Tema 3: Índice

¿Para qué sirven los índices?

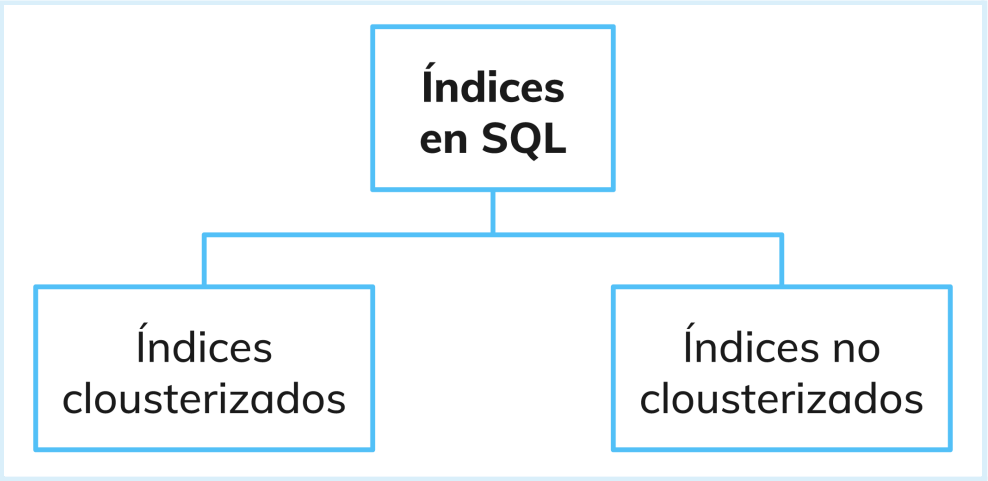
Los índices permiten agilizar la ejecución de consultas. Usar un índice para realizar una búsqueda por un determinado campo en una tabla con **muchos** registros permite obtener el resultado de manera más rápida que si no lo usáramos. Para tablas con pocos registros, el uso de un índice puede no notarse o, incluso, puede empeorar la *performance*.

Un índice SQL es una tabla de búsqueda rápida que permite encontrar los registros que los usuarios necesitan buscar con mayor frecuencia. Podemos pensar en un índice de un libro que indica en qué página se encuentra cada tema o en un índice de imágenes que nos indica en qué página se encuentra determinada imagen.

Otra analogía posible es pensar en el índice como un diccionario o una guía telefónica. En el diccionario, todas las palabras están ordenadas alfabéticamente y, para encontrar una, nos dirigimos directamente a la sección del diccionario, donde suponemos que se encuentra la primera letra de la palabra que buscamos. En el caso de la guía telefónica, los números de teléfono están ordenados por el apellido de los titulares de la línea. Para buscar el número de alguien, no lo buscamos en cada página, sino que también suponemos en qué página del libro estarán el apellido que buscamos.

En SQL, es posible distinguir entre los índices clousterizados y los índices no clousterizados:

Figura 5: Índices en SQL



Fuente: elaboración propia.

Veamos en detalle cada uno de ellos.

Índices clousterizados

Un índice clousterizado (o agrupado) define el orden en el que los datos de una tabla son almacenados físicamente. Debido a que los datos pueden almacenarse de una sola forma, no puede haber más de un índice clousterizado por tabla.

Este tipo de índice se implementa como una estructura de árbol que admite la recuperación rápida de los registros utilizando los valores de las columnas que se definieron para el índice.
Si una tabla no tiene un índice clousterizado, no estará almacenada de forma ordenada.

Los índices clousterizados son definidos en las columnas que se usan con mayor frecuencia en consultas. Cuando se crea una PRIMARY KEY para una tabla, automáticamente se crea un índice clousterizado asociado.

La sintaxis en SQL Server para crear un índice de este tipo es la siguiente:

CREATE CLUSTERED INDEX *nombre_indice* ON *tabla* (*columna*);
Veamos un ejemplo:
Vamos a crear nuevamente la tabla profesor sin clave primaria, usando el siguiente código:

```
CREATE TABLE `profesor_index` (
  `id` int NOT NULL,
  `nombre` varchar(45) NOT NULL,
```

```
  `apellido` varchar(45) NOT NULL,
  `fecha_nacimiento` date NOT NULL,
  `salario` float NOT NULL
);
```

Luego, insertaremos datos en ella, a partir de los registros de la tabla original, ordenados por apellido:

```
insert into profesor_index
select * from alkemy.profesor
order by apellido;
```

Cuando consultamos la tabla “profesor_index”, obtenemos registros ordenados como fueron insertados:

Tabla 39. Profesor_index

	id	nombre	apellido	fecha_nacimiento	salario
	4	Lucía	Díaz	1991-02-24	45000
	5	Raúl	Martínez	1980-10-15	85000
	2	María Emilia	Paz	1984-07-15	72000
	1	Juan	Pérez	1990-06-06	55000
	6	Mabel	Ríos	1982-06-12	83000

Fuente: elaboración propia.

Ahora, crearemos la clave primaria, que creará automáticamente un índice clousterizado:

```
alter table profesor_index add constraint pk_id primary key(id);
```

Por último, vamos a consultar nuevamente la tabla profesor_index. Veremos los registros ordenados por el campo id que fue incluido como índice:

Tabla 40. Profesor_index

	id	nombre	apellido	fecha_nacimiento	salario
▶	1	Juan	Pérez	1990-06-06	55000
	2	María Emilia	Paz	1984-07-15	72000
	3	Martín	Correa	1987-12-07	63000
	4	Lucía	Díaz	1991-02-24	45000
	5	Raúl	Martínez	1980-10-15	85000
	6	Mabel	Ríos	1982-06-12	83000

Fuente: elaboración propia.

Índices no clousterizados

A diferencia de los índices clousterizados, los no clousterizados no ordenan físicamente los datos de la tabla, por lo tanto, pueden definirse varios en cada tabla. Este tipo de índice crea una estructura separada, que le permite localizar la fila deseada de la tabla real (similar a los índices de los libros).

Cuando una consulta es lanzada contra una columna en la cual el índice es creado, la base de datos primero irá al índice y buscará la dirección de la fila correspondiente en la tabla; luego, irá a esa dirección de fila y obtendrá otros valores de columna. Es debido a este paso adicional que los índices no agrupados son más lentos que los índices agrupados.

Para crear este tipo de índices, también deben considerarse las columnas consultadas con frecuencia y analizar si el costo de crear un índice se justifica con la mejora en la *performance*.

Para entender mejor su funcionamiento, veamos un ejemplo: supongamos que estamos escribiendo un libro de programación de cinco capítulos. El segundo capítulo se titula “Tipos de datos” y comienza en la página 19; el tercer capítulo se titula “Estructuras de repetición” y comienza en la página 50. Al finalizar la edición del libro y de su índice, nos damos cuenta de que nos olvidamos de incluir un subtema que lleva el título de “Tipo de dato booleano” en el segundo capítulo. Ante esa situación, volvemos a ese capítulo y agregamos dos páginas más para ese subtema y damos el trabajo por finalizado. Sin embargo, es necesario actualizar también el índice, ya que, al agregar dos páginas en el capítulo 2, el capítulo 3 ya no comenzará en la página 50, sino que iniciará en la página 52, al igual que los otros capítulos: sus páginas iniciales serán incrementadas en dos unidades. Lo mismo sucede con los índices en SQL: al ejecutar sentencias DML sobre la tabla, es necesario actualizar también los datos del índice.

Hay que ser cuidadosos con la cantidad de índices no clousterizados en una tabla. Como vimos en el ejemplo anterior, cada sentencia DML implica un procesamiento extra en la estructura del índice. Si tenemos muchos índices no clousterizados por tabla, un INSERT implicara insertar el registro en la tabla y actualizar los datos de cada una de las estructuras de cada uno de los índices para esa tabla.

Sintaxis en SQL Server para índices no clousterizados:

CREATE NONCLUSTERED INDEX *nombre_indice* ON *tabla* (*columna*);

PK, FK e índices

La creación de claves primarias y foráneas implica también la creación de índices.

Cuando creamos una clave primaria, automáticamente se crea un índice clousterizado. Como ya vimos anteriormente, al igual que una tabla puede tener solamente una clave primaria, solo puede existir un índice clousterizado por tabla.

Similar a las claves primarias, cuando creamos una clave foránea, automáticamente se crea un índice no clousterizado. Debido a que podemos tener varias claves foráneas en una tabla, también podemos tener varios índices no clousterizados en ella.

Cuando una tabla tiene índices definidos, siempre es mejor hacer consultas sobre esos campos. Es por esto que, si hacemos consultas complejas con JOIN, siempre es mejor utilizar los campos de clave primaria y foránea a la hora de realizar el cruce de tablas. En un contexto sin índices, el motor de base de datos deberá escanear (leer) completamente ambas tablas en busca de valores que coincidan. Utilizando índices, la búsqueda será mucho más rápida, puesto que los índices permitirán encontrar las coincidencias con más rapidez.

Documentación

Documentación de Microsoft sobre los índices:

Microsoft (2019). Índices agrupados y no agrupados descritos. Recuperado de https://docs.microsoft.com/es-es/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described?view=sql-server-ver15

Diferencias entre los tipos de índices:

Richardson, B. (2018). ¿Cuál es la diferencia entre Índices Agrupados y No Agrupados en SQL Server? Recuperado de https://www.sqlshack.com/es/cual-es-la-diferencia-entre-indices-agrupados-y-no-agrupados-en-sql-server/

Guía de diseño de índices:

Microsoft (2019a). Guía de diseño y de arquitectura de índices de SQL Server. Recuperado de https://docs.microsoft.com/es-es/sql/relational-databases/sql-server-index-design-guide?view=sql-server-ver15

Videos

Índices clousterizados y no clousterizados:

Fuente: Cardenas Valenzuela, V. H. [Victor Hugo Cardenas Valenzuela] (s. f.). *Explicación de los Índices en el SQL Server de Microsoft* [YouTube]. Recuperado de https://www.youtube.com/watch?v=XWX1YvS5Kec

Ejercitación 3

Objetivo

El objetivo de este ejercicio es poder visualizar la diferencia y el impacto que tienen los índices aplicados a una tabla.

1) Para visualizar el concepto de “índices clousterizados”, ingresa a https://sqliteonline.com/ utilizando el motor “MariaDB” y sigue los siguientes pasos:

a. Crea una tabla persona sin PRIMARY KEY y solamente con dos campos: id y nombre.
create table persona(id bigint(20), nombre varchar(50));

b. Inserta datos siguiendo un orden no secuencial para el id.
insert into persona values (2,'Nombre 1');

insert into persona values (7,'Nombre 2');

insert into persona values (1,'Nombre 3');

insert into persona values (5,'Nombre 4');

c. Consulta los datos para visualizar el orden de registros.

d. Agrega una clave primaria para el campo id.

alter table persona add CONSTRAINT pk_id PRIMARY key(id);

e. Consulta los datos para visualizar el orden de registros.

f. Contesta la siguiente pregunta: ¿Por qué, después de agregar la clave primaria, el orden de los registros es diferente?

2) Selecciona las diferencias entre un índice clousterizado y un índice no clousterizado:

a. La cantidad de campos incluidos en el índice: el índice clousterizado solo puede incluir un campo y el índice no clousterizado puede incluir varios.

b. Una tabla puede tener solo un índice no clousterizado y muchos índices clousterizados.

c. **Al crear un índice clousterizado, se ordena físicamente la tabla. En cambio, al crear un índice no clousterizado, se crea una estructura aparte para ese índice.**

Formato de entrega: la actividad se entrega a través de una URL correspondiente al repositorio sobre el que se haya trabajado.