МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский университет ИТМО»

##### ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

## ЛАБОРАТОРНАЯ РАБОТА №2

### по дисциплине

### «РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ ХРАНЕНИЯ ДАННЫХ»

### Вариант № 798

##### ***Выполнил:*** Студент группы P3319

##### Билобрам Денис Андреевич

#### Преподаватель:

##### Шибаев Семён Сергеевич

Санкт-Петербург, 2025

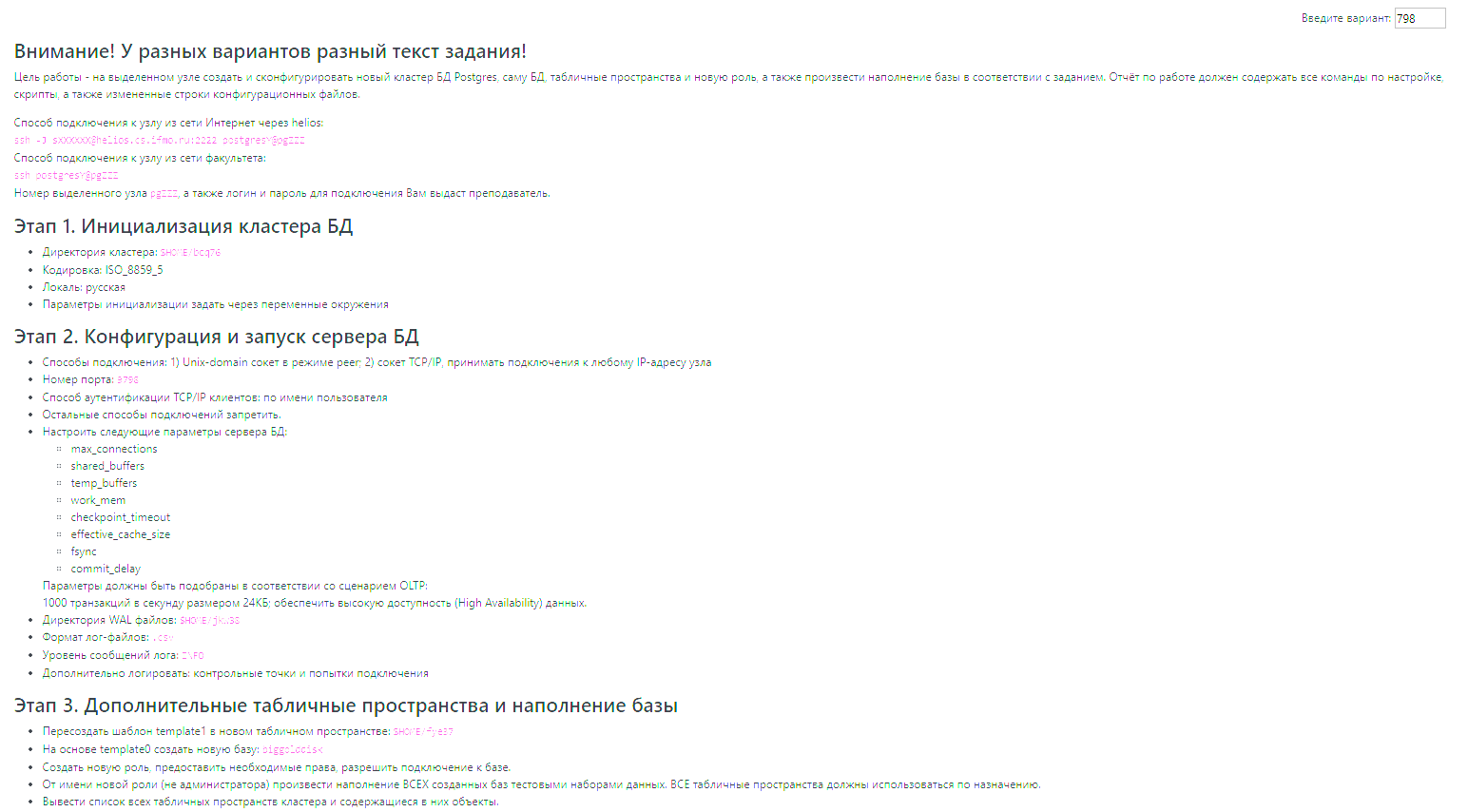
# Cодержание

[Задание: 3](#_Toc191644366)

[Выполнение: 3](#_Toc191644367)

[Вывод: 5](#_Toc191644368)

# Задание:



# Выполнение:

**Этап 1 – Инициализация кластера СУБД**

1. **Создание каталога кластера**

[postgres6@pg194 ~]$ mkdir -p "$HOME/bcq76"

1. **Экспорт переменных окружения для инициализации**

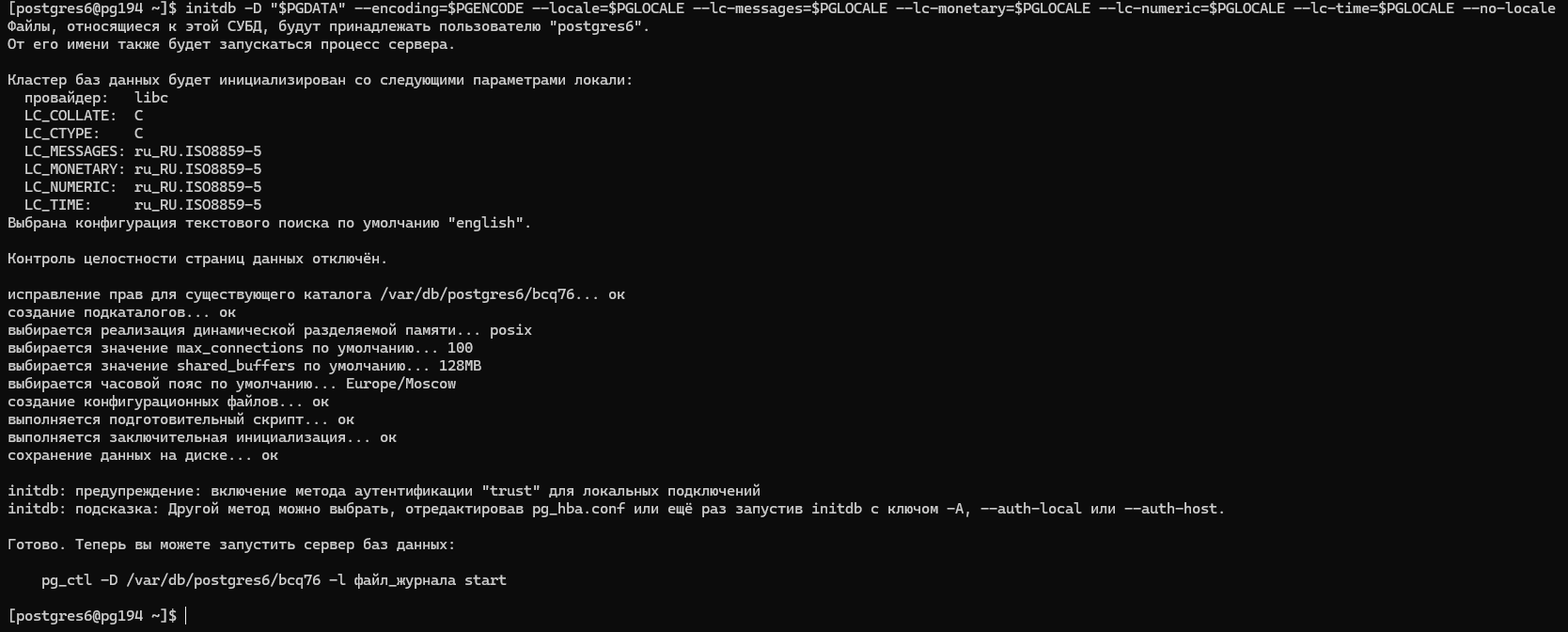
[postgres6@pg194 ~]$ export PGDATA="$HOME/bcq76"

[postgres6@pg194 ~]$ export PGLOCALE="ru\_RU.ISO8859-5"

[postgres6@pg194 ~]$ export PGENCODE="ISO8859-5"

[postgres6@pg194 ~]$ export PGWAL="$HOME/jkw38"

[postgres6@pg194 ~]$ initdb -D "$PGDATA" –waldir=$PGWAL --encoding=$PGENCODE --locale=$PGLOCALE --lc-messages=$PGLOCALE --lc-monetary=$PGLOCALE --lc-numeric=$PGLOCALE --lc-time=$PGLOCALE --no-locale



**Этап 2 – Конфигурация и запуск сервера**

1. **Настройка параметров подключения**

*# PGDATA/pg\_hba.conf*

# Разрешить локальные подключения через Unix-сокет с peer-аутентификацией

local all all peer

# Разрешить подключения по TCP/IP через имя пользователя

host all postgres6 0.0.0.0/0 trust

host all labuser 0.0.0.0/0 trust

# Запретить любые другие подключения по сети

host all all 0.0.0.0/0 reject

host all all ::/0 reject

*# PGDATA/pg\_ident.conf*

# MAPNAME SYSTEM-USERNAME PG-USERNAME

ident\_map client\_os\_user postgres6

1. **Конфигурация** **postgresql.conf для сети**

*# PGDATA/postgresql.conf*

listen\_addresses = '\* '

port = 9798

1. **Оптимизация параметров под OLTP-нагрузку**

* **max\_connections** = 200 – Максимальное число одновременных соединений. Установлено с запасом для поддержки множества конкурирующих транзакций (значение 200 – условно, подбирается исходя из ожиданий нагрузки и ресурсов сервера).
* **shared\_buffers** = 512MB – Объём памяти, выделяемый под буферизацию страниц БД. OLTP-нагрузка с 1000 TPS \* 24KB/s (~24 MB/s) выигрывает от увеличения shared\_buffers. Значение ~512 МБ позволит эффективнее кэшировать активно используемые данные (рекомендуется 20-25% от объёма ОЗУ сервера).
* **temp\_buffers** = 16MB – Объём памяти для временных таблиц, на соединение. Для интенсивных транзакций увеличиваем значение по сравнению с дефолтным (8MB), чтобы операции с временными данными (если возникнут) реже обращались к диску. 16 МБ на соединение – разумный запас для OLTP, если такие таблицы используются.
* **work\_mem** = 16MB – Объём памяти для операций сортировки и хеширования на запрос. В OLTP-запросах обычно небольшие выборки, но при 1000 TPS и возможных параллельных сложных запросах, увеличение work\_mem до 16 МБ обеспечит выполнение небольших сортировок в памяти. (Важно помнить, что этот объём выделяется на операцию сортировки для каждого соединения, поэтому чрезмерное завышение может привести к перерасходу памяти при множестве одновременных запросов).
* **effective\_cache\_size** = 2GB – Оценка доступной файловой кеш-памяти операционной системы для PostgreSQL. Этот параметр не выделяет память, а влияет на планировщик запросов. Значение ~2 ГБ подразумевает, что суммарно (shared\_buffers + системный кеш) сервер может рассчитывать на ~2 ГБ данных в памяти. Мы выбираем 2GB как пример при условии, что сервер имеет ~4GB+ ОЗУ.
* **fsync = on** – Флаг синхронизации на диск. **Обязательно оставляем включённым** для гарантированной надёжности (ACID). В режиме высокой доступности (HA) журнал предзаписи должен надёжно фиксироваться на диске при каждой транзакции. (Параметр fsync по умолчанию *on*, но мы явно подчёркиваем, что он не отключается во имя производительности, так как это недопустимо для HA).
* **commit\_delay** = 10000 – Задержка (в микросекундах) перед фиксацией транзакции для групповой записи. Значение 10000 мкс = 10 мс. В условиях 1000 TPS такая небольшая задержка позволяет сгруппировать несколько транзакций перед записью WAL, уменьшая количество отдельных flush-операций диска. Это может повысить пропускную способность без заметного влияния на время отклика, особенно когда одновременно выполняются десятки транзакций (групповое подтверждение).
* **checkpoint\_timeout** = 3min – Интервал между принудительными контрольными точками (checkpoint). Уменьшаем его с типичных 5 минут до **3 минут** для снижения времени восстановления после сбоя (более частые checkpoint уменьшают объём WAL, который нужно проиграть при восстановлении, что важно для HA). Хотя более частые checkpoint слегка увеличивают нагрузку на диск, интервал 3 мин при интенсивной записи (~24MB/s) создаст разумный баланс: за три минуты накопится до ~4-5 GB изменений, которые применятся в checkpoint. Это уменьшит время простоя на восстановление в случае сбоя по сравнению с 5-10 минутами журнала.

1. **Настройка логов**

*# PGDATA/postgresql.conf*

logging\_collector = on

log\_destination = 'csvlog'

log\_directory = 'log'

log\_filename = 'postgresql-%u.csv' # шаблон имени файлов логов (csv)

log\_rotation\_age = 1d

log\_min\_messages = info

log\_connections = on

log\_checkpoints = on

**Этап 3 – Табличные пространства и наполнение базы**

1. **Пересоздадим template1 в новом табличном пространстве**

[postgres6@pg194 ~/bcq76]$ mkdir -p "$HOME/fye37"

[postgres6@pg194 ~/bcq76]$ chmod 700 "$HOME/fye37"

[postgres6@pg194 ~/bcq76]$ psql -h 127.0.0.1 -p 9798 -d postgres -U postgres6

postgres=# CREATE TABLESPACE fye37 LOCATION '/var/db/postgres6/fye37';

CREATE TABLESPACE

postgres=# UPDATE pg\_database SET datistemplate = FALSE WHERE datname = 'template1';

UPDATE 1

postgres=# DROP DATABASE template1;

DROP DATABASE

postgres=# CREATE DATABASE template1

WITH TEMPLATE = template0

TABLESPACE = fye37

OWNER = postgres6

ALLOW\_CONNECTIONS = true;

CREATE DATABASE

postgres=# UPDATE pg\_database

SET datistemplate = TRUE

WHERE datname = 'template1';

UPDATE 1

1. **Создание базы данных biggolddisk**

postgres=# CREATE DATABASE biggolddisk

WITH TEMPLATE = template0

OWNER = postgres6

TABLESPACE = pg\_default

CONNECTION LIMIT = -1;

CREATE DATABASE

1. **Создание роли**

postgres=# \c biggolddisk postgres6

Вы подключены к базе данных "biggolddisk" как пользователь "postgres6".

biggolddisk=# GRANT CONNECT ON DATABASE biggolddisk TO labuser;

GRANT

biggolddisk =# GRANT USAGE ON SCHEMA public TO labuser;

GRANT

biggolddisk =# GRANT CREATE ON SCHEMA public TO labuser;

GRANT

biggolddisk=# \c postgres postgres6

Вы подключены к базе данных "postgres" как пользователь "postgres6".

postgres=# GRANT USAGE ON SCHEMA public TO labuser;

GRANT

postgres=# GRANT CREATE ON SCHEMA public TO labuser;

GRANT

postgres=# \c template1 postgres6

Вы подключены к базе данных "template1" как пользователь "postgres6".

template1=# GRANT USAGE ON SCHEMA public TO labuser;

GRANT

template1=# GRANT CREATE ON SCHEMA public TO labuser;

GRANT

1. **Наполнение баз данных тестовыми данными**

[postgres6@pg194 ~/bcq76]$ psql -h 127.0.0.1 -p 9798 -d postgres -U labuser

postgres=> CREATE TABLE test\_data\_pg (

id SERIAL PRIMARY KEY,

info TEXT

);

CREATE TABLE

postgres=> INSERT INTO test\_data\_pg (info) VALUES ('Hello from Postgres base');

INSERT 0 1

[postgres6@pg194 ~/bcq76]$ psql -h 127.0.0.1 -p 9798 -d biggolddisk -U labuser

biggolddisk=> CREATE TABLE test\_data\_bg (

product\_id SERIAL PRIMARY KEY,

name TEXT,

price NUMERIC(10,2)

);

CREATE TABLE

biggolddisk=> INSERT INTO test\_data\_bg (name, price) VALUES ('Gadget', 99.99);

INSERT 0 1

[postgres6@pg194 ~/bcq76]$ psql -h 127.0.0.1 -p 9798 -d template1 -U labuser

template1=> CREATE TABLE test\_data\_tpl (

note\_id SERIAL PRIMARY KEY,

note TEXT

);

INSERT INTO test\_data\_tpl (note) VALUES ('Record in template1');

CREATE TABLE

INSERT 0 1

1. **Cписок табличных пространств и таблиц в них**

template1=# SELECT spcname, pg\_tablespace\_location(oid)

FROM pg\_tablespace;

SELECT

n.nspname AS schema,

c.relname AS table\_name,

CASE

WHEN c.reltablespace = 0 THEN 'pg\_default'

ELSE t.spcname

END AS tablespace

FROM pg\_class c

JOIN pg\_namespace n ON n.oid = c.relnamespace

LEFT JOIN pg\_tablespace t ON c.reltablespace = t.oid

WHERE n.nspname = 'public'

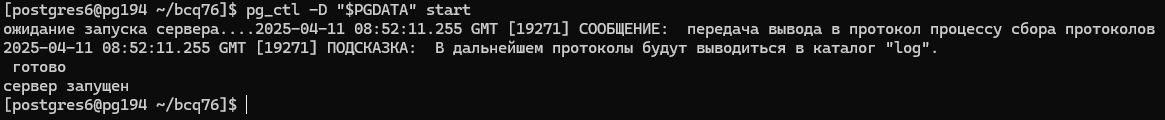
AND c.relkind = 'r'

ORDER BY table\_name;

# Результат работы:

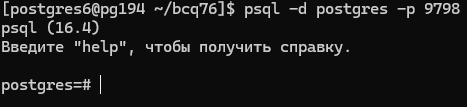
# Запуск сервера

# [postgres6@pg194 ~/bcq76]$ pg\_ctl -D "$PGDATA" start



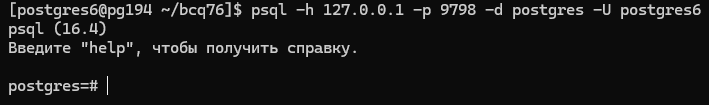
Подключение через локальный Unix-сокет в режиме peer:

[postgres6@pg194 ~/bcq76]$ psql -d postgres -p 9798

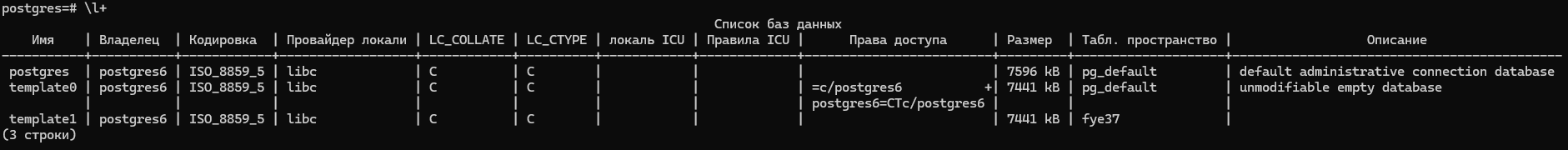


Подключение по TCP/IP через имя пользователя

[postgres6@pg194 ~/bcq76]$ psql -h 127.0.0.1 -p 9798 -d postgres -U postgres6



Пересоздание template1 в новом табличном пространстве



Таблицы с указанием табличного пространства



# Вывод:

При выполнении лабораторной работы, я использовал знания о системных каталогах PostgreSQL, продемонстрировал способы динамического извлечения метаданных из системных каталогов для получения информации о любой таблице базы данных.