Университет ИТМО
Факультет программной инженерии и компьютерной техники

Лабораторная работа №2

по дисциплине «Распределенные системы хранения данных»

Выполнил:

Студент группы P3331

Дворкин Борис Александрович

Вариант: 5523

Преподаватель:
Николаев Владимир Вячеславович

г. Санкт-Петербург

2024 г.

Содержание

Описание задания

Цель работы - на выделенном узле создать и сконфигурировать новый кластер БД Postgres, саму БД, табличные пространства и новую роль, а также произвести наполнение базы в соответствии с заданием. Отчёт по работе должен содержать все команды по настройке, скрипты, а также измененные строки конфигурационных файлов.

Способ подключения к узлу из сети Интернет через helios:
`ssh -J sXXXXXX@helios.cs.ifmo.ru:2222 postgresY@pgZZZ`
Способ подключения к узлу из сети факультета:
`ssh postgresY@pgZZZ`
Номер выделенного узла `pgZZZ`, а также логин и пароль для подключения Вам выдаст преподаватель.

Этап 1. Инициализация кластера БД

- Директория кластера: `$HOME/jno88`
- Кодировка: KOI8-R
- Локаль: русская
- Параметры инициализации задать через аргументы команды

Этап 2. Конфигурация и запуск сервера БД

- Способы подключения: 1) Unix-domain сокет в режиме peer; 2) сокет TCP/IP, принимать подключения к любому IP-адресу узла
- Номер порта: `9523`
- Способ аутентификации TCP/IP клиентов: по паролю SHA-256
- Остальные способы подключений запретить.
- Настроить следующие параметры сервера БД:
    - max_connections
    - shared_buffers
    - temp_buffers
    - work_mem
    - checkpoint_timeout
    - effective_cache_size
    - fsync
    - commit_delay
- Параметры должны быть подобраны в соответствии со сценарием OLAP:
  8 одновременных пользователей, пакетная запись/чтение данных по 192МБ.

- Директория WAL файлов: `$PGDATA/pg_wal`
- Формат лог-файлов: `.log`
- Уровень сообщений лога: `WARNING`
- Дополнительно логировать: попытки подключения и завершение сессий

Этап 3. Дополнительные табличные пространства и наполнение базы

- Создать новое табличное пространство для индексов: `$HOME/myj85`
- На основе template1 создать новую базу: `wetredmom`
- Создать новую роль, предоставить необходимые права, разрешить подключение к базе.
- От имени новой роли (не администратора) произвести наполнение ВСЕХ созданных баз тестовыми наборами данных. ВСЕ табличные пространства должны использоваться по назначению.
- Вывести список всех табличных пространств кластера и содержащиеся в них объекты.

Выполнение

Этап 0. Расчёт объёма ОЗУ

1. **Общая память для пользовательских операций:**
    - 8 пользователей × 192 МБ = 1536 МБ
2. **Распределение памяти в PostgreSQL для OLAP:** Оптимальное распределение памяти согласно докам предполагает, что:
    - **work_mem:** ~30% от общей памяти для всех соединений
    - **shared_buffers:** ~25-35% от общей памяти
    - **Системные нужды:** ~20-25% от общей памяти
    - **Другие операции PostgreSQL:** ~15-20% от общей памяти
3. **Расчет общего объема ОЗУ:** Если 1536 МБ (для работы всех пользователей) составляет ~30% общей памяти, то:
    - Общая память ≈ 1536 МБ ÷ 0.3 ≈ 5120 МБ ≈ 5 ГБ

Таким образом, идеальная конфигурация для заданных условий предполагает наличие примерно 5-6 ГБ ОЗУ.

Этап 1. Инициализация кластера БД

Сначала создадим директорию для кластера:

```
mkdir -p $HOME/jno88
```

Инициализируем кластер PostgreSQL с указанными параметрами:

```
initdb --encoding=KOI8-R --locale=ru_RU.KOI8-R
--lc-messages=ru_RU.KOI8-R \
  --lc-monetary=ru_RU.KOI8-R --lc-numeric=ru_RU.KOI8-R
--lc-time=ru_RU.KOI8-R \
  -D $HOME/jno88
```

**Директория WAL файлов по умолчанию создается как `$HOME/jno88/pg_wal`, где `$HOME/jno88` — это директория кластера.

Этап 2. Конфигурация и запуск сервера БД

Настройка параметров подключения

Отредактируем файл `$HOME/jno88/postgresql.conf`:

```
# Редактирование файла конфигурации
cat > $HOME/jno88/postgresql.conf << EOF
# Основные параметры подключения
listen_addresses = '*'  # Принимать подключения с любого IP-адреса
port = 9523             # Номер порта
unix_socket_directories = '$HOME/jno88'  # Директория Unix-сокета
password_encryption = 'scram-sha-256' # Аутефикация по sha-256

# OLAP (8 пользователей, 192МБ пакетная запись/чтение)
max_connections = 20 # 8 пользователей, запас для администрирования
shared_buffers = 1536MB # Общий буфер для кэширования страниц
данных (25% от ОЗУ для OLAP)
temp_buffers = 160MB    # (10% work mem) для сложных запросов
work_mem = 192MB    # для операций сортировки (8 users × 192MB)
checkpoint_timeout = 15min  # big интервал для контрольных точек
effective_cache_size = 3GB  # 50% от доступной памяти
fsync = on             # Гарантия целостности данных
commit_delay = 50 # Задержка для группировки коммитов (50 мкс.)
# Суть задержки в том, чтоб повысить пропускную способность
нагруженных OLAP сценариев с 1000-ми транзакций. Группируем
транзакции и уменьшаем количество fsync().

# Директория WAL файлов
wal_level = replica # minimal + point-in-time recovery, позволяет
создавать точные физические реплики бд (без logical - логической
репликации)
max_wal_size = 1GB
min_wal_size = 80MB

# Настройки журналирования
log_destination = 'stderr'
logging_collector = on
log_directory = 'log'
```

```
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'
log_file_mode = 0600
log_min_messages = warning      # Уровень сообщений: WARNING
log_connections = on            # Логирование попыток подключения
log_disconnections = on         # Логирование завершения сессий
EOF
```

Методы аутентификации сами по себе определяются в файле **pg_hba.conf**, а не в **postgresql.conf**. Можно задать через аргументы к initdb, но мне проще так, ибо неоднократно так делал на работе и в проектах:

```
cat > $HOME/jno88/pg_hba.conf << EOF
# TYPE  DATABASE      USER            ADDRESS                 METHOD
# Unix domain socket соединения (peer аутентификация)
local   all           all                                     peer

# Подключения по IPv4 с любого адреса (аутентификация по паролю
SHA-256)
host  all             all             0.0.0.0/0
scram-sha-256
EOF
```

Запуск сервера PostgreSQL

Запустим сервер PostgreSQL:

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

Установка метода аутентификации по умолчанию

```
psql -h pg106 -p 9523 -U postgres0 postgres -c "ALTER SYSTEM SET
password_encryption = 'scram-sha-256';"
pg_ctl -D $HOME/jno88 restart
```

Проверяем возможные подключения:

```
# Подключение через Unix-сокет
psql -h /var/db/postgres0/jno88 -p 9523 postgres

# Подключение через TCP/IP (IPv4)
psql -h 127.0.0.1 -p 9523 postgres

# Удалённое подключение с гелиоса
psql -h pg106 -p 9523 -U postgres0 postgres
```

**Для того чтобы sha-256 работал, надо задать пароль для postgres0 пользователя:

```
# Подключаемся через Unix-сокет (не требует пароля)
psql -h /var/db/postgres0/jno88 -p 9523 postgres

# Внутри psql устанавливаем пароль для пользователя postgres
ALTER USER postgres0 PASSWORD '1234';
```

Этап 3. Дополнительные табличные пространства и наполнение базы

Создание табличного пространства для индексов

```
# Создание директории для табличного пространства индексов
mkdir -p $HOME/myj85
chmod 700 $HOME/myj85

# Создание табличного пространства
psql -h pg106 -p 9523 -U postgres0 postgres -c "CREATE TABLESPACE
index_space LOCATION '$HOME/myj85';"

# Создание новой базы данных
psql -h pg106 -p 9523 -U postgres0 postgres -c "CREATE DATABASE
wetredmom TEMPLATE template1;"
```

```
# Создание новой роли с необходимыми правами
psql -h pg106 -p 9523 -U postgres0 postgres -c "CREATE ROLE data_user
WITH LOGIN PASSWORD 'data123';"
psql -h pg106 -p 9523 -U postgres0 postgres -c "GRANT CONNECT ON
DATABASE wetredmom TO data_user;"
psql -h pg106 -p 9523 -U postgres0 postgres -c "GRANT CONNECT ON
DATABASE template1 TO data_user;"
psql -h pg106 -p 9523 -U postgres0 wetredmom -c "GRANT CREATE ON
SCHEMA public TO data_user;"
psql -h pg106 -p 9523 -U postgres0 template1 -c "GRANT CREATE ON
SCHEMA public TO data_user;"
psql -h pg106 -p 9523 -U postgres0 postgres -c "GRANT CREATE ON
TABLESPACE index_space TO data_user;"
```

Создание новой роли и назначение прав

```
# Создание новой роли с возможностью подключения и паролем
psql -h 127.0.0.1 -p 9523 postgres -c "CREATE ROLE data_user WITH
LOGIN PASSWORD 'secure_password';"

# Предоставление прав на базу данных
psql -h pg106 -p 9523 -U postgres0 postgres -c "GRANT CONNECT ON
DATABASE wetredmom TO data_user;"
psql -h pg106 -p 9523 -U postgres0 wetredmom -c "GRANT CREATE ON
SCHEMA public TO data_user;"
psql -h pg106 -p 9523 -U postgres0 postgres -c "GRANT CREATE ON
TABLESPACE index_space TO data_user;"
```

Наполнение базы данных тестовыми данными

Создадим скрипт для наполнения wetredmom:

```
cat > $HOME/populate_wetredmom.sql << EOF
-- Простая таблица в основном табличном пространстве (pg_default)
CREATE TABLE test_data (
     id SERIAL PRIMARY KEY,
     name TEXT,
     value INTEGER
```

```
);

-- Индекс в специальном табличном пространстве для индексов
CREATE INDEX idx_test_data_name ON test_data(name) TABLESPACE
index_space;

-- Минимальное наполнение данными
INSERT INTO test_data (name, value) VALUES
('тест1', 100),
('тест2', 200),
('тест3', 300);
EOF

# Выполнение скрипта для базы wetredmom от имени data_user
PGPASSWORD=data123 psql -h 127.0.0.1 -p 9523 -U data_user -d
wetredmom -f $HOME/populate_wetredmom.sql
```

Создадим скрипт для наполнения template1:

```
cat > $HOME/populate_template1.sql << EOF
-- Простая таблица в основном табличном пространстве (pg_default)
CREATE TABLE template_sample (
      id SERIAL PRIMARY KEY,
      info TEXT
);

-- Индекс в специальном табличном пространстве для индексов
CREATE INDEX idx_template_sample_info ON template_sample(info)
TABLESPACE index_space;

-- Минимальное наполнение
INSERT INTO template_sample (info) VALUES
('образец1'),
('образец2');
EOF

# Выполнение скрипта для базы template1 от имени data_user
PGPASSWORD=data123 psql -h 127.0.0.1 -p 9523 -U data_user -d
```

```
template1 -f $HOME/populate_template1.sql
```

Проверки:

```
# Проверка логов
cat $HOME/jno88/log/$(ls -t $HOME/jno88/log | head -1)

# Проверка конфигурации WAL
psql -h 127.0.0.1 -p 9523 postgres -c "SHOW data_directory;"
psql -h 127.0.0.1 -p 9523 postgres -c "SHOW log_filename;"

# Проверка аутентификации
psql -h 127.0.0.1 -p 9523 -U data_user wetredmom -c "SELECT
current_user;"

# Проверка табличных пространств
psql -h 127.0.0.1 -p 9523 postgres -c "\db+"

# Проверка созданных таблиц и индексов
psql -h 127.0.0.1 -p 9523 wetredmom -c "\dt+"
psql -h 127.0.0.1 -p 9523 wetredmom -c "\di+"
```

Вывод информации о табличных пространствах:

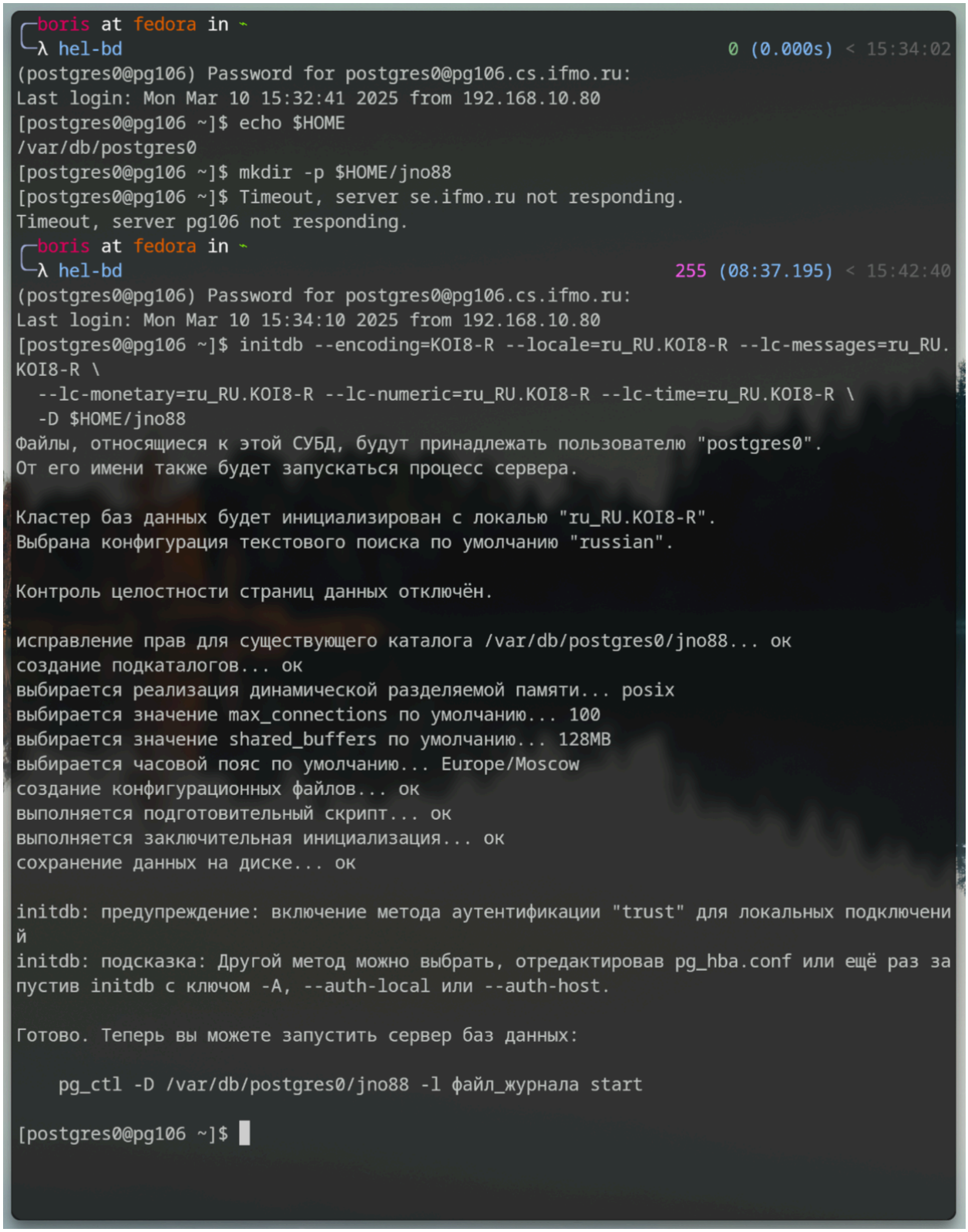
```
┌──s368090 at helios in ~
└─λ  cat table_spaces.sql         0 (11.726s) < 16:53:42

with ts_info as (
     select oid, spcname
     from pg_tablespace
),
db_ts as (
     select ts.spcname as ts_name,
     db.datname as obj_name,
     'database' as obj_type,
     NULL as schema_name
     from pg_database db
     join ts_info ts on db.dattablespace = ts.oid
),
rel_obj as (
```

```
     select
     coalesce(t.spcname, 'pg_default') AS ts_name,
     c.relname as obj_name,
     case c.relkind
          when 'r' then 'table'
          when 'i' then 'index'
          when 'S' then 'sequence'
          when 'v' then 'view'
          when 'm' then 'materialized view'
          when 'c' then 'composite type'
          when 't' then 'TOAST table'
          when 'f' then 'foreign table'
          when 'p' then 'partitioned table'
          when 'I' then 'partitioned index'
          else c.relkind::text
     end as obj_type,
     n.nspname as schema_name
     from pg_class c
     join pg_namespace n on c.relnamespace = n.oid
     left join ts_info t on c.reltablespace = t.oid
     where c.relkind in ('r', 'i', 'S', 'v', 'm', 'c', 't', 'f',
'p', 'I')
),
empty_ts as (
     select
     ts.spcname AS ts_name,
     '<empty>' AS obj_name,
     NULL AS obj_type,
     NULL AS schema_name
     from ts_info ts
     where not exists (
     select 1 from db_ts where ts_name = ts.spcname
     )
     and not exists (
     select 1 from rel_obj where ts_name = ts.spcname
     )
)
select
     case
```

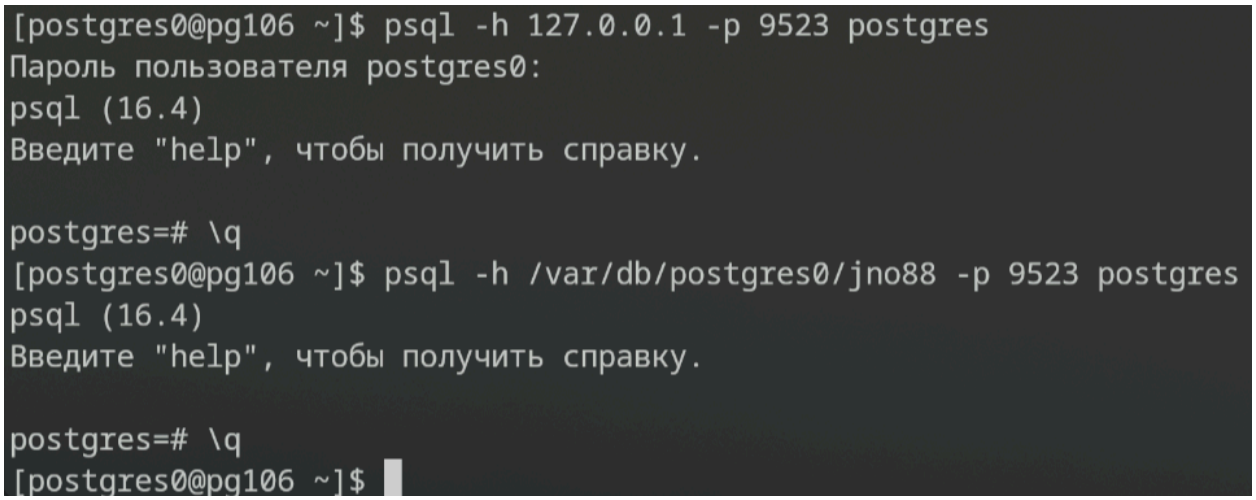
```
     when row_number() over (partition by ts_name order by obj_type,
obj_name) = 1
     then ts_name
     else NULL
     end as "tablespace",
     case
     when schema_name IS NOT NULL AND obj_name IS NOT NULL AND
obj_name != '<empty>'
          then schema_name || '.' || obj_name
     else obj_name
     end as "object",
     obj_type as "type"
from (
     select * from db_ts
     union all
     select * from rel_obj
     union all
     select * from empty_ts
) all_obj
order by ts_name, obj_type, schema_name, obj_name;
```

Демонстрация

Инициализируем кластер:



Подключаемся:



Вывод

Выполнив лабораторную работу по распределённым системам хранения данных я приобрёл бесценный жизненный опыт, а также небольшой набор профессиональных навыков.

Для начала, я открыл для себя различные способы подготовки к этим лабораторным. Не было чем-то чересчур экстраординарным, лишь сам себе напомнил в очередной раз to be humble и что несмотря на part тайм работу, экзамены в гибдд, похороны и любые другие жизненные трудности, на пару всё же лучше приходить готовым, всё что для этого нужно это чуть более заранее планировать время, даже если его очень мало, и не лениться. Всегда можно чуть больше напрячься, от этого только спать лучше будешь 😀

Затем с професиональной точки зрения. Чем дальше, тем чаще я занимаю роли тимлида в различных проектах, квинтэссенцией этого было руководство командой бэкендеров из 3х человек в реальной компании, и полное руководство командой из 8ми человек на курсе разработки мобильных приложений. Оказалось что невероятно сложно брать на себя ответственность за принятие определённых решений - разрабатывать комплексную микросервисную архитектуру, продумывать коммуникацию между сервисами и в том числе - **где и как хранить данные**. Вот тут-то и вступает предмет рсхд и вся его польза. В данной лабораторной я на примитивном уровне потыкал конфигурации кластера, настроил свой для заданного ТЗ с OLAP запросами, научился его менять БЕЗ перезагрузки сервера, что может быть очень важно в реальных системах, обосновал выбор определённых определённых параметров - таких как размер кешей, задержки между записями, конфигурации логирования, сетевые подключения и т.д., залез внутрь кластера и понял из каких директорий он состоит, и познакомился с тем как на самом деле устроен постгрес изнутри.

Это было невероятно увлекательно и теперь я ещё лучше и глубже понимаю как работать с СУБД и в частности с постгресом и из чего они состоят. В продакшене до сих пор в большинстве своём используется постгрес, и я не вижу причин от него отказываться, поэтому я действительно считаю что это крутые и полезные навыки. Знания конкретных команд? - забудутся. Понимание устройства и принципов? - **останутся!**