UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

# Diplomatura en Bases de Datos

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
**www.sceu.frba.utn.edu.ar/e-learning**

# Módulo 1: Introducción y Fundamentos de Bases de Datos

## Unidad 4: Estructura interna de las bases de datos

# Presentación:

En esta Unidad exploramos la estructura interna de las bases de datos, para lo cual desarrollamos: Log de transacciones, dispositivos de almacenamiento y problemas de crecimiento, estructura de páginas y tareas de mantenimiento de las bases de datos

# Objetivos:

## Que los participantes:

- Entiendan la función del log de transacciones y los puntos de control.
- Manejen el esquema de almacenamiento físico
- Controlen los problemas de crecimiento de los dispositivos y las estrategias asociadas.
- Tengan en cuenta la estructura de páginas dentro de las bases de datos y su implicancia a la hora del diseño de tablas.
- Comprendan la importancia del mantenimiento de las bases de datos y sus procedimientos asociados.

## Bloques temáticos:

1. Log de transacciones.
2. Dispositivos de almacenamiento.
3. Problemas de crecimiento en las bases de datos.
4. Páginas.
5. Mantenimiento de las bases de datos.

# Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

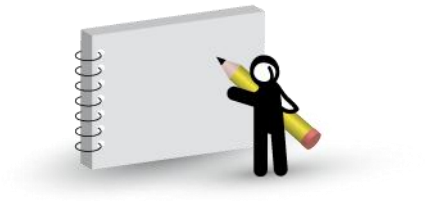
- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

**** El MEC es el modelo de E-learning colaborativo de nuestro Centro.***



## Tomen nota:

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

# 1. Log de transacciones

El primer concepto que necesitamos manejar es el de transacción. Recordemos por ejemplo los asientos en la contabilidad. El debe y el haber deben quedar balanceados. Con todo, cada imputación a una cuenta suele reflejarse en un único registro con lo cual nuestro asiento tendrá que tener, al menos, dos registros.

¿Qué pasaría si uno se carga y el otro no? (Los motivos pueden ser variados, salta un error, se corta la luz, etc) La respuesta es que se generaría una inconsistencia en los datos y queremos evitarlo.

Para evitar la inconsistencia nos gustaría crear una bolsa que contenga todos los datos del asiento. La bolsa nos garantizaría integridad. O se hacen todos los cambios de la bolsa o no se hace ninguno. Este es el concepto de la transacción que es el nombre que le damos dentro del ámbito de las bases de datos a esa bolsa.

Es muy importante que mientras vamos vaciando la bolsa el resto de los que están trabajando sobre la base de datos no vean ningún cambio hasta que la bolsa está completamente vacía.

**Nota Conceptual:**

Según la Wikipedia una transacción es una interacción con una estructura de datos compleja, compuesta por varios procesos que se han de aplicar uno después del otro. La transacción debe realizarse de una sola vez y sin que la estructura a medio manipular pueda ser alcanzada por el resto del sistema hasta que se hayan finalizado todos sus procesos.

Para una transacción vamos a necesitar varias cosas:

- Indicar donde una transacción comienza (principio de la bolsa)
- Indicar donde una transacción termina (fin de la bolsa)
- Como deshacer una transacción a medio camino

Para declarar el inicio de una transacción usamos:

BEGIN TRANSACTION

Para indicar que queremos que ejecute la transacción y haga públicos los cambios:

COMMIT TRANSACTION

Y, finalmente, si queremos renunciar a los cambios y que todo quede como estaba antes de comenzar con la transacción:

ROLLBACK TRANSACTION

Todas las transacciones se van almacenando en un log de transacciones. Un proceso viene escribiendo en forma retardada los cambios que esas transacciones indican en las tablas del sistema.

Si quieren repasarlo en inglés les dejo el vínculo en el cual Microsoft explica el uso del log de transacciones:

https://docs.microsoft.com/en-us/sql/relational-databases/logs/the-transaction-log-sql-server?view=sql-server-2017

Si les interesa aprender sobre la administración del log y su arquitectura interna pueden revisar:

https://docs.microsoft.com/en-us/sql/relational-databases/sql-server-transaction-log-architecture-and-management-guide?view=sql-server-2017

En este punto se recomienda realizar los ejercicios prácticos 4.1 y 4.2

# 2. Dispositivos de almacenamiento

Toda base de datos tiene, por lo menos, dos espacios de almacenamiento (archivos)

Uno es para guardar todos los objetos que hemos venido viendo como tablas, índices, etc.

El otro es para guardar las transacciones.

El espacio de almacenamiento para los datos puede ser Primario o Secundario. Cuando se crea una base de datos se crea tanto su dispositivo primario de almacenamiento como su log de transacciones.

Los dispositivos secundarios son opcionales. Pueden usarse para enviar partes de los datos a otros discos por motivos de performance o cuando se necesita superar el tamaño máximo de almacenamiento que un único archivo permite.

Por defecto el dispositivo de almacenamiento primario de datos y el log de transacciones se crean dentro del mismo directorio. La extensión por defecto para los archivos primarios de datos es mdf, ldf es la extensión por defecto para el log de transacciones y se recomienda, para los archivos secundarios de datos usar ndf.

Para ahorrar espacio de almacenamiento no es posible ubicar los archivos secundarios en una unidad que tenga habilitada la compresión del sistema de archivos. Es conveniente, llegado el caso habilitar la compresión de datos interna de la base de datos y no la compresión del sistema de archivos.

Si se habilitan diferentes instancias del motor de base de datos entonces cada instancia utilizará un directorio distinto para los archivos de sus bases de datos.

# 3. Problemas de Crecimiento en las bases de datos

Las bases de datos se van cargando de información que, aún valiosa desde una perspectiva analítica es inútil para las operaciones.

Vamos a repasar cuales son las limitaciones de las diferentes versiones gratuitas de SQL Server, vamos a discutir los errores con los que nos podemos encontrar y, finalmente, técnicas para remover de los equipos que sostienen los aplicativos que soportan la operación diaria los datos innecesarios.

Para empezar consideremos las versiones de SQL Server Express Edition y sus límites de tamaño:

| Versión | Límite |
|---|---|
| 2000 Desktop | 2 Gb |
| 2005 Express | 4 Gb |
| 2008 Express | 4 Gb |
| 2008 R2 Express | 10 Gb |
| 2012/2014 Express | 10 Gb |

Mi primer error favorito es llenar la base de datos temporal.

Cuando ejecutamos una sentencia SQL el motor de base de datos crea objetos temporarios que necesita para poder resolver la consulta. Esos objetos se guardan en una base de datos temporal.

Si nos olvidamos de, por ejemplo, poner una condición al escribir un join entre tablas grandes podemos obligar al motor a guardar el producto cartesiano de las dos tablas para luego trabajar desde allí. Si las dos tablas tienen 1000 registros, por ejemplo, el producto cartesiano tendrá 1.000.000 lo que puede resultar pesado pero manejable.

Pero, si las tablas en cuestión tienen 1.000.000 de registros (algo no tan raro) entonces el producto cartesiano tendría 1.000.000.000.000 registros. Eso no entra en casi ningún lado.

## Cuidado al escribir JOINs

Si la base de datos temporal tiene un límite señalado de crecimiento al llegar a ese punto se produce un error y la consulta cancela sin generar grandes daños.

Pero, si por inadvertencia dejamos a la base de datos temporal sin un límite de crecimiento definido entonces sigue creciendo hasta agotar el espacio disponible en el sistema de archivos.

En ese momento se produce un error mucho más complicado ya que el motor de base de datos no puede escribir en la base de datos temporal pero tampoco puede escribir en el log de transacciones.

En ese momento el motor de base de datos no tiene forma de asegurar la consistencia de los datos y pone a la base entera en estado sospechoso.

Para recuperarse del error primero hay que liberar espacio. No vamos a describir el proceso de recuperación pues excede el alcance del presente curso y debe dejarse para un DBA experimentado.

Mi segundo error favorito es hacer explotar el log de transacciones. Cuando se lanza una actualización masiva puede llevar al log de transacciones a superar el máximo de tamaño permitido.

En esa circunstancia el motor de base de datos devuelve un error y la sentencia cancela. Se puede entonces purgar el log y no hay inconvenientes en seguir trabajando.

Si el log de transacciones está marcado para crecer cuando sea necesario entonces este error puede llevarnos a una catástrofe mayor. Si el crecimiento es tal que llena el espacio disponible de nuestro disco se vuelve a producir el problema señalado en mi primer error favorito.

Por estos motivos no es recomendable dejar habilitado el crecimiento automático en sistemas productivos.

Respecto a cómo evitar el crecimiento desmedido de los sistemas transaccionales debemos tener en cuenta:

- **Ciclo de vida de cada dato:**
  Cada pieza de información es útil durante un tiempo. Luego puede moverse a otro repositorio que no forme parte de la operación cotidiana.

- **No crear índices demás:**
  Los índices aceleran las búsquedas pero, ralentizan las operaciones de actualización (update, insert y delete) por lo que siempre hay que mantener un equilibrio. No es recomendable crear índices por las dudas.

- **Partir tablas:**
  Recuerdo que una vez trabajaba para un gran Call Center con miles de operadores que hacían llamadas salientes. Cada vez que había que hacer algo sobre la tabla de llamadas era un infierno. Las transacciones no terminaban, los backups se trababan, en fin, se volvía inutilizable. Mi opción en ese momento era abandonar MySQL y pasarnos a SQL Server.

  Mi intuición estaba equivocada. Seguro que hay un SQL Server que podía manejar esa carga y seguro que hay un MySQL que puede manejar esa carga. Sin embargo, les cuento, ambas afirmaciones, aunque verdaderas, no son relevantes. La primer pregunta que nos tenemos que hacer no es si se puede manejar la carga

sino si hay realmente que cargarla...

En el caso que nos ocupa la forma de resolver el problema era separar la tabla de llamadas en varias que contuvieran sólo los llamados de un único mes.

Luego mediante vistas fuimos reconstruyendo la información, primero de las llamadas que pertenecían a un año y luego de la totalidad de las llamadas.

Una vez por año había que modificar la definición de la vista e incluir doce nuevas vistas para incluir un nuevo año. No parece tanto trabajo como para tener que sufrir algo insufriblemente lento.

- **No confundir el entorno operativo con el analítico**
  Dentro del entorno analítico se privilegian la cantidad de información los tiempos de consulta en detrimento de la agilidad de las altas, bajas y modificaciones.

  Dentro del entorno operativo se debe conseguir un balance los tiempos que demoran todas las tareas. Aquí si corresponde que parte de la información se archive.

  No es necesario que el archivo se haga en otro servidor o en otra base de datos (aunque presenta ventajas a la hora de proteger a un entorno de los requerimientos de los otros)

# 4. Páginas

La información se almacena en páginas de tamaño fijo. Un objeto , por ejemplo una tabla, puede ocupar una o varias páginas.

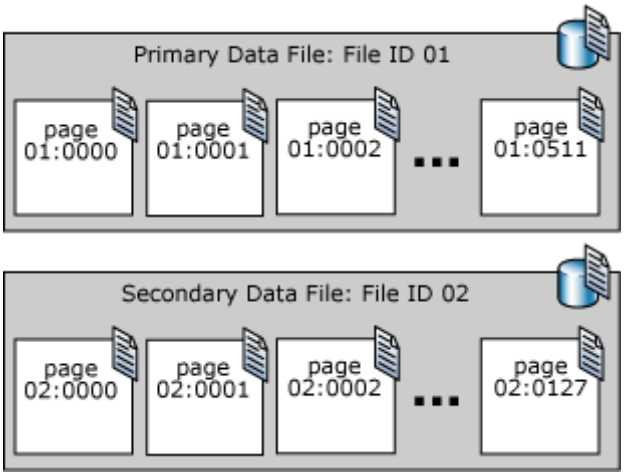
Las páginas se numeran dentro de un dispositivo (archivo) empezando por la cero.

Para fijar unívocamente una página dentro de un SQL Server es preciso señalar el nombre del dispositivo o archivo a la cual la página pertenece y su número.

El tamaño de las páginas ha ido creciendo a lo largo del tiempo. Yo recuerdo que en SQL Server 6.5 el tamaño de la página estaba en la zona de 2 Kb. Actualmente (o, por lo menos, la última versión en la que revisé este dato estaba en 8Kb.)

Una página puede tener diversos tipos de información de acuerdo al objeto del cual se trate.

Si el objeto (por ejemplo una tabla) necesita continuar en otra página (porque todos sus registros no caben en la primera) la primera página sabe cuál es el número de la siguiente página y la siguiente sabe cuál es el número de la anterior. De esa manera es rápido recorrer los objetos en cualquier sentido. (A esto se lo llama una lista doblemente encadenada)




La primer página de cada dispositivo es una página de encabezamiento que contiene información sobre los atributos del archivo.

De hecho varias páginas al principio del archivo contienen las claves para interpretarlo, por ejemplo las tablas de alocación que indican que páginas contienen la información de que objetos.

# 5. Mantenimiento de las bases de datos

Existen una serie de tareas que, si no se desarrollan en forma periódica, nos llevan al desperdicio de recursos de almacenamiento y a una degradación de la performance. Vamos a sugerir algunas de ellas:

- Backup
- Recuperación de espacio
- Regeneración de índices

**Backup**

Estrategias de backup

En principio tenemos tres grandes formas de realizar un backup:

- Completo
- Incremental
- Diferencial

El backup completo saca una foto de la base de datos en su estado actual. Es voluminoso para hacerlo cada día y, cabe esperar, que al hacer un backup completo cada día estemos reflejando muchas veces la misma información con el consiguiente desperdicio de espacio y tiempo.

El backup incremental se pensó como una solución a este problema. En vez de respaldar la foto de los datos se copia el log de transacciones. Entonces, aplicando nuevamente las transacciones podemos reproducir el estado final deseado de los datos.

De esta manera no hay redundancia. Al no haber redundancia no se desperdician tiempo y espacio. Sin embargo la redundancia es también un factor de seguridad. ¿Qué pasa si sólo uno de los backups incrementales se pierde? Resulta imposible reconstruir el estado final.

Como la confiabilidad de ningún medio físico es del 100% entonces nos encontramos frente una situación delicada.

La confiabilidad absoluta NO existe en el mundo tecnológico

El ser humano tiene porcentajes de confiabilidad operativos, en general, bajos

Algunos administradores eligen una estrategia híbrida haciendo periódicamente un backup completo y cubriendo el corto plazo con uno incremental donde la probabilidad de que uno se pierda no resulta tan inaceptable.

Otro problema del backup es el tiempo que lleva. Normalmente se trata de hacer los backups a la noche (fuera del horario operativo) de manera de no recargar la capacidad del equipo ni perder performance en la gestión del día a día.

Esto nos deja una ventana que tiende a llenarse con operaciones del tipo "Batch" (como la actualización de los Datawarehouses) que tienden también, por idénticos motivos, a ponerse en esa misma ventana.

Si alguno de los procesos se extiende entonces empuja a los demás.

Otro concepto que conviene explorar es el de recuperación de los datos. Hacer el backup es necesario pero no suficiente. Debemos asegurarnos que podemos restaurarlos. Como en todos los sistemas complejos siempre es posible que haya circunstancias no previstas por las cuales la restauración de los datos se vuelve imposible y entonces es como si al backup no lo hubiéramos hecho nunca.

Normalmente hay limitaciones de hardware, por ejemplo servidores compatibles, donde probar la restauración. Sin una prueba periódica y completa de los procesos de emergencia nunca sabremos cuando realmente estamos protegidos.

Si bien los procesos de backup se hacen desde herramientas especializadas revisemos las sentencias SQL que están en la base de todo el proceso:

*Backup Completo*

BACKUP DATABASE [nombre de la base de datos] TO [nombre del dispositivo]

Se puede utilizar también a continuación la clausula WITH para indicar opciones de backup:

COMPRESSION / NO_COMPRESSION (Para ahorro de espacio o de velocidad)

ENCRIPTION (Para indicar un algoritmo de encriptación)

NAME (Para asignarle un nombre al backup)

FORMAT (Para pisar todos los backups previos que existan en ese dispositivo)

Ejemplo:

```
BACKUP DATABASE AdventureWorks2012
TO DISK = 'Z:\SQLServerBackups\AdventureWorks2012.Bak'
   WITH FORMAT,
      MEDIANAME = 'Z_SQLServerBackups',
      NAME = 'Full Backup of AdventureWorks2012'
```

*Backup Incremental*

El backup incremental va almacenando el log de transacciones. Para restaurarlo hay que partir de un backup full y restaurarle luego todos los logs que se generaron desde ese backup full.

El código para generar un backup del log de transacciones es:

BACKUP LOG [Nombre de la base de datos] TO [Nombre del dispositivo]

Por ejemplo:

```
BACKUP LOG AdventureWorks2012
  TO MyAdvWorks_FullRM_log1;
```

Para recuperar un backup incremental necesitamos, después de recuperar el backup completo original ejecutar la recuperación de los logs y aplicarlas.

RESTORE LOG [nombre de la base de datos] FROM [nombre del dispositivo] WITH NORECOVERY.

Este paso deberemos repetirlo por cada backup del transaction log y la base de datos nos queda en el estado en el que estaba antes de inciar el último backup que hayamos restablecido.

*Backup Diferencial*

Tenemos que empezar por fijar el concepto de "extent": Llamamos extent al conjunto de 8 páginas físicamente contiguas.

Un backup diferecia captura el estado de cualquier extent que haya cambiado desde que se creó el backup completo con la opción INIT y el momento en el que se inicia el backup diferencial.

Por este motivo el tamaño de un backup diferencial es una función de la cantidad de datos que haya cambiado desde el backup full.

Para realizar un backup diferencial primero necesitamos crear un backup full:

```
BACKUP DATABASE [nombre de la base de datos]
  TO [nombre del dispositivo]
  WITH INIT;
```

Luego de transcurrido el tiempo establecido (una día, por ejemplo) creamos el backup diferencial para dar cuenta de los cambios acontecidos durante ese tiempo:

```
BACKUP DATABASE [nombre de la base de datos]
  TO [nombre del dispositivo]
  WITH DIFFERENTIAL
```

Por ejemplo, para la creación del backup full inicial:

```
BACKUP DATABASE MyAdvWorks
  TO MyAdvWorks_1
  WITH INIT
```

y, luego de transcurrido el tiempo:

```
BACKUP DATABASE MyAdvWorks
  TO MyAdvWorks_1
  WITH DIFFERENTIAL
```

Y para recuperar la información respaldada primero tenemos que recuperar el backup original:

```
RESTORE DATABASE [nombre de la base de datos]
  FROM [nombre del dispositivo]
  WITH NORECOVERY;
```

La opción NORECOVERY deja la base de datos habilitada para seguir recuperando el backup diferencial:

```
RESTORE DATABASE [nombre de la base de datos]
  FROM [nombre del dispositivo]
```

```
  WITH FILE = 2,
  RECOVERY;
```

File = 2 implica que estamos restaurando el segundo archivo de la serie de backups diferenciales. (Una forma cómoda de volver al día en particular que nos interese)

Por ejemplo:

```
RESTORE DATABASE MyAdvWorks
  FROM MyAdvWorks_1
  WITH NORECOVERY;
GO

RESTORE DATABASE MyAdvWorks
  FROM MyAdvWorks_1
  WITH FILE = 2,
  RECOVERY
```

**Recuperación de espacio**

A medida que se van agregando y borrando registros las páginas que guardan nuestra información empiezan a tener espacios disponibles.

Si páginas enteras quedan sin usar (porque se borraron los datos que contenían) el archivo que guarda la base de datos como un todo dentro del sistema operativo no se acorta.

Ese crecimiento del archivo debido a la presencia de espacios en blanco es un problema de espacio y, además, puede causar un detrimento en la performance. Las operaciones de lectura secuencial (por ejemplo contar o sumar todo un archivo) proceden más aprisa si todos los registros están almacenados en forma contigua.

Para poder realizar esta tarea de recuperación de espacio y de reorganización del mismo debemos emplear el comando DBCC.

Una descripción completa del alcance de DBCC queda fuera del alcance del curso por tratarse de una herramienta típica del arsenal del DBA.

DBCC SHRINKDATABASE [nombre de la base de datos], [ Porcentaje Objetivo],

[ , { NOTRUNCATE | TRUNCATEONLY } ]

Porcentaje objetivo es la parte de la base de datos que nos proponemos que quede libre después de haberse comprimido.

NOTRUNCATE

Al seleccionar la acción NOTRUNCATE las páginas que quedan vacías se mueven al final del archivo y no son eliminadas. Esta opción se aplica sólo a los archivos de datos (mdf) los logs (ldf) no son afectados.

TRUNCATE

Si usamos la opción TRUNCATE las páginas vacías son eliminadas para cumplir, en la medida de lo posible, con el objetivo de reducción planteado.

TRUNCATEONLY

Esta opción es más rápida ya que libera al sistema operativo todo el espacio libre existente al fin del archivo pero sin reacomodar las páginas para liberar más datos. Esta opción sí afecta al log de transacciones.

Si estamos tratando de afectar específicamente al archivo de datos sin impactar en el log de transacciones (algo muy importante si estamos usando un esquema de backup incremental) entonces recurrimos a DBCC SHRINKFILE

DBCC SHRINKFILE

(

{ nombre del dispositivo ó número del dispositivo }

{ [ , EMPTYFILE ]

| [ [ , target_size ] [ , { NOTRUNCATE | TRUNCATEONLY } ] ]

}

)

[ WITH NO_INFOMSGS ]

**Regeneración de los índices**

Un índice tiende a fragmentarse cada vez que se van usando las sentencias de INSERT, UPDATE y DELETE a los datos que representa. Esta fragmentación causa que las operaciones se vuelvan más lentas degradando la velocidad de respuesta de todas las aplicaciones que dependen de estos índices.

Se puede corregir la fragmentación de un índice recurriendo a su reorganización o, simplemente, reconstruyéndolo.

Como pueden suponer reconstuir un índice pasa por borrarlo con DROP y luego reconstruirlo con CREATE INDEX. Por supuesto que esto remueve cualquier fragmentación que exista, deja a todas las páginas con el factor de llenado de diseño y, más importante aún, pone a todas las hojas que forman el índice en forma contigua lo que acelera y facilita su recorrido. La cantidad de recursos del sistema (disco, memoria y procesamiento) involucrados en esta tarea es importante.

Para evitar ese último inconveniente existe la opción de reorganizar un índice. Esto trabaja a nivel de las hojas del índice tanto si este es clusterizado como si no. Las páginas quedan ordenadas y de acuerdo al valor del factor de llenado pre-existente. Las divisiones del arbol que no son hojas quedan intactas y, por lo tanto, pueden quedar desbalanceadas con lo que el tiempo de búsqueda tampoco será el mejor posible.

Antes de lanzarnos a la tarea de defragmentar un índice necesitamos evaluar la necesidad de hacerlo y seleccionar el método más adecuado para nuestra circunstancia.

La función construida dentro del motor de base de datos para medir la fragmentación de un índice es sys.dm_db_index_physical_stats.

Esta función se puede aplicar a:

- Un índice en particular
- Todos los índices de una tabla
- Todos los índices de una base de datos
- Todos los índices en todas las bases de datos de un mismo motor

La función sys.dm_db_index_physical_stats devuelve una tabla con

- Porcentaje de fragmentación promedio
- Cantidad de fragmentos
- Tamaño promedio del fragmento

La función lleva los siguientes parámetros:

- Id de la base de datos (Poner NULL para la base actual)
- Id del objeto (Poner NULL para todas las tablas)
- Número del índice (Poner NULL para todos los índices)
- Número de la partición (para índices con particiones, fuera del alcance del curso, NULL para todas las particiones)
- Modo (Se refiere al tipo de índice, NULL para todos)

Para interpretar el resultado deberemos mirar el porcentaje de fragmentación promedio.

- Si se encuentra por debajo del 5% la sugerencia es no hacer nada.
- Si está entre el 5% y el 30% conviene reorganizar
- Por encima del 30% es habitualmente más práctico reconstruir el índice

La reconstrucción de un índice puede hacerse tanto On Line como Off Line. La reorganización se hace siempre On Line.

La sintaxis es:

Para reorganizar:

ALTER INDEX [nombre del índice] ON [nombre de la tabla] REORGANIZE

y, como pueden esperar, para reconstruir es:

```
ALTER INDEX [nombre del índice] ON [nombre de la tabla] REBUILD
```

Una buena noticia es que cuando un índice es creado o reconstruido (no cuando es reorganizado) se actualizan las estadísticas que guarda SQL Server sobre ese índice. Sin embargo, a partir de SQL 2012 esto no implica que se revisen todos los registros de la tabla para compilar las estadísticas sino que se recurre a una técnica de muestreo a menos que se especifique la cláusula FULLSCAN.

Ejemplo:

```
USE AdventureWorks2012;
GO
-- Busca el porcentaje de fragmentación de todos los índices  en una tabla

SELECT a.index_id, name, avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats (DB_ID(N'AdventureWorks2012'),
     OBJECT_ID(N'HumanResources.Employee'), NULL, NULL, NULL) AS a
   JOIN sys.indexes AS b
     ON a.object_id = b.object_id AND a.index_id = b.index_id;
GO
```

El resultado que obtendremos será parecido a:

| Index_Id | Name | Avg_Fragmentation_in_percent |
|---|---|---|
| 1 | PK_Employee_BusinessEntityID | 0 |
| 2 | IX_Employee_OrganizationalNode | 0 |
| 5 | AK_Employee_LoginID | 66.67 |

**Creación y actualización de estadísticas**

Una cosa que no nos hemos cuestionado es como hace el motor de base de datos para decidir qué estrategia, y por ende, qué combinación de índices usar para resolver una consulta determinada.

El motor de bases de datos trata de buscar el camino óptimo. Para eso necesita calcular cuánto le costará ir por cada camino posible. Para hacerlo recurre a estadísticas sobre cuantas entradas tiene por cada clave en cada uno de los índices que podría valer la pena usar.

Esta información se crea cuando se genera el índice tomando una muestra de la tabla existente al momento de su creación. Dependiendo de cómo se vayan luego cargando los datos esas estadísticas pueden quedar desactualizadas.

Para la mayoría de las consultas las estadísticas que automáticamente se almacenan son suficientes. En algunas ocasiones puede hacer falta crear estadísticas adicionales para eso contamos con la sentencia CREATE STATISTICS.

La sintaxis es:

CREATE STATISTICS [nombre de la estadística]

ON [nombre de la tabla o del índice] (lista de columnas sobre las que crear estadísticas)

WITH FULLSCAN

La opción WITH FULLSCAN fuerza a la revisión de todo el contenido de la tabla para generar las estadísticas en vez de conformarse con una muestra de los registros. (Esta última sería la opción por defecto si no se toma esta opción)

Actualización de las estadísticas

Una vez creadas las estadísticas ya sean las que se generan por defecto como las especificadas por un usuario estas no se actualizan por sí mismas.

Estadísticas desactualizadas que ya no reflejen la composición de los datos pueden llevar al motor de base de datos a tomar una decisión sub-optima a la hora de seleccionar la estrategia a seguir para resolver una dada consulta.

La forma en la cual nos protegemos de esto es mediante la periódica actualización de las estadísticas. Para conseguirlo recurrimos a la sentencia UPDATE STATISTICS

La sintaxis correspondiente es:

UPDATE STATISTICS [nombre de la tabla]

Podemos también agregar los nombres del índice o de las estadísticas concretas que querramos actualizar.

Tenemos también disponible la opción WITH FULLSCAN para evitar el muestreo.

Si queremos consultar cuando se hizo la última actualización de las estadísticas recurrimos a:

EXEC sp_updatestats

# Bibliografía utilizada y sugerida

Date, C. J. An Introduction to Database Systems, Pearson Educación, Mexico, 2001

Medina, Santiago, SQL Server 2012 Guía práctica de administración, INFORBOOKS'S S.L. España, Barcelona 2014

Nield, Thomas, Geting Started with SQL, O'Really, USA Sebastopol CA, 2016

## Lo que vimos:

En esta unidad vimos los principales conceptos relacionados con la arquitectura interna de una base de datos, así como las tareas que tienen que ver con las copias de seguridad (backup) y mantenimiento.

## Lo que viene:

Dentro del próximo módulo vamos a ver cómo aplicar los conceptos que hemos desarrollado a cuatro de las bases de datos relacionales más difundidas en el mercado:

- MySQL
- SQL Server
- Oracle
- DB2

En particular, pondremos foco en cómo instalarlas, usar sus herramientas administrativas desde la interface gráfica y realizar la creación y destrucción de objetos.