# Práctica de nivel físico. Índices.

Vamos a realizar una pequeña práctica de índices. Como ya sabréis los índices en bases de datos son estructuras de información adicionales que el gestor asocia (generalmente a una tabla) y que le permiten acelerar determinadas operaciones (principalmente consultas). Sin embargo, estas estructuras adicionales tienen un coste extra de creación y mantenimiento, por lo cual otras operaciones podrán verse perjudicadas por la presencia de los mismos, en particular operaciones de modificación (INSERT, DELETE, UPDATE).

Existen varios tipos de índices, los más habituales son los índices basados en árboles-B+ (este es el que habitualmente se denomina simplemente índice), los índices basados en bitmaps, los índices sobre funciones y los índices de join.

Cada tipo de índice está pensado para acelerar la búsqueda de determinado tipo de información bajo determinadas suposiciones. Así, por ejemplo, los índices de tipo árbol-b+ son muy efectivos cuando los claves de indexación tienen una tasa alta en la variabilidad (una muestra dispersa) de valores, esto es, el cociente resultante entre dividir el número de valores diferentes para la clave entre el número de filas existentes es un número elevado (la mayor tasa la poseen las claves de índices que además son claves de tipo PRIMARY KEY o UNIQUE, ya que estos tendrán una tasa de 1).

Si la tasa de valores es próxima a 0, esto es, hay realmente pocos valores diferentes en las claves del índice (una muestra no dispersa o concentrada), entonces es mejor utilizar índices de tipo bitmap.

Usar índices de árbol-b+ con atributos que no presentan alta variabilidad o un árbol de tipo bitmap con un atributo que sí presenta alta variabilidad puede llegar a no proporcionar beneficio apreciable y además a perjudicar muy gravemente el rendimiento de las operaciones de modificación.

Los índices basados en funciones son índices cuyas claves de indexación no está compuesta por atributos de las tablas, sino por los valores resultantes de una expresión (denominada función). El objetivo de este tipo de índices es acelerar la búsqueda basadas en dichas expresiones.

Los índices de tipo join son estructuras que almacenan información de qué filas de una tabla se reúnen (join) con qué filas de otra tabla. Se debe observar que no se almacena el join de las tablas, sino simplemente la información de qué filas se logran reunir con cuál otras. Como se puede deducir el objetivo es acelerar operaciones de join específicas.

> **Nota:** Aunque no lo vemos en profundidad, el número de tipos y variantes de índices es mucho más complejo de lo que hemos visto en teoría. Así existen índices densos (aquellos que tienen una entrada de índice por cada fila de la tabla) e índices dispersos (aquellos que tienen una entrada de índice para varias filas de la tabla); también existen índices no ordenados e índices ordenados (según requieran o no que la información de la tabla esté organizada siguiendo o no un orden específico).

Además, hemos comentado que un índice es una estructura adicional, por lo que no debería afectar a la estructura propiamente dicha de la tabla sobre la que se define. Hace tiempo que esto no es así, ya que en los gestores de bases de datos modernos un índice sí puede alterar la estructura de almacenamiento subyacente a una tabla. Ejemplo de esto los encontramos en los denominados índices fat y semi-fat. Un índice semi-fat es aquel en el que los atributos que forman parte de la clave del índice, que están almacenados en el índice, se eliminan de la estructura de almacenamiento de la tabla (esto es, los campos clave del índice se almacenan únicamente en el índice, y en la estructura de la tabla están el resto de campos). Un índice fat es aquel en el que todos los atributos de la tabla forman parte de la clave del índice, en este caso toda la tabla se almacena en la estructura de almacenamiento del índice y desaparece la estructura de la tabla propiamente dicha.

Desde esta perspectiva los sistemas de almacenamiento de tablas de tipo cluster podrían ser interpretados también como un almacenamiento basado en índice.

Cualquier usuario con privilegio de RESOURCE, puede construir índices en su esquema, al igual que tablas, vistas, etc. Sin embargo, para cuantificar de forma objetiva la mejora de rendimiento vamos a necesitar activar el autotrace de SQLPlus, lo que requiere de ciertas operaciones de configuración para usuarios convencionales (debido a que requiere almacenar los planes de ejecución y estadísticas en el esquema del usuario). Por este motivo para simplificarnos el trabajo vamos a realizar gran parte de la práctica usando un usuario con privilegios de DBA.

1. Para no ensuciar los esquemas de SYS, ni de SYSTEM, vamos a crearnos un usuario normal y le vamos a dar privilegios de DBA (asignándole el rol DBA a dicho usuario):

A partir de ahora nos conectamos como este usuario que acabamos de crear y continuamos con la actividad de índices.

2. Vamos a crearnos una tabla denominada PRUEBA con 6 atributos, todos ellos serán de tipo NUMBER(16, 0). Los campos serán respectivamente CLAVE, DISPERSO, CONCENTRADO, IDISPERSO, ICONCENTRADO, BCONCENTRADO. El campo CLAVE será además la clave primaria de esta tabla.

Los atributos se van a corresponder con (aún no hay que crear los índices, excepto la propia clave primaria):

- CLAVE contendrá una clave primaria.
- DISPERSO contendrá valores con alta variabilidad (una muestra dispersa) y no estará indexado de ninguna forma.
- CONCENTRADO contendrá valores con baja variabilidad (una muestra concentrada) y tampoco estará indexado.

- IDISPERSO contendrá valores con alta variabilidad (una muestra dispersa) y tendrá un índice de tipo árbol-B+.
- ICONCENTRADO contendrá valores con baja variabilidad (una muestra concentrada) y tendrá un índice de tipo árbol-B+.
- BCONCENTRADO contendrá valores con baja variabilidad (una muestra concentrada) y tendrá un índice de tipo bitmap.

Vamos a rellenar esta tabla según las especificaciones anteriores con 100000 filas copiando y ejecutando el siguiente código PL/SQL:

```
DECLARE

I NUMBER(16,0);

R NUMBER(16,0);

BEGIN

FOR I IN 1..500000 LOOP

  R := DBMS_RANDOM.VALUE(1,1000000000);

  INSERT INTO PRUEBA VALUES(I, R, MOD(R,11), 1000000000-R, MOD(1000000000-R, 11),
MOD(2000000000-R, 11));

END LOOP;

END;
```

No hay que olvidar realizar un COMMIT cuando finalice el bloque de código anterior.

3. Ahora vamos a crear los tres índices que hemos indicado sobre la tabla:

- Uno denominado PID sobre el atributo IDISPERSO
- Uno denominado PIC sobre el atributo ICONCENTRADO
- Uno denominado PBC de tipo BITMAP sobre el atributo BCONCENTRADO

Ahora vamos a activar el modo AUTOTRACE y mejorar las estadísticas, usando las sentencias:

```
SET AUTOTRACE ON;

ALTER SESSION SET STATISTICS_LEVEL='ALL';
```

Para trabajar con este modo, si estamos trabajando con SQLDeveloper, a partir de ahora usaremos la opción Ejecutar script (F5) en lugar de Sentencia de ejecución (CTRL+Intro). La consecuencia de esta activación es que cada vez que ejecutemos una consulta Oracle mostrará el resultado de la misma, pero además mostrará el árbol del plan de ejecución y las estadísticas recolectadas de la misma. Ten en cuenta que esta información adicional no se muestra en SQLDeveloper si ejecutamos las consultas usando la opción de Sentencia de ejecución.

En el plan de ejecución veremos si se está haciendo uso del índice y en las estadísticas podremos observar determinadas variables de interés recolectadas durante la ejecución de la sentencia. Dependiendo de la versión de SQLPlus y del intérprete utilizado la cantidad de estadísticas recolectadas y mostradas puede diferir. Para nosotros hay dos estadísticas claves: A-time y Reads del plan y "consistent gets" y "physical read total bytes" del autotrace. Esto nos indica el tiempo total acumulado, las lecturas realizadas, cuántas lecturas de bloque en RAM se han realizado y la cantidad de bytes totales leída de los dispositivos de almacenamiento (no incluye los bytes leídos de buffer si los hubiere).

Interpretar los comportamientos de consultas puede llegar a ser muy complejo, ya que posiblemente los sistemas de caché de Oracle puedan falsear el comportamiento de una consulta (si parte de la información necesaria ya está en el sistema de caché). Esto es debido a que si Oracle encuentra la consulta ya realizada o bien parte de su información ya cacheada obtendrá la información sin necesidad de lanzar ningún tipo de acceso a los dispositivos.

Hay una serie de interferencias que nos van a afectar:

- Que Oracle tenga cacheado el resultado completo de la consulta (pues ya se ha realizado antes). Esta interferencia es evitable vaciando la estructura que almacena estos resultados (el SHARED_POOL).
- Que Oracle tenga cacheado los bloques de la tabla prueba (puede acceder a los datos de la tabla sin necesidad de acceder al dispositivo físico). Esta interferencia es evitable vaciando la caché que contiene los bloques de nuestra tabla (el BUFFER_CACHE).
- Que el sistema sobre el que se monta Oracle tenga cacheado los bloques de la tabla en su propia caché:
    - Esto no ocurre en Windows (Oracle accede a los ficheros de datos indicándole a Windows que no haga caché de esos accesos, para evitar una doble caché).
    - Esto puede ocurrir en Linux (ya que Oracle por defecto no le indica a Linux que haga un acceso directo haciendo un bypass de la caché). Existe una forma de configurar los sistemas Linux para que esto no ocurra, pero como hemos indicado no es el comportamiento por defecto de la mayoría de las distribuciones.
    - Esto ocurre casi siempre en virtualización, ya que el "host" posiblemente esté cacheando los accesos al fichero que contiene el disco del "guest".

Para evitar las interferencias de las cachés de Oracle vamos a ejecutar siempre las dos siguientes sentencias inmediatamente antes de cualquiera de las consultas solicitadas (siempre delante de cada consulta que queramos "medir"):

```
ALTER SYSTEM FLUSH SHARED_POOL;

ALTER SYSTEM FLUSH BUFFER_CACHE;
```

Estas dos sentencias se encargarán de vaciar tanto el pool compartido como la caché del buffer.

Cuando un bloque está cacheado en alguna parte de la memoria principal este no es solicitado al dispositivo, sino que se usa la copia existente en memoria. Toda solicitud suele pasar por la caché primero, si el bloque solicitado está en la caché se produce un caché hit, si no está en la caché previamente entonces se produce un caché miss y se lee del dispositivo.

Cuando tenemos un hit, la lectura efectiva del dispositivo no es necesaria, por lo que el tiempo que se tarda para obtener esta información es mucho menor que si se hubiera tenido que leer el bloque del dispositivo, lo que sí ocurriría en un miss.

Desgraciadamente podemos contrarrestar las interferencias de Oracle, pero no podemos contrarrestar las interferencias debido al sistema operativo, por lo que no tendremos en consideración lo que la consulta tarda ni las métricas del sistema operativo. Por suerte las dos variables que vamos a observar, “consistent gets” y “physical read total bytes”, computan los accesos a los dispositivos ordenados por Oracle (debidos a miss de Oracle) sin importar si hacen o no hit en la caché del Sistema Operativo, dándonos un resultado fiable del coste de la ejecución en cuanto a acceso a dispositivo se refiere.

Recordamos nuevamente que para evitar los hits de Oracle debéis vaciar tanto el SHARED_POOL como BUFFER_CACHE inmediatamente antes de la ejecución de cada una de las sentencias a medir.

4. Las consultas que vamos a realizar serán las siguientes:

- SELECT COUNT(*) FROM PRUEBA WHERE CLAVE = 50000;
- SELECT COUNT(*) FROM PRUEBA WHERE DISPERSO = 50000;
- SELECT COUNT(*) FROM PRUEBA WHERE CONCENTRADO = 5;
- SELECT COUNT(*) FROM PRUEBA WHERE IDISPERSO = 50000;
- SELECT COUNT(*) FROM PRUEBA WHERE ICONCENTRADO = 5;
- SELECT COUNT(*) FROM PRUEBA WHERE BCONCENTRADO = 5;

Copiar y rellenar la siguiente tabla:

-

| | | | | | | |
|---|---|---|---|---|---|---|
| SELECT COUNT(*) FROM PRUEBA WHERE | CLAVE = 50000 | DISPERSO = 50000; | CONCENTRADO = 5; | IDISPERSO = 50000; | ICONCENTRADO = 5; | BCONCENTRADO = 5; |
| PLAN | INDEX UNIQUE SCAN\| PRUEBA_PK | | | | | |
| A-Time | | | | | | |
| Reads | | | | | | |
| “consisten t gets | | | | | | |
| “physical read total bytes | | | | | | |

-

5. Basándonos en los resultados obtenidos, principalmente en los valores para los campos "consistent gets" y "physical read total bytes", interpreta los resultados obtenidos en cada una de las consultas y compáralos entre ellas. Establece cual de las consultas efectuadas es más rápida, cuál más lenta y porqué.

6. Ahora procederemos a observar el principal inconveniente de los índices, su penalización en operaciones de escritura que afecten atributos indexados.

Para ello, y recordando ejecutar las dos sentencias de vaciado de caché inmediatamente antes de cada una de las siguientes instrucciones, ejecuta y compara las estadísticas (en particular las mismas variables observadas anteriormente más la variable denominada "cell physical IO interconnect bytes", pues estas operaciones además de leer van a escribir y este estadístico nos suma el total de bytes leídos y escritos), de las siguientes sentencias:

- UPDATE PRUEBA SET DISPERSO = DISPERSO + 7;
- UPDATE PRUEBA SET IDISPERSO = IDISPERSO + 7;

¿Qué operación es más costosa? ¿Por qué?

7. Ahora vamos a crear un índice de función. Estos índices no se definen sobre un atributo, sino sobre una expresión. Para ello ejecutemos y observemos las estadísticas de la siguiente función (recuerda vaciar las cachés):

- SELECT COUNT(*) FROM PRUEBA WHERE IDISPERSO BETWEEN 10000 AND 20000;

En el plan de ejecución deberías poder observar que se realiza una operación denominada INDEX RANGE SCAN. Dado que estamos realizando una búsqueda por un rango y el índice está ordenado se puede realizar la búsqueda del rango usando el propio índice.

8. No obstante, imagina que habitualmente realizamos esta operación (vacía las cachés antes):

- SELECT COUNT(*) FROM PRUEBA WHERE IDISPERSO+ICONCENTRADO BETWEEN 10000 AND 20000;

Deberías observar que ahora la operación que se efectúa es un TABLE ACCESS FULL.
Compara las estadísticas de ambas instrucciones, ¿qué ha pasado?

9. Construyamos ahora el índice sobre función denominado FIX sobre la expresión IDISPERSO+ICONCENTRADO.

10. Vuelve a ejecutar la misma consulta de antes (vacía las cachés antes):
- SELECT COUNT(*) FROM PRUEBA WHERE IDISPERSO+ICONCENTRADO BETWEEN 10000 AND 20000;

11. Compara los resultados de esta consulta antes y después de la creación del índice FIX. ¿Qué ha pasado?

12. Ahora vuelve a ejecutar la operación de actualización que realizamos antes (no olvides vaciar la caché):

- UPDATE PRUEBA SET IDISPERSO = IDISPERSO + 7;

13. ¿Existe alguna diferencia significativa respecto a las estadísticas de la primera vez que la ejecutamos? ¿Cuál? ¿A qué se debe?