

Лабораторная работа № 10

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.

Время выполнения лабораторной работы (аудиторные часы): 4 часа

Оборудование и программное обеспечение: ПК с установленной PostgreSQL, pgAudit, текстовый редактор для написания SQL-запросов.

1. Теоретические сведения

1.1 Аудит действий пользователей с использованием pgAudit

pgAudit — расширение для PostgreSQL, добавляющее возможности ведения аудита на уровне базы данных. Основное назначение pgAudit — логирование всех операций, которые выполняются пользователями базы данных, что обеспечивает контроль и безопасность данных. С помощью pgAudit можно фиксировать SQL-команды, такие как SELECT, INSERT, UPDATE, DELETE, а также административные команды (GRANT, REVOKE, ALTER и др.). Эти операции записываются в системный журнал, что позволяет отслеживать любые действия в базе данных и выявлять нарушения безопасности.

Комбинированное использование pgAudit и триггеров позволяет отслеживать как SQL-команды, так и изменения в данных. pgAudit записывает все действия на уровне команд, а триггеры фиксируют каждое изменение на уровне строк и столбцов, добавляя данные о времени, пользователе и типе операции.

Для работы **pgAudit** необходима его установка и настройка в конфигурационном файле PostgreSQL postgresql.conf. Основные параметры настройки включает pgaudit.log, который определяет типы операций, которые должны фиксироваться. Например, READ для операций чтения, WRITE для операций записи, DDL для команд создания и модификации структур.

После включения pgAudit начинает запись указанных действий в системный журнал. Журнал содержит информацию о типе действия, пользователе, времени и SQL-запросе, что позволяет анализировать данные при расследовании инцидентов или проверке соответствия требованиям безопасности.

query	calls	total_exec_time	rows	shared_blks_hit	shared_blks_read
SELECT TT1_Fld1642 FROM _Const9308 T1 WHERE (T1_Fld10059 = CAST(\$1 AS NUMERIC))	1	0.00402	1	1	0
PREPARE spsqlstmt_2446 AS select checkpoints, timed from pg_catalog pg_stat_bgwriter	2	0.09187	2	0	0
SELECT TT1_Fld15065 FROM _Const15064 T1 WHERE (T1_Fld10059 = CAST(\$1 AS NUMERIC))	1	0.00385	1	1	0
SELECT TT1_Fld10705 FROM _Const10704 T1 WHERE (T1_Fld10059 = CAST(\$1 AS NUMERIC))	1	0.00381	1	1	0
SELECT TT1_Fld27411 FROM _Const27410 T1 WHERE (T1_Fld10059 = CAST(\$1 AS NUMERIC))	1	0.00383	1	1	0
SELECT TT1_Fld15024 FROM _Const15023 T1 WHERE (T1_Fld10059 = CAST(\$1 AS NUMERIC))	1	0.00411	1	1	0
SELECT TT1_Fld10198 FROM _Reference10009 T1 WHERE (T1_Fld10059 = CAST(\$1 AS NUMERIC))	1	0.00434	1	1	0
SELECT TT1_Fld15093 FROM _Const15092 T1 WHERE (T1_Fld10059 = CAST(\$1 AS NUMERIC))	1	0.00375	1	1	0
SELECT TT1_Fld16213 FROM _Const16212 T1 WHERE (T1_Fld10059 = CAST(\$1 AS NUMERIC))	1	0.00385	1	1	0
SELECT TT1_Fld9123 FROM _Const9374 T1 WHERE (T1_Fld10059 = CAST(\$1 AS NUMERIC))	1	0.00406	1	1	0
SELECT TT1_Fld73001 FROM _Const9327 T1 WHERE (T1_Fld10059 = CAST(\$1 AS NUMERIC))	1	0.00375	1	1	0
SELECT TT1_Fld12132 FROM _Const12131 T1 WHERE (T1_Fld10059 = CAST(\$1 AS NUMERIC))	1	0.00412	1	1	0
SELECT TT1_Fld18867 FROM _Reference10009 T1 WHERE (T1_Fld10059 = CAST(\$1 AS NUMERIC))	1	0.00534	1	1	0

Рисунок 8.1 - Пример журнала действий с pgAudit

1.2 Использование триггеров для мониторинга изменений данных

Триггеры в **PostgreSQL** — это объекты базы данных, которые выполняют заданный код при возникновении определённых событий, таких как INSERT, UPDATE или DELETE. Использование триггеров позволяет создавать механизмы мониторинга, фиксируя каждое изменение данных. Триггеры часто применяются для записи изменений в отдельную таблицу, сохраняя старое и новое значение данных, имя пользователя и время операции.

Триггеры создаются на языке PL/pgSQL и могут реагировать на любое изменение данных в таблице. Например, триггер, установленный для события UPDATE, может записывать старые и новые значения данных в таблицу журнала изменений. Это позволяет легко проследивать историю изменений.

Пример функции и триггера для записи изменений:

```
CREATE OR REPLACE FUNCTION log_changes()
RETURNS TRIGGER AS $$
BEGIN
INSERT INTO audit_log (table_name, operation, user, old_value,
new_value, timestamp)
VALUES (TG_TABLE_NAME, TG_OP, current_user, OLD, NEW, NOW());
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

В данной функции:

- TG_TABLE_NAME — имя таблицы, в которой произошло изменение.
- TG_OP — тип операции (UPDATE, DELETE и др.).
- current_user — текущий пользователь.
- OLD и NEW — значения до и после изменения.

После создания функции настраивается триггер:

```
CREATE TRIGGER track_changes  
AFTER UPDATE ON monitored_table  
FOR EACH ROW EXECUTE FUNCTION log_changes();
```

Этот триггер выполняется после каждого обновления строки в таблице `monitored_table` и записывает изменения в `audit_log`.

2. Задание для самостоятельного выполнения работы

1. Установить расширение **pgAudit** в PostgreSQL.
2. Настроить параметры логирования в файле `postgresql.conf`, чтобы включить аудит заданных операций (в зависимости от выбранного варианта).
3. В соответствии с вариантом задания, включить аудит для конкретных операций (например, `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `DDL`).
4. Выполнить несколько тестовых запросов, соответствующих выбранному варианту, чтобы проверить запись этих операций в системный журнал.
5. Создать основную таблицу в соответствии с вариантом (например, `clients`, `orders`, `employees` и т.д.).
6. Добавить в таблицу несколько строк данных, чтобы иметь возможность тестировать триггеры и аудит.
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).
8. Написать функцию на PL/pgSQL, которая будет записывать изменения в таблицу `audit_log` (в зависимости от типа операции в вашем варианте).
9. Создать триггер на основе функции, который будет выполняться для заданных операций (например, `INSERT`, `UPDATE`, `DELETE`) в основной таблице.
10. Выполнить действия над основной таблицей (добавление, изменение, удаление записей) в соответствии с заданием вашего варианта.
11. Проверить таблицу `audit_log` и системный журнал, чтобы убедиться, что все изменения зафиксированы согласно настройкам **pgAudit** и триггера.

Вариант	Задание
1.	Настроить pgAudit для логирования операций SELECT и INSERT в таблице clients. Создать триггер для записи изменений в таблицу audit_log при выполнении UPDATE в clients.
2.	Настроить pgAudit для логирования всех операций (SELECT, INSERT, UPDATE, DELETE) в таблице orders. Создать триггер, фиксирующий изменения в orders с сохранением старого значения в audit_log.
3.	Включить логирование только административных операций (CREATE, ALTER, DROP) с помощью pgAudit . Создать триггер, записывающий операции DELETE в таблице products в журнал audit_log.
4.	Настроить pgAudit для логирования операций UPDATE в таблице employees. Создать триггер, который фиксирует изменения в employees и записывает в audit_log как старое, так и новое значения полей position и salary.
5.	Настроить pgAudit для логирования всех DDL-операций в базе данных. Создать триггер, который записывает изменения в таблице inventory при операциях INSERT и UPDATE, добавляя время и пользователя в audit_log.
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.
7.	Настроить pgAudit для логирования операций INSERT и UPDATE в таблице sales. Создать триггер для записи изменений при операциях UPDATE в sales, добавляя в audit_log старое и новое значения полей amount и status.
8.	Включить логирование операций DELETE в таблице users с помощью pgAudit . Создать триггер, который записывает изменения при INSERT в users, фиксируя в audit_log время, пользователя и введённые данные.
9.	Настроить pgAudit для фиксации операций UPDATE и DELETE в таблице logistics. Создать триггер, записывающий все изменения в logistics при UPDATE, добавляя в audit_log старые значения полей location и status.
10.	Настроить pgAudit для логирования всех операций (SELECT, INSERT, UPDATE, DELETE) в таблице accounts. Создать триггер для записи изменений при UPDATE и DELETE в accounts, включая все данные старой и новой строк в audit_log.

Контрольные вопросы

1. Что такое **pgAudit** и для чего оно используется в PostgreSQL?
2. Какие типы операций можно логировать с помощью **pgAudit**?
3. Какие настройки в файле `postgresql.conf` необходимо изменить для включения **pgAudit**?
4. В чем преимущество использования **pgAudit** по сравнению с другими методами аудита?
5. Что такое триггер, и как он работает в PostgreSQL?
6. Для чего используется таблица `audit_log` в данной лабораторной работе?
7. Какие данные можно записывать в таблицу `audit_log` при помощи триггера?
8. Чем отличается логирование с помощью **pgAudit** от использования триггеров для мониторинга изменений?
9. Какие команды SQL необходимы для создания функции и триггера в PostgreSQL?
10. Какие преимущества и недостатки имеют **pgAudit** и триггеры при организации мониторинга базы данных?

Литература

1. PostgreSQL Documentation. PostgreSQL 15 Documentation [Электронный ресурс]. — URL: <https://www.postgresql.org/docs/>
2. PostgreSQL Audit Extension (pgAudit). GitHub Repository [Электронный ресурс]. — URL: <https://github.com/pgaudit/pgaudit>
3. Веремеенко, А. В. Использование триггеров для автоматического аудита данных в базе данных PostgreSQL / А. В. Веремеенко, Д. М. Петров // Современные информационные технологии. — 2023. — Т. 19, № 2. — С. 53–60.
4. Иванов, П. Н. Аудит безопасности в PostgreSQL: возможности и расширения / П. Н. Иванов // Информационные технологии и безопасность. — 2022. — Т. 14, № 3. — С. 101–108.
5. Smith, R., & Brown, T. Implementing Database Triggers for Effective Data Change Tracking in PostgreSQL // Journal of Database Management. — 2023. — Vol. 18, No. 4. — P. 231–245.
6. Симонов, И. Ю. Применение PostgreSQL для мониторинга изменений и обеспечения безопасности данных / И. Ю. Симонов, А. П. Ковалев // Вестник технологий баз данных. — 2021. — № 1. — С. 12–19.
7. Мартынова, Л. А. Основы SQL и аудит баз данных. Учебное пособие / Л. А. Мартынова. — М.: Инфра-М, 2022. — 256 с.