`, `pg_dump`, `pg_restore`:

- `-c` (или `--command`) — запуск команды SQL без выхода в интерактивный режим
- `-f file.sql` — выполнение команд из файла `file.sql`
- `-l` (или `--list`) — выводит список доступных баз данных
- `-U` (или `--username`) — указание имени пользователя
- `-W` (или `--password`) — указание пароля пользователя
- `-d db_name` — подключение к базе данных `db_name`
- `-h` — имя хоста
- `-p` — указание порта
- `-s` — пошаговый режим, то есть, нужно будет подтверждать все команды
- `-S` — однострочный режим, то есть, переход на новую строку будет выполнять запрос (избавляет от ; в конце конструкции SQL)
- `-V` — версия PostgreSQL без входа в интерактивный режим

Командная строка. Запуск приложения

Указать полный путь до файла, либо перейти в папку с исполняемым файлом:

- `c:\program files\postgresql\10\bin\psql`

или

- `cd c:\program files\postgresql\10\bin\psql`

Командная строка. Подключение к БД

Указать данные для подключения:

- `psql -h localhost -p 5432 -U postgres -d postgres`

Подключиться к базе данных postgres и схеме hr и получить данные из таблицы persons

- `psql -h 84.201.163.203 -p 19001 -U netology -d postgres
-c "set search_path to hr;" -c "select * from persons;"`

Интерактивный режим



2

Как работать с PostgreSQL

- ① Через командную строку
- ② Через интерактивный режим, представленный программой psql.

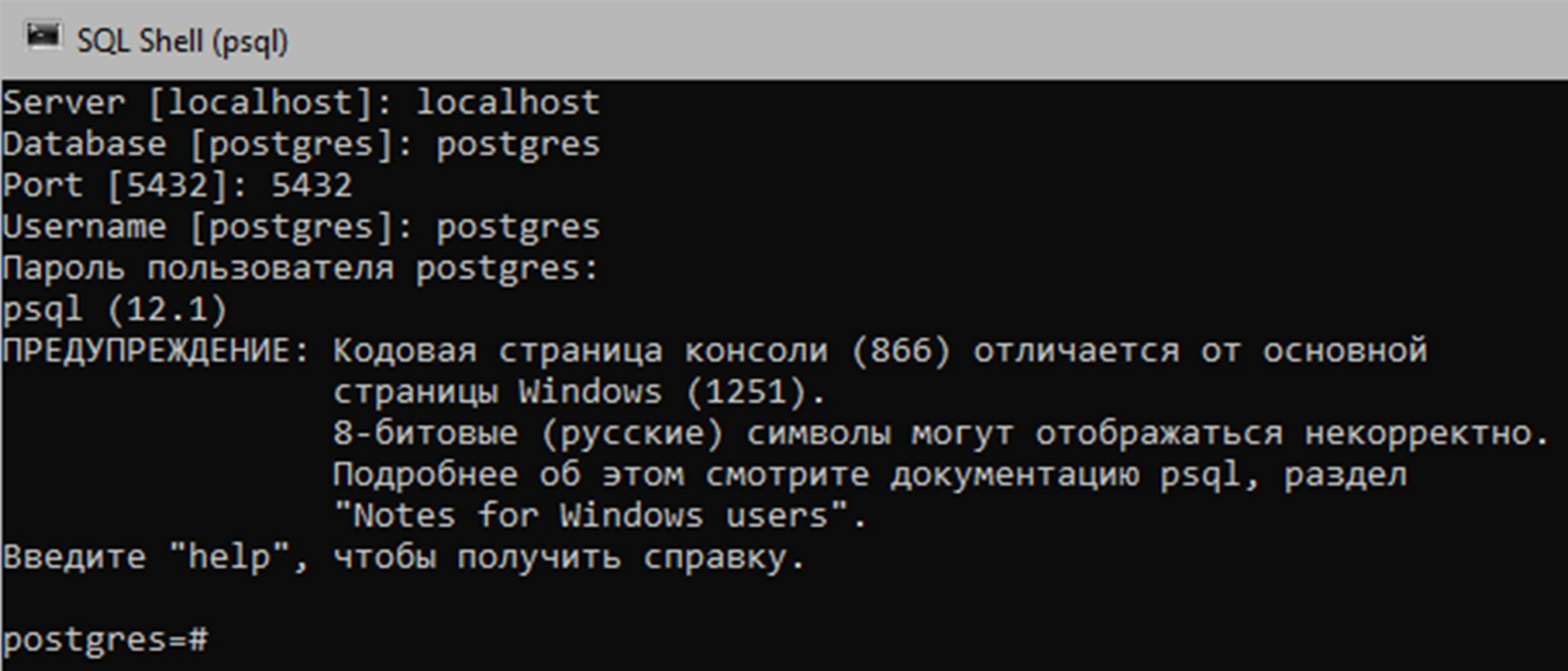
Интерактивный режим универсален в плане используемой операционной системы, если в командной строке в зависимости от операционной системы есть различия в командах и синтаксисе, то интерактивный режим везде одинаковый.

Для работы потребуется установленный локально клиент PostgreSQL

<https://www.enterprisedb.com/downloads/postgres-postgresql-downloads>

Интерактивный режим. Подключение

После запуска приложения `psql`, если изначально не указывали параметры подключения, то можно указать их в явном виде:



```
SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: postgres
Port [5432]: 5432
Username [postgres]: postgres
Пароль пользователя postgres:
psql (12.1)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
                  страницы Windows (1251).
                  8-битовые (русские) символы могут отображаться некорректно.
                  Подробнее об этом смотрите документацию psql, раздел
                  "Notes for Windows users".
Введите "help", чтобы получить справку.

postgres=#
```

При вводе пароля символы не отображаются.

Интерактивный режим. Кодировка в Windows

`psql` — это «консольное приложение».

В Windows консольные окна используют кодировку символов отличную от той, что используется для остальной системы, нужно проявить особую осторожность при использовании 8-битных символов.

Если `psql` обнаружит проблемную кодовую страницу консоли, он предупредит вас при запуске.

Изменение кодовой страницы консоли

- ① Установите консольный шрифт в **Lucida Console**, потому что растровый шрифт не работает с кодовой страницей **ANSI**.
- ② В верхнем левом углу окна кликнуть на иконку, во всплывающем меню выбрать «Свойства», вкладка «Шрифт» и выбрать нужный.
 - Выполнить команду: `!\ chcp 1251` -- проверить кодировку, если без изменений, выполнить следующий пункт.
 - Выполнить команду: `set client_encoding='win1251';`

Интерактивный режим. Команды psql

- `\connect db_name` - подключиться к базе с именем db_name
- `\du` - список пользователей
- `\dp (или \z)` - список таблиц, представлений, последовательностей, прав доступа к ним
- `\di` - индексы
- `\ds` - последовательности
- `\dt` - список таблиц
- `\dt+` - список всех таблиц с описанием
- `\dv` - представления
- `\dS` - системные таблицы
- `\d+` - описание таблицы
- `\o` - пересылка результатов запроса в файл
- `\l` - список баз данных

Интерактивный режим. Команды psql

- `\i` - читать входящие данные из файла
- `\d "table_name"` - описание таблицы
- `\i` - запуск команды из внешнего файла, например `\i /my/directory/my.sql`
- `\pset` - команда настройки параметров форматирования
- `\echo` - выводит сообщение
- `\set` - устанавливает значение переменной среды. Без параметров выводит список текущих переменных
- `\unset` - удаляет
- `\?` - справочник psql
- `\help` - справочник SQL
- `\q` (или `Ctrl+D`) - выход с программы

<https://postgrespro.ru/docs/postgresql/12/app-psql>

Интерактивный режим

Давайте создадим базу данных: `create database netology;`

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

Выведем список баз данных: `\l`

```
postgres=# \l
```

Имя	Владелец	Кодировка	Список баз данных LC_COLLATE	LC_CTYPE	Права доступа
netology	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	
postgres	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	
template0	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	=c/postgres + postgres=CTc/postgres
template1	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	=c/postgres + postgres=CTc/postgres

(4 строки)

Выберем базу данных: `\c netology;`

```
postgres=# \c netology;  
Вы подключены к базе данных "netology" как пользователь "postgres".
```

Интерактивный режим

Создадим таблицу: `create table students(id serial primary key, first_name varchar(50) not null, last_name varchar(50) not null);`

```
netology=# create table students(id serial primary key,first_name varchar(50) not null, last_name varchar(50) not null);
CREATE TABLE
```

Посмотрим все таблицы в базе: `\d`

```
netology=# \d
          Список отношений
  Схема |      Имя      |      Тип      | Владелец
-----+-----+-----+-----
 public | students      | таблица       | postgres
 public | students_id_seq | последовательность | postgres
(2 строки)
```

Посмотрим содержимое определенной таблицы: `\d students`

```
netology=# \d students
          Таблица "public.students"
  Столбец |      Тип      | Правило сортировки | Допустимость NULL | По умолчанию
-----+-----+-----+-----+-----
   id    | integer       |                     | not null           | nextval('students_id_seq'::regclass)
 first_name | character varying(50) |                     | not null           |
 last_name | character varying(50) |                     | not null           |
Индексы:
    "students_pkey" PRIMARY KEY, btree (id)
```


Интерактивный режим

- Синтаксис всех запросов в интерактивном режиме такой же, как и в любом интерфейсе.
- Запросы между собой разделяются в обязательном порядке символом ;

Резервное копирование и восстановление



3

Резервное копирование и восстановление

Как и все, что содержит важные данные, базы данных PostgreSQL следует регулярно сохранять в резервной копии. Хотя эта процедура по существу проста, важно четко понимать лежащие в ее основе приемы и положения.

Существует три фундаментально разных подхода к резервному копированию данных в PostgreSQL:

- Выгрузка в SQL
- Копирование на уровне файлов
- Непрерывное архивирование

Каждый из них имеет свои сильные и слабые стороны.

Выгрузка в SQL

Данный подход заключается в генерации текстового файла с командами **SQL**, которые при выполнении на сервере пересоздадут базу данных в том же самом состоянии, в котором она была на момент выгрузки.

PostgreSQL предоставляет для этой цели вспомогательные программы **pg_dump**, **pg_dumpall** и **pg_restore**.

С помощью данных программ можно как создавать резервные копии, так и восстанавливать данные. Запуск происходит из командной строки.

С перечнем параметров для каждой утилиты можно ознакомиться в Документации:

- <https://postgrespro.ru/docs/postgresql/12/app-pgdump>
- <https://postgrespro.ru/docs/postgresql/12/app-pg-dumpall>
- <https://postgrespro.ru/docs/postgresql/12/app-pgrestore>

pg_dump

Синтаксис:

- `pg_dump` <параметры> <имя базы> > <файл для сохранения копии>

Чтобы создать бэкап схемы public базы postgres:

- `pg_dump -d postgres -n public > "c:\backups\public.sql"`

Выгрузка базы данных в специальном бинарном формате для работы с `pg_restore`:

- `pg_dump -Fc postgres > "c:\backups\db.backup"`

pg_dump

Выгрузка базы данных в архив:

- `pg_dump -Ft postgres > "c:\backups\db.tar"`

Создание бэкап со структурой базы postgres без данных:

- `pg_dump -s -d postgres > "c:\backups\db_structure.sql"`

Восстановим данные из ранее созданного бэкапа в другую базу данных:

- `psql -d newbase < "c:\backups\public.sql"`

pg_dumpall

Утилита `pg_dumpall` предназначена для записи (выгрузки) всех баз данных кластера PostgreSQL в один файл в формате скрипта.

Этот файл содержит команды SQL, так что передав его на вход `psql`, можно восстановить все базы данных.

Для формирования этого файла вызывается `pg_dump` для каждой базы данных в кластере.

`pg_dumpall` также выгружает глобальные объекты, общие для всех баз данных, то есть роли и табличные пространства.

pg_dumpall

Так как утилита `pg_dumpall` читает таблицы из всех баз данных, для получения полного содержимого баз запускать ее, как правило, нужно от имени суперпользователя.

Также права суперпользователя требуются при последующем выполнении сохраненного скрипта, чтобы он смог добавить роли и создать базы данных.

Чтобы создать бэкап кластера:

- `pg_dumpall > "c:\backups\all_bases.sql"`

pg_restore

Утилита `pg_restore` предназначена для восстановления базы данных PostgreSQL из архива, созданного командой `pg_dump` в любом из не текстовых форматов. Она выполняет команды, необходимые для восстановления того состояния базы данных, в котором база была сохранена.

Утилита `pg_restore` может работать в двух режимах. Если указывается имя базы данных, `pg_restore` подключается к этой базе данных и восстанавливает содержимое архива непосредственно в неё. В противном случае создается SQL-скрипт с командами, необходимыми для пересоздания базы данных, который затем выводится в файл или в стандартное устройство вывода. Сформированный скрипт будет в точности соответствовать выводу `pg_dump` в простом текстовом формате.

Создадим новую базу данных и восстановим в нее ранее созданный бэкап:

```
createdb -T template0 newbase
```

```
pg_restore -d newbase "c:\backups\db.backup"
```

Резервное копирование на уровне файлов

Следующий подход резервного копирования является непосредственное копирование файлов, в которых PostgreSQL хранит содержимое базы данных.

Как правило, папка с базой данных - это папка **Data** в папке с установленным PostgreSQL.

После переноса папки и инициализации базы необходимо запустить приложение **initdb**.

Как пример:

initdb -D "c:\program files\postgresql\10\data" или

initdb -D library\postgresql\10\data

Резервное копирование на уровне файлов

Однако существуют два ограничения:

- Чтобы полученная резервная копия была рабочей, сервер баз данных должен быть остановлен. Такие полумеры, как запрещение всех подключений к серверу, работать не будут.
- Если есть необходимость сделать копию только части данных, то скопировать некоторые файлы или папки нельзя. Это не будет работать, потому что информацию, содержащуюся в этих файлах, нельзя использовать без файлов журналов транзакций, `pg_xact/*`, которые содержат состояние всех транзакций. Без этих данных файлы таблиц непригодны к использованию. Соответственно нельзя восстановить только одну таблицу и соответствующие данные `pg_xact`, потому что в результате нерабочими станут все другие таблицы в кластере баз данных.

Резервное копирование на уровне файлов

Также нужно помнить, что правильно настроенный сервер позволяет делать снимки томов жестких дисков без остановки базы данных и в дальнейшем можно работать с этими снимками.

Непрерывное архивирование и восстановление

`pg_basebackup` - предназначен для создания резервных копий работающего кластера баз данных PostgreSQL. Процедура создания копии не влияет на работу других клиентов. Полученные копии могут использоваться для обеих стратегий восстановления — на заданный момент в прошлом и в качестве отправной точки для ведомого сервера при реализации трансляции файлов или потоковой репликации.

`pg_basebackup` создаёт бинарную копию файлов кластера, контролируя режим создания копии автоматически. Резервные копии всегда создаются для кластера целиком и невозможно создать копию для какой-либо сущности базы отдельно.

Непрерывное архивирование и восстановление

Так как утилита `pg_basebackup` использует протокол репликации, то для работы нужно убедиться, что выполнены следующие условия:

- Раскомментированы строки в `pg_hba.conf`

```
# Allow replication connections from localhost, by a user with the
# replication privilege.
host      replication      all             127.0.0.1/32         md5
host      replication      all             ::1/128              md5
```

- Раскомментированы строки в `postgresql.conf`:

```
max_replication_slots = 10
max_wal_senders = 10
wal_level = replica
```

После того, как строки были раскомментированы, нужно перезагрузить процесс `PostgreSQL`

Непрерывное архивирование и восстановление

Можно выполнить два запроса, чтобы убедиться, что все настроено верно:

- `SELECT` name, setting
`FROM` pg_settings
`WHERE` name `IN` ('wal_level','max_wal_senders','max_replication_slots');
- `SELECT` type, database, user_name, address, auth_method
`FROM` pg_hba_file_rules
`WHERE` database = '{replication}';

Создаем резервную копию:

- `pg_basebackup -D c:\backups\ -P`

CREATE USER / ROLE



4

CREATE USER / ROLE

- **CREATE USER** — создать роль в базе данных, по умолчанию подразумевается параметр **LOGIN**
- **CREATE ROLE** — создать роль в базе данных, по умолчанию подразумевается параметр **NOLOGIN**

* **LOGIN / NOLOGIN**

*Эти предложения определяют, разрешается ли новой роли вход на сервер; то есть, может ли эта роль стать начальным авторизованным именем при подключении клиента. Можно считать, что роль с атрибутом **LOGIN** соответствует пользователю. Роли без этого атрибута бывают полезны для управления доступом в базе данных, но это не пользователи в обычном понимании.*

<https://postgrespro.ru/docs/postgresql/12/sql-createrole>

CREATE USER / ROLE

Создадим нового пользователя **reviser** с указанием пароля, без прав суперпользователя без прав создавать базы данных, без прав создавать новых пользователей и укажем, что пароль будет действовать до конца 2030 года:

- **CREATE ROLE** reviser **WITH LOGIN PASSWORD** '12345'
NOSUPERUSER NOCREATEDB NOCREATEROLE VALID UNTIL '31.12.2030';

Параметры **NOSUPERUSER**, **NOCREATEDB** и **NOCREATEROLE** можно не указывать в явном виде, так как они подразумеваются по умолчанию.

ALTER USER / ROLE

Поменяем пароль для пользователя reviser:

- ALTER ROLE reviser WITH PASSWORD '54321';

Изменим время действия пароля:

- ALTER ROLE reviser VALID UNTIL '31.12.2040';

Если бы захотели дать пользователю права на создание ролей и баз данных:

- ALTER ROLE reviser CREATEROLE CREATEDB;

DROP USER / ROLE

Для удаления пользователя или роли используется следующий запрос:

- `DROP ROLE IF EXISTS` reviser;

Если на эту роль есть ссылки в какой-либо базе данных в кластере, возникнет ошибка и роль не будет удалена. Прежде чем удалять роль, необходимо удалить все принадлежащие ей объекты или сменить их владельца, а также лишить ее данных ей прав для других объектов. Для этой цели можно применить команды `REASSIGN OWNED` и `DROP OWNED`.

- `REASSIGN OWNED BY` reviser `TO` new_user;
- `DROP OWNED BY` reviser;

DCL

5

DCL

Data Control Language (DCL) – группа операторов определения доступа к данным. Эти операторы нужны для управления разрешениями доступа к данным и выполнения операций над объектами базы данных. Права назначаются на пользователя или на роли.

В данную группу входят следующие операторы:

- **GRANT** – предоставляет пользователю или группе разрешения на определенные операции с объектами;
- **REVOKE** – отзывает выданные разрешения;

GRANT

Дать все права пользователю к базе данных с возможностью назначать права:

- `GRANT ALL PRIVILEGES ON DATABASE postgres TO reviser WITH GRANT OPTION;`

Дать пользователю право использовать обертку postgres_fdw:

- `GRANT USAGE ON FOREIGN DATA WRAPPER postgres_fdw TO reviser;`

Дать пользователю право просматривать объекты в схеме public:

- `GRANT USAGE ON SCHEMA public TO reviser;`

Дать пользователю право использовать операторы `SELECT` и `DELETE` ко всем таблицам в схеме public:

- `GRANT SELECT, DELETE ON ALL TABLES IN SCHEMA public TO reviser;`

<https://postgrespro.ru/docs/postgresql/12/sql-grant>

REVOKE

Забрать у пользователя право использовать оператор **DELETE** ко всем таблицам в схеме public:

- **REVOKE DELETE ON ALL TABLES IN SCHEMA** public **FROM** reviser;

Забрать у пользователя право просматривать объекты в схеме public:

- **REVOKE USAGE ON SCHEMA** public **FROM** reviser;

Забрать у пользователя право использовать обертку postgres_fdw:

- **REVOKE USAGE ON FOREIGN DATA WRAPPER** postgres_fdw **FROM** reviser;

Забрать все права у пользователя к базе данных и каскадно удалить те права, которые мог выдать пользователь:

- **REVOKE ALL PRIVILEGES ON DATABASE** postgres **FROM** reviser **CASCADE**;

<https://postgrespro.ru/docs/postgresql/12/sql-revoke>

TCL

6

TCL

Transaction Control Language (TCL) – группа операторов для управления транзакциями.

Эта группа состоит из следующих операторов:

- **BEGIN** – служит для определения начала транзакции;
- **COMMIT** – применяет транзакцию;
- **ROLLBACK TO** – откатывает все изменения, сделанные в контексте текущей транзакции;
- **SAVEPOINT** – устанавливает промежуточную точку сохранения внутри транзакции.

TCL

Пример транзакции:

- Вносим новую запись в страны:
- Ставим точку сохранения:
- Вносим новую запись в города:
- Допустили ошибку с городом:
- Вносим корректную запись в города:
- Завершаем транзакцию:

```
BEGIN;
```

```
INSERT INTO country (country) VALUES ('Россия');
```

```
SAVEPOINT my_savepoint;
```

```
INSERT INTO city (city, country_id) VALUES ('Варна', (SELECT  
CURRVAL('country_country_id_seq')));
```

```
ROLLBACK TO my_savepoint;
```

```
INSERT INTO city (city, country_id) VALUES ('Москва', (SELECT  
CURRVAL('country_country_id_seq')));
```

```
COMMIT;
```

Транзакции. Блокировки.

7

Транзакция

Транзакция - это последовательность операций, выполняемых в логическом порядке пользователем, либо программой, которая работает с базой данных.

Вспомним требования **ACID**:

- **Атомарность** Транзакция должна выполняться как единая операция доступа к базе данных и может быть выполнена полностью либо не выполнена совсем.
- **Согласованность** Свойство согласованности гарантирует выполнение ограничений целостности базы данных после окончания обработки транзакции.
- **Изолированность** Транзакции должны выполняться независимо друг от друга, и доступ к данным, изменяемым с помощью одной транзакции, для других транзакций должен быть запрещен, пока изменения не будут завершены.
- **Долговечность** Если транзакция выполнена успешно, то произведенные ею изменения в данных не должны быть потеряны ни при каких обстоятельствах.

MVCC

PostgreSQL поддерживает целостность данных, реализуя модель MVCC (Multiversion Concurrency Control, Многоверсионное управление конкурентным доступом).

Это означает, что каждый SQL-оператор видит снимок данных (версию базы данных) на определенный момент времени, вне зависимости от текущего состояния данных.

Это защищает операторы от несогласованности данных, возможной, если другие конкурирующие транзакции внесут изменения в те же строки данных, и обеспечивает тем самым изоляцию транзакций для каждого сеанса баз данных.

MVCC, отходя от методик блокирования, принятых в традиционных СУБД, снижает уровень конфликтов блокировок и таким образом обеспечивает более высокую производительность в многопользовательской среде.

MVCC

Основное преимущество использования модели MVCC по сравнению с блокированием заключается в том, что блокировки MVCC, полученные для чтения данных, не конфликтуют с блокировками, полученными для записи, и поэтому чтение никогда не мешает записи, а запись чтению.

PostgreSQL гарантирует это даже для самого строгого уровня изоляции транзакций, используя инновационный уровень изоляции SSI (Serializable Snapshot Isolation, Сериализуемая изоляция снимков).

MVCC. Как это работает.

В таблице с платежами есть запись:

	123 payment_id ↑↓	123 amount ↑↓
1	15 000	2,99

Так же знаем, что сумма всех платежей равна:

	123 sum ↑↓
1	67 426,51

Хотим изменить запись:

- `UPDATE payment SET amount = 4.99 WHERE payment_id = 15000;` --транзакция 1
данные записываются на жесткий диск еще одной строкой, но статус “в процессе”

Параллельно была запущена транзакция на получение суммы платежей:

- `SELECT SUM(amount) FROM payment` --транзакция 2

Так как транзакция 1 не была завершена, то транзакция 2 читает первую версию строки

и получаем ожидаемый результат:

	123 sum ↑↓
1	67 426,51

MVCC. Как это работает.

Другим сотрудником был также запущен запрос на получение всех платежей

- `SELECT SUM(amount) FROM payment` --транзакция 3

Представим, что пользователь отказался доплачивать и транзакция 1 была отменена, но вторая версия строки все еще на диске со статусом “отменена”.

Транзакция 3 из двух строк выберет первую, так как статус “завершена”

и получим такой же результат:

123 sum ↕	
1	67 426,51

Но, если бы транзакция была успешно завершена, то транзакция 3 получила бы данные из второй версии строки, у которой был бы статус “завершена”:

123 sum ↕	
1	67 428,51

MVSS. VACUUM.

Удалением ненужных снимков данных и соответственно строк, относящихся к этим снимкам занимается процесс **VACUUM**.

Этот процесс работает параллельно с другими процессами и ничего не блокирует.

При этом сами файлы не уменьшаются, в них появляется свободное пространство, которое потом заполняется новыми данными.

MVCC. VACUUM.

Есть процесс **VACUUM FULL**, который полностью перестраивает файлы данных, делая их компактными. Данный процесс блокирует таблицы и на запись, и на чтение.

Обычно очисткой занимается процесс **AUTOVACUUM**. Это фоновый процесс, который периодически очищает таблицы. Таких процессов два вида:

- **AUTOVACUUM launcher** – фоновый процесс, который реагирует на активность изменения данных и запускает рабочие процессы по очистке таблиц;
- **AUTOVACUUM worker** – запускаются по необходимости и выполняют очистку.

Уровни изоляций

READ COMMITTED - уровень изоляции транзакции, выбираемый в PostgreSQL по умолчанию.

В транзакции, работающей на этом уровне, запрос **SELECT** видит только те данные, которые были зафиксированы до начала запроса. Он никогда не увидит незафиксированных данных или изменений, внесенных в процессе выполнения запроса параллельными транзакциями. По сути запрос **SELECT** видит снимок базы данных в момент начала выполнения запроса.

Однако **SELECT** видит результаты изменений, внесенных ранее **в этой же транзакции**, даже если они еще не зафиксированы.

Команды **INSERT**, **UPDATE** и **DELETE** ведут себя подобно **SELECT** при поиске целевых строк - они найдут только те целевые строки, которые были зафиксированы на момент начала команды.

Уровни изоляций

В режиме **REPEATABLE READ** видны только те данные, которые были зафиксированы до начала транзакции, но не видны незафиксированные данные и изменения, произведенные другими транзакциями в процессе выполнения данной транзакции.

Однако запрос будет видеть эффекты предыдущих изменений в своей транзакции, несмотря на то, что они не зафиксированы.

Этот уровень отличается от **READ COMMITTED** тем, что запрос в транзакции данного уровня видит снимок данных на момент начала первого оператора в транзакции (не считая команд управления транзакциями), а не начала текущего оператора.

Таким образом, последовательные команды **SELECT** в одной транзакции видят одни и те же данные - они не видят изменений, внесенных и зафиксированных другими транзакциями после начала их текущей транзакции.

Уровни изоляций

Уровень **SERIALIZABLE** обеспечивает самую строгую изоляцию транзакций. На этом уровне моделируется последовательное выполнение всех зафиксированных транзакций, как если бы транзакции выполнялись одна за другой, последовательно, а не параллельно.

Этот режим изоляции работает так же, как и **REPEATABLE READ**, при наличии сбоев необходимо повторять транзакции, только **SERIALIZABLE** дополнительно отслеживает условия, при которых результат параллельно выполняемых сериализуемых транзакций может не согласовываться с результатом этих же транзакций, выполняемых по очереди.

Это отслеживание не приносит дополнительных препятствий для выполнения, кроме тех, что присущи режиму **REPEATABLE READ**, но тем не менее создаёт некоторую добавочную нагрузку, а при выявлении исключительных условий регистрируется аномалия сериализации и происходит сбой сериализации.

Изоляции транзакций

Стандарт описывает следующие особые условия, недопустимые для различных уровней изоляции:

- «Грязное» чтение - транзакция читает данные, записанные параллельной незавершенной транзакцией.
- Неповторяемое чтение - транзакция повторно читает те же данные, что и раньше, и обнаруживает, что они были изменены другой транзакцией (которая завершилась после первого чтения).
- Фантомное чтение - транзакция повторно выполняет запрос, возвращающий набор строк для некоторого условия, и обнаруживает, что набор строк, удовлетворяющих условию, изменился из-за транзакции, завершившейся за это время.
- Аномалия сериализации - результат успешной фиксации группы транзакций оказывается несогласованным при всевозможных вариантах исполнения этих транзакций по очереди.

Изоляции транзакций

«Грязное» чтение:

Счет А: 0 рублей.

Счет Б: 10 000 рублей.

Транзакция 1: перевод со счета Б на счет А 5 000 рублей - в процессе.

Транзакция 2: покупка в магазине со счета А на 3 000 рублей.

При грязном чтении транзакция 2 видит пополнение счета А на 5 000 рублей и соответственно средств на покупку товара достаточно.

Со счета А списывается 3000 рублей, на счету остается 2 000 рублей и транзакция фиксируется.

При этом транзакция 1 была отменена и деньги фактически переведены не были, таким образом получаем:

Счет А: 2 000 рублей

Счет Б: 10 000 рублей

Магазин: -3 000 рублей

Изоляции транзакций

Неповторяемое чтение:

Счет А: 0 рублей.

Счет Б: 10 000 рублей.

Транзакция 1: перевод со счета Б на счет А 5 000 рублей - завершена.

Транзакция 2: покупка в магазине со счета А на 3 000 рублей - в процессе.

Транзакция 3: перевод со счета А на счет Б 4 000 рублей - завершена.

Транзакция 2 видит, что на счете А есть 5 000 и можно совершить платеж, так же видит начало выполнения транзакции 3 и заново начинает читать данные со счета и видит, что строка изменилась и теперь на счету А 1 000 рублей и отменяет транзакцию из-за недостатка денег.

Если бы не было повторного чтения и транзакции выполнялись последовательно, то сперва выполнялась бы транзакция 2, а транзакция 3 была бы отменена.

Изоляции транзакций

Фантомное чтение:

Транзакция 1: Получение данных по платежам пользователя А - в процессе.

Предполагаем получить 20 записей.

Транзакция 2: Пользователь А совершил еще одну покупку - завершена.

Происходит повторное чтение таблицы по платежам, видим новую запись и выводим в результат 21 запись.

В случае удаления каких-либо записей, в транзакции 1 получили бы меньше записей.

Изоляции транзакций

Аномалия сериализации:

Счет А: 5 000 рублей.

Счет Б: 10 000 рублей.

Транзакция 1:

- Получение данных о сумме на счету А -- 5 000
- Перевод со счета А на счет Б 5 000 рублей

Транзакция 2:

- Получение данных о сумме на счету Б -- 10 000
- Перевод со счета Б на счет А 10 000 рублей

В случае последовательного выполнения транзакций будет ошибка о сумме на счету.

Изоляции транзакций

Уровни изоляции:

Уровень изоляции	«Грязное» чтение	Неповторяемое чтение	Фантомное чтение	Аномалия сериализации
Read uncommitted (Чтение незафиксированных данных)	Допускается, но не в PG	Возможно	Возможно	Возможно
Read committed (Чтение зафиксированных данных)	Невозможно	Возможно	Возможно	Возможно
Repeatable read (Повторяемое чтение)	Невозможно	Невозможно	Допускается, но не в PG	Возможно
Serializable (Сериализуемость)	Невозможно	Невозможно	Невозможно	Невозможно

Виды изоляций подробно расписаны в Документации:

<https://postgrespro.ru/docs/postgresql/12/transaction-iso>

Явные блокировки

Для управления параллельным доступом к данным в таблицах PostgreSQL предоставляет несколько режимов явных блокировок. Эти режимы могут применяться для блокировки данных со стороны приложения в ситуациях, когда MVCC не даёт желаемый результат.

Выделяют следующие виды блокировок:

- блокировки на уровне таблицы;
- блокировки на уровне строк;
- блокировки на уровне страниц.

Более подробно по ссылке в Документации: <https://postgrespro.ru/docs/postgresql/12/explicit-locking>

Полезные представления и функции

- `pg_locks` - даёт доступ к информации о блокировках, удерживаемых активными процессами на сервере баз данных
- `pg_stat_activity` - для каждого серверного процесса будет присутствовать по одной строке с информацией, относящейся к текущей деятельности этого процесса
- `pg_stat_database` - содержит по одной строке со статистикой уровня базы данных для каждой базы кластера, и ещё одну для общих объектов
- `pg_stat_user_functions` - для каждой отслеживаемой функции будет содержать по одной строке со статистикой по выполнениям этой функции
- `pg_stat_get_backend_idset ()` → setof integer - выдаёт идентификаторы активных в настоящий момент серверных процессов (от 1 до числа активных процессов).
- `pg_stat_get_backend_activity (integer)` → text - выдаёт текст последнего запроса этого серверного процесса.

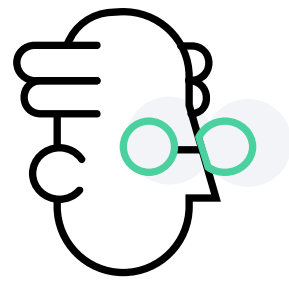
Итоги занятия



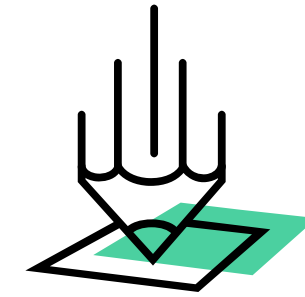
Основные тезисы



Разобрали отличие
командной строки от
интерактивного режима



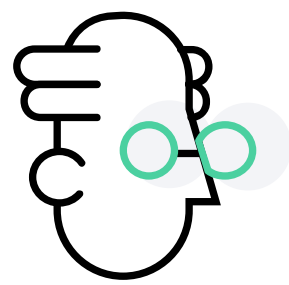
Изучили TCL и DCL запросы



Узнали о транзакциях и
блокировках



Научились создавать
резервные копии и
восстанавливать резервные
копии



Разобрали, как создавать
пользователей и управлять
правами

Домашнее задание

1

2

3

Дополнительные материалы

1

2

Николай Хащанов

Full-stack Developer

