Universidad Nacional del Nordeste

Facultad de Ciencias Exactas y Naturales y Agrimensura

Carrera: Licenciatura en Sistemas de Información

Cátedra: Base de Datos I

Profesor: Vallejos, Walter Oscar

Tema: Procedimientos y funciones almacenada

Año: 2023

Grupo 8

Integrantes:

- Alcaraz, Mauricio
- Cruz, Julian Luis
- Peloso, Nicolás Anibal

ÍNDICE:

CAPÍTULO I: INTRODUCCIÓN 4
Tema 4
Planteamiento del problema 4
Objetivo del Trabajo Práctico 4
Objetivos Generales 4
Objetivos Específicos 5
CAPÍTULO II: MARCO CONCEPTUAL O REFERENCIAL 6
Procedimientos Almacenados 6
Ventajas de usar procedimientos almacenados 6
Tipos de procedimientos almacenados 7
Funciones 8
Ventajas de las funciones definidas por el usuario 8
Tipos de funciones 9
Instrucciones válidas en una función 9
Diferencias entre Funciones y Procedimientos almacenados 10
Backup y Restore 12
Estrategias de copias de seguridad y restauración 12
Elección del modelo de recuperación adecuado 13
Modos de recuperación de las bases de datos 13
Diseñar la estrategia de copia de seguridad 14
Probar las copias de seguridad 15
Réplicas 15
¿De qué nos sirve replicar una base de datos? 15
¿Qué hay que tener en cuenta para poder implementar las técnicas de replicación de una base de datos? 15
¿Es posible trabajar con distintos motores de bases de datos? 16
¿Cómo es posible implementar esto a partir de una base de datos? 16
Vistas 17
Utilidades de las vistas 17
Vistas Indexadas 18
Crear índices en una vista 18
Pasos para la generación de Vistas Indexadas 18
Cláusulas para indexación de vistas 19
Requisitos adicionales 20
Índices columnares en SQL Server 20
Optimización de consultas a través de índices: 21
¿Cómo funcionan los índices? 21
Tipos de Índices 21
Ventajas y Desventajas: 22

Manejo de permisos a nivel de usuario de base de datos **24**
Roles fijos de base de datos en sql server 24
Roles especiales para SQL Database 25
Roles de Base de Datos Definidos por el Usuario(User-Defined Database Roles) 26
Gestionando Roles de Base de Datos Definidos por el Usuario Usando T-SQL 26
Autorización 26
Transacciones y Transacciones anidadas **27**
Condiciones de terminación 27
Características de una transacción 27
Importancia de las transacciones en bases de datos 28
Casos de uso de las transacciones 28
Primitivas para el manejo de transacciones 28
Tipos de Transacciones Clasificación de acuerdo a su estructura 29
Restricciones para una transacción anidada 30
Estados de una transacción 30
Triggers **31**
CAPÍTULO III: METODOLOGÍA SEGUIDA **34**
Herramientas utilizadas 34
CAPÍTULO IV: DESARROLLO DEL TEMA / PRESENTACIÓN DE RESULTADOS **35**
Diccionario de Datos: 35
Restricciones 38
CAPÍTULO V: CONCLUSIONES **50**
VI. BIBLIOGRAFÍA. **51**

CAPÍTULO I: INTRODUCCIÓN

Tema

El tema de este proyecto de estudio son los procedimientos y funciones en bases de datos relacionales, en donde se buscará comprender y aplicar de forma práctica estas herramientas dado un problema y modelo de datos específicos. El problema a investigarse trata sobre la base de datos de un consorcio en donde intervienen varias entidades y relaciones.

Planteamiento del problema

Al iniciar con este tema se nos dio como tarea realizar 3 procedimientos almacenados para 3 tablas distintas de entidades relacionadas con el dominio del problema y la creación de una función, que permita calcular la edad en base a la fecha de nacimiento de los administradores.
A raíz de eso nos surgió la pregunta: ¿Que es un procedimiento almacenado y que es una función y cómo funcionan en sql ?
para responder a esa pregunta tuvimos que buscar cual era el alcance de cada una y en qué se diferenciaban, para qué servían y cómo podríamos aplicarlas en nuestro trabajo, esperamos que estas funcionaran de forma similar a como son en otros lenguajes de programacion, asi como tambien ver las limitantes que tiene cada una y que es lo que pueden hacer las funciones que los procedimientos no y viceversa, saber si era posible utilizar una función dentro de un procedimiento almacenado o cómo influyen estos en el rendimiento de un sistema y que tan seguros pueden llegar a ser así como las posibilidades que tienen para poder reutilizarse.

Objetivo del Trabajo Práctico

El objetivo de este proyecto de estudio es poder comprender y aplicar de forma práctica los conceptos teóricos adquiridos que esperamos sean de gran utilidad tanto para nosotros como para los lectores del documento, para ello nos enfocamos en lo siguiente:

Objetivos Generales

- Brindar una visión general de los procedimientos y funciones en bases de datos SQL.
- Explorar cómo los procedimientos y funciones pueden mejorar la eficiencia y reutilización del código.
- Demostrar cómo diseñar, crear y ejecutar procedimientos y funciones en SQL.
- Comprender el uso de parámetros y variables en procedimientos y funciones.

- Identificar escenarios comunes en los que los procedimientos y funciones son especialmente útiles.

Objetivos Específicos

- Explicar la diferencia entre procedimientos y funciones en SQL.
- Detallar la sintaxis para crear procedimientos almacenados en diferentes sistemas de gestión de bases de datos.
- Mostrar ejemplos de procedimientos que realicen tareas comunes, como la actualización de registros o la generación de informes.
- Ilustrar cómo las funciones pueden utilizarse para realizar cálculos y devolver resultados.
- Abordar la seguridad y buenas prácticas en el uso de procedimientos y funciones en bases de datos.

CAPÍTULO II: MARCO CONCEPTUAL O REFERENCIAL

Procedimientos Almacenados:

Un procedimiento almacenado de SQL Server es un conjunto de instrucciones Transact-SQL a las que se les da un nombre, que se almacena en el servidor. 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. Entre otras, 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
Los comandos de un procedimiento se ejecutan en un único lote de código. Esto puede reducir significativamente el tráfico de red entre el servidor y el cliente porque únicamente se envía a través de la red la llamada que va a ejecutar el procedimiento. Sin la encapsulación de código que proporciona un procedimiento, cada una de las líneas de código tendría que enviarse a través de la red.

Mayor seguridad
Varios usuarios y programas cliente pueden realizar operaciones en los objetos de base de datos subyacentes a través de un procedimiento, aunque los usuarios y los programas no tengan permisos directos sobre esos objetos subyacentes. El procedimiento controla qué procesos y actividades se llevan a cabo y protege los objetos de base de datos subyacentes. Esto elimina la necesidad de conceder permisos en cada nivel de objetos y simplifica los niveles de seguridad.
Al llamar a un procedimiento a través de la red, solo está visible la llamada que va a ejecutar el procedimiento. Por tanto, los usuarios malintencionados no pueden ver los nombres de los objetos de la base de datos y las tablas, insertar sus propias instrucciones Transact-SQL ni buscar datos críticos.

El uso de parámetros de procedimientos ayuda a protegerse contra ataques por inyección de código SQL. Dado que la entrada de parámetros se trata como un valor literal y no como código ejecutable, resulta más difícil para un atacante insertar un comando en las instrucciones Transact-SQL del procedimiento y poner en peligro la seguridad, los procedimientos pueden cifrarse, lo que ayuda a ofuscar el código fuente.

Reutilización del código

El código de cualquier operación de base de datos redundante resulta un candidato perfecto para la encapsulación de procedimientos. De este modo, se elimina la necesidad de escribir de nuevo el mismo código, se reducen las inconsistencias de código y se permite que cualquier usuario o aplicación que cuente con los permisos necesarios pueda acceder al código y ejecutarlo.

Mantenimiento más sencillo

Cuando las aplicaciones cliente llaman a procedimientos y mantienen las operaciones de base de datos en la capa de datos, solo deben actualizarse los cambios de los procesos en la base de datos subyacente. El nivel de aplicación permanece independiente y no tiene que tener conocimiento sobre los cambios realizados en los diseños, las relaciones o los procesos de la base de datos.

Rendimiento mejorado

De forma predeterminada, un procedimiento se compila la primera vez que se ejecuta y crea un plan de ejecución que vuelve a usarse en posteriores ejecuciones. Como el procesador de consultas no tiene que crear un nuevo plan, normalmente necesita menos tiempo para procesar el procedimiento.

Tipos de procedimientos almacenados

Definidos por el usuario

Un procedimiento definido por el usuario se puede crear en una base de datos definida por el usuario o en todas las bases de datos del sistema excepto en la base de datos Resource .

Temporales

Los procedimientos temporales son una forma de procedimientos definidos por el usuario. Los procedimientos temporales son iguales que los procedimientos permanentes salvo porque se almacenan en tempdb. Hay dos tipos de procedimientos temporales: locales y globales. Se diferencian entre sí por los nombres, la visibilidad y la disponibilidad. Los procedimientos temporales locales tienen como primer carácter de sus nombres un solo signo de número (#); solo son visibles en la conexión actual del usuario y se eliminan cuando se cierra la conexión. Los procedimientos temporales globales presentan dos signos de número (##) antes

del nombre; son visibles para cualquier usuario después de su creación y se eliminan al final de la última sesión en la que se usa el procedimiento.

Sistema
Los procedimientos del sistema se incluyen con SQL Server. Están almacenados físicamente en la base de datos interna y oculta Resource y se muestran de forma lógica en el esquema sys de cada base de datos definida por el sistema y por el usuario. Dado que los procedimientos del sistema empiezan con el prefijo sp_ , es recomendable que no se utilice este prefijo cuando se asigne un nombre a los procedimientos definidos por el usuario.

Extendidos definidos por el usuario
Los procedimientos extendidos permiten crear rutinas externas en un lenguaje de programación, como C. Estos procedimientos son bibliotecas DLL que una instancia de SQL Server puede cargar y ejecutar de forma dinámica.
Estos se quitarán en una versión futura de SQL Server. No es recomendable utilizar esta característica en nuevos trabajos de desarrollo y modificar lo antes posible las aplicaciones que actualmente la utilizan.

Funciones:

Una función es un conjunto de sentencias que operan como una unidad lógica, al igual que las funciones de los lenguajes de programación, las funciones definidas por el usuario de SQL Server son rutinas que aceptan parámetros, realizan una acción, como un cálculo complejo, y devuelven el resultado de esa acción como un valor. El valor devuelto puede ser un valor escalar único o un conjunto de resultados.

Ventajas de las funciones definidas por el usuario

Programación modular.
Puede crear la función una vez, almacenarla en la base de datos y llamarla desde el programa tantas veces como desee. Las funciones definidas por el usuario se pueden modificar, independientemente del código de origen del programa.

Ejecución más rápida.
Al igual que los procedimientos almacenados, las funciones definidas por el usuario de Transact-SQL reducen el costo de compilación del código de Transact-SQL almacenando los planes en la caché y reutilizándolos para ejecuciones repetidas. Esto significa que no es necesario volver a analizar y optimizar la función definida por el usuario con cada uso, lo que permite obtener tiempos de ejecución mucho más rápidos.

Reducción del tráfico de red.
Una operación que filtra datos basándose en restricciones complejas que no se puede expresar en una sola expresión escalar se puede expresar como una función. La función se puede invocar luego en la cláusula WHERE para reducir el número de filas que se envían al cliente.

Tipos de funciones

Funciones escalares
Las funciones escalares definidas por el usuario devuelven un único valor de datos del tipo definido en la cláusula RETURNS. En una función escalar insertada, el valor escalar es el resultado de una sola instrucción. Para una función escalar de varias instrucciones, el cuerpo de la función puede contener una serie de instrucciones de Transact-SQL que devuelven el único valor. El tipo devuelto puede ser de cualquier tipo de datos excepto text, ntext, image, cursor y timestamp.

Funciones con valores de tabla
Las funciones con valores de tabla definidas por el usuario devuelven un tipo de datos table. Las funciones insertadas con valores de tabla no tienen cuerpo; la tabla es el conjunto de resultados de una sola instrucción SELECT.

Funciones del sistema
SQL Server proporciona numerosas funciones del sistema que se pueden usar para realizar diversas operaciones. No se pueden modificar.

Instrucciones válidas en una función

No todas las sentencias SQL son válidas dentro de una función. Entre los tipos de instrucciones válidos en una función se incluyen:

- Las instrucciones DECLARE se pueden usar para definir variables y cursores de datos locales de la función.
- La asignación de valores a objetos locales de la función, como el uso de SET para asignar valores a variables locales escalares y de tabla.
- Las operaciones de cursores que hacen referencia a cursores locales que están declarados, abiertos, cerrados y no asignados en la función. No se permiten las instrucciones FETCH que devuelven datos al cliente. Solo se permiten las instrucciones FETCH que asignan valores a variables locales mediante la cláusula INTO.
- Instrucciones de control de flujo excepto instrucciones TRY...CATCH.
- Instrucciones SELECT que contienen listas de selección con expresiones que asignan valores a las variables locales para la función.

- Instrucciones UPDATE, INSERT y DELETE que modifican las variables de tabla locales de la función.
- Instrucciones EXECUTE que llaman a un procedimiento almacenado extendido.

Los parámetros de entrada pueden ser de cualquier tipo, excepto timestamp, cursor y table. Las funciones definidas por el usuario no permiten parámetros de salida. NO es posible emplear en ellas funciones no deterministas (como getdate()) ni sentencias de modificación o 14 actualización de tablas o vistas. Si podemos emplear sentencias de asignación, de control de flujo (if), de modificación y eliminación de variables locales. SQL Server admite. Las funciones definidas por el usuario se crean con la instrucción "create function" y se eliminan con "drop function". Una función escalar retorna un único valor. Como todas las funciones, se crean con la instrucción "create function". Las funciones que retornan una tabla pueden emplearse en lugar de un "from" de una consulta.

Diferencias entre Funciones y Procedimientos almacenados

Las diferencias más generales entre procedimientos y funciones es que cada una es llamada de modo diferente y para propósitos distintos:

1. Un procedimiento no nos regresa un valor. En su lugar, es llamado con una declaración CALL para realizar una operación como modificar una tabla o procesar la recuperación de registros.
2. Una función es llamada dentro de una expresión y nos regresa un valor único directamente al que lo llama para ser utilizado en la expresión.
3. No se puede invocar una función con una instrucción CALL, ni puedes invocar un procedimiento en una expresión.
4. Normalmente, las funciones se utilizan para cálculos en los que normalmente se utilizan procedimientos para ejecutar la lógica de negocio.
5. Dentro de una función, no está permitido modificar una tabla que ya está siendo utilizada (para leer o escribir) por la sentencia que invocó la función o disparador.

La sintaxis para la creación de rutinas es algo diferente para procedimientos y funciones:

1) Los parámetros de procedimiento se pueden definir como sólo de entrada, sólo de salida o ambos. Esto significa que un procedimiento puede devolver valores al que los llama usando parámetros de salida. Se pueden acceder a estos valores en sentencias que siguen a la instrucción CALL. Las funciones sólo tienen parámetros de entrada. Como resultado, aunque ambos procedimientos y funciones pueden tener parámetros, la declaración de parámetros de procedimiento difiere de la de funciones.

2) Las funciones devuelven valores, por lo que debe haber una cláusula RETURNS en la definición de función para indicar el tipo de datos del valor que se regresara. Además, debe haber al menos una declaración RETURN dentro del cuerpo de la función para devolver un valor al que llama. RETURNS y RETURN no aparecen en las definiciones de los procedimientos.
    a) Para invocar un procedimiento almacenado, utilice la instrucción CALL. Para invocar una función almacenada, refiérase a ella en una expresión. La función devuelve un valor durante la evaluación de la expresión.
    b) Un procedimiento se invoca mediante una instrucción CALL y sólo puede devolver valores utilizando variables de salida. Una función puede ser llamada desde dentro de una sentencia como cualquier otra función (es decir, invocando el nombre de la función), y puede devolver un valor escalar.
    c) Especificar un parámetro como IN, OUT o INOUT sólo es válido para PROCEDIMIENTO. Para una FUNCIÓN, los parámetros siempre se consideran parámetros IN. Si no se da ninguna palabra clave antes de un nombre de parámetro, es un parámetro IN por defecto. Los parámetros para las funciones almacenadas no están precedidos por IN, OUT o INOUT. Todos los parámetros de función se tratan como parámetros IN.

Para definir un procedimiento o una función almacenada, utilice CREATE PROCEDURE o CREATE FUNCTION respectivamente:

```
CREATE PROCEDURE proc_name ([parameters])
 [characteristics]
 routine_body

CREATE FUNCTION func_name ([parameters])
 RETURNS data_type      //diferente
 [characteristics]
 routine_body
```

Una extensión de MySQL para el procedimiento almacenado (no para las funciones) es que un procedimiento puede generar un conjunto de resultados, o incluso varios conjuntos de resultados, que el que llama procesa de la misma manera que el resultado de una sentencia SELECT. Sin embargo, el contenido de tales conjuntos de resultados no se puede utilizar directamente en la expresión.

Los procedimientos y funciones almacenados no comparten el mismo espacio de nombres. Es posible tener un procedimiento y una función con el mismo nombre en una base de datos.
En Procedimientos almacenados se puede utilizar SQL dinámico(instrucciones preparadas de SQL PREPARE, EXECUTE, DEALLOCATE **) pero no en funciones o disparadores.**

Backup y Restore

La copia de seguridad (backup) y la capacidad de restauración (restore) 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, por ejemplo:

- Errores de medios.
- Errores de usuario. Ej. quitar una tabla por accidente.
- Errores de hardware. Ej. una unidad de disco dañada o la pérdida permanente de un servidor.
- Desastres naturales.

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 estas requiere una estrategia.
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 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.

Elección del modelo de recuperación adecuado

Las operaciones de copias de seguridad y restauración se producen en el contexto de un modelo de recuperación. Este 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 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 de recuperación completa. La elección del modelo de recuperación más óptimo depende de los requisitos empresariales.

Modos de recuperación de las bases de datos

Existen tres modos de recuperación en SQL Server, "Simple" (Simple), "Full" (Completa) y "Bulk-Logged" (Registros extensos)

1. Recuperación Simple (Simple Recovery Model)

   En este modo, SQL Server mantiene registros de transacciones mínimos. Una vez que una transacción se completa con éxito, los registros se liberan y no se mantienen para la recuperación.
   Las copias de seguridad de registros de transacciones no son posibles en este modo. Solo se pueden realizar copias de seguridad completas de la base de datos.
   Este modo es adecuado para bases de datos que no requieren la capacidad de recuperación a un punto en el tiempo y donde la pérdida de datos desde la última copia de seguridad completa es aceptable.

2. Recuperación Completa (Full Recovery Model)

En este modo, SQL Server mantiene un registro completo de todas las transacciones realizadas en la base de datos.
Permite la recuperación a un punto en el tiempo, lo que significa que puedes restaurar la base de datos a un estado específico en el pasado, utilizando copias de seguridad de registros de transacciones.
Las copias de seguridad de registros de transacciones son esenciales y deben programarse y gestionarse adecuadamente para evitar que los archivos de registro crezcan descontroladamente.
Adecuado para bases de datos críticas con requisitos de recuperación avanzados.

3. Recuperación de Registros Extensos (Bulk-Logged Recovery Model)

   Este modo es una especie de híbrido entre "Simple" y "Full." Al igual que "Full," se registran todas las transacciones, pero hay excepciones.
   Algunas operaciones de carga masiva (como la carga de datos con índices agrupados) pueden minimizar el registro de transacciones, lo que puede mejorar el rendimiento en comparación con "Full."
   No permite la recuperación a un punto en el tiempo después de ciertas operaciones masivas. Por lo tanto, después de realizar operaciones a granel, debes realizar una copia de seguridad completa de la base de datos para restaurar a un punto específico en el tiempo.

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, se 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?
- ¿Cuál es la probabilidad de que se produzcan cambios y actualizaciones?
- ¿Es probable que los cambios tengan lugar solo en una pequeña parte de la base de datos o en una grande?
- ¿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?

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 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.

Réplicas

Decimos que una réplica de una base de datos **A** es otra base de datos **B** que contiene una copia total o parcial de **A**, y es actualizada con una periodicidad conocida.

¿De qué nos sirve replicar una base de datos?

Replicar bases de datos nos puede servir para, por ejemplo:

- Crear Nodos de Contingencia, esto significa que, si se cuenta con una base de datos **B** en un servidor distinto al de la base de datos **A**, y esta misma sufre un inconveniente; se podrá seguir operando con los mismos datos en modo lectura o consulta con la base datos **B**, ya que, al ser una copia de **A**, servirá de contingencia (o de respaldo) sin necesidad de interferir la BD A.
- Por cuestiones de proximidad geográfica. Si se necesita consultar una base de datos de una empresa que está distribuida geográficamente, se puede tener nodos de consultas mucho más cercanos de donde se encuentra la sucursal central, sin necesidad de hacer consultas lejanas navegando por distintos tipos de redes.
- Para procesamientos centralizados. Existen empresas que están distribuidas por varias regiones, pero tienen un lugar de almacenamiento común entre todas las sucursales, donde se centralizan todos los datos que se producen en las mismas, obteniendo así una base de datos general con información referida a toda la empresa.

- Escalabilidad y optimización de carga de trabajo. Al tener una réplica de una base de datos tenemos la posibilidad de realizar en ella trabajos pesados como reportes, etc., que no afecten al rendimiento de la base principal.

¿Qué hay que tener en cuenta para poder implementar las técnicas de replicación de una base de datos?

En la replicación existen 3 roles; el “Publicador”, el “Distribuidor”, y el “Suscriptor”.

El “Publicador” será el servidor Instancia de SQL server que estará encargado de Publicar la Base de Datos que es la que se va a Replicar a los demás servidores.

El “Distribuidor” será el encargado de distribuir a los suscriptores la Base de Datos que ha publicado el Publicador. Normalmente el Publicador y el Distribuidor se configuran en el mismo Servidor.

El “Suscriptor” será la Instancia que reciba la Réplica del Publicador.

O sea, para llevar a cabo una réplica de base de datos es necesario tener configuradas 2 instancias distintas en el motor de base de datos, una publicadora y otra suscriptora. Y además tener en cuenta que existen dos procesos distintos.

Además de tener configuradas 2 instancias distintas, también debemos tener en cuenta que se trabaja con diferentes versiones de SQL server, porque para que una instancia pueda ser publicadora se necesita que sea desde la versión SQL developer y adelante, una versión SQL express sólo puede ser suscriptora de una réplica.

¿Es posible trabajar con distintos motores de bases de datos?

Sí, se puede trabajar con distintos motores de bases de datos para implementar una replicación de base de datos siempre y cuando estén correctamente configurados.

En este caso particular usaremos sólo el motor de bases de datos SQL SERVER.

¿Cómo es posible implementar esto a partir de una base de datos?

Se trabaja principalmente sobre las configuraciones de las instancias y del motor de base de datos.

Vistas

Una vista es una tabla virtual cuyo contenido está definido por una consulta. Al igual que una tabla, una vista consta de un conjunto de columnas y filas de datos con un nombre. Sin embargo, a menos que esté indexada, una vista no existe como conjunto de valores de datos almacenados en una base de datos. Las filas y las columnas de datos proceden de tablas a las que se hace referencia en la consulta que define la vista y se producen de forma dinámica cuando se hace referencia a la vista.

Las VISTAS, se utilizan a menudo como filtros de las tablas de las que recogen datos y es por ello por lo que se utilizan como mecanismos de seguridad, ya que permiten al usuario tener acceso de visibilidad de los datos, pero no a las tablas reales subyacentes. También se utilizan para simplificar y personalizar la visualización de los datos para el usuario.

En SQL Server, existen diferentes tipos de vistas:

- **Vistas de usuario:** tablas virtuales definidas por el usuario cuyo contenido está definido por una consulta.
- **Vistas indexadas:** cuyos datos se han almacenado como una tabla real ya que se han indizado a través de un índice clúster único. Este tipo de vistas mejoran mucho el rendimiento en algunos tipos de consultas que devuelven muchos registros.
- **Vistas con particiones:** combinan datos horizontales con particiones de un conjunto de tablas miembro en uno o más servidores.
- **Vistas del sistema:** específicas para consultar metadatos del sistema.

Utilidades de las vistas

Puede crear vistas en el motor de base de datos de SQL Server mediante SQL Server Management Studio o Transact-SQL. Se puede usar una vista para lo siguiente:

- Permiten centrar, y alterar la percepción de la base de datos para cada usuario
- Un mecanismo de seguridad ya que genera una suerte de virtualización/acceso directo de la tabla y solo permite ver cierta cantidad de datos de esa tabla, pero no se les permite ver las tablas subyacentes a esa vista

- Proporciona una interfaz compatible con versiones anteriores para emular una tabla cuyo esquema podría haber cambiado

Vistas Indexadas

Una vista indexada es una vista que se ha materializado. Esto significa que se ha calculado la definición de la vista y que los datos resultantes se han almacenado como una tabla. Se puede indicar una vista creando un índice clúster único en ella. Las vistas indexadas pueden mejorar de forma considerable el rendimiento de algunos tipos de consultas. Las vistas indexadas funcionan mejor para consultas que agregan muchas filas. No son adecuadas para conjuntos de datos subyacentes que se actualizan frecuentemente.

Crear índices en una vista

El primer índice creado en una vista debe ser un **índice clúster único**. Después de haber creado el índice clúster único, puede crear más índices no clúster. La creación de un índice clúster único en una vista mejora el rendimiento de las consultas, ya que la vista se almacena en la base de datos de la misma manera que se almacena una tabla con un índice agrupado. El optimizador de consultas puede utilizar vistas indexadas para acelerar la ejecución de las consultas. No es necesario hacer referencia a la vista en la consulta para que el optimizador tenga en cuenta esa vista para una sustitución.

Pasos para la generación de Vistas Indexadas

Para crear una vista indexada, es necesario seguir los pasos descritos a continuación, que son fundamentales para la correcta implementación de la vista indexada:

1. Compruebe que las opciones SET son correctas para todas las tablas existentes a las que se hará referencia en la vista.
2. Compruebe que las opciones SET de la sesión están establecidas correctamente antes de crear cualquier tabla y la vista.
3. Compruebe que la definición de vista sea determinista.
4. Verifique que la tabla base y la vista tengan el mismo propietario.
5. Cree la vista con la opción WITH SCHEMABINDING.
6. Cree el índice clúster único en la vista.

Cláusulas para indexación de vistas

También podemos agregar cláusulas a un índice de una vista, la elección de los valores dependerá de los requisitos específicos y de la naturaleza de los datos y consultas que se ejecuten en la vista indexada, con el fin de considerar el rendimiento, el espacio en disco y otros factores al configurar estas opciones.

1. **PAD_INDEX = OFF:** Esta cláusula controla si se rellena la página de índice con valores de relleno para mantener un espacio constante en cada página. Cuando está configurado en "OFF", no se rellenan páginas de índice. Esto puede ahorrar espacio en disco, pero puede causar cierta fragmentación en el índice.
2. **STATISTICS_NORECOMPUTE = OFF:** Esta cláusula controla si se deben actualizar las estadísticas del índice automáticamente. Cuando está configurado en "OFF", SQL Server actualizará automáticamente las estadísticas del índice para mejorar el rendimiento de las consultas.
3. **SORT_IN_TEMPDB = OFF:** Esta cláusula controla si el proceso de creación del índice se realiza en la base de datos TEMPDB o en la base de datos actual. Cuando está configurado en "OFF", el proceso se realiza en la base de datos actual.
4. **IGNORE_DUP_KEY = OFF:** Esta cláusula controla si se deben permitir valores duplicados en el índice. Cuando está configurado en "OFF", no se permiten valores duplicados y se generará un error si se intenta insertar un valor duplicado.
5. **DROP_EXISTING = OFF:** Esta cláusula controla si se debe eliminar un índice existente con el mismo nombre antes de crear la vista indexada. Cuando está configurado en "OFF", no se eliminará un índice existente y se generará un error si ya existe un índice con el mismo nombre.

Requisitos adicionales

También se deben cumplir los siguientes requisitos, además de las opciones SET y los requisitos de función deterministas.

- El usuario que ejecuta CREATE INDEX debe ser el propietario de la vista.
- La vista se debe crear mediante la opción WITH SCHEMABINDING.
- La vista solo debe hacer referencia a tablas base que estén en la misma base de datos que la vista. La vista no puede hacer referencia a otras vistas.
- Si GROUP BY está presente, la definición de VIEW debe contener COUNT_BIG(*), pero no HAVING. Estas restricciones GROUP BY solo se pueden aplicar a la definición de vista indizada. Una consulta puede usar una

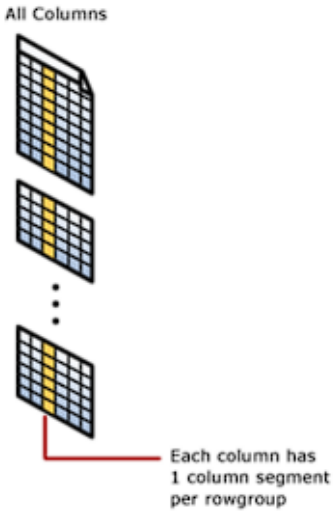
vista indexada en su plan de ejecución, aunque no cumpla estas restricciones GROUP BY.
- Al crear el índice, la opción de índice IGNORE_DUP_KEY debe establecerse en OFF (la configuración predeterminada).
- En la definición de vista, se debe hacer referencia a las tablas mediante nombres de dos partes, esquema.nombredetabla.
- Las funciones definidas por el usuario a las que se hace referencia en la vista se deben crear con la opción WITH SCHEMABINDING.
- Para hacer referencia a las funciones definidas por el usuario a las que se hace referencia en la vista, se deben usar nombres de dos partes, ..

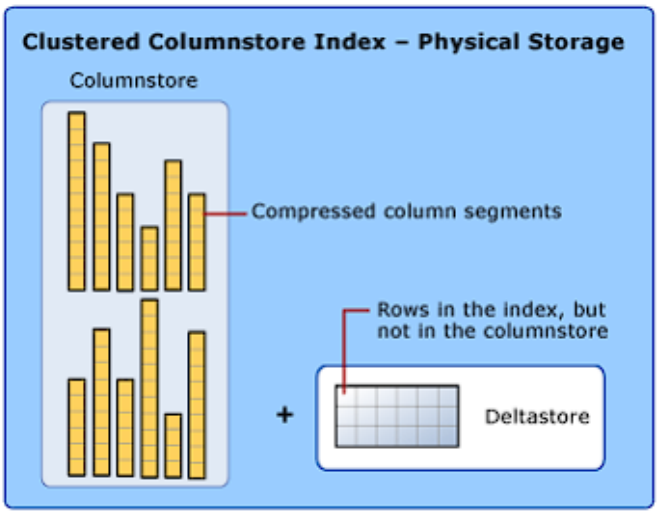
Índices columnares en SQL Server

Los índices de almacén de columnas son el estándar para almacenar y consultar las tablas de hechos de almacenamiento de datos de gran tamaño.
Este índice usa el almacenamiento de datos basado en columnas y el procesamiento de consultas para lograr ganancias de hasta 10 veces el rendimiento de las consultas en el almacenamiento de datos sobre el almacenamiento tradicional orientado a filas.
También puede lograr ganancias de hasta 10 veces la compresión de datos sobre el tamaño de los datos sin comprimir.

Teniendo en cuenta que los índices columnares son un tipo de índice que se utilizan para mejorar el rendimiento de consultas en bases de datos con grandes cantidades de datos. Podemos brindar una visión general acerca de los mismos.

● Almacenamiento Eficiente: Almacenan cada columna de forma independiente, permitiendo un almacenamiento más eficiente (especialmente en tablas con muchas columnas).
● Compresión de Datos: Generalmente aplican técnicas de compresión de datos, reduciendo espacio de almacenamiento, e indirectamente, acelera la lectura de los mismos.
● Mejora del Rendimiento: Los índices columnares están diseñados para consultas analíticas y de agregación, como consultas que involucran sumas, promedios o filtrado de datos en una o varias columnas. Estos índices suelen acelerar significativamente estas consultas.
● No Adecuados para Operaciones de Actualización Frecuente: Los índices columnares pueden no ser la mejor opción para tablas con muchas operaciones de inserción, actualización o eliminación, ya que pueden ralentizar estas operaciones.





Optimización de consultas a través de índices:

¿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 nuestras necesidades. 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.

Otros índices comunes:

- 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 utilizados 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 realizas 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 tienes columnas que se utilizan comúnmente para filtrar datos, es importante indexarlas para acelerar la recuperación de registros.

8. Columnas con Alto Cardenalidad:
   - 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 recordar 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 tu 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.

Manejo de permisos a nivel de usuario de base de datos

Los roles de usuario en las bases de datos son entidades de seguridad que agrupan a otras entidades de seguridad, similares a los grupos del sistema operativo Microsoft Windows. Un rol es un objeto de base de datos que agrupa uno o más privilegios y puede ser asignado a un usuario. Los roles de nivel de base de datos se aplican a toda la base de datos en lo que respecta a su ámbito de permisos.

Tipos de roles: Existen dos tipos de roles en el nivel de base de datos: los roles fijos de base de datos que están predefinidos en la base de datos y los roles de base de datos definidos por el usuario que el usuario puede crear.

Además de los inicios de sesión y usuarios se proporciona un conjunto de roles predefinidos tanto a nivel de servidor como a nivel de base de datos. Estos roles predefinidos contienen conjuntos de permisos comunes que se pueden asignar a los usuarios según sus necesidades. en sql son los siguientes

Roles fijos de base de datos en sql server

- **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.

Roles especiales para SQL Database

Los roles de base de datos solo existen en la base de datos "master" que es una base de datos especial en SQL Server. Los permisos de estos roles están limitados a acciones que se realizan en la base de datos "master". Solo los usuarios de esta base de datos "master" pueden ser miembros de estos roles. Los inicios de sesión (usuarios con acceso al servidor) no pueden ser miembros de estos roles directamente pero se pueden crear usuarios basados en inicios de sesión y luego agregar esos usuarios a estos roles.
Estos roles y permisos se aplican principalmente a la base de datos "master" y suelen estar relacionados con la administración y configuración del servidor.

1. **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.
2. **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.
3. **loginmanager:** Puede crear y eliminar inicios de sesión en la base de datos "master" virtual.

Roles de Base de Datos Definidos por el Usuario(User-Defined Database Roles)

Por lo general, los roles de base de datos definidos por el usuario se aplican cuando un grupo de usuarios de la base de datos necesita llevar a cabo un conjunto común de actividades dentro de una base de datos y no existe un grupo de Windows aplicable. Estos roles se crean y eliminan utilizando tanto el Management Studio como las declaraciones Transact-SQL **CREATE ROLE**, **ALTER ROLE** y **DROP ROLE**.

Gestionando Roles de Base de Datos Definidos por el Usuario Usando T-SQL

La declaración CREATE ROLE crea un nuevo rol de base de datos definido por el usuario en la base de datos actual.
La declaración ALTER ROLE cambia el nombre de un rol de base de datos definido por el usuario.
La declaración DROP ROLE elimina un rol de la base de datos. Los roles que son propietarios de objetos de base de datos (objetos securables) no pueden eliminarse de la base de datos. Para eliminar dicho rol, primero debes transferir la propiedad de esos objetos

Autorización

Solo los usuarios autorizados pueden efectuar declaraciones o realizar operaciones en una entidad. Si un usuario no autorizado intenta realizar cualquiera de estas tareas, la ejecución de la declaración Transact-SQL o la operación en el objeto de la base de datos será rechazada.
Existen tres declaraciones Transact-SQL relacionadas con la autorización:

- **GRANT:** La declaración GRANT otorga permisos a objetos securizables
- **DENY:** La declaración DENY impide que los usuarios realicen acciones. Esto significa que la declaración elimina permisos existentes de las cuentas de usuario o evita que los usuarios obtengan permisos a través de su pertenencia a un grupo/rol que podría concederse en el futuro
- **REVOKE:** La declaración REVOKE elimina uno o varios permisos previamente otorgados o denegados.

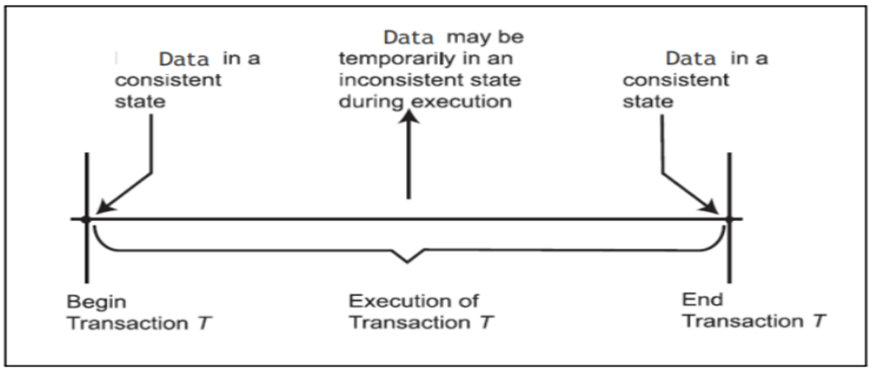
TRANSACCIONES Y TRANSACCIONES ANIDADAS

¿Una pregunta que debemos hacernos es cómo se gestionan los datos en las bases de datos? Existen diferentes puntos que desembocan en una buena gestión de las mismas y las transacciones en base de datos son una parte fundamental de este proceso.

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.

Condiciones de terminación

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

Características de una transacción

Una transacción en base de datos debe cumplir con las siguientes características, conocidas como las propiedades ACID:

● **Atomicidad:** Una transacción se considera atómica, lo que significa que se ejecuta como una unidad completa o no se ejecuta en absoluto. Si alguna de las operaciones falla, se deshacen todas las operaciones anteriores, asegurando que los datos no queden en un estado inconsistente.

● **Consistencia:** Una transacción debe llevar la base de datos desde un estado válido a otro estado válido. Esto garantiza que se respeten las restricciones y reglas definidas para los datos.

● **Aislamiento:** Cada transacción se ejecuta de manera aislada y no se ve afectada por otras transacciones concurrentes. Esto evita problemas de concurrencia y garantiza la integridad de los datos.

● **Durabilidad:** Una vez que una transacción se ha completado correctamente, los cambios realizados en la base de datos se mantienen permanentemente, incluso en caso de fallos del sistema. Los datos actualizados son duraderos y no se perderán.

Un ejemplo que podemos analizar es, imaginar que estás realizando una compra en línea. Cuando seleccionas los productos y procedes al pago, se lleva a cabo una transacción en la base de datos. En este caso, la transacción incluye operaciones como reducir el inventario de los productos comprados, registrar la transacción en la cuenta del cliente y generar un recibo de compra. Si algo falla durante este proceso, todas las operaciones se deshacen para evitar inconsistencias.

Importancia de las transacciones en bases de datos

Las transacciones son vitales en entornos donde múltiples usuarios pueden acceder y modificar la misma información al mismo tiempo. Garantizan la integridad y consistencia de los datos, evitando conflictos y asegurando que los cambios se realicen de manera segura.

Casos de uso de las transacciones

Las transacciones son utilizadas en una amplia gama de aplicaciones y sistemas, como:

● **Sistemas bancarios:** Cuando realizas una transferencia de dinero en línea, se utiliza una transacción para asegurar que el dinero se deduzca de una cuenta y se acredite en otra de manera precisa.

● **Sistemas de reserva**: Al reservar un vuelo o una habitación de hotel, se utiliza una transacción para garantizar que la disponibilidad se actualice correctamente y que no se realicen reservas duplicadas.

● **Sistemas de gestión de inventarios:** Cuando se registra una venta o se actualiza el inventario de productos, una transacción asegura que los datos se actualicen de manera coherente.

Primitivas para el manejo de transacciones

● **BEGIN TRAN:** comienza una transacción y aumenta en 1 @@TRANCOUNT.

● **COMMIT TRAN:** reduce en 1 @@TRANCOUNT y si @@TRANCOUNT llega a 0 guarda la transacción.

● **ROLLBACK TRAN**: deshace la transacción actual, o si estamos en transacciones anidadas deshace la más externa y todas las internas. Además pone @@TRANCOUNT a 0.

● **SAVE TRAN:** guarda un punto (con nombre) al que podemos volver con un ROLLBACK TRAN si es que estamos en transacciones anidadas y no queremos deshacer todo hasta la más externa.

● **ABORT_TRANSACTION**: deshacer operación.

Tipos de Transacciones Clasificación de acuerdo a su estructura

Transacciones planas: Estas transacciones tienen un punto de partida simple

(Begin_transaction) y un punto simple de terminación (End_transaction).

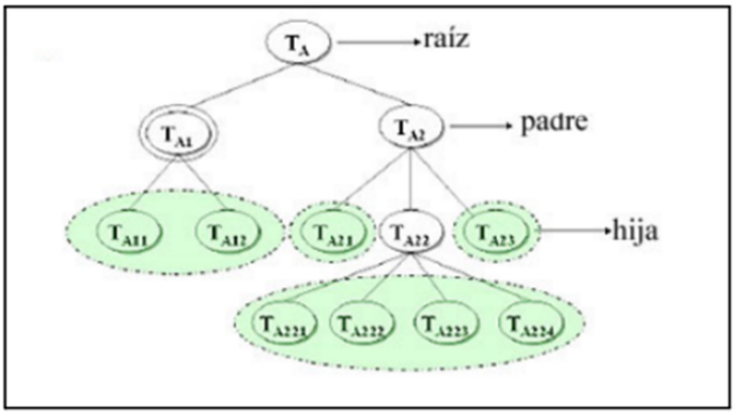
```
Begin_transaction RESERVAR
begin
      EXEC SQL   UPDATE cuentas SET saldo=saldo-1000 WHERE id_cuenta=1111
      EXEC SQL   UPDATE cuentas SET saldo=saldo+1000 WHERE id_cuenta=222
end
```

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}
```

Una transacción anidada dentro de otra transacción conserva las mismas propiedades que la de sus padres, esto implica, que puede contener así mismo transacciones dentro de ella.



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).

Estados de una transacción

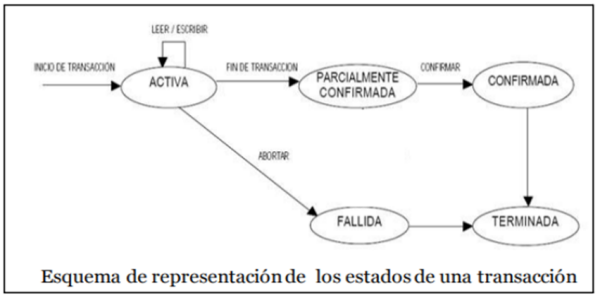
● Transacción Activa: se encuentra en este estado justo después de iniciar su ejecución.

● 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.



Esquema de representación de los estados de una transacción

TRIGGERS

Un trigger es un script que se usa en lenguaje de programación SQL. Consiste en una serie de reglas predefinidas que se asocian a una tabla. Estas reglas se aplican a la base de datos cuando se realizan determinadas operaciones en la tabla, por ejemplo, al añadir, actualizar o eliminar registros. Dicho de otra manera, el trigger desencadena determinadas acciones de forma automática en las tablas de la base de datos cuando se insertan, modifican y se añaden nuevos datos. Generalmente es muy utilizado para auditorías en empresas. Por otra parte, entre sus principales ventajas es que todas estas funciones se pueden realizar desde la propia base de datos, es decir, no es necesario recurrir a lenguajes externos de programación.

Pasos para seguir para la creación y utilización del trigger

1. Creo la tabla auxiliar auditoria_conserje, la cual tendrá los mismos atributos que la tabla conserje, agregando tres atributos más los cuales serán explicados en el diccionario de datos

```
create table auditoria_conserje(
idconserje int,
apeynom varchar(50),
tel varchar(50),
fechnac date,
estciv varchar (1),
fechaModif datetime,
usuario_accion varchar(100),
accion varchar (20)
);
```

2. Luego creo los triggers, para que sean activados de forma automáticamente al realizar un insert, update o delete sobre la tabla conserje. Estos triggers pueden hacerlos todo juntos, pero a modo de mejor comprensión de los hará de forma separada

```
create trigger  trg_auditoria_conserje_insertar
on conserje  for insert
as
declare  @idconserje int, @apeynom varchar(50), @tel varchar(50), @fechnac date,
@estciv varchar (1),
       @fechaModif date,@usuario varchar(50),@accion varchar(20)
select  @idconserje= idconserje,@apeynom=ApeyNom,
@tel=tel,@fechnac=fechnac,@estciv=estciv,
@fechaModif=getdate(),@usuario=SYSTEM_USER from inserted
insert into auditoria_conserje values (@idconserje, @apeynom,@tel, @fechnac,
@estciv,
       getdate(),@usuario,'accion insert')


create trigger  trg_auditoria_conserje_modificar
on conserje  for update
as
declare  @idconserje int, @apeynom varchar(50), @tel varchar(50), @fechnac date,
@estciv varchar (1),
       @fechaModif date,@usuario varchar(50)
select  @idconserje= idconserje,@apeynom=ApeyNom,
@tel=tel,@fechnac=fechnac,@estciv=estciv,
@fechaModif=getdate(),@usuario=SYSTEM_USER  from inserted
insert into auditoria_conserje values (@idconserje, @apeynom,@tel, @fechnac,
@estciv,
       getdate(),@usuario,'accion modify')



create trigger  trg_auditoria_conserje_borrar
on conserje  for update
as
declare  @idconserje int, @apeynom varchar(50), @tel varchar(50), @fechnac date,
@estciv varchar (1),
       @fechaModif date,@usuario varchar(50)
select  @idconserje= idconserje,@apeynom=ApeyNom,
@tel=tel,@fechnac=fechnac,@estciv=estciv,
@fechaModif=getdate(),@usuario=SYSTEM_USER  from deleted
insert into auditoria_conserje values (@idconserje, @apeynom,@tel, @fechnac,
@estciv,
       getdate(),@usuario,'accion delete')
```

3. Para corroborar si los triggers están activados sobre la tabla conserje ejecuto la siguiente sentencia:

```
sp_helptrigger conserje
```

CAPÍTULO III: METODOLOGÍA SEGUIDA

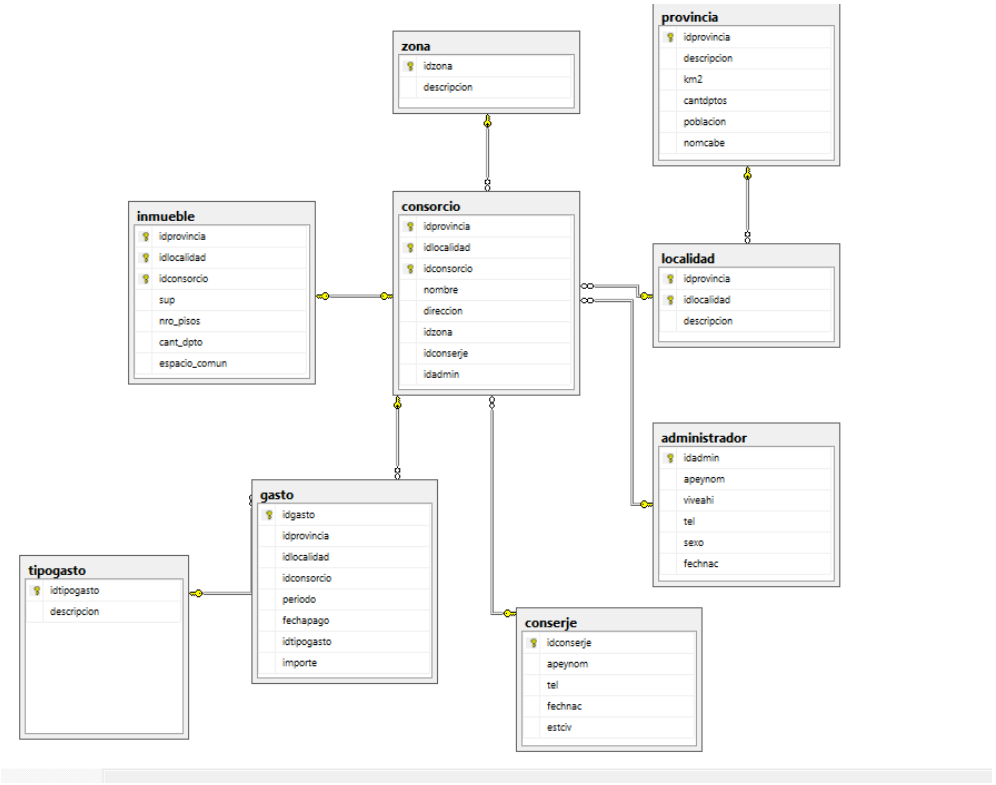
Herramientas utilizadas

Las herramientas que utilizamos para la realización del proyecto fueron las detalladas a continuación

- *WhatsApp*
- *GitHub*
- *SQL Server Management Studio*
- *Google Docs*

CAPÍTULO IV: DESARROLLO DEL TEMA / PRESENTACIÓN DE RESULTADOS

Imagen del diagrama de entidad relación de la base de datos utilizada para realizar el caso de estudio



Diccionario de Datos:

Descripción de las todas las entidades y sus relaciones

provincia: esta tabla contiene datos sobre las provincias en las que se pueden encontrar los consorcios

localidad: esta tabla contiene el nombre de las localidades de las distintas provincias, se relaciona con la tabla: provincia

zona: esta tabla contiene los nombres de las zonas donde pueden encontrarse los consorcios

conserje: esta tabla contiene los datos personales de los conserjes que trabajan en los distintos consorcios.

administrador: esta tabla contiene los datos personales de los distintos administradores de los distintos consorcios.

tipogasto: esta tabla almacena todos los tipos de gastos que se pueden dar en los consorcios

consorcio: esta tabla almacena los datos de todos los consorcios, se relaciona con las tablas: provincia, localidad, zona, conserje y admin

gasto: esta tabla contiene el importe de todos los gastos realizados por los distintos consorcios, se relaciona con las tablas: provincia, localidad, consorcio y tipogasto

Provincia			
Nombre	Tipo(longitud)	Acepta Valores Null	Descripcion
idprovincia	INT	NO	Clave primaria para identificar la provincia
descripcion	VARCHAR(50)	SI	Nombre de la provincia
km2	INT	SI	km2 que tiene la provincia
cantdptos	INT	SI	Cantidad de departamentos que tiene la provincia
poblacion	INT	SI	Numero de avitantes que tiene la poblacion
nomcabe	VARCHAR(50)	SI	Nombre de la capital de la provincia

Localidad			
Nombre	Tipo(longitud)	Acepta Valores Null	Descripcion
idprovincia	INT	NO	Clave foranea para identificar la provincia
idlocalidad	INT	NO	Clave primaria para identificar la localidad
descripcion	VARCHAR(50)	SI	Nombre de la localidad

Zona			
Nombre	Tipo(longitud)	Acepta Valores Null	Descripcion
idzona	INT	NO	Clave primaria para identificar la zona
descripcion	VARCHAR(50)	SI	Nombre de la zona

Conserje			
Nombre	Tipo(longitud)	Acepta Valores Null	Descripcion
idconserje	INT	NO	Clave primaria para identificar al conserje
apeynom	VARCHAR(50)	SI	Apellido y nombre del conserje
tel	VARCHAR(20)	SI	Telefono del conserje
fechnac	DATETIME	SI	fecha de nacimiento del conserje
estciv	VARCHAR(1)	NO	estado civil del conserje

Administrador			
Nombre	Tipo(longitud)	Acepta Valores Null	Descripcion
idadmin	INT	NO	Clave primaria para identificar al administrador
apeynom	VARCHAR(50)	SI	Apellido y Nombre del administrador
viveahi	VARCHAR(1)	NO	Indica si el administrador vive o no en la localidad
tel	VARCHAR(20)	SI	Telefono del adminstrador
sexo	VARCHAR(1)	NO	Sexo del administrador
fechnac	DATETIME	SI	fecha de nacimiento del administrador

TipoGasto			
Nombre	Tipo(longitud)	Acepta Valores Null	Descripcion
idtipogasto	INT	NO	Clave primaria para identificar el tipo de gasto
descripcion	VARCHAR(50)	SI	Nombre del tipo de gasto

Consorcio			
Nombre	Tipo(longitud)	Acepta Valores Null	Descripcion
idprovincia	INT	NO	Clave foranea para identificar la provincia
idlocalidad	INT	NO	Clave foranea para identificar la localidad
idconsorcio	INT	NO	Clave primaria para identificar el consorcio
nombre	VARCHAR(50)	SI	Nombre del consorcio
direccion	VARCHAR(250)	SI	Direccion del consorcio
idzona	INT	NO	Clave foranea para identificar la zona
idconserje	INT	NO	Clave foranea para identificar al conserje
idadmin	INT	NO	Clave foranea para identificar al admin

Gasto			
Nombre	Tipo(longitud)	Acepta Valores Null	Descripcion
idgasto	INT	NO	Clave primaria para identificar el gasto
idprovincia	INT	NO	Clave foranea para identificar la provincia
idlocalidad	INT	NO	Clave foranea para identificar la localidad
idconsorcio	INT	NO	Clave foranea para identificar el consorcio
periodo	INT	NO	Numero del perioro en que se realizo el gasto
fechapago	DATETIME	SI	fecha en la que se realizo el gasto
idtipogasto	INT	NO	Clave foranea para identificar el tipogasto
importe	DECIMAL(8,2)	SI	importe total del gasto

Restricciones

Nombre Entidad	Tipo de restriccion	Nombre de restriccion	Columna
provincia	PRIMARY KEY	PK__provinci__1AF96EE85C28B	idprovincia
localidad	PRIMARY KEY FOREIGN KEY	PK_localidad	idlocalidad, idprovincia
localidad	FOREIGN KEY	FK_localidad_pcia	idprovincia
zona	PRIMARY KEY	PK__zona__D75A19E2E6BBAE23	idzona
conserje	PRIMARY KEY	PK__conserje__095979F6436082	idconserje
conserje	CHECK	CK_estadocivil	estciv
administrador	PRIMARY KEY	PK__administ__5B93BD6E7A630	idadmin
administrador	CHECK	CK_habitante_viveahi	viveahi
administrador	CHECK	CK_sexo	sexo
tipogasto	PRIMARY KEY	PK__tipogast__8CBCB42109CAD6	idtipogasto
consorcio	PRIMARY KEY FOREIGN KEY	PK_consorcio	idlocalidad, idprovincia, idconsorcio
consorcio	FOREIGN KEY	FK_consorcio_pcia	idlocalidad, idprovincia
consorcio	FOREIGN KEY	FK_consorcio_zona	idzona
consorcio	FOREIGN KEY	FK_consorcio_conserje	idconserje
consorcio	FOREIGN KEY	FK_consorcio_admin	idadmin
gasto	PRIMARY KEY	PK_gasto	idgasto
gasto	FOREIGN KEY	FK_gasto_consorcio	idlocalidad, idprovincia, idconsorcio
gasto	FOREIGN KEY	FK_gasto_tipo	idtipogasto

Procedimientos y Funciones

A continuación se mostraran capturas de pantallas de los códigos de procedimientos y la función realizados

```sql
-- ============================================
-- Author:      <Peloso Nicolas>
-- Description: <Agrega un nuevo registro de consorcio en la BD>
-- ============================================

USE base_consorcio
GO

CREATE PROCEDURE agregarConsorcio
@Provincia INT,
@Localidad INT,
@Consorcio INT,
@Nombre VARCHAR(50),
@Direccion VARCHAR(255),
@Zona INT,
@Conserje INT,
@Admin INT

      AS
BEGIN
        INSERT INTO consorcio(idprovincia,idlocalidad,idconsorcio,nombre,direccion, idzona,idconserje,idadmin)
            VALUES (@Provincia,@Localidad,@Consorcio,@Nombre,@Direccion, @Zona, @Conserje, @Admin)

END

--DROP PROCEDURE agregarConsorcio
--Para probar
/*EXECUTE agregarConsorcio @Provincia=5, @Localidad=6, @Consorcio= 50, @Nombre='Nicolás', @Direccion='General Viamonte 1658',
@Zona=3,@Conserje='2',@Admin=1
SELECT * FROM consorcio WHERE idConsorcio=50 */
```

```sql
-- ============================================
-- Author:      <Peloso Nicolas>
-- Description: <Modifica un registro existente de consorcio en la BD>
-- ============================================

USE base_consorcio
GO

CREATE PROCEDURE modificarConsorcio
@Provincia INT,
@Localidad INT,
@Consorcio INT,
@Nombre VARCHAR(50),
@Direccion VARCHAR(255),
@Zona INT,
@Conserje INT,
@Admin INT

      AS
BEGIN

    IF @Consorcio IS NOT NULL
        UPDATE consorcio SET idprovincia= @Provincia, idlocalidad= @Localidad,nombre= @Nombre,direccion= @Direccion,idzona= @Zona,
        idconserje= @Conserje,idadmin= @Admin
        WHERE idconsorcio= @Consorcio
END

--DROP PROCEDURE modificarConsorcio
--Para probar
/*EXECUTE modificarConsorcio @Provincia=6, @Localidad=5, @Consorcio= 50, @Nombre='Anibal', @Direccion='Juan Jose Castelli 1217',
@Zona=4, @Conserje='1', @Admin=2
SELECT * FROM consorcio WHERE idConsorcio=50
```

```sql
-- ============================================
-- Author:      <Peloso Nicolas>
-- Description: <Elimina un registro de consorcio en la BD>
-- ============================================

USE base_consorcio
GO

CREATE PROCEDURE eliminarConsorcio
@Consorcio INT

AS
BEGIN

    DELETE FROM consorcio WHERE idconsorcio=@Consorcio
END

--DROP PROCEDURE eliminarConsorcio
--Para probar
/*EXECUTE eliminarConsorcio @Consorcio=50
SELECT * FROM consorcio WHERE idconsorcio=50 */
```

```sql
-- ============================================
-- Author:      <Peloso Nicolas>
-- ============================================
USE base_consorcio
GO

/*Devuelve la edad del admin*/
CREATE FUNCTION calcularEdad(@ID INT)
RETURNS VARCHAR(200)
AS
-- Returns the stock level for the product.
BEGIN
    DECLARE @Edad VARCHAR(200)

    SELECT @Edad= DATEDIFF(YEAR,fechnac, GETDATE())
        FROM administrador
    WHERE idadmin= @ID
    RETURN @Edad
END;

--DROP FUNCTION calcularEdad

/*
Para probar

SELECT *, dbo.calcularEdad(idadmin) AS Edad FROM administrador;
*/
```

Backups y Restore

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

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 capturaron 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.

Resultados

A continuación, se presentan capturas de pantalla del código realizado y los resultados obtenidos.

```
-- 1° se verifica el modo de recuperacion de la base de datos
use base_consorcio

-- Verificar el modo de backup de la base de datos

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


-- 2° cambiamar el modo de recuperacion

USE master; -- Asegúrarse de estar en el contexto de la base de datos master
ALTER DATABASE base_consorcio -- Base de datos a la que se hará el backup
SET RECOVERY FULL; -- Modo de recuperacion

-- Posibles modos de backup: "SIMPLE", "BULK_LOGGED", "FULL"


-- 3 Realizacion del backup de la base de datos

BACKUP DATABASE base_consorcio
TO DISK = 'C:\backup\consorcioBackup.bak' --Ruta para guardar el archivo de Backup.bak'
WITH FORMAT, INIT;
```

```sql
-- Para Realizar la prueba

-- Se ingresaron 10 registros

select * from gasto;

insert into gasto (idconsorcio,periodo,fechapago,idtipogasto,importe) values (1,5,GETDATE(),4,2100);

insert into gasto (idconsorcio,periodo,fechapago,idtipogasto,importe) values (2,3,GETDATE(),1,76312);

insert into gasto (idconsorcio,periodo,fechapago,idtipogasto,importe) values (3,8,GETDATE(),3,1000);

insert into gasto (idconsorcio,periodo,fechapago,idtipogasto,importe) values (4,4,GETDATE(),3,20431);

insert into gasto (idconsorcio,periodo,fechapago,idtipogasto,importe) values (5,11,GETDATE(),5,20128);

insert into gasto (idconsorcio,periodo,fechapago,idtipogasto,importe) values (6,10,GETDATE(),4,500);

insert into gasto (idconsorcio,periodo,fechapago,idtipogasto,importe) values (7,6,GETDATE(),1,47180);

insert into gasto (idconsorcio,periodo,fechapago,idtipogasto,importe) values (8,9,GETDATE(),2,2510);

insert into gasto (idconsorcio,periodo,fechapago,idtipogasto,importe) values (9,1,GETDATE(),5,20150);

insert into gasto (idconsorcio,periodo,fechapago,idtipogasto,importe) values (10,2,GETDATE(),3,33123);
```

```sql
-- 4 Realizacion del backup del log de la base de datos completa

BACKUP LOG base_consorcio
TO DISK = 'C:\backup\LogBackup.trn' --Ruta para el archivo LogBackup.trn
WITH FORMAT, INIT;


-- Se ingreasan 10 registros mas

select * from gasto

insert into gasto (idconsorcio,periodo,fechapago,idtipogasto,importe) values (15,1,GETDATE(),1,12700);

insert into gasto (idconsorcio,periodo,fechapago,idtipogasto,importe) values (11,1,GETDATE(),1,5100);

insert into gasto (idconsorcio,periodo,fechapago,idtipogasto,importe) values (12,11,GETDATE(),3,3200);

insert into gasto (idconsorcio,periodo,fechapago,idtipogasto,importe) values (13,7,GETDATE(),2,72000);

insert into gasto (idconsorcio,periodo,fechapago,idtipogasto,importe) values (14,10,GETDATE(),5,5100);

insert into gasto (idconsorcio,periodo,fechapago,idtipogasto,importe) values (16,7,GETDATE(),4,2200);

insert into gasto (idconsorcio,periodo,fechapago,idtipogasto,importe) values (18,9,GETDATE(),3,13100);

insert into gasto (idconsorcio,periodo,fechapago,idtipogasto,importe) values (19,4,GETDATE(),2,57000);

insert into gasto (idconsorcio,periodo,fechapago,idtipogasto,importe) values (20,11,GETDATE(),5,2100);

insert into gasto (idconsorcio,periodo,fechapago,idtipogasto,importe) values (21,2,GETDATE(),1,100);
```

```sql
--5 Realizacion del backup del log en una ubicacion diferente

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

--6 Restauracion del backup de la base de datos

use master

-- Para restaurar una base de datos desde un archivo de respaldo

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

-- Restaurar un archivo de registro de transacciones

RESTORE LOG base_consorcio
FROM DISK = 'C:\backup\LogBackup.trn'
WITH RECOVERY; --Indica que la base de datos se dejará en un estado de recuperación completa  y permitirá que la base de datos esté
--disponible para su uso normal.

-- Segundo log

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

select * from gasto
```

Estado guardado en el backup

Results | Messages

	idgasto	idprovincia	idlocalidad	idconsorcio	periodo	fechapago	idtipogasto	importe
7...	7993	24	17	6	1	2017-01-21 00:00:00.000	3	37883.07
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...	8002	NULL	NULL	2	3	2023-11-20 10:48:54.427	1	76312.00
8...	8003	NULL	NULL	3	8	2023-11-20 10:48:54.430	3	1000.00
8...	8004	NULL	NULL	4	4	2023-11-20 10:48:54.430	3	20431.00
8...	8005	NULL	NULL	5	11	2023-11-20 10:48:54.430	5	20128.00
8...	8006	NULL	NULL	6	10	2023-11-20 10:48:54.430	4	500.00
8...	8007	NULL	NULL	7	6	2023-11-20 10:48:54.430	1	47180.00
8...	8008	NULL	NULL	8	9	2023-11-20 10:48:54.430	2	2510.00
8...	8009	NULL	NULL	9	1	2023-11-20 10:48:54.430	5	20150.00
8...	8010	NULL	NULL	10	2	2023-11-20 10:48:54.430	3	33123.00
8...	8011	NULL	NULL	1	5	2023-11-20 10:49:16.547	4	2100.00

Query executed successfully. | DESKTOP-2MBNP1G\SQLEXPRESS ... | DESKTOP-2MBNP1G\Nicola... | base_consorcio | 00:00:00 | 8010 rows

Estado final con los 20 registros guardados

Results | Messages

	idgasto	idprovincia	idlocalidad	idconsorcio	periodo	fechapago	idtipogasto	importe
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...	8013	NULL	NULL	1	5	2023-11-19 22:55:14.730	4	2100.00
8...	8017	NULL	NULL	2	3	2023-11-19 22:58:20.163	1	76312.00
8...	8018	NULL	NULL	3	8	2023-11-19 22:58:20.163	3	1000.00
8...	8019	NULL	NULL	4	4	2023-11-19 22:58:20.163	3	20431.00
8...	8020	NULL	NULL	5	11	2023-11-19 22:58:20.163	5	20128.00
8...	8022	NULL	NULL	7	6	2023-11-19 22:58:20.167	1	47180.00
8...	8023	NULL	NULL	8	9	2023-11-19 22:58:20.167	2	2510.00
8...	8024	NULL	NULL	9	1	2023-11-19 22:58:20.167	5	20150.00
8...	8025	NULL	NULL	10	2	2023-11-19 22:58:20.167	3	33123.00
8...	8026	NULL	NULL	6	10	2023-11-19 22:58:57.593	4	500.00
8...	9001	NULL	NULL	15	1	2023-11-19 23:11:31.777	1	12700.00
8...	9002	NULL	NULL	11	1	2023-11-19 23:11:31.777	1	5100.00
8...	9003	NULL	NULL	12	11	2023-11-19 23:11:31.777	3	3200.00
8...	9004	NULL	NULL	13	7	2023-11-19 23:11:31.777	2	72000.00
8...	9006	NULL	NULL	16	7	2023-11-19 23:11:31.780	4	2200.00
8...	9007	NULL	NULL	18	9	2023-11-19 23:11:31.780	3	13100.00
8...	9008	NULL	NULL	19	4	2023-11-19 23:11:31.780	2	57000.00
8...	9009	NULL	NULL	20	11	2023-11-19 23:11:31.780	5	2100.00
8...	9010	NULL	NULL	21	2	2023-11-19 23:11:31.780	1	100.00
8...	9011	NULL	NULL	14	10	2023-11-19 23:12:17.747	5	5100.00

Query executed successfully. | DESKTOP-2MBNP1G\SQLEXPRESS ... | DESKTOP-2MBNP1G\Nicola... | master | 00:00:01 | 8020 rows

Vistas

```
4  ------------------------------------
5  -- CREAR VIEW conserje_view
6  ------------------------------------
7  CREATE VIEW conserje_view AS
8  (
9      SELECT
10         apeynom,
11         tel,
12         fechnac
13     FROM  [dbo].[conserje]
14 )
15 GO
16 ------------------------------------
17 -- Ver los datos en la view
18 ------------------------------------
19 SELECT * FROM conserje_view
```

100 %

Results | Messages

	apeynom	tel	fechnac
1	Camilo Sexto de Guadalajara	374449272	1983-09-10 00:00:00.000
2	ALCARAZ SONIA	374449272	1970-07-10 00:00:00.000
3	ALMADA DE R. EMERENCIANA	374449372	1984-04-01 00:00:00.000
4	RAMIREZ JORGE ESTEBAN	373449472	1969-02-12 00:00:00.000
5	MARIN DE P. ANTOLINA	373449572	1995-06-27 00:00:00.000
6	GAYOZO RAMON E.	373449672	1966-11-08 00:00:00.000
7	BILLORDO GABRIEL B.	373449772	1976-06-18 00:00:00.000
8	MONTENEGRO PETRONA A.	372449872	1978-04-27 00:00:00.000
9	PAREDES GRACIELA NOEMI	372449972	1992-03-24 00:00:00.000
10	RAMIREZ RAMONA LA PAZ	372449072	1985-02-19 00:00:00.000
11	MACIEL SEGUNDA	372449172	1994-04-20 00:00:00.000
12	BLANCO MARIA ESCOLASTICA	372449272	1988-03-03 00:00:00.000
13	SAUCEDO PANTALEON	372449372	1986-04-17 00:00:00.000
14	PALERMO JOSE	375449472	1976-02-02 00:00:00.000

```
24  ------------------------------------
25  ALTER VIEW [dbo].[conserje_view] AS
26  (
27      SELECT
28          apeynom,
29          tel,
30          fechnac,
31          estciv --ESTE ES EL NUEVO CAMPO
32      FROM  [dbo].[conserje]
33  )
34  GO
35  ------------------------------------
36  -- Ver los cambios en la view
37  ------------------------------------
38  SELECT * FROM conserje_view
39
```

00 %

Results | Messages

	apeynom	tel	fechnac	estciv
1	Camilo Sexto de Guadalajara	374449272	1983-09-10 00:00:00.000	S
2	ALCARAZ SONIA	374449272	1970-07-10 00:00:00.000	C
3	ALMADA DE R. EMERENCIANA	374449372	1984-04-01 00:00:00.000	S
4	RAMIREZ JORGE ESTEBAN	373449472	1969-02-12 00:00:00.000	C
5	MARIN DE P. ANTOLINA	373449572	1995-06-27 00:00:00.000	S
6	GAYOZO RAMON E.	373449672	1966-11-08 00:00:00.000	C
7	BILLORDO GABRIEL B.	373449772	1976-06-18 00:00:00.000	S
8	MONTENEGRO PETRONA A.	372449872	1978-04-27 00:00:00.000	C
9	PAREDES GRACIELA NOEMI	372449972	1992-03-24 00:00:00.000	S
10	RAMIREZ RAMONA LA PAZ	372449072	1985-02-19 00:00:00.000	C
11	MACIEL SEGUNDA	372449172	1994-04-20 00:00:00.000	S
12	BLANCO MARIA ESCOLASTICA	372449272	1988-03-03 00:00:00.000	C
13	SAUCEDO PANTALEON	372449372	1986-04-17 00:00:00.000	S
14	PALERMO JOSE	375449472	1976-02-02 00:00:00.000	C

Query executed successfully. | DESKTOP-2MBNP1G\SQLEXPRESS ... | DESKTOP-2MBNP1G\Nicola... | base_consorcio | 00:00:00 | 220 rows

```
51  DROP VIEW IF EXISTS [dbo].[conserje_view]
52
53
54  ------------------------------------
55  -- ACTUALIZAR VIEW conserje_view
56  ------------------------------------
57  UPDATE [dbo].[conserje]
58  SET
59        apeynom = 'Camilo Sexto de Guadalajara'
60  WHERE idconserje = 1
61
62  ------------------------------------
63  -- Ver los cambios en la view
64  ------------------------------------
65  SELECT * FROM conserje_view
```

100 %

Results | Messages

	apeynom	tel	fechnac	estciv
1	Camilo Sexto de Guadalajara	374449272	1983-09-10 00:00:00.000	S
2	ALCARAZ SONIA	374449272	1970-07-10 00:00:00.000	C
3	ALMADA DE R. EMERENCIANA	374449372	1984-04-01 00:00:00.000	S
4	RAMIREZ JORGE ESTEBAN	373449472	1969-02-12 00:00:00.000	C
5	MARIN DE P. ANTOLINA	373449572	1995-06-27 00:00:00.000	S
6	GAYOZO RAMON E.	373449672	1966-11-08 00:00:00.000	C
7	BILLORDO GABRIEL B.	373449772	1976-06-18 00:00:00.000	S
8	MONTENEGRO PETRONA A.	372449872	1978-04-27 00:00:00.000	C
9	PAREDES GRACIELA NOEMI	372449972	1992-03-24 00:00:00.000	S
10	RAMIREZ RAMONA LA PAZ	372449072	1985-02-19 00:00:00.000	C
11	MACIEL SEGUNDA	372449172	1994-04-20 00:00:00.000	S
12	BLANCO MARIA ESCOLASTICA	372449272	1988-03-03 00:00:00.000	C
13	SAUCEDO PANTALEON	372449372	1986-04-17 00:00:00.000	S
14	PALERMO JOSE	375449472	1976-02-02 00:00:00.000	C

Query executed successfully. | DESKTOP-2MBNP1G\SQLEXPRESS ... | DESKTOP-2MBNP1G\Nicola... | base_consorcio | 00:00:00 | 220 rows

```
41  ------------------------------------
42  -- ELIMINAR VIEW conserje_view
43  ------------------------------------
44  IF OBJECT_ID('[dbo].[conserje_view]', 'V') IS NOT NULL
45  DROP VIEW [dbo].[conserje_view]
46
47
48  ------------------------------------
49  -- ELIMINAR VIEW conserje_view
50  ------------------------------------
51  DROP VIEW IF EXISTS [dbo].[conserje_view]
52
53
54  ------------------------------------
```

00 %

Messages

Command(s) completed successfully.

Vistas Indexadas

```
USE base_consorcio

-------------------------------------
-- CREAR VISTA INDEXADA ConsorcioDetalles_view
-------------------------------------
CREATE VIEW ConsorcioDetalles_view AS
(
    SELECT
        AdministradorApeyNom = ADM.apeynom,
        ConsorcioNombre      = CON.nombre,
        Periodo              = G.periodo,
        FechaPago            = G.fechapago
    FROM  [dbo].[consorcio] CON
    INNER JOIN [dbo].[administrador] ADM
        ON CON.idadmin = ADM.idadmin
    INNER JOIN [dbo].[gasto] G
        ON CON.idconsorcio = G.idconsorcio
    INNER JOIN [dbo].[tipogasto] TG
        ON G.idtipogasto = TG.idtipogasto
)
GO
-------------------------------------
-- Ver los datos en la view
-------------------------------------
SELECT * FROM [dbo].[ConsorcioDetalles_view]
```

Results | Messages

	AdministradorApeyNom	ConsorcioNombre	Periodo	FechaPago
1	Perez Juan Manuel	EDIFICIO-111	6	2013-06-16 00:00:00.000
2	SEGOVIA ALEJANDRO H.	EDIFICIO-2481	6	2013-06-16 00:00:00.000
3	NAHMIAS DE K. NIDIA	EDIFICIO-3161	6	2013-06-16 00:00:00.000
4	GOMEZ MATIAS GABRIEL	EDIFICIO-4211	6	2013-06-16 00:00:00.000
5	CARDOZO MAXIMA	EDIFICIO-531	6	2013-06-16 00:00:00.000
6	RATTI JUAN E.	EDIFICIO-6331	6	2013-06-16 00:00:00.000
7	MARTINEZ EDUARDO.	EDIFICIO-2011	6	2013-06-16 00:00:00.000
8	BENITEZ ANSELMA B.	EDIFICIO-21181	6	2013-06-16 00:00:00.000
9	DEL VALLE ANDRES	EDIFICIO-2231	6	2013-06-16 00:00:00.000
10	TORRES LUIS.	EDIFICIO-2311	6	2013-06-16 00:00:00.000
11	TORRES ELADIA	EDIFICIO-2441	6	2013-06-16 00:00:00.000
12	VALLEJOS CYNTHIA ELIZABET	EDIFICIO-1481	6	2013-06-16 00:00:00.000
13	GARCIA MARIA F.	EDIFICIO-1551	6	2013-06-16 00:00:00.000
14	PARRAS DE ESQUIVEL MARIA	EDIFICIO-1641	6	2013-06-16 00:00:00.000

```sql
-------------------------------------
-- MODIFICAR VISTA INDEXADA ConsorcioDetalles_view
-------------------------------------
ALTER VIEW [dbo].[ConsorcioDetalles_view]
WITH SCHEMABINDING
AS
(

    SELECT
        IdGasto         = g.idgasto,
        Administrador   = adm.apeynom,
        ConsorcioNombre = c.nombre,
        Periodo         = g.periodo,
        fechapago       = g.fechapago,
        TipoGasto       = tg.descripcion
    FROM [dbo].[gasto] g
    INNER JOIN [dbo].[tipogasto] tg
        ON g.idtipogasto = tg.idtipogasto
    INNER JOIN  [dbo].[consorcio] c
        ON g.idconsorcio = c.idconsorcio
        AND g.idprovincia = c.idprovincia
        AND g.idlocalidad = c.idlocalidad
    INNER JOIN [dbo].[administrador] adm
        ON c.idadmin = adm.idadmin
)
GO

SELECT * FROM [dbo].[ConsorcioDetalles_view]
```

Results | Messages

	IdGasto	Administrador	ConsorcioNombre	Periodo	fechapago	TipoGasto
1	1	Perez Juan Manuel	EDIFICIO-111	6	2013-06-16 00:00:00.000	OTROS
2	2	Perez Juan Manuel	EDIFICIO-111	3	2013-03-11 00:00:00.000	Sueldos
3	3	Perez Juan Manuel	EDIFICIO-111	7	2013-07-09 00:00:00.000	Sueldos
4	4	Perez Juan Manuel	EDIFICIO-111	7	2013-07-08 00:00:00.000	Sueldos
5	5	Perez Juan Manuel	EDIFICIO-111	8	2013-08-14 00:00:00.000	Limpieza
6	6	Perez Juan Manuel	EDIFICIO-111	1	2013-01-25 00:00:00.000	Aportes
7	7	Perez Juan Manuel	EDIFICIO-111	7	2013-07-26 00:00:00.000	Aportes
8	8	Perez Juan Manuel	EDIFICIO-111	8	2013-08-03 00:00:00.000	Sueldos
9	9	Perez Juan Manuel	EDIFICIO-111	8	2013-08-22 00:00:00.000	Sueldos
10	10	Perez Juan Manuel	EDIFICIO-111	3	2013-03-26 00:00:00.000	Sueldos
11	11	BASUALDO DELMIRA	EDIFICIO-122	3	2013-03-28 00:00:00.000	Limpieza

Query executed successfully. | DESKTOP-2MBNP1G\SQLEXPRESS ... | DESKTOP-2MBNP1G\Nicola... | base_consorcio | 00:00:00 | 8000 rows

Manejo de Permisos a nivel de Usuario de Base de Datos

```sql
/*
Creamos usuarios de prubea
*/

CREATE LOGIN UsuarioAdminitrador WITH PASSWORD = 'admin123';
CREATE LOGIN UsuarioComum WITH PASSWORD = 'contrausuario';

CREATE USER UsuarioAdministrador FOR LOGIN UsuarioAdministrador ;
CREATE USER UsuarioComum FOR LOGIN UsuarioComum;


/*

Creamos dos roles
*/

CREATE ROLE Administrador;
CREATE ROLE Comun;


/*
Le damos permisos  a los roles
*/

GRANT SELECT TO Administrador;


GRANT CREATE TABLE TO Comun;
GRANT ALTER ON SCHEMA::dbo TO Comun;

/*

Le asignamos a cada role los usuarios
*/

ALTER ROLE Administrador ADD MEMBER UsuarioAdministrador;
ALTER ROLE Comun ADD MEMBER UsuarioComun;


--Pruebas

--NOTA: Es mejor probar en querys diferentes para evitar algun problema

/*
Testeo el funciomiento del rol de administrador
*/

EXECUTE AS USER = 'UsuarioAdministrador';
SELECT * FROM conserje;

/*
Ejemplo de lo que no puede hacer
CREATE TABLE tablaEjemplo1 (ID INT, Nombre VARCHAR(50));
*/

REVERT;


/*
Testeo el funcionamiento del rol de comun
*/
EXECUTE AS USER = 'UsuarioComun';
CREATE TABLE tablaEjemplo (ID INT, Nombre VARCHAR(50));
ALTER TABLE tablaEjemplo ADD Descripcion VARCHAR(100);
/*
Ejemplo de lo que no puede hacer
SELECT * FROM conserje;
*/
```

Messages

```
Msg 229, Level 14, State 5, Line 69
The SELECT permission was denied on the object 'conserje', database 'base_consorcio', schema 'dbo'.
```

```sql
--Probar con permisos de usuarios

--Usuario administrador
--Ejecutar solo para la creacion a nivel de servidor.
USE master;
CREATE LOGIN UserAdmin WITH PASSWORD = 'admin';
--Usuario de Solo Lectura
CREATE LOGIN NormalUser WITH PASSWORD = 'user123';

-- Se asigna el permiso de Admin para UserAdmin
USE [base_consorcio];
CREATE USER UserAdmin FOR LOGIN UserAdmin;
ALTER ROLE db_owner ADD MEMBER UserAdmin

-- Con la palabra 'GRANT' asignamos el permiso SELECT sobre la view [conserje_view]
CREATE USER NormalUser FOR LOGIN NormalUser;
GRANT SELECT ON [dbo].[conserje_view] TO NormalUser;

--Consulta SELECT con el usuario administrador
EXECUTE AS LOGIN = 'UserAdmin'
SELECT * FROM [dbo].[conserje_view]
REVERT --REVERT en SQL Server se utiliza para volver al contexto original

--Consulta SELECT con el usuario observador
EXECUTE AS LOGIN = 'NormalUser'
SELECT * FROM [dbo].[conserje_view]
REVERT

--Al intentar realizar un UPDATE con el 'UsuarioObservador' es rechazado.
USE [base_consorcio];
EXECUTE AS LOGIN = 'NormalUser';
UPDATE [dbo].[conserje_view]
   SET [estciv] = 'C'
 WHERE apeynom LIKE 'ESPINOZA JULIO'
GO
SELECT * FROM [dbo].[conserje_view]
REVERT;
```

CAPÍTULO V: CONCLUSIONES

A lo largo de este proyecto de estudio hemos aprendido que los procedimientos y funciones ofrecen ventajas significativas en la administración de bases de datos como son la capacidad de mejorar la eficiencia y la reutilización del código.
Los procedimientos almacenados permiten agrupar tareas comunes en un solo bloque de código, lo que facilita la ejecución y el mantenimiento de las operaciones en la base de datos.
Además, hemos comprendido que la utilización de parámetros y variables en procedimientos y funciones aumenta la flexibilidad y adaptabilidad de estas herramientas.
Por lo tanto, concluímos que el trabajo con los procedimientos y funciones que implica como diseñar, crear y ejecutar estos elementos son unas herramientas esenciales para cualquier profesional de bases de datos. Este proyecto nos ha permitido adquirir conocimientos valiosos en esta área y aplicarlos de manera práctica al problema dado.

Otras observaciones: Aunque puede parecer que los Procedimientos Almacenados pueden hacer lo mismo o hasta más que una Función, cada una tiene sus propias limitaciones y usos.
Los usos típicos para Procedimientos Almacenados es la validación de datos, integrados dentro de la estructura de la base de datos , la "encapsulación" de un API para un proceso complejo que podría requerir la ejecución de varias consultas SQL, tales como la manipulación de un conjunto de datos enormes para producir un resultado resumido y también pueden ser usados para el control de gestión de operaciones.
Mientras que para las Funciones se usan cuando se necesita recibir un único valor simple que se obtiene de una operación recurrente.

VI. BIBLIOGRAFÍA.

Procedimientos y funciones almacenadas

- Softwero. MySQL: Procedimientos Almacenados vs Funciones http://www.softwero.com/2017/07/mysql-procedimientos-almacenados-vs-funciones.html#:~:text=b

- Microsoft Learn. Crear procedimientos almacenados y funciones definidas por el usuario https://learn.microsoft.com/es-es/training/modules/create-stored-procedures-table-valued-functions/

- Microsoft Learn. Procedimientos almacenados (motor de base de datos) https://learn.microsoft.com/es-es/sql/relational-databases/stored-procedures/stored-procedures-database-engine?view=sql-server-ver16

- Microsoft Learn. Funciones definidas por el usuario https://learn.microsoft.com/es-es/sql/relational-databases/user-defined-functions/user-defined-functions?view=sql-server-ver16

Backup y Restore

- Microsoft Learn. Restauración en línea (SQL Server) https://learn.microsoft.com/es-es/sql/relational-databases/backup-restore/online-restore-sql-server?view=sql-server-ver16#log-backups-for-online-restore

- Microsoft Learn. Copia de seguridad y restauración de bases de datos de SQL Server 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-an d-resources

Réplicas

- Microsoft Learn. Replicación de SQL Server https://learn.microsoft.com/es-es/sql/relational-databases/replication/sql-server-replication?view=sql-server-ver16

Vistas y Vistas Indexadas

- Microsoft Ignite, 18/08/2023, Creacion de Vistas, manejo de vistas aplicadas en Sql Server, Sql Azure server y Azure Sql Managed Instance.
- SqlLearning, 2023, VISTAS EN SQL, (Create, Alter and DROP) en vistas de SQL.
- Newsmactic, 2021, Qué es una vista indexada y cómo mejorar el rendimiento de consultas en SQL Server, definiciones e implementación de vistas indexadas.
- il_masacratore, 11/02/2013, SQL Server: Vistas indizadas y el porqué de usarlas para cargas de dwh, Implementación de vistas indizadas en Sql Server.

Manejo de Permisos a nivel de Usuario de Base de Datos

- Microsoft Learn. Roles en el nivel de base de datos https://learn.microsoft.com/es-es/sql/relational-databases/security/authentication-access/database-level-roles?view=sql-server-ver16
- IBM. https://www.ibm.com/docs/en/db2woc?topic=SS6NHC/com.ibm.swg.im.dashdb.security.doc/doc/udef_user_roles.html
- Dušan Petković,"Microsoft SQL Server 2019 A Begginer's Guide",7a edicion,2019.

Transacciones y transacciones anidadas

Silberchatz, korth, sudarshan. Fundamentos de base de datos (4ta Edición). (pág 367).

-Mercedes Marqués. (2011). Base de datos. https://bdigital.uvhm.edu.mx/wp-content/uploads/2020/05/Bases-de-Datos.pdf

Wikipedia. Transacciones(informática). https://es.wikipedia.org/wiki/Transacci%C3%B3n_(inform%C3%A1tica)#:~:text=Una%20transacci%C3%B3n%20en%20un%20sistema,en%20forma%20indivisible%20o%20at%C3%B3mica

- Transact-SQL-Reference. https://learn.microsoft.com/en-us/sql/t-sql/language-reference?view=sql-server-ver16