МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

##### ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

**ЛАБОРАТОРНАЯ РАБОТА №2**

**по дисциплине**

**«Распределённые системы хранения данных»**

Вариант №1745

##### ***Выполнил:*** Студент группы P3334 Баянов Равиль Динарович

#### Преподаватель:

##### Николаев Владимир

##### Вячеславович

Санкт-Петербург

2025

**Содержание**

[**Задание** 3](#_Toc192433642)

[**Описание этапов выполнения** 4](#_Toc192433643)

[**Этап 1. Инициализация кластера БД** 4](#_Toc192433644)

[**Задание** 4](#_Toc192433645)

[**Этап 2. Конфигурация и запуск сервера БД** 4](#_Toc192433646)

[**Этап 3. Дополнительные табличные пространства и наполнение базы** 4](#_Toc192433647)

[**Вывод** 5](#_Toc192433648)

# **Задание**

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.

**Номер выделенного узла:** pg108

**Логин:** postgres0

**Пароль:** DLvc7uCc

# **Описание этапов выполнения**

**Этап 1. Инициализация кластера БД**

**Задание:**

* Директория кластера: $HOME/zeb22
* Кодировка: ISO\_8859\_5
* Локаль: русская
* Параметры инициализации задать через переменные окружения

**Выполнение:**

|  |
| --- |
| mkdir zeb22 # Создание директории, в которой будет располагаться кластер  # Задание переменных окружения для конфигурации кодировки и локали кластера  export PGDATA=$HOME/zeb22 # Указание директории для кластера  export LANG=ISO\_8859\_5  export LC\_COLLATE=ISO\_8859\_5  export LC\_CTYPE=ISO\_8859\_5  export LC\_ALL=ISO\_8859\_5  initdb # Инициализация кластера (все переменыне окружения применятся для этой утилиты) |

**Этап 2. Конфигурация и запуск сервера БД**

**Задание:**

* Способы подключения: 1) Unix-domain сокет в режиме peer; 2) сокет TCP/IP, принимать подключения к любому IP-адресу узла
* Номер порта: 9745
* Способ аутентификации TCP/IP клиентов: по паролю в открытом виде
* 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 OLTP:  
300 одновременных пользователей, 2 сессий на каждого; каждая сессия инициирует до 7 транзакций на запись размером 8КБ; обеспечить максимальную производительность.

* Директория WAL файлов: $PGDATA/pg\_wal
* Формат лог-файлов: .log
* Уровень сообщений лога: INFO
* Дополнительно логировать: контрольные точки и попытки подключения

**Выполнение:**

|  |
| --- |
| # Изменения в файле postgres.conf  listen\_adresses = '\*' # Принимаем подключения по любом IP-адресу  port = 9745  max\_connections = 600 # 2 \* 300 (кол-во соединений для кластера)  shared\_buffers = 1GB # Размер кэша для страниц баз данных (20-30% от RAM для OLTP)  temp\_buffers = 32MB # Размер памяти для временных таблиц одной сессии  work\_mem = 8MB # Размер памяти для различных операций в рамках одного запроса  fsync = on # Гарантия того, что данные сохранятся на диск перед завершением транзакции, теряется скорость, но в любом случае данные мы терять не хотим  commit\_delay = 1000 # 1 мс - время ожидания перед группированием операций в одну транзакцию для передачи на диск (уменьшаем число дисковых операций)  log\_destination = 'stderr' # Перенаправляем лог ошибок в стандартный поток вывода ошибок для перемещения его в файл с логами  log\_directory = 'pg\_log' # Директория, в которой будут храниться логи  log\_filename = 'postgresql-%Y-%m-%d\_%H%M%S.log' # паттерн названия файла с логами (указание даты создания)  logging\_collector = on # Включаем перенаправление потока stderr в файл  log\_file\_mode = 0740 # Указываем права для лог файлов  log\_rotation\_age = 1d # Время через которое логи очищаются и перезаписываются  log\_min\_message = info # Минимальный уровень сообщений, записывающихся в логи  log\_chekpoints = on # Логирование информации о записи данных на диск  log\_connections = on  log\_disconnection = on  # Изменения в файле pg\_hba.conf  local all all peer # Unix-domain сокет в режиме peer  host all all 0.0.0.0/0 password # пособ аутентификации TCP/IP клиентов: по паролю в открытом виде для IPv4 для всех IP-адресов  host all all ::0/0 password # для IPv6  # Всё остальное закомментим, чтобы исключить другие виды подключений |

**Логирование:**

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Этап 3. Дополнительные табличные пространства и наполнение базы**

**Задание:**

* Пересоздать шаблон template1 в новом табличном пространстве: $HOME/rez82
* На основе template0 создать новую базу: nicebluemom
* 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:**

|  |
| --- |
| mkdir rez82 # Создание директории для нового табличного пространства  # После подключения к самой БД  CREATE TABLESPACE rez82 LOCATION '/var/db/postgres0/rez82'; # Создание табличного пространства  UPDATE pg\_database  SET dattablespace = (SELECT oid FROM pg\_tablespace WHERE spcname = 'rez82')  WHERE datname = 'template1'; # Перемещение шаблона template1 в пространство rez82  CREATE DATABASE nicebluemom WITH TEMPLATE template0 TABLESPACE rez82; # Создание БД nicebluemom в пространстве rez82  CREATE ROLE test\_role WITH LOGIN PASSWORD 'test\_role'; # Создание роли  ALTER ROLE test\_role CREATEDB CREATEROLE; # Добавление возможности создания БД и ролей для новой роли  GRANT CONNECT ON DATABASE postgres TO test\_role; Разрешение подключаться к БД postgres  GRANT CONNECT ON DATABASE postgres0 TO test\_role; Разрешение подключения к postgres0  GRANT CONNECT ON DATABASE nicebluemom TO test\_role; Разрешение подключения к nicebluemom  # Разрешение манипулировать данными в БД для новой тестовой роли  GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO test\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO test\_role;  SET ROLE test\_role; # Смена роли |

Теперь создадим скрипт заполнения всех пространств кластера. Представим, что: nicebluemom – это сервис для наставничества (менторства) для студентов.

postgres - Главная база пользователей  
nicebluemom - База курсов и наставничества  
postgres0 - База транзакционных данных (оценки, регистрация, взаимодействие)

**Скрипт:**

|  |
| --- |
| -- Переключаемся на базу данных nuicebluemom  \c nicebluemom;  -- Убедитесь, что таблицы создаются в нужном табличном пространстве  **SET** default\_tablespace = 'rez82';  -- Таблица клиентов  **CREATE** **TABLE** customers (  customer\_id SERIAL **PRIMARY** KEY,  first\_name VARCHAR(50),  last\_name VARCHAR(50),  email VARCHAR(100) **UNIQUE** **NOT** **NULL**,  phone VARCHAR(15),  address TEXT  );  -- Таблица продуктов  **CREATE** **TABLE** products (  product\_id SERIAL **PRIMARY** KEY,  name VARCHAR(100) **NOT** **NULL**,  description TEXT,  price DECIMAL(10, 2) **NOT** **NULL**,  stock\_quantity INT **NOT** **NULL**  );  -- Таблица заказов  **CREATE** **TABLE** orders (  order\_id SERIAL **PRIMARY** KEY,  customer\_id INT **REFERENCES** customers(customer\_id) **ON** **DELETE** CASCADE,  order\_date TIMESTAMP **DEFAULT** CURRENT\_TIMESTAMP,  status VARCHAR(20) **CHECK** (status **IN** ('pending', 'shipped', 'delivered', 'canceled'))  );  -- Таблица позиций заказа (каждый заказ может включать несколько продуктов)  **CREATE** **TABLE** order\_items (  order\_item\_id SERIAL **PRIMARY** KEY,  order\_id INT **REFERENCES** orders(order\_id) **ON** **DELETE** CASCADE,  product\_id INT **REFERENCES** products(product\_id) **ON** **DELETE** CASCADE,  quantity INT **NOT** **NULL**,  price DECIMAL(10, 2) **NOT** **NULL**  );  -- Таблица транзакций (для отслеживания платежей за заказы)  **CREATE** **TABLE** transactions (  transaction\_id SERIAL **PRIMARY** KEY,  order\_id INT **REFERENCES** orders(order\_id) **ON** **DELETE** CASCADE,  transaction\_date TIMESTAMP **DEFAULT** CURRENT\_TIMESTAMP,  amount DECIMAL(10, 2) **NOT** **NULL**,  status VARCHAR(20) **CHECK** (status **IN** ('pending', 'completed', 'failed'))  );  -- Индексы для улучшения производительности  **CREATE** INDEX idx\_order\_customer **ON** orders(customer\_id);  **CREATE** INDEX idx\_product\_name **ON** products(name);  -- Вставка клиента  **INSERT** **INTO** customers (first\_name, last\_name, email, phone, address)  **VALUES** ('John', 'Doe', 'john.doe@example.com', '123-456-7890', '1234 Elm St, Springfield');  -- Вставка продукта  **INSERT** **INTO** products (name, description, price, stock\_quantity)  **VALUES** ('Smartphone', 'Latest model of smartphone', 599.99, 100);  -- Создание заказа  **INSERT** **INTO** orders (customer\_id, status)  **VALUES** (1, 'pending');  -- Вставка позиции заказа  **INSERT** **INTO** order\_items (order\_id, product\_id, quantity, price)  **VALUES** (1, 1, 1, 599.99);  -- Вставка транзакции  **INSERT** **INTO** transactions (order\_id, amount, status)  **VALUES** (1, 599.99, 'pending');  -- Переключаемся на базу данных postgres0  \c postgres0;  -- Таблица поставщиков  **CREATE** **TABLE** suppliers (  supplier\_id SERIAL **PRIMARY** KEY,  name VARCHAR(100) **NOT** **NULL**,  contact\_name VARCHAR(100),  contact\_phone VARCHAR(15),  address TEXT  );  -- Вставка поставщика  **INSERT** **INTO** suppliers (name, contact\_name, contact\_phone, address)  **VALUES** ('Tech Corp', 'Alice Smith', '987-654-3210', '7890 Tech Lane, Silicon Valley'); |

**Вывод все объектов:**

|  |
| --- |
| **WITH** spaced\_tables **AS** (  **SELECT**  COALESCE(t.spcname, 'pg\_default') **AS** spcname,  c.relname,  ROW\_NUMBER() **OVER** (**PARTITION** **BY** COALESCE(t.spcname, 'pg\_default') **ORDER** **BY** c.relname) **AS** rn  **FROM** pg\_tablespace t  **FULL** **JOIN** pg\_class c **ON** c.reltablespace = t.oid  **ORDER** **BY** spcname, c.relname  )  **SELECT**  **CASE** **WHEN** rn = 1 **THEN** spcname **ELSE** **NULL** **END** **AS** spcname,  relname  **FROM** spaced\_tables; |

# **Сложности**

При выполнении данной лабораторной работы возникло не мало сложностей. Первый нюанс, с которым я столкнулся – это то, что совсем неясно было какой оперативной памятью обладает узел, на котором я работаю, следовательно были проблемы с конфигурацией кластера так, чтобы подключение и логирование работали корректно и без ошибок. Приходилось подстраивать размеры кэша и разбираться детально в том, как работает поле logging\_collectors в конфигурационном файле postgres.conf. Также были сложности в наполнении кластера, так как это отняло много времени и зачастую мои скрипты не работали правильно. Также к этому не сразу удалось установить зависимость параметров кластера БД с OLTP подходом. Но все трудности были устранены.

# **Вывод**

Выполнив данную лабораторную работу, я создал и сконфигурировал свой кластер баз данных, научился настраивать базу данных для любых нужд. Узнал, как можно регулировать подключение к базе данных в зависимости от ролей, табличных пространств и настроек подключения. Также у меня получилось наполнить базу данных по технологии OLTP и теперь в моих силах настраивать кластеры PostgreSQL под любую задачу.