Instituto Tecnológico de Costa Rica

Ingeniería en Computación

Proyecto Bases de Datos II

AutoAudit

María Paula Castillo Chinchilla

Rafael Odio Mendoza

Jairo González Hidalgo

Aaron Líos Cubillo

Sede San Carlos

Septiembre 2025

Tabla de contenido

Introducción

El siguiente proyecto solicita el desarrollo de una extensión para PostgreSQL llamada AutoAudit. La extensión debe de tener agregar habilidades de auto-auditoría al motor de bases de datos. Esto con el proposito de trazar las operaciones de creación, edición y eliminación (INSERT, UPDATE, DELETE) de las variables y tablas que realizan los usuarios dentro de una base de datos, eliminando la necesidad de instalar dependencias externas.

En el entorno laboral, la auditoría es un requisito para la seguridad, el cumplimiento normativo y el análisis forense. Pero, su implementación tiende a dificultarse a causa de ciertos elementos que complejizan la operación del sistema. AutoAudit presenta una solución estandarizada, reproducible y acoplada al ecosistema de extensiones de PostgreSQL, coexistiendo con sus mecanismos de “triggers” y “event triggers” (PostgreSQL Global Development Group, 2024).

Con el fin de asegurar compatibilidad con diversas estructuras de tablas y facilitar consultas posteriores, los estados previos y posteriores de los registros se almacenan en formato JSON. Además, la extensión incorpora una única función de trigger inmutable y un mecanismo para propagar automáticamente el trigger de auditoría tanto a todas las tablas existentes al momento de la instalación, como a las tablas que se creen posteriormente, mediante un trigger de evento a nivel de sistema.

Así, la auditoría se habilita de manera uniforme, minimizando la intervención operativa y el riesgo de fallo.

En este proyecto consiguió realizar la implementación de la función de auditoría y su tabla de eventos, la activación automática de triggers en todas las tablas, y el empaquetado según el estándar oficial de extensiones de PostgreSQL. La documentación describe los criterios de diseño, las decisiones técnicas y el procedimiento de instalación y uso, con el objetivo de ofrecer una base clara para validación, demostración y futuras mejoras. AutoAudit busca aportar una solución libre, mantenible y efectiva a la necesidad de auditoría integral en bases de datos PostgreSQL.

Análisis del problema

Necesidad

- Trazabilidad completa de cambios en la base de datos: quién, cuándo, desde dónde y qué datos cambiaron.
- Seguridad y cumplimiento (auditorías internas/externas, normativas).
- Análisis forense y depuración de incidentes.
- Control de calidad de datos.

Problema actual

- Soluciones ad-hoc con triggers por tabla: fáciles de omitir y difíciles de mantener.
- Auditoría a nivel de aplicación: no cubre accesos directos a la BD.
- Cobertura incompleta (sin BEFORE/AFTER completos o sin DELETE).
- Riesgos de seguridad por falta de aislamiento y privilegios adecuados.

Desafíos técnicos

- Esquemas heterogéneos: difícil capturar “antes” y “después” de forma uniforme.
- Propagación automática a nuevas tablas.
- Rendimiento y crecimiento del almacenamiento.
- Integridad y no repudio de los registros.

Requisitos derivados

- Cobertura INSERT/UPDATE/DELETE con metadatos (tabla, usuario, IP, timestamp).
- Snapshots before/after en JSON para compatibilidad universal.
- Única función de trigger inmutable, aplicada automáticamente a tablas existentes y futuras (event trigger).
- Esquema exclusivo con privilegios restringidos.
- Instalación estandarizada como extensión (CREATE EXTENSION autoaudit;).

Solución del problema

Enfoque general

Se plantea una extensión de PostgreSQL denominada AutoAudit que automatiza la auditoría de todas las operaciones DML (INSERT, UPDATE, DELETE) sobre todas las tablas de una base de datos. La solución encapsula su lógica en un esquema propio, con objetos controlados por el rol instalador (postgres/administrador), garantizando uniformidad, seguridad y mantenibilidad.

Diseño del modelo de auditoría

- Definir el esquema autoaudit y la tabla audit_log con campos: id (PK), operation, schema_name, table_name, event_ts (timestamptz), actor (rol), client_ip, before_row (jsonb), after_row (jsonb).
- Establecer privilegios: solo el rol instalador administra autoaudit; el resto solo consulta (si se requiere).
- Índices sugeridos: por event_ts, (schema_name, table_name), actor y operación para consultas y mantenimiento.

Captura de contexto y serialización

- Determinar metadatos con funciones nativas: CURRENT_USER, inet_client_addr(), TG_TABLE_SCHEMA, TG_TABLE_NAME, statement_timestamp().
- Serializar OLD/NEW con to_jsonb(OLD) y to_jsonb(NEW), manejando:
    - INSERT: before_row = NULL, after_row = NEW.
    - UPDATE: ambos poblados.

    - DELETE: before_row = OLD, after_row = NULL.

Función de trigger inmutable y única

- Implementar una función en PL/pgSQL que inserte en audit_log según TG_OP y retorne la fila adecuada (NEW u OLD).
- Mantener una sola función para toda la base, evitando divergencias lógicas y facilitando pruebas.

Despliegue automatizado de triggers

- Crear triggers de fila AFTER INSERT OR UPDATE OR DELETE para todas las tablas actuales (excluyendo autoaudit.*).
- Definir un event trigger (CREATE TABLE) que, al detectarse nuevas tablas, les adjunte el trigger de auditoría automáticamente.

Empaquetado como extensión

- Proveer autoaudit.control y el script de versión autoaudit--1.0.sql con:
    - Creación de esquema, tabla, índices, función, triggers y event trigger.
    - Salvaguardas idempotentes (IF NOT EXISTS) y revocación de privilegios por defecto.
- Activación estándar: CREATE EXTENSION autoaudit; en cada base donde se requiera.

Operación, consulta y mantenimiento

- Consulta típica: select * from autoaudit.audit_log where table_name = 'mi_tabla' order by event_ts desc;
- Sugerencias de operación:
    - Políticas de retención/archivado por fecha.
    - Particionado por rango temporal si el volumen es alto.
    - Monitoreo de espacio e índices.

Diagrama conceptual (texto)

- Aplicaciones/usuarios -> Operaciones DML en tablas public y otras
- Triggers de fila (INSERT/UPDATE/DELETE) invocan función autoaudit.f_audit()
- f_audit() construye metadatos + JSON before/after
- Inserción en autoaudit.audit_log
- Consultas administrativas leen audit_log

Creación de la tabla de auditoría

```
-- ========================================
CREATE TABLE autoaudit.audit_log (
    event_id       BIGSERIAL PRIMARY KEY,
    operation_type TEXT NOT NULL,  -- INSERT, UPDATE, DELETE
    table_name     TEXT NOT NULL,
    event_time     TIMESTAMPTZ NOT NULL DEFAULT now(),
    executed_by    TEXT NOT NULL,  -- current_user
    client_ip      INET,           -- client connection ip
    old_data       JSONB,
    new_data       JSONB
);
```

Función de trigger

```sql
-- ========================================
CREATE OR REPLACE FUNCTION autoaudit.audit_trigger()
RETURNS trigger
LANGUAGE plpgsql
SECURITY DEFINER
AS $$
DECLARE
    v_old JSONB;
    v_new JSONB;
    v_executor TEXT;
BEGIN
    BEGIN
        v_executor := current_setting('role', true);
    EXCEPTION
        WHEN others THEN v_executor := NULL;
    END;
    IF v_executor IS NULL OR v_executor = '' OR v_executor = 'none' THEN v_executor := session_user;
    END IF;
    IF (TG_OP = 'DELETE') THEN
        v_old := to_jsonb(OLD);
        v_new := NULL;
    ELSIF (TG_OP = 'UPDATE') THEN
        v_old := to_jsonb(OLD);
        v_new := to_jsonb(NEW);
    ELSIF (TG_OP = 'INSERT') THEN
        v_old := NULL;
        v_new := to_jsonb(NEW);
    END IF;
    INSERT INTO autoaudit.audit_log (operation_type, table_name, executed_by, client_ip, old_data, new_data)
    VALUES (
        TG_OP,              -- Operation: INSERT, UPDATE, DELETE
        TG_TABLE_NAME,      -- Affected table
        v_executor,       -- User/role executing the statement
        inet_client_addr(), -- client connection IP
        v_old,              -- Row state before the operation
        v_new               -- Row state after the operation
    );
    IF (TG_OP = 'DELETE') THEN RETURN OLD;
    ELSE RETURN NEW;
    END IF;
END;
$$;
```

Análisis de resultados

Estado final del proyecto

Creación de esquema y privilegios

Estado: Completo

Observaciones: Esquema creado y completo.

Implementación de la función de auditoría

Estado: Completo

Observaciones: Función única autoaudit.audit_trigger() captura INSERT/UPDATE/DELETE y serializa OLD/NEW a JSONB.

Creación de triggers en todas las tablas existentes

Estado: Completo

Observaciones: attach_triggers_to_all_tables() recorre las tablas de usuario y aplica el trigger automáticamente.

Trigger de evento para nuevas tablas

Estado: Completo

Observaciones: Event trigger autoaudit_attach_new_tables agrega el trigger tras cada CREATE TABLE.

Creación y configuración de la extensión

Estado: Completo

Observaciones: Instalación estándar con CREATE EXTENSION autoaudit; manual de uso verificado.

Registro de metadatos y before/after en JSON

Estado: Completo

Observaciones: audit_log contiene operation_type, table_name, event_time, executed_by, client_ip, old_data, new_data.

Resumen de logros

- La extensión está completa, registra perfectamente las operaciones y los cambios realizados informando el nombre del usuario que la realizó.
- La extensión puede ser instalada con normalidad como cualquier otra extensión para PostgreSQL.
- La extensión funciona bien, y no presentó errores durante las pruebas.
- Se cumple con todos los requisitos.

Conclusiones

Se logró que la extensión registre de forma consistente todas las operaciones INSERT, UPDATE y DELETE en las tablas de usuario. La extensión garantiza una cobertura total de los cambios que se solicitan dentro del enunciado del proyecto. Con la implementación del esquema de auditoría centralizado que almacena y registra todos los datos en los archivos JSON, los cuales resultan bastante útiles debido a su compatibilidad y su flexibilidad. Se automatizó el seguimiento y supervisión de datos actuales y futuras, esto se logró mediante una única función trigger. Finalmente, se aseguró que un usuario no pueda realizar acciones para las cuales no cuenta con los privilegios necesarios.

Recomendaciones

1. Se podrían agregar más elementos centrados en el análisis forense, como vistas temáticas que se centren en apartados específicos. Por ejemplo:
    - Una vista centrada en el usuario que realizó un cambio específico.
    - Una centrada en la tabla que fue modificada.
    - Una centrada en los cambios realizados, resaltando el antes y el después de una tabla o una variable modificada.
2. Se podría implementar un registro de “logging” que registre el nombre de cada usuario que ingresa y la hora a la que lo hace.
3. También se podría implementar un sistema que detecte cambios en horarios no laborales, esto con el propósito de detectar actividades sospechosas o algún tipo de sabotaje hacia la base de datos. Esto se complementaría con el registro de “logging”

Bibliografía

- PostgreSQL Global Development Group. (2024). PostgreSQL 16 Documentation: Triggers and Event Triggers. https://www.postgresql.org/docs/current/triggers.html