УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Распределенные системы хранения данных»

Лабораторная работа №3

Вариант 97

Студент

Иванов Е.Д.

P33111

Преподаватель

Николаев В. В.

Санкт-Петербург, 2023 г.

Задание: вариант 97.

Лабораторная работа включает настройку резервного копирования данных с основного узла на резервный, а также несколько сценариев восстановления. Узел из предыдущей лабораторной работы используется в качестве основного; новый узел используется в качестве резервного. В сценариях восстановления необходимо использовать копию данных, полученную на первом этапе данной лабораторной работы.

Требования к отчёту
Отчет должен быть самостоятельным документом (без ссылок на внешние ресурсы), содержать всю последовательность команд, содержимое скриптов по каждому пункту задания. Для демонстрации результатов приводить команду вместе с выводом (самой наглядной частью вывода, при необходимости).

1. Резервное копирование

1.1 Настроить резервное копирование с основного узла на резервный следующим образом: Периодические обособленные (standalone) полные копии. Полное резервное копирование (pg_basebackup) по расписанию (cron) два раза в сутки. Необходимые файлы WAL должны быть в составе полной копии, отдельно их не архивировать. Срок хранения копий на основной системе - 1 неделя, на резервной - 1 месяц. По истечении срока хранения, старые архивы должны автоматически уничтожаться.

1.2 Подсчитать, каков будет объем резервных копий спустя месяц работы системы, исходя из следующих условий: Средний объем измененных данных за сутки: ~500 МБ.

1.3 Проанализировать результаты.

2. Потеря основного узла
Этот сценарий подразумевает полную недоступность основного узла. Необходимо восстановить работу СУБД на резервном узле, продемонстрировать успешный запуск СУБД и доступность данных.

3. Повреждение файлов БД

Этот сценарий подразумевает потерю данных (например, в результате сбоя диска или файловой системы) при сохранении доступности основного узла. Необходимо выполнить полное восстановление данных из резервной копии и перезапустить СУБД на основном узле.

Ход работы:
3.1 Симулировать сбой: удалить с диска директорию любой таблицы со всем содержимым.
3.2 Проверить работу СУБД, доступность данных, перезапустить СУБД, проанализировать результаты.
3.3 Выполнить восстановление данных из резервной копии, учитывая следующее условие: Исходное расположение дополнительных табличных пространств недоступно - разместить в другой директории и скорректировать конфигурацию.
3.4 Запустить СУБД, проверить работу и доступность данных, проанализировать результаты.

4. Логическое повреждение данных

Этот сценарий подразумевает частичную потерю данных (в результате нежелательной или ошибочной операции) при сохранении доступности основного узла. Необходимо выполнить восстановление данных на основном узле следующим способом:
Генерация файла на резервном узле с помощью pg_dump и последующее применение файла на основном узле. Восстановление с использованием архивных WAL файлов.(СУБД должна работать в режиме архивирования WAL, потребуется задать параметры восстановления).

Ход работы:
4.1 В каждую таблицу базы добавить 2-3 новые строки, зафиксировать результат.
4.2 Зафиксировать время и симулировать ошибку:
В любой таблице с внешними ключами изменить внешние ключи случайным образом (INSERT, UPDATE)
4.3 Продемонстрировать результат.
4.4 Выполнить восстановление данных указанным способом.
4.5 Продемонстрировать и проанализировать результат.

Основной узел: ssh postgres1@pg160 ****
Резервный узел: ssh postgres0@pg155 ****

Ход выполнения:

1) Резервное копирование

Создадим юзера для будущих реплик
create role replica replication login password '1234';

Для создания бэкапов я использовал следующую команду:
pg_basebackup -D "${FULL_PATH_NAME}" -p 9120 -U replica -X stream -T "${INDEX_SPACE}"="${NEW_INDEX_SPACE}" -T "${TABLE_SPACE}"="${NEW_TABLE_SPACE}"
Таким образом получается создать полную копию без отдельного архивирования WAL файлов. Здесь replica – созданный мною пользователь с правами на репликацию(также для этого я изменил pg_hba.conf для возможности подключения для replication с использованием пароля, а пароль автоматически берётся из файла .pgpass)
Флаг -T используется для переноса в бэкап

Далее я создал два скрипта – один для создания бэкапов на основном и резервном узлах, а другой – для удаления бэкапов с основного узла и резервного спустя прошедшее время(согласно варианту).

Скрипт создания бэкапов:

```
#!/bin/bash

BACKUP_DIR="/var/db/postgres1/backups"
INDEX_SPACE="/var/db/postgres1/u10/dir1"
TABLE_SPACE="/var/db/postgres1/u11/tables"

DATE=$(date +%Y%m%d%H%M%S)
BACKUP_NAME="backup_${DATE}"

FULL_PATH_NAME="${BACKUP_DIR}/${BACKUP_NAME}"
NEW_INDEX_SPACE="/var/db/postgres1/index_backups/${BACKUP_NAME}"
NEW_TABLE_SPACE="/var/db/postgres1/tables_backups/${BACKUP_NAME}"

SECOND_STORAGE="postgres0@pg155:~"

pg_basebackup -D "${FULL_PATH_NAME}" -p 9120 -U replica -X stream -T "${INDEX_SPACE}"="${NEW_INDEX_SPACE}"
-T "${TABLE_SPACE}"="${NEW_TABLE_SPACE}"

/usr/local/bin/rsync -av "${FULL_PATH_NAME}" "${SECOND_STORAGE}/backups/${BACKUP_NAME}"
--rsync-path="/usr/local/bin/rsync"
/usr/local/bin/rsync -av "${NEW_INDEX_SPACE}" "${SECOND_STORAGE}/index_backups/${BACKUP_NAME}"
--rsync-path="/usr/local/bin/rsync"
/usr/local/bin/rsync -av "${NEW_TABLE_SPACE}" "${SECOND_STORAGE}/tables_backups/${BACKUP_NAME}"
--rsync-path="/usr/local/bin/rsync"
```

Скрипт удаления бэкапов(на основном узле):

```
#!/bin/bash

BACKUP_DIR_MAIN="/var/db/postgres1/backups"
INDEX_DIR_MAIN="/var/db/postgres1/index_backups"
TABLES_DIR_MAIN="/var/db/postgres1/tables_backups"

SECONDS_IN_WEEK=$(( 7 * 24 * 3600 ))

current_time=$(date +%s)

for backup_dir in "$BACKUP_DIR_MAIN"/*; do
    file_modified_time=$(stat -f %m "$backup_dir")
    time_diff=$((current_time - file_modified_time))

    if [ "$time_diff" -gt "$SECONDS_IN_WEEK" ]; then
        rm -rf "$backup_dir"
        echo "file(dir) deleted $backup_dir"
    fi
done

for index_dir in "$INDEX_DIR_MAIN"/*; do
    file_modified_time=$(stat -f %m "$index_dir")
    time_diff=$((current_time - file_modified_time))

    if [ "$time_diff" -gt "$SECONDS_IN_WEEK" ]; then
        rm -rf "$index_dir"
        echo "file(dir) deleted $index_dir"
    fi
done

for tables_dir in "$TABLES_DIR_MAIN"/*; do
    file_modified_time=$(stat -f %m "$tables_dir")
    time_diff=$((current_time - file_modified_time))
```

```
    if [ "$time_diff" -gt "$SECONDS_IN_WEEK" ]; then
        rm -rf "$tables_dir"
        echo "file(dir) deleted $tables_dir"
    fi
done
```

Скрипт удаления бэкапов(на резервном узле):

```
#!/bin/bash

BACKUP_DIR_MAIN="/var/db/postgres0/backups"
INDEX_DIR_MAIN="/var/db/postgres0/index_backups"
TABLES_DIR_MAIN="/var/db/postgres0/tables_backups"

SECONDS_IN_MONTH=$(( 30 * 24 * 3600 ))

current_time=$(date +%s)

for backup_dir in "$BACKUP_DIR_MAIN"/*; do
    file_modified_time=$(stat -f %m "$backup_dir")
    time_diff=$((current_time - file_modified_time))

    if [ "$time_diff" -gt "$SECONDS_IN_MONTH" ]; then
        rm -rf "$backup_dir"
        echo "file(dir) deleted $backup_dir"
    fi
done

for index_dir in "$INDEX_DIR_MAIN"/*; do
    file_modified_time=$(stat -f %m "$index_dir")
    time_diff=$((current_time - file_modified_time))

    if [ "$time_diff" -gt "$SECONDS_IN_MONTH" ]; then
        rm -rf "$index_dir"
        echo "file(dir) deleted $index_dir"
    fi
done

for tables_dir in "$TABLES_DIR_MAIN"/*; do
    file_modified_time=$(stat -f %m "$tables_dir")
    time_diff=$((current_time - file_modified_time))

    if [ "$time_diff" -gt "$SECONDS_IN_MONTH" ]; then
        rm -rf "$tables_dir"
        echo "file(dir) deleted $tables_dir"
    fi
done
```

Добавим расписание задач-скриптов через *crontab -e (и на резервном узле тоже):*

```
[postgres1@pg160 ~]$ crontab -e
crontab: no crontab for postgres1 - using an empty one
0 0,12 * * * bash /var/db/postgres1/create_script.sh
10 0,12 * * * bash /var/db/postgres1/delete_script_main.sh
```

```
[postgres0@pg155 ~]$ crontab -l
10 0,12 * * * bash /var/db/postgres0/delete_script_reserve.sh
```

Каждый день в 00:00 и 12:00 запускается скрипт создания бэкапа(и создание его копии на резервном узле). Также каждый день в 00:10 и 12:10 происходит запуск скрипта удаления старых бэкапов(в соответствие с временем их последней модификации)

Также, перед запуском всех скриптов в формате cron задач, я проверил то, что созданный бэкап будет сохранять все данные. Для этого я остановил кластер на первом узле и запустил бэкап, подключился к нему по тому же порту, проверил наличие таблиц и данных.

```
[postgres1@pg160 ~]$ pg_ctlackups/backup_20230611130456/ start
ожидание запуска сервера....2023-06-11 13:09:36.541 MSK [43551] СООБЩЕНИЕ:  передача вывода в протокол процессу сбора протоколов
2023-06-11 13:09:36.541 MSK [43551] ПОДСКАЗКА:  В дальнейшем протоколы будут выводиться в каталог "log".
 готово
сервер запущен
[postgres1@pg160 ~]$ psql -p 9120 -U replica -d whitebunny10
psql (14.2)
Введите "help", чтобы получить справку.

whitebunny10=# \dd
        Описание объекта
 Схема | Имя | Объект | Описание
-------+-----+--------+----------
(0 строк)

whitebunny10=# \d
                  Список отношений
 Схема  |       Имя       |        Тип         | Владелец
--------+-----------------+--------------------+-----------
 public | football        | таблица            | postgres1
 public | football_id_seq | последовательность | postgres1
(2 строки)

whitebunny10=# select * from footbal;
ОШИБКА:  отношение "footbal" не существует
СТРОКА 1: select * from footbal;
                        ^
whitebunny10=# select * from football;
 id | name
----+-------
  1 | zenit
  2 | barsa
(2 строки)
```

Таким образом, всё работает при запуске кластера из бэкапа на основном узле.

Проведём расчёты объёма резервных копий за месяц работы: сначала найдём размер бэкапа

Объём бэкапа без wal файлов 10MB,
Также WAL файлы хранятся по 16MB по умолчанию, поэтому будем считать, что в день добавляется два бэкапа с размером на 256MB каждый больше предыдущего(в сумме считаю, что изменений за день будет записано 512MB).

Тогда на основном узле будет занято место бэкапами за последние 7 дней(из 30)
По формуле арифметической прогрессии найдем сумму объёмов:
$(a_{46} + a_{60})/2 \ * \ 14 \ = \ 188706\,MB \sim 184,3\,GB$
На резервном узле также будут сохранены бэкапы за первые 23 дня:

$(a_1 + a_{60})/2 * 60 = 455460\ MB \sim 444,8\ GB$

На обоих узлах в сумме объём данных получится около 629 GB за месяц работы

Анализ результатов: такие копии занимают очень много места(а каждая последующая больше предыдущей). Если бы копии были инкрементальными, то такой проблемы не было. Также можно архивировать WAL файлы, при достижении определённого размера, чтобы бэкапы весили меньше.

2) Потеря основного узла

Итак, надо создать кластер из созданного ранее бэкапа
Для этого я инициализирую новый кластер, а после подменяю datadir на каталог бэкапа.
На основном узле бэкап бы запустился(как я показывал ранее), но на резервном узле возникает проблема с путями до *indexspace* и *tablespace*. Для того, чтобы кластер можно было запустить надо поменять ссылки на но новые пути до index,table space.

```
[postgres0@pg155 ~/backups/backup_20230611173917/backup_20230611173917/pg_tblspc]$ ls -l
total 1
lrwx------  1 postgres0  postgres  53 11 июня  17:39 16386 -> /var/db/postgres1/index_backups/backup_20230611173917
lrwx------  1 postgres0  postgres  54 11 июня  17:39 16389 -> /var/db/postgres1/tables_backups/backup_20230611173917
[postgres0@pg155 ~/backups/backup_20230611173917/backup_20230611173917/pg_tblspc]$ ln -s ~/index_backups/backup_20230611173917/backup_20230611173917 16386
ln: 16386: File exists
[postgres0@pg155 ~/backups/backup_20230611173917/backup_20230611173917/pg_tblspc]$ rm 16386
[postgres0@pg155 ~/backups/backup_20230611173917/backup_20230611173917/pg_tblspc]$ ln -s ~/index_backups/backup_20230611173917/backup_20230611173917
[postgres0@pg155 ~/backups/backup_20230611173917/backup_20230611173917/pg_tblspc]$ ls
16386   16389
[postgres0@pg155 ~/backups/backup_20230611173917/backup_20230611173917/pg_tblspc]$ ls -l
total 1
lrwxr-xr-x  1 postgres0  postgres  75 11 июня  18:09 16386 -> /var/db/postgres0/index_backups/backup_20230611173917/backup_20230611173917
lrwx------  1 postgres0  postgres  54 11 июня  17:39 16389 -> /var/db/postgres1/tables_backups/backup_20230611173917
[postgres0@pg155 ~/backups/backup_20230611173917/backup_20230611173917/pg_tblspc]$ ln -s ~/tables_backups/backup_20230611173917/backup_20230611173917 16389
ln: 16389: File exists
[postgres0@pg155 ~/backups/backup_20230611173917/backup_20230611173917/pg_tblspc]$ rm 16389
[postgres0@pg155 ~/backups/backup_20230611173917/backup_20230611173917/pg_tblspc]$ ln -s ~/tables_backups/backup_20230611173917/backup_20230611173917
[postgres0@pg155 ~/backups/backup_20230611173917/backup_20230611173917/pg_tblspc]$ ls -l
total 1
lrwxr-xr-x  1 postgres0  postgres  75 11 июня  18:09 16386 -> /var/db/postgres0/index_backups/backup_20230611173917/backup_20230611173917
lrwxr-xr-x  1 postgres0  postgres  76 11 июня  18:10 16389 -> /var/db/postgres0/tables_backups/backup_20230611173917/backup_20230611173917
[postgres0@pg155 ~/backups/backup_20230611173917/backup_20230611173917/pg_tblspc]$
```

После этого кластер запускается без проблем:

```
[postgres0@pg155 ~]$ pg_ctl -D 'backups/backup_20230611173917/backup_20230611173917' -l logfile start
ожидание запуска сервера.... готово
сервер запущен
[postgres0@pg155 ~]$ psql -p 9120 -U replica -d whitebunny10
Пароль пользователя replica:
psql (14.2)
Введите "help", чтобы получить справку.

whitebunny10=# \d
                  Список отношений
 Схема  |       Имя       |        Тип        | Владелец
--------+-----------------+-------------------+-----------
 public | football        | таблица           | postgres1
 public | football_id_seq | последовательность | postgres1
(2 строки)

whitebunny10=# select * from football;
 id | name
----+-------
  1 | zenit
  2 | barsa
(2 строки)
```

Видно, что данные на месте, а значит восстановить потерянный узел удалось успешно. В целом, если резервный узел имеет схожую файловую систему(в нашем

случае необходимо было лишь поднастроить ссылки на табличные пространства, что можно учесть в скрипте переноса бэкапа на резервный узел), то восстановление не создаёт проблем.

3) Повреждение файлов БД

Я запустил кластер с одного из бэкапов и удалил табличное пространство индексов.

```
[postgres1@pg160 ~]$ pg_ctl -D 'backups/backup_20230611173917' -l logfile start
ожидание запуска сервера.... готово
сервер запущен
[postgres1@pg160 ~]$ cd index_backups/backup_20230611173917/
[postgres1@pg160 ~/index_backups/backup_20230611173917]$ rm -rf ./*
[postgres1@pg160 ~/index_backups/backup_20230611173917]$
```

```
[postgres1@pg160 ~/index_backups/backup_20230611173917]$ psql -p 9120 -U replica -d whitebunny10
psql (14.2)
Введите "help", чтобы получить справку.

whitebunny10=# \d
                  Список отношений
 Схема  |       Имя       |        Тип         | Владелец
--------+-----------------+--------------------+-----------
 public | football        | таблица            | postgres1
 public | football_id_seq | последовательность | postgres1
(2 строки)

whitebunny10=# select * from football;
ОШИБКА:  не удалось открыть файл "pg_tblspc/16386/PG_14_202107181/16384/16399": Нет такого файла или каталога
whitebunny10=# insert into football values (default, 'shahtar');
ОШИБКА:  не удалось открыть файл "pg_tblspc/16386/PG_14_202107181/16384/16399": Нет такого файла или каталога
whitebunny10=#
```

При этом подключиться к базе можно(потому что табличное пространство за это не отвечает), но при операциях, затрагивающих взаимодействие с табличным пространством возникает ошибка.

Теперь для восстановления индексного пространства надо поменять ссылку в pg_tblspc на каталог с индексным пространством из бэкапа(желательно последнего). Этот процесс полностью аналогичен процессу в пункте 2, так как там также не было доступа к табличным пространствам после переноса на резервный узел.

```
[postgres1@pg160 ~/backups/backup_20230611173917/pg_tblspc]$ ls -l
total 1
lrwx------  1 postgres1  postgres  53 11 июня  17:39 16386 -> /var/db/postgres1/index_backups/backup_20230611173917
lrwx------  1 postgres1  postgres  54 11 июня  17:39 16389 -> /var/db/postgres1/tables_backups/backup_20230611173917
[postgres1@pg160 ~/backups/backup_20230611173917/pg_tblspc]$ ln -s /var/db/postgres1/u10/dir1 16386
ln: 16386/dir1: File exists
[postgres1@pg160 ~/backups/backup_20230611173917/pg_tblspc]$ rm -rf 16386
[postgres1@pg160 ~/backups/backup_20230611173917/pg_tblspc]$ ln -s /var/db/postgres1/u10/dir1
[postgres1@pg160 ~/backups/backup_20230611173917/pg_tblspc]$ ls -l
total 1
lrwxr-xr-x  1 postgres1  postgres  26 11 июня  20:09 16386 -> /var/db/postgres1/u10/dir1
lrwx------  1 postgres1  postgres  54 11 июня  17:39 16389 -> /var/db/postgres1/tables_backups/backup_20230611173917
```

После этого сервер запускается и работает в штатном режиме:

```
[postgres1@pg160 ~/backups]$ pg_ctl -D backup_20230611173917 -l logfile start
ожидание запуска сервера.... готово
сервер запущен
```

```
[postgres1@pg160 ~/backups]$ pg_ctl -D backup_20230611173917 -l logfile start
ожидание запуска сервера.... готово
сервер запущен
[postgres1@pg160 ~/backups]$ psql -p 9120 -U replica -d whitebunny10
psql (14.2)
Введите "help", чтобы получить справку.

whitebunny10=# select * from football;
 id | name
----+-------
  1 | zenit
  2 | barsa
(2 строки)
```

Результаты: при повреждении табличного пространства ошибка об этом появится при любом взаимодействии с ним внутри БД, а после этого можно понять в чём проблема. При создании бэкапов с маппингом табличных пространств создаются копии табличных пространств, так что можно быстро восстановить табличное пространство из любого бэкапа.

4) Логическое повреждение данных

Создадим dump и добавим в таблицу логически неверные данные:

```
[postgres1@pg160 ~]$ pg_dump -p 9120 -U replica -d whitebunny10 > dump.sql
```

```
[postgres1@pg160 ~]$ ls
backups                 dump.sql                tables_backups          u11
create_script.sh        index_backups           u08
delete_script_main.sh   logfile                 u10
[postgres1@pg160 ~]$ psql -p 9120 -U replica -d whitebunny10
psql (14.2)
Введите "help", чтобы получить справку.

whitebunny10=# insert into football values (default, 'nba');
INSERT 0 1
whitebunny10=# insert into football values (default, 'hokkey');
INSERT 0 1
whitebunny10=# select * from football;
 id |  name
----+--------
  1 | zenit
  2 | barsa
  3 | nba
  4 | hokkey
(4 строки)
```

Теперь можно вернуть состояние из dump’а, например, удалив таблицу или БД и

внеся дамп как команды на ввод(я удалил таблицу):

```
[postgres1@pg160 ~]$ psql -p 9120 -d whitebunny10 -U replica
psql (14.2)
Введите "help", чтобы получить справку.

whitebunny10=# drop table football;
DROP TABLE
```

```
[postgres1@pg160 ~]$ psql -p 9120 -d whitebunny10 -U replica < ./dump.sql
SET
SET
SET
SET
SET
 set_config
------------

(1 строка)

SET
SET
SET
SET
SET
SET
CREATE TABLE
ALTER TABLE
CREATE SEQUENCE
ALTER TABLE
ALTER SEQUENCE
ALTER TABLE
COPY 2
 setval
--------
      2
(1 строка)

ALTER TABLE
[postgres1@pg160 ~]$ psql -p 9120 -d whitebunny10 -U replica
psql (14.2)
Введите "help", чтобы получить справку.

whitebunny10=# select * from football;
 id | name
----+-------
  1 | zenit
  2 | barsa
(2 строки)
```

Таким образом, логическое восстановление прошло успешно.

Вывод: в ходе лабораторной работы познакомился с методами физического и логического резервного копирования кластера или данных. Также разобрался с восстановлением работоспособности кластера(или целостности данных) и присущими проблемами.