4.2

Создаем стенд

Текущий стенд состоит из 1-ого узла: `vs01.sXXXXXX`, (XXXXXX - ваш номер студента).
Работа стенда 6 часов. 2 попытки запуска стенда.

1. Над текстом нажмите кнопку "Создать стенд". Каждый стенд создан под определенную тему курса. Сейчас вы находитесь в теме - **4. Валидация резервных копий** , этот стенд не подойдет под практические занятия из других пунктов курса.

Запуск обычно идёт до 10 минут, в редких случаях - до 30 минут.

2. После создания стенда вам нужно зайти по SSH на **sbox.slurm.io** с логином и паролем из личного кабинета (https://edu.slurm.io/):

```
ssh s000001@sbox.slurm.io
```

s000001 нужно заменить на ваш номер студента

Далее вам нужно перейти на сам стенд (sbox является просто jump-хостом и не понадобится кроме как для входа):

```
ssh vs01.s000001
```

s000001 снова нужно заменить на ваш личный номер студента

На хосте vs01 у вас есть полные права (включая беспарольный sudo):

```
sudo -i
```

Будьте осторожны и не сломайте стенд до прохождения всех заданий :)

Практика: Валидация резервных копий

Можете запустить скрипт на стенде `~/practice/lecture1/3.backup_validation.sh`, который запускал спикер в видео.

Или пройтись по шагам текущей практике
- https://gitlab.slurm.io/postgres/slurm_course/-/blob/main/practice/lecture1/3.backup_validation.md, также представлена она далее ниже по тексту.

Валидация резервных копий

Проверка резервных копий, наверное, одна из самых сложных проблем резервного копирования. Невозможно сто процентов быть уверенным, что созданная резервная копия может быть восстановлена, так как нельзя гарантировать, что данные кластера PostgreSQL будут успешно записаны на диск и останутся неизменными продолжительное время при сбое файловой системы или выхода из строя дискового рейд массива. Но есть меры, которые позволяют снизить риск возникновения таких ситуаций к минимуму или, во всяком случае, узнать о сбоях в хранении базы данных заблаговременно до появления критических ситуаций в работе СУБД PostgreSQL. Давайте попробуем смоделировать ситуацию, когда возможен сбой в базе данных.

1. Инициализируем новый кластер баз данных. Контрольные суммы выключены. Для этого остановим кластер на порту 5432 и удалим его каталог.

```
postgres$ pg_ctl -D /var/lib/pgsql/12/main5432 stop

postgres$ rm -rf /var/lib/pgsql/12/main5432/*

postgres$ pg_ctl initdb -D /var/lib/pgsql/12/main5432

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 "en_US.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/pgsql/12/main5432 ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ... UTC
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
syncing data to disk ... ok
initdb: warning: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the option -A, or
--auth-local and --auth-host, the next time you run initdb.
Success. You can now start the database server using:

    /usr/pgsql-12/bin/pg_ctl -D /var/lib/pgsql/12/main5432 -l logfile start
```

2. Последняя команда показывает, как надо запустить кластер. Меняем немного команду, для нашего удобства. Путь до файлов кластера баз данных должен быть заканчиваться на main5432, номер порта на котором будет запущен кластер. И флаг `-l logfile` нам не нужен, будем писать в стандартный файл логов postgres-а, путь до которого можно посмотреть в конфигурационном файле `postgresql.conf`.

```
postgres$ pg_ctl -D /var/lib/pgsql/12/main5432 start

waiting for server to start....2021-03-30 19:53:24.086 UTC [25044] LOG:  starting
PostgreSQL 12.6 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.8.5 20150623 (Red Hat
4.8.5-44), 64-bit
```

```
2021-03-30 19:53:24.086 UTC [25044] LOG:  listening on IPv4 address "127.0.0.1", port
5432
2021-03-30 19:53:24.090 UTC [25044] LOG:  listening on Unix socket
"/var/run/postgresql/.s.PGSQL.5432"
2021-03-30 19:53:24.097 UTC [25044] LOG:  listening on Unix socket
"/tmp/.s.PGSQL.5432"
2021-03-30 19:53:24.109 UTC [25044] LOG:  redirecting log output to logging collector
process
2021-03-30 19:53:24.109 UTC [25044] HINT:  Future log output will appear in directory
"log".
 done
server started
```

3. Проверяем состояние базы данных

```
postgres$ pg_ctl -D /var/lib/pgsql/12/main5432 status

pg_ctl: server is running (PID: 25044)
/usr/pgsql-12/bin/postgres "-D" "/var/lib/pgsql/12/main5432"
```

4. Создадим новую базу данных

```
postgres$ psql -c "create database test_checksum";
CREATE DATABASE
```

5. Запустим утилиту pgbench. Она сформирует тестовые таблицы, заполнить их данными. Вообще утилита pgbench предназначена для проведения нагрузочных тестов.

```
postgres$ pgbench -i -d test_checksum

dropping old tables...
NOTICE:  table "pgbench_accounts" does not exist, skipping
NOTICE:  table "pgbench_branches" does not exist, skipping
NOTICE:  table "pgbench_history" does not exist, skipping
NOTICE:  table "pgbench_tellers" does not exist, skipping
creating tables...
generating data...
100000 of 100000 tuples (100%) done (elapsed 0.12 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done.
```

6. Посмотрим какие данные есть в таблице pgbench_branches

```
postgres$ psql -d test_checksum -c "select * from pgbench_branches"
bid  | bbalance | filler
-----+----------+--------
   1 |        0 |
(1 row)
```

7. Найдем, где физически находится данная таблица на диске. Для этого воспользуемся системной функцией pg_relation_filepath

```
postgres$ psql -d test_checksum -c "select pg_relation_filepath('pgbench_branches')"
pg_relation_filepath
----------------------
 base/16428/16442
(1 row)
```

8. Остановим сервер PostgeSQL

```
postgres$ pg_ctl -D /var/lib/pgsql/12/main5432/ stop
```

```
waiting for server to shut down.... done
server stopped
```

9. Зайдем в файл таблицы `pgbench_branches` и в конце файла вставим произвольный текст

```
postgres$ nano /var/lib/pgsql/12/main5432/base/16428/16442
```

10. Запустим наш кластер

```
postgres$ pg_ctl -D /var/lib/pgsql/12/main5432/ start
waiting for server to start....2021-03-30 18:37:22.712 UTC [24423] LOG:  starting
PostgreSQL 12.6 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.8.5 20150623 (Red Hat
4.8.5-44), 64-bit
2021-03-30 18:37:22.712 UTC [24423] LOG:  listening on IPv4 address "127.0.0.1", port
5432
2021-03-30 18:37:22.715 UTC [24423] LOG:  listening on Unix socket
"/var/run/postgresql/.s.PGSQL.5432"
2021-03-30 18:37:22.721 UTC [24423] LOG:  listening on Unix socket
"/tmp/.s.PGSQL.5432"
2021-03-30 18:37:22.734 UTC [24423] LOG:  redirecting log output to logging collector
process
2021-03-30 18:37:22.734 UTC [24423] HINT:  Future log output will appear in directory
"log".
 done
server started
```

11. Посмотрим какие данные у нас в таблице `pgbench_branches`

```
postgres$ psql -d test_checksum -c "select * from pgbench_branches"
```

12. Пересоздадим таблицу `pgbench_branches`

```
postgres$ psql -d test_checksum -c "drop table pgbench_branches"

postgres$ pgbench -i -d test_checksum

dropping old tables...
NOTICE:  table "pgbench_branches" does not exist, skipping
creating tables...
generating data...
100000 of 100000 tuples (100%) done (elapsed 0.15 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done.
```

13. Проверим включены ли у нас контрольные суммы или нет

```
postgres$ psql -d test_checksum -c "select name,setting from pg_settings where name =
'data_checksums';"
      name      | setting
----------------+---------
 data_checksums | off
(1 row)
```

С 12 версии PostgreSQL стала выпускаться утилита `pg_checksums`, которая позволяет включать или отключать контрольные сумму в базе данных. До 12 версии можно было включить контрольные суммы только переездом на ту же версию PostgreSQL через логические резервные копии.

14. Чтобы включить контрольные суммы нужно остановить текущий экземпляр PostgreSQL

```
postgres$ pg_ctl -D /var/lib/pgsql/12/main5432 stop
waiting for server to shut down.... done
server stopped
```

15. Выполнить утилиту `pg_checksums`, передав в качестве входного параметра путь до файлов базы данных

```
postgres$ pg_checksums -D /var/lib/pgsql/12/main5432 -e
Checksum operation completed
Files scanned:  1274
Blocks scanned: 6036
pg_checksums: syncing data directory
pg_checksums: updating control file
Checksums enabled in cluster
```

16. Запускаем кластер PostgreSQL

```
postgres$ pg_ctl -D /var/lib/pgsql/12/main5432 start
```

17. Теперь проделаем опять те же действия внесению по внесению произвольного текста в файл баз данных. Для этого смотрим какие есть записи в таблице pgbench_branches

```
postgres$ psql -d test_checksum -c "select * from pgbench_branches"
bid  | bbalance | filler
-----+----------+--------
   1 |        0 |
(1 row)
```

18. Находим путь до файла

```
postgres$ psql -d test_checksum -c "select pg_relation_filepath('pgbench_branches')"
pg_relation_filepath
----------------------
 base/16384/16420
(1 row)
```

19. Останавливаем кластер PostgreSQL

```
postgres$ pg_ctl -D /var/lib/pgsql/12/main5432 stop
waiting for server to shut down.... done
server stopped
```

20. Вносим произвольный текст в файл base/16384/16420

```
postgres$ nano /var/lib/pgsql/12/main5432/base/16384/16420
```

21. Запускаем кластер PostgreSQL

```
postgres$ pg_ctl -D /var/lib/pgsql/12/main5432 start
waiting for server to start....2021-03-30 20:13:10.697 UTC [25287] LOG:  starting
PostgreSQL 12.6 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.8.5 20150623 (Red Hat
4.8.5-44), 64-bit
2021-03-30 20:13:10.697 UTC [25287] LOG:  listening on IPv4 address "127.0.0.1", port
5432
2021-03-30 20:13:10.701 UTC [25287] LOG:  listening on Unix socket
"/var/run/postgresql/.s.PGSQL.5432"
2021-03-30 20:13:10.708 UTC [25287] LOG:  listening on Unix socket
"/tmp/.s.PGSQL.5432"
2021-03-30 20:13:10.721 UTC [25287] LOG:  redirecting log output to logging collector
process
2021-03-30 20:13:10.721 UTC [25287] HINT:  Future log output will appear in directory
"log".
 done
```

```
server started
```

22. Пробуем прочитать данные с таблицы `pgbench_branches`

```
postgres$ psql -d test_checksum -c "select * from pgbench_branches"
WARNING:  page verification failed, calculated checksum 10191 but expected 56854
ERROR:  invalid page in block 0 of relation base/16384/16420
```

И получаем ошибку о несовпадении контрольный сумм на страницы данных

23. Но как нам все-таки прочитать данные из таблицы даже при том, что у нас контрольные суммы не совпадают?
Для этого есть специальный системный параметр `ignore_checksum_failure`. При значении `on` данный параметр позволяет игнорировать ошибки, связанные с контрольными суммами. Попробуем им воспользоваться и извлечь данные из таблицы `pgbench_branches`

```
postgres$ psql -d test_checksum -c "SET ignore_checksum_failure = on; select * from
pgbench_branches"
WARNING:  page verification failed, calculated checksum 10191 but expected 56854
    bid     |  bbalance  | filler
------------+------------+--------
 1717990241 | 1685284467 |
(1 row)
```

Видим, что запрос отработал, но данные сильно отличаются от исходных данных. Повреждена страница данных.

Домашнее задание
Физическое повреждение индекса

1. Найти файл индекса. Остановить экземпляр PostgreSQL. Внести в него изменения и запустить экземпляр PostgreSQL
2. Попробовать обратиться к индексу в запросе select
3. Проверить индекс с помощью расширения amcheck

Чек-лист

1. Был выбран индекс из базы данных.
2. Найден путь расположения индекса
3. Остановлен экземпляр PostgreSQL и внесены изменения в тело
4. Запущен экземпляр PostgreSQL и при обращении к индексу выдается ошибка
5. Проведена БД с помощью расширения amcheck

Решение:

1. Найти файл индекса. Остановить экземпляр PostgreSQL. Внести в него изменения и запустить экземпляр PostgreSQL

```
postgres$ psql -d test_checksum

psql (12.6)
Type "help" for help.
5432=> create extension amcheck;
CREATE EXTENSION
5432=> SELECT bt_index_check(c.oid), c.relname, c.relpages
FROM pg_index i
JOIN pg_opclass op ON i.indclass[0] = op.oid
JOIN pg_am am ON op.opcmethod = am.oid
JOIN pg_class c ON i.indexrelid = c.oid
JOIN pg_namespace n ON c.relnamespace = n.oid
WHERE am.amname = 'btree'
AND c.relpersistence != 't'
AND i.indisready AND i.indisvalid;
5432=> \di
                        List of relations
Schema |          Name           | Type  |  Owner   |      Table
-------+-------------------------+-------+----------+-----------------
public | pgbench_accounts_pkey | index | postgres | pgbench_accounts
public | pgbench_branches_pkey | index | postgres | pgbench_branches
public | pgbench_tellers_pkey  | index | postgres | pgbench_tellers
(3 rows)
```

```
5432=> SELECT pg_relation_filepath('pgbench_accounts_pkey');
pg_relation_filepath
----------------------
base/24586/24607
(1 row)

postgres$ nano ./12/main5432/base/24586/24607
```

2. Попробовать обратиться к индексу в запросе `select`

```
postgres$ psql -d test_checksum

psql (12.6)
Type "help" for help.
5432=> select * from pgbench_accounts where aid=1;
ERROR:  invalid page in block 3 of relation base/24621/24642
```

3. Проверить индекс с помощью расширения `amcheck`

```
5432=> SELECT bt_index_check(c.oid), c.relname, c.relpages

FROM pg_index i
JOIN pg_opclass op ON i.indclass[0] = op.oid
JOIN pg_am am ON op.opcmethod = am.oid
JOIN pg_class c ON i.indexrelid = c.oid
JOIN pg_namespace n ON c.relnamespace = n.oid
WHERE am.amname = 'btree'
AND c.relpersistence != 't'
AND i.indisready AND i.indisvalid;
ERROR:  item order invariant violated for index "pgbench_accounts_pkey"
DETAIL:  Lower index tid=(1,13) (points to heap tid=(0,12)) higher index
tid=(1,14) (points to heap tid=(0,10)) page lsn=0/11A8C1A0.
```