МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
 «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ДГТУ)

Факультет «Информатика и вычислительная техника»

Кафедра «Кибербезопасность информационных систем»

Лабораторная работа №11

по дисциплине «Основы построения защищённых баз данных»

Выполнил: обучающийся гр. ВКБ43

Ковалев Данил Петрович

Проверил:

Скляров Алексей Викторович

Тема: Реализация мандатного и дискреционного разграничение доступа в PostgreSQL

Цель: изучить и применить модели мандатного и дискреционного управления доступом в PostgreSQL.

Вариант 6. Создайте таблицу inventory с полями item\_id, product\_id,

quantity. Реализуйте мандатное управление доступом: метки restricted запрещают изменение данных пользователями с уровнем доступа ниже restricted. Настройте дискреционный доступ для роли warehouse\_role, которая может только просматривать данные.

Задание 1. Подготовка к работе. Убедитесь, что PostgreSQL установлен и настроен. Создайте тестовую базу данных для выполнения задания.

Для выполнения лабораторной работы используется PostgreSQL версии – 13.20 и Fedora 42. В целях выполнения лабораторной работы была необходимость в сборке PostgreSQL из исходного кода с выставлением определенных флагов, так как в репозиториях RPM все образы скомпилированы без флага SELinux.

Для начала обновим полностью все пакеты в операционной системе, команда и результат представлены на рисунке 1.

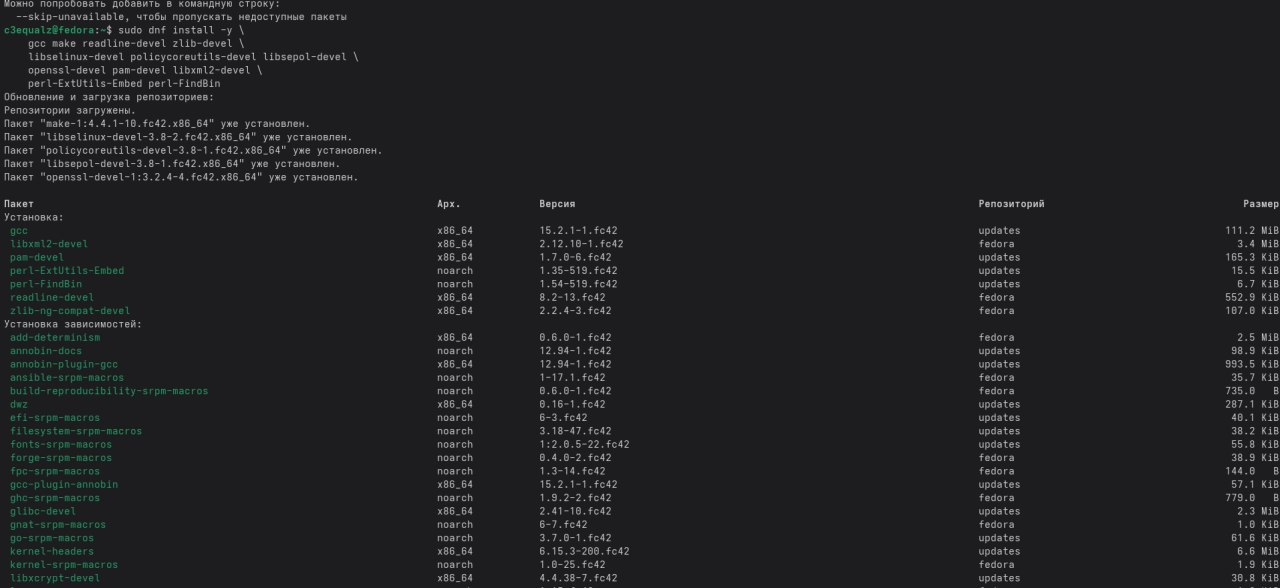


Рисунок 1 – обновление пакетов в ОС

Теперь через wget нам нужно скачать пакет с официального сайта PostgreSQL. В исходниках находится tar архив, который нужно распаковать и открыть. Для выполнения данных действий представлены команды на рисунке 2.

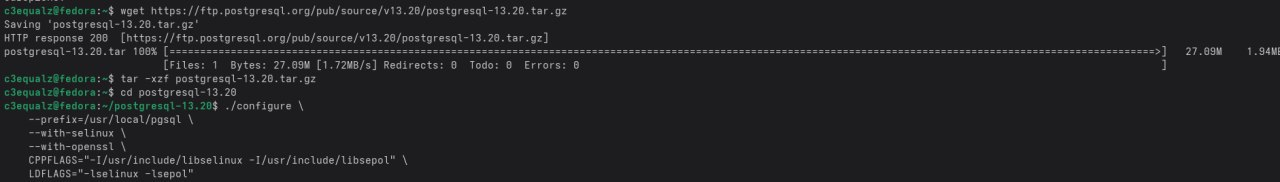


Рисунок 2 – скачка PostgreSQL с официального сайта

Произведем конфигурацию сборки с SELinux и соберем пакет, который у нас будет использоваться для установки. Команды представлены на рисунке 3.



Рисунок 3 – сборка пакета

Установим наш собранный пакет командой, которая представлена на рисунке 4.



Рисунок 4 – установка пакета

Теперь нам нужно собрать sepgsql, для этого в начале перейдем в директорию cd contrib/sepgsql, а после этого с помощью системы сборки make установим. Команды и результат представлен на рисунке 5.

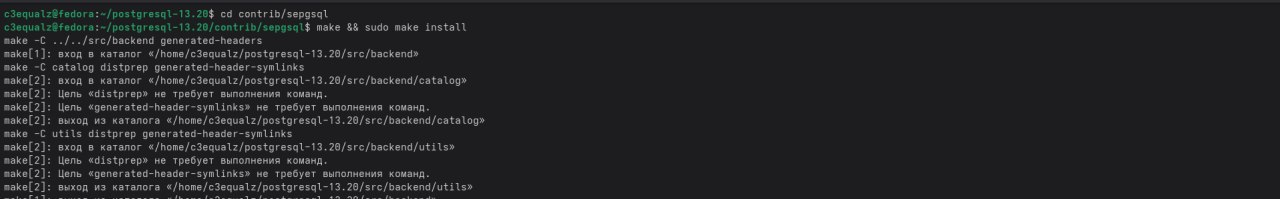


Рисунок 5 – сборка sepgsql

Теперь файлы, которые были связаны с sepgsql переносим в extensions, указывая метаданные для сборки. Для этого используем команду, которая представлена на рисунке 6.



Рисунок 6 – метаданные для sepgsql

Добавляем путь к бинарным файлам в переменную окружения PATH, чтобы можно было пользоваться psql без явного указания всего пути. Для этого нужно выполнить команды, которые представлены на рисунке 7.

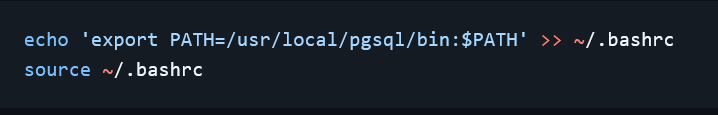


Рисунок 7 – указание переменных окружения для psql

Теперь создаем пользователя postgres и инициализируем основную базу данных с метаданными. Минимальная настройка для первого запуска представлена на рисунке 8.

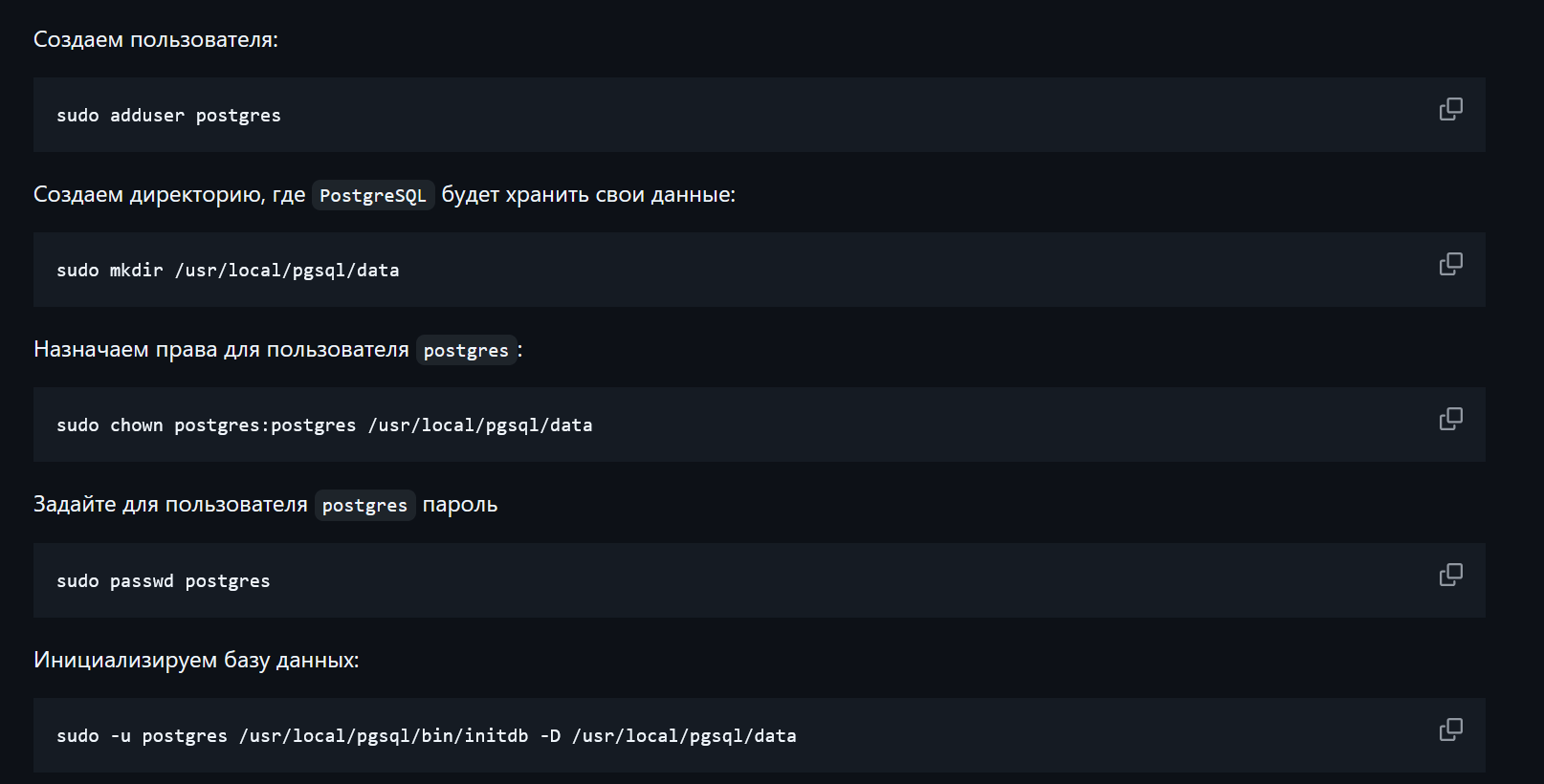


Рисунок 8 – инициализация СУБД

Теперь укажем, что в shared\_preload\_libraries используется sepgsql. Команда для этого представлена на рисунке 9.



Рисунок 9 – конфигурация postgresql.conf

Теперь перейдем к созданию Unit, чтобы systemd мог запускать процесс и оперировать им. Для этого создадим service файл для запуска: sudo nano /etc/systemd/system/postgresql.service. Данные, которые содержит данный файл представлен на рисунке 10.

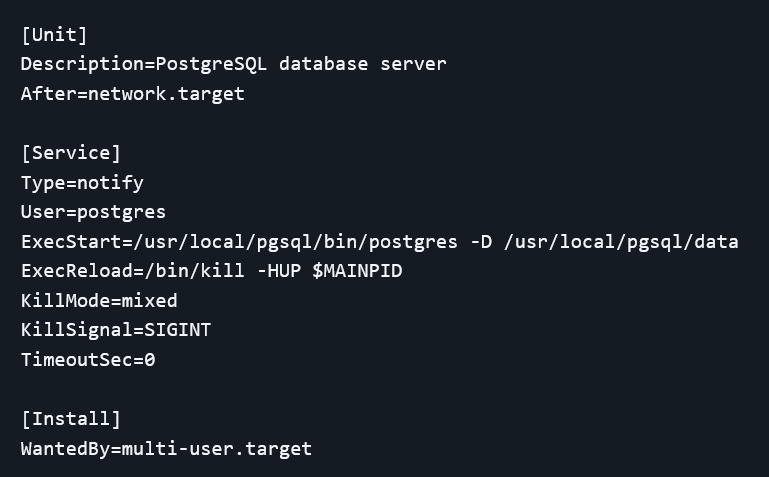


Рисунок 10 – конфигурация postgresql.service

Теперь перезагрузим демона, чтобы он увидел наш новый service файл, который будет работать. Команда представлена на рисунке 11.



Рисунок 11 – перезагрузка демона

Включим теперь процесс PostgreSQL, используя команду на рисунке 12.



Рисунок 12 – включение демона для postgresql

Задание 2. Выбор варианта задания. Получите свой вариант задания из таблицы вариантов (см. таблицу ниже). Ознакомьтесь с требованиями своего варианта и выполните соответствующие шаги.

Перед созданием всех объектов для базы данных нам нужно выключить SELinux, используя команду, которая представлена на рисунке 13.

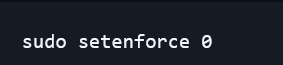


Рисунок 13 – выключение SELinux

Теперь создадим базу данных для выполнения лабораторной работы. Команда представлена на рисунке 14.

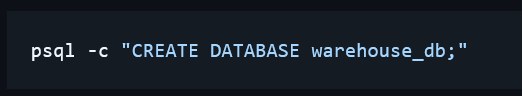


Рисунок 14 – создание объекта базы данных

У sepgsql есть требование, что нужно применять расширение к схеме. В документации данный момент не указывается, было выявлено опытным путем. Для создания схемы используется команда, которая представлена на рисунке 15.

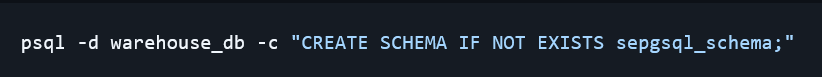


Рисунок 15 – создание схемы

Теперь создадим расширение в схеме, используя команду, которая представлена на рисунке 16.

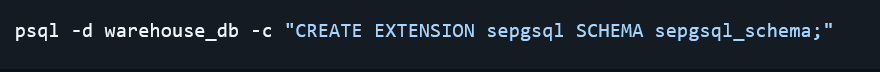


Рисунок 16 – создание расширения sepgsql для схемы

Теперь после первоначальных настроек можно вернуть обратно SELinux в Enforce режим. Для этого используется команда, которая представлена на рисунке 17.



Рисунок 17 – переключение SELinux в Enforce режим

Теперь посмотрим по метаданным в базе данных, что расширение установлено. Для этого воспользуемся командами, которые представлены на рисунке 18. На рисунке видно, что расширения удачно установились и применились. Пример из документации с прогоном sql файлов для всех баз данных в моем случае не работал.

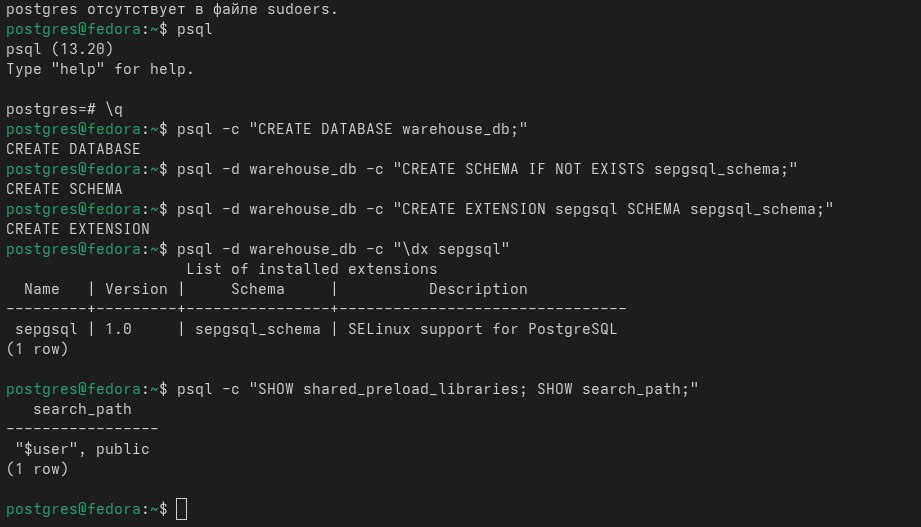
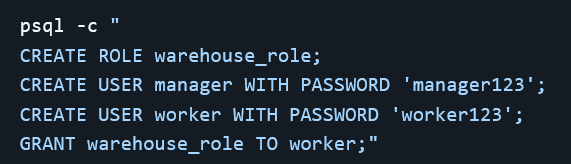


Рисунок 18 – просмотр предустановленных библиотек

Задание 3. Создание пользователей и ролей. Создайте пользователей и роли, указанные в вашем варианте задания. Назначьте роли пользователям.

Создадим теперь роль и пользователей для DAC. Для этого используется команды, представленные на рисунке 19. Для моего задания используются пользователи – manager и worker.

Рисунок 19 – роль и пользователи для DAC

Задание 4. Реализация мандатного управления доступом. Настройте метки безопасности для объектов базы данных в соответствии с вашим вариантом. Используйте расширение sepgsql.

Навесим теперь временные лейблы на столбцы в базе данных. Они являются временными по той причине, чтобы мы проверили корректность работы DAC, позже будет изменено значение. После этого удостоверимся, что лейблы корректно применились. Команды представлены на рисунке 20.

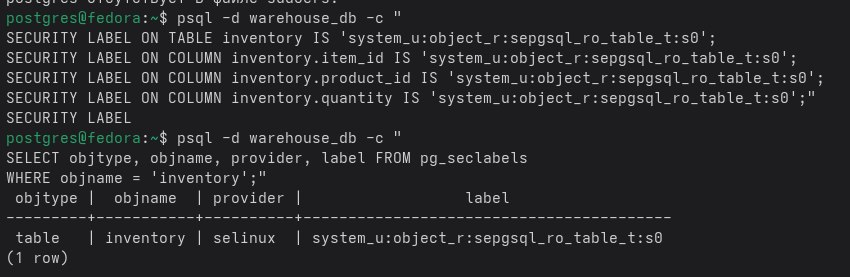


Рисунок 20 – установка временных лейблов и просмотр их применения

Задание 5. Реализация дискреционного управления доступом. Настройте права доступа для пользователей и ролей с использованием команд GRANT и REVOKE в соответствии с вашим вариантом. Проверьте, что права доступа соответствуют требованиям задания.

Выдаем теперь права чтение на роль warehouse\_role, для этого используется команды, которые представлены на рисунке 21.



Рисунок 21 – выдача прав для роли

Теперь выдадим права на изменения данных для manager. Команда представлена на рисунке 22.

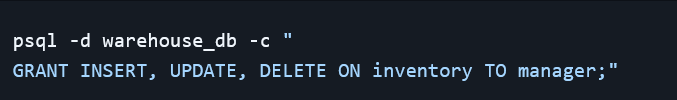


Рисунок 22 – выдача прав для менеджера

Посмотрим теперь какие есть права за какого пользователя и что за доступы. Результат представлен на рисунке 23.

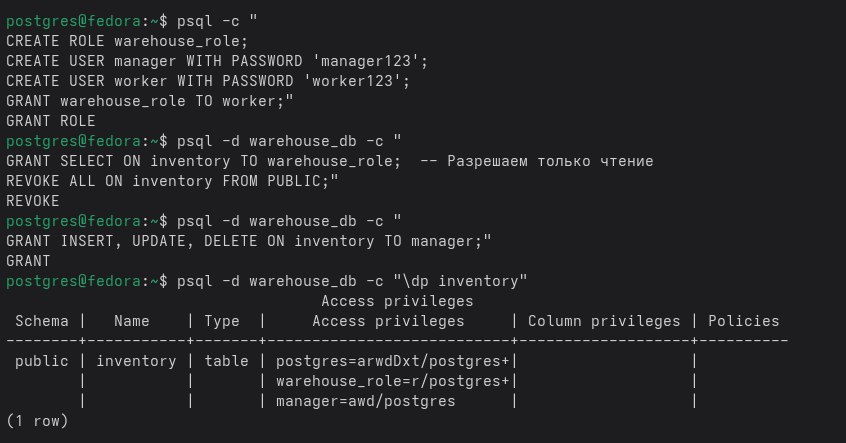


Рисунок 23 – права на таблицу

Остается выдать права на SEQUENCE, чтобы менеджер точно мог заполнять данные. Команда представлена на рисунке 24.



Рисунок 24 – выдача прав для менеджера

Задание 6. Протестировать все настройки.

Проведем проверку, что DAC полностью работает и корректно все настроено. Для этого за пользователя worker попробуем просмотреть и вставить данные, то же самое повторим и для пользователя с ролью warehouse role. Результат представлен на рисунке 25.

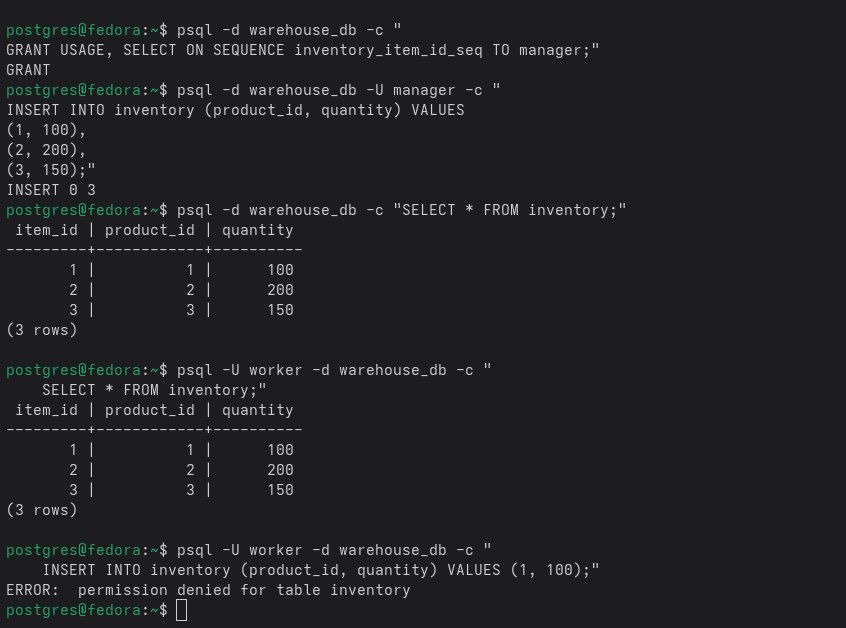


Рисунок 25 – проверка корректности DAC

Теперь перейдем к тестированию MAC, для этого поменяем LABELS, добавив больше прав в виде s0:c20, как указано в мануалах sepgsql на Github. Код для изменения лейблов представлен на рисунке 26.

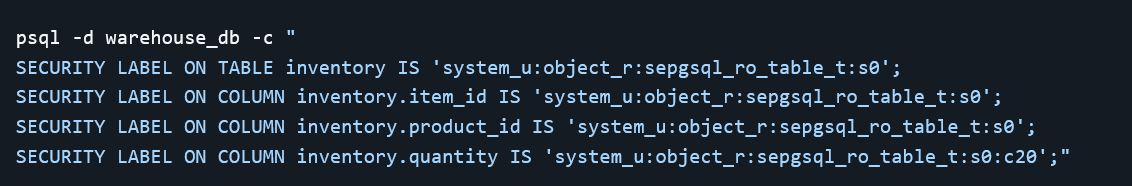


Рисунок 26 – изменение лейблов для тестирования MAC

Теперь попробуем вставить данные, указав и не указав атрибут c20. В результате получим такой результат, который представлена на рисунке 27.

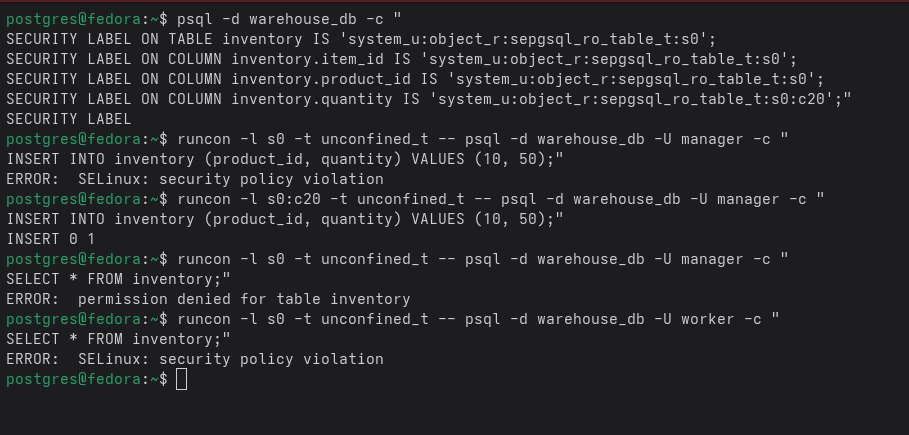


Рисунок 27 – проверка работоспособности MAC

Теория

Вопрос 1. Чем отличается мандатное управление доступом от дискреционного?

**Дискреционное управление доступом (DAC):**

* Контроль основан на идентификаторе пользователя и принадлежности к группам
* Владелец объекта самостоятельно определяет права доступа
* Пример: классические права UNIX (rwx) и права в PostgreSQL (GRANT/REVOKE)
* Гибкое, но менее безопасное

**Мандатное управление доступом (MAC):**

* Контроль основан на системных политиках безопасности
* Права определяются системой, а не владельцем объекта
* Использует метки безопасности (SELinux, SE-PostgreSQL)
* Более строгое, но сложнее в настройке

Вопрос 2. Как установить метки безопасности для объектов базы данных в PostgreSQL?

-- Активация расширения

CREATE EXTENSION sepgsql;

-- Установка метки для таблицы

SECURITY LABEL ON TABLE inventory IS 'system\_u:object\_r:sepgsql\_secret\_table\_t:s0:c0.c1023';

-- Установка метки для столбца

SECURITY LABEL ON COLUMN inventory.quantity IS 'system\_u:object\_r:sepgsql\_secret\_table\_t:s0:c0.c1023';

-- Установка метки для последовательности

SECURITY LABEL ON SEQUENCE inventory\_item\_id\_seq IS 'system\_u:object\_r:sepgsql\_ro\_table\_t:s0';

Вопрос 3. Какие команды используются для настройки дискреционного управления доступом?

-- Предоставление прав

GRANT SELECT, INSERT ON table\_name TO role\_name;

GRANT ALL PRIVILEGES ON table\_name TO role\_name;

GRANT USAGE ON SCHEMA schema\_name TO role\_name;

-- Отзыв прав

REVOKE INSERT ON table\_name FROM role\_name;

REVOKE ALL PRIVILEGES ON table\_name FROM role\_name;

-- Создание ролей и пользователей

CREATE ROLE read\_only\_role;

CREATE USER user\_name WITH PASSWORD 'password';

GRANT read\_only\_role TO user\_name;

Вопрос 4. Как протестировать, что права доступа настроены правильно?

-- Проверка текущих прав

\dp table\_name

SELECT \* FROM information\_schema.table\_privileges WHERE table\_name = 'inventory';

-- Тестирование от имени другого пользователя

SET ROLE test\_user;

SELECT \* FROM inventory; -- Должно работать если есть права

INSERT INTO inventory VALUES (...); -- Должно fail если нет прав

RESET ROLE;

-- Или через командную строку

psql -U test\_user -d database\_name -c "SELECT \* FROM inventory;"

Вопрос 5. Какие преимущества и ограничения есть у мандатного управления доступом?

**Преимущества MAC:**

* Более высокий уровень безопасности
* Защита от несанкционированного изменения прав
* Контроль доступа на основе системных политик
* Защита от некоторых видов атак (например, privilege escalation)

**Ограничения MAC:**

* Сложность настройки и администрирования
* Возможные проблемы с совместимостью
* Требует глубоких знаний системы безопасности
* Может ограничивать легитимные операции

**Вопрос 6.** Какие ограничения существуют при использовании мандатного управления доступом (MAC) в PostgreSQL?

* Ограниченная поддержка в разных дистрибутивах PostgreSQL
* Сложность отладки и диагностики проблем
* Возможное влияние на производительность
* Не все объекты БД поддерживают метки безопасности
* Требует активации SELinux на системном уровне

Задание 7. Как реализовать настройку меток безопасности для строк таблицы с помощью расширения sepgsql?

В текущей реализации sepgsql не поддерживает метки на уровне строк - <https://github.com/SELinuxProject/selinux-notebook/blob/main/src/notebook-examples/sepgsql/testdb-example.sql> . Метки безопасности применяются только на уровне:

* Таблиц
* Столбцов
* Последовательностей
* Схем
* Функций

Задание 8. Какие проблемы могут возникнуть при использовании дискреционного управления доступом (DAC) в системах с большим количеством

пользователей?

* Сложность управления правами для множества пользователей
* Возможность ошибок при назначении прав
* Риск предоставления избыточных прав
* Трудности аудита и отслеживания изменений
* Проблемы с наследованием прав в сложных иерархиях ролей

Вопрос 9. Чем отличается команда GRANT от команды REVOKE, и в каких случаях они используются?

**GRANT** - предоставляет права: GRANT SELECT, INSERT ON table\_name TO user\_name;

REVOKE – отзывает права: REVOKE INSERT ON table\_name FROM user\_name;

Вопрос 10. Как проверять и тестировать права доступа пользователей или ролей в PostgreSQL?

-- Проверка прав конкретного пользователя

SELECT \* FROM information\_schema.role\_table\_grants

WHERE grantee = 'user\_name';

-- Проверка всех прав на таблицу

\dp table\_name

-- Тестирование через временное переключение ролей

SET ROLE test\_user;

-- Выполнение тестовых операций

SELECT \* FROM sensitive\_table; -- Проверка доступа

RESET ROLE;

-- Автоматизированное тестирование через скрипты

psql -U test\_user -d dbname -f test\_queries.sql

-- Проверка через системные представления

SELECT \* FROM pg\_roles WHERE rolname = 'user\_name';

SELECT \* FROM pg\_user WHERE usename = 'user\_name';