МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное образовательное учреждение высшего образования “Национальный исследовательский университет ИТМО”

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №3

по дисциплине

“Распределённые системы хранения данных”

вариант 234562

Выполнили:

Иванов Матвей Сергеевич

Шульга Артём Игоревич

группа P33111

Преподаватель:

Николаев Владимир Вячеславович

г. Санкт-Петербург, 2024

Задание

Цель работы - настроить процедуру периодического резервного копирования базы данных, сконфигурированной в ходе выполнения лабораторной работы №2, а также разработать и отладить сценарии восстановления в случае сбоев.

Узел из предыдущей лабораторной работы используется в качестве основного. Новый узел используется в качестве резервного. Учётные данные для подключения к новому узлу выдаёт преподаватель. В сценариях восстановления необходимо использовать копию данных, полученную на первом этапе данной лабораторной работы.

Этап 1. Резервное копирование

- Настроить резервное копирование с основного узла на резервный следующим образом:
    - Периодические полные копии + непрерывное архивирование.
    - Включить для СУБД режим архивирования WAL;
    - Настроить копирование WAL (scp) на резервный узел;
    - Настроить полное резервное копирование (pg_basebackup) по расписанию (cron) раз в неделю.
    - Созданные полные копии должны сразу копироваться (scp) на резервный хост.
    - Срок хранения копий на основной системе - 1 неделя, на резервной - 4 недели.
    - По истечении срока хранения, старые архивы и неактуальные WAL должны автоматически уничтожаться.
- Подсчитать, каков будет объем резервных копий спустя месяц работы системы, исходя из следующих условий:
    - Средний объем новых данных в БД за сутки: 50 МБ.
    - Средний объем измененных данных за сутки: 750 МБ.
- Проанализировать результаты.

Этап 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.

Этап 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:

- Симулировать сбой:
    - удалить с диска директорию WAL со всем содержимым.
- Проверить работу СУБД, доступность данных, перезапустить СУБД, проанализировать результаты.
- Выполнить восстановление данных из резервной копии, учитывая следующее условие:
    - исходное расположение дополнительных табличных пространств недоступно - разместить в другой директории и скорректировать конфигурацию.
- Запустить СУБД, проверить работу и доступность данных, проанализировать результаты.

Этап 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.

Ход работы:

- В каждую таблицу базы добавить 2-3 новые строки, зафиксировать результат.
- Зафиксировать время и симулировать ошибку:
    - удалить любые две таблицы (DROP TABLE)
- Продемонстрировать результат.
- Выполнить восстановление данных указанным способом.
- Продемонстрировать и проанализировать результат.

Данные для подключения к узлам:

1. Основной узел
    a. Название pg165
    b. Пользователь: postgres0
2. Резервный узел
    a. Название pg189
    b. Пользователь: postgres6

Ход выполнения

Этап 1. Резервное копирование

В начале подружим наши узлы. Сгенерируем ssh-ключи на каждой из выделенных нод и добавим их публичные ключи в authorized_keys друг друга.

```
# Для основного узла
ssh-keygen -t rsa
ssh-copy-id -i ~/.ssh/id_rsa.pub postgres0@pg189
# Для резервного узла
ssh-keygen -t rsa
ssh-copy-id -i ~/.ssh/id_rsa.pub postgres0@pg165
```

Для начала добавим конфигурации в postgresql.conf на основном узле:

```
# Установим минимальный уровень WAL на реплику
wal_level = replica

# Включим для СУБД режим архивирования WAL
archive_mode = on

# Настроим копирование WAL (scp) на резервный узел
archive_command = 'scp %p postgres0@pg189:~/backups/wal/%f'

# Установим принудительную смену сегмента WAL каждые 30 минут
# 30 минут из расчёта, что 24 * 60 мин.сут. / (50 МБ + 750 МБ)  *  16 МБ (размер WAL)
archive_timeout = 1800

#  Зададим команду для получения архивов WAL файлов при восстановлении:
restore_command = 'scp postgres0@pg189:~/backups/wal/%f “%p”'
```

Обновим конфигурацию Postgres:

pg_ctl -D $HOME/u22/pg1651 restart

Перезагрузка обязательна, так как мы включили режим архивирования WAL

Скрипт для бэкапов

Теперь напишем bash скрипт, который будет делать следующие действия:

1. осуществлять полное резервное копирование кластера БД
2. отправлять копию бэкапа на резервный сервер

```
#!/bin/bash
# Название файла: backup.sh
# Устанавливаем флаг, чтобы скрипт прерывался,
# если одна из команд завершится с ошибкой
set -e

# Формируем название для бэкапа на основе текущей даты
TODAY=$(date +"%d%m%y")
BACKUP_DIR=$HOME/backups
BACKUP=$BACKUP_DIR/$TODAY

# Пишем в $BACKUP, пишем файлы как есть (-Fp то есть plain)
# Флагом -T мы указываем, куда нужно писать бэкап
# для созданного в прошлой ЛР табличного пространства
pg_basebackup -D $BACKUP/u22/pg1651/ -Fp -p 9165 -T
$HOME/u02/zne78=$BACKUP/u02/zne78/

# Сжимаем в архив и удаляем директорию
# Можно было бы сделать более оптимально через опцию --remove-files у tar,
# но на выделенной ноде он слишком старый для такого
cd $BACKUP_DIR
tar -czf $TODAY.tar.gz $TODAY
rm -rf $TODAY

# Копируем архив на резервный сервер
scp $TODAY.tar.gz postgres0@pg189:~/backups/
```

Проверим работу скрипта, запустим его:

```
bash backup.sh
```

И посмотрим содержимое backups на обоих узлах:

```
// backups на основном узле
-rw-r--r--  1 postgres0  postgres  6899185  6 апр.  20:33 060424.tar.gz


// backups на резервном узле
```

```
-rw-r--r--  1 postgres0  postgres  6899185  6 апр.  20:33 060424.tar.gz
drwxr-xr-x  2 postgres0  postgres        5  6 апр.  20:33 wal
```

// В директории backups/wal на резервном узле можем увидеть файлы
`000000010000000000000008` `000000010000000000000009`
`000000010000000000000009.00000028.backup`

Скрипт для очистки

Также напишем bash скрипт, который будет удалять старые бэкапы (которые были модифицированы более недели назад) с точностью до минуты:

```
#!/bin/bash
# Название файла: cleanup.sh
# Устанавливаем флаг, чтобы скрипт прерывался,
# если одна из команд завершится с ошибкой
set -e
# Задаём директорию для очистки и количество минут жизни файлов (неделя)
BACKUP_DIR=$HOME/backups
CURRENT_TIME=$(date +%s)
MAX_MINUTES=$((7 * 24 * 60))

# Проходимся по файлам в директории и удаляем старые
for file in $BACKUP_DIR/*
do
  if [[ -f "$file" ]]; then
    MODIFIED_TIME=$(date -r $file +%s)
    DIFF=$(( (CURRENT_TIME - MODIFIED_TIME) / 60 ))
    if [[ $DIFF -gt $MAX_MINUTES ]]; then
      rm -f $file
    fi
  fi
done
```

Проверим работу скрипта для очистки, для этого создадим тестовые файлы с заданными датами модификации:
// 1 января, должен быть удалён
`touch -d "2024-01-01T00:00:00" backups/test.txt`
// 7 дней назад, должен быть удалён
`touch -d "$(date -r $(($(date +%s) - 60*60*24*7)) +%Y-%m-%dT%H:%M:%S)" backups/test2.txt`
// 6 дней назад, должен остаться

```
touch -d "$(date -r $(($(date +%s) - 60*60*24*6)) +%Y-%m-%dT%H:%M:%S)"
backups/test3.txt
```

Получим следующие файлы в директории backups:

```
-rw-r--r--  1 postgres0  postgres  6899185  6 апр.  20:33 060424.tar.gz
-rw-r--r--  1 postgres0  postgres        0  1 янв.  00:00 test.txt
-rw-r--r--  1 postgres0  postgres        0 30 марта 21:32 test2.txt
-rw-r--r--  1 postgres0  postgres        0 31 марта 21:41 test3.txt
```

Запустим скрипт

```
bash cleanup.sh
```

И выведем содержимое директории:

```
-rw-r--r--  1 postgres0  postgres  6899185  6 апр.  20:33 060424.tar.gz
-rw-r--r--  1 postgres0  postgres        0 31 марта 21:41 test3.txt
```

Аналогичный скрипт очистки добавим на резервный узел, лишь с изменением, чтобы файлы очищались за 4 недели, а также очищались неактуальные WAL архивы (с помощью утилиты pg_archivecleanup):

```
MAX_MINUTES=$((4 * 7 * 24 * 60))
... # всё то же самое, что и в cleanup.sh

WAL_DIR=$BACKUP_DIR/wal
PREVIOUS=""

for file in $(ls -r $WAL_DIR/*.backup)
do
  if [[ -f "$file" ]]; then
    MODIFIED_TIME=$(date -r $file +%s)
    DIFF=$(( (CURRENT_TIME - MODIFIED_TIME) / 60 ))
    if [[ $DIFF -gt $MAX_MINUTES ] && [ $PREVIOUS ]]; then
      # Так как pg_archivecleanup удаляет архивы строго до данного
      # То берём предыдущий файл
      pg_archivecleanup -d $WAL_DIR $PREVIOUS
      # Удаляем ненужный **.backup файл
      rm -f $file
      break
    fi
    PREVIOUS=$(basename $file)
  fi
done
```

Проверим работу на резервном сервере, для этого снова запустим backup.sh. Получим следующие файлы в backups/wal на резервном узле:

```
000000010000000000000008                    000000010000000000000000A
000000010000000000000009                    000000010000000000000000B
000000010000000000000009.00000028.backup    000000010000000000000000C
                                 000000010000000000000000C.00000028.backup
```

И чтобы проверить, что скрипт отработает, изменим дату модификации для 000000010000000000000009.00000028.backup:

```
touch -d "2024-01-01T00:00:00"
backups/wal/000000010000000000000009.00000028.backup
```

Запустим скрипт очистки на резервном узле и посмотрим удалились ли файлы:

```
bash cleanup.sh
```

Содержимое директории:

```
000000010000000000000000C    000000010000000000000000C.00000028.backup
```

Значит скрипт отработал корректно. (Файлы **A и **B удалились, так как это не записанные в бэкап **9 WAL файлы, для **C они уже учтены, так что в них больше нет необходимости). Ну и важно заметить, что хоть и в нашем примере остался только один последний WAL для бэкапа, в реальности же их будет оставаться 4 со всеми промежуточными WAL, что обеспечивает ещё большую надежность.

Установка на cron

Поставим исполнение приведённых выше файлов на cron, чтобы они отрабатывали регулярно. Для этого, с помощью команды crontab -e добавим записи в crontab файл.

Для основного узла:

```
0 12 * * 0 /var/db/postgres0/backup.sh
30 * * * * /var/db/postgres0/cleanup.sh
```

Для резервного узла:

```
30 * * * * /var/db/postgres0/cleanup.sh
```

Объём резервных файлов

Подсчитаем объём резервных файлов за месяц при условии, что

- Средний объем новых данных в БД за сутки: 50 МБ.
- Средний объем измененных данных за сутки: 750 МБ.

Будем считать, что на сервере активирован auto vacuum, который будет удалять старые данные.

Размер кластера БД на начало месяца:

```
du -hs u22/ u02/
 10M  u22/
4,7M  u02/
```

Округлим начальное значение до 15 МБ

Для начала разберемся на что будет влиять каждый из данных параметров в нашем случае при создании полного бэкапа БД:

- Средний объем новых данных в БД за сутки:
  - Объём хранимых файлов данных кластером
  - Объём WAL файлов
- Средний объем измененных данных за сутки:
  - Объём WAL файлов

Изменённые данные не будут увеличивать объём хранимых данных, так как если у нас, например, у одного пользователя был баланс 10.000, а стал 15.000, новых данных не добавилось.

Как мы уже рассчитывали ранее для данных условий, мы знаем, что новые сегменты WAL файлов создаются каждые 30 минут и каждый сегмент имеет размер 16 МБ, получаем, что за сутки объём WAL файлов составит:

Сутки = 16 МБ * 24 часа * 2 раза в час = 768 МБ

Тогда за неделю 768 МБ * 7 = 5 376 МБ = 5.25 ГБ

Нам важен именно объём за неделю, так как каждую неделю будет происходить полное резервное копирование и старые (неактуальные) WAL файлы очищаются

Значит за месяц у нас будет создано 4 резервные копии и мы получим размер WAL:

S_{wal} = 5.25 * 4 = 21 ГБ

Объём данных можно рассчитывать по формуле арифметической прогрессии:

$$S_n = \frac{2 * a_0 + d * (n-1)}{2} * n$$

Тогда объём бэкапов файлов данных за 1, 2, 3 и 4 недели:

$$S_7 = \frac{2 * 15 + 50 * (7-1)}{2} * 7 = 1\,155 \text{ МБ} \approx 1.13 \text{ ГБ}$$
$$S_{14} = \frac{2 * 15 + 50 * (14-1)}{2} * 14 = 4\,760 \text{ МБ} \approx 4.65 \text{ ГБ}$$
$$S_{21} = \frac{2 * 15 + 50 * (21-1)}{2} * 21 = 10\,815 \text{ МБ} \approx 10.56 \text{ ГБ}$$
$$S_{28} = \frac{2 * 15 + 50 * (21-1)}{2} * 21 = 19\,320 \text{ МБ} \approx 18.87 \text{ ГБ}$$

Получаем в сумме объём резервных файлов данных будет:

$$S = S_7 + S_{14} + S_{21} + S_{28} = 35.21 \text{ ГБ}$$

Так как каждый из этих резервных файлов будет также сжат с помощью gzip с степенью сжатия 6, который даёт коэффициент сжатия:

```
gzip -l 060424.tar.gz
compressed uncompressed  ratio uncompressed_name
   6899185     54723584  87.3% 060424.tar
```

Это значит, что наши tar.gz архивы будут занимать на ~87% меньше места, значит:

$$S_{comp} = S * (1 - 0.87) = 35.21 * 0.13 \approx 4.58 \text{ ГБ}$$

Учитывая, что 87% не абсолютный показатель и может слегка варьироваться, то предположим, что $S_{comp} \approx 5$ ГБ

Просуммируем занимаемый объём WAL и объем архивов, получим:

$$S_{total} = S_{wal} + S_{comp} = 21 + 5 = 26 \text{ ГБ}$$

Что является вполне приемлемым результатом.

Но также стоит заметить, что по заданию ЛР требуется хранить WAL архивы за 4 недели и удалять более старые не актуальные архивы. Так вот они занимают внушительный размер, но полезными нам при восстановлении будут лишь WAL архивы созданные после последнего бэкапа и, кроме как сильного повышения надёжности, нет смысла хранить более старые WAL архивы.

Этап 2. Потеря основного узла

Изначальные данные в основном узле:

```
postgres0=# SELECT * FROM dobro;
 id | dobro_amount | dobro_reciever_id
----+--------------+-------------------
  1 |            2 |                 3
  2 |            3 |                 1
  3 |            1 |                 2
  4 |            5 |                 1
  5 |            6 |                 7
(5 строк)
```

Определим последний созданный бэкап в папке backups на резервном узле и распакуем его с помощью команды:
tar -xvf backups/070424.tar.gz

Подкорректируем символические ссылки на табличные пространства в pg_tblspc:
Посмотрим куда ведут ссылки:
ls -l # /var/db/postgres0/backups/070424/u02/zne78
Удалим ссылки: rm -rf 16386
Перевяжем на корректное место: ln -s /var/db/postgres0/070424/u02/zne78 16386

Очистим pg_wal в нашем распакованным PG_DATA, а затем заполним его имеющимися WAL-файлами из backups/wal следующими командами:
rm -rf 070424/u22/pg1651/pg_wal
cp -r backups/wal/ 070424/u22/pg1651/pg_wal (внимательно со слешами)

Создадим recovery.signal:
touch 070424/u22/pg1651/recovery.signal

Также поставим настройку восстановления в postgresql.conf:
restore_command = 'cp ~/backups/wal/%f "%p"'

Заблокируем все подключения к БД на время восстановления, добавив строчки сверху:

```
#type database    user   address       method
host   all         all    all           reject

#type database    user   method
local  all         all    reject
```

Запустим сервер:
pg_ctl -D 070424/u22/pg1651 start

После успешного восстановления (recovery.signal удалён автоматически) восстановим возможность подключения к БД, удалив добавленные строчки.

Потом проверим состояние нашей БД:

```
postgres0=# \d
           Список отношений
 Схема  |  Имя  |   Тип   | Владелец
--------+-------+---------+-----------
 public | dobro | таблица | postgres0
 public | inf   | таблица | postgres0
(2 строки)

postgres0=# SELECT * dobro;
ERROR:  syntax error at or near "dobro"
СТРОКА 1: SELECT * dobro;
                   ^
postgres0=# SELECT * FROM dobro;
 id | dobro_amount | dobro_reciever_id
----+--------------+-------------------
  1 |            2 |                 3
  2 |            3 |                 1
  3 |            1 |                 2
  4 |            5 |                 1
  5 |            6 |                 7
(5 строк)

postgres0=# select * FROM inf;
 id | another_id | some_other_id
----+------------+---------------
(0 строк)
```

Этап 3. Повреждение файлов БД

Симулируем сбой, удаляя pg_wal:
rm -rf u22/pg1651/pg_wal

Проверим БД, подключимся к ней:

```
[postgres0@pg165 ~]$ psql -p 9165
psql (14.2)
Введите "help", чтобы получить справку.

postgres0=# \d
           Список отношений
 Схема  |  Имя  |   Тип   | Владелец
--------+-------+---------+----------
 public | dobro | таблица | postgres0
 public | inf   | таблица | postgres0
(2 строки)

postgres0=# SELECT * FROM dobro;
 id | dobro_amount | dobro_reciever_id
----+--------------+-------------------
  1 |            2 |                 3
  2 |            3 |                 1
  3 |            1 |                 2
  4 |            5 |                 1
  5 |            6 |                 7
(5 строк)
```

Подключение вышло, SELECT также работает.
Перезапустим БД

```
[postgres0@pg165 ~]$ pg_ctl start -D u22/pg1651
ожидание запуска сервера....2024-04-09 17:19:48.347 MSK [23132] LOG:  redirecting log output to logging collector process
2024-04-09 17:19:48.347 MSK [23132] HINT:  Future log output will appear in directory "log".
 прекращение ожидания
pg_ctl: не удалось запустить сервер
Изучите протокол выполнения.
[postgres0@pg165 ~]$
```

Это произошло из-за того, что при запуске БД он читает файлы из pg_wal, а в этом случае оно отсутствует. Во время работы же некоторые файлы находятся в оперативной памяти
Посмотрим лог выполнения, процесс падает с ошибкой:
"required WAL directory ""pg_wal"" does not exist"

Можно также проверить, что будет, если не удалять pg_wal целиком, а лишь очистить содержимое. Тогда процесс тоже падает, но уже с другой ошибкой:
PANIC XX000 "could not locate a valid checkpoint record"
Что означает, что Postgres не может поднятся, так как не нашёл ни одной записи о чекпоинте.

Повторим все действия по восстановлению из прошлого этапа, но за некоторым исключением:

- Переименуем u02 в u03
- WAL скопируем из резервного узла: scp -r postgres0@pg189:~/backups/wal ./070424/u22/pg1651/pg_wal

- Restore command не меняем, так как мы её уже корректно заполнили до этого

Запустим БД:
pg_ctl -D 070424/u22/pg1651 start

Проверим работоспобность:

```
[postgres0@pg165 ~]$ psql -p 9165
psql (14.2)
Введите "help", чтобы получить справку.

postgres0=# \d
           Список отношений
 Схема  |  Имя  |   Тип   | Владелец
--------+-------+---------+-----------
 public | dobro | таблица | postgres0
 public | inf   | таблица | postgres0
(2 строки)

postgres0=# SELECT * FROM dobro;
 id | dobro_amount | dobro_reciever_id
----+--------------+-------------------
  1 |            2 |                 3
  2 |            3 |                 1
  3 |            1 |                 2
  4 |            5 |                 1
  5 |            6 |                 7
(5 строк)
```

Все данные, которые были сохранены, были восстановлены.

Этап 4. Логическое повреждение данных

Добавим две строчки в таблицу dobro:

```
postgres0=# INSERT INTO dobro (id, dobro_amount, dobro_reciever_id)
VALUES (6, 666, 3), (7, 993, 7);
INSERT 0 2
postgres0=# SELECT * FROM dobro;
 id | dobro_amount | dobro_reciever_id
----+--------------+-------------------
  1 |            2 |                 3
  2 |            3 |                 1
  3 |            1 |                 2
  4 |            5 |                 1
  6 |          666 |                 3
  7 |          993 |                 7
(6 строк)

postgres0=# INSERT INTO dobro (id, dobro_amount, dobro_reciever_id)
VALUES (5, 6, 7);
INSERT 0 1
postgres0=# SELECT * FROM dobro;
 id | dobro_amount | dobro_reciever_id
----+--------------+-------------------
  1 |            2 |                 3
  2 |            3 |                 1
  3 |            1 |                 2
  4 |            5 |                 1
  6 |          666 |                 3
  7 |          993 |                 7
  5 |            6 |                 7
(7 строк)
```

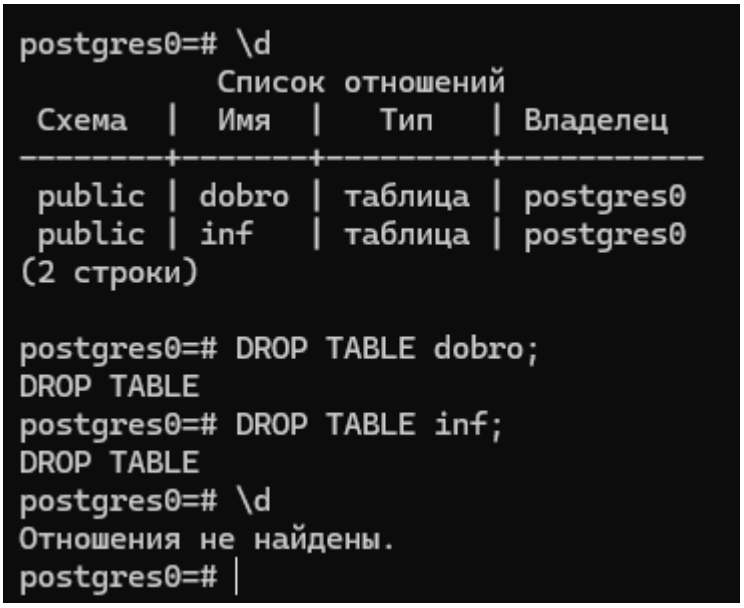
А в пустую таблицу inf добавим три строчки:

```
postgres0=# \d
          Список отношений
 Схема  |  Имя  |   Тип   | Владелец
--------+-------+---------+-----------
 public | dobro | таблица | postgres0
 public | inf   | таблица | postgres0
(2 строки)

postgres0=# SELECT * FROM inf;
 id | another_id | some_other_id
----+------------+---------------
(0 строк)

postgres0=# INSERT INTO inf (id, another_id, some_other_id)
postgres0-# VALUES
postgres0-# (0, 2, 3),
postgres0-# (1, 5, 6);
INSERT 0 2
postgres0=# SELECT * FROM inf;
 id | another_id | some_other_id
----+------------+---------------
  0 |          2 |             3
  1 |          5 |             6
(2 строки)
```

Фиксируем время (необходимо делать это средствами Postgres для фиксации транзакции):

```
postgres0=# SELECT now();
              now
-------------------------------
 2024-04-10 03:51:23.031138+03
(1 строка)
```

Удалим наши таблицы:



Поднимем резервный узел как в прошлых этапах, но при этом изменим настройку восстановления:

```
recovery_target_time='2024-04-10 03:51:23.031138+03'
recovery_target_inclusive=off # before recovery target
```

И сделаем дамп памяти:

```
pg_dump -p 9165 -Fc > postgres0.dump
```

Перенесём дамп памяти на основной узел:

```
scp postgres0.dump postgres0@pg165:~/
```

Применим этот файл и посмотрим результат:

```
pg_restore -p 9165 -d postgres0 postgres0.dump
```

```
[postgres0@pg165 ~]$ pg_restore -p 9165 -d postgres0 postgres0.dump
[postgres0@pg165 ~]$ psql -p 9165
psql (14.2)
Введите "help", чтобы получить справку.

postgres0=# \d
           Список отношений
 Схема  |  Имя  |   Тип   | Владелец
--------+-------+---------+----------
 public | dobro | таблица | postgres0
 public | inf   | таблица | postgres0
(2 строки)

postgres0=# SELECT * FROM dobro;
 id | dobro_amount | dobro_reciever_id
----+--------------+-------------------
  1 |            2 |                 3
  2 |            3 |                 1
  3 |            1 |                 2
  4 |            5 |                 1
  6 |          666 |                 3
  7 |          993 |                 7
  5 |            6 |                 7
(7 строк)

postgres0=# SELECT * FROM inf;
 id | another_id | some_other_id
----+------------+---------------
  0 |          2 |             3
  1 |          5 |             6
(2 строки)

postgres0=#
```

Можно видеть, что все данные до происшествия были восстановлены.

Вывод

В результате выполнения данной работы была успешно настроена процедура периодического резервного копирования базы данных, созданной в лабораторной работе №2, а также разработаны и отлажены сценарии восстановления для обеспечения надежности и безопасности системы. Это позволило обеспечить сохранность данных и снизить риски потери информации, а также гарантировать возможность быстрого восстановления в случае сбоев, что повышает надежность и доступность приложения.

Доп:

Вывод был:

```
[postgres0@pg165 ~/backups/test/u22/pg1651/pg_tblspc]$ ls -l
total 1
lrwx------  1 postgres0  postgres  42 10 апр.  11:58 16386 -> /var/db/postgres0/backups/070424/u02/zne78
```

ln -s /var/db/postgres0/backups/test/u02/zne78/
mv 16386
ls -l

```
[postgres0@pg165 ~/backups/test/u22/pg1651]$ ls -l pg_tblspc/
total 1
lrwxr-xr-x  1 postgres0  postgres  41 10 апр.  12:14 16386 -> /var/db/postgres0/backups/test/u02/zne78/
```

Отредактируем pg_hba

```
host    all     all     all     reject
local   all     all             reject
```

```
[postgres0@pg165 ~/backups/test/u22/pg1651]$ ls
1                   base                pg_dynshmem         pg_multixact        pg_stat             PG_VERSION          recovery.signal
2                   current_logfiles    pg_hba.conf         pg_notify           pg_stat_tmp         pg_wal
backup_label        global              pg_ident.conf       pg_replslot         pg_subtrans         pg_xact
backup_label.old    log                 pg_logical          pg_serial           pg_tblspc           postgresql.auto.conf
backup_manifest     pg_commit_ts        pg_logs.csv         pg_snapshots        pg_twophase         postgresql.conf
[postgres0@pg165 ~/backups/test/u22/pg1651]$
```

Копируем сохраненные WAL файлы:
rm -rf pg_wal
cp -r ~/backups/wal pg_wal

Восстанавливаем работу:

pg_ctl -D 070424/u22/pg1651 start