CAPÍTULO 16

Diseño físico y refinación de la base de datos

En el último capítulo explicamos diversas técnicas para poder procesar eficazmente las consultas. En éste explicamos algunos problemas que afectan al rendimiento de una aplicación que se ejecuta en un DBMS. En particular, explicamos las opciones disponibles que los administradores de bases de datos tienen para almacenar bases de datos y algunas de las soluciones y técnicas que utilizar para refinar la base de datos a fin de mejorar su rendimiento. En primer lugar, la Sección 16.1 explica los problemas que surgen en el diseño físico de la base de datos que tienen que ver con el almacenamiento y el acceso a los datos. Después, en la Sección 16.2 explicamos cómo mejorar el rendimiento de la base de datos a través de la refinación, la indexación de datos, el diseño y las propias consultas.

16.1 Diseño físico de las bases de datos relacionales

En esta sección explicamos los factores de diseño físico que afectan al rendimiento de las aplicaciones y las transacciones; y después comentamos las directrices específicas para los RDBMSs.

16.1.1 Factores que influyen en el diseño físico de una base de datos

El diseño físico es una actividad cuyo objetivo no sólo es crear la estructuración adecuada de los datos en su almacenamiento, sino también hacer algo que garantice un buen rendimiento. Para un esquema conceptual dado, hay muchas alternativas de diseño físico en un DBMS dado. No es posible tomar decisiones significativas sobre el diseño físico y analizar el rendimiento mientras no conozcamos las consultas, las transacciones y las aplicaciones que se esperan ejecutar sobre la base de datos. Debemos analizar estas aplicaciones, su frecuencia de ejecución esperada, cualquier restricción de tiempo en su ejecución y la frecuencia estimada de las operaciones de actualización. A continuación explicamos estos factores.

A. Análisis de las consultas y las transacciones de la base de datos. Antes de emprender el diseño físico de la base de datos, debemos tener una buena idea del uso que se espera hacer de la base de datos, que lo definiremos en un formulario de alto nivel, así como de las consultas y las transacciones que es previsible ejecutar en la base de datos. Por cada consulta, debemos especificar lo siguiente:

1. Los ficheros a los que la consulta accederá.[1]
2. Los atributos con los que se especificará cualquier condición de selección para la consulta.
3. Si la condición de selección es una condición de igualdad, desigualdad o de rango.
4. Los atributos con los que se especificará cualquier condición de concatenación para enlazar varias tablas u objetos para la consulta.
5. Los atributos cuyos valores la consulta recuperará.

Los atributos mencionados en los puntos 2 y 4 son candidatos para la definición de estructuras de acceso. Para cada transacción u operación de actualización, debemos especificar lo siguiente:

1. Los ficheros que se actualizarán.
2. El tipo de operación en cada fichero (inserción, actualización o eliminación).
3. Los atributos con los que se especifican las condiciones de selección para una eliminación o una actualización.
4. Los atributos cuyos valores cambiarán a causa de una operación de actualización.

Una vez más, los atributos mencionados en el punto 3 son candidatos para las estructuras de acceso. Por el contrario, los atributos del punto 4 son candidatos a evitarse en una estructura de acceso, puesto que la modificación de esos atributos requeriría la actualización de las estructuras de acceso.

B. Análisis de la frecuencia de ejecución esperada de consultas y transacciones. Además de identificar las características de las consultas y transacciones esperadas, debemos considerar la frecuencia de invocación que esperamos. La información sobre esta frecuencia, junto con la información de los atributos recopilada en cada consulta y transacción, se utiliza para recopilar una lista acumulativa de frecuencias de uso esperadas para todas las consultas y transacciones. Esto se expresa como la frecuencia esperada de uso de cada atributo de cada fichero como un atributo de selección o un atributo de concatenación, sobre todas las consultas y transacciones. Por regla general, para grandes volúmenes de procesamiento, se aplica la *regla informal del 80-20*, que dice que aproximadamente el 80% del procesamiento se debe al 20% de las consultas y las transacciones. Por consiguiente, en situaciones prácticas, rara vez es necesario recopilar unas estadísticas y tasas de invocación exhaustivas para todas las consultas y transacciones; es suficiente determinar el 20% de las más importantes.

C. Análisis de las restricciones de tiempo para consultas y transacciones. Algunas consultas y transacciones pueden tener restricciones de rendimiento severas. Por ejemplo, una transacción puede tener la restricción de que debe terminar en 5 segundos el 95% de las veces que se la invoca y que nunca debe tardar más de 20 segundos. Dichas restricciones de rendimiento fijan prioridades más extensas en los atributos que son candidatos a las rutas de acceso. Los atributos de selección que las consultas y las transacciones utilizan con restricciones de tiempo se convierten en candidatos de alta prioridad a las estructuras de acceso principales.

D. Análisis de las frecuencias esperadas de las operaciones de actualización. Debemos especificar una cantidad mínima de rutas de acceso para un fichero que se actualiza con frecuencia, porque la actualización de las propias rutas de acceso ralentiza las operaciones de actualización.

E. Análisis de las restricciones de unicidad en los atributos. Las rutas de acceso deben especificarse en todos los atributos (o conjuntos de atributos) clave que son candidatos a ser la clave principal o que están restringidos para ser únicos. La existencia de un índice (o de otra ruta de acceso) hace suficiente explorar el índice sólo cuando se verifica esta restricción, puesto que todos los valores del atributo existirán en los nodos hoja del índice.

---

[1] Por simplicidad utilizamos el término *ficheros*. Lo podemos sustituir por tablas o relaciones.

Una vez recopilada la información precedente, podemos afrontar las decisiones sobre el diseño físico de la base de datos, que consisten principalmente en decidir las estructuras de almacenamiento y las rutas de acceso para los ficheros de base de datos.

16.1.2 Decisiones sobre el diseño físico de la base de datos

La mayoría de los sistemas relacionales representan cada relación base como un fichero de base de datos físico. Las opciones de ruta de acceso incluyen especificar el tipo de fichero para cada relación y los atributos sobre los que deben especificarse los índices. Como máximo, uno de los índices de cada fichero puede ser un índice principal o agrupado. Es posible crear cualquier cantidad de índices secundarios.[2]

Decisiones de diseño sobre la indexación. Los atributos cuyos valores son requeridos por las condiciones de igualdad o de rango (operación de selección) y los que son claves o participan en condiciones de concatenación (operación de concatenación) requieren rutas de acceso.

El rendimiento de las consultas depende ampliamente de los índices o de los esquemas de dispersión existentes para acelerar el procesamiento de las selecciones y las concatenaciones. Por el contrario, durante las operaciones de inserción, eliminación o actualización, la existencia de índices supone un coste, que se puede justificar por una mayor eficacia a la hora de tramitar las consultas y las transacciones.

Las decisiones sobre el diseño físico de la indexación se encuadran en las siguientes categorías:

1. **Cuándo indexar un atributo.** Si un atributo es clave o si se utiliza para alguna consulta, para una condición de selección (igualdad o rango de valores) o para una concatenación, hay una justificación inicial para crear un índice por ese atributo. Un factor a favor de crear muchos índices es que algunas consultas puedan procesarse explorando simplemente los índices, sin necesidad de recuperar datos.
2. **Qué atributo o atributos indexar.** Un índice se puede crear con uno o varios atributos. Si varios atributos de una relación están implicados todos ellos en varias consultas, [por ejemplo, (NumEstiloPrenda, Color) en una base de datos de inventario de prendas de vestir], está justificado un índice multiatributo. La ordenación de los atributos dentro de un índice multiatributo debe corresponder a las consultas. Por ejemplo, el índice anterior asume que las consultas estarían basadas en una ordenación de los colores por cada NumEstiloPrenda, y no a la inversa.
3. **Cuándo configurar un índice agrupado.** Como máximo, en una tabla sólo puede haber un índice principal o agrupado porque esto implica que el fichero está ordenado físicamente por ese atributo. En la mayoría de los RDBMSs, esto se especifica con la palabra clave CLUSTER. (Si el atributo es una clave, se crea un índice principal, mientras que un índice agrupado se crea si el atributo no es una clave). Si una tabla requiere varios índices, la decisión sobre cuál debe ser un índice agrupado depende de si es necesario mantener la tabla ordenada por ese atributo. Los índices agrupados favorecen las consultas por rango. Si varios atributos requieren consultas por rango, deben evaluarse los beneficios relativos antes de decidir el atributo por el que agrupar. Si la consulta se puede responder con una simple búsqueda en el índice (sin recuperar los registros de datos), el índice correspondiente *no* debe agruparse, puesto que el beneficio principal del agrupamiento se consigue si hay que recuperar los registros. Se puede configurar un índice agrupado como un índice multiatributo si la recuperación de rango por esa clave compuesta resulta de utilidad en la creación de un informe (por ejemplo, un índice sobre CodPostal, IdAlmacén e IdProducto pueden ser un índice agrupado para los datos de ventas).
4. **Cuándo utilizar un índice de dispersión sobre un índice de árbol.** En general, los RDBMSs utilizan árboles B^+ para la indexación. No obstante, en algunos sistemas también se proporcionan índices ISAM y de dispersión (consulte el Capítulo 14). Los árboles B^+ soportan consultas de igualdad y de

[2] El lector debe repasar los distintos tipos de índices descritos que describimos en la Sección 13.1. Para entender mejor esta explicación, también resulta útil familiarizarse con los algoritmos de procesamiento de consultas descritos en el Capítulo 15.

rango sobre el atributo utilizado como clave de búsqueda. Los índices de dispersión funcionan bien con las condiciones de igualdad, en particular durante las concatenaciones para encontrar el(los) registro(s) coincidente(s).

5. **Cuándo utilizar la dispersión dinámica para el fichero.** En los ficheros que son muy volátiles (es decir, aquellos cuyo tamaño aumenta y se reduce continuamente) sería conveniente uno de los esquemas de dispersión dinámica que explicamos en la Sección 13.9. Actualmente, la mayoría de los RDBMSs comerciales no los ofrecen.

Desnormalización como decisión de diseño para acelerar las consultas. El objetivo último durante la normalización (consulte los Capítulos 10 y 11) es separar los atributos relacionados lógicamente en tablas a fin de minimizar la redundancia, y así evitar las anomalías que pueden llevar a un procesamiento extra a la hora de mantener la coherencia de la base de datos. Los ideales que suelen seguirse son las formas normales Boyce-Codd (consulte el Capítulo 10).

Los ideales mencionados se sacrifican a veces en favor de una ejecución más rápida de las consultas y las transacciones más frecuentes. Este proceso de almacenamiento del diseño lógico de la base de datos (que puede estar en BCNF o 4NF) en una forma normal más débil, por ejemplo 2NF o 1NF, se denomina desnormalización. Normalmente, el diseñador añade a una tabla los atributos que son necesarios para responder consultas o producir informes, de modo que se evita una concatenación con otra tabla, que contiene el atributo recién añadido. Esto reintroduce una dependencia funcional parcial o una dependencia transitiva en la tabla, surgiendo por tanto problemas de redundancia asociados (consulte el Capítulo 10). Existe una contrapartida entre la actualización adicional necesaria para mantener la coherencia frente al esfuerzo necesario para efectuar una concatenación a fin de incorporar los atributos adicionales necesarios en el resultado. Por ejemplo, considere la siguiente relación:

ASSIGN (IdEmp, IdProy, NombreEmp, TítuloTrabjEmp, PorcentajeAsignado, NombreProy, IdDirectorProy, NomDirectorProy),

que corresponde exactamente a las cabeceras de un informe denominado *Lista de asignación de empleados*. Esta relación sólo está en 1NF debido a las siguientes dependencias funcionales:

IdProy $\rightarrow$ NombreProy, IdDirectorProy
IdDirectorProy $\rightarrow$ NomDirectorProy
e IdEmp $\rightarrow$ NombreEmp, TítuloTrabjEmp

Esta relación puede ser preferible al diseño en 2FN (y 3FN) y consiste en estas tres relaciones:

EMP (IdEmp, NombreEmp, TítuloTrabjEmp)
PROY (IdProy, NombreProy, IdDirectorProy)
EMP_PROY (IdEmp, IdProy, PorcentajeAsignado),

porque para producir el informe *Lista de asignación de empleados* (con todos los campos mostrados anteriormente en ASSIGN), el diseño de la última multi-relación requiere las siguientes concatenaciones:

$$\text{EMP\_PROY} * \text{EMP} * \text{PROY} \bowtie_{\text{IdDirectorProy} = \text{IdEmp}} \text{EMP}$$

En la expresión anterior, se necesita la concatenación final para obtener NomDirectorProy a partir de IdDirectorProy.

Otras formas de desnormalización consisten en almacenar tablas adicionales para mantener las dependencias funcionales originales que se pierden durante una descomposición BCNF. Por ejemplo, la Figura 10.13 mostraba la relación ENSEÑAR(Estudiante, Curso, Profesor) con las dependencias funcionales {{Estudiante, Curso} $\rightarrow$ Profesor, Profesor $\rightarrow$ Curso}. Una descomposición sin pérdida de ENSEÑAR en T1(Estudiante, Profesor) y T2(Profesor, Curso) *no* permite responder consultas del tipo "*¿a qué curso asistió el estudiante*

Pérez y qué impartió la profesora María?" sin tener que concatenar T1 y T2. Por consiguiente, una posible solución sería almacenar T1, T2 y ENSEÑAR, lo que reduce el diseño de BCNF a 3NF. Aquí, ENSEÑAR es una concatenación consistente en las otras dos tablas, lo que representa una redundancia extrema. Cualquier actualización de T1 y T2 debería aplicarse a ENSEÑAR. Una estrategia alternativa es considerar que T1 y T2 son tablas actualizables, mientras que ENSEÑAR se puede crear como una vista.

16.2 Visión general de la refinación de una base de datos en los sistemas relacionales

Una vez implantada una base de datos y ya en funcionamiento, el uso real de aplicaciones, transacciones, consultas y vistas revela factores y áreas problemáticas que durante el diseño físico inicial pudieron no tenerse en cuenta. Los factores de diseño físico mencionados en la Sección 16.1.1 pueden revisarse mediante la obtención de estadísticas de patrones de uso reales. La utilización de recursos, así como el procesamiento interno de DBMS (por ejemplo, la optimización de consultas), pueden monitorizarse para revelar cuellos de botella, como la confrontación de los mismos datos o dispositivos. Es posible hacer una mejor estimación de los volúmenes de actividad y de los tamaños de los datos. Por consiguiente, es necesario monitorizar y revisar constantemente el diseño físico de la base de datos. Los objetivos de la refinación son los siguientes:

- Conseguir que las aplicaciones se ejecuten más rápidamente.
- Reducir el tiempo de respuesta de las consultas y las transacciones.
- Mejorar el rendimiento global de las transacciones.

La línea divisoria entre el diseño físico y la refinación es muy fina. Las mismas decisiones de diseño que vimos en la Sección 16.1.2 se vuelven a visitar durante la fase de refinación, que es un ajuste continuado del diseño. A continuación ofrecemos una breve panorámica del proceso de refinación.[3] Como entrada de este proceso tenemos las estadísticas relacionadas con los factores mencionados en la Sección 16.1.1. En particular, los DBMSs pueden recopilar internamente las siguientes estadísticas:

- Tamaños de las tablas individuales.
- Número de valores distintos en una columna.
- Número de veces que una consulta o transacción en particular se emite y ejecuta en un intervalo de tiempo.
- Las veces que las diferentes fases requieren el procesamiento de consultas y transacciones (para un conjunto dado de consultas o transacciones).

Éstas y otras estadísticas crean un perfil del contenido y uso de la base de datos. De la monitorización de las actividades del sistema de bases de datos y de los procesos se puede obtener esta otra información:

- **Estadísticas de almacenamiento**. Datos sobre la asignación de almacenamiento en los espacios de tablas, espacios de índices y almacenes de búfer.
- **Estadísticas de rendimiento de E/S y de los dispositivos**. Actividad de lectura/escritura total (paginación) en el disco.
- **Estadísticas de procesamiento de consultas/transacciones**. Tiempos de ejecución de las consultas y las transacciones, optimización de los tiempos durante la optimización de las consultas.

[3] Los lectores interesados deben consultar Shasha (1992) y Shasha y Bonnett (2002) si desean una explicación detallada de la refinación.

- **Estadísticas relacionadas con el bloqueo/inicio de sesión**. Tasas de emisión de los diferentes tipos de bloqueos, tasas de rendimiento de transacciones, y actividad de registro.[4]
- **Estadísticas sobre los índices**. Número de niveles de un índice, número de páginas hoja no contiguas, etcétera.

Muchas de las estadísticas anteriores están relacionadas con las transacciones, el control de la concurrencia y la recuperación, que se explican en los Capítulos 17 a 19. La refinación de una base de datos puede enfrentarnos a estos problemas:

- Cómo evitar una contención de bloqueo excesiva, incrementándose en consecuencia la concurrencia entre las transacciones.
- Cómo minimizar el coste añadido del *logging* y de una descarga excesiva de datos.
- Cómo optimizar el tamaño del búfer y planificar los procesos.
- Cómo asignar recursos como los discos, la memoria RAM y los procesos a fin de conseguir el uso más eficaz.

La mayoría de estos problemas pueden resolverse configurando los parámetros físicos apropiados del DBMS, cambiando las configuraciones de dispositivos, modificando los parámetros del sistema operativo, y otras actividades parecidas. Las soluciones tienden a estar estrechamente relacionadas con sistemás específicos. Los DBAs normalmente están preparados para hacer frente a estos problemas de refinación en DBMSs específicos. Vamos a explicar la refinación de diversas decisiones de diseño de bases de datos.

16.2.1 Refinación de los índices

Es posible tener que revisar la decisión tomada inicialmente sobre los índices por las siguientes razones:

- Ciertas consultas pueden tardar demasiado en ejecutarse por carecer de un índice.
- No es posible utilizar ciertos índices.
- Algunos índices pueden sufrir demasiadas actualizaciones porque el índice se ha creado sobre un atributo que experimenta cambios frecuentes.

La mayoría de los DBMSs tienen un comando o utilidad de rastreo (*trace*) que el DBA puede utilizar para saber cómo se ha ejecutado una consulta (qué operaciones se han llevado a cabo y en qué orden, y qué estructuras de acceso secundario se han utilizado). Al analizar esta planificación de ejecución, es posible diagnosticar las causas de los problemas anteriores. Basándonos en el análisis pueden descartarse algunos índices y crearse otros.

El objetivo de la refinación es evaluar dinámicamente los requisitos, que a veces fluctúan estacionalmente o durante periodos de tiempo de un mes o una semana, así como reorganizar los índices a fin de conseguir el mejor rendimiento global. La eliminación y la creación de índices supone un coste añadido que puede justificarse si se busca la mejora del rendimiento. La actualización de una tabla queda generalmente suspendida mientras se elimina o crea un índice; hay que considerar esta pérdida de servicio. Además de eliminar o crear índices, y de cambiar de un índice no agrupado a otro agrupado y viceversa, la **reconstrucción del índice** puede mejorar el rendimiento. La mayoría de los RDBMSs utilizan árboles B^+ para el índice. Si hay muchas eliminaciones en la clave del índice, las páginas del índice pueden contener espacio desaprovechado, que puede reclamarse durante una operación de reconstrucción. De forma parecida, demasiadas inserciones pueden provocar desbordamientos en un índice agrupado que afecten al rendimiento. La reconstrucción de un índice agrupado equivale a reorganizar la tabla entera ordenada por esa clave.

[4] El lector puede ojear los Capítulos 17–19 si desea una explicación de estos términos.

Las opciones disponibles para la indexación y la forma en que se define, crea y reorganiza, varía de un sistema a otro. A modo de ilustración, considere los índices escaso y denso del Capítulo 14. Los índices escasos tienen un puntero de índice por cada página (bloque de disco) en el fichero de datos; los índices densos tienen un puntero de índice por cada registro. Sybase proporciona índices agrupados como índices escasos en forma de árboles B^+, mientras que INGRES ofrece índices agrupados escasos como ficheros ISAM, e índices agrupados densos como árboles B^+. En algunas versiones de Oracle y DB2, la opción de configurar un índice agrupado está limitada a un índice denso (con muchas más entradas de índice), y el DBA tiene que trabajar con esta limitación.

16.2.2 Refinación del diseño de la base de datos

En la Sección 16.1.2 vimos la necesidad de una posible desnormalización, que es una opción de mantener todas las tablas como relaciones BCNF. Si el diseño físico de una base de datos no satisface los objetivos esperados, podemos volver al diseño lógico de la base de datos, realizar ajustes en dicho diseño y volver a mapearlo a un nuevo conjunto de tablas e índices físicos.

Como explicamos, el diseño entero de la base de datos tiene que estar guiado por los requisitos de procesamiento y los requisitos de datos. Si los requisitos de procesamiento cambian dinámicamente, el diseño tiene que responder introduciendo cambios en el esquema conceptual si es necesario y reflejando esos cambios en el esquema lógico y en el diseño físico. Estos cambios pueden ser de la siguiente naturaleza:

- Las tablas existentes pueden concatenarse (desnormalizarse) porque con frecuencia se necesitan juntos ciertos atributos de dos o más tablas: esto reduce el nivel de normalización de BCNF a 3NF, 2NF o 1NF.[5]
- Para el conjunto de tablas dado, puede haber opciones de diseño alternativas; todas ellas logran 3NF o BCNF. En el Capítulo 11 ilustramos los diseños equivalentes alternativos. Uno puede reemplazarse por otro.
- Una relación de la forma R(K,A, B, C, D, ...), con K como un conjunto de atributos clave, que está en BCNF puede almacenarse en varias tablas que también están en BCNF [por ejemplo, R1(K, A, B), R2(K, C, D,), R3(K, ...)] replicando la clave K en cada tabla. Cada tabla agrupa conjuntos de atributos a los que se accede conjuntamente. Por ejemplo, la tabla EMPLEADO(Dni, Nombre, Tlf, Categ, Sueldo) puede dividirse en dos tablas: EMP1(Dni, Nombre, Tlf) y EMP2(Dni, Categ, Sueldo). Si la tabla original tiene muchas filas (por ejemplo, 100.000) y las consultas sobre los números de teléfono y los sueldos son completamente distintas y se producen con frecuencias muy diferentes, entonces esta separación de tablas puede funcionar mejor. Es lo que también se conoce como **división vertical**.
- El (los) atributo(s) de una tabla pueden repetirse en otra aunque esto introduzca redundancia y una anomalía potencial. Por ejemplo, NombrePieza puede duplicarse en las tablas dondequiera que aparezca el NumPieza (como *foreign key*), pero puede haber una tabla maestra denominada PIEZAS_MAESTRA(NumPieza, NombrePieza, ...) donde se garantice la actualización del nombre de la pieza.
- Así como la división vertical divide una tabla verticalmente en varias tablas, la **división horizontal** toma sectores horizontales de una tabla y los almacena como tablas distintas. Por ejemplo, los datos de ventas por producto se pueden separar en diez tablas basadas en las diez líneas de producto. Todas las tablas tienen el mismo conjunto de columnas (atributos), pero contienen un conjunto distinto de productos (tuplas). Si una consulta o una transacción se aplica a todos los datos de producto, puede que tenga que ejecutarse contra todas las tablas y que haya que combinar los resultados.

[5] 3NF y 2NF se dirigen a diferentes tipos de problemas de dependencia que son independientes entre sí; por tanto, el orden de normalización (o desnormalización) entre ellas es arbitrario.

En la práctica son comunes estos tipos de ajustes que se diseñan para satisfacer la gran cantidad de consultas y transacciones, sacrificando o no las formas normales.

16.2.3 Refinación de consultas

Ya hemos explicado cómo el rendimiento de una consulta depende de la selección adecuada de los índices y cómo éstos deben refinarse después de analizar las consultas que arrojan un rendimiento pobre, tarea para la que se utilizan los comandos del RDBMS que muestran la planificación de la ejecución de la consulta. Hay principalmente dos indicaciones que sugieren la necesidad de refinar una consulta:

1. Una consulta efectúa demasiados accesos al disco (por ejemplo, una consulta por coincidencia exacta explora una tabla entera).
2. La planificación de la consulta muestra que no se están utilizando los índices pertinentes.

Algunos casos típicos de situaciones que incitan a refinar una consulta son los siguientes:

1. Muchos optimizadores de consultas no utilizan los índices en presencia de expresiones aritméticas (por ejemplo, Sueldo/365 > 10.50), comparaciones numéricas de atributos de tamaños y precisiones diferentes (por ejemplo, Aqty = Bqty donde Aqty es del tipo INTEGER y Bqty es del tipo SMALLINTEGER), comparaciones con NULL (por ejemplo, FechaNac IS NULL), y comparaciones de subcadenas (por ejemplo, Apellido LIKE '%ez').
2. No se utilizan con frecuencia los índices en las consultas anidadas con IN; por ejemplo, la siguiente consulta:

```
SELECT   Dni  FROM EMPLEADO
WHERE    Dno  IN (SELECT NúmeroDpto FROM DEPARTAMENTO
                  WHERE DniDirector = '333445555' );
```

   no puede utilizar el índice por Dno en EMPLEADO, mientras que el uso de Dno = NúmeroDpto en la cláusula WHERE con la consulta de un solo bloque puede provocar que se utilice el índice.
3. Algunos DISTINCTs pueden ser redundantes y pueden evitarse sin tener que cambiar el resultado. DISTINCT a menudo provoca una operación de ordenación y debe evitarse siempre que sea posible.
4. El uso innecesario de tablas de resultado temporales pueden evitarse colapsando varias consultas en una sola *a menos que* se necesite la relación temporal para algún procesamiento intermedio.
5. En algunas situaciones que implican el uso de consultas correlativas, los temporales son útiles. Considere la siguiente consulta:

```
SELECT   Dni
FROM     EMPLEADO E
WHERE    Sueldo = SELECT MAX (Sueldo)
                  FROM EMPLEADO AS M
                  WHERE M.Dno = E.Dno;
```

   Esto tiene el peligro potencial de buscar en toda la tabla EMPLEADO M interior por *cada* tupla de la tabla EMPLEADO E exterior. Para que esto sea más eficaz, se puede dividir en dos consultas, de modo que la primera calcula el sueldo máximo de cada departamento:

```
SELECT   MAX (Sueldo) AS SueldoMax, Dno INTO TEMP
FROM     EMPLEADO
GROUP BY Dno;

SELECT   Dni
```

```
FROM      EMPLEADO, TEMP
WHERE     Sueldo = SueldoMax AND EMPLEADO.Dno = TEMP.Dno;
```

6. Si son posibles varias opciones para una condición de concatenación, elija una que utilice un índice agrupado y evite las que contienen comparaciones de cadena. Por ejemplo, asumiendo que el atributo Nombre es una clave candidata en EMPLEADO y ESTUDIANTE, es mejor utilizar EMPLEADO.Dni = ESTUDIANTE.Dni como condición de concatenación que EMPLEADO.Nombre = ESTUDIANTE.Nombre si Dni tiene un índice agrupado en una o en las dos tablas.
7. Una rareza de los optimizadores de consultas es que el orden de las tablas en la cláusula FROM puede afectar al procesamiento de la concatenación. Si es el caso, es posible tener que cambiar este orden para que se explore la más pequeña de las dos relaciones y se utilice la más grande con un índice adecuado.
8. Algunos optimizadores de consultas funcionan peor con las consultas anidadas que con las no anidadas. Hay cuatro tipos de consultas anidadas:
   - Subconsultas no correlativas con agregados en la consulta interior.
   - Subconsultas no correlativas sin agregados.
   - Subconsultas correlativas con agregados en la consulta interior.
   - Subconsultas correlativas sin agregados.

   De estos cuatro tipos, el primero normalmente no presenta problemas, puesto que la mayoría de los optimizadores evalúan la consulta interior una vez. Sin embargo, para una consulta del segundo tipo, como la del ejemplo del punto 2 anterior, la mayoría de los optimizadores de consultas no pueden utilizar un índice sobre Dno en EMPLEADO. Los mismos optimizadores lo podrán hacer si la consulta se escribe como una consulta no anidada. La transformación de las subconsultas correlativas puede implicar la configuración de tablas temporales. Mostrar ejemplos detallados queda fuera de los objetivos de este libro.[6]
9. Por último, muchas aplicaciones están basadas en vistas que definen los datos de interés para esas aplicaciones. A veces, una consulta puede plantearse directamente contra una tabla base, en lugar de pasar por una vista definida por una JOIN.

16.2.4 Directrices adicionales para la refinación de una consulta

En determinadas situaciones se aplican técnicas adicionales para mejorar las consultas:

1. Una consulta con varias condiciones de selección conectadas mediante OR no puede estar instando al optimizador de consultas a utilizar un índice. Dicha consulta puede dividirse y expresarse como una unión de consultas, cada una con una condición sobre un atributo que provoca la utilización de un índice. Por ejemplo,

```
SELECT    Nombre, Apellidos, Sueldo, Edad[7]
FROM      EMPLEADO
WHERE     Edad > 45 OR Sueldo < 50000;
```

puede ejecutarse con una exploración secuencial ofreciendo un rendimiento pobre. Con una división de este tipo:

---

[6] Si desea más información, consulte Shasha (1992).

[7] Hemos modificado el esquema y utilizado Edad en EMPLEADO en lugar de FechaNac.

```
SELECT    Nombre, Apellidos, Sueldo, Edad
FROM      EMPLEADO
WHERE     Edad > 45

UNION

SELECT    Nombre, Apellidos, Sueldo, Edad
FROM      EMPLEADO
WHERE     Sueldo < 50000;
```

pueden utilizarse los índices por Edad y Sueldo.

2. Para acelerar una consulta, se pueden intentar las siguientes transformaciones:
   - La condición NOT puede transformarse en una expresión positiva.
   - Los bloques SELECT incrustados con IN, = ALL, = ANY y = SOME pueden reemplazarse por concatenaciones.
   - Si se configura una concatenación de igualdad entre dos tablas, el predicado de rango (condición de selección) en el atributo de concatenación configurado en una tabla puede repetirse para la otra tabla.
3. Las condiciones WHERE pueden reescribirse para que utilicen los índices de varias columnas. Por ejemplo,

```
SELECT    NumRegión, TipoProd, Mes, Ventas
FROM      ESTADIS_VENTAS
WHERE     NumRegión = 3 AND ((TipoProd BETWEEN 1 AND 3) OR
          (TipoProd BETWEEN 8 AND 10));
```

puede utilizar sólo un índice por NumRegion y buscar por todas las páginas hoja del índice una coincidencia de TipoProd. En cambio, con

```
SELECT    NumRegión, TipoProd, Mes, Ventas
FROM      ESTADIS_VENTAS
WHERE     (NumRegión = 3 AND (TipoProd BETWEEN 1 AND 3))
          OR (NumRegión = 3 AND (TipoProd BETWEEN 8 AND 10));
```

puede utilizarse un índice compuesto (NumRegion, TipoProd) y funcionar de un modo mucho más eficaz.

En esta sección hemos cubierto la mayoría de las situaciones comunes en que la ineficacia de una consulta puede corregirse con alguna acción correctiva sencilla, como el uso de un temporal, evitar ciertos tipos de estructuras, o evitar el uso de vistas. El objetivo es utilizar, siempre que sea posible, los índices de un solo atributo o compuestos por varios atributos existentes. Esto evita la exploración completa de bloques de datos o de nodos hoja del índice. Los procesos redundantes, como la ordenación, deben evitarse a toda costa. Los problemas y los remedios dependerán de los trabajos de un optimizador de consultas dentro de un RDBMS. Los fabricantes de RDBMSs ofrecen a los administradores de bases de datos información detallada en forma de manuales.

16.3 Resumen

En este capítulo hemos visto los factores que afectan a las decisiones sobre el diseño físico de una base de datos, y ofrece directrices para elegir entre distintas alternativas de diseño físico. Como parte de la refinación

de una base de datos, hemos hablado de los cambios en el diseño lógico, las modificaciones de indexación y la introducción de cambios en las consultas. Lo explicado es sólo una muestra representativa de una gran cantidad de medidas y técnicas adoptadas en el diseño de grandes aplicaciones comerciales de DBMSs relacionales.

Preguntas de repaso

16.1. ¿Cuáles son los factores que influyen en el diseño físico de una base de datos?

16.2. Explique las decisiones tomadas durante el diseño físico de una base de datos.

16.3. Explique las directrices de los RDBMSs en cuanto al diseño físico de una base de datos.

16.4. Explique los tipos de modificaciones que pueden aplicarse al diseño lógico de una base de datos relacional.

16.5. ¿Bajo qué situaciones se utilizaría la desnormalización del esquema de una base de datos? Exponga algunos ejemplos de desnormalización.

Bibliografía seleccionada

Wiederhold (1986) abarca todas las fases del diseño de una base de datos, haciendo hincapié en el diseño físico. O'Neil (1994) ofrece una detallada explicación del diseño físico y de los problemas de transacción en referencia a los RDBMSs comerciales. Navathe y Kerschberg (1986) explica todas las fases del diseño de una base de datos y apunta el rol de los diccionarios de datos. Rozen y Shasha (1991) y Carlis y March (1984) presentan diferentes modelos para el problema del diseño físico de una base de datos. Shasha (1992) desarrolló unas directrices para la refinación de las bases de datos, que también aparecen, pero más elaboradas, en Shasha y Bonnett (2002).