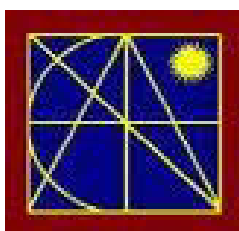




UNIVERSIDAD NACIONAL DEL NORDESTE



Facultad de Ciencias Exactas y Naturales y Agrimensura

Cátedra: Bases de Datos I

Año: 2023

Informe:

Proyecto integrador de Bases de Datos I

Alumnos:

Dominguez, Anibal Benjamin	DNI: 44197704
Almada, Tomás Emanuel	DNI: 44876943
Romero, Maria Cecilia	DNI: 36469060
Romero, Marcos Lautaro	DNI: 43266218
Rodriguez, Maria Agustina	DNI: 40565365

Índice

1. Introducción	3
1.1 Objetivo del trabajo Práctico	3
2. Marco Conceptual y Referencial	4
2.1 Porque realizar copias de seguridad	4
3. Metodología Implementada	4
4. Desarrollo del tema	5
4.1 Estrategia de Seguridad y restauración	5
4.2. Elección del modelo de recuperación adecuado	6
4.3. Diseñar la estrategia de copia de seguridad	7
4.4. Probar las copias de seguridad	8
4.5. Restauración en línea	8
4.6. Resultados	9
5. Implementación de temas	
5.1 Tema: Optimización de consultas mediante índices	12
5.2 Tema: Manejo de Permisos a Nivel de usuarios de base datos	18
5.3 Tema: Transacciones y transacciones anidadas	21
5.4 Tema: Triggers de auditoría	22
5.5 Tema: Procedimientos y funciones de almacenado	29
6. Conclusiones	38
7. Bibliografía	39

1 - Introducción

En un entorno donde los datos son el activo más valioso, tanto para empresas como para usuarios individuales, la necesidad de proteger la información contra pérdidas accidentales, ataques cibernéticos o fallos de hardware se ha vuelto imperativo en el mundo de la informática. La importancia de las copias de seguridad (backup) y la capacidad de restauración (restore) son los temas que se van abordar en este documento.

La copia de seguridad y la capacidad de restauración proporcionan una protección fundamental sobre la información de los usuarios, ya sea a nivel personal o empresarial de sus bases de datos. Para minimizar el riesgo de una pérdida de datos catastrófica, se debe realizar de forma periódica copias de seguridad de las bases de datos para conservar las modificaciones realizadas en los mismos. Una estrategia bien diseñada de copia de seguridad y restauración ayuda a proteger las bases de datos frente a la pérdida de datos provocada por diversos errores.

1.1 Objetivo del Trabajo Práctico

Este trabajo se realiza con el propósito de dar a entender el uso de los Backup y la importancia de mantener un respaldo de nuestras bases de datos en caso de accidentes. Explicar de la forma más entendible como se realizan dichos **Backup** y luego **restaurarlos** en caso de necesitarlos.

1. Objetivos Generales

Se espera que los backup sean un resguardo en caso de emergencia en el que la base de datos sufra algún daño o pérdida. De esta forma, anticipar una posible pérdida de información importante, necesaria en la actualidad ya que cualquier retraso puede implicar graves problemas.

2. Objetivos Específicos

Implementar el backup de una bases de datos, y comprobar que se haya guardado correctamente mediante la restauración (restore) de la misma, para así tener verificado la totalidad del backup.

2 - Marco Conceptual o Referencial

2.1. Por qué realizar copias de seguridad

La copia de seguridad de las bases de datos, la ejecución de procedimientos de restauración de prueba de las copias de seguridad y el almacenamiento de las copias en una ubicación segura y fuera del sitio contribuyen a proteger los datos ante posibles accidentes. Las copias de seguridad son la única forma de proteger los datos.

Con las copias de seguridad válidas de una base de datos se puede recuperar los datos en caso de que se produzcan errores, por ejemplo:

- Errores de medios.
- Errores de usuario. Por ejemplo, quitar una tabla por error.
- Errores de hardware. Por ejemplo, una unidad de disco dañada o la pérdida permanente de un servidor.
- Desastres naturales.

Además, las copias de seguridad de una base de datos son útiles para fines administrativos habituales, como copiar una base de datos de un servidor a otro.

3 - Metodología Implementada

Para esta primera instancia se usaron: bases de datos en SQL Server y Transact-SQL para programar las tareas referidas al backup y restore.

En nuestro equipo creemos que seguir esta metodología para garantizar la integridad de los datos y la disponibilidad de la información en caso de fallos o pérdida de datos está bien definida .

En primer lugar, se inició realizando una copia de seguridad completa de la base de datos utilizando la instrucción "BACKUP DATABASE" para respaldar todos los datos y objetos asociados. Para poder hacer esto, primero se tuvo que verificar que el modo de recuperación de bases de datos se encuentre en el modo adecuado para realizar un backup en línea. Por ello, se cambió dicho modo al estado "FULL" mediante la instrucción "SET RECOVERY FULL;".

Luego, se estableció una estrategia de respaldo para asegurar que los cambios posteriores se capturarán de manera efectiva. En caso de una falla o pérdida de datos, el proceso de restauración se lleva a cabo mediante la instrucción "RESTORE DATABASE", que recupera la base de datos a un estado consistente con el último backup válido. Es fundamental documentar y automatizar este proceso para

garantizar una recuperación eficiente y minimizar el tiempo de inactividad en situaciones críticas.

4 - Desarrollo del tema

El componente **copia de seguridad (backup)** y **restauración (restore)** de SQL Server proporciona una protección fundamental para proteger los datos críticos almacenados en las bases de datos de SQL Server. Para minimizar el riesgo de una pérdida de datos catastrófica, se debe realizar de forma periódica copias de seguridad de las bases de datos para conservar las modificaciones realizadas en los datos. Una estrategia bien diseñada de copia de seguridad y restauración le ayuda a proteger las bases de datos frente a la pérdida de datos provocada por diversos errores.

4.1. Estrategias de copias de seguridad y restauración

Las operaciones de copia de seguridad y restauración deben personalizarse para un entorno concreto y funcionar con los recursos disponibles. Por lo tanto, un uso confiable de las copias de seguridad y la restauración para la recuperación requiere una estrategia de copia de seguridad y restauración. Una estrategia de copia de seguridad y restauración bien diseñada equilibra los requisitos empresariales de disponibilidad máxima de los datos y la pérdida mínima de datos, al tiempo que se tiene en cuenta el costo de mantenimiento y almacenamiento de las copias de seguridad.

Diseñar una estrategia de copia de seguridad y restauración eficaz requiere mucho cuidado en el planeamiento, la implementación y las pruebas. Es necesario realizar pruebas: no se tendrá una estrategia de copia de seguridad hasta que se hayan restaurado correctamente las copias de seguridad en todas las combinaciones incluidas en la estrategia de restauración y se haya probado la base de datos restaurada en busca de coherencia física. Se debe tener en cuenta varios factores. Entre ellas se incluyen las siguientes:

- Los objetivos de la organización con respecto a las bases de datos de producción, especialmente los requisitos de disponibilidad y protección de datos frente a pérdidas o daños.
- La naturaleza de cada una de las bases de datos: el tamaño, los patrones de uso, la naturaleza del contenido, los requisitos de los datos, etc.

- Restricciones de los recursos, como hardware, espacio para almacenar los medios de copia de seguridad, seguridad física de los medios almacenados, etc.

4.2. Elección del modelo de recuperación adecuado

Las operaciones de copias de seguridad y restauración se producen en el contexto de un **modelo de recuperación**. El modelo de recuperación es una propiedad de la base de datos que controla la forma en que se administra el registro de transacciones. Por tanto, el modelo de recuperación de una base de datos determina qué tipos de copias de seguridad y qué escenarios de restauración se admiten para la base de datos, así como el tamaño de las copias de seguridad del registro de transacciones. Normalmente, en las bases de datos se usa el modelo de recuperación simple o el modelo de recuperación completa. El modelo de recuperación completa puede aumentarse cambiando al modelo de recuperación optimizado para cargas masivas de registros antes de las operaciones masivas.

La mejor elección de modelo de recuperación para la base de datos depende de los requisitos empresariales. Para evitar la administración del registro de transacciones y simplificar la realización de copias de seguridad y restauración, utilice el modelo de **recuperación simple**. Para minimizar el riesgo de pérdida de trabajo, a costa de una sobrecarga de trabajo administrativo, utilice el modelo de **recuperación completa**. Para minimizar el impacto en el tamaño del registro durante las operaciones de registro masivo, a la vez que permite la recuperación de dichas operaciones, use el modelo de **recuperación optimizado** para cargas masivas de registros.

4.3. Diseñar la estrategia de copia de seguridad

Una vez seleccionado un modelo de recuperación que cumpla los requisitos de su empresa para una base de datos específica, debe planear e implementar una estrategia de copias de seguridad. La estrategia de copias de seguridad óptima depende de distintos factores, de entre los cuales destacan los siguientes:

- ¿Cuántas horas al día requieren las aplicaciones acceso a la base de datos?

Si prevé un período de poca actividad, se recomienda programar las copias de seguridad de bases de datos completas en dicho período.

- ¿Cuál es la probabilidad de que se produzcan cambios y actualizaciones?

Si se realizan cambios frecuentes, tenga en cuenta los siguientes aspectos:

- ❖ Con el modelo de recuperación simple, considere la posibilidad de programar copias de seguridad diferenciales entre copias de seguridad de bases de datos completas. Una copia de seguridad diferencial solo incluye los cambios desde la última copia de seguridad de base de datos completa.
- ❖ Con el modelo de recuperación completa, debe programar copias de seguridad de registros frecuentes. La programación de copias de seguridad diferenciales entre copias de seguridad completas puede reducir el tiempo de restauración al disminuir el número de copias de seguridad del registro que se deben restaurar después de restaurar los datos.
- ¿Es probable que los cambios tengan lugar solo en una pequeña parte de la base de datos o en una grande?

Para una base de datos grande en la que los cambios se concentran en una parte de los archivos o grupos de archivos, las copias de seguridad parciales o de archivos pueden ser útiles.

- ¿Cuánto espacio en disco necesitará una copia de seguridad completa de la base de datos?
- ¿Hasta qué punto en el pasado su empresa requiere que se mantengan las copias de seguridad?

4.4. Probar las copias de seguridad

No tendrá una estrategia de restauración hasta que compruebe las copias de seguridad. Es muy importante comprobar cuidadosamente la estrategia de copia de seguridad de cada una de las bases de datos restaurando una copia de la base de datos en un sistema de prueba. Debe comprobar la restauración de cada tipo de copia de seguridad que pretenda utilizar. También se recomienda que, una vez restaurada la copia de seguridad, realice comprobaciones de coherencia de la base de datos a través de DBCC CHECKDB de la base de datos para validar que el medio de copia de seguridad no se ha dañado.

4.5. Restauración en línea

La restauración de datos mientras la base de datos está en línea se denomina **restauración en línea**. Se considera que una base de datos está en línea siempre que el grupo de archivos principal esté en línea, aunque alguno de los grupos de archivos secundarios esté sin conexión. En todos los modelos de recuperación se puede restaurar un archivo sin conexión mientras la base de datos está en línea. En el modelo de recuperación completa, también se pueden restaurar páginas mientras la base de datos está en línea.

Durante una operación de restauración de archivos en línea, los archivos que se estén restaurando y su grupo de archivos están sin conexión. Si algunos de dichos archivos está en línea cuando se inicia una restauración en línea, la primera instrucción de la restauración deja sin conexión el grupo de archivos al que pertenece el archivo. Por el contrario, durante una restauración en línea de una página, solo esa página está sin conexión.

El escenario de restauración en línea implica los siguientes pasos básicos:

- Restaurar los datos.
- Restaurar el registro utilizando WITH RECOVERY para la última restauración del registro. Así, se ponen en línea los datos restaurados.

4.6. Resultados

A continuación, se presenta el código utilizado para realizar las pruebas de copia de seguridad de la base de datos, y la restauración de dichas copias de seguridad. También, se presentan capturas de pantalla que muestran los resultados obtenidos.

-- 1 Verificar el modo de recuperación de la base de datos

use base_consortio

```
SELECT name, recovery_model_desc  
FROM sys.databases  
WHERE name = 'base_consortio';
```

-- 2 cambiamos el modo de recuperacion

```
USE master; -- Asegúrate de estar en el contexto de la base de datos master  
ALTER DATABASE base_consortio  
SET RECOVERY FULL;
```

-- 3 Realizamos backup de la base de datos


```
BACKUP DATABASE base_consortio
TO DISK = 'C:\backup\consorcio_backup.bak'
WITH FORMAT, INIT;
```

-- Agregamos 10 registros

```
select * from gasto;
```

```
insert into gasto
(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)
values (1,1,1,5,GETDATE(),5,1200);
```

```
insert into gasto
(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)
values (1,2,2,5,GETDATE(),2,1630);
```

```
insert into gasto
(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)
values (3,20,2,2,GETDATE(),4,500);
```

```
insert into gasto
(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)
values (5,3,1,3,GETDATE(),3,1520);
```

```
insert into gasto
(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)
values (5,12,3,4,GETDATE(),5,1120);
```

```
insert into gasto
(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)
values (6,45,2,4,GETDATE(),4,2000);
```

```
insert into gasto
(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)
values (14,36,2,2,GETDATE(),1,1740);
```

```
insert into gasto
(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)
values (18,3,1,2,GETDATE(),2,1520);
```

```
insert into gasto
(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)
values (2,48,1,5,GETDATE(),2,1500);
```

insert into gasto

(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)

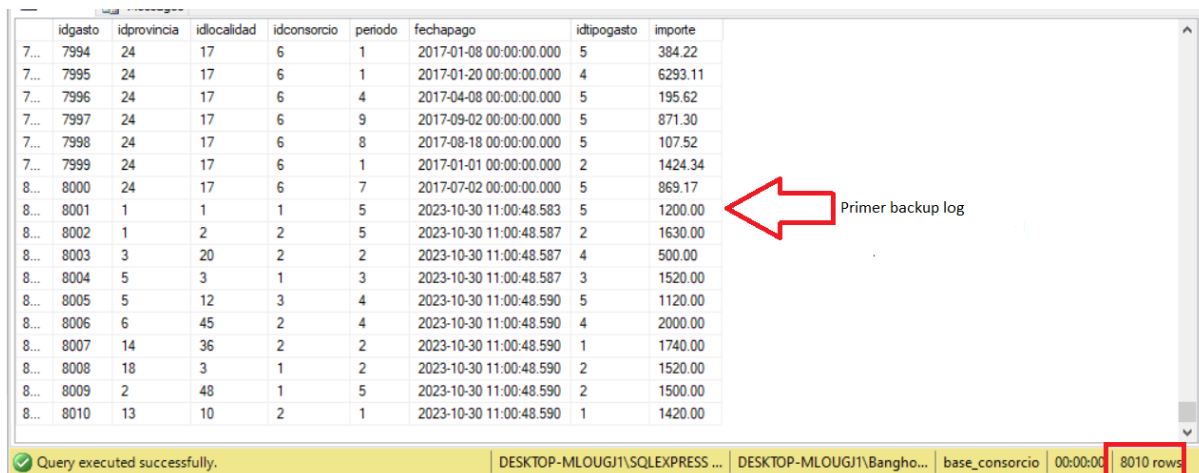
values (13,10,2,1,GETDATE(),1,1420);

-- 4 Realizamos backup del log de la base de datos

BACKUP LOG base_consortio

TO DISK = 'C:\backup\LogBackup.trn'

WITH FORMAT, INIT;



	idgasto	idprovincia	idlocalidad	idconsorcio	periodo	fechapago	idtipogasto	importe
7...	7994	24	17	6	1	2017-01-08 00:00:00.000	5	384.22
7...	7995	24	17	6	1	2017-01-20 00:00:00.000	4	6293.11
7...	7996	24	17	6	4	2017-04-08 00:00:00.000	5	195.62
7...	7997	24	17	6	9	2017-09-02 00:00:00.000	5	871.30
7...	7998	24	17	6	8	2017-08-18 00:00:00.000	5	107.52
7...	7999	24	17	6	1	2017-01-01 00:00:00.000	2	1424.34
8...	8000	24	17	6	7	2017-07-02 00:00:00.000	5	869.17
8...	8001	1	1	1	5	2023-10-30 11:00:48.583	5	1200.00
8...	8002	1	2	2	5	2023-10-30 11:00:48.587	2	1630.00
8...	8003	3	20	2	2	2023-10-30 11:00:48.587	4	500.00
8...	8004	5	3	1	3	2023-10-30 11:00:48.587	3	1520.00
8...	8005	5	12	3	4	2023-10-30 11:00:48.590	5	1120.00
8...	8006	6	45	2	4	2023-10-30 11:00:48.590	4	2000.00
8...	8007	14	36	2	2	2023-10-30 11:00:48.590	1	1740.00
8...	8008	18	3	1	2	2023-10-30 11:00:48.590	2	1520.00
8...	8009	2	48	1	5	2023-10-30 11:00:48.590	2	1500.00
8...	8010	13	10	2	1	2023-10-30 11:00:48.590	1	1420.00

Query executed successfully. | DESKTOP-MLOUGJ1\SQLEXPRESS ... | DESKTOP-MLOUGJ1\Bangho... | base_consortio | 00:00:00 | 8010 rows

-- Insertamos 10 registros más

select * from gasto

insert into gasto

(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)

values (1,1,1,5,GETDATE(),5,1200);

insert into gasto

(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)

values (2,48,1,5,GETDATE(),2,1500);

insert into gasto

(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)

values (3,16,1,2,GETDATE(),4,2300);

insert into gasto

(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)

```
values (4,21,1,3,GETDATE(),3,1000);
```

```
insert into gasto
```

```
(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)  
values (5,3,1,2,GETDATE(),5,1500);
```

```
insert into gasto
```

```
(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)  
values (6,45,2,4,GETDATE(),4,2000);
```

```
insert into gasto
```

```
(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)  
values (8,17,2,2,GETDATE(),1,1300);
```

```
insert into gasto
```

```
(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)  
values (9,14,1,3,GETDATE(),2,1700);
```

```
insert into gasto
```

```
(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)  
values (12,7,4,2,GETDATE(),3,2100);
```

```
insert into gasto
```

```
(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)  
values (13,10,2,1,GETDATE(),1,1100);
```

--5 Realizamos backup del log en otra ubicacion

```
BACKUP LOG base_consorcio  
TO DISK = 'C:\backup\logs\LogBackup2.trn'  
WITH FORMAT, INIT;
```

--6 Restauramos el backup de la base de datos

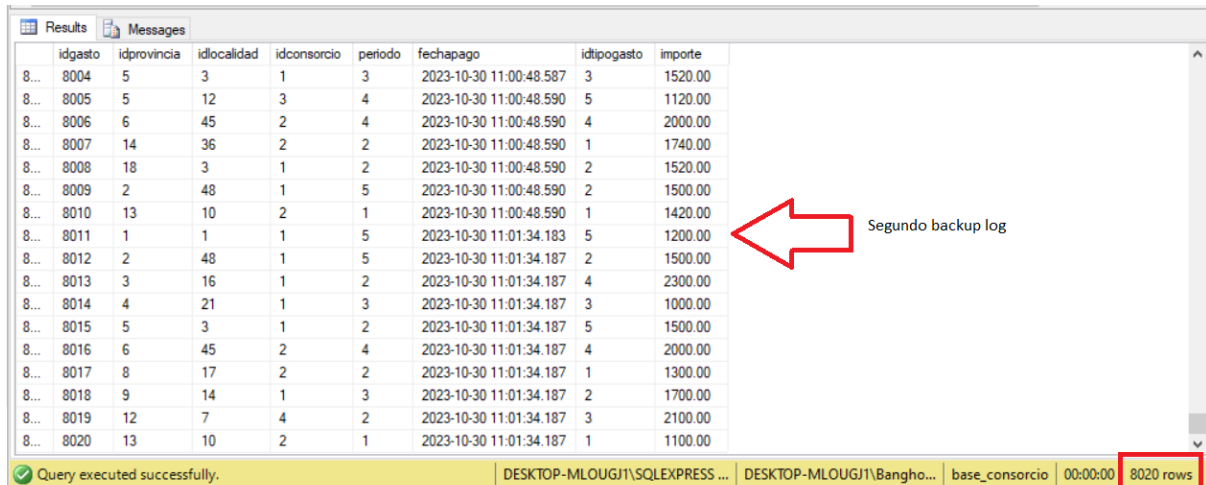
```
use master
```

```
RESTORE DATABASE base_consorcio  
FROM DISK = 'C:\backup\consorcio_backup.bak'  
WITH REPLACE, NORECOVERY;
```

```
RESTORE LOG base_consorcio  
FROM DISK = 'C:\backup\LogBackup.trn'  
WITH RECOVERY;
```

-- Segundo log

```
RESTORE LOG base_consortio
FROM DISK = 'C:\backup\logs\LogBackup2.trn'
WITH RECOVERY;
```



	idgasto	idprovincia	idlocalidad	idconsorcio	periodo	fechapago	idtipogasto	importe
8...	8004	5	3	1	3	2023-10-30 11:00:48.587	3	1520.00
8...	8005	5	12	3	4	2023-10-30 11:00:48.590	5	1120.00
8...	8006	6	45	2	4	2023-10-30 11:00:48.590	4	2000.00
8...	8007	14	36	2	2	2023-10-30 11:00:48.590	1	1740.00
8...	8008	18	3	1	2	2023-10-30 11:00:48.590	2	1520.00
8...	8009	2	48	1	5	2023-10-30 11:00:48.590	2	1500.00
8...	8010	13	10	2	1	2023-10-30 11:00:48.590	1	1420.00
8...	8011	1	1	1	5	2023-10-30 11:01:34.183	5	1200.00
8...	8012	2	48	1	5	2023-10-30 11:01:34.187	2	1500.00
8...	8013	3	16	1	2	2023-10-30 11:01:34.187	4	2300.00
8...	8014	4	21	1	3	2023-10-30 11:01:34.187	3	1000.00
8...	8015	5	3	1	2	2023-10-30 11:01:34.187	5	1500.00
8...	8016	6	45	2	4	2023-10-30 11:01:34.187	4	2000.00
8...	8017	8	17	2	2	2023-10-30 11:01:34.187	1	1300.00
8...	8018	9	14	1	3	2023-10-30 11:01:34.187	2	1700.00
8...	8019	12	7	4	2	2023-10-30 11:01:34.187	3	2100.00
8...	8020	13	10	2	1	2023-10-30 11:01:34.187	1	1100.00

Query executed successfully. DESKTOP-MLOUGJ1\SQLEXPRESS ... DESKTOP-MLOUGJ1\Bangho... base_consortio 00:00:00 8020 rows

5 - Implementación de Temas

5.1 Tema: Optimización de consultas mediante índices

Introducción

La optimización de consultas a través de índices es una técnica utilizada en las bases de datos para acelerar la velocidad de las consultas. Los índices son usados para encontrar rápidamente los registros que tengan un determinado valor en alguna de sus columnas. Sin un índice, el sistema de gestión de bases de datos tiene que iniciar con el primer registro y leer a través de toda la tabla para encontrar los registros relevantes.

Marco conceptual

Para poder entender por qué las consultas basadas en índices pueden, en ciertos casos, optimizar el rendimiento a la hora de realizar consultas en la base de datos, es necesario primero que establezcamos las diferencias entre las consultas basadas en índices y las consultas sin índices.

Las consultas basadas en índices y las consultas sin índices son fundamentalmente las mismas en términos de la sintaxis SQL. La diferencia radica en cómo el sistema de gestión de bases de datos (DBMS) procesa estas consultas.

Cuando se realiza una consulta en una tabla sin índices, el DBMS tiene que realizar una búsqueda completa de la tabla, también conocida como búsqueda secuencial. Esto significa que tiene que revisar cada fila de la tabla para encontrar las filas que cumplen con la condición de la consulta. Esto llega a ser muy lento si la tabla tiene muchas filas.

Por otro lado, cuando se realiza una consulta en una tabla con un índice en la columna que se está buscando, el DBMS puede utilizar el índice para encontrar las filas que cumplen con la condición de tu consulta de manera mucho más rápida. En lugar de tener que buscar en todas las filas de la tabla, puede buscar directamente en el índice, que está diseñado para permitir búsquedas rápidas, y luego recuperar sólo las filas correspondientes de la tabla. Esto se conoce como búsqueda indexada.

Por lo tanto, la principal diferencia entre las consultas basadas en índices y las consultas sin índices es la velocidad a la que se pueden ejecutar. Las consultas basadas en índices suelen ser mucho más rápidas para las tablas grandes, pero requieren que se mantenga un índice, lo cual puede ralentizar las operaciones de escritura y ocupar espacio adicional en disco.

Tenemos como objetivo, entonces, explicar mediante un ejemplo práctico proporcionado por la cátedra el funcionamiento de este tipo de consultas y su impacto en el rendimiento.

¿Cómo funcionan los índices?

Los índices funcionan de manera similar al índice de un libro, guardando parejas de elementos: el elemento que se desea indexar y su posición en la base de datos. Para buscar un elemento que esté indexado, sólo hay que buscar en el índice dicho elemento para, una vez encontrado, devolver un registro que se encuentre en la posición marcada por el índice.

Tipos de Índices

Existen varios tipos de índices que se pueden utilizar en función de las necesidades que se presenten. Los más comunes son:

- Índice agrupado (CLUSTERED): Este índice determina el orden físico de los registros en la tabla. Cada tabla puede tener solo un índice agrupado porque define la estructura de almacenamiento de la tabla misma. Los datos de la tabla se almacenan en el orden especificado por el índice agrupado. Por lo tanto, estos son útiles para consultas que recuperan un rango de datos ordenados de manera específica.

- Índice no agrupado (NONCLUSTERED): Los índices no agrupados son adicionales a la tabla principal y no afectan el orden físico de los registros en la tabla. Puedes tener varios índices no agrupados en una tabla. Estos índices contienen copias de una o varias columnas de la tabla con un puntero a la fila correspondiente. Son adecuados para acelerar consultas de búsqueda y facilitar la selección de datos basados en valores en las columnas indexadas.

- Índice Compuesto: Un índice compuesto se crea en múltiples columnas. Puede ser útil para mejorar el rendimiento de las consultas que involucran una combinación de valores de esas columnas.

- Índices únicos: Este tipo de índice garantiza que los valores en la columna indexada sean únicos en la tabla. Se utiliza para hacer cumplir restricciones de unicidad en una columna específica.

- Índices textuales: Se utilizan para realizar consultas que involucran la extracción de información textual de la base de datos.

Ventajas y Desventajas

- Ventajas: De ser utilizados correctamente, los índices pueden mejorar el rendimiento de la siguiente manera:

- Rapidez: Los índices permiten una mayor rapidez en la ejecución de las consultas y recuperación de datos.
- Eficiencia: Los índices mejoran la eficiencia al permitir un acceso más rápido a las filas de una tabla.

- Desventajas: Los índices deben ser equilibrados y ser creados a partir de un análisis de las consultas más frecuentes, de lo contrario pueden generar:

- Mayor complejidad: El utilizar y administrar adecuadamente una gran cantidad de base de datos distribuidos en cientos de nodos es una tarea muy compleja.

- Consumo de recursos: Los índices también consumen recursos, tanto de almacenamiento como de procesamiento, sobre todo si se utilizan de manera incorrecta

Elección de columnas a indexar

La elección de las columnas a indexar en una base de datos es un aspecto crítico para lograr un rendimiento eficiente en las consultas. Indexar las columnas adecuadas puede acelerar la recuperación de datos, pero indexar en exceso o de manera inapropiada puede tener efectos negativos en el rendimiento general de la base de datos. Aquí hay algunas pautas para elegir las columnas a indexar:

1. Columnas de Frecuente Búsqueda: Las columnas que se utilizan con frecuencia en cláusulas WHERE en consultas de búsqueda son candidatas ideales para la indexación. Esto incluye columnas que se utilizan en condiciones de igualdad o comparaciones de rango.

2. Columnas de Unicidad: Las columnas que deben contener valores únicos, como identificadores, son buenos candidatos para índices únicos. Esto garantiza que no haya duplicados en la columna y acelera la búsqueda por valor exacto.

3. Columnas de JOIN: Si se realizan operaciones de JOIN con frecuencia en tablas relacionadas, indexar las columnas involucradas en las condiciones de JOIN mejora significativamente el rendimiento.

4. Columnas de Ordenamiento: Columnas utilizadas en operaciones de ordenamiento (por ejemplo, en cláusulas ORDER BY) deben considerarse para la indexación, ya que acelerarán estas operaciones.

5. Columnas Filtradas: En algunos casos, es útil crear índices en columnas calculadas o filtradas. Esto permite optimizar consultas basadas en cálculos específicos.

6. Columnas de Texto Completo: Si tienes columnas de texto en las que se realizan búsquedas de texto completo, considera la creación de índices de texto completo para mejorar la velocidad de estas consultas.

7. Columnas de Filtros Comunes: Si se tienen columnas que se utilizan comúnmente para filtrar datos, es importante indexarlas para acelerar la recuperación de registros.

8. Columnas con alta Cardinalidad: Las columnas con un alto número de valores distintos (alta cardinalidad) suelen beneficiarse de la indexación, ya que reducen la cantidad de registros que deben ser explorados.

9. Monitoreo y Ajuste: Realiza un seguimiento del rendimiento de las consultas y ajusta los índices según sea necesario. La creación de índices innecesarios puede ralentizar las operaciones de escritura.

Es importante tener en cuenta que no es necesario (y, a veces, es contraproducente) indexar todas las columnas en una tabla. La elección de las columnas a indexar debe basarse en las consultas y patrones de acceso a los datos reales de la aplicación. Además, el mantenimiento de índices es esencial, ya que los índices deben actualizarse cuando se realizan cambios en los datos. El equilibrio entre el rendimiento de las consultas y el costo de mantenimiento de los índices es fundamental.

Funcionamiento del motor de base de datos

En esta sección, exploraremos en detalle cómo funciona el motor de base de datos al utilizar índices agrupados y no agrupados en el contexto de consultas SQL.

Índices No Agrupados (Non-Clustered Index)

1. Búsqueda en el Índice No Agrupado:

- En el caso de un índice no agrupado, el motor de base de datos accede al índice no agrupado correspondiente.

- Busca directamente la clave de búsqueda en el índice no agrupado.

2. Ubicación de la Clave de Búsqueda:

- Una vez que se encuentra la clave de búsqueda en el índice no agrupado, el motor obtiene uno o más identificadores de filas (punteros) que corresponden a esa clave en la tabla principal.

3. Recuperación de Datos Parciales:

- Con los identificadores de filas obtenidos del índice no agrupado, el motor accede a la tabla principal.
- Recupera solo las columnas necesarias que cumplen con los criterios de la consulta, en lugar de todas las columnas de la tabla.

4. Entrega de Resultados:

- Los registros parciales que cumplen con los criterios de la consulta se entregan como resultado de la consulta. Índices

Agrupados (Clustered Index)

1. Búsqueda en el Índice Agrupado:

- Cuando se ejecuta una consulta, el motor de base de datos accede al índice agrupado correspondiente.
- Busca directamente la clave de búsqueda (por ejemplo, un valor de columna) en el índice agrupado.

2. Ubicación del Registro:

- Una vez que se encuentra la clave de búsqueda en el índice agrupado, el motor obtiene el identificador del registro en la tabla principal.
- Este identificador generalmente es un puntero que apunta al registro físico en la tabla.

3. Recuperación de Datos Completos:

- Con el identificador del registro, el motor accede a la tabla principal y recupera el registro completo que cumple con los criterios de la consulta.
- Esto incluye todas las columnas del registro.

4. Entrega de Resultados:

- Los registros que cumplen con los criterios de la consulta se entregan como resultado de la consulta.

5.2 Tema: Manejo de Permisos a Nivel de usuarios de base datos

Este tema consiste crear usuarios para la base de datos a los que puede otorgar distintos roles que definen las funciones a las que tienen acceso o que pueden usar para navegar o trabajar en la base de datos.

En nuestro caso, creamos dos tipos de usuarios, el primero con el rol definido con la sentencia “db_datareader”, lo cual permite al usuario únicamente leer las bases de datos y sus tablas, no tienen acceso a nada más. Aunque también se le asigna un rol más, “db_backupoperator”, lo cual le permite realizar backups de la base de datos, aunque solamente eso, no puede restaurarlas.

En el segundo caso, creamos un usuario definido por la sentencia “db_ddladmin”, el cual tiene acceso absolutamente a todas las funciones de la base de datos, tal como se asemeja la sentencia, es el “administrador” aunque no confundir con el dueño real de la base de datos:

```

create login manuel with password='Password123';
create login juan with password='Password123';

--Se crea los usuarios con los login anteriores
create user manuel for login manuel
create user juan for login juan

--Se asignan los roles a los usuarios
alter role db_datareader add member manuel;
alter role db_ddladmin add juan;

exec sp_addrolemember 'db_datareader','manuel';
exec sp_addrolemember 'db_backupoperator','manuel';

exec sp_addrolemember 'db_ddladmin','juan';

```

Otros tipos de roles que existen (Conocidos como roles fijos):

- **db_owner:** Los miembros del rol fijo de base de datos db_owner pueden realizar todas las actividades de configuración y mantenimiento en la base de datos, y también pueden drop la base de datos en SQL Server. (En SQL Database y Azure Synapse, algunas actividades de mantenimiento requieren permisos a nivel de servidor y no se pueden realizar por db_owners).
- **db_securityadmin:** Los miembros del rol fijo de base de datos db_securityadmin pueden modificar la pertenencia a roles únicamente para roles personalizados y administrar permisos. Los miembros de este rol pueden elevar potencialmente sus privilegios y se deben supervisar sus acciones.
- **db_accessadmin:** Los miembros del rol fijo de base de datos db_accessadmin pueden agregar o eliminar el acceso a la base de datos para inicios de sesión de Windows, grupos de Windows e inicios de sesión de SQL Server.
- **db_backupoperator:** Los miembros del rol fijo de base de datos db_backupoperator pueden crear copias de seguridad de la base de datos.
- **db_ddladmin:** Los miembros del rol fijo de base de datos db_ddladmin pueden ejecutar cualquier comando del lenguaje de definición de datos (DDL) en una base de datos. Los miembros de este rol pueden potencialmente

aumentar sus privilegios manipulando código que puede ser ejecutado bajo altos privilegios y sus acciones se deben supervisar.

- **db_datawriter:** Los miembros del rol fijo de base de datos db_datawriter pueden agregar, eliminar o cambiar datos en todas las tablas de usuario. En la mayoría de los casos de uso, este rol se combinará con la membresía db_datareader para permitir la lectura de los datos que se van a modificar.
- **db_datareader:** Los miembros del rol fijo de base de datos db_datareader pueden leer todos los datos de todas las tablas y vistas de usuario. Los objetos de usuario pueden existir en cualquier esquema, excepto sys e INFORMATION_SCHEMA.
- **db_denydatawriter:** Los miembros del rol fijo de base de datos db_denydatawriter no pueden agregar, modificar ni eliminar datos de tablas de usuario de una base de datos.
- **db_denydatareader:** Los miembros del rol fijo de base de datos db_denydatareader no pueden leer datos de las tablas y vistas de usuario dentro de una base de datos.

Luego tenemos un par de roles mas especiales:

Estos roles y permisos se aplican principalmente a la base de datos "master" y suelen estar relacionados con la administración y configuración del servidor.

- **dbmanager:** Puede crear y eliminar bases de datos, convirtiéndose en el propietario. Tiene todos los permisos en las bases de datos que crea, pero no en otras.
- **db_exporter:** Aplicable a grupos de SQL dedicados de Azure Synapse Analytics. Permite actividades de exportación de datos con permisos como CREATE TABLE, ALTER ANY SCHEMA, ALTER ANY EXTERNAL DATA SOURCE, y ALTER ANY EXTERNAL FILE FORMAT.
- **loginmanager:** Puede crear y eliminar inicios de sesión en la base de datos "master" virtual.

5.3 Tema: Transacciones y transacciones anidadas

Una transacción en base de datos es una secuencia de operaciones que se realizan de manera indivisible, operaciones agrupadas como una unidad. Es como un paquete que contiene múltiples acciones que deben completarse en su totalidad o deshacerse por completo. Las operaciones que contiene una transacción se van almacenando temporalmente, no a nivel de disco. Es hasta que termina la transacción que se tienen efecto de manera permanente o no. Esto asegura que los datos se mantengan consistentes y evita problemas en caso de errores o fallos en el sistema.

Una transacción aplica a datos recuperables, puede estar formada por operaciones simples o compuestas y su intención es que sea atómica.

Una transacción siempre termina, aun en la presencia de fallas. Si una transacción termina de manera exitosa se dice que la transacción hace un commit (consumación). Si la transacción se detiene sin terminar su tarea, se dice que la transacción aborta. Cuando la transacción es abortada, su ejecución se detiene y todas las acciones ejecutadas hasta el momento se deshacen (undone) regresando a la base de datos al estado antes de su ejecución. A esta operación también se le conoce como rollback.

Estados de una transacción

- Transacción Activa: se encuentra en este estado justo después de iniciar su ejecución.
- Transacción Parcialmente Confirmada: en este punto, se han realizado las operaciones de la transacción pero no han sido almacenados de manera permanente.
- Transacción Confirmada: Ha concluido su ejecución con éxito y se almacenan de manera permanente.
- Transacción Fallida: En este caso, es posible que la transacción deba ser cancelada.
- Transacción Terminada: indica que la transacción ha abandonado el sistema.

Tipos de Transacciones:

- Transacciones planas: Estas transacciones tienen un punto de partida simple (Begin_transaction) y un punto simple de terminación (End_transaction).

```

Begin_transaction RESERVAR
begin
    EXEC SQL  UPDATE  cuentas  SET saldo=saldo-1000 WHERE id_cuenta=1111
    EXEC SQL  UPDATE  cuentas  SET saldo=saldo+1000 WHERE id_cuenta=222
end

```

- Transacciones anidadas: las operaciones de una transacción anidada pueden incluir otras transacciones.

```

BeginTransaction Reservación
    BeginTransaction Vuelo
    ...
    EndTransaction {Vuelo}
    BeginTransaction Hotel
    ...
    endTransaction {Hotel}
    BeginTransaction Car
    ...
    endTransaction {Car}

EndTransaction {Reservación}

```

Existen restricciones para una transacción anidada:

- Debe empezar después que su padre y debe terminar antes que él.
- El commit de una transacción padre está condicionada al commit de sus transacciones hijas.
- Si alguna transacción hija aborta (rollback), la transacción padre también será abortada (rollback).

5.4 Tema: Triggers de auditoría.

La gestión efectiva de bases de datos implica no sólo la garantía de integridad y consistencia de los datos, sino también la capacidad de rastrear y auditar cambios realizados en las tablas clave.

La auditoría en bases de datos se convierte en un componente esencial para garantizar la transparencia, seguridad y responsabilidad en el manejo de la información.

Este informe aborda la implementación de funciones de auditoría en tres tablas críticas de una base de datos relacional: administrador, conserje, y consorcio.

En la implementación de sistemas de bases de datos, los triggers (desencadenadores) desempeñan un papel crucial al permitir la ejecución automática de acciones en respuesta a eventos específicos en las tablas.

Los triggers, o desencadenadores, son objetos de base de datos que responden automáticamente a ciertos eventos ocurridos en una tabla o vista. Estos eventos pueden ser operaciones de inserción (INSERT), actualización (UPDATE), eliminación (DELETE), o incluso eventos específicos del sistema.

Eventos Desencadenadores:

INSERT: Se activa después de que se inserta una nueva fila en la tabla.

UPDATE: Se activa después de que se actualiza una fila existente en la tabla.

DELETE: Se activa después de que se elimina una fila de la tabla.

Momento de Activación:

BEFORE: Se ejecuta antes de que se realice la operación principal (INSERT, UPDATE, DELETE). Permite modificar los datos antes de que se almacenen en la tabla.

AFTER: Se ejecuta después de que se completa la operación principal. Puede usarse para realizar acciones posteriores a la modificación de datos.

Uso común de Triggers:

Auditoría: Registrar cambios en una tabla para seguimiento y control.

Validaciones: Enforcing reglas de negocio o restricciones específicas antes de permitir ciertas operaciones.

Sincronización de datos: Mantener la consistencia entre tablas relacionadas.

Variables Especiales:

inserted y deleted: Tablas especiales que almacenan las filas afectadas por las operaciones (INSERT, UPDATE, DELETE). Se utilizan dentro de triggers para acceder a los datos antes y después de la operación.

Manejo de Errores y Transacciones:

Es crucial manejar errores dentro de los triggers para evitar problemas de integridad. Las transacciones permiten agrupar operaciones y realizar un rollback completo si algo falla.

Cuidado con el Rendimiento:

Los triggers pueden tener un impacto en el rendimiento, especialmente si realizan operaciones costosas. Es importante optimizar el código del trigger para minimizar su impacto en las operaciones regulares.

En el proyecto se utilizó triggers INSTEAD OF DELETE para evitar la eliminación no autorizada de registros en la tabla administrador y consorcio, y se implementó triggers AFTER INSERT, AFTER UPDATE, y AFTER DELETE para realizar el seguimiento de cambios en la tabla conserje. También para registrar información relevante en tablas de auditoría Auditoria_Administrador_Eliminacion, Auditoria_Consorcio. Además, el trigger AFTER INSERT en la tabla conserje registra información adicional en la tabla auditoria_conserje después de realizar una inserción.

Auditoría en la tabla Administrador

Objetivo: Evitar eliminaciones no autorizadas y realizar un seguimiento de las inserciones.

Implementación:

Tabla de Auditoría:

Se ha creado la tabla Auditoria_Administrador_Eliminacion para registrar la eliminación de registros en la tabla administrador. Esta tabla almacena información relevante, como el identificador del administrador, nombre, fecha de eliminación y el usuario que realizó la acción.

```
-- Crear la tabla de auditoría para registrar la eliminación de registros de la tabla administrador.  
CREATE TABLE Auditoria_Administrador_Eliminacion (  
    idadmin INT,  
    apeynom VARCHAR(50),  
    fechaEliminacion DATETIME,  
    usuario_accion VARCHAR(100)  
);
```

Trigger trg_administrador_no_borrar:

Se implementó un trigger INSTEAD OF DELETE en la tabla administrador. Este trigger evita la eliminación directa de registros y, en su lugar, realiza una inserción en la tabla de auditoría antes de lanzar un error y realizar un rollback de la transacción.

El manejo de errores y transacciones en los triggers es crucial para garantizar la integridad de los datos. En el trigger trg_administrador_no_borrar, se utiliza THROW y ROLLBACK TRANSACTION para manejar el caso de intento de eliminación no autorizada en la tabla administrador, lo cual es una buena práctica.


```

CREATE TRIGGER trg_administrador no borrar
ON administrador
INSTEAD OF DELETE
AS
BEGIN
    -- El trigger verifica si hay registros en la tabla deleted, que contiene los registros que están siendo eliminados
    IF (SELECT COUNT(*) FROM deleted) > 0
    BEGIN
        -- Si hay registros en deleted, el trigger realiza una inserción en la tabla
        -- Inserta información relevante en la tabla de auditoría Auditoria_Administrador_Eliminacion.
        INSERT INTO Auditoria_Administrador_Eliminacion (idadmin, nombre, fechaEliminacion, usuario accion)
        SELECT d.idadmin,
        d.apeynom, GETDATE(),
        SYSTEM_USER
        FROM deleted AS d;

        -- Después de la inserción en la tabla de auditoría, el trigger lanza un error (THROW) con el mensaje
        -- 'No se puede eliminar ningún registro de la tabla administrador' y realiza un rollback de la transacción.
        -- Esto significa que la operación de eliminación en la tabla administrador se revierte y no se completará.

        THROW 50000, 'No se puede eliminar ningún registro de la tabla administrador', 1;
        ROLLBACK TRANSACTION;
    END
    ELSE
    BEGIN
        -- Si no hay registros en deleted, se realiza un commit de la transacción.
        -- Esto confirma la eliminación de registros en la tabla administrador.
        COMMIT TRANSACTION;
    END
END;

```

Auditoría en la tabla Conserje

Objetivo: Registrar inserciones, modificaciones y eliminaciones.

Implementación:

Tabla de Auditoría:

Se creó la tabla auditoria_conserje para almacenar información adicional relacionada con la tabla conserje. Esta tabla incluye campos como el identificador del conserje, nombre, teléfono, fecha de nacimiento, estado civil, fecha de modificación y usuario que realizó la acción.

```

CREATE TABLE auditoria_conserje (
    idconserje INT,
    apeynom VARCHAR(50),
    tel VARCHAR(50),
    fechnac DATE,
    estciv VARCHAR(1),
    fechaModif DATETIME,
    usuario_accion VARCHAR(100)
);

```

trg_auditoria_conserje_insertar: Se activa después de una operación INSERT en la tabla conserje. Captura la información de la fila insertada y la registra en la tabla de auditoría.

```
-- Se crea un trigger que se activa automáticamente después de una operación INSERT en la tabla conserje.
CREATE TRIGGER trg_auditoria_conserje_insertar
ON conserje
FOR INSERT
AS
BEGIN
    DECLARE @idconserje INT, @apeynom VARCHAR(50), @tel VARCHAR(50), @fechnac DATE, @estciv VARCHAR(1),
            @fechaModif DATETIME, @usuario VARCHAR(50);

    -- Captura la información de la fila insertada utilizando la tabla inserted.
    SELECT
        @idconserje = idconserje,
        @apeynom = ApeyNom,
        @tel = tel,
        @fechnac = fechnac,
        @estciv = estciv,
        @fechaModif = GETDATE(),
        @usuario = SYSTEM_USER
    FROM inserted;

    -- Inserta esta información en la tabla auditoria_conserje junto con la fecha actual (fechaModif),
    -- el usuario del sistema actual (usuario_accion), y la acción realizada (accion).

    INSERT INTO auditoria_conserje VALUES (@idconserje, @apeynom, @tel, @fechnac, @estciv, @fechaModif, @usuario);
END;
```

trg_auditoria_conserje_modificar: Se activa después de una operación UPDATE en la tabla conserje. Captura la información de la fila actualizada y la registra en la tabla de auditoría.

```
CREATE TRIGGER trg_auditoria_conserje_modificar
ON conserje
FOR UPDATE
AS
BEGIN
    DECLARE @idconserje INT, @apeynom VARCHAR(50), @tel VARCHAR(50), @fechnac DATE, @estciv VARCHAR(1),
            @fechaModif DATETIME, @usuario VARCHAR(50), @accion VARCHAR(20);
    -- -- Captura la información de la fila actualizada utilizando la tabla inserted.
    SELECT @idconserje = idconserje,
        @apeynom = ApeyNom,
        @tel = tel,
        @fechnac = fechnac,
        @estciv = estciv,
        @fechaModif = GETDATE(),
        @usuario = SYSTEM_USER
    FROM inserted;

    -- Inserta esta información en la tabla auditoria_conserje junto con la fecha actual (fechaModif),
    -- el usuario del sistema actual (usuario_accion), y la acción realizada (accion).

    INSERT INTO auditoria_conserje VALUES (@idconserje, @apeynom, @tel, @fechnac, @estciv, @fechaModif, @usuario);
END;
```

trg_auditoria_conserje_borrar: Se activa después de una operación DELETE en la tabla conserje. Captura la información de la fila eliminada y la registra en la tabla de auditoría.

```

CREATE TRIGGER trg_auditoria_conserje_borrar
ON conserje
FOR DELETE
AS
BEGIN
    DECLARE @idconserje INT, @apeynom VARCHAR(50), @tel VARCHAR(50), @fechnac DATE, @estciv VARCHAR(1),
            @fechaModif DATETIME, @usuario VARCHAR(50);

    -- -- Captura la información de la fila eliminada utilizando la tabla deleted.
    SELECT @idconserje = idconserje,
           @apeynom = ApeyNom,
           @tel = tel,
           @fechnac = fechnac,
           @estciv = estciv,
           @fechaModif = GETDATE(),
           @usuario = SYSTEM_USER
    FROM deleted;

    -- Inserta esta información en la tabla auditoria_conserje junto con la fecha actual (fechaModif),
    -- el usuario del sistema actual (usuario_accion), y la acción realizada (accion).
    INSERT INTO auditoria_conserje VALUES (@idconserje, @apeynom, @tel, @fechnac, @estciv, @fechaModif, @usuario);

END;

```

Auditoría en la tabla Consorcio

Objetivo: Realizar un seguimiento de eliminaciones en la tabla consorcio.

Implementación:

Tabla de Auditoría:

Se creó la tabla Auditoria_Conorcio para registrar información sobre eliminaciones en la tabla consorcio.

```

] BEGIN
] CREATE TABLE Auditoria_Conorcio (
    idauditoria INT IDENTITY PRIMARY KEY,
    idprovincia INT,
    idlocalidad INT,
    idconsorcio INT,
    nombre VARCHAR(50),
    direccion VARCHAR(250),
    idzona INT,
    idconserje INT,
    idadmin INT,
    fechaAccion DATETIME,
    UsuarioAccion VARCHAR(50)
);
- END
-

```

Trigger trg_auditoria_conorcio:

Se implementó un trigger INSTEAD OF DELETE en la tabla consorcio. Captura la información de la fila eliminada y la registra en la tabla de auditoría antes de completar la operación de eliminación.

```

CREATE TRIGGER trg_auditoria_conorcio
ON consorcio
--FOR DELETE
INSTEAD OF DELETE
AS
BEGIN
    DECLARE @usuario VARCHAR(50) --variable @usuario para almacenar el nombre del usuario del sistema.
    SELECT @usuario = SYSTEM_USER

    IF EXISTS (SELECT 1 FROM deleted) -- Comprueba si hay registros en la tabla deleted (los registros que se están eliminando).
        -- La tabla deleted contiene las filas que están siendo eliminadas como resultado de la operación DELETE.
    BEGIN
        -- Si existen registros, inserta información relevante en la tabla Auditoria_Conorcio
        -- Si hay registros en deleted, realiza una inserción en la tabla de auditoría Auditoria_Conorcio.
        -- Las columnas seleccionadas para la inserción provienen de la tabla deleted (las filas que están siendo eliminadas).
        INSERT INTO Auditoria_Conorcio (idprovincia, idlocalidad, idconsorcio, nombre, direccion, idzona, idconserje, idadmin, fechaAccion, UsuarioAccion)
        SELECT d.idprovincia, d.idlocalidad, d.idconsorcio, d.nombre, d.direccion, d.idzona, d.idconserje, d.idadmin, GETDATE(), @usuario
        FROM deleted AS d;
    END
END

```

5.5 Tema: Procedimientos y funciones de almacenado

Procedimientos almacenados

Un procedimiento almacenado es un grupo de una o varias instrucciones Transact-SQL o una referencia a un método de Common Runtime Language (CLR) de Microsoft .NET Framework. Los procedimientos se asemejan a las construcciones de otros lenguajes de programación, porque pueden:

- Aceptar parámetros de entrada y devolver varios valores en forma de parámetros de salida al programa que realiza la llamada.
- Contener instrucciones de programación que realicen operaciones en la base de datos. Pueden contener llamadas a otros procedimientos.
- Devolver un valor de estado a un programa que realiza una llamada para indicar si la operación se ha realizado correctamente o se han producido errores, y el motivo de estos.

Ventajas de usar procedimientos almacenados

- Tráfico de red reducido entre el cliente y el servidor
- Mayor seguridad
- Reutilización del código
- Mantenimiento más sencillo
- Rendimiento mejorado.

Tipos de procedimientos almacenados

- Definidos por el usuario
- Temporales
- Sistema
- Extendidos definidos por el usuario

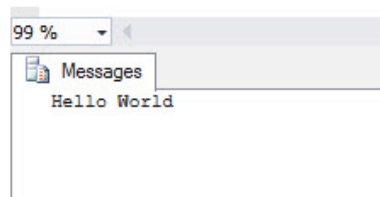
Estructura de un procedimiento en SQLServer

```
CREATE PROCEDURE HelloWorldprocedure
AS
PRINT 'Hello World'
```

Ejecute el código y después llame al procedimiento almacenado en SQL:

```
exec HelloWorldprocedure
```

Si ejecuta el código, podrá ver el mensaje "Hello World":



Funciones almacenadas

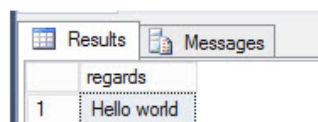
Las funciones almacenadas comparten similitudes con los procedimientos almacenados, pero se centran en devolver una variable y no múltiple como los procedimientos. Son de estructura y funcionalidad más rígida y admiten menos cláusulas y funcionalidades que los procedimientos.

Estructura de una función en SQLServer

```
CREATE FUNCTION dbo.helloworldfunction()
RETURNS varchar(20)
AS
BEGIN
    RETURN 'Hello world'
END
```

Podemos llamar a la función utilizando un select:

La función devolverá el siguiente mensaje:



Implementación y Resultados

Procedimientos para Consorcio

- Agregar un nuevo consorcio

Código implementado:

```

88 GO
89
90 CREATE PROCEDURE agregarConsortio
91     @Provincia INT,
92     @Localidad INT,
93     @Consortio INT,
94     @Nombre VARCHAR(50),
95     @Direccion VARCHAR(255),
96     @Zona INT,
97     @Conserje INT,
98     @Admin INT
99
100 AS
101 BEGIN
102     INSERT INTO consorcio(idprovincia,idlocalidad,idconsorcio,nombre,direccion, idzona,idconserje,idadmin)
103     VALUES (@Provincia,@Localidad,@Consortio,@Nombre,@Direccion, @Zona, @Conserje, @Admin)
104
105 END

```

Ejecutamos el procedimiento

```

163 --Agregar un nuevo consorcio
164 EXEC agregarConsortio @Provincia = 1, @Localidad = 3, @Consortio = 1, @Nombre = "EDIFICIO-1000",
165 @Direccion = "Av. Corrientes 1257", @Zona = 1, @Conserje = 29, @Admin = 3;

```

Resultado:

Se puede observar que se ha agregado el consorcio correctamente, indicado con la flecha roja. Entre los resultados se encuentra en el puesto cinco.

Results								
	idprovincia	idlocalidad	idconsorcio	nombre	direccion	idzona	idconserje	idadmin
4	8	31	4	EDIFICIO-8314	LISANDRO SEGOVIA N° 1924	1	75	26
5	10	18	4	EDIFICIO-333	Av. Fascio 1543	1	32	10
6	11	26	4	EDIFICIO-11264	PASAJE DODERO N° 2440	5	59	42
7	12	7	4	EDIFICIO-1274	AEROPUERTO P. NIVEYRO 250VIVIENDAS, MZ I N° 21	2	54	47
8	14	42	4	EDIFICIO-14424	PEDRO NUMA SOTO (CONT. PARAGUAY) N° 1640, 1º pis...	2	45	56
9	15	13	4	EDIFICIO-15134	SAN MARTIN N° 2710	1	40	61

- Modificar datos de un consorcio existente

Código implementado

```

110 CREATE PROCEDURE modificarConsortio
111     @Provincia INT,
112     @Localidad INT,
113     @Consortio INT,
114     @Nombre VARCHAR(50),
115     @Direccion VARCHAR(255),
116     @Zona INT,
117     @Conserje INT,
118     @Admin INT
119
120 AS
121 BEGIN
122     IF EXISTS (
123         SELECT 1 FROM consorcio WHERE idprovincia = @Provincia AND idlocalidad = @Localidad AND idconsorcio = @Consortio
124     )
125     BEGIN
126         UPDATE consorcio
127         SET nombre = @Nombre,
128             direccion = @Direccion,
129             idzona = @Zona,
130             idconserje = @Conserje,
131             idadmin = @Admin
132         WHERE
133             idprovincia = @Provincia
134             AND idlocalidad = @Localidad
135             AND idconsorcio = @Consortio;
136     END
137     ELSE
138     BEGIN
139         -- Si no existe la combinación de claves primarias, puedes manejarlo según tus necesidades.
140         -- Puedes optar por no hacer nada, insertar un nuevo registro o lanzar un mensaje de error.
141         -- En este ejemplo, no se hace nada en caso de que no exista la combinación de claves primarias.
142         PRINT 'No se encontró el consorcio con las claves primarias proporcionadas.';
143     END
144 END
145

```

Ejecutamos el procedimiento

```

167 --Modificar un consorcio ya existente
168 EXEC modificarConsortio @Consortio = 4, @Provincia = 10, @Localidad = 18, @Nombre = "ERNESTO III",
169 @Direccion = "Av. Cordoba 2054", @Zona = 2, @Conserje = 33, @Admin = 13;
170

```

Resultado

Siguiendo con el ejemplo anterior, modificamos los datos nombre, dirección, zona, conserje y administrador del edificio, el procedimiento se ejecuta correctamente.

	idprovincia	idlocalidad	idconsorcio	nombre	direccion	idzona	idconserje	idadmin
3	6	106	4	EDIFICIO-61064	PEDRO ESNAOLA (CAMPUS) N° 5328	4	80	21
4	8	31	4	EDIFICIO-8314	LISANDRO SEGOVIA N° 1924	1	75	26
5	10	18	4	ERNESTO III	Av. Cordoba 2054	2	33	13
6	11	26	4	EDIFICIO-11264	PASAJE DODERO N° 2440	5	59	42
7	12	7	4	EDIFICIO-1274	AEROPUERTO P. NIVEYRO 250VIVIENDAS, MZ I N° 21	2	54	47
8	14	42	4	EDIFICIO-14424	PEDRO NIIMA SOTO (CONT. PARAGUAY) N° 1640 18 piso	2	46	66

- Eliminar consorcio

Código implementado


```

148 GO
149
150 CREATE PROCEDURE eliminarConsortorio
151 @Provincia INT,
152 @Localidad INT,
153 @Consortorio INT
154
155 AS
156 BEGIN
157     DELETE FROM consorcio WHERE idprovincia = @Provincia AND idLocalidad = @Localidad AND idconsorcio = @Consortorio
158 END
159
160

```

Ejecutamos el procedimiento

```

171 --Eliminar un consorcio existente
172 EXEC eliminarConsortorio @Provincia = 10, @Localidad = 18, @Consortorio = 4;
173

```

Resultado:

El código se ejecuta de manera exitosa, borrando así el consorcio anteriormente creado. Observe que ahora el puesto cinco es ocupado por el consorcio que le seguía al consorcio eliminado.

	idprovincia	idlocalidad	idconsorcio	nombre	direccion	idzona	idconserje	idadmin
1	3	27	4	EDIFICIO-3274	MARIANO MORENO Nº 1626, 2º piso, Dpto D	4	93	8
2	5	12	4	EDIFICIO-5124	FRANKLIN Nº 524, -º piso, Dpto -, Unidad -	6	85	16
3	6	106	4	EDIFICIO-61064	PEDRO ESNAOLA (CAMPUS) Nº 5328	4	80	21
4	8	31	4	EDIFICIO-8314	LISANDRO SEGOVIA Nº 1924	1	75	26
5	11	26	4	EDIFICIO-11264	PASAJE DODERO Nº 2440	5	59	42
6	12	7	4	EDIFICIO-1274	AEROPUERTO P. NIVEYRO 250VIVIENDAS, MZ I Nº 21	2	54	47
7	14	42	4	EDIFICIO-14424	PEDRO NUMA SOTO (CONT. PARAGUAY) Nº 1640, 1º pis...	2	45	56
8	15	12	4	EDIFICIO-15124	SAN MARTIN Nº 2710	1	40	61

Procedimientos para Administrador

- Insertar nuevo Administrador

Código implementado

```

]CREATE PROCEDURE InsertarAdministrador
(
    @apeynom varchar(50),
    @viveahi varchar(1),
    @tel varchar(20),
    @sexo varchar(1),
    @fechnac datetime
)
AS
]BEGIN
]    INSERT INTO administrador (apeynom, viveahi, tel, sexo, fechnac)
]    VALUES (@apeynom, @viveahi, @tel, @sexo, @fechnac);
]    END
]

```

Ejecutamos el procedimiento

```

-- Insertar datos utilizando el procedimiento almacenado correspondiente:
EXEC InsertarAdministrador 'Tina Rodriguez', 'S', '40565369', 'F', '09-08-1997';

```

Resultado:

Bueno, el nuevo administrador se inserta correctamente.Ejecución exitosa.

Results

Messages

	idadmin	apeynom	viveahi	tel	sexo	fechnac	
165	165	ARMOA DEMETRIA	N	3651235689	F	1992-08-15 00:00:00.000	
166	166	LUGO DE R. RAMONA	S	3652235689	F	1975-10-18 00:00:00.000	
167	167	LOPEZ ELOYSA	N	3653235689	F	1989-06-08 00:00:00.000	
168	168	LOPEZ CONRADO	S	3654235689	M	1983-10-26 00:00:00.000	
169	169	TALavera CONSTANCIA	N	3655235689	F	1986-01-02 00:00:00.000	
170	170	VARGAS MIGUEL	S	3676235689	M	1986-03-15 00:00:00.000	
171	171	MACIEL PAULINA	N	3677235689	F	1970-03-18 00:00:00.000	
172	172	SAUCEDO FAUSTINA	S	3678235689	F	1987-07-23 00:00:00.000	
173	173	GONZALEZ LIDIA ESTER	N	3671235689	F	1984-11-20 00:00:00.000	
174	174	ARAUJO GUILLERMO JOSE	S	3672235689	M	1985-10-13 00:00:00.000	
175	175	Tina Rodriguez	S	40565369	F	1997-09-08 00:00:00.000	

- Modificar datos de un Administrador existente

Código implementado

GO

```
CREATE PROCEDURE ModificarAdministrador
(
    @idadmin int,
    @apeynom varchar(50),
    @viveahi varchar(1),
    @tel varchar(20),
    @sexo varchar(1),
    @fechnac datetime
)
AS
BEGIN
    UPDATE administrador
    SET apeynom = @apeynom,
        viveahi = @viveahi,
        tel = @tel,
        sexo = @sexo,
        fechnac = @fechnac
    WHERE idadmin = @idadmin;
END
```

Ejecutamos el procedimiento

```
-- Modificar datos de un administrador utilizando el procedimiento almacenado correspondiente:
EXEC ModificarAdministrador 175, 'Maria Tina Rodriguez', 'N', '40565369', 'F', '09-08-1997';
```

Resultado:

El procedimiento se ejecuta de manera exitosa, los datos son modificados correctamente.

Results

Messages

	idadmin	apeynom	viveahi	tel	sexo	fechnac	
165	165	ARMOA DEMETRIA	N	3651235689	F	1992-08-15 00:00:00.000	
166	166	LUGO DE R. RAMONA	S	3652235689	F	1975-10-18 00:00:00.000	
167	167	LOPEZ ELOYSA	N	3653235689	F	1989-06-08 00:00:00.000	
168	168	LOPEZ CONRADO	S	3654235689	M	1983-10-26 00:00:00.000	
169	169	TALAVERA CONSTANCIA	N	3655235689	F	1986-01-02 00:00:00.000	
170	170	VARGAS MIGUEL	S	3676235689	M	1986-03-15 00:00:00.000	
171	171	MACIEL PAULINA	N	3677235689	F	1970-03-18 00:00:00.000	
172	172	SAUCEDO FAUSTINA	S	3678235689	F	1987-07-23 00:00:00.000	
173	173	GONZALEZ LIDIA ESTER	N	3671235689	F	1984-11-20 00:00:00.000	
174	174	ARAUJO GUILLERMO JOSE	S	3672235689	M	1985-10-13 00:00:00.000	
175	175	Maria Tina Rodriguez	N	40565369	F	1997-09-08 00:00:00.000	

- Eliminar un Administrador

Para este caso no se pudo implementar con éxito el procedimiento borrarAdministrador, ya que un administrador tiene muchas dependencias dentro de la base de datos de consorcio. Una solución la eliminación en cascada usando el comando “ ON DELETE CASCADE”, pero no es una práctica muy recomendable, por lo que se decidió no implementar dicha solución solo para este caso.

Procedimientos para Gasto

- Insertar un nuevo registro de gasto

Código implementado

GO

```
CREATE PROCEDURE InsertarGasto(
    @idprovincia INT,
    @idlocalidad INT,
    @idconsorcio INT,
    @periodo INT,
    @fechapago DATETIME,
    @idtipogasto INT,
    @importe DECIMAL(8, 2)
)
AS
BEGIN
    INSERT INTO gasto (idprovincia, idlocalidad, idconsorcio, periodo, fechapago, idtipogasto, importe)
    VALUES (@idprovincia, @idlocalidad, @idconsorcio, @periodo, @fechapago, @idtipogasto, @importe);
END;
```

Ejecutamos el procedimiento

```
--Insertar registro en la tabla Gasto
EXEC InsertarGasto @idprovincia = 1, @idlocalidad = 3, @idconsorcio = 1,
@periodo = 12, @fechapago = '20231205', @idtipogasto = 3, @importe = 45256.10;
```

Resultado:

Bueno, observamos que se registra un nuevo gasto en la tabla de gastos, este registro se posiciona al final de la lista.

	idgasto	idprovincia	idlocalidad	idconsorcio	periodo	fechapago	idtipogasto	importe
7...	7995	24	17	6	1	2017-01-20 00:00:00.000	4	6293.11
7...	7996	24	17	6	4	2017-04-08 00:00:00.000	5	195.62
7...	7997	24	17	6	9	2017-09-02 00:00:00.000	5	871.30
7...	7998	24	17	6	8	2017-08-18 00:00:00.000	5	107.52
7...	7999	24	17	6	1	2017-01-01 00:00:00.000	2	1424.34
8...	8000	24	17	6	7	2017-07-02 00:00:00.000	5	869.17
8...	8003	1	3	1	12	2023-12-05 00:00:00.000	3	45256.10

- Modificar los datos de registro de un gasto

Código implementado

```
GO

CREATE PROCEDURE ModificarGasto
(
    @idgasto INT,
    @idprovincia INT,
    @idlocalidad INT,
    @idconsorcio INT,
    @periodo INT,
    @fechapago DATETIME,
    @idtipogasto INT,
    @importe DECIMAL(8, 2)
)
AS
BEGIN
    IF EXISTS (
        SELECT 1 FROM consorcio WHERE idprovincia = @idprovincia AND idlocalidad = @idlocalidad AND idconsorcio = @idconsorcio
    )
    BEGIN
        UPDATE gasto
        SET
            periodo = @periodo,
            fechapago = @fechapago,
            idtipogasto = @idtipogasto,
            importe = @importe
        WHERE
            idgasto = @idgasto
    END
END;

```

Ejecutamos el código

```
--Modificar un registro existente de la tabla Gasto
EXEC ModificarGasto @idgasto = 8003, @idprovincia = 1, @idlocalidad = 3, @idconsorcio = 1,
@periodo = 1, @fechapago = '20240105', @idtipogasto = 5, @importe = 20256.10

```

Resultado:

Siguiendo con el ejemplo anterior, modificamos los datos de periodo, fecha de pago, tipo de pago e importe, el procedimiento se ejecuta de manera exitosa,

	idgasto	idprovincia	idlocalidad	idconsorcio	periodo	fechapago	idtipogasto	importe
7...	7995	24	17	6	1	2017-01-20 00:00:00.000	4	6293.11
7...	7996	24	17	6	4	2017-04-08 00:00:00.000	5	195.62
7...	7997	24	17	6	9	2017-09-02 00:00:00.000	5	871.30
7...	7998	24	17	6	8	2017-08-18 00:00:00.000	5	107.52
7...	7999	24	17	6	1	2017-01-01 00:00:00.000	2	1424.34
8...	8000	24	17	6	7	2017-07-02 00:00:00.000	5	869.17
8...	8003	1	3	1	1	2024-01-05 00:00:00.000	5	20256.10

- Eliminar el registro de un gasto

Código implementado

GO

```
]CREATE PROCEDURE BorrarGasto
(
    @idgasto INT
)
AS
]BEGIN
]    DELETE FROM gasto
]    WHERE idgasto = @idgasto;
]END;
```

Ejecutamos el procedimiento

```
--Eliminar un registro de ta la tabla Gasto
EXEC BorrarGasto @idgasto = 8003;
```

Resultado:

El procedimiento se ejecuta correctamente eliminando el registro del gasto creado anteriormente, ocupando la última posición ahora el registro de gasto anterior al eliminado.

	idgasto	idprovincia	idlocalidad	idconsorcio	periodo	fechapago	idtipogasto	importe
7...	7994	24	17	6	1	2017-01-08 00:00:00.000	5	384.22
7...	7995	24	17	6	1	2017-01-20 00:00:00.000	4	6293.11
7...	7996	24	17	6	4	2017-04-08 00:00:00.000	5	195.62
7...	7997	24	17	6	9	2017-09-02 00:00:00.000	5	871.30
7...	7998	24	17	6	8	2017-08-18 00:00:00.000	5	107.52
7...	7999	24	17	6	1	2017-01-01 00:00:00.000	2	1424.34
8...	8000	24	17	6	7	2017-07-02 00:00:00.000	5	869.17

6 - Conclusiones

Esta investigación nos ha dejado en claro la importancia de los backups. Al final del día, vivimos en una época donde la información digital importante puede estar en alto riesgo, y el mínimo fallo en algún sistema puede implicar el freno temporal de algún sistema o pérdidas masivas de datos realmente importantes y necesarios. Por eso es importante conocer la función de los backup y los restore, y mucho más es, realizarlos regularmente en la medida de lo posible para mantener las bases de datos resguardadas en caso de alguna emergencia, descuido o falla.

Los datos importantes deben siempre tener un respaldo, incluso los propios respaldos deberían tenerlos, todo para asegurar la protección de los mismos.

Herramientas Utilizadas

- Discord - Usado para todas las reuniones del grupo.
- SQL Server Management
- GitHub - Donde se resguarda toda la información del proyecto.

7 - Bibliografía

- <https://learn.microsoft.com/es-es/sql/relational-databases/backup-restore/online-restore-sql-server?view=sql-server-ver16#log-backups-for-online-restore>
- <https://learn.microsoft.com/es-mx/sql/relational-databases/backup-restore/back-up-and-restore-of-sql-server-databases?view=sql-server-ver16#more-information-and-resources>