



TRIGGERS DE AUDITORÍA EN BASES DE DATOS

CAPÍTULO I: INTRODUCCIÓN

a. Auditoría y Aplicación de Triggers en Sistemas de Base de Datos

El trabajo se centra en la investigación de los triggers de auditoría dentro de los sistemas de base de datos. Estos triggers, que a menudo se implementan para rastrear y registrar cambios específicos en una base de datos, han ganado prominencia en las prácticas modernas de administración de bases de datos debido a su capacidad para asegurar la integridad de los datos.

b. Planteamiento del problema:

Las bases de datos actuales están sujetas a constantes cambios y actualizaciones. Esto plantea la necesidad de tener un mecanismo de seguimiento para rastrear dichos cambios, especialmente cuando se trata de operaciones críticas que pueden impactar la integridad o seguridad de los datos. Aquí es donde entran en juego los triggers de auditoría. Sin embargo, la correcta implementación y gestión de estos triggers es complejo llevar a cabo. Hay cuestiones muy importantes que se deben tener en cuenta ¿Cómo se pueden implementar eficazmente base de datos? ¿Qué desafíos se presentan en su implementación y cómo pueden superarse?

c. Objetivo General:

Analizaremos la importancia y aplicabilidad de los triggers de auditoría en los sistemas de bases de datos, identificando los desafíos comunes en su implementación y proponiendo soluciones para una implementación eficaz.

i. Objetivos Específicos:

- Definimos qué son los triggers de auditoría para entender su relevancia en las prácticas modernas de administración de bases de datos.
- Exploramos las mejores prácticas y técnicas para la implementación efectiva de triggers de auditoría en diferentes sistemas de base de datos.
- Analizamos casos de estudio donde la implementación de triggers de auditoría ha sido crítica para garantizar la



CAPÍTULO II: MARCO CONCEPTUAL O REFERENCIAL

Definición de Triggers de Auditoría:

Un trigger, en el contexto de bases de datos, es un conjunto de instrucciones que se ejecutan automáticamente en respuesta a ciertos eventos dentro de una base de datos. Los triggers de auditoría, en particular, son diseñados para monitorizar y registrar operaciones que afecten al contenido de la base de datos, proporcionando así un registro de actividad y posibilitando la detección de acciones no autorizadas o inesperadas.

Importancia de la Auditoría en Bases de Datos:

En la era actual de la información, donde las organizaciones dependen en gran medida de sus sistemas de información y bases de datos, la integridad y seguridad de estos datos es primordial. La auditoría permite no solo asegurar la integridad de los datos, sino también cumplir con regulaciones, identificar posibles brechas de seguridad y mantener un registro histórico de las transacciones.

Tipos de Triggers:

- Triggers DML (Data Manipulation Language): Se activan en respuesta a eventos DML, como INSERT, UPDATE y DELETE. Estos son, con frecuencia, los utilizados en operaciones de **auditoría**.
- Triggers DDL (Data Definition Language): Se disparan en respuesta a eventos DDL, como la creación, modificación o eliminación de estructuras de base de datos.
- Triggers de LOGON y LOGOFF: Registran eventos relacionados con la conexión y desconexión de usuarios.

Ventajas de la Implementación de Triggers de Auditoría:

- Transparencia: Proporcionan un registro detallado de todas las transacciones realizadas.
- Prevención de Fraudes: Ayudan a detectar y prevenir actividades sospechosas en tiempo real.
- Cumplimiento Normativo: Asisten en el cumplimiento de regulaciones y estándares relacionados con la protección y privacidad de datos.



CAPÍTULO III: METODOLOGÍA SEGUIDA

- a. Nuestro equipo ha decidido emprender este proyecto con una metodología estratégica y detallada. Comenzamos analizando atentamente el contexto del ejercicio propuesto, dedicando tiempo y esfuerzo para comprender a profundidad los objetivos del mismo. Posteriormente, nos sumergimos en una exhaustiva investigación sobre los triggers de auditoría, buscando adquirir un conocimiento integral acerca de su función, relevancia y mecanismos de operación. Una vez fortalecidos con este marco teórico, procedimos a implementar y aplicar estos triggers en los scripts que nos han sido proporcionados como parte esencial de la consigna inicial del trabajo.
- b. En la realización de este proyecto, hemos adoptado una diversidad de recursos y herramientas para garantizar un enfoque exhaustivo y actualizado. Con ayuda audiovisual analizamos numerosos videos de la plataforma YouTube, donde expertos en la materia detallan y explican con precisión el funcionamiento de los triggers. Simultáneamente, hemos consultado variados artículos en línea, que nos han proporcionado perspectivas adicionales sobre el tema.
- Además, conscientes de la importancia de la comunicación efectiva y constante entre los miembros del equipo, hemos implementado plataformas comunicacionales. WhatsApp y Discord han sido esenciales en este sentido, permitiéndonos mantenernos al tanto de cada actualización, modificación y avance en el proyecto. Estas herramientas nos han facilitado una colaboración fluida y coordinada, asegurando que todos los integrantes estuvieran informados y alineados en cada etapa del proceso.

CAPÍTULO IV: DESARROLLO DEL TEMA / PRESENTACIÓN



DE RESULTADOS

A continuación explicaremos paso a paso cómo llevamos a cabo la realización de tres diferentes triggers en el entorno de SQL server, “Insert”, “Delete” y “Edit”, todos estos fueron realizados dentro de la tabla conserje en el apartado dedicado para este tipo especial de operación: Triggers

I. TABLA AUDITORÍA

Primeramente antes de crear un nuevo Trigger, debemos presentar un breve contexto de estos, como sabemos un trigger de auditoría es conjunto de instrucciones que se ejecutan automáticamente en respuesta a ciertos eventos dentro de una base de datos, su función se basa en monitorear y registrar operaciones que afecten al contenido de la base de datos, proporcionando así un registro de actividad y posibilitando la detección de acciones no autorizadas o inesperadas.

Este denominado registro de actividad anteriormente mencionado requiere un espacio en el cual guardar cada uno de sus registros, el lugar ideal para esto se encuentra de la misma base de datos, ya que en esta gozaremos de un acceso fácil e inmediato cuando sea requerido, por ello crearemos la **tabla de auditoría**.

```
USE base_consortio
go

CREATE TABLE auditoria (
    id_auditoria int identity primary key,
    tabla_afectada varchar(100),
    columna_afectada varchar(100),
    accion varchar(10),
    fecha_hora datetime,
    usuario varchar(50),
    valor_anterior varchar(max),
    valor_actual varchar(max),
);
```

Esta tabla está diseñada tanto para representar una visualización rápida a datos que se encuentran en las tablas que activaron el trigger (como por ejemplo el nombre de la tabla donde se realizó, los valores alterados, la columna afectada, entre otros) como también para almacenar datos de importancia relacionados a esta acción realizada, tales como el usuario llevo esta acción a cabo, qué tipo de acción se realizó o la fecha y hora en la cual ocurrió.

id_auditoria (int, identidad, clave principal): este campo es una clave principal de incremento automático que se utiliza para identificar de forma única cada registro de auditoría en la tabla. Proporciona una manera rápida y consistente de acceder a todos los registros de auditoría.

tabla_afectada: almacena el nombre de la tabla afectada por la operación monitoreada. Esto le indicará qué tablas se han modificado.

columna_afectada: indica la columna específica que se cambió. Si se



trata de una operación de inserción, este campo puede estar en blanco o contener un valor NULL.

acción: especifica el tipo de acción que se realizará en la tabla afectada. Por ejemplo, puede contener valores como "INSERT", "UPDATE" y "DELETE" para indicar el tipo de operación realizada. **date_time** (fecha y hora): registra la fecha y hora exactas en que se realizó la operación de verificación. Esto proporciona un registro temporal de la operación.

usuario: almacena el nombre del usuario o aplicación que inició la operación monitoreada. Esto le ayuda a realizar un seguimiento de quién realizó cambios.

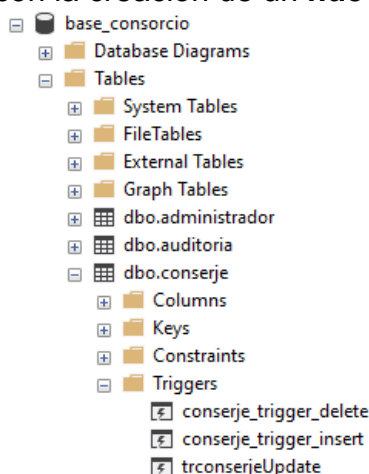
valor_anterior y valor_actual: estos campos se utilizan para almacenar los valores anteriores y actuales de las columnas afectadas, respectivamente. Esto le permite ver qué datos se cambiaron durante el proceso de auditoría. Esto es importante para comprender los cambios realizados en sus datos.

En síntesis, se busca que cada vez que se realice una acción de carácter insertar, eliminación o edición los datos relacionados se almacenen en la tabla auditoría, es decir que, cada vez que se inserte un conserje nuevo podremos visualizarlo en la tabla auditoría.

Esta tabla resulta muy útil en situaciones donde, por ejemplo, se requiere saber en qué fecha se creó un usuario o se necesita deshacer un dato editado o borrado.

II. NUEVO TRIGGER

Una vez saldada esta explicación de la tabla auditoría, podemos continuar con la creación de un **nuevo trigger**.



Los triggers no se guardan automáticamente en la carpeta "Triggers", por lo que debemos crearlos de forma manual dentro de esta en el interior de la carpeta donde nos interese que estén presentes.

En este caso, la tabla de conserjes: *base_consortio/Tables/bdo.conserje/Triggers*

Dicho script se verá de la siguiente manera:



```
SQLQuery1.sql - DE...K4LG\Usuario (56) - DESKTOP-RDRK4LG\S...io - dbo.auditoria  DESKTOP-RDRK4LG\S...io -
-- =====
-- Template generated from Template Explorer using:
-- Create Trigger (New Menu).SQL
--
-- Use the Specify Values for Template Parameters
-- command (Ctrl-Shift-M) to fill in the parameter
-- values below.
--
-- See additional Create Trigger templates for more
-- examples of different Trigger statements.
--
-- This block of comments will not be included in
-- the definition of the function.
-- =====
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =====
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- =====
CREATE TRIGGER <Schema_Name, sysname, Schema_Name>.<Trigger_Name, sysname, Trigger_Name>
ON <Schema_Name, sysname, Schema_Name>.<Table_Name, sysname, Table_Name>
AFTER <Data_Modification_Statements, , INSERT,DELETE,UPDATE>
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for trigger here

END
GO
```

Como podemos observar, este archivo ya contiene palabras guía en cuanto a la estructura de un Trigger así como una serie de comentarios que podemos seguir para su desarrollo, estos poseen el objetivo de guiarnos y servirnos de ayuda en esta primera instancia.

III. TRIGGER DE AUDITORÍA: INSERT

Este Trigger se activará cada vez que se inserte un nuevo registro en la tabla conserje, rellenando campos y guardando información en la tabla de auditoría. Cabe aclarar que la tabla conserje almacena información de personas con el cargo de conserje, los campos del cual consta son id, nombre y apellido, fecha de nacimiento, teléfono y estado civil.

Inicialmente escribiremos la siguiente función:

```
ALTER TRIGGER conserje_trigger_insert
ON dbo.conserje
AFTER INSERT
AS
```

Utilizamos la instrucción **ALTER TRIGGER** para alterar el trigger al cual llamamos **conserje_trigger_insert**.

En la siguiente línea especificamos con **ON** en que tabla se activará el trigger y luego después de que acción debe hacerlo (**AFTER INSERT**), en este caso, cuando se inserte un nuevo registro.



Luego, continuaremos con un conjunto de instrucciones **INSERT INTO**, está se encarga de agregar nuevas filas de datos a una tabla existente, en este caso nuestra tabla auditoría, continuamos de esta manera especificando los valores que deseamos agregar para cada columna de la tabla, los cuales son: la tabla y columnas afectadas, los valores anteriores y actuales, el usuario que realizó la acción, la fecha y que tipo de acción fue, en este caso, inserción.

Con **SELECT** detallamos uno por uno con qué datos queremos que se rellenen estos campos antes nombrados: el id, apellido y nombre, teléfono, fecha de nacimiento y estado civil.

Cabe aclarar la función de un par de instrucciones utilizadas, como **system_user**: la cual devuelve el nombre del usuario del sistema que está realizando la operación de inserción y **CURRENT_TIMESTAMP**, que responde con la fecha y hora actuales cuando se realiza la operación de inserción.

```
INSERT INTO dbo.auditoria(tabla_afectada, columna_afectada, valor_Anterior, valor_actual, usuario, fecha_hora, accion)
SELECT 'dbo.conserje', 'idconserje', NULL, inserted.idconserje, SYSTEM_USER, CURRENT_TIMESTAMP, 'Insert' FROM inserted;

INSERT INTO dbo.auditoria(tabla_afectada, columna_afectada, valor_Anterior, valor_actual, usuario, fecha_hora, accion)
SELECT 'dbo.conserje', 'apeynom', NULL, inserted.apeynom, SYSTEM_USER, CURRENT_TIMESTAMP, 'Insert' FROM inserted;

INSERT INTO dbo.auditoria(tabla_afectada, columna_afectada, valor_Anterior, valor_actual, usuario, fecha_hora, accion)
SELECT 'dbo.conserje', 'tel', NULL, inserted.tel, SYSTEM_USER, CURRENT_TIMESTAMP, 'Insert' FROM inserted;

INSERT INTO dbo.auditoria(tabla_afectada, columna_afectada, valor_Anterior, valor_actual, usuario, fecha_hora, accion)
SELECT 'dbo.conserje', 'fechnac', NULL, inserted.fechnac, SYSTEM_USER, CURRENT_TIMESTAMP, 'Insert' FROM inserted;

INSERT INTO dbo.auditoria(tabla_afectada, columna_afectada, valor_Anterior, valor_actual, usuario, fecha_hora, accion)
SELECT 'dbo.conserje', 'estciv', NULL, inserted.estciv, SYSTEM_USER, CURRENT_TIMESTAMP, 'Insert' FROM inserted;
```

Una vez que el trigger estuvo terminado lo guardamos y lo pusimos a prueba insertando un nuevo conserje con nombre Bricia Diaz en dicha tabla, relleno con datos de ejemplo:

	idconserje	apeynom	tel	fechnac	estciv
	4	Fabio Gonzalez	1234121212	2003-01-12 00:0...	S
	5	Jazmin Baez	3786453212	2003-05-09 00:0...	S
	6	Bricia Diaz	3794121212	2003-01-06 00:0...	S

Luego de haber terminado, ingresamos a los registros de la tabla auditoría, donde ya podíamos visualizar que la acción que acabamos de llevar a cabo se había guardado correctamente:

	id_auditoria	tabla_afectada	columna_afectada	accion	fecha_hora	usuario	valor_anterior	valor_actual
1	1	dbo.conserje	idconserje	Insert	2023-10-28 20:55:34.007	DESKTOP-5BGRFCQ\aleja	NULL	2
2	2	dbo.conserje	apeynom	Insert	2023-10-28 20:55:34.007	DESKTOP-5BGRFCQ\aleja	NULL	Bricia
3	3	dbo.conserje	tel	Insert	2023-10-28 20:55:34.007	DESKTOP-5BGRFCQ\aleja	NULL	234234
4	4	dbo.conserje	fechnac	Insert	2023-10-28 20:55:34.007	DESKTOP-5BGRFCQ\aleja	NULL	Jan 6 1999 12:00AM
5	5	dbo.conserje	estciv	Insert	2023-10-28 20:55:34.007	DESKTOP-5BGRFCQ\aleja	NULL	S

IV. TRIGGER DE AUDITORÍA: DELETE



Este Trigger se activará cada vez que se elimine un registro en la tabla conserje, de igual manera como en la acción de insertar se rellenan los campos y guardará información en la tabla de auditoría. Como ya sabemos la tabla conserje almacena información de personas con el cargo de conserje, los campos del cual consta son id, nombre y apellido, fecha de nacimiento, teléfono y estado civil.

Escribiremos la siguiente función:

```
CREATE TRIGGER conserje_trigger_delete
ON [dbo].[conserje]
AFTER DELETE
AS
BEGIN
    SET NOCOUNT ON;
```

Donde la única diferencia que podemos notar además del nombre es la instrucción **AFTER DELETE**, la cual especifica que después de cada eliminación se debe crear un registro por cada campo relacionado en la tabla de auditoría.

Continuamos con el conjunto de instrucciones de **INSERT INTO** como lo hicimos anteriormente, diferenciando la acción de insertar por la de eliminar

```
INSERT INTO dbo.auditoria(tabla_afectada, columna_afectada, valor_Anterior, valor_actual, usuario, fecha_hora, accion)
SELECT 'dbo.conserje', 'idconserje', deleted.idconserje, NULL, SYSTEM_USER, CURRENT_TIMESTAMP, 'Delete' FROM deleted;

INSERT INTO dbo.auditoria(tabla_afectada, columna_afectada, valor_Anterior, valor_actual, usuario, fecha_hora, accion)
SELECT 'dbo.conserje', 'apeynom', deleted.apeynom, NULL, SYSTEM_USER, CURRENT_TIMESTAMP, 'Delete' FROM deleted;

INSERT INTO dbo.auditoria(tabla_afectada, columna_afectada, valor_Anterior, valor_actual, usuario, fecha_hora, accion)
SELECT 'dbo.conserje', 'tel', deleted.tel, NULL, SYSTEM_USER, CURRENT_TIMESTAMP, 'Delete' FROM deleted;

INSERT INTO dbo.auditoria(tabla_afectada, columna_afectada, valor_Anterior, valor_actual, usuario, fecha_hora, accion)
SELECT 'dbo.conserje', 'fechnac', deleted.fechnac, NULL, SYSTEM_USER, CURRENT_TIMESTAMP, 'Delete' FROM deleted;

INSERT INTO dbo.auditoria(tabla_afectada, columna_afectada, valor_Anterior, valor_actual, usuario, fecha_hora, accion)
SELECT 'dbo.conserje', 'estciv', deleted.estciv, NULL, SYSTEM_USER, CURRENT_TIMESTAMP, 'Delete' FROM deleted;
```

Después de esto, verificamos que el trigger esté funcionando correctamente eliminando el registro “Fabio Gomez” de la tabla conserje

	id_auditoria	tabla_afectada	columna_afect...	accion	fecha_hora	usuario	valor_anterior	valor_actual
	1	dbo.conserje	idconserje	Insert	2023-10-30 23:0...	DESKTOP-RDRK...	NULL	1
	2	dbo.conserje	apeynom	Insert	2023-10-30 23:0...	DESKTOP-RDRK...	NULL	Fabio Gomez
	3	dbo.conserje	tel	Insert	2023-10-30 23:0...	DESKTOP-RDRK...	NULL	3786121212
	4	dbo.conserje	fechnac	Insert	2023-10-30 23:0...	DESKTOP-RDRK...	NULL	Feb 20 2003 12:...
	5	dbo.conserje	estciv	Insert	2023-10-30 23:0...	DESKTOP-RDRK...	NULL	C
	6	dbo.conserje	idconserje	Delete	2023-10-30 23:0...	DESKTOP-RDRK...	1	NULL
	7	dbo.conserje	apeynom	Delete	2023-10-30 23:0...	DESKTOP-RDRK...	Fabio Gomez	NULL
	8	dbo.conserje	tel	Delete	2023-10-30 23:0...	DESKTOP-RDRK...	3786121212	NULL
	9	dbo.conserje	fechnac	Delete	2023-10-30 23:0...	DESKTOP-RDRK...	Feb 20 2003 12:...	NULL
	10	dbo.conserje	estciv	Delete	2023-10-30 23:0...	DESKTOP-RDRK...	C	NULL

V. TRIGGER DE AUDITORÍA: UPDATE



VI. Aplicación de optimización de consultas con índices en nuestro trabajo

Uno de los temas a aplicar en nuestro trabajo fue la optimización de consultas con índices. Investigando al respecto y observando el trabajo de nuestros compañeros que realizaron este tema, pudimos concluir de qué forma podríamos aplicar el concepto de índices a nuestro trabajo de “Triggers de Auditoría”.

(agregar concepto índice)

Primeramente, lo que hicimos fue crear un index de la siguiente manera:

```
CREATE INDEX idx_idconserje ON dbo.conserje (idconserje);
```

El código anterior representa la creación de un índice no agrupado en la base de datos consorcio, afectando a la tabla conserje. A continuación encontraremos los detalles:

- **CREATE INDEX:** Este es el comando utilizado para crear un nuevo índice en una tabla existente en SQL Server.
- **index idconserje:** Es el nombre del índice. Generalmente, se elige un nombre que refleje la columna o las columnas que el índice cubre para facilitar su identificación.
- **ON dbo.conserje:** Indica que el índice se está creando en la tabla conserje que pertenece al esquema dbo (database owner). El esquema dbo es el predeterminado en las bases de datos de SQL Server para los objetos de base de datos.
- **(idconserje):** Denota la columna de la tabla en la que se creará el índice. En este caso, idconserje es la columna que contiene el identificador único para cada fila en la tabla conserje.

Este índice no agrupado permitirá que las consultas que buscan en la columna idconserje se realicen más rápidamente porque el servidor de base de datos puede utilizar el índice para encontrar los datos necesarios sin escanear toda la tabla. Es útil para mejorar el rendimiento en operaciones de búsqueda, filtro y unión donde idconserje es un criterio clave.

A continuación realizamos un ejemplo con el objetivo de mejorar el entendimiento a modo de ejemplificación:

Escenario Sin Índice:

Supongamos que se requiere buscar en la base de datos de consorcio en la tabla un conserje con ID = 70 sin aplicar el índice que definimos anteriormente. Ejecutamos la siguiente consulta:

```
SELECT * FROM conserje c WHERE c.idconserje = 70;
```

Activando las “estadísticas para cliente” se nos permite ver los tiempos de respuesta de Microsoft SQL Server a continuación:



	Trial 1	Average
Network Statistics		
Number of server roundtrips	2	→ 2.0000
TDS packets sent from client	2	→ 2.0000
TDS packets received from server	2	→ 2.0000
Bytes sent from client	208	→ 208.0000
Bytes received from server	440	→ 440.0000
Time Statistics		
Client processing time	7	→ 7.0000
Total execution time	33	→ 33.0000
Wait time on server replies	26	→ 26.0000

Query executed successfully. | localhost (16.0 RTM) | DESKTOP-F4QRKAL\Usuari... | base_consortio | 00:00:00 | 1 rows

Se nos informa de un tiempo de respuesta de 33 milisegundos.

Nuevamente repetiremos la misma consulta posteriormente de la creación del índice visto anteriormente, recibiendo como resultado el siguiente número:

	Trial 1	Average
Network Statistics		
Number of server roundtrips	2	→ 2.0000
TDS packets sent from client	2	→ 2.0000
TDS packets received from server	2	→ 2.0000
Bytes sent from client	208	→ 208.0000
Bytes received from server	440	→ 440.0000
Time Statistics		
Client processing time	2	→ 2.0000
Total execution time	5	→ 5.0000
Wait time on server replies	3	→ 3.0000

Query executed successfully. | localhost (16.0 RTM) | DESKTOP-F4QRKAL\Usuari... | base_consortio | 00:00:00 | 17 rows

Podemos darnos cuenta que, luego de la implementación del índice y la realización de la misma consulta, el tiempo de respuesta actual se redujo de forma significativa. Esta vez nos encontramos con un tiempo de ejecución de 5 milisegundos.

CONCLUSIÓN - Beneficios de triggers y JSON

Tanto los triggers como los índices son herramientas complementarias en SQL Server: por una parte los triggers aseguran la integridad y la lógica de negocio automatizada tras las operaciones de datos, mientras que los índices mejoran la velocidad y el rendimiento de las consultas utilizadas dentro de los triggers. Juntos, pueden mejorar la eficiencia de las tareas de mantenimiento, siempre y cuando se utilicen de manera prudente para evitar un impacto negativo en el rendimiento de la base de datos.

VII. Aplicación de optimización de consultas con índices en nuestro trabajo

JSON en SQL Server ofrece una solución versátil y eficiente para manejar información semiestructurada en un entorno de base de datos relacional. Esta integración permite almacenar, consultar y modificar datos JSON directamente dentro de la base de datos, lo que es particularmente útil cuando se trata con datos cuya estructura puede variar.



A continuación realizamos un ejemplo de cómo incluir datos de tipo JSON y como los triggers se activan y guardan dichas modificaciones en la tabla auditoria:

Inserción de registro:

```
INSERT INTO conserje (apeynom, tel, fechnac, estciv)
VALUES ('{"apellido": "GAUNA", "nombre": "PEDRO"}', 3794651348, '1998-12-12', 'S');
```

Resultados obtenidos que se almacena en la tabla auditoria después de la inserción de datos:

id...	tabla_afectada	columna_afectada	accion	fecha_hora	usuario	valor_anterior	valor_actual
1	dbo.conserje	idconserje	Insert	2023-11-19 21:39:36.560	DESKTOP-F4QRKAL\Usuario	NULL	222
2	dbo.conserje	apeynom	Insert	2023-11-19 21:39:36.560	DESKTOP-F4QRKAL\Usuario	NULL	{"apellido": "GAUNA", "nombre": "PEDRO"}
3	dbo.conserje	tel	Insert	2023-11-19 21:39:36.560	DESKTOP-F4QRKAL\Usuario	NULL	3794651348
4	dbo.conserje	fechnac	Insert	2023-11-19 21:39:36.560	DESKTOP-F4QRKAL\Usuario	NULL	Dec 12 1998 12:00AM
5	dbo.conserje	estciv	Insert	2023-11-19 21:39:36.560	DESKTOP-F4QRKAL\Usuario	NULL	S

Actualización del registro:

```
UPDATE conserje
SET apeynom = JSON_MODIFY(apeynom, '$.nombre', 'SERGIO')
WHERE idconserje = 221;
```

Resultados obtenidos que se almacena en la tabla auditoria después de la actualización del registro:

6	dbo.conserje	idconserje	Update	2023-11-19 21:41:24.857	DESKTOP-F4QRKAL\Usuario	221	221
7	dbo.conserje	apeynom	Update	2023-11-19 21:41:24.857	DESKTOP-F4QRKAL\Usuario	{"apellido": "...	{"apellido": "GAUNA", "nombre": "SERGIO"}
8	dbo.conserje	tel	Update	2023-11-19 21:41:24.857	DESKTOP-F4QRKAL\Usuario	3794651348	3794651348
9	dbo.conserje	fechnac	Update	2023-11-19 21:41:24.857	DESKTOP-F4QRKAL\Usuario	Dec 12 19...	Dec 12 1998 12:00AM
10	dbo.conserje	estciv	Update	2023-11-19 21:41:24.857	DESKTOP-F4QRKAL\Usuario	S	S

Actualización del registro:

```
UPDATE conserje
SET apeynom = JSON_MODIFY(apeynom, '$.nombre', NULL)
WHERE idconserje = 221;
```

Resultados obtenidos que se almacena en la tabla auditoria después de la actualización del registro:

11	dbo.conserje	idconserje	Update	2023-11-19 22:31:14.830	DESKTOP-F4QRKAL\Usuario	221	221
12	dbo.conserje	apeynom	Update	2023-11-19 22:31:14.830	DESKTOP-F4QRKAL\Usuario	{"apellido": "GAUNA", "nombre": "SERGI...	{"apellido": "GAUNA"}
13	dbo.conserje	tel	Update	2023-11-19 22:31:14.830	DESKTOP-F4QRKAL\Usuario	3794651348	3794651348
14	dbo.conserje	fechnac	Update	2023-11-19 22:31:14.830	DESKTOP-F4QRKAL\Usuario	Dec 12 1998 12:00AM	Dec 12 1998 12:00AM
15	dbo.conserje	estciv	Update	2023-11-19 22:31:14.830	DESKTOP-F4QRKAL\Usuario	S	S

CONCLUSIÓN:


Esta integración nos permite tener una flexibilidad en la representación de datos, un registro preciso de los cambios en cada columna del registro. Además nos brinda capacidad de consulta y modificación gracias a las funciones y operaciones específicas (JSON_VALUE, JSON_QUERY, JSON_MODIFY, etc) para datos JSON facilitan la consulta y modificación eficientes de datos semiestructurados directamente en la base de datos.



CAPÍTULO V: CONCLUSIONES

Al culminar este proyecto, sentimos que hemos adquirido una comprensión profunda y bien fundamentada sobre los triggers. Enfrentar el desafío de investigar un tema desde cero y, posteriormente, aplicarlo en un contexto preestablecido ha sido una tarea retadora para nosotros. Este proceso no solo nos ha brindado un conocimiento técnico, sino también la satisfacción de haber potenciado la eficacia del sistema con el que trabajamos.

VI. BIBLIOGRAFÍA.

- <https://ed.team/blog/triggers-que-son-y-para-que-sirven>
- <https://es.stackoverflow.com/questions/567905/crear-un-trigger-que-almacene-en-una-tabla-de-auditor%C3%ADa-el-registro-de-todos-los>
-  Control de cambios o auditoría de tablas en SQL Server
- <https://learn.microsoft.com/es-es/sql/relational-databases/triggers/dml-triggers?view=sql-server-ver16>
- <https://learn.microsoft.com/es-es/sql/relational-databases/triggers/use-the-inserted-and-deleted-tables?view=sql-server-ver16>
- <https://www.manageengine.com/products/eventlog/logging-guide/auditing-using-sql-server-triggers.html#:~:text=DML%20triggers%20are%20the%20procedures,the%20modification%20of%20your%20database.>
- <https://javifer2.wordpress.com/2019/10/07/auditoria-de-tabla-con-triggers/>
- https://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/base-de-datos-ii/2019/ii/guia-7.pdf
- <https://learn.microsoft.com/es-es/sql/t-sql/statements/alter-trigger-transact-sql?view=sql-server-ve>
- <https://chat.openai.com>