

Programación básica de servidor de base de datos (3): rendimiento de consultas

En esta sesión se introducen las herramientas clásicas para codificar los procedimientos necesarios para un servicio de paginación de resultados.

Objetivos

1. Saber hacer copias de seguridad y recuperarlas.
2. Conocer las opciones de indización del servidor de base de datos.
3. Saber decidir cuando aplicar índices por su coste y beneficio.
4. Conocer el procesamiento de tablas en memoria.
5. Saber analizar el plan de ejecución de una consulta.
6. Ser capaz de cambiar manualmente el plan de ejecución de una consulta.
7. Saber cómo rediseñar consultas para mejorar su rendimiento.

Copias de seguridad



En este archivo se encuentra la [copia de seguridad de la base de datos P103](#)¹ con datos suficientes como para desarrollar la presente sesión. Descárgala y restáurala en su servidor. Es una copia hecha desde SQL Server Express 2016 —puede no funcionar en versiones anteriores—.

El script de generación de la misma lo puedes consultar [en este otro enlace](#). No es necesariamente la forma más eficiente de rellenar la BD —tampoco es habitual tener que hacerlo— pero hay ejemplos diversos de cómo resolver los distintos problemas que plantea.

¹ Esto es información adicional que no te afecta realmente. Una base de datos se almacena en disco duro como un conjunto de ficheros de distinto uso. Uno de ellos es la bitácora (*log*). La información de este puede ocupar mucho espacio que, para los objetivos de esta sesión, no necesitamos. La forma de vaciar ese fichero es:

```
ALTER DATABASE P103
```

```
SET RECOVERY SIMPLE;
```

```
GO
```

```
--Reducimos el log de transacciones a 1 MB.
```

```
DBCC SHRINKFILE(P103_Log, 1);
```

```
GO
```

```
--Cambiamos nuevamente el modelo de recuperación a Completo.
```

```
ALTER DATABASE P103
```

```
SET RECOVERY FULL;
```

```
GO
```

En cuanto a la copia de seguridad y restauración, este es uno de los métodos clásicos para

- recuperación ante desastres
- migración de datos

Se pueden realizar y programar distintos tipos de copia:

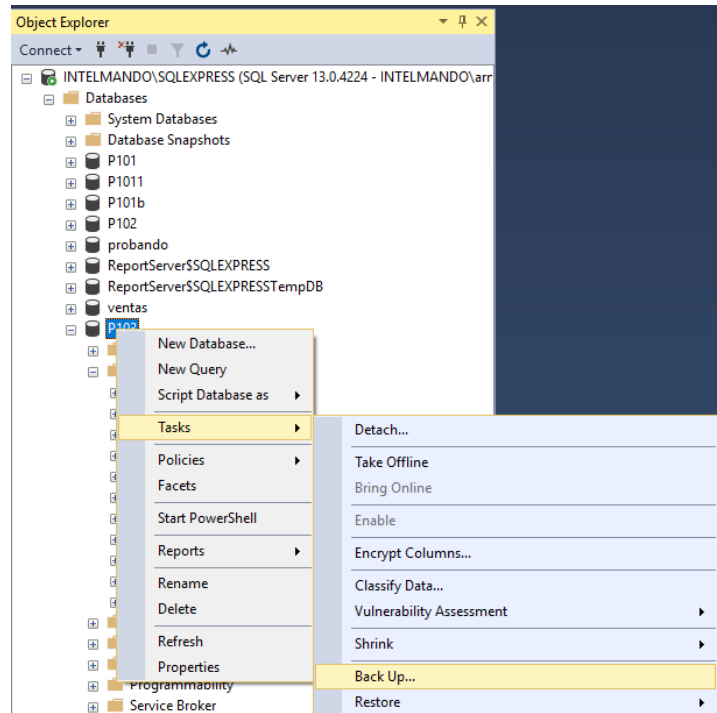
1. De BD completa
2. Parciales
3. De archivo (de BD)
 - a. Completa
 - b. Diferencial
 - c. Incremental

En nuestro caso es bastante simple, solo hay que iniciar el asistente desde SSMS y elegir el origen de datos.

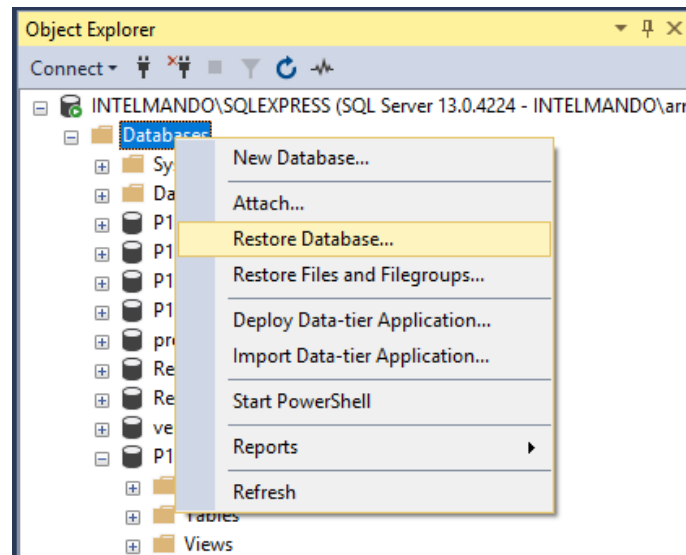


[Realizar copias de seguridad y restaurar bases de datos de SQL Server](#)

Backup



Restore



Restore Database - P103

A tail-log backup of the source database will be taken. View this setting on the Options page.

Select a page

- General
- Files
- Options

Connection

INTELMANDO\SQLEXPRESS
[INTELMANDO\amando]

[View connection properties](#)

Progress

Done

Script Help

Source

Database:

Device: J:\P103.bak

Database: P103

Destination

Database: P103

Restore to: The last backup taken (jueves, 1 de noviembre de 2018 10:46:44) [Timeline...](#)

Restore plan

Backup sets to restore:

Restore	Name	Component	Type	Server	Database	Posit
<input checked="" type="checkbox"/>	P103-Full Database Backup	Database	Full (Copy Only)	INTELMANDO\SQLEXPRESS	P103	1

[Verify Backup Media](#)

OK Cancel Help

Es posible que tengas que, además, redefinir la ruta de los archivos a restaurar de la siguiente forma:

Restore Database - P103

Ready

Select a page

- General
- Files
- Options

Connection

INTELMANDO\SQLEXPRESS
[INTELMANDO\amando]

[View connection properties](#)

Progress

Done

Script Help

Restore database files as

☒ Relocate all files to folder

Data file folder: M:\Program Files\Microsoft SQL Server\MSSQL15.SQLEXPRESS\MSSQL\DATA

Log file folder: M:\Program Files\Microsoft SQL Server\MSSQL15.SQLEXPRESS\MSSQL\DATA

Logical File Name	File Type	Original File Name	Restore As
P103	Rows Data	M:\Program Files\Microsoft SQL ...	M:\Program Files\Microsoft SQL Server\MSSQL15
P103_log	Log	M:\Program Files\Microsoft SQL ...	M:\Program Files\Microsoft SQL Server\MSSQL15
P103_mod	FILESTREAM ...	M:\Program Files\Microsoft SQL ...	M:\Program Files\Microsoft SQL Server\MSSQL15

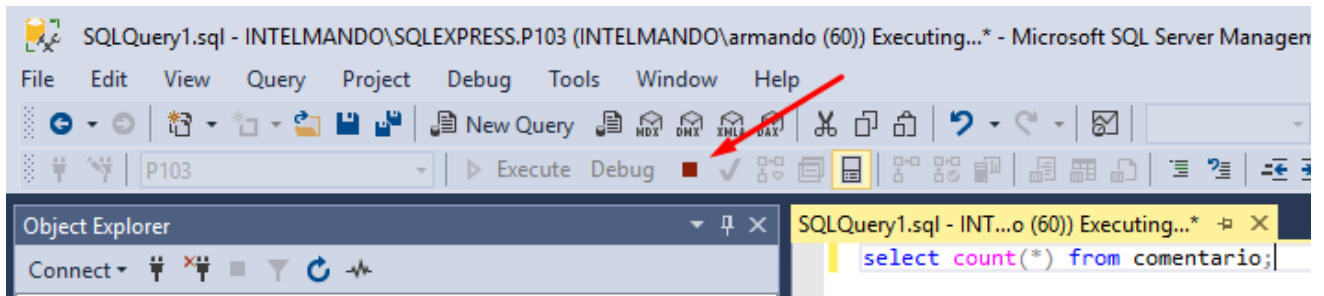
OK Cancel Help

Estadísticas

El servidor de base de datos va recopilando datos estadísticos durante su funcionamiento. Si quisiéramos conocer la cantidad de filas de una tabla nuestra primera intuición sería ejecutar

```
select count(*) from comentario;
```

Cancela la consulta cuando te canses. No obstante, el resultado sería 4.038.070 filas



Ahora prueba:

```
SELECT OBJECT_NAME(OBJECT_ID) TableName, st.row_count  
FROM sys.dm_db_partition_stats st  
WHERE OBJECT_NAME(OBJECT_ID) = N'comentario'
```

Hemos hecho uso de las estadísticas guardadas en el catálogo del sistema. Esta información le sirve a SQL Server para la optimización previa y automática que realiza ante cualquier consulta.



E00

Obtén todas las cuentas de filas a partir de la información estadística del catálogo —no vale usar lo anterior y explicitar los nombres de tabla ;) —. ¿Probando a buscar en Internet "SQL SERVER count(*) alternativa"?

Results			Messages	
	NAME	rowcnt		
1	comentario	4038070		
2	comkeyw	1246401		
3	conexion	0		
4	palabrasclave	294		
5	perfil	3		
6	sigue	500000		
7	usuario	1402411		
8	valora	150000		

Hay muchas formas de hacer lo anterior. Sirva como pista que de estos archivos del catálogo del sistema podemos obtener la información suficiente.

- `object_id` es el identificador de objeto de base de datos
- de `sys.tables`: tablas de usuario (`name`)
- de `sys.indexes`: el identificador de índice en el objeto (`index_id`)
- de `sys.dm_db_partition_stats`: la cuenta de filas (`row_count`)

```
use P103
```

```
go
```

```
select i.object_id,i.name,i.index_id,i.is_primary_key,o.object_id,o.name  
from sys.indexes i join sys.tables o on i.object_id=o.object_id  
WHERE i.index_id < 2 order by o.name
```

object_id	name	index_id	is_primary_key	object_id	name
773577794	PK_comentario	1	1	773577794	comentario
869578136	PK_comkeyw	1	1	869578136	comkeyw
661577395	PK_conexion	1	1	661577395	conexion
837578022	PK_palabrasclave	1	1	837578022	palabrasclave
565577053	PK_perfil_3BB8AEC342572116	1	1	565577053	perfil
709577566	PK_sigue	1	1	709577566	sigue
597577167	PK_usuario	1	1	597577167	usuario
933578364	PK_valora	1	1	933578364	valora

La columna `index_id`, aparte de identificar al índice dentro del objeto, tiene la particularidad de que en su valor 0 indica un índice de tipo montón (*heap*), y en el 1 de tipo agrupado (*cluster*). Se da la circunstancia de una tabla solo admite un índice *heap* o uno *cluster*, y no ambos. Sea uno u otro, el índice guarda un identificador único por cada fila de la tabla.

Por otra parte, de los índices se guarda la información estadística de la cantidad de valores almacenados en el índice. En realidad, si hablamos de claves primarias (índice agrupado) o tablas sin ella (índice de montón), ese valor coincide con la cuenta de filas de la tabla.

```
select i.object_id,i.name,  
       ddps.row_count  
from sys.indexes i join sys.dm_db_partition_stats ddps  
  on i.OBJECT_ID = ddps.OBJECT_ID AND i.index_id = ddps.index_id  
WHERE i.index_id < 2
```

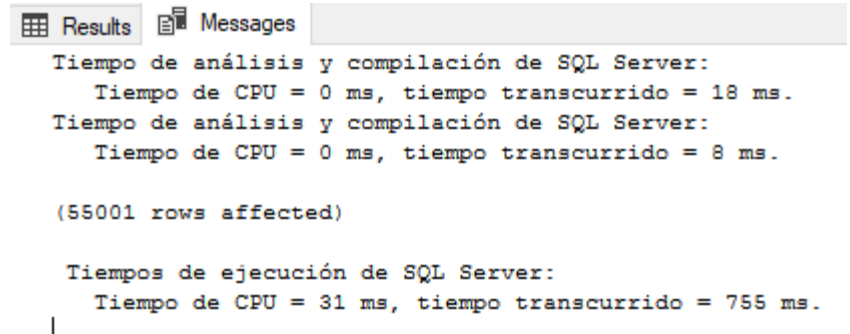
Con esta consulta obtenemos todos los índices, propios y del sistema. Ya solo queda combinar las dos consultas anteriores, con lo que restringiríamos la búsqueda a las tablas que nos interesan.

No es, ni mucho menos, la única aplicación de las estadísticas, de hecho forman parte fundamental de los recursos utilizados en el planificador de consultas. Otro ejemplo es la obtención de tiempos. Para ello vamos a utilizar la orden [SET STATISTICS TIME](#).

Ejecuta lo siguiente 2 veces anotando tiempos:

```
SET STATISTICS TIME ON;  
GO  
select * from usuario where usuId between 65000 and 120000  
SET STATISTICS TIME OFF;  
GO
```

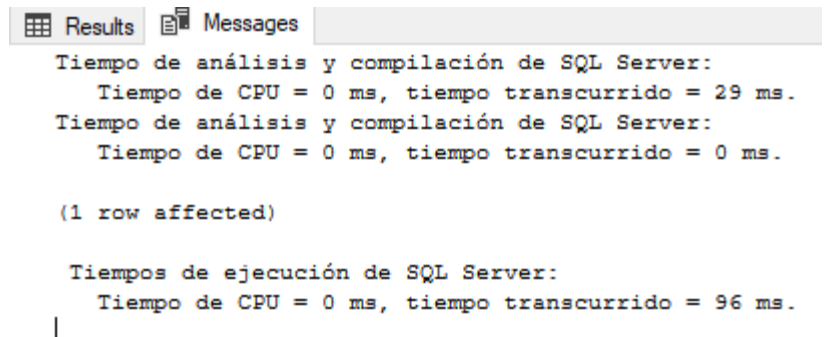
De momento, nada más. Verás una recopilación de datos en la pestaña de mensajes parecida a esta. Obviamente, tus valores serán distintos.



```
Results Messages  
Tiempo de análisis y compilación de SQL Server:  
Tiempo de CPU = 0 ms, tiempo transcurrido = 18 ms.  
Tiempo de análisis y compilación de SQL Server:  
Tiempo de CPU = 0 ms, tiempo transcurrido = 8 ms.  
  
(55001 rows affected)  
  
Tiempos de ejecución de SQL Server:  
Tiempo de CPU = 31 ms, tiempo transcurrido = 755 ms.
```

Sin embargo, fíjate en esta otra ejecución:

```
SET STATISTICS TIME ON;  
GO  
select count(*) from usuario where usuId between 65000 and 120000  
SET STATISTICS TIME OFF;  
GO
```



```
Results Messages  
Tiempo de análisis y compilación de SQL Server:  
Tiempo de CPU = 0 ms, tiempo transcurrido = 29 ms.  
Tiempo de análisis y compilación de SQL Server:  
Tiempo de CPU = 0 ms, tiempo transcurrido = 0 ms.  
  
(1 row affected)  
  
Tiempos de ejecución de SQL Server:  
Tiempo de CPU = 0 ms, tiempo transcurrido = 96 ms.
```

Es evidente, y para los siguientes ejercicios lo tendremos en cuenta a menudo, que la presentación de resultados —mostrar las filas— consume mucho tiempo.

Otra de las optimizaciones automáticas del sistema es el uso de memoria caché. Ejecuta

```
DBCC FREEPROCCACHE WITH NO_INFOMSGS  
DBCC DROPCLEANBUFFERS WITH NO_INFOMSGS;
```

```
SET STATISTICS TIME ON;  
GO  
select count(*) from usuario where perfil=0  
SET STATISTICS TIME OFF;  
GO
```

```
Results Messages
Tiempo de análisis y compilación de SQL Server:
    Tiempo de CPU = 0 ms, tiempo transcurrido = 1 ms.
Tiempo de análisis y compilación de SQL Server:
    Tiempo de CPU = 0 ms, tiempo transcurrido = 0 ms.

(1 row affected)

Tiempos de ejecución de SQL Server:
    Tiempo de CPU = 141 ms, tiempo transcurrido = 2361 ms.

Completion time: 2021-10-14T19:58:02.3889897+02:00
```

Y otra vez, pero solo

```
SET STATISTICS TIME ON;
GO
select count(*) from usuario where perfil=0
SET STATISTICS TIME OFF;
GO
```

```
Results Messages
Tiempo de análisis y compilación de SQL Server:
    Tiempo de CPU = 0 ms, tiempo transcurrido = 0 ms.
Tiempo de análisis y compilación de SQL Server:
    Tiempo de CPU = 0 ms, tiempo transcurrido = 0 ms.

(1 row affected)

Tiempos de ejecución de SQL Server:
    Tiempo de CPU = 156 ms, tiempo transcurrido = 162 ms.

Completion time: 2021-10-14T19:58:48.9179871+02:00
```

La segunda ejecución ha sido más rápida por efecto de la caché. Si queremos comparar tiempos debemos purgar esa caché, cosa que se consigue que esas dos líneas de código ejecutadas en primer lugar:

```
DBCC FREEPROCCACHE WITH NO_INFOMSGS
DBCC DROPCLEANBUFFERS WITH NO_INFOMSGS;
```

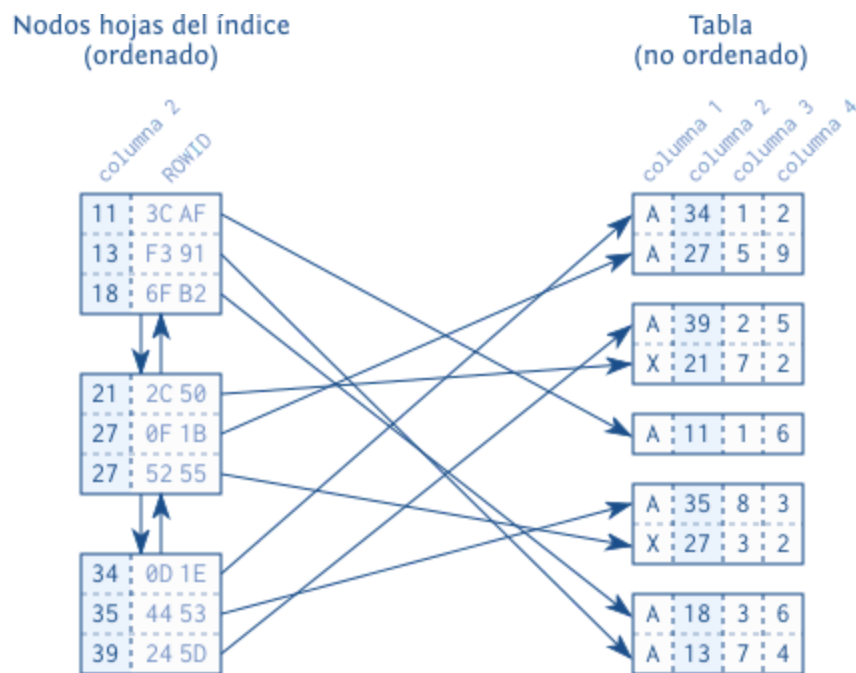
Nuestra intención es comparar el rendimiento de la definición de distintos tipos de índices. Por eso necesitaremos esa purga.

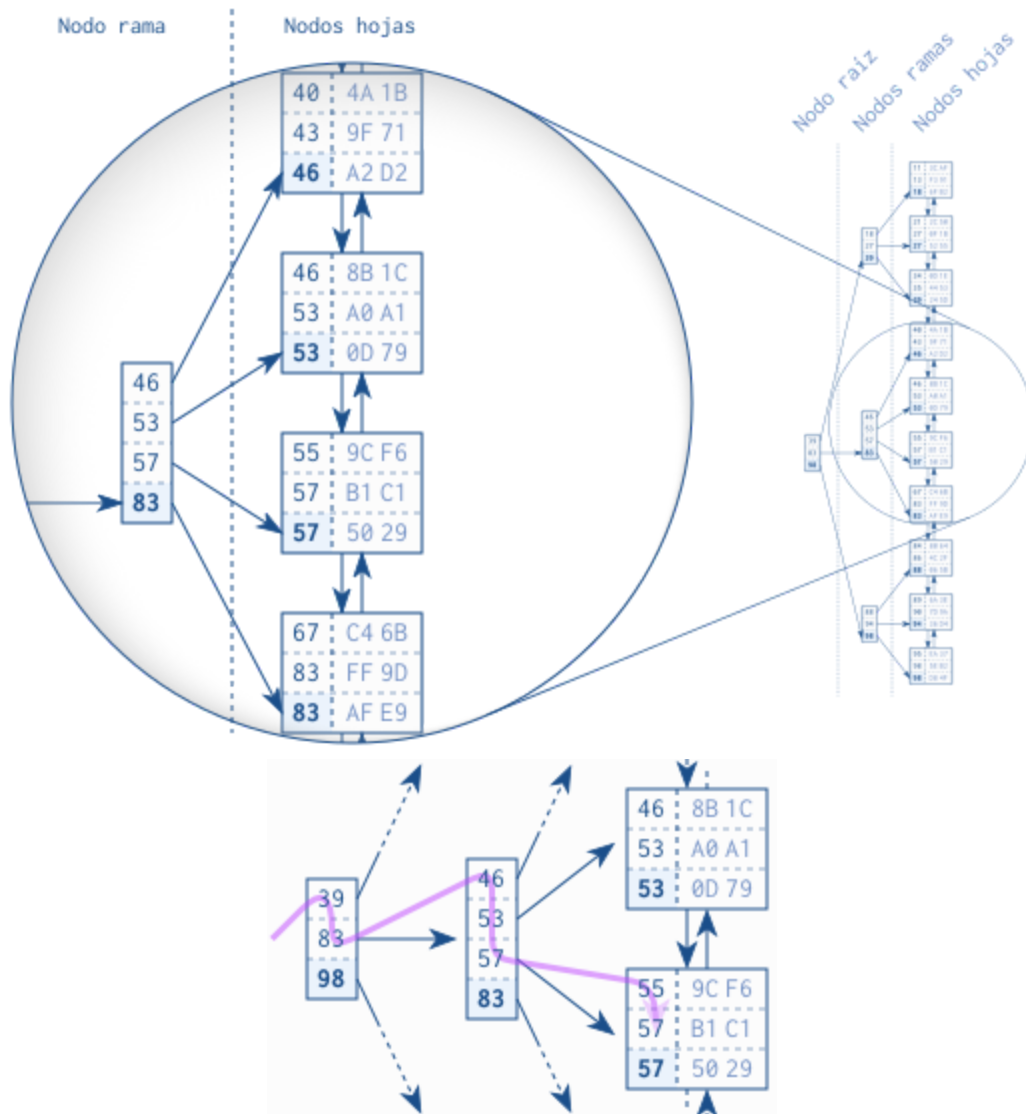
Índices

Los índices son estructuras auxiliares que aceleran el acceso y recuperación de filas. No obstante, llevan un coste asociado en su mantenimiento.

Los más habituales son los de organización hash y árboles-B, pero cada producto los implementa a su manera y ofrece herramientas propias para gestionarlos. Esto se refleja en las opciones que la orden CREATE INDEX acepta en cada uno.

Sin entrar en detalles de organización, es una especie de tabla auxiliar que solo mantiene claves y posiciones de fila. Se supone que es más rápido y eficiente buscar un dato clave en el índice —que puede incluso mantenerse en una caché en memoria RAM— y, obtenida la posición de la fila dentro de la tabla, ir directamente a esa fila y recuperarla.





<http://use-the-index-luke.com/es/sql/i%CC%81ndice-anatomi%CC%81a>

Índices generados automáticamente al crear tablas: claves primarias

Para otras columnas hay que definirlos explícitamente, bien dentro de la propia estructura de la tabla:

```
USE basedatos;
CREATE TABLE tabla
(
    tablaID int NOT NULL,
    CONSTRAINT AK_tablaID UNIQUE(tablaID)
);
```

Bien en una orden separada:

```
CREATE INDEX IX_VendorID ON ProductVendor (VendorID);
```

Para eliminar un índice:

DROP INDEX [IF EXISTS] *nomidx* ON *tabla*

Montones (*heap*)

Antes de entrar en materia, una consideración. Una tabla puede crearse sin índices de ningún tipo. Aquellas tablas que tienen un recorrido secuencial y completo, o de tamaño pequeño, son las perfectas candidatas.

Vamos a copiar la tabla usuario desordenada y sin índices²:

```
USE P103
go
--creamos tabla con los datos
drop table if exists usuariosH
CREATE TABLE usuariosH(
    usuId int NOT NULL,
    password varbinary(64) NULL,
    apodo nvarchar(15) NULL,
    email nvarchar(254) NOT NULL,
    nombre nvarchar(50) NOT NULL,
    apellidos nvarchar(75) NOT NULL,
    nacido date NOT NULL,
    perfil nchar(3) NULL
)
insert into usuariosH select top(1402411) * from usuario order by newid();
```

² Para versiones de SQL Server anteriores a 2016, "if exists" no formaba parte de Transact-SQL. Sigue siendo válido, no obstante:

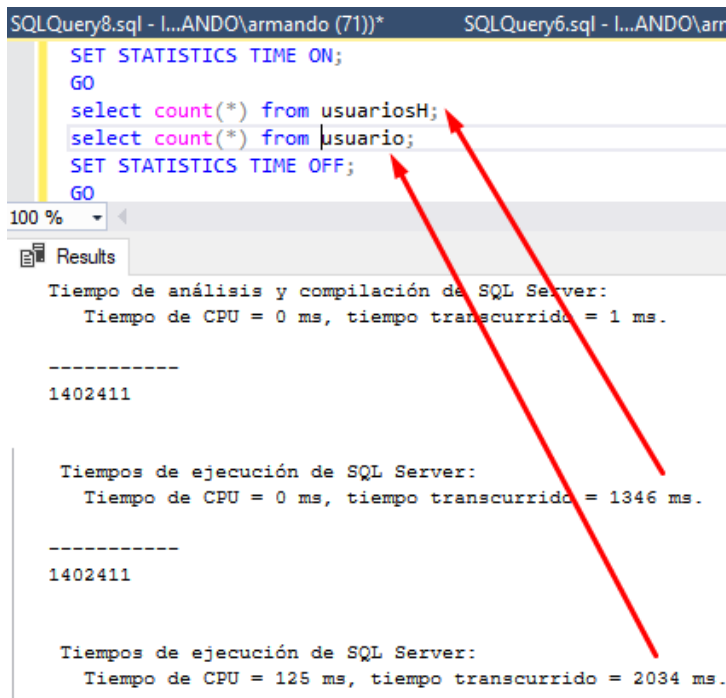
```
IF OBJECT_ID('usuariosH', 'U') IS NOT NULL
    DROP TABLE usuariosH;
go
```



E01

Ahora, con la **"salida de resultados a texto"**, ejecuta lo siguiente anotando tiempos:

```
DBCC FREEPROCCACHE WITH NO_INFOMSGS
DBCC DROPCLEANBUFFERS WITH NO_INFOMSGS; -- necesitamos limpiar caché para comparar
SET STATISTICS TIME ON;
GO
select count(*) from usuariosH;
select count(*) from usuario;
SET STATISTICS TIME OFF;
```



The screenshot shows a SQL query window with the following text:

```
SQLQuery8.sql - I...ANDO\armando (71))* SQLQuery6.sql - I...ANDO\arn
SET STATISTICS TIME ON;
GO
select count(*) from usuariosH;
select count(*) from usuario;
SET STATISTICS TIME OFF;
GO
```

Below the query window, the 'Results' pane shows the following output:

```
100 %
Results
Tiempo de análisis y compilación de SQL Server:
Tiempo de CPU = 0 ms, tiempo transcurrido = 1 ms.

-----
1402411

Tiempos de ejecución de SQL Server:
Tiempo de CPU = 0 ms, tiempo transcurrido = 1346 ms.

-----
1402411

Tiempos de ejecución de SQL Server:
Tiempo de CPU = 125 ms, tiempo transcurrido = 2034 ms.
```

Red arrows point from the SQL query text to the corresponding execution results: one arrow points from 'select count(*) from usuariosH;' to the first execution block, and another arrow points from 'select count(*) from usuario;' to the second execution block.

El primer dato de tiempos es el de compilación de las órdenes. Los que nos interesan son los dos siguientes —los que van a continuación de la cuenta de filas (1402411)—. La primera consulta ha tardado 1346 ms, y la segunda 2034 ms —en este ejemplo de ejecución—.

Nota: nos interesa "tiempo transcurrido" como el tiempo total de ejecución que debería incluir el tiempo de CPU. Si ves que el tiempo de CPU es mayor que el transcurrido, lo que parece incoherente, es porque se han usado varios procesadores en paralelo.

¿Alguna diferencia entre las dos ejecuciones? ¿Y entre las dos consultas? ¿Por qué?

**E02**

Ahora ejecuta este conjunto de consultas, y compara los tiempos de la primera (usuariosH) y la segunda (usuario).

```
DBCC FREEPROCCACHE WITH NO_INFOMSGS
DBCC DROPCLEANBUFFERS WITH NO_INFOMSGS;
GO
SET STATISTICS TIME ON;
GO
select * from usuariosH where usuId = 65245;
select * from usuario where usuId = 65245;
SET STATISTICS TIME OFF;
```

¿Alguna diferencia? ¿Por qué?

**E03**

Igualmente, pero fíjate en el tiempo de ejecución de la esquina inferior derecha de SSMS

```
set nocount on
DBCC FREEPROCCACHE WITH NO_INFOMSGS
DBCC DROPCLEANBUFFERS WITH NO_INFOMSGS;
GO
declare @contador integer=1402412
while (@contador<1500000)
begin
insert into usuariosH (usuID,nombre,password,apodo,email,apellidos,nacido,perfil)
values
(@contador,'pepito',convert(varbinary,'oooo'),'pep','nada@nada.com','López','2018-11-2',0)
set @contador=@contador+1
end
```

Y ahora

```
declare @contador integer=1402412
while (@contador<1500000)
begin
insert into usuario (nombre,password,apodo,email,apellidos,nacido,perfil)
values ('pepito',convert(varbinary,'oooo'),'pep','nada@nada.com','López','2018-11-2',0)
set @contador=@contador+1
end
```

¿Alguna diferencia? ¿Por qué?

**E03b**

Limpia las tablas de esas ejecuciones:

```
SET STATISTICS TIME ON;
delete from usuariosH where usuID>=1402412
```



```
delete from usuario where usuID>=1402412  
SET STATISTICS TIME OFF;
```

¿Alguna diferencia? ¿Por qué?

Tipos de índices

- Agrupado (*clustered*)
- No agrupado (*nonclustered*)
- Único (*unique*)
- Filtrado
- columnstore
- Hash
- Índice no agrupado optimizado para memoria (*memory-optimized nonclustered*)

La distinción más importante la haremos entre

- clustered (**agrupado**)
Se trata de un índice ordenado. No está exento de [limitaciones o recomendaciones](#), la más importante que no se puede definir más de un índice agrupado por tabla. La restricción primary key genera un índice agrupado automáticamente, aunque se puede modificar este comportamiento.
- nonclustered (**no agrupado**)
Son los que se usan típicamente para acelerar consultas y tienen mucho que ver con los predicados habituales —en consultas que se ejecutan muchas veces— en el *where*.

Además, los índices pueden ser o no **UNIQUE** (de valor único). El caso más notorio es la clave primaria, aunque no se especifica explícitamente, el sistema la asigna automáticamente.

En el caso de los índices agrupados, aun cuando no se especifique la condición UNIQUE, añade una columna, interna y transparente para el usuario, de unicidad de 4 bytes, que utiliza cuando sea necesario.

Un **índice filtrado**, según la [Guía de diseño de índices de SQL Server](#), "es un índice no clúster optimizado, especialmente indicado para atender consultas que realizan selecciones a partir un subconjunto bien definido de datos".

```
CREATE NONCLUSTERED INDEX nomidx
  ON tabla (columna(s))
  WHERE filtro;
GO
```

Si quieres consultar los índices de una determinada tabla de la base de datos seleccionada:

```
select i.object_id,i.name,i.index_id,i.type,i.type_desc,i.is_primary_key,
       o.object_id,o.name
from sys.indexes i join sys.tables o on i.object_id=o.object_id
order by o.name
```

Ten en cuenta que las columnas

i.index_id	i.type	i.type_desc
0 = Montón	0 = Montón	HEAP
1 = Índice clúster	1 = Clúster	CLUSTERED
> 1 = Índice no clúster	2 = No clúster	NONCLUSTERED
	3 = XML	XML
	4 = Espacial	SPATIAL
	5 = índice clúster de almacén de columnas.	Almacén de columnas agrupado
	6 = índice no agrupado de almacén de columnas.	Almacén de columnas no agrupado
	7 = índice hash no clúster.	HASH no clúster

Más información en [sys.indexes](#).



E04

Considera la siguiente consulta. La tabla tiene más de 4 millones de filas, y la cantidad de ellas que tienen valor en la columna "respondea" no llega al 1%. Crea un índice filtrado.

USE P103

go

```
select count(*) from comentario where respondea is not null
```



E04b

Define otro índice filtrado para

```
select usuld, respondea from comentario where respondea is not null
```



- Índices
- Guía de diseño de índices de SQL Server
- Índices agrupados y no agrupados descritos



Departamento
de Lenguajes
y Sistemas
Informáticos

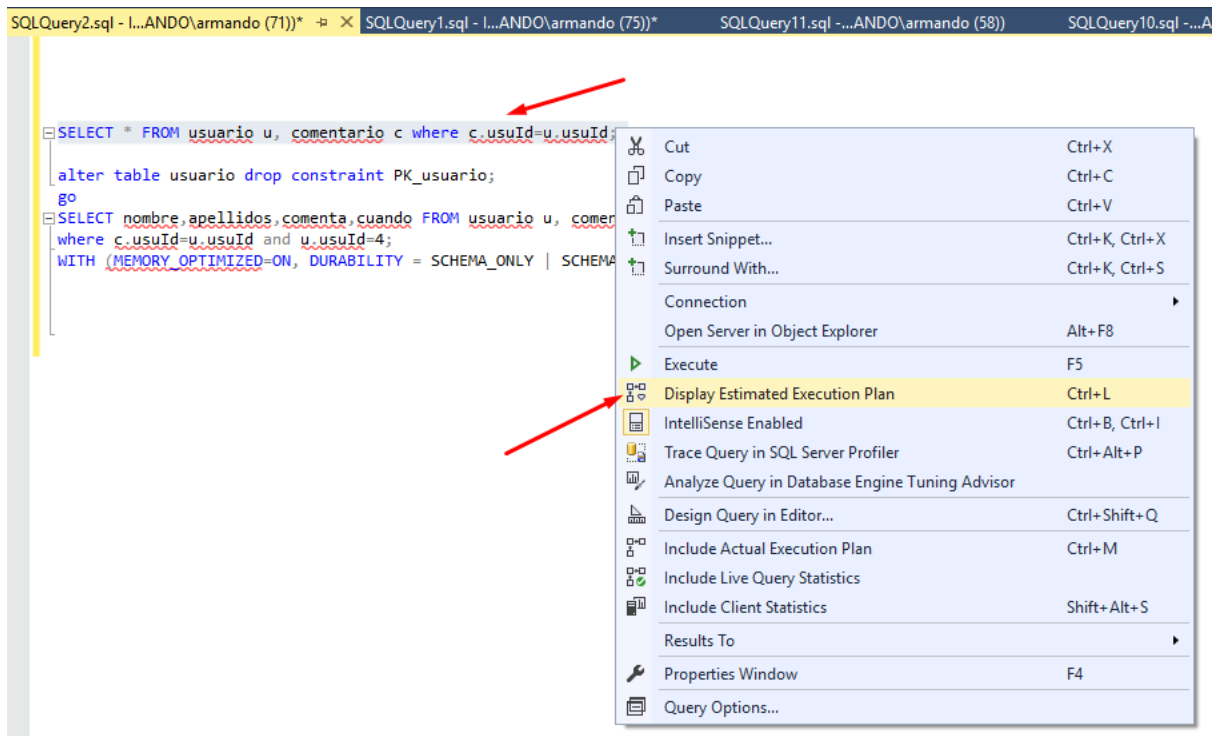


Universitat d'Alacant
Universidad de Alicante

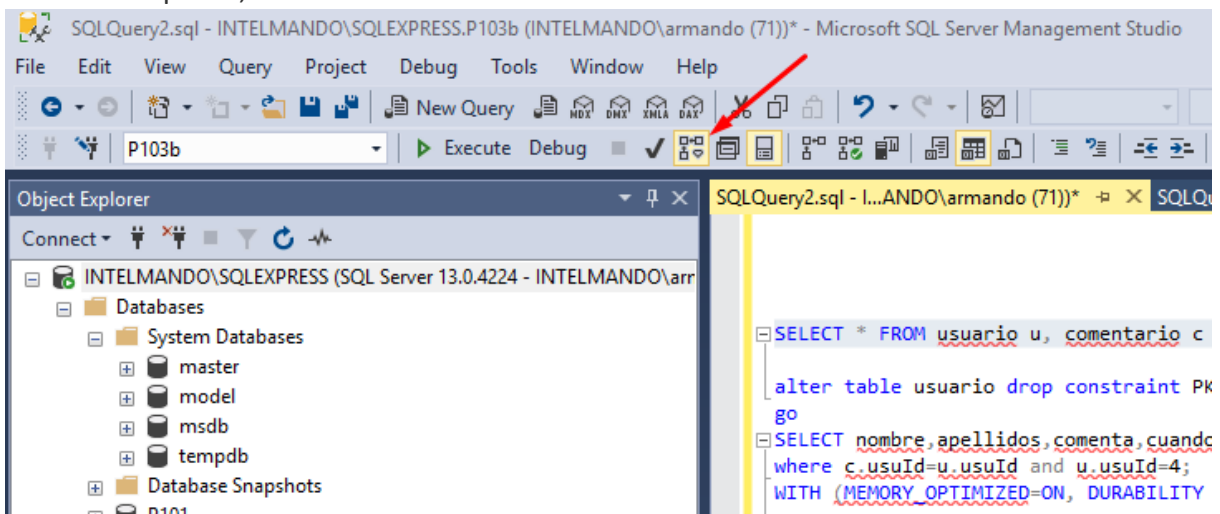
Planes de ejecución

El optimizador de consultas realiza un análisis previo buscando el mejor procedimiento para ejecutar una consulta concreta. El mejor procedimiento es aquel que balancea tiempo de respuesta y recursos —procesador, memoria, etc.—. Es totalmente transparente para el usuario pero este puede forzar la elección del plan a usar en determinadas consultas y en momentos concretos. Así mismo, el servidor mantiene una caché en memoria RAM para reutilizar planes ya definidos en una ejecución anterior, lo que resta ese tiempo de análisis y disminuye el tiempo de ejecución total.

Para ver el plan de estimado se puede utilizar el menú de contexto de SSMS sobre la selección de una consulta,



pulsando en la botonera superior,



o ejecutando

```
SET SHOWPLAN_XML ON;
```

Esto evita la salida de resultados y, en cambio, genera un enlace que nos abre el visor gráfico del plan de la consulta. En él nos podemos encontrar:

1. Acceso a tabla y índice
 - a. Index Seek, Clustered Index Seek
 - b. Index Scan, Clustered Index Scan
 - c. Key Lookup (Clustered)
 - d. RID Lookup (Heap)
 - e. Table Scan
2. Operaciones de unión (join)
 - a. Nested Loops
 - b. Hash Match
 - c. Merge Join
3. Ordenar y agrupar
 - a. Sort
 - b. Sort (Top N Sort)
 - c. Stream Aggregate
 - d. Hash Match (Aggregate)
4. Sentencias Top-N
 - a. Top

[Markus Winand, Use The Index, Luke, a guide to database performance for developers](#)

Resumiendo todas las operaciones antes listadas, **seek** busca en índices o direcciones físicas mientras **scan** lee tablas o índices enteros —potencial candidato a una redefinición de índices—. **Key Lookup** [es un poco más complicado](#), pero quédate con que tampoco indica una operación costosa.

Cuando se trata de un *join* hablamos de dos tablas únicamente. Si fueran más de dos, se irán uniendo las restantes consecutivamente al resultado previo. Dependiendo de lo que se pida, del tipo de tablas y de los índices existentes se aplicará un método u otro.



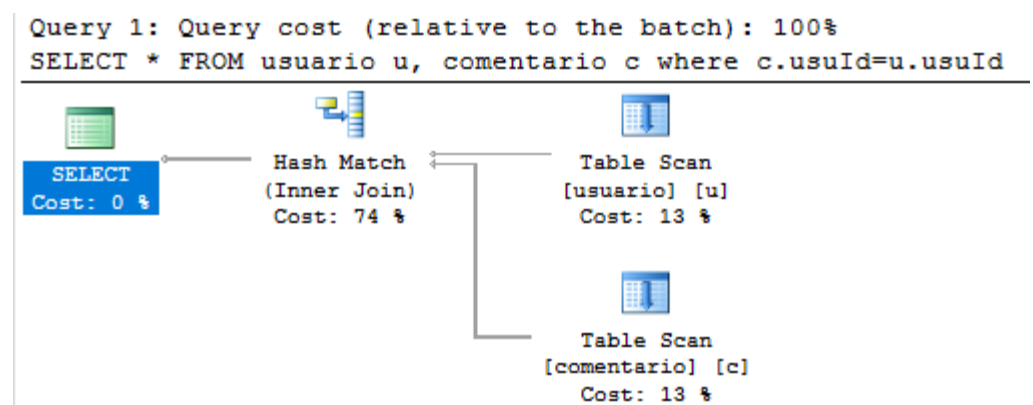
Ejecuta el siguiente [script](#). **No ejecutes** las órdenes que se muestran a continuación, es un extracto del script anterior para que se vea claro con qué datos estamos trabajando.

```
USE P103b
GO
create table usuario (
  usuId int,
  password varbinary(64),
  apodo nvarchar(15),
  email nvarchar(254) not null,
```

```
nombre nvarchar(50) not null,  
apellidos nvarchar(75) not null,  
nacido date not null,  
perfil nchar(3) default '0'  
);  
  
create table comentario (  
usuId int,  
numcom smallint,  
comenta nvarchar(1000) not null,  
respondea int,  
respondeanum smallint,  
cuando datetime2  
);  
GO  
insert into usuario values  
(1,NULL,'mercenario','pepe103@gmail.com','José Francisco Manuel','Pérez  
Pérez','2001-07-21','0'),  
(2,NULL,'mercenaria','apm52@gmail.com','Ana','Pérez Montesinos','1990-11-02','0'),  
(3,NULL,'castor','pepe103@gmail.com','José ','Pérez Pérez','1970-04-01','0'),  
(4,NULL,'leia','julipa@hotmail.com','Julia','Pastor Agulló','1970-07-11','0');  
  
insert into comentario values  
(1,1,'Feliz año nuevo!!',null,null,'2018-01-01 00:00:00'),  
(2,1,'Igualmente!!',1,1,'2018-01-01 00:01:00'),  
(4,1,'Todos los años lo mismo...',1,1,'2018-01-01 00:01:30'),  
(1,2,'Ya salió el aguafiestas',4,1,'2018-01-01 08:23:25'),  
(4,2,'Qosdén',1,2,'2018-01-01 17:10:56');  
GO
```

Copia y pega la siguiente orden en SSMS y pide el plan de ejecución de la siguiente consulta:

```
SELECT * FROM usuario u, comentario c where c.usuId=u.usuId;
```



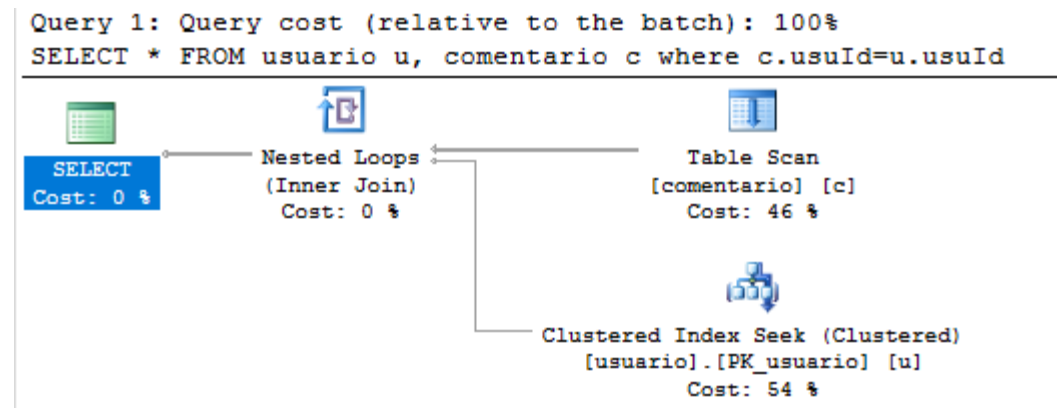
No tenemos ningún tipo de índice. Además tenemos que devolver absolutamente todas las filas. Por tanto, es necesario realizar *scan* sobre las dos tablas y mezclarlas, según el servidor, con un *hash match*. Si pasas el ratón por encima de las operaciones o las flechas de la secuencia de operaciones verás más datos.

Creamos un índice;

```
alter table usuario alter column usuId int not null;
go
alter table usuario with nocheck add constraint PK_usuario primary key(usuId);
go
```

Y volvemos a pedir el plan:

```
SELECT * FROM usuario u, comentario c where c.usuId=u.usuId;
```



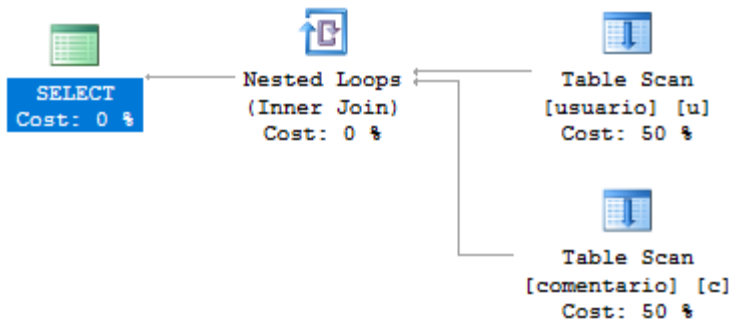
¿Qué va a hacer el servidor, en qué ha cambiado el plan?



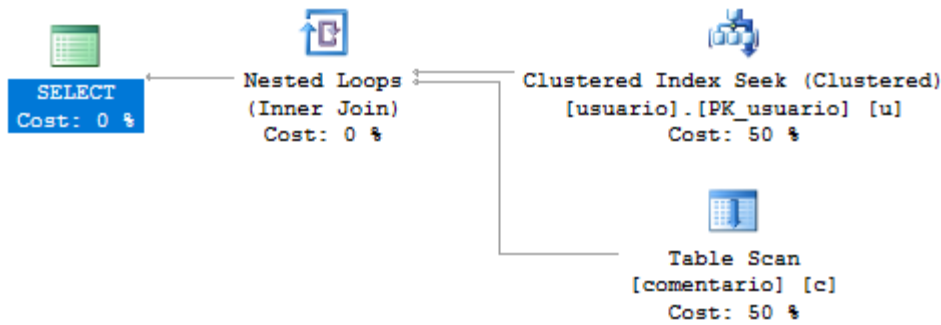
E06

Elimina el índice y pide el plan de la siguiente consulta:

```
alter table usuario drop constraint PK_usuario;
go
-----
SELECT nombre,apellidos,comenta,cuando FROM usuario u, comentario c
where c.usuId=u.usuId and u.usuId=4;
```

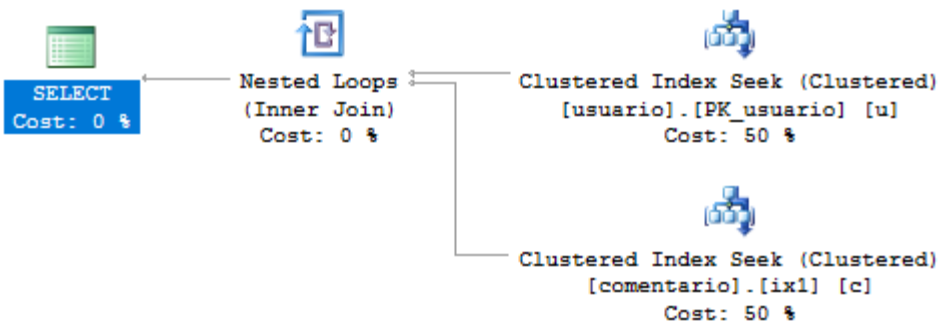


Vuelve a crear el índice y a pedir el plan:



Añadimos otro índice a la tabla COMENTARIO y pedimos el plan:

```
create clustered index ix1 on comentario(usuId);
```



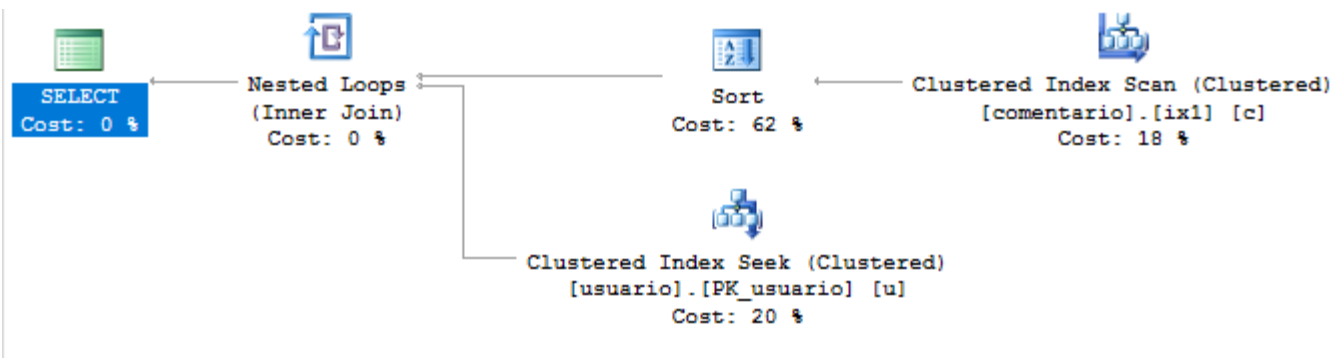
¿Qué ha ocurrido?



E07

Compara el plan anterior con el de la siguiente consulta:

```
SELECT nombre,apellidos,comenta,cuando FROM usuario u, comentario c
where c.usuId=u.usuId and c.respondea is not null
order by c.cuando;
```



Tablas con particiones

Si una tabla mantiene una gran cantidad de datos, las búsquedas pueden complicarse. Una solución posible consiste en particionar la tabla. Sin embargo, esta decisión debe basarse en criterios objetivos y contrastados.

Podríamos pensar en la tabla COMENTARIO y su columna "cuando". En nuestra base de datos todos los comentarios son del año 2017 pero podríamos particionar por meses. Eso nos daría algo parecido a 12 tablas más pequeñas, siempre y cuando nuestras consultas pudieran aprovecharse de esa partición en particular. Este escenario sería una **partición horizontal** de la tabla, es decir, por filas. Por ejemplo, la selección de los comentarios mejor valorados de cada mes.

También se puede abordar una **partición vertical**, decidiendo qué columnas van a cada una. Nuevamente, la decisión de una u otra y en qué datos concretos depende del uso previsto de la tabla.

Sin entrar en demasiados detalles, la base de datos debe preparar unos ficheros para almacenar los llamados *filegroups*. Ya después, cada tabla debe especificar en qué *filegroup* se va a almacenar.

**E08**

Realiza la partición horizontal de la tabla COMENTARIO en otra base de datos que crees para la ocasión. Compara tiempos de ejecución entre la tabla original y la particionada.

NOTA: hemos elegido la columna "cuando" para la discusión porque parece la más intuitiva. No obstante, si definimos una clave primaria —(usuld, numcom) en el caso de la tabla comentario— las columnas de particionamiento deben ser un subconjunto de las que definen esa clave primaria. Lo más fácil es crear una base de datos nueva, configurar el particionamiento en ella y volver a crear la tabla comentario que, después, rellenarás con datos desde la tabla original ya creada en P103. Tienes 3 opciones para este ejercicio:

1. Particionar por "cuando", pero debes incluir esta columna en la clave primaria.
2. Particionar por otro criterio, por ejemplo rangos de usuld o de numcom —en un entorno real tendríamos que tener en cuenta los futuros valores de estas columnas para tener un particionamiento equilibrado.
3. Particionar por "cuando", pero no definir clave primaria. Dificultaría el join con otras tablas, pero para este ejemplo no es problema.

Hay una cuarta opción que es el uso de [columnas calculadas](#), por ejemplo para trabajar exclusivamente con el número del mes.

Para insertar desde otra BD:

```
insert into nuevatabla select * from P103.dbo.comentario
```

Si solo quieres probar —para la prueba de tiempos sí necesitarás insertar todas las filas—, limita la consulta:

```
insert into nuevatabla select top(1000) * from P103.dbo.comentario
```



- Milica Medic, [Particionamiento de tablas de bases de datos en SQL Server](#), SQLShack, December 4, 2015
- [Crear tablas e índices con particiones](#)

Tablas en memoria

La idea es simple: todo lo que resida en memoria RAM va más rápido. ¿Desventajas? Es ciertamente finita y no se puede abusar de ella. No obstante, para ciertas cosas y previendo usos podemos aprovecharnos de este tipo de procesamiento y almacenamiento. Uno de estos usos es evitar la base de datos tempdb para tablas temporales.

SQL Server ofrece [OLTP en memoria](#), Procesamiento de Transacciones En Línea (*OnLine Transaction Processing*), que es un tema bastante más amplio que lo que aquí vamos a ver, y que se trata de alojar tablas en memoria. La creación de este tipo de tablas es básicamente el mismo que el de las "normales", añadiendo una cláusula with al final.

```
create table ...  
WITH (MEMORY_OPTIMIZED=ON, DURABILITY = SCHEMA_ONLY | SCHEMA_AND_DATA)
```

Tenemos dos opciones de durabilidad, SCHEMA_ONLY y SCHEMA_AND_DATA. La primera solo mantiene la estructura de la tabla al finalizar la sesión, la segunda mantiene esquema y datos como una tabla en disco duro. Existen limitaciones a tener en cuenta para una y otra opción.

También hay que preparar a la base de datos para poder manejar este tipo de tablas, no es una opción activada por defecto. Disponemos de un script publicado en [GitHub](#), entre otros sitios, que podemos adaptar con facilidad. La base de datos restaurada ya viene con esa característica activada.

Por último, ténganse en cuenta las limitaciones impuestas por cada edición del producto, en concreto [la cantidad máxima de datos almacenables en memoria](#).

**E09**

Ejecuta lo siguiente y compara tiempos

--Preparación, accede al script y ejecútalo; una vez es suficiente, las tablas temporales se mantendrán para toda la sesión

[Nombres y Apellidos, script SQL.](#)

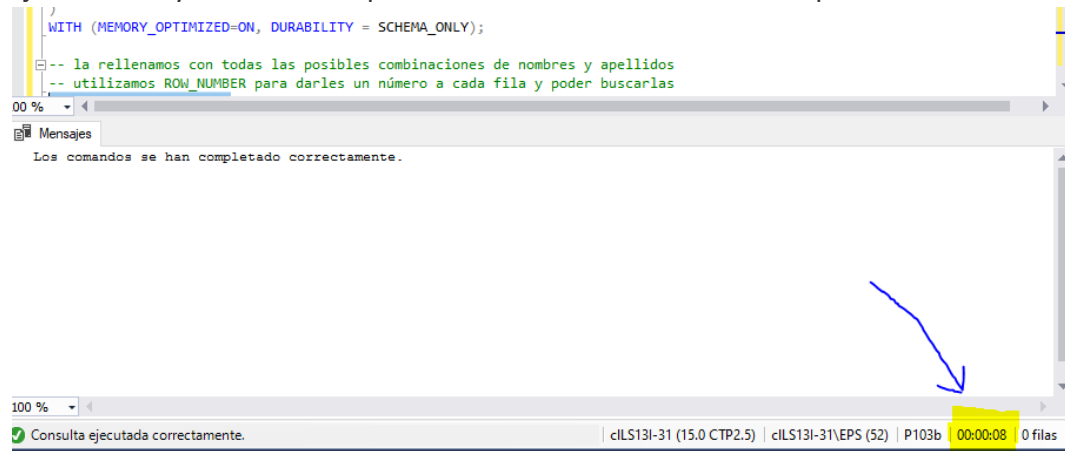
```
--Creación de tablas  
use P103b  
go  
drop table if exists combinas  
drop table if exists ccombinas
```

Ejecuta esto y anota el tiempo —basta la información de la esquina inferior derecha de MSSM—:

```
-- esta tabla residirá en memoria RAM  
create table combinas (  
orden int NOT NULL INDEX ix1 NONCLUSTERED,  
nombre varchar(25),  
apellidos varchar(75),
```

```
)  
WITH (MEMORY_OPTIMIZED=ON, DURABILITY = SCHEMA_ONLY);  
  
-- la rellenamos con todas las posibles combinaciones de nombres y apellidos  
-- utilizamos ROW_NUMBER para darles un número a cada fila y poder buscarlas  
insert into combinas  
select row_number() over (order by n.nombre) as orden,n.nombre,a1.apellido+' '+a2.apellido  
from #nombre n,#apellido a1,#apellido a2
```

Ejecuta esto y anota el tiempo —basta la información de la esquina inferior derecha de MSSM—:



```
-- lo mismo pero en una tabla normal  
create table ccombinas (  
orden int NOT NULL INDEX ix1 NONCLUSTERED,  
nombre varchar(25),  
apellidos varchar(75),  
);  
insert into ccombinas  
select row_number() over (order by n.nombre) as orden,n.nombre,a1.apellido+' '+a2.apellido  
from #nombre n,#apellido a1,#apellido a2
```

¿Qué ha llevado menos tiempo?

Ejercicios

Con la base de datos P103 restaurada —tendrás que hacer un nuevo *restore*—, resuelve estas consultas y haz lo que debas para mejorar sus tiempos de respuesta.

**E10**

Planteada la consulta siguiente, establece los índices necesarios para acelerar la respuesta:

```
select respondea, respondeanum, cuando
  from comentario
 where respondea is not null and cuando='2017-12-19'
```

Se entiende que es una consulta tipo, la fecha puede ser cualquiera.

**E11**

Comentarios de abril que son respuestas a otros comentarios. La salida debe mostrar

1. cuándo se respondió
2. quién respondió
3. en qué comentario suyo
4. a quién estaba respondiendo
5. y en qué comentario de aquél

La salida debe estar ordenada por usuario y número de comentario de los inicios de cadena, y fecha de la respuesta.

La tabla comentario tiene estas columnas:

- usuld int, -- el usuario que comenta
- numcom smallint, -- su comentario número x
- comenta nvarchar(1000) not null, -- lo que comenta
- respondea int, -- si tiene valor, su comentario es respuesta a este usuario...
- respondeanum smallint, -- ...en su comentario y
- cuando datetime2 -- cuándo ha comentado usuld

Los comentarios se identifican por (usuld, numcom).

El valor (respondea, respondeanum) es una referencia a otro comentario, si ambas columnas tienen valor son respuestas a otro comentario (respondea is not null and respondeanum is not null).

Para obtener el mes de la fecha: [month\(cuando\)](#)

¿Qué índices acelerarían la consulta?

**E12**

La misma consulta pero con únicamente para aquellos comentarios que han recibido más de 2 respuestas directas.

La forma de calcular si ha habido 3 o más respuestas es contar valores de (respondea, respondeanum) y agregarlos:

```
select respondea, respondeanum, count(*)  
from comentario  
group by respondea, respondeanum  
having count(*) > 2
```

El resultado anterior es “comentarios a los que se ha respondido más de dos veces”.

Group by busca cada pareja de valores de (respondea, respondeanum) y, con cada uno, realiza el cálculo de agregación que le pidamos —contar filas, en este caso—.

Supongamos:

A	10	comento
A	10	comento que
B	13	respondo
B	13	comento
B	13	respondo

El resultado de la consulta anterior sería:

A	10	2
B	13	3

Una vez obtenido este resultado, si queremos filtrar los resultados debemos usar *having*: `having count(*) > 2`

B	13	3
---	----	---

Si utilizamos la consulta anterior como una tabla en otra consulta más compleja:

```
select ...  
from tabla t,  
(select respondea, respondeanum, count(*)  
from comentario  
group by respondea, respondeanum  
having count(*) > 2) tt  
where t.x = tt.respondea and t.y=tt.respondeanum
```

La consulta en rojo se comportará como una tabla con el nombre “tt”. Y “t”, en realidad, es la propia tabla comentario utilizada ahora para recuperar la fecha “cuando”:

```
select t.usuld, t.numcom, tt.respondea, tt.respondeanum, t.cuando  
from comentario t,  
(  
    select respondea, respondeanum, count(*)  
    from comentario  
    group by respondea, respondeanum  
    having count(*) > 2  
) tt  
where t.usuld = tt.respondea and t.numcom=tt.respondeanum  
and month(t.cuando)=4
```

Entiéndase que la consulta tt se va a ejecutar primero, se convertirá en un resultado intermedio —en una tabla nueva y temporal— que se combinará con la tabla t (comentario).

Más sobre [group by](#) y [having](#).

¿Nos hacen falta más índices o tenemos los necesarios?



E13

Añade ahora, a la salida, nombre,apellidos y correo de los usuarios.

¿Nos hacen falta más índices o tenemos los necesarios?