Университет ИТМО
Факультет программной инженерии и компьютерной техники

Лабораторная работа №3

по дисциплине «Распределенные системы хранения данных»

Выполнил:

Студент группы P3331

Дворкин Борис Александрович

Вариант: 98126

Преподаватель:
Николаев Владимир Вячеславович

г. Санкт-Петербург

2024 г.

Содержание

Описание задания

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.

Способ подключения к узлам из сети Интернет через helios:
1. Основной узел: `ssh -J s368090@se.ifmo.ru:2222 postgres0@pg106`
2. Резервный узел: `ssh -J s368090@se.ifmo.ru:2222 postgres1@pg115`

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: `200МБ`.
  - Средний объем измененных данных за сутки: `200МБ`.
- 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:
    - в любой таблице с внешними ключами подменить значения ключей на случайные (INSERT, UPDATE)
- Продемонстрировать результат.
- Выполнить восстановление данных указанным способом.
- Продемонстрировать и проанализировать результат.

Выполнение

Этап 1. Настройка резервного копирования

Концепции резервного копирования PostgreSQL

PostgreSQL использует две ключевые технологии для надежного резервного копирования:

1. **WAL (Write-Ahead Log)** - журнал предварительной записи, который содержит все изменения данных в хронологическом порядке
2. **pg_basebackup** - утилита для создания полных резервных копий базы данных

Для эффективного резервного копирования я настрою:

- Архивирование WAL-журналов (непрерывное)
- Периодические полные резервные копии (еженедельные)

Шаг 1. Включение архивирования WAL:

```
# Подключение к основному узлу
ssh -J s368090@se.ifmo.ru:2222 postgres0@pg106

# Редактирование postgresql.conf
ee /var/db/postgres0/jno88/postgresql.conf

# postgresql.conf:
wal_level = replica   # Минимальный уровень для архивирования
archive_mode = on     # Включаем архивирование
archive_command = 'scp %p postgres1@pg115:~/wal_archive/%f'
# затем запускаем сервер
pg_ctl -D $HOME/jno88 -l $HOME/jno88/server.log start
# или перезапускаем:
pg_ctl -D $HOME/jno88 -l $HOME/jno88/server.log restart
```

Создаем директории для архивирования на обоих узлах:

```
# На основном узле
```

```
mkdir -p ~/wal_archive

# На резервном узле
ssh -J s368090@se.ifmo.ru:2222 postgres1@pg115
# или:
ssh postgres1@pg115
mkdir -p ~/wal_archive
```

Шаг 2. Настройка SSH без пароля

Для автоматического копирования нам нужна аутентификация без пароля между узлами:

```
# На основном узле
ssh-keygen -t rsa
ssh-copy-id -i ~/.ssh/id_rsa.pub postgres1@pg115
```

Шаг 3. Скрипт для полного резервного копирования

```
ee ~/pg_backup_full.sh
```

Скрипт:

```
#!/bin/bash

# Set variables with absolute paths
DATE=$(date +%Y-%m-%d)
BACKUP_DIR="/var/db/postgres0/base_backups/$DATE"
REMOTE_HOST="postgres1@pg115"
REMOTE_BACKUP_DIR="/var/db/postgres1/base_backups"

# Create backup directories if they don't exist
mkdir -p /var/db/postgres0/base_backups
ssh $REMOTE_HOST "mkdir -p $REMOTE_BACKUP_DIR"

# Remove existing backup directory if it exists
if [ -d "$BACKUP_DIR" ]; then
      rm -rf "$BACKUP_DIR"
fi
```

```
# Create full backup
echo "Starting backup to $BACKUP_DIR"
pg_basebackup -D "$BACKUP_DIR" -U postgres0 -p 9523 -h pg106 -Ft -z
-P

# Copy backup to remote server
echo "Copying backup to remote server"
scp -r "$BACKUP_DIR" "$REMOTE_HOST:$REMOTE_BACKUP_DIR/"

# Clean up old backups on local server (older than 7 days)
echo "Cleaning up old backups on local server"
find /var/db/postgres0/base_backups -type d -mtime +7 -exec rm -rf {}
\; 2>/dev/null || true

# Clean up old backups on remote server (older than 28 days)
echo "Cleaning up old backups on remote server"
ssh $REMOTE_HOST "find $REMOTE_BACKUP_DIR -type d -mtime +28 -exec rm
-rf {} \; 2>/dev/null || true"

echo "Backup completed successfully"
```

Делаем его исполняемым:

```
chmod +x ~/pg_backup_full.sh
```

Шаг 4. Добавляем задачу в cron:

```
crontab -e  # Открыть планировщик

# Добавить строку (каждое воскресенье в 00:00):
0 0 * * 0 ~/pg_backup_full.sh > ~/backup.log 2>&1
```

```
[postgres0@pg106 ~]$ crontab -e
crontab: no crontab for postgres0 - using an empty one
crontab: installing new crontab
[postgres0@pg106 ~]$ crontab -l
0 0 * * 0 ~/pg_backup_full.sh > ~/backup.log 2>&1
```

Шаг 5. Проверяем работу архивации:

```
# тестовая транзакция
CREATE TABLE test (id SERIAL PRIMARY KEY, data TEXT);
INSERT INTO test (data) VALUES ('Hello, WAL!');

# проверяем наличие WAL в архиве на резервном узле:
ls ~/wal_archive
# Получаем сегменты: 000000010000000000000001
```

```
[postgres0@pg106 ~]$ sh pg_backup_full.sh
Starting backup to /var/db/postgres0/base_backups/2025-04-07
ПРЕДУПРЕЖДЕНИЕ:  специальный файл "./.s.PGSQL.9523" пропускается
ПРЕДУПРЕЖДЕНИЕ:  специальный файл "./.s.PGSQL.9523" пропускается
31752/31752 КБ (100%), табличное пространство 3/3
Copying backup to remote server
backup_manifest                                100%  183KB  74.7MB/s   00:00
16428.tar.gz                                   100%  244   567.9KB/s   00:00
16409.tar.gz                                   100%  501     1.6MB/s   00:00
base.tar.gz                                    100% 4178KB 107.6MB/s   00:00
pg_wal.tar.gz                                  100%   17KB  40.8MB/s   00:00
Cleaning up old backups on local server
Cleaning up old backups on remote server
Backup completed successfully
[postgres0@pg106 ~]$ ssh postgres1@pg115
Last login: Mon Apr  7 18:18:01 2025 from 192.168.11.106
[postgres1@pg115 ~]$ ls
base_backups    wal_archive
[postgres1@pg115 ~]$ ls base_backups/
2025-04-07
[postgres1@pg115 ~]$ ls war
ls: war: No such file or directory
[postgres1@pg115 ~]$ ls wal_archive/
000000010000000000000001
000000010000000000000002
000000010000000000000003
000000010000000000000004
000000010000000000000005
000000010000000000000005.00000028.backup
```

Шаг 6. Расчет объема резервных копий:

Условия:

- Новые данные: 200 МБ/сутки
- Измененные данные: 200 МБ/сутки

Расчет:

- **Полные копии**: 1 копия в неделю. Размер каждой копии = (200 МБ + 200 МБ) × 7 дней = 2800 МБ. За месяц (4 недели) → 4 × 2800 МБ = **11.2 ГБ**.
- **WAL-архивы**: 400 МБ/день × 30 дней = **12 ГБ**.
- **Итого**: 11.2 ГБ + 12 ГБ = **23.2 ГБ** (без сжатия).

Этап 2. Потеря основного узла

Цель: восстановить работу на резервном узле

Останавливаем работу СУБД на основном узле

```
 pg_ctl -D $HOME/jno88 -l $HOME/jno88/server.log stop
```

Выбор и распаковка последней резервной копии:

```
# Создаем директорию для восстановления данных на резервном узле
mkdir -p /var/db/postgres1/data_recovered

# Находим последнюю полную резервную копию (не только WAL файлы)
LATEST_BACKUP_DIR=$(find /var/db/postgres1/base_backups -type d | 
sort | tail -1)
echo "Используем директорию резервной копии: $LATEST_BACKUP_DIR"

# Извлекаем все tar-архивы из этой директории
for tarfile in $LATEST_BACKUP_DIR/*.tar.gz; do
     if [ -f "$tarfile" ]; then
     echo "Upkacking $tarfile..."
     tar -xzf "$tarfile" -C /var/db/postgres1/data_recovered
     fi
done
```

```
# Если там есть несжатые файлы/директории, копируем их тоже
if [ -d "$LATEST_BACKUP_DIR/base" ] || [ -d
"$LATEST_BACKUP_DIR/global" ]; then
     echo "Copying uncompressed files..."
     cp -r $LATEST_BACKUP_DIR/* /var/db/postgres1/data_recovered/
fi
```

```
Используем директорию резервной копии: /var/db/postgres1/base_backups/2025-04-07
Распаковываем /var/db/postgres1/base_backups/2025-04-07/16409.tar.gz...
Распаковываем /var/db/postgres1/base_backups/2025-04-07/16428.tar.gz...
Распаковываем /var/db/postgres1/base_backups/2025-04-07/base.tar.gz...
Распаковываем /var/db/postgres1/base_backups/2025-04-07/pg_wal.tar.gz...
[postgres1@pg115 ~]$
```

Настроим PostgreSQL на запуск из восстановленных данных:

```
# Создаем файл recovery.signal
touch /var/db/postgres1/data_recovered/recovery.signal

# Настраиваем конфигурацию postgresql.conf
# Вам может потребоваться отредактировать существующий файл вместо
создания нового
cat > /var/db/postgres1/data_recovered/postgresql.conf << EOF
# Основные настройки
listen_addresses = '*'
port = 5433

# Настройки восстановления
restore_command = 'cp /var/db/postgres1/wal_archive/%f %p'
```

Важный моментик: В PostgreSQL табличные пространства реализованы через символические ссылки в директории `pg_tblspc`. Когда я делаю резервную копию с помощью `pg_basebackup`, происходит следующее:

1. Инструмент копирует содержимое каталога данных, включая директорию `pg_tblspc`

2. При копировании символические ссылки в `pg_tblspc` превращаются в обычные директории
3. Это нарушает структуру, которая нужна PostgreSQL для доступа к табличным пространствам

Восстановим структуру табличных пространств:

```
# Создаем директории для табличных пространств
mkdir -p /var/db/postgres1/tablespaces/16428/PG_16_202307071
mkdir -p /var/db/postgres1/tablespaces/16409/PG_16_202307071

# Удаляем существующие директории (если они есть)
rm -rf /var/db/postgres1/data_recovered/pg_tblspc/16428
rm -rf /var/db/postgres1/data_recovered/pg_tblspc/16409

# Создаем символические ссылки
ln -s /var/db/postgres1/tablespaces/16428
/var/db/postgres1/data_recovered/pg_tblspc/16428
ln -s /var/db/postgres1/tablespaces/16409
/var/db/postgres1/data_recovered/pg_tblspc/16409

# Устанавливаем правильные права доступа
chmod 700 /var/db/postgres1/tablespaces/16428
chmod 700 /var/db/postgres1/tablespaces/16409

# Запускаем сервер
pg_ctl -D /var/db/postgres1/data_recovered -o "-p 5433" start

# Результат:
[postgres1@pg115 ~]$ pg_ctl -D /var/db/postgres1/data_recovered -o "-p 5433" start
ожидание запуска сервера....2025-04-07 15:57:30.928 GMT [60293] СООБЩЕНИЕ:  запускается PostgreSQL 16
4 on amd64-portbld-freebsd14.1, compiled by FreeBSD clang version 18.1.6 (https://github.com/llvm/llvm
-project.git llvmorg-18.1.6-0-g1118c2e05e67), 64-bit
2025-04-07 15:57:30.928 GMT [60293] СООБЩЕНИЕ:  для приёма подключений по адресу IPv6 "::" открыт порт
 5433
2025-04-07 15:57:30.928 GMT [60293] СООБЩЕНИЕ:  для приёма подключений по адресу IPv4 "0.0.0.0" открыт
 порт 5433
2025-04-07 15:57:30.952 GMT [60293] СООБЩЕНИЕ:  для приёма подключений открыт Unix-сокет "/tmp/.s.PGSQ
L.5433"
2025-04-07 15:57:30.979 GMT [60296] СООБЩЕНИЕ:  система БД была выключена: 2025-04-07 15:57:27 GMT
2025-04-07 15:57:30.990 GMT [60293] СООБЩЕНИЕ:  система БД готова принимать подключения
 готово
сервер запущен
```

```
[postgres1@pg115 ~/data_recovered]$ psql -h pg115 -p 5433 -U postgres0 postgres
psql (16.4)
Введите "help", чтобы получить справку.

postgres=# \d
                    Список отношений
 Схема  |        Имя         |        Тип         | Владелец
--------+--------------------+--------------------+-----------
 public | test               | таблица            | postgres0
 public | test1table         | таблица            | postgres0
 public | test1table_id_seq  | последовательность | postgres0
 public | test_id_seq        | последовательность | postgres0
(4 строки)
```

Этап 3. Повреждение файлов бд

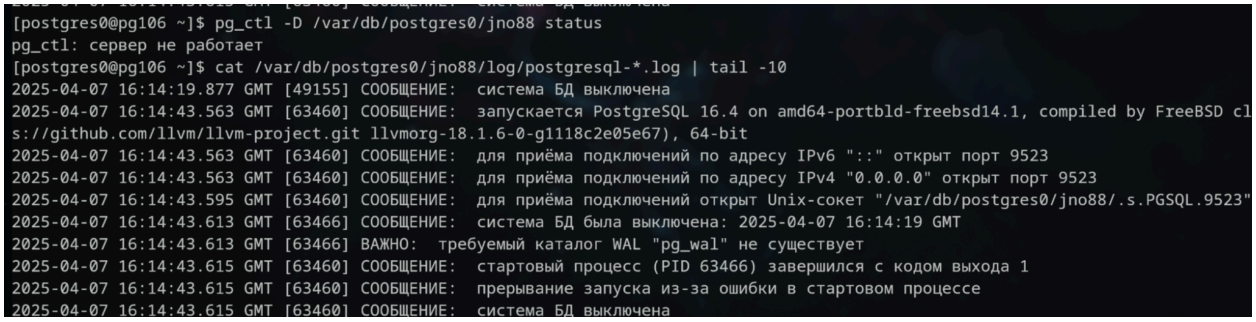
Симуляция сбоя основного узла:

```
# Остановка PostgreSQL
pg_ctl -D ~/jno88 stop

# Удаление директории WAL
rm -rf ~/jno88/pg_wal

# Попытка запуска PostgreSQL
pg_ctl -D ~/jno88 start
```

```
[postgres0@pg106 ~]$ pg_ctl -D /var/db/postgres0/gqj51 stop
pg_ctl: каталог "/var/db/postgres0/gqj51" не существует
[postgres0@pg106 ~]$ pg_ctl -D /var/db/postgres0/jno88 stop
ожидание завершения работы сервера.... готово
сервер остановлен
[postgres0@pg106 ~]$ rm -rf /var/db/postgres0/jno88/pg_wal
[postgres0@pg106 ~]$ pg_ctl -D /var/db/postgres0/jno88 start
ожидание запуска сервера....2025-04-07 16:14:43.563 GMT [63460]
2025-04-07 16:14:43.563 GMT [63460] ПОДСКАЗКА:  В дальнейшем пр
 прекращение ожидания
pg_ctl: не удалось запустить сервер
Изучите протокол выполнения.
[postgres0@pg106 ~]$
```

Проверка последствий сбоя:

```
# Проверка статуса PostgreSQL
pg_ctl -D ~/jno88 status

# Просмотр журналов ошибок
cat ~/jno88/log/postgresql-*.log | tail -10
```



Скрипт восстановленя данных из резервной копии:

```
echo "Restoring physical..."

mkdir -p ~/jno88_restored
chmod 0700 ~/jno88_restored
mkdir -p ~/new_tablespaces/16409
mkdir -p ~/new_tablespaces/16428
chmod -R 0700 ~/new_tablespaces

LATEST_BACKUP=$(ls -1d /var/db/postgres0/base_backups/* | tail -1)
echo "Using backup: $LATEST_BACKUP"

# Извлеку резервную копию
echo "Unpacking archives..."
for tarfile in $LATEST_BACKUP/*.tar.gz; do
  tar -xzf "$tarfile" -C ~/jno88_restored/
done

# Создам директорию pg_wal если её еще нет
if [ ! -d ~/jno88_restored/pg_wal ]; then
  mkdir -p ~/jno88_restored/pg_wal
  chmod 0700 ~/jno88_restored/pg_wal
```

```
fi

echo "Configuring tablespaces dir..."
mkdir -p ~/jno88_restored/pg_tblspc chmod 0700
~/jno88_restored/pg_tblspc

# Настрою новые расположения табличных пространств
rm -rf ~/jno88_restored/pg_tblspc/16409
rm -rf ~/jno88_restored/pg_tblspc/16428
ln -s ~/new_tablespaces/16409 ~/jno88_restored/pg_tblspc/16409
ln -s ~/new_tablespaces/16428 ~/jno88_restored/pg_tblspc/16428

# Настрою восстановление в postgresql.conf
echo "recovery_target_timeline = 'latest'" >>
~/jno88_restored/postgresql.conf
echo "restore_command = 'cp /var/db/postgres0/wal_archive/%f %p'" >>
~/jno88_restored/postgresql.conf

# Создам файл recovery.signal для запуска восстановления
touch ~/jno88_restored/recovery.signal
chmod 0600 ~/jno88_restored/recovery.signal

# Запущу PostgreSQL из восстановленных данных
pg_ctl -D ~/jno88_restored -o "-p 5433" start

# даю время секунд на запуск
sleep 10

# Проверю работу и доступ к данным
psql -h pg106 -p 5433 -U postgres0 postgres -c "SELECT count(*) FROM
test1table;"
```

Проверим результат восстановления:

```
[postgres0@pg106 ~]$ rm -rf jno88_restored/
[postgres0@pg106 ~]$ pg_ctl -D ~/jno88_restored -o "-p 5433" start
pg_ctl: каталог "/var/db/postgres0/jno88_restored" не существует
[postgres0@pg106 ~]$ sh restore.sh
Using backup: /var/db/postgres0/base_backups/2025-04-07
Unpacking archives...
Configuring tablespaces dir...
ожидание запуска сервера....2025-04-21 03:40:42.494 GMT [22724] СООБЩЕНИЕ:  передача вывода в протоко
л процессу сбора протоколов
2025-04-21 03:40:42.494 GMT [22724] ПОДСКАЗКА:  В дальнейшем протоколы будут выводиться в каталог "lo
g".
... готово
сервер запущен
Пароль пользователя postgres0:
 id |    data
----+-------------
  1 | Hello, WAL!
(1 строка)

[postgres0@pg106 ~]$
```

Этап 4. Логическое повреждение данных

Шаг 1. Добавим тестовые данные в таблицы:

```
psql -h pg106 -p 5433 -U postgres0 postgres -c "INSERT INTO test
(data) VALUES ('New record 1'), ('New record 2'), ('New record 3');"
psql -h pg106 -p 5433 -U postgres0 postgres -c "SELECT * FROM test;"
```

```
[postgres0@pg106 ~]$ psql -h pg106 -p 5433 -U postgres0 postgres -c "INSERT INTO test (data) VALUES ('New record
 1'), ('New record 2'), ('New record 3');"
psql -h pg106 -p 5433 -U postgres0 postgres -c "SELECT * FROM test;"
Пароль пользователя postgres0:
INSERT 0 3
Пароль пользователя postgres0:
 id |     data
----+--------------
  1 | Hello, WAL!
  2 | New record 1
  3 | New record 2
  4 | New record 3
(4 строки)

[postgres0@pg106 ~]$
```

Шаг 2. Создадим логический дамп таблицы:

```
ssh postgres1@pg115
ssh postgres0@pg106 "pg_dump -h pg106 -p 5433 -U postgres0 -t test
```

```
postgres > /tmp/test_dump.sql"
```

Шаг 3. Зафиксируем время и симулируем ошибку:

```
echo "Curr time:"
date
echo "Simulating logical data corruption..."
psql -h pg106 -p 5433 -U postgres0 postgres -c "UPDATE test SET id =
999 WHERE id = 1;"
```

```
Curr time:
понедельник, 21 апреля 2025 г. 07:09:38 (MSK)
Simulating logical data corruption...
Пароль пользователя postgres0:
UPDATE 1
[postgres0@pg106 ~]$
```

Шаг 3. Проверим результат повреждения:

```
psql -h pg106 -p 5433 -U postgres0 postgres -c "SELECT * FROM test;"
```

```
[postgres0@pg106 ~]$ psql -h pg106 -p 5433 -U postgres0 postgres -c "SELECT * FROM test;"
Пароль пользователя postgres0:
 id  |     data
-----+-------------
   2 | New record 1
   3 | New record 2
   4 | New record 3
 999 | Hello, WAL!
(4 строки)
```

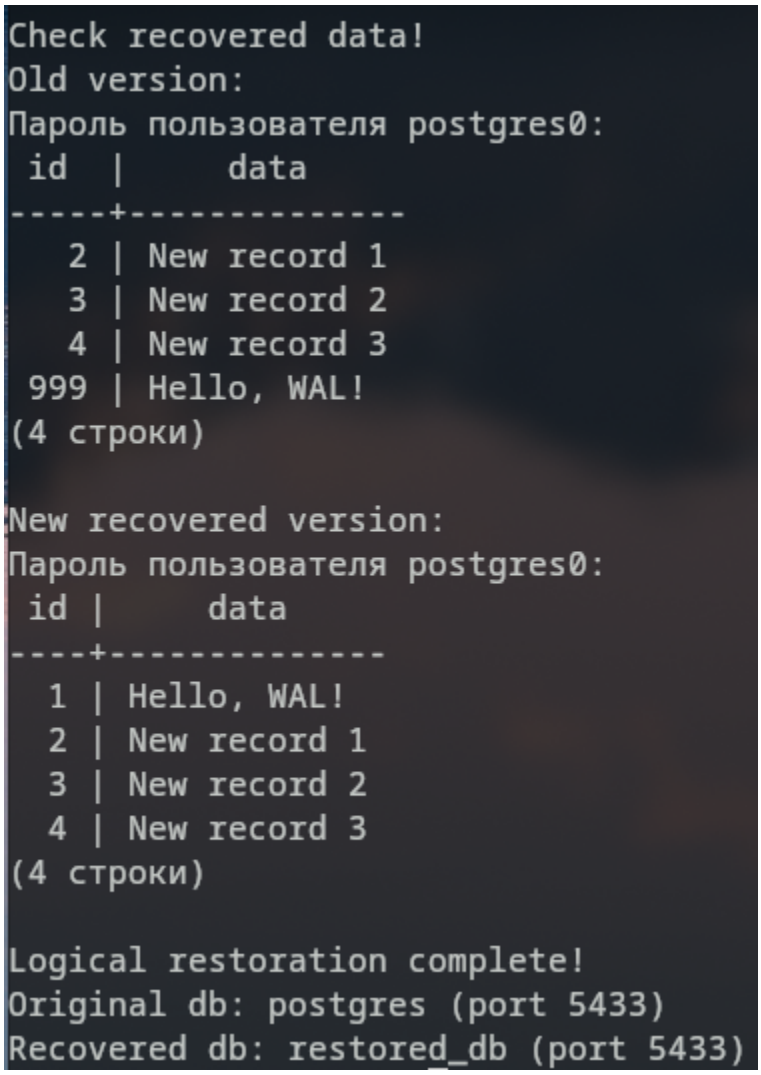
Шаг 4. Восстановим данные из резервного узла:

Скрипт logic_restore.sh:

```
#!/bin/bash
echo "Restoring logical..."

# 3. Создание временной базы для восстановления
echo "Creating tmp db..."
psql -h pg106 -p 5433 -U postgres0 postgres -c "DROP DATABASE IF
```

```
EXISTS restored_db;"
psql -h pg106 -p 5433 -U postgres0 postgres -c "CREATE DATABASE
restored_db;"

# 4. Восстановление данных из дампа
echo "Recovering..."
psql -h pg106 -p 5433 -U postgres0 restored_db -f /tmp/test_dump.sql

# 5. Проверка результата восстановления
echo "Check recovered data!"
echo "Old version:"
psql -h pg106 -p 5433 -U postgres0 postgres -c "SELECT * FROM test;"
echo "New recovered version:"
psql -h pg106 -p 5433 -U postgres0 restored_db -c "SELECT * FROM
test;"

echo "Logical restoration complete!"
echo "Original db: postgres (port 5433)"
echo "Recovered db: restored_db (port 5433)"
```

Шаг 5. Проверим результат восстановления:



Вывод

Настройка непрерывного архивирования и восстановления PostgreSQL

Задача казалась простой - настроить бэкапы, делать их раз в неделю, архивировать WAL, потом симулировать пару разных катастроф и успешно восстановиться. (Он не понимал что его ждёт дальше)

Этап 1: Резервное копирование

Для начала, я сделал обычный скрипт и запихнул его в cron. Нюанс был в том, что скрипт изначально был кривой, но это отличное упражнение в bash скриптинге, где можно и задавать переменные и по scp перекидывать файлы и редактировать уже существующие или добавлять им строчки.

Пока я это делал, я совсем забыл что надо добавить непрерывное копирование WAL файлов. И я его добавил, но сам об этом забыл и через две недели открыв лабу сам себя запутал. Как именно - будет ниже, пока что этот этап я сделал корректно.

Этап 2: Потеря основного узла

Гениальная симуляция: "Представь, что основной узел пропал". Казалось бы, что может пойти не так - а всё просто, И тут начались приключения. Сначала ошибка: "не удалось открыть каталог pg_tblspc/16409/PG_16_202307071". Потом PostgreSQL стал запускаться, но падал при восстановлении с ошибкой "не удалось создать каталог pg_tblspc/16409".

Корень всех бед - табличные пространства и их проклятые символические ссылки. Оказывается, PostgreSQL может быть очень капризным по поводу своих симлинков.

После двух часов магических пассов и размахивания `rm -rf` сервер всё-таки заработал. Гора мелких ошибок научила главному - если что-то не получается, начни с самого простого.

Затем, я просто сам себя закопал - в какой-то момент я настолько много раз пытался всё это завести, что я переписывал скрипты и тестировал новые подходы, и в этом процессе насоздавал несколько бдшек, и когда показывал преподавателю работу - то на основном узле изменял одну бд, а на резервном показывал другую)) Вот и вся сложность этого этапа - надо быть предельно осторожным, внимательным и максимально декомпозировать все шаги каждого этапа, вместо того чтобы сразу пытаться писать скрипт и потом сто лет его фиксить.

Этап3-4:

На данных этапах проблем особо не возникло, после всей возни на предыдущих я хорошо задокументировал весь процесс и всё аккуратно сделал. Ещё из прошлого этапа я помнил трюк - если новые wal файлы не создаются, можно сделать SELECT pg_switch_wal(); Поэтому таким образом я создал ещё один wal файл, удостоверился что он создался, и удалил все wal файлы, чтобы симулировать сбой. А дальше дело техники - как на предыдущем этапе восстанавливаем всё что нужно, кладём в PGDArecovery.signal - пустой файлик который сигнализирует постгресу что надо запуститься в режиме восстановления, найти настройки в конфигурации restore_command, применить wal записи для восстановления данных и удалить этот сигнальный файл после успешного восстановления.

На этапе 4 был сделан pg_dump, используешь -f для указания filename и туда даёшь сгенерированный дамп. Для понимания, там лежит просто набор команд нужных для восстановления. Я так неоднократно делал на работе, поскольку моей задачей было синхронизировать 5 различных баз контуров (продакшн, тестинг, и т.п.), а потом для них сделать миграционную структуру, где можно просто ормкой после изменения создавать миграции и удобно их отказывать/накатывать.

В заключение хочу сказать, что может я и мазохист, было весело 🤭 Столько различных навыков пересекается, а по клавиатуре бить одно удовольствие. В будущем, конечно, буду создавать побольше элиасов. То, что мне лень было собирать fish shell было огромной ошибкой, без автодополнения очень тяжело писать длинные команды подключения - в целом, это одна из основных причин всех возникших проблем.