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 №10

по дисциплине «Основы построения защищённых баз данных»

Выполнил: обучающийся гр. ВКБ43

Ковалев Данил Петрович

Проверил:

Скляров Алексей Викторович

Тема: аудит действий и мониторинг изменений данных в PostgreSQL с использованием pgAudit и триггеров

Цель: настроить pgAudit для фиксации действий пользователей и создать триггеры для мониторинга изменений данных.

Вариант 6. Настроить pgAudit для логирования операций SELECT в таблице transactions. Создать триггер, который фиксирует все DELETE операции в transactions и записывает их в audit\_log.

Задание 1. Установить расширение pg\_audit в PostgreSQL.

Для взаимодействия с базой данных использовалась система контейнеризации Docker. Конфигурация работы PostgreSQL для работы с Docker представлена на рисунке 1.

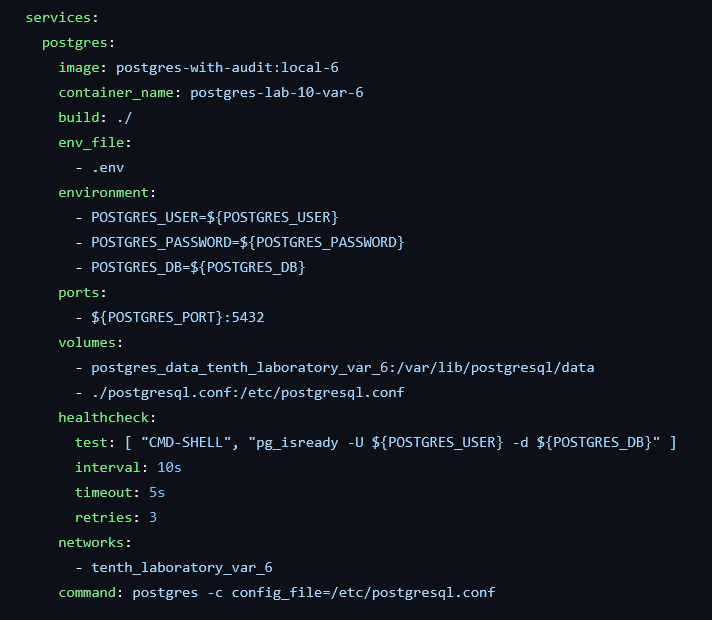


Рисунок 1 – конфигурация Docker compose для работы с PostgreSQL

Для установки pgaudit в Docker я написал такой вот Dockerfile, который представлен на рисунке 2.

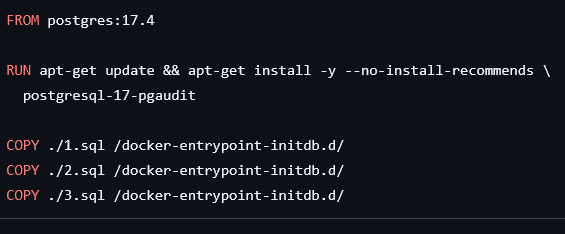


Рисунок 2 – установка pgaudit в Docker

Задание 2. Настроить параметры логирования в файле postgresql.conf, чтобы включить аудит заданных операций (в зависимости от выбранного варианта).

В моем варианте требуется только логгировать операции SELECT. В документации pgaudit было указано, что для таких целей нужно выставить режим read. Результат настройки представлен на рисунке 3.



Рисунок 3 – настройка postgresql.conf

Задание 3. В соответствии с вариантом задания, включить аудит для конкретных операций (например, SELECT, INSERT, UPDATE, DELETE, DDL).

В случае 6 варианта требуется логировать только SELECT операции, в следствии чего была написана конфигурация, как на рисунке 3. На рисунке 4 показано создание роли для pgaudit.



Рисунок 4 – создание роли для pgaudit

По моему варианту требуется в дальнейшем создать таблицу transactions, которая будет рассмотрена больше. Ниже на рисунке 5 демонстрируется правила логгирования pgaudit для данной таблицы.

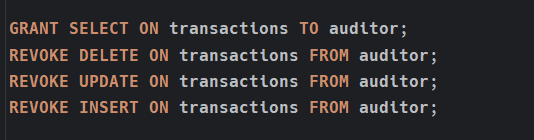


Рисунок 5 – выдача прав для логгирования таблицы transactions

Задание 4. Выполнить несколько тестовых запросов, соответствующих выбранному варианту, чтобы проверить запись этих операций в системный журнал.

При подключении к базе данных из-под Pycharm Professional выводятся такие вот логи, которые представлены ниже на рисунке 6.

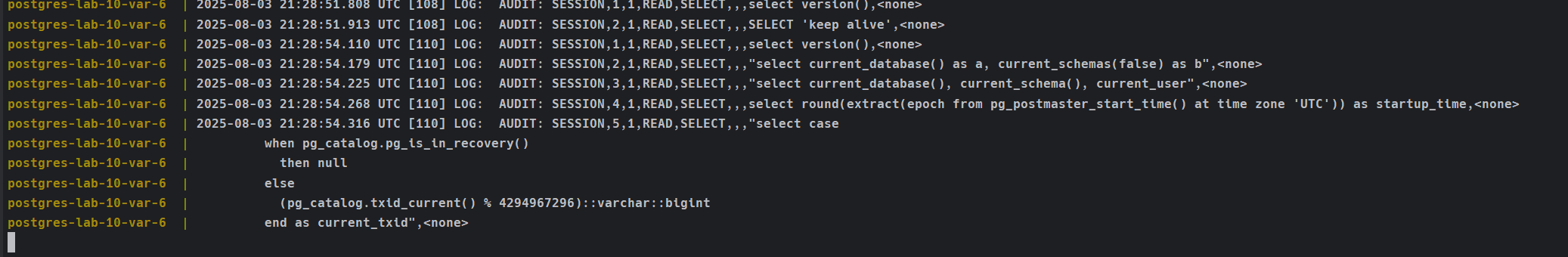


Рисунок 6 – логгирование подключения к базе данных через Pycharm

Попробуем теперь выполнить запрос на просмотр содержимого таблицы, для запроса буду использовать команду, которая представлена на рисунке 7.

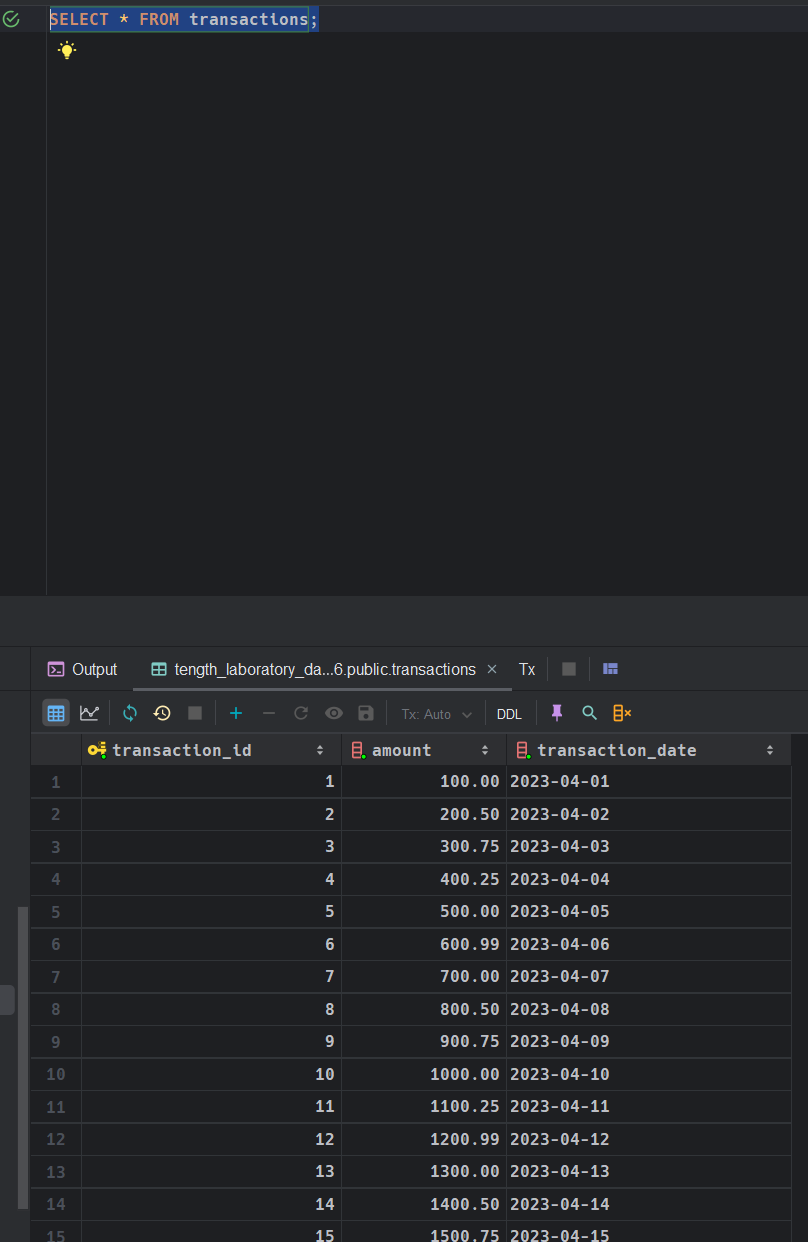


Рисунок 7 – удачный запрос для просмотра содержимого таблицы transactions

В логах на рисунке 8 видно, что запрос прошел удачно и все корректно работает.

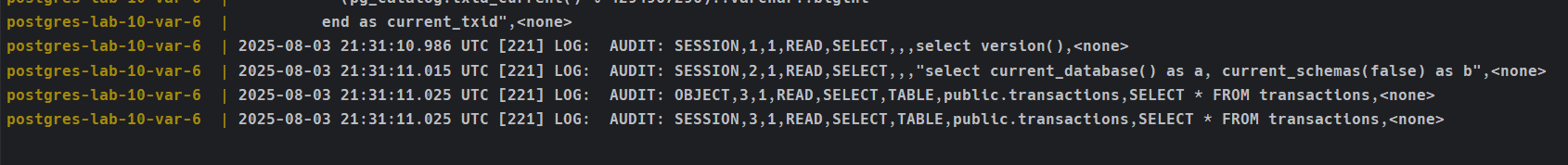


Рисунок 8 – отображение логов для SELECT запроса

Попробуем теперь вставить данные и удостовериться, что данные не отображаются. В данном случае я использовал команду, которая представлена на рисунке 9.

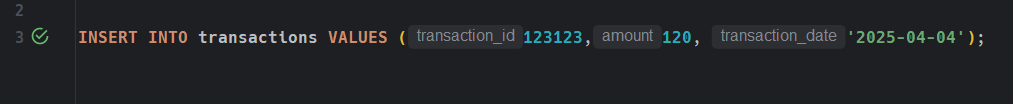


Рисунок 9 – вставка данных

В логах видно, что вставка не логгируется, но какое-нибудь удаление будет логгироваться, потому что там обязательно должно быть WHERE, который под капотом делает SELECT запрос. Логи представлены на рисунке 10.

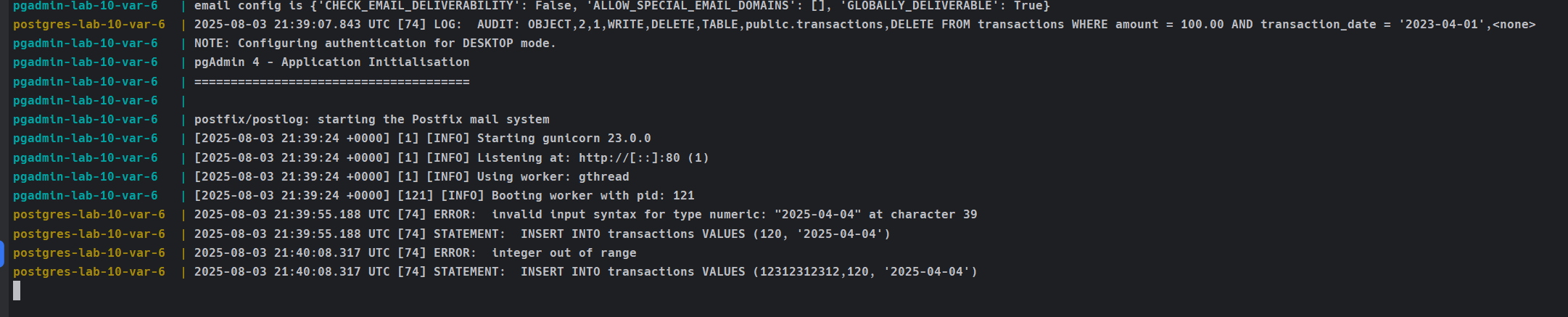


Рисунок 10 – отображение логов

Задание 5. Создать основную таблицу в соответствии с вариантом (например, clients, orders, employees и т.д.).

В соответствии с моим вариантом была создана таблица, которая представлена на рисунке 11.

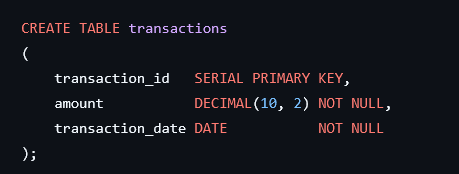


Рисунок 11 – таблица transactions

Задание 6. Добавить в таблицу несколько строк данных, чтобы иметь возможность тестировать триггеры и аудит.

В таблицу было сгенерировано изначально большое количество данных. На рисунке 12 приложена лишь малая часть.



Рисунок 12 – заполнение таблицы transactions

Задание 7. Создать таблицу audit\_log для записи истории изменений. Структура таблицы должна включать: Table\_name (название таблицы, где произошло изменение), operation (тип операции, например, INSERT, UPDATE, DELETE), user (имя пользователя, который выполнил операцию), old\_value и new\_value (старое и новое значение, если применимо), timestamp (время изменения).

Для выполнения задания был написан такой скрипт, который представлен ниже на рисунке 13.

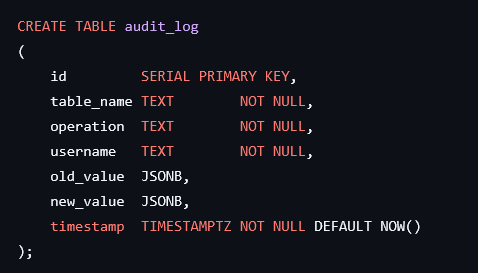


Рисунок 13 – таблица для задания

Задание 8. Написать функцию на PL/pgSQL, которая будет записывать изменения в таблицу audit\_log (в зависимости от типа операции в вашем варианте).

Для выполнения задания была написана функция “log\_transaction\_deletes”, которая вставляет данные в audit\_log, сохраняя имя таблицы, название операции, имя пользователя, старое значение и новое значение. В результате получился код, который представлен на рисунке 14.

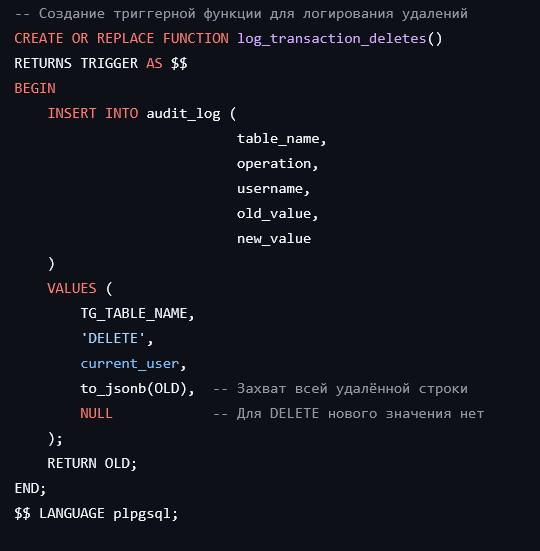


Рисунок 14 – функция для записи старых записей в audit\_log

Задание 9. Создать триггер на основе функции, который будет выполняться для заданных операций (например, INSERT, UPDATE, DELETE) в основной таблице.

В моем варианте требуется сохранять старые значения в audit\_log только при операции DELETE над таблицей transactions. В таком случае был написан триггер, который представлен на рисунке 15.

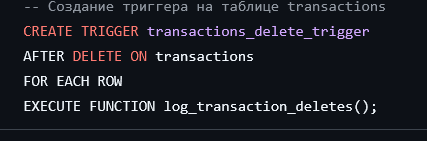


Рисунок 15 – триггер, который отслеживает удаление в таблице transactions

Задание 10. Выполнить действия над основной таблицей (добавление, изменение, удаление записей) в соответствии с заданием вашего варианта.

Для выполнения действий будет использоваться команда, которая представлена ниже на рисунке 16. Это простой DELETE, который удаляет по условию, ничего заумного нет.

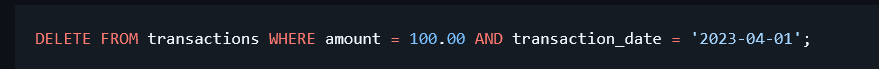


Рисунок 16 – запрос на удаление записи из таблицы transactions

Успешность выполнения запроса представлено на рисунке 17.

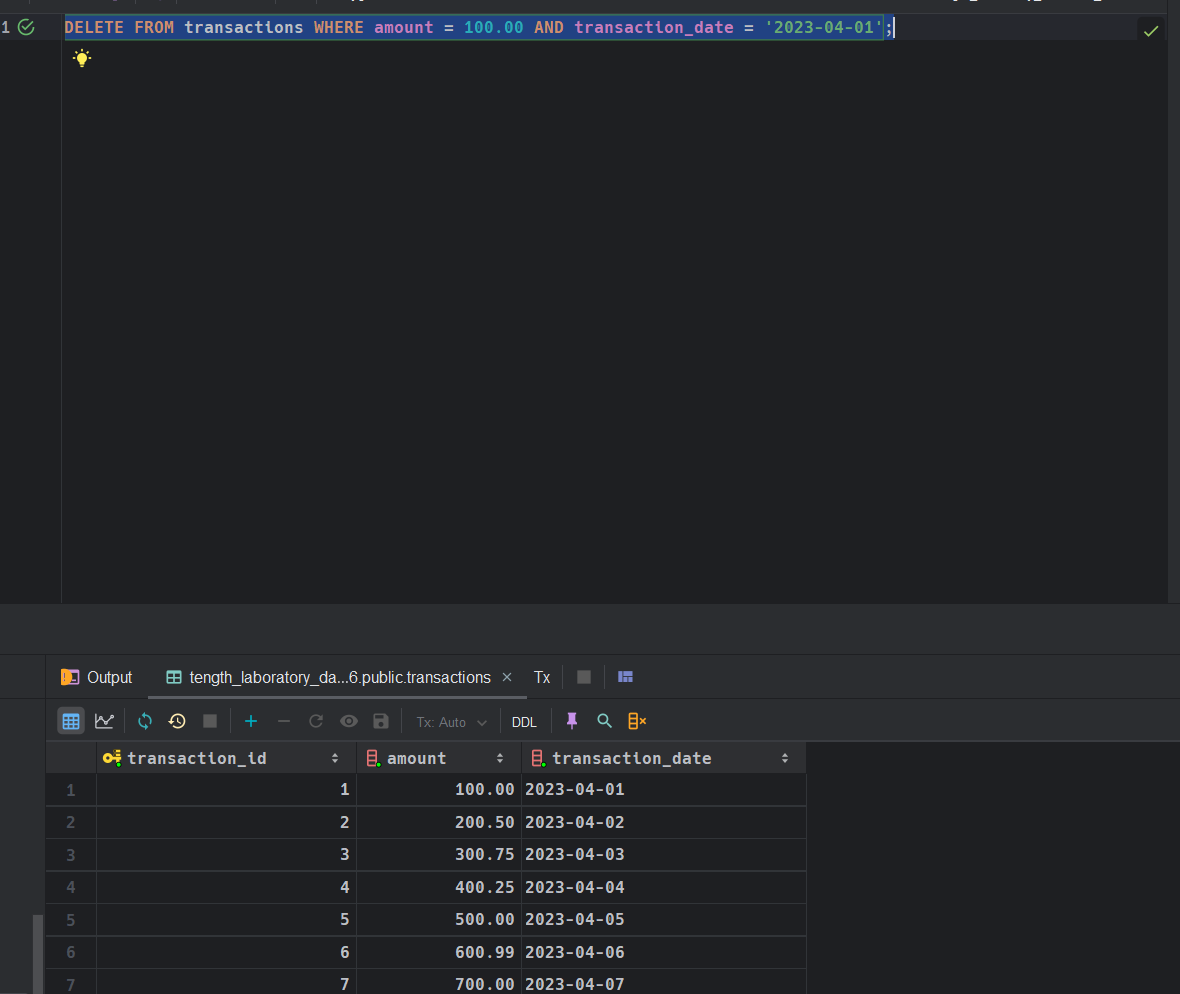


Рисунок 17 – факт успешности выполнения запроса

Задание 11. Проверить таблицу audit\_log и системный журнал, чтобы убедиться, что все изменения зафиксированы согласно настройкам pgAudit и триггера.

В таблице audit\_log видно, что все изменения были успешно применены и зафиксированны. Результат представлен на рисунке 18.

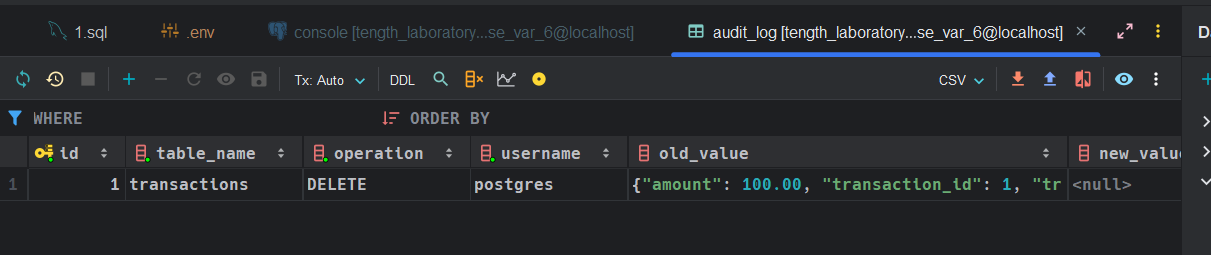


Рисунок 18 – результат выполнения 11 задания

Теория

Вопрос 1. Что такое pgAudit и для чего оно используется в PostgreSQL?

**pgAudit** — это расширение для PostgreSQL, предназначенное для **подробного аудита действий пользователей в базе данных**. Оно перехватывает и записывает в стандартный лог PostgreSQL информацию о том, кто, когда и какие именно SQL-команды выполнял. Используется для обеспечения соответствия требованиям безопасности и нормативным актам (GDPR, PCI DSS, HIPAA, SOX, ФЗ-152), расследование инцидентов безопасности, отслеживание изменений конфиденциальных данных, мониторинг активности пользователей и администраторов.

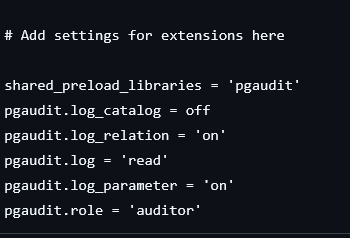
Вопрос 2. Какие типы операций можно логировать с помощью pgAudit?

pgAudit позволяет логировать операции на разных уровнях:

* **SESSION:** Запись **всех** операторов в рамках сессии (очень подробно, может быть много данных).
* **OBJECT:** Запись операторов, затрагивающих **конкретные объекты** (таблицы, представления и т.д.), которые были явно указаны для аудита.
* **READ:** Запись операторов SELECT и COPY TO (чтение данных), что особенно важно для аудита конфиденциальности.
* **Функции (DDL, DML, DCL, MISC):** можно настроить аудит для определенных классов операторов:
  + DDL (CREATE, ALTER, DROP - изменение структуры БД)
  + DML (INSERT, UPDATE, DELETE, TRUNCATE - манипуляция данными)
  + DCL (GRANT, REVOKE - управление правами)
  + ROLE (CREATE ROLE, ALTER ROLE, DROP ROLE)
  + MISC (разные команды, например DISCARD, FETCH, CHECKPOINT)
  + WRITE (INSERT, UPDATE, DELETE, TRUNCATE, COPY FROM)
  + FUNCTION (вызовы функций и процедур)

Вопрос 3. Какие настройки в файле postgresql.conf необходимо изменить для включения pgAudit?

Для включения pg\_audit нужно указать shared\_preload\_libraries, остальные параметры будут выставлены по умолчанию.



Вопрос 4. В чем преимущество использования pgAudit по сравнению с другими методами аудита?

* **Производительность:** реализован на уровне сервера C-кодом, минимизируя накладные расходы. Значительно эффективнее триггерных решений для аудита SELECT и массовых операций.
* **Полнота:** может аудировать **любые** SQL-операторы, включая SELECT, SET, команды администрирования, что критично для соответствия нормам.
* **Необходность:** записывает **фактически выполненный** сервером оператор после разбора и перезаписи (например, с подставленными синонимами), а не тот, что отправил клиент.
* **Контекст:** легко интегрируется со стандартным логом, содержащим временные метки, имя пользователя, имя БД, IP-адрес клиента (если настроено).
* **Гибкость:** Точечная настройка аудита (сессия/объект, классы операторов).

Вопрос 5. Что такое триггер, и как он работает в PostgreSQL?

**Триггер** — это специальная **хранимая функция в PostgreSQL**, которая **автоматически** выполняется ("срабатывает") при наступлении определенного **события** (INSERT, UPDATE, DELETE, TRUNCATE) на указанной **таблице** или представлении.

**Как работает:**

* 1. Пользователь выполняет операцию (напр., UPDATE).
  2. СУБД определяет, что для этой операции на целевой таблице есть триггер(ы).
  3. **До** или **после** выполнения основной операции (или **вместо** нее для представлений) СУБД вызывает связанную с триггером функцию.
  4. Функция триггера выполняется. Она имеет доступ к данным изменяемой строки через специальные переменные OLD (данные до изменения для UPDATE/DELETE) и NEW (данные после изменения для INSERT/UPDATE).
  5. Результат работы функции (например, модификация NEW, отмена операции) влияет на выполнение основной операции.

**Вопрос 6.** Для чего используется таблица audit\_log в данной лабораторной работе?

Таблица audit\_log (или подобная ей) в лабораторной работе **служит пользовательским хранилищем для записей аудита, генерируемых триггерами.** Вместо записи в лог сервера, триггеры вставляют строки с информацией об изменении (кто, когда, что сделал, старые/новые значения) непосредственно в эту таблицу в самой БД. Это позволяет:

* + Легко запрашивать историю изменений с помощью SQL.
  + Хранить аудит структурированно в реляционном виде.
  + Интегрировать аудит данных с прикладной логикой.

Задание 7. Какие данные можно записывать в таблицу audit\_log при помощи триггера?

В таблицу audit\_log можно записывать практически любую информацию, доступную триггерной функции в момент срабатывания:

* **Метаданные операции:**
  + operation\_type (INSERT/UPDATE/DELETE/TRUNCATE)
  + table\_name / schema\_name
  + timestamp (время операции)
  + user\_name (текущий пользователь БД)
  + application\_name (если задано клиентом)
  + client IP address (требует дополнительной настройки на стороне сервера или передачи приложением)
* **Данные изменяемой строки:**
  + OLD.\* / NEW.\* (весь кортеж до/после изменения)
  + Конкретные измененные колонки (напр., OLD.column\_name, NEW.column\_name)
  + Различия между OLD и NEW (для UPDATE).

Задание 8. Чем отличается логирование с помощью pgAudit от использования триггеров для мониторинга изменений?

|  |  |  |
| --- | --- | --- |
| Характеристика | pgAudit | Триггеры |
| Уровень | Сервер / Сессия / Оператор | Строка таблицы |
| Что аудирует | Любые SQL-команды (SELECT, DDL, SET и т.д.) | Только DML (INSERT, UPDATE, DELETE, TRUNCATE) |
| Производительность | Высокая (нативный C-код) | Низкая (накладные расходы на вызов функции для *каждой* строки) |
| Данные строк | Нет доступа к OLD/NEW | Есть доступ к OLD/NEW |
| Хранение | Файлы лога сервера | Таблица(ы) в БД |
| SELECT аудит | Да | Нет |
| Аудит DDL / прав доступа | Да | Нет (только через Event Triggers - отдельный механизм) |
| Гибкость данных | Стандартный/CSV лог | Полная (структура таблицы audit\_log на ваше усмотрение) |
| Контекст (IP, приложение) | Да (через стандартный лог | Сложно / Обычно нет |
| Назначение | Безопасность, соответствие нормам | Отслеживание изменений данных, кастомная бизнес-логика |

Вопрос 9. Какие команды SQL необходимы для создания функции и триггера в PostgreSQL?

CREATE FUNCTION … для создания функции, а для триггера CREATE TRIGGER …

Вопрос 10. Какие преимущества и недостатки имеют pgAudit и триггеры при организации мониторинга базы данных?

* **pgAudit:**
  + Преимущества: Высокая производительность (особенно для чтения и массовых операций), аудит ЛЮБЫХ команд (SELECT, DDL, DCL), минимальное влияние на бизнес-логику приложений, интеграция со стандартным логом (контекст), лучше подходит для соответствия строгим нормам.
  + Недостатки: нет возможности загрузки логов на удаленный сервер. Слишком неудобно его настраивать под Docker. Редко применяется на практике, потому что логи чаще всего собирают с ORM библиотек, которые умеют транслировать свой код в SQL запросы, посылая информацию в stdout.
* **Триггеры:**
  + Преимущества: Прямой доступ к данным строк (OLD/NEW), возможность хранить аудит структурированно в таблицах БД (легкий SQL-анализ), гибкость в формате и содержании записей аудита, могут использоваться для сложной бизнес-логики помимо аудита.
  + Недостатки: Серьезное падение производительности (особенно на частых UPDATE/DELETE), аудитят ТОЛЬКО DML (не SELECT, не DDL, не GRANT), увеличивают сложность схемы БД, могут усложнить миграции, не фиксируют контекст вызова (IP, приложение) без костылей, не видят операций, выполненных суперпользователем в обход триггеров (хотя редко).