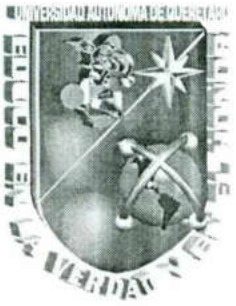


UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
FACULTAD DE INFORMÁTICA

OPTIMIZACIÓN DE LA BASE DE DATOS ORACLE 9i

TESINA

QUE PARA OBTENER EL TÍTULO DE
LICENCIADO EN INFORMÁTICA

PRESENTA
ANA LILIA MARTÍNEZ DOMÍNGUEZ

DIRIGIDA POR
I.S.C. JABEL RESÉNDIZ GONZÁLEZ

SANTIAGO DE QUERÉTARO, QRO. A 3 DE DICIEMBRE DEL 2004.

CARTA DE ACEPTACIÓN

Por este medio, se otorga constancia de aceptación de tesina para obtener el título de Licenciado en Informática, que presenta el pasante *ANA LILIA MARTÍNEZ DOMÍNGUEZ*, con el tema denominado *"OPTIMIZACIÓN DE LA BASE DE DATOS ORACLE 8i"*.

Este trabajo fue desarrollado como una investigación derivada del curso de titulación *"Administración de Base de Datos Oracle 8i"*, dando cumplimiento a uno de los requisitos contemplados en el artículo 34 del reglamento de titulación vigente, en lo referente a la opción de titulación por realización y aprobación de cursos de actualización.

Se extiende la presente para los fines legales a que haya lugar y para su inclusión en todos los ejemplares impresos de la tesina, a los seis días del mes de diciembre de dos mil cuatro.

ATENTAMENTE

I.S.C. JABEL RESÉNDIZ GONZÁLEZ
PROF. CURSO DE TITULACIÓN

ÍNDICE

AGRADECIMIENTOS

A mi Madre...
Porque ha forjado un futuro digno para mí con amor y esfuerzo.

A mi Familia...
Gracias por su apoyo incondicional, por creer en mí y darme ese entusiasmo para lograr cada objetivo, los quiero muchísimo.

A mis Amigos...
Aquellas personas que me han brindado momentos agradables, gracias por compartir alegrías, emociones, secretos, problemas, esfuerzos y amistad dentro y fuera de esta facultad.

A mis Profesores...
Con profundo respeto y admiración, porque nos han impulsado a que siempre seamos mejores seres humanos y profesionistas, por compartir su tiempo y espacio.
Ante todo gracias por su apoyo, estimulo y amistad.

GRACIAS

INTRODUCCIÒN

La optimización del rendimiento es parte de la existencia de toda aplicación de base de datos, y cuanto antes se resuelva, es más probable que se haga con éxito. La mayoría de los problemas de rendimiento no son síntomas aislados, sino resultado del diseño del sistema. Por lo tanto, los esfuerzos de optimización deben enfocarse hacia la identificación y resolución de los defectos subyacentes que provocan un rendimiento inaceptable.

La optimización es el paso final de un proceso de cuatro etapas: la planificación, la implementación y la supervisión deben precederlo. Si realiza la optimización sin ningún propósito concreto, no estará siguiendo el ciclo completo de actividad y, probablemente, nunca resolverá los problemas subyacentes que provocaron el problema de rendimiento.

Esta tesina se ocupa de las actividades relacionadas con el proceso de optimización de los objetos de base de datos. En los siguientes capítulos veremos actividades de optimización para las siguientes áreas:

- *Diseño de aplicaciones*. Nada de lo que haga como DBA tendrá un impacto tan importante en el rendimiento del sistema como el diseño de la aplicación. Al diseñar una aplicación, puede dar varios pasos para hacer uso de la tecnología disponible de una manera eficaz y adecuada, tal y como se describe en este capítulo.

- *SQL*. Como con el diseño de aplicaciones, la optimización del código SQL parece haber sido eliminada hace tiempo de las labores de un DBA. No obstante, los DBA deben participar en la revisión del código SQL que se escribe como parte de la aplicación.

- *Utilización de la memoria.* Puede utilizar las utilidades STATSPACK y las consultas en tablas del diccionario de datos para identificar áreas problemáticas en la asignación de memoria de la base de datos.

- *Almacenamiento de datos.* El modo en que la base de datos *almacena* datos tiene también efecto sobre el rendimiento de las consultas. Si los datos están fragmentados entre varias extensiones, entonces resolver una consulta puede provocar que la base de datos busque las filas asociadas en varias ubicaciones físicas.

- *Tratamiento de datos.* Hay varias tareas de manipulación de datos (que implican normalmente manipulación de grandes cantidades de datos) que pueden involucrar al DBA. Tiene varias opciones al cargar y borrar grandes volúmenes de datos, como se describe en este capítulo.

- *Almacenamiento físico y lógico.* Las operaciones de E/S física asociadas a las bases de datos deben distribuirse por igual y gestionarse correctamente, asimismo, su asociación lógica.

- *Tráfico de red.* Como los DBA suelen tener poco control sobre la gestión de la red, es importante utilizar las funciones que ofrece la base de datos para reducir el número de paquetes de red necesarios para que los datos sean entregados.

CAPÌTULO 1
OPTIMIZACIÓN DEL DISEÑO DE APLICACIONES

1.1 Diseño efectivo de tablas

Ninguna aplicación de cierta importancia funcionará en la Tercera Forma Normal [Koch, 1998].

Sin tener en cuenta lo bien diseñada que esté su base de datos, un diseño de tablas malo dará lugar a un rendimiento también malo. No solamente eso, sino que un seguimiento demasiado rígido de las técnicas formales de diseño de tablas relaciónales dará lugar a un mal rendimiento. Esto se debe al hecho de que, aunque los diseños de tablas estrictamente relaciónales (de los que se dice que están en la *tercera forma normal)* son desde el punto de vista lógico deseables, no lo son desde el punto de vista físico.

El problema de estos diseños es que, aunque reflejan con precisión la forma en la que los datos de una aplicación están relacionados con otros datos, no reflejan la manera habitual que emplearán los usuarios para acceder a dichos datos. Una vez evaluados los requisitos de acceso a los datos por parte de los usuarios, un diseño de tablas estrictamente relacional resultará inviable para muchas consultas de gran tamaño. Normalmente, los primeros problemas surgirán en consultas que devuelven un gran número de columnas. Estas columnas se suelen distribuir entre varias tablas, obligando a realizar una combinación (join) durante la consulta. Si una de las tablas combinadas es de gran tamaño, entonces el rendimiento de la consulta completa puede verse afectado negativamente.

Al diseñar las tablas para una aplicación, los desarrolladores deben por tanto considerar la posibilidad de desnormalizar los datos (por ejemplo, crear pequeñas tablas de resumen partiendo de tablas grandes y estáticas). ¿Se pueden derivar dinámicamente esos datos de las tablas estáticas de gran tamaño a voluntad? Por supuesto. Pero si los usuarios lo piden con frecuencia, y los datos, básicamente, no han cambiado, entonces tiene sentido almacenar periódicamente dichos datos en el formato en el que los pidan los usuarios.

Por ejemplo, algunas aplicaciones almacenan datos históricos y actuales en la misma tabla. Cada registro puede tener una columna de indicación de tiempo, de manera que el registro actual de un conjunto de resultados sea el que tenga la indicación de tiempo más reciente. Cada vez que un usuario consulte la tabla para obtener el registro más reciente, el usuario tendrá que realizar una subconsulta (como **where timestamp_col = (select(max(timestamp_col) from table where *emp_name='some name')*.** Si dos tablas de este tipo están combinadas, habrá dos subconsultas. En una base de datos pequeña, esto puede no suponer un problema de rendimiento pero, a medida que aumente el número de tablas y filas, aparecerán los problemas de rendimiento. Separar los datos históricos de los actuales o almacenarlos en una tabla distinta supondrá más trabajo para los DBA y desarrolladores pero, a largo plazo, mejorará el rendimiento de la aplicación.

El diseño de tablas basado en los usuarios, en lugar del diseño de tablas basado en la teoría, producirá un sistema que satisfará mejor los requisitos del usuario. Las opciones de diseño incluyen separar una sola tabla en varias, y lo contrario (combinar varias tablas en una). El énfasis debe ponerse en proporcionar a los usuarios la forma de acceso más directa posible a los datos que deseen en el formato que deseen.

1.2 Distribución de los requisitos del procesador

Uno de los factores que puede estar limitando el rendimiento de su base de datos podría ser la disponibilidad de recursos de la CPU. A falta de poder proporcionar capacidad de proceso adicional para los servidores disponibles, tiene varias opciones para gestionar los recursos disponibles:

- La carga de trabajo del procesador debe planificarse: programe las consultas de larga duración o los programas de actualización para las horas de menos intensidad de trabajo. En lugar de ejecutarlas con baja prioridad mientras los usuarios conectados están realizando transacciones, ejecútelas con la prioridad normal en el momento adecuado. Mantener un nivel de prioridad normal al mismo tiempo que se planifican las tareas correctamente minimizará los posibles conflictos de bloqueos, de operaciones de anulación y de utilización del procesador.

- Aproveche la oportunidad de trasladar físicamente las necesidades de procesamiento de un servidor a otro. Siempre que sea posible, aísle el servidor de base de datos de los requisitos de procesamiento de la aplicación.

- Tenga en cuenta la utilización de la tecnología RAC (Real Application Cluster) de Oracle, antes conocida como Oracle Parallel Server (servidor paralelo de Oracle), para distribuir los requisitos de acceso a una misma base de datos entre varias instancias.

- Utilice las funciones de gestión de recursos de base de datos introducidas en Oracle8i. Puede utilizar el Administrador de recursos de base de datos para establecer planes de asignación de recursos y grupos de consumidores de recursos. Puede utilizar las posibilidades que le ofrece Oracle para cambiar las asignaciones de recursos disponibles para los grupos de consumidores.

- Utilice la opción de consulta en paralelo o PQO (Parallel Query Option) para distribuir los requisitos de proceso de instrucciones SQL a lo largo de varios procesadores. El paralelismo puede utilizarse con casi cualquier comando SQL, incluidos **select, create table as select, create index, recover** y las opciones de carga de SQL*Loader Direct Path.

El grado en que una transacción se paraleliza depende del grado definido de paralelismo para la transacción. Cada tabla puede tener un grado definido de paralelismo y una consulta puede anular el grado predeterminado de paralelismo utilizando la indicación PARALLEL. Oracle tiene en cuenta el número de procesadores disponibles en el servidor y el número de discos en los que se almacenan los datos de la tabla para determinar el grado predeterminado de paralelismo.

El máximo paralelismo disponible se establece en el nivel de la instancia. El parámetro de inicialización PARALLEL_MAX_SERVERS indica el número máximo de procesos de servidor de consulta paralela que pueden utilizar, en un momento determinado,

todos los procesos de la base de datos. Por ejemplo, si se configura PARALLEL_MAX_SERVERS con el valor 32 para su instancia y ejecuta una consulta que utiliza 30 procesos de servidor de consulta paralela para sus operaciones de consulta y ordenación, entonces solamente dos procesos de servidor de consulta paralela están disponibles para el resto de los usuarios de la base de datos. Así, debe gestionar con cuidado el paralelismo que desee autorizar a sus consultas y operaciones por lotes. Puede utilizar el parámetro de inicialización PARALLEL_ADAPTIVE_MULTI_USER para limitar el paralelismo de las operaciones en un entorno multiusuario. La característica PARALLEL_ADAPTIVE_MULTI_USER se activa automáticamente si se configura el valor del parámetro de inicialización PARALLEL_AUTOMATIC_TUNING con el valor TRUE. A partir de Oracle8i se puede limitar el paralelismo disponible para los grupos de consumidores de recursos definidos dentro de su base de datos

Para cada tabla puede configurar un grado predeterminado de paralelismo mediante la cláusula **parallel** de los comandos **create table y alter table.** El grado de paralelismo le indica a Oracle cuántos procesos de servidor de consulta paralela debe intentar utilizar para cada parte de la operación. Por ejemplo, si una consulta que realizó las operaciones de exploración de tabla y ordenación de datos tenía un grado de paralelismo de 5, entonces se podrían estar utilizando 11 procesos de servidor de consulta paralela (5 para explorar, 5 para ordenar y 1 para coordinar los otros 10 procesos). Puede especificar además un grado de paralelismo para un índice al crearlo, mediante la cláusula **parallel** del comando **create index.**

El número mínimo de procesos de servidor de consulta en paralelo iniciados se configura mediante el parámetro de inicialización PARALLEL_MIN_SERVERS. En general, debe configurar este parámetro con un valor muy bajo (menor que 5). Darle a este parámetro un valor bajo obligará a Oracle a iniciar repetidamente nuevos procesos de servidor de consulta en paralelo, pero disminuirá enormemente la cantidad de memoria empleada por procesos de servidor de consulta en paralelo inactivos. Si configura un valor alto para PARALLEL_MIN_SERVERS, entonces podría tener, con cierta frecuencia, procesos de servidor de consulta en paralelo inactivos en su servidor, siguiendo en posesión de la memoria que hubieran adquirido previamente pero sin realizar función alguna. Puede utilizar esta función para acelerar consultas en paralelo que espera terminar en pocos segundos. Puede configurar un parámetro de tiempo de inactividad (oculto en Oracle9i) que indique a Oracle los minutos que puede estar inactivo un proceso de servidor de consulta en paralelo antes de que la base de datos lo dé por terminado. Paralelizar las operaciones distribuye las necesidades de procesamiento entre varios procesadores; no obstante, debe utilizar estas funciones con cuidado. Si utiliza un grado de paralelismo de 5 para una consulta de gran tamaño, entonces tendrá cinco procesos distintos accediendo a los datos. Si tiene tantos procesos accediendo a los datos, entonces puede crear contienda en los accesos a los discos en los que los datos están almacenados, empeorando el rendimiento. Cuando utilice la PQO, debe aplicarla de manera selectiva a las tablas cuyos datos estén bien distribuidos por entre muchos dispositivos físicos. Además debe evitar utilizarla para todas las tablas; como hemos observado anteriormente, una sola consulta puede utilizar todos los procesos de servidor de consulta en paralelo disponibles, impidiendo las operaciones en paralelo para el resto de las transacciones de la base de datos.

Si la tabla está particionada, entonces la operaciones DML en paralelo estarán limitadas en las versiones anteriores a Oracle9i. En Oracle8.O y Oracle8i, una tabla particionada solamente puede utilizar un proceso de servidor de consulta en paralelo por cada partición cuando ejecuta operaciones DML en paralelo. Oracle9i ofrece paralelismo entre particiones, permitiendo que sus consultas sean servidas por varios procesos de servidor de consulta paralela por cada partición.

1.3 Diseño efectivo de aplicaciones

Además de los temas de diseño de aplicaciones descritos anteriormente, hay varias directrices generales para las aplicaciones Oracle.

En primer lugar, las aplicaciones deben minimizar el número de veces que piden datos de la base de datos. Las opciones incluyen el uso de secuencias, bloques PL/SQL y desnormalización de tablas. Se pueden utilizar objetos de base de datos distribuidos tales como instantáneas y (desde Oracle 8i) vistas materializadas que faciliten la reducción del número de veces que se consulta una base de datos.

Es posible que código SQL medianamente ineficaz afecte negativamente al rendimiento de su base de datos si se ejecuta con la suficiente frecuencia.

En segundo lugar, los distintos usuarios de la misma aplicación deben consultar la base de datos de una manera muy similar. Unas rutas de acceso coherentes aumentan la probabilidad de que las peticiones se resuelvan con información que ya está disponible en la SGA. La compartición de datos incluye no solamente las tablas y filas recuperadas, sino también las consultas que se utilizan. Si las consultas son idénticas, entonces la versión analizada de una consulta puede ya existir en el área SQL compartida (véase V$SGASTAT), reduciendo el tiempo necesario para procesar la consulta. En Oracle9i, las nuevas mejoras de compartición de cursores efectuadas en el optimizador aumentan la probabilidad de reutilización de instrucciones dentro del área compartida, aunque la aplicación ha de diseñarse teniendo en mente la reutilización de instrucciones.

Tercero, debe restringir el uso de SQL dinámico. El SQL dinámico, que emplea el paquete DBMS_SQL, siempre se procesa de nuevo incluso aunque exista una consulta idéntica en el área compartida. El SQL dinámico es una característica útil, pero no debe utilizarse como la forma de acceso más frecuente a la base de datos de una aplicación.

Los procedimientos almacenados están disponibles para su uso durante el desarrollo de aplicaciones. Cuando se utilizan, el mismo código puede ejecutarse varias veces, aprovechando así el área compartida. También puede compilar manualmente los procedimientos, las funciones y los paquetes para evitar la compilación en tiempo real. Cuando se crea un procedimiento, Oracle lo compila automáticamente. Si, con posterioridad, el procedimiento deja de ser válido, la base de datos debe recompilarlo antes de ejecutarlo. Para evitar incurrir en este coste de compilación en el momento de la ejecución, utilice el comando **alter procedure** mostrado en el siguiente listado:

```
alter procedure MY_RAISE compile;
```

Puede ver el texto SQL para todos los procedimientos de una base de datos mediante la columna Text de la vista DBA_SOURCE. La vista USER_SOURCE mostrará los

procedimientos que son propiedad del usuario que realiza la consulta. También se puede acceder al texto para los paquetes, las funciones y los cuerpos de paquetes mediante las vistas DBA_SOURCE y USER_SOURCE, que a su vez hacen referencia a una tabla llamada SYS.SOURCE$. Dado que SYS.SOURCE$ es parte del diccionario de datos, esto significa que el código de los procedimientos se almacena en el espacio de tablas SYSTEM. Así, si utiliza procedimientos almacenados y paquetes, debe asegurarse de asignar más espacio al espacio de tablas SYSTEM (aumentando su tamaño a, posiblemente, más del doble de su tamaño original).

Las primeras dos directrices de diseño tratadas (limitar el número de accesos de usuario y coordinar sus peticiones) requieren que el desarrollador de la aplicación sepa todo lo posible sobre cómo se utilizan los datos y las rutas de acceso implicadas. Por esta razón, es fundamental que los usuarios estén tan implicados en el diseño de las aplicaciones tanto como lo están en el diseño de tablas. Si los usuarios pasan muchas horas haciendo dibujos de tablas con los analistas de modelos de datos y poco tiempo con los desarrolladores de las aplicaciones hablando sobre las formas de acceso a los datos, es probable que la aplicación no satisfaga las necesidades de los usuarios.

CAPÌTULO 2
OPTIMÍZACÍÓN DEL CÓDIGO SQL

Una aplicación bien diseñada puede seguir experimentando problemas de rendimiento si el código SQL que utiliza no está debidamente optimizado. El diseño de aplicaciones y los problemas con el código SQL provocan la *mayoría* de los problemas de rendimiento en bases de datos correctamente diseñadas.

La clave para optimizar código SQL es minimizar la ruta de búsqueda que utiliza la base de datos para encontrar los datos. En la mayoría de las tablas Oracle, cada fila tiene un identificador asociado. El identificador de fila contiene información sobre la ubicación física de la fila (su archivo, el bloque contenido en ese archivo y la fila contenida en el bloque de la base de datos).

Cuando se ejecuta una consulta sin cláusula **where,** la base de datos realizará normalmente una exploración completa de tabla, leyendo todos los bloques de la tabla. Durante una exploración completa de tabla, la base de datos localiza el primer bloque de la tabla y lee después secuencialmente el resto de los bloques de la misma. Para tablas de gran tamaño, las exploraciones completas de tabla pueden emplear mucho tiempo. Ejecutar la consulta en paralelo reduce ese tiempo.

Cuando se consultan determinadas filas, la base de datos puede utilizar un índice para acelerar la recuperación de las filas deseadas. Un índice organiza los valores lógicos en una tabla asociándolos a sus identificadores de fila, que a su vez los organizan en una determinada ubicación física. Los índices pueden ser únicos (en cuyo caso no hay más de una aparición por cada valor) o no únicos. Los índices solamente almacenan identificadores de fila para los valores de columna de las columnas indexadas. No se almacena nada en el índice si la columna tiene el valor NULL para una determinada fila.

Se puede crear un índice de varias columnas. Esto se llama *índice concatenado, y* se utilizará si se emplea su columna principal en la cláusula **where** de la consulta. En Oracle9i, el optimizador puede también utilizar un enfoque de *exploración por omisión,* en el que se emplea un índice concatenado incluso si su columna principal no está en la cláusula **where** de la consulta. Los métodos de exploración por omisión no son tan eficaces como los que utilizan la columna principal del índice.

Los índices deben diseñarse específicamente en función de la ruta de acceso necesaria. Tenga en cuenta el caso de un índice concatenado de tres columnas. Como se muestra en el siguiente listado, se crea en las columnas City, State y Zip de la tabla EMPLOYEE:

```
Create index CITY_ST_ZIP_NDX
on EMPLOYEE(City, State, Zip)
tablespace INDEXES;
```

Si se ejecuta una consulta de la forma:

```
select * from EMPLOYEE where State='NJ';
```

entonces el índice *no* se utilizará en Oracle8i, porque su columna *principal* (City) no se utiliza en la cláusula **where.** En Oracle9i, la función de exploración por omisión del

optimizador permite utilizar el CITY_ST_ZIP_NDX para satisfacer esta consulta. Si los usuarios van a ejecutar con frecuencia este tipo de consulta, entonces se debería cambiar el orden de las columnas del índice, situando a State en primer lugar para reflejar el patrón de uso real.

Es fundamental que los datos de la tabla estén tan ordenados como sea posible. Si los usuarios ejecutan con frecuencia consultas por *rangos* (seleccionando los valores que estén dentro de un determinado rango), entonces el tener los datos ordenados requerirá la lectura de menos bloques de datos mientras se resuelve la consulta, mejorando así el rendimiento. Las entradas ordenadas en el índice apuntarán a un juego de bloques adyacentes de la tabla en lugar de a bloques que estén desperdigados a lo largo del archivo o archivos de datos.

Por ejemplo, una consulta por rangos del tipo mostrado en este listado:

```
select  *
from EMPLOYEE
where Empno between 1 and 100;
```

requerirá la lectura de menos bloques de datos si los registros físicos de la tabla EMPLOYEE están ordenados por la columna Empno. Esto mejoraría el rendimiento de la consulta. Para garantizar que las filas están correctamente ordenadas en la tabla, extraiga los registros a un archivo sin formato, ordene los registros del archivo, y después borre los antiguos registros y cárguelos de nuevo desde el archivo ordenado.

Como alternativa a la extracción de datos a un archivo sin formato, puede utilizar los procedimientos de ordenación internos de Oracle para ordenar sus datos. Lo ideal es reordenar las filas de una tabla creando una segunda tabla mediante el comando **create table as select.** Antes de Oracle9i, los comandos **create table as select e insert as select** no permiten especificar una cláusula **order by.**

Para evitar esta limitación en versiones anteriores a Oracle8i, cree una vista en la tabla base. Las vistas sí soportan la cláusula **order by,** de modo que se puede crear una vista y hacer una selección a partir de ella para llenar una tabla ordenada.

```
Create or replace view EMPLOYEE_VIEW as
select *  from COMPANY
order by Empno;
```

Ahora puede crear una tabla seleccionando desde EMPLOYEE_VIEW; el efecto será la creación de una copia duplicada de EMPLOYEE, con las filas adecuadamente ordenadas.

```
Create table EMPLOYEE_ORDERED
as select * from EMPLOYEE_VIEW;
```

En el ejemplo anterior, los datos estaban ordenados por el valor Empno. Pero, en lugar de ordenar por valor, es posible que a menudo necesite ordenar los datos por una columna de atributo, como la columna Name. Si los datos están ordenados para soportar las consultas por rangos más utilizadas y están almacenados de manera compacta dentro de

cada bloque, entonces puede minimizar el número de bloques leídos durante cada consulta y mejorar así el rendimiento de sus consultas.

2.1 Impacto del orden en la velocidad de carga

Los índices afectan al rendimiento tanto de las consultas como de las cargas de datos. Durante las inserciones, el orden de las filas tiene un impacto importante sobre el rendimiento de la carga. Incluso en entornos muy indexados, ordenar correctamente las filas antes de **insertar** puede mejorar el rendimiento en un 50 por 100.

A medida que un índice crece, Oracle asigna nuevos bloques. Si se añade una nueva entrada de índice más allá de la última entrada, la nueva se añadirá al último bloque del índice. Si la nueva entrada provoca que Oracle sobrepase el espacio disponible en ese bloque, la entrada se trasladará a un nuevo bloque, dejando intactas todas las entradas originales. Esta asignación de bloques produce muy poco impacto en el rendimiento.

Si las filas insertadas no están ordenadas, entonces las nuevas entradas de índice se escribirán en bloques de nodo de índice existentes. Si no hay más espacio en el nodo donde se añade el nuevo valor, el nodo se dividirá en dos. El 50 por 100 de las entradas de índice se quedará en el nodo original y el otro 50 se trasladará a un nuevo nodo. Como resultado de ello, el rendimiento sufre durante las cargas (debido a la actividad adicional de gestión del espacio) y durante las consultas (ya que el índice contiene más espacio no utilizado, lo que requiere la lectura de más bloques).

En las pruebas de laboratorio efectuadas, se creó un índice sobre la columna Name de la tabla EMPLOYEE. Cuando se insertaron las filas con los valores Name muy desordenados, el número de bloques utilizados por el índice aumentó en el 50 por 100 y la velocidad de carga disminuyó en más del 50 por 100. Si tiene varios índices en su tabla, ordene las filas durante las inserciones según el índice que se utilice con más frecuencia durante las consultas de gran tamaño si desea ajustar las operaciones de consulta por rangos; para ajustar las operaciones **insert,** ordene por el índice más complejo.

Se produce una disminución notable en el rendimiento de las operaciones de carga cuando un índice aumenta su número de niveles internos. El momento en el que se produce el aumento depende del tamaño de bloque de la base de datos y de la longitud de los valores clave del índice. Para ver el número de niveles, analice un índice y seleccione para dicho índice el valor de la columna Blevel en la tabla DBA_INDEXES. Para un mejor rendimiento de carga, mantenga el valor de Blevel tan bajo como sea posible para todos los índices.

Debido al modo en que Oracle gestiona sus índices internamente, las velocidades de carga se verán afectadas cada vez que se añade un nuevo índice (ya que no es probable que las filas se inserten con los datos correctamente ordenados para varias columnas). Desde el punto de vista estricto de la velocidad de carga, es mejor tener unos pocos índices multicolumna que muchos índices de una única columna.

2.2 Opciones de indexación adicionales

Si los datos no son muy selectivos, entonces puede tener en cuenta el uso de índices de tipo mapa de bits. Los índices de tipo mapa de bits son más eficaces para efectuar

consultas en juegos de datos de gran tamaño, estáticos y con pocos valores distintos. Puede crear índices de tipo mapa de bits e índices normales (de árbol binario) en la misma tabla, y Oracle realizará las conversiones de índices necesarias de manera dinámica durante el proceso de las consultas. No se puede tener un índice normal y uno de tipo mapa de bits al mismo tiempo en la misma columna dentro de una tabla.

Si dos tablas se consultan conjuntamente con cierta frecuencia, entonces un clus-ter de tablas pueden ser eficaces en la mejora del rendimiento. Los clusters almacenan filas de varias tablas en los mismos bloques de datos físicos, basándose en sus valores lógicos (la clave de cluster).

Las consultas en las que el valor de una columna se compara con un valor exacto (en lugar de con un rango de valores) se llaman consultas de *equivalencia.* Un *cluster hash* almacena una fila en una ubicación específica basándose en su valor en la columna de clave de cluster. Cada vez que se inserta una fila, se utiliza su valor de clave de cluster para determinar en qué bloque se debe almacenar; esta misma lógica puede utilizarse durante las consultas para localizar rápidamente bloques de datos necesarios para la recuperación. Los clusters hash se han diseñado para mejorar el rendimiento de las consultas de equivalencia; no serán tan útiles para mejorar el rendimiento de las consultas por rangos tratadas anteriormente.

Los *índices inversos* proporcionan otra solución de optimización para las consultas de equivalencia. En un índice inverso, los bytes del índice se almacenan en orden inverso. En un índice tradicional, dos valores consecutivos se almacenan uno al lado del otro. En un índice inverso, los valores consecutivos no se almacenan uno al lado de otro. Por ejemplo, los valores 2004 y 2005 se almacenan como 4002 y 5002, respectivamente, en un índice inverso. Aunque no sean adecuados para exploraciones por rangos, los índices inversos pueden reducir la contienda en bloques de índices si se realizan muchas consultas de equivalencia. No se puede crear un índice de tipo mapa de bits inverso.

A partir de Oracle8i se pueden crear *índices funcionales.* Antes de Oracle8i cualquier consulta que aplicaba una función sobre una columna no podía utilizar el índice de esa columna. Así, esta consulta no podía utilizar un índice en la columna Name:

```
select * from EMPLOYEE
where UPPER(Name) = 'JONES';
```

pero ésta, sí:

```
select * from EMPLOYEE
where Name = 'JONES' ;
```

porque la segunda consulta no aplica ninguna función sobre la columna Name. Desde Oracle8i, se pueden crear índices que permiten que los accesos basados en funciones sean soportados por los accesos de índice. En lugar de crear un índice sobre la columna Name, se puede crear un índice sobre la expresión UPPER (Name), como se muestra en este listado:

```
create index EMP_UPPER_NAME on
EMPLOYEE(UPPER(Name));
```

Aunque los índices funcionales pueden ser útiles, asegúrese de tener en cuenta los siguientes puntos cuando los cree:

- ¿Puede limitar las funciones que se van a aplicar sobre la columna? Si es así, ¿puede conseguir que no se aplique ninguna función sobre la columna?
- ¿Tiene suficiente espacio de almacenamiento para los índices adicionales?
- Cuando elimine la tabla, estará eliminando más índices (y por lo tanto más extensiones) que antes. ¿Cómo afectará eso al tiempo necesario para eliminar la tabla?

Los índices basados en funciones son útiles, pero debe implementarlos con moderación. Cuantos más índices cree en una tabla, más tiempo necesitarán todas las operaciones **insert, update y delete.** Para crear un índice basado en una función debe tener el privilegio de sistema QUERY REWRITE, y los siguientes parámetros deben configurarse en el archivo de parámetros de inicialización:

```
QUERY_REWRITE_ENABLED=TRUE
```

2.3 Generación de planes de ejecución

¿Cómo puede determinar la ruta de acceso que empleará la base de datos para realizar una consulta? Esta información puede visualizarse mediante el comando **explain plan.** Este comando evaluará la ruta de ejecución para una consulta y situará su resultado en una tabla (llamada PLAN_TABLE) de la base de datos. En el siguiente listado se muestra un comando **explain plan** de ejemplo:

```
explain plan
set Statement_Id =    'TEST'
for
select * from EMPLOYEE
where City > 'Y%' ;
```

La primera línea de este comando le indica a la base de datos que explique su plan de ejecución para la consulta sin ejecutarla realmente. La segunda línea etiqueta los registros de esta consulta de la tabla PLAN_TABLE con un Statement_Id igual a TEST. Después de la palabra clave **for,** se prueba la consulta que se desea analizar.

La cuenta que esté ejecutando este comando debe tener una tabla llamada PLAN_TABLE en su esquema. Oracle proporciona los comandos **create table** necesarios para crear esta tabla. El archivo, llamado utlxplan.sql, suele estar situado en el subdirectorio

/rdbms/admin del directorio principal del software Oracle. Los usuarios pueden ejecutar este script para crear la tabla en sus esquemas.

Debe borrar y volver a crear la tabla de planes, PLAN_TABLE, después de cada actualización de Oracle, ya que los scripts de actualización pueden añadir nuevas columnas.

Consulte la tabla de planes utilizando la consulta del siguiente listado. Los registros de dicha tabla están relacionados unos con otros, de modo que puede utilizarse la cláusula **connect by** de la instrucción **select** para evaluar la jerarquía.

```
select ID ID_plus_exp,
Parent_ID parent_id_plus_exp,
LPAD(' ',2*(level-1))|| /* Sangrado para el nivel */
Operation|| /* La operación */
DECODE(other_tag,null,'','*')||/*mostrará un '*' si es
paralelo */
DECODE(options,null,'','('||options||') ')||/* muestra las
opciones */
DECODE(object_name,null,'',' of ''||object_name||'''') ||
DECODE(object_type,null,'', '||object_type||')') ||
DECODE(id,0,decode(optimizer,null,'','optimizer='||optimizer))
||
DECODE(cost,null,'','(cost='||cost||/* muestra info. de coste
*/
DECODE(cardinality,null,'','card='||cardinality) || /* calidad
de cardinal */
DECODE(bytes,null,'',' bytes='||bytes) ||')') plan_plus_exp,
object_node object_node_plus_exp /* info. de paralelo y
remoto */
from PLAN_TABLE
start with ID=0 and Statement_ID='TEST'
connect by prior ID=Parent_ID and Statement_ID='TEST'
order by ID,Position;
```

Esta consulta informará sobre los tipos de operaciones que la base de datos debe realizar para resolver la consulta. Las primeras tres columnas del resultado de ejemplo se muestran en el siguiente listado:

```
i    p    PLAN_PLUS_EXP
---  ---- ----------------------------------------
0         SELECT STATEMENT optimizer=CHOOSE
1    0    TABLE ACCESS (BY INDEX ROWID) of 'EMPLOYEE'
2    1    INDEX (RANGE SCAN) of 'CITY_ST_ZIP_NDX' NON-UNIQUE)
```

Para leer el plan de ejecución, lea el orden de operaciones desde dentro hacia afuera de la jerarquía hasta que llegue a un juego de operaciones situado al mismo nivel de sangrado; después lea de arriba abajo. En este ejemplo, no hay operaciones al mismo nivel de sangrado, por lo tanto lea el orden de operaciones de dentro a fuera. La primera

operación es la exploración por rangos del índice, seguida del acceso a la tabla; la operación SELECT STATEMENT le muestra el resultado al usuario. Cada operación tiene un valor ID (la primera columna) y un valor ID padre. En planes de ejecución más complejos, puede necesitar utilizar los valores ID padres para determinar el orden de las operaciones.

Este plan muestra que los datos que se devuelven al usuario vienen a través de un acceso a la tabla mediante el identificador de fila. Los identificadores de fila son proporcionados por una exploración por rangos del índice, utilizando el índice CITY_ST_ ZIP_NDX descrito anteriormente.

Desde Oracle8i, se pueden utilizar esquemas almacenados para guardar la ruta de ejecución para una consulta.

Puede utilizar el comando **set autotrace on** de SQL*Plus para generar automáticamente el resultado de **explain plan** y la información de traza asociada para cada consulta que ejecute. La información de salida generada por el comando autotrace no se mostrará hasta después de completarse la consulta, mientras que la información de salida de **explain plan** se genera sin llegar a ejecutar el comando. Para activar la información de salida generada por el comando autotrace, una tabla PLAN_TABLE debe haber sido creada en el esquema en el que se empleará la utilidad de traza automática, o debe haber sido creada en el esquema SYSTEM y haberse concedido el acceso al esquema que empleará la utilidad de traza automática. El script plustrce.sql crea el rol PLUSTRACE. Ubicado en el subdirectorio sqlplus/admin del directorio principal del software de Oracle, el archivo se ejecuta automáticamente como parte de los scripts de creación de bases de datos de Oracle9i. Los usuarios deben tener el rol PLUSTRACE activado antes de ejecutar set **autotrace on.**

Cuando evalúe la salida del comando **explain plan,** debe asegurarse de que la consulta utilice los índices más selectivos (es decir, los índices casi más próximos a comportarse como índices únicos). Si se utiliza un índice no selectivo, puede estar obligando a la base de datos a realizar lecturas innecesarias para resolver la consulta, debe centrar sus esfuerzos de Optimización en asegurarse de que las instrucciones SQL que más recursos utilizan estén empleando los índices más selectivos posibles.

Si se analizan las tablas y los índices, tanto el comando **set autotrace on** como la consulta **explain plan** mostrados en el listado anterior indicarán un «coste>> por cada paso. El coste está relacionado con el número de operaciones de entrada y de salida y procesos internos que Oracle debe ejecutar para realizar el paso. Los costes que se indican son acumulativos, de modo que el coste de un paso incluye el coste de dicho paso más los costes de todos sus pasos hijos. Los costes no pueden compararse entre consultas de diferentes tipos, y la versión de menor coste de la consulta puede no ser la mejor consulta para sus usuarios.

A partir de Oracle9i, puede utilizar una vista V$, V$SQL_PLAN, para ver el plan de ejecución para todas las instrucciones actualmente existentes en la caché de biblioteca. El siguiente listado proporciona una breve versión del plan de ejecución para cada instrucción encontrada:

```
select SQL_Text,
LPAD('',2*(Level-1))||Operation||' '||Options||''
||Object_Name||''
||DECODE(Object_Node,'','','['||Object_Node||'] ')
||DECODE(Optimizar, ' ', ' ', ' ['||Optimizer||'] ')
||DECODE(id,0, 'Cost = '||Position) Query
from V$SQLAREA, V$SQL_PLAN
where V$SQLAREA.Address = V$SQL_PLAN.Address and
V$SQLAREA.Hash_Value = V$SQL_PLAN.Hash_Value connect by prior
ID = Parent_ID and prior V$SQL_PLAN.Address =
V$SQL_PLAN.Address and prior V$SQIAREA.Hash_Value =
V$SQL_PLAN.Hash_Value
start with ID = 0
and V$SQLAREA.Address = V$SQL_PLAN.Address
and V$SQLAREA.Hash_Value = V$SQL_PLAN.Hash_Value
order by V$SQLAREA.Address
```

En general, las aplicaciones orientadas a transacciones (como, por ejemplo, sistemas multiusuario utilizados para entradas de datos) juzgan el rendimiento basándose en el tiempo necesario para devolver la primera fila de un conjunto de resultados. Para aplicaciones orientadas a transacciones, debe centrar sus esfuerzos en utilizar índices para reducir el tiempo necesario para devolver la primera fila del conjunto de resultados.

Si la aplicación está orientada a procesos por lotes (con transacciones de gran tamaño e informes), debe centrarse en mejorar el tiempo necesario para completar la transacción entera en lugar del tiempo necesario para devolver la primera fila de la transacción. Mejorar el rendimiento global de la transacción puede requerir el uso de exploraciones de tabla completas en lugar de accesos a índices (y puede mejorar el rendimiento global de la aplicación).

Si la aplicación está distribuida a lo largo de varias bases de datos, céntrese en reducir el número de veces que se utilizan los enlaces de base de datos en las consultas. Si se accede con frecuencia a una base de datos remota durante una consulta, entonces el coste de acceder a dicha base de datos remota se paga cada vez que se accede a los datos remotos. Incluso aunque el coste de acceder a los datos remotos sea bajo, acceder a ellos miles de veces supondrá finalmente una carga adicional de rendimiento sobre su aplicación. Consulte la sección «Reducción del tráfico de red>> más adelante; se le ofrecen sugerencias adicionales de optimización para bases de datos distribuidas.

CAPÌTULO 3
OPTIMIZACIÓN DEL USO DE LA MEMORIA

La caché de buffers de bloques de datos y el área compartida se gestionan mediante un algoritmo *LRU* (least recently used, menos recientemente utilizado). Un área pre-determinada se reserva para albergar la información; cuando se llena, los datos menos recientemente utilizados se eliminan de la memoria y se escriben definitivamente en el disco. Un área de memoria con un tamaño adecuado mantiene los datos a los que se accede con más frecuencia en la memoria, mientras que el acceso a los datos menos frecuentemente utilizados requiere lecturas físicas.

La *tasa de aciertos* es una medida del grado de acierto con el que la caché de buffers de datos está gestionando las peticiones de datos. En su forma general, se calcula como

Tasa de aciertos = (Lecturas lógicas - Lecturas físicas)/Lecturas lógicas

Así, una tasa de aciertos perfecta tendría un valor de 1.00. En ese caso, todas las peticiones de bloques de base de datos (lecturas lógicas) serían satisfechas sin que sea necesario acceder a datos de los archivos de datos (lecturas físicas); todas las solicitudes serían resueltas utilizando los datos que ya están en memoria. La tasa de aciertos global para una aplicación disminuirá debido a su actividad de procesos por lotes. Tenga en cuenta que la tasa de aciertos de una base de datos es acumulativa, reflejando todo el proceso realizado desde la última vez que se inició la base de datos. Una sola consulta mal escrita o un proceso por lotes de larga duración afectarán negativamente la tasa de aciertos durante intervalos de tiempo específicos.

Puede ver a las consultas que están realizando las lecturas físicas y lógicas en la base de datos mediante la vista V$SQL.V$SQL informa del número acumulado de lecturas lógicas y físicas que cada consulta está realizando actualmente en el área compartida, así como el número de veces que se ejecutó cada consulta. El script del siguiente listado mostrará el texto SQL de las consultas presentes en el área compartida, enumerando en primer lugar las consultas que más operaciones de entrada y salida (Disk_reads) requieren. La consulta muestra también el número de lecturas lógicas (representado por la columna Buffer_gets) por ejecución.

```
select Buffer_Gets,
       Disk_Reads,
       Executions,
       Buffer_Gets/Executions B_E,
       SQL_Text
from V$SQL
order by Disk_Reads desc;
```

Si se ha vaciado el área compartida, las consultas ejecutadas antes del vaciado ya no estarán accesibles mediante V$SQL. No obstante, el impacto de dichas consultas aún puede observarse, siempre que los usuarios sigan conectados. La vista V$SESS_IO registra las

lecturas lógicas acumuladas y las lecturas físicas realizadas en cada sesión de usuario. Puede consultar V$SESS_IO para conocer la tasa de acierto. de cada sesión, tal y como se muestra en el siguiente listado:

```
select SESS.Username,
       SESS_IO.Block_Gets,
       SESS_IO.Consistent_Gets,
       SESS_IO.Physical_Reads,
       round(100*(SESS_IO.Consistent_Gets
           +SESS_IO.Block_Gets-SESS_IO.Physical_Reads)/
           (decode(SESS_IO.Consistent_Gets,0,1,
        SESS_IO.Consistent_Gets+SESS_IO.Block_Gets)),2)
           session_hit_ratio
from V$SESS_IO sess_io, V$SESSION sess
where SESS.Sid = SESS_IO.Sid
and SESS.Username is not null
order by Username;
```

Puede manipular las acciones del algoritmo LRU en la caché de buffers de bloques de datos mediante la opción **cache.** Normalmente, cuando se ejecuta una exploración completa de tabla, los bloques de la tabla se sitúan en la zona de bloques menos recientemente utilizados de la lista LRU, de forma que serán sobreescritos primero. Cuando utilice la opción **cache,** Oracle situará estos bloques en la zona de bloques más recientemente utilizados de la lista LRU. Los datos de dicha tabla seguirán estando sujetos a los algoritmos LRU que gestionan las caches de la SGA, pero permanecerán en la SGA más tiempo que si se les hubiera tratado normalmente. La opción **cache** puede especificarse en el nivel de tabla mediante los comandos **create table y alter table,** así como mediante indicaciones en las consultas. La opción **cache** resulta útil principalmente para tablas a las que se accede con frecuencia y que no cambian muy a menudo. Después puede ejecutar consultas que volverán a cargar las tablas más utilizadas en las caches de la SGA cada vez que la base de datos se inicie de nuevo.

Para ver los objetos cuyos bloques están actualmente en la caché de buffers de bloques de datos, consulte la tabla X$BH en el esquema de SYS, como se muestra en el siguiente listado. En él, los objetos SYS y SYSTEM están excluidos del resultado, de forma que el DBA puede centrarse en las tablas de aplicación y en los índices presentes en la SGA:

```
select Object_Name,
       Object_Type ,
       count(*) Num_Buff
from SYS.X$BH a, SYS.DBA_OBJECTS b
where A.Obj = B.Object_Id
and Owner not in ( ' SYS ' , ' SYSTEM' )
group by Object_Name, Object_Type;
```

Con todas las áreas de la SGA (los buffers de bloques de datos, la memoria caché de diccionario y el área compartida), el énfasis debe ponerse en compartir los datos entre los usuarios. Cada una de estas áreas debe ser lo bastante grande como para albergar los datos que se le pidan más habitualmente a la base de datos. En el caso del área compartida, debe ser lo bastante grande como para albergar las versiones analizadas de las consultas más habitualmente utilizadas. Cuando tienen el tamaño adecuado, las áreas de memoria de la SGA pueden mejorar enormemente el rendimiento de las consultas individuales y de la base de datos como un todo.

Puede crear un *área de bloques de gran tamaño* dentro de la SGA que se utilizará cuando Oracle pida áreas de memoria contiguas de gran tamaño dentro del área compartida (como por ejemplo durante el uso del servidor multihebra). Para crear el área de bloques de gran tamaño, establezca un valor (en bytes) para el parámetro de inicialización LARGE_POOL_SIZE. De manera predeterminada, el área no se crea. Configurar algunos parámetros como PARALLEL_AUTOMATIC_TUNING = TRUE también asignará al parámetro LARGE_POOL_SIZE un valor distinto de cero.

Puede reservar un área dentro del área compartida para objetos de gran tamaño mediante el parámetro de inicialización SHARED_POOL_RESERVED_SIZE. El <<tamaño reservado>> se deja aparte para las entradas en el área compartida de objetos de gran tamaño (como paquetes grandes).

En lugar de reservar espacio en el área compartida, quizá prefiera *ubicar permanentemente* paquetes en la memoria de manera selectiva. Llevar a cabo esta operación inmediatamente después de iniciar la base de datos aumentará la probabilidad de que una sección lo suficientemente grande de espacio libre contiguo esté disponible en memoria. El procedimiento KEEP del paquete DBMS_SHARED_POOL designa los paquetes que se deben ubicar permanentemente en el área compartida. Como se muestra en siguiente listado, primero debe hacer referencia al objeto que desee ubicar:

```
alter procedure APPOWNER.ADD_CLIENT compile;
execute DBMS_SHARED_POOL.KEEP('APPOWNER.ADD_CLIENT','P');
```

La ubicación permanente en memoria de paquetes está más asociada a la gestión de aplicaciones que a su optimización, pero puede afectar al rendimiento. Si puede evitar la gestión dinámica de áreas de memoria fragmentadas, reducirá el trabajo que Oracle tiene que hacer en la gestión del área compartida.

Oracle lee varios bloques al mismo tiempo durante una exploración completa de tabla. El número de bloques leídos durante cada lectura física se determina mediante la configuración del parámetro de inicialización DB_FILE_MULTIBLOCK_READ_COUNT. El número de bloques leídos en un determinado momento está limitado por el tamaño del buffer de E/S de su sistema operativo. Si el tamaño del buffer de su sistema operativo es 128 KB. y el tamaño de bloque de su base de datos es 4 KB, entonces debe asignar a DB_FILE_MULTIBLOCK_READ_COUNT un valor de 32 (128 KB dividido por el tamaño de bloque de 4 KB). Si lo configura a un valor inferior, el rendimiento de sus exploraciones completas de tabla se verá negativamente afectado.

Puede determinar la configuración máxima de multibloque en su entorno mediante los siguientes pasos:

1. Cree un nuevo espacio de tablas con un único archivo de datos.
2. Cree una sola tabla no indexada en ese espacio de tablas.
3. Consulte V$FILESTAT para verificar las estadísticas iniciales para la prueba.
4. Realice una exploración completa de tabla.
5. Consulte V$FILESTAT para determinar las estadísticas finales para la prueba, y résteles las estadísticas iniciales. Divida el valor PhyBlkRds por PhyRds para determinar el número de lecturas multibloque efectivas.
6. Elimine el espacio de tablas.

El siguiente listado ilustra estos pasos:

```
create tablespace testar
datafile 'E:\0racle\0radata\orcl\tester.dbf' size 10M reuse
default storage (initial 1M next 1M pctincrease 0);


Tablespace created.
```

El siguiente paso crea una tabla basada en una ya existente en esta base de datos.

```
create table TESTING
tablespace tester
as select * from perftest.emp
where rownum < 50000;

select relative_fno from dba_data_files
where tablespace_name = 'TESTER';

RELATIVE_FNO
------------
           9

SQL> select phyrds, phyblkrd from v$filestat where file#=9;

PHYRDS   PHYBLKRD
------   --------
     0          0
```

Realiza una exploración completa de tabla:

```
SQL> select count(*) from testing;

COUNT(*)
--------
    49999

select phyrds, phyblkrd from v$filestat where file#=9;

PHYRDS   PHYBLKRD
------   --------
    154       1220

drop tablespace tester including contents;
```

Dividir la columna PhyBlkRd entre la columna PhyRds da como resultado 7.92; el número de lecturas multibloque eficaces es 8. Si el espacio de tablas no era nuevo, tendría que restar las estadísticas iniciales (paso 3) de las finales (paso 5) para determinar el cambio de estadísticas durante la prueba. Entonces podrá cambiar el valor del parámetro DB_FILE_MULTIBLOCK_READ_COUNT en el nivel de sesión y repetir la prueba. No configure el parámetro DB_FILE_MULTIBLOCK_READ_COUNT del archivo de parámetros a un valor superior al valor que calcule.

Cuando cree áreas de buffers, puede especificar el tamaño del à*rea de preservación* y el tamaño del *área de reciclado.* Al igual que el área reservada del área compartida, el área de preservación conserva entradas, mientras que el área de reciclado se recicla con más frecuencia. Puede especificar el tamaño del área de preservación mediante el parámetro DB_KEEP_CACHE_SIZE, como se muestra en el siguiente listado:

```
DB_KEEP_CACHE_SIZE=20M
DB_RECYCLE_CACHE_SIZE=4m
```

En versiones anteriores a Oracle9i el tamaño de las áreas de buffers de preservación y reciclado reducía el espacio disponible en la caché de buffers de bloques de datos porque las tres áreas eran parte del área definida por el parámetro de inicialización DB_BLOCK_BUFFERS. En Oracle9i, las áreas de memoria definidas por DB_KEEP_CACHE_SIZE y DB_RECYCLE_CACHE_SIZE son áreas adicionales e independientes del área definida por DB_CACHE_SIZE. Para que una tabla utilice una de las nuevas áreas de buffers, especifique el nombre del área de buffers mediante el parámetro **buffer_pool** dentro de la cláusula **storage** de la tabla. Por ejemplo, si quiere que una tabla sea rápidamente eliminada de la memoria, asóciela al área RECYCLE. El área predeterminada se llama DEFAULT, de forma que puede utilizar el comando **alter table** para asociar una tabla al área DEFAULT con posterioridad.

3.1 Cómo especificar el tamaño de la SGA

Para crear la SGA, debe especificar valores para los siguientes parámetros de inicialización:

SGA_MAX_SIZE En Oracle9i, puede especificar el tamaño máximo hasta el que puede crecer la SGA. El tamaño del área compartida y la caché de buffers de bloques de datos puede cambiar dinámicamente.

SHARED_POOL_SIZE El tamaño del área compartida.

DB_BLOCK_SIZE Será el tamaño de bloque de base de datos predeterminado para la base de datos tal y como se haya establecido durante la creación de la base de datos.

DB_CACHE SIZE Este parámetro reemplaza el parámetro DB_BLOCK_BUFFERS utilizado en anteriores versiones del RDBMS de Oracle. El tamaño de la caché se especifica en bytes en lugar de en bloques. Oracle redondea el tamaño a unidades de 4 MB si el tamaño indicado por SGA_MAX_SIZE es menor que 128 MB. En caso contrario, redondeará a unidades de 16 MB.

DB_nK CACHE SIZE Si va a utilizar varios tamaños distintos de bloques de base de datos dentro de una misma base de datos, debe especificar un valor para el parámetro DB_CACHE_SIZE y. al menos, un valor para un parámetro de tipo DB_nK_CACHE_SIZE. Por ejemplo, si su tamaño de bloque de base de datos estándar es 4 KB. puede especificar también una caché para los espacios de tablas con un tamaño de bloque de 8 KB mediante el parámetro DB 8K CACHE SIZE

Por ejemplo, puede especificar:

```
MAX_SGA_SIZE=500M
SHARED_POOL_SIZE=8OM
DB_BLOCK_SIZE=8192
DB_CACHE_SIZE=160M
DB_4K_BLOCK_SIZE= 4M
```

Dentro de la SGA estarán disponibles 4 MB para datos consultados desde objetos en espacios de tablas con tamaños de bloque de 4 KB. Los objetos que utilizan el tamaño de bloque estándar de 8 KB emplearán la caché de 160 MB. Mientras la base de datos está abierta, puede cambiar los valores de los parámetros SHARED_POOL_SIZE y DB_CACHE_SIZE mediante el comando **alter system.**

3.2 Utilización del optimizador basado en costes

Con cada versión de su software, Oracle ha incorporado nuevas funciones a su optimizador y ha mejorado las funciones ya existentes. Como resultado de ello, el rendimiento cuando se utiliza el optimizador basado en costes (CBO) debe haber ido mejorando de manera progresiva. Aunque la indicación RULE y la optimización basada en reglas están disponibles en Oracle8, el papel de la optimización basada en reglas disminuirá probablemente con el tiempo, y debería empezar a modificar su sistema para que emplee la optimización basada en costes si es que no lo ha hecho ya.

El uso eficaz del optimizador basado en costes requiere que las tablas y los índices *de* su aplicación se analicen con regularidad. La frecuencia con la que analice los objetos depende de la velocidad de cambio de los objetos. Para aplicaciones de transacciones por lotes, debe analizar de nuevo los objetos tras cada conjunto de gran tamaño de transacciones por lotes. Para aplicaciones OLTP, debe analizar de nuevo los objetos con cierta periodicidad (por ejemplo con un proceso semanal o uno nocturno).

Las estadísticas sobre los objetos se recopilan mediante los paquetes DBMS_STATS o DBMS_UTILITY (el comando **analyze** va a quedar obsoleto en Oracle9i). Si analiza una tabla, entonces sus índices asociados se analizan también automátic amenté.

Puede ver las estadísticas de la tabla COMPANY y de sus índices accediendo a las tablas DBA_TABLES, DBA_TAB_COL_STATISTICS y DBA_INDEXES. Se siguen proporcionando algunas estadísticas ligadas a columnas en la tabla DBA_TAB_COLUMNS, pero se proporcionan únicamente por compatibilidad con versiones anteriores. Las estadísticas sobre las columnas de tablas particionadas se pueden encontrar en DBA_PART_COL_STATISTICS.

El análisis de las tablas genera histogramas sobre los datos de las tablas. Un histograma refleja la distribución de valores de datos dentro de una tabla. Por ejemplo, puede haber muchos valores distintos para una columna, en cuyo caso la columna puede parecer ideal como valor de restricción para una consulta. No obstante, el 90 por 100 de dichos valores pueden agruparse juntos, quedando el 10 por 100 restante de los valores fuera de la agrupación. Si su consulta realiza una exploración por rangos con un valor de restricción que esté dentro de la agrupación, un índice puede no ayudar al rendimiento de su consulta. El uso de un índice para valores exteriores a la agrupación afectaría en gran medida al rendimiento.

¿Cómo sabe el optimizador dónde están las agrupaciones de valores de datos? Cuando se ejecuta el comando **analyze,** se le puede indicar a Oracle que genere un histograma para la agrupación. De manera predeterminada, Oracle creará un histograma que divide los valores de datos en 75 *divisiones. Cada* división tiene el mismo número de registros que las demás. Cuántas más porciones cree, mejor se reflejará la distribución de los valores de la base de datos mediante dicho histograma. Puede especificar el número de divisiones que desee utilizar mediante el parámetro **size** del comando **analyze.** El máximo número de divisiones por tabla es 254.

Para analizar todos los objetos en un esquema, puede utilizar el procedimiento ANALYZE_SCHEMA del paquete DBMS_UTILITY. Como se muestra en el siguiente listado, tiene dos parámetros: el nombre del esquema y la opción de análisis empleada (COMPUTE, calcular, o ESTIMATE, estimar).

```
execute DBMS_UTILITY.ANALYZE_SCHEMA('APPOWNER','COMPUTE');
```

Cuando se ejecuta el comando del listado anterior, se mostrarán todos los objetos que pertenecen al esquema APPOWNER, utilizando la opción **compute statistics.** Si está utilizando optimización basada en reglas, entonces las estadísticas, aunque no hayan sido utilizadas durante el proceso de optimización general, proporcionarán información útil a los desarrolladores durante el proceso de optimización de consultas.

Puede utilizar el paquete DBMS_STATS para analizar esquemas, tablas, columnas e índices. Los procedimientos del paquete DBMS_STATS almacenan estadísticas en una tabla local para su posterior traslado al diccionario de datos. Para que el optimizador utilice las estadísticas, debe trasladarlas al diccionario de datos. Puede crear varios juegos de estadísticas para su uso durante las pruebas de rendimiento.

Dentro del paquete DBMS_STATS, puede utilizar el procedimiento GATHER_TABLE_STATS para recopilar estadísticas sobre tablas, columnas e índices. Para las estadísticas sobre índices, utilice el procedimiento GATHER_INDEX_STATS. GATHER_SCHEMA_STATS, GATHER_DATABASE_STATS y GATHER_SYSTEM_STATS reúnen estadísticas para el esquema, la base de datos y las operaciones de E/S del sistema, respectivamente.

Por ejemplo, el siguiente comando reúne las estadísticas sobre la tabla PERFTEST.EMP. Para ver las estadísticas recopiladas, consulte DBA_TABLES.

```
begin
DBMS_STATS .GATHER_TABLE_STATS (' perf test' , ' emp ') ;
end;
```

Durante la creación o reconstrucción de un índice, puede utilizar la cláusula **compute statistics** para recopilar estadísticas a medida que el índice se va llenando.

3.3 Consecuencias del uso de Compute Statistics

En los ejemplos de la sección anterior, la opción **compute statistics** del comando **analyze** se utilizó para recopilar estadísticas sobre los objetos. Oracle ofrece además una opción **estimate statistics,** que muestrea bloques aleatorios de varios bloques distintos repartidos por todas las extensiones del segmento. Si elige utilizar **estimate statistics,** analice todo el contenido de la tabla que le sea posible (puede especificar un porcentaje de las filas que desee analizar).

Para generar las estadísticas más precisas, debe utilizar la opción **compute statistics** siempre que le sea posible. Para la mayoría de los entornos, la opción **estímate statistics** genera estadísticas aceptables si se analiza entre el 6 y el 8 por 100 de la tabla. Para los índices utilice un porcentaje más elevado cuando emplee la opción **estimate statistics.**

Hay algunos aspectos de administración asociados a la opción **compute statistics.** Concretamente, pueden ser necesarias grandes cantidades de espacio de segmento temporal (hasta cuatro veces el tamaño de la tabla). Debe asegurarse de que el usuario que realice el análisis tenga los valores de espacio de tablas temporal adecuados y que el espacio de tablas temporal pueda gestionar los requisitos de espacio. A medida que la tabla crece con

el tiempo, las necesidades de espacio de segmento temporal de **compute statistics** aumentarán. Si ha particionado la tabla o el índice, puede analizar particiones individuales, evitando tener que recalcular las estadísticas para los datos estáticos del resto de particiones.

CAPÌTULO 4
OPTIMIZACIÓN DEL ALMACENAMIENTO DE DATOS

La posición oficial de Oracle es que no se produce una sobrecarga notable cuando se accede a objetos con 4.096 extensiones o menos. En realidad es preferible tener un número mayor de extensiones en distintos archivos para obtener el mejor rendimiento de las consultas paralelas.

La fragmentación del espacio libre puede ralentizar el rendimiento cuando se almacenan nuevos registros. Si el espacio libre de un espacio de tablas está fragmentado, entonces la base de datos puede tener que combinar dinámicamente extensiones libres contiguas para crear una sola extensión que sea lo bastante grande como para gestionar los nuevos requisitos de espacio. La optimización del almacenamiento de datos ha de tener en cuenta tanto la gestión del espacio ocupado como la del espacio libre, tal y como se describe en las siguientes secciones. Los DBA deben utilizar espacios de tablas gestionados localmente, a partir de Oracle8i, para evitar por completo este problema.

La mayoría de las plataformas de base de datos de ámbito corporativo utilizan tecnologías RAID (Redundant Array of Independen! Disks: matriz redundante de discos independientes) para mejorar el rendimiento de las operaciones de E/S del sistema. Los sistemas RAID distribuyen datos de un solo archivo entre varios discos. Las peticiones de datos efectuadas sobre un archivo son distribuidas entre varios discos, reduciendo así la carga de E/S sobre cada disco individual. Como resultado de ello, se producirán menos cuellos de botella en las operaciones de E/S.

4.1 Desfragmentación de los segmentos

Cuando se crea un objeto de base de datos (como una tabla o un índice), se asigna a un espacio de tablas mediante valores predeterminados de usuario o instrucciones específicas. Se crea un *segmento* en ese espacio de tablas para albergar los datos asociados a ese objeto. El espacio que se asigna al segmento nunca se libera hasta que éste se borra o trunca.

Un segmento está formado por secciones llamadas *extensiones.* Las extensiones en sí mismas son conjuntos de bloques de Oracle contiguos. Cuando las extensiones existentes ya no puedan albergar datos nuevos, el segmento obtendrá otra extensión. Este proceso de extensión seguirá hasta que no haya más espacio libre disponible en los archivos de datos del espacio de tablas, o hasta que se alcance un número máximo interno de extensiones por segmento. Para simplificar la gestión de segmentos, debe utilizar un conjunto coherente de tamaños de extensiones. Los tamaños que elija deben ser múltiplos del tamaño del bloque de E/S del sistema operativo, y deben ser múltiplos uno de otro. Por ejemplo, puede crear todas sus tablas pequeñas con tamaños de extensión de 1 MB, tablas de tamaño medio con tamaños de extensión de 4 MB y tablas grandes con tamaños de extensión de 16 MB. Si sus extensiones tienen el tamaño adecuado, entonces las consultas a las tablas no se verán afectadas por el número de extensiones de la tabla. No obstante, las operaciones DDL sí pueden verse negativamente afectadas.

Oracle soporta dos tipos de gestión interna del espacio: espacios de tablas gestionados por diccionario y espacios de tablas gestionados localmente. En los espacios de tablas

gestionados por diccionario, los datos de gestión del espacio se almacenan en el diccionario de datos, en tablas llamadas SYS.UET$ y SYS.FET$. En espacios de tablas gestionados localmente, los datos de gestión del espacio se almacenan en un mapa de bits dentro de los propios archivos de datos del espacio de tablas.

Cuando un objeto asigna una extensión en un espacio de tablas gestionado por diccionario, Oracle actualiza las entradas de la tabla de extensiones utilizadas, SYS.UET$. Al mismo tiempo, actualiza las entradas de la tabla de extensiones libres, S YS.FET$. La tabla SYS.UET$ tiene un solo registro por cada extensión de la base de datos, y SYS.FETS tiene una fila por cada extensión libre de la base de datos. Si tiene un gran número de extensiones en una tabla o índice, entonces los comandos DDL que actualizan implícitamente las tablas SYS.UET$ y SYS.FET$ pueden afectar al rendimiento de sus comandos.

Por ejemplo, suponga una tabla que tiene 10.000 extensiones (por sí sola o como colección de particiones que contienen juntas 10.000 extensiones). Cuando elimine esa tabla, Oracle tendrá que realizar 10.000 operaciones de actualización **(update)** de SYS.UET$ (ya que los datos de SYS.UET$ siempre deben ser coherentes para todos los usuarios de la base de datos). Al mismo tiempo, Oracle debe actualizar SYS.FET$ y las demás tablas de diccionario de datos utilizadas para el mantenimiento de los objetos (para privilegios, columnas, etc.). SYS.UET$ no está debidamente optimizada para dar soporte a tablas con miles de extensiones. En un entorno de pruebas, un comando **drop table** de una tabla de 5.000 extensiones tardó dos minutos en completarse. En el mismo entorno, fueron necesarios diez minutos para completar un comando **drop table** de una tabla de 10.000 extensiones. A medida que se añadían más extensiones, el tiempo necesario para eliminar la tabla aumentaba exponencialmente.

Aunque el número de extensiones por segmento es ilimitado, un número límite de extensiones que se puede emplear en la práctica para espacios de tablas gestionados por diccionario es 10.000.

Cuando considere el impacto del número de extensiones sobre el rendimiento de sus comandos DDL, debe tener en cuenta no solamente sus tablas, sino también los índices, las particiones y las particiones de índices que se eliminarán junto con la tabla. Si tiene una tabla con 100 particiones, cada una con 300 extensiones y cada una de las extensiones con un índice local que a su vez contiene 300 extensiones, entonces para eliminar la tabla será necesario eliminar 60.000 extensiones (lo que puede producir problemas de rendimiento). Para resolver este problema, Oracle9i introdujo el concepto de espacios de tablas gestionados localmente, en los que la información sobre la ocupación de extensiones se almacena en un mapa de bits en la cabecera del archivo de datos en lugar de en el diccionario de datos. Los espacios de tablas gestionados localmente se tratan con más detalle a continuación.

El sistema de supervisión proporcionado en el Capítulo 6 comprueba la vista de diccionario de datos DBA_SEGMENTS para determinar los segmentos que tienen diez o más extensiones. En el siguiente listado se muestra una consulta general de la vista DBA_SEGMENTS. La siguiente consulta recuperará el nombre del espacio de tablas, el propietario, el nombre de segmento y el tipo de segmento para cada segmento de la base de datos junto con el número de extensiones y bloques utilizados por el segmento.

```
select
```

```
    Tablespace_Name, /*Nombre del espacio de tablas*/
    Owner, /*Propietario del segmento*/
    Segment_Name, ,/*Nombre del segmento*/
    Segment_Type, /*Tipo de segmento (ej. TABLE, INDEX)*/
    Extents, /*Número de extensiones en el segmento*/
    Blocks,/*Número de bloques de base de datos en el
segmento*/
    Bytes /*Número de bytes en el segmento*/
from DBA_SEGMENTS
```

Los tipos de segmento incluyen los tipos denominados TABLE, TABLE PARTITION, INDEX, INDEX PARTITION, LOBINDEX, LOBSEGMENT, NESTED TABLE, CLUSTER, ROLLBACK, TEMPORARY, DEFERRED ROLLBACK, TYPE2 UNDO (nuevo en Oracle9i) y CACHE. La vista DBA_SEGMENTS no permite determinar el tamaño de las extensiones individuales de un segmento. Para ver el tamaño de cada extensión, consulte la vista DBA_EXTENTS, como se muestra en este listado:

```
select
    Tablespace_Name, /*Nombre del espacio de tablas*/
    Owner, /*Propietario del segmento*/
    Segment_Name, ,/*Nombre del segmento*/
    Segment_Type, /*Tipo de segmento (ej. TABLE, INDEX)*/
    Extents, /*Número de extensiones en el segmento*/
    Blocks,/*Número de bloques de base de datos en el
segmento*/
    Bytes /*Número de bytes en el segmento*/
from DBA_SEGMENTS
where Segment_Name = 'nombre_segmento'
order by Extent_ID;
```

La consulta descrita en el listado anterior selecciona la información sobre las extensiones para un solo segmento (identificado mediante la cláusula **where).** Devuelve la información de almacenamiento asociada a las extensiones del segmento, incluyendo el tamaño y la ubicación de cada extensión de datos. Más adelante en este mismo capítulo se muestra una consulta similar que se utiliza para determinar la distribución de extensiones libres y ocupadas de un espacio de tablas.

Si la tabla es una tabla organizada por índice, puede reconstruirse en línea mediante la cláusula **move** del comando **alter table.** Si la tabla está organizada de forma estándar, reconstruirla requiere utilizar los comandos Export/Import, o utilizar las opciones de reconstrucción de objetos .

El comando Export admite el indicador COMPRESS. El indicador COMPRESS provocará que el comando Export, al leer una tabla, determine la cantidad total de espacio reservado por esa tabla. A continuación escribirá en el archivo de volcado de exportación un nuevo parámetro de almacenamiento **initial** (equivalente al total del espacio asignado) para la tabla. Si, después, la tabla es eliminada, y se utiliza el comando Import para volver a

crearla, entonces sus datos deberían todos caber dentro de la nueva extensión inicial de mayor tamaño.

Tenga en cuenta que es el espacio *asignado, y* no el *ocupado,* lo que se comprime. Una tabla vacía con 300 MB asignados en tres extensiones de 100 MB se comprimirá en una única extensión vacía de 300 MB. No se reclamará espacio adicional alguno. Además, la base de datos no comprobará que el tamaño de la nueva extensión inicial **(initial)** sea mayor que el tamaño del archivo de datos de mayor tamaño para el espacio de tablas. Como las extensiones no pueden estar repartidas entre varios archivos de datos, tratar de crear una extensión más grande que un archivo de datos dará como resultado un error durante la importación.

4.2 Evaluación del uso de índices

En Oracle9i se puede activar la supervisión de índices para determinar si se están utilizando. Si un índice no se está utilizando, puede borrarlo, ahorrando espacio y mejorando el rendimiento mediante la eliminación de una sobrecarga innecesaria durante las operaciones DML. La cantidad de índices no utilizados que sea posible eliminar reducirá el número de índices que deberán reconstruirse al crear de nuevo un esquema. Como la supervisión de índices se activa y desactiva en el nivel de sistema, debe activar o desactivar esta función cuando la actividad de su base de datos sea mínima o cuando inicie su base de datos. Debe ejecutar el comando para activar o desactivar la supervisión de índices índice por índice, ya que no hay disponible un comando en el nivel de esquema. El siguiente listado muestra el comando que activa la supervisión de índices sobre el índice EMPLOYEE_IDX:

```
alter index EMPLOYEE_IDX monitoring usage;
```

Para desactivar la supervisión de la utilización del índice, ejecute el comando:

```
alter index EMPLOYEE_IDX nomonitoring usage;
```

Cuando active la supervisión de la utilización de índices, hágalo sólo para su sesión en particular y no para la base de datos entera.

Para ver los resultados de la sesión de supervisión, utilice la vista V$OBJECT_USAGE. La vista muestra información sobre los nombres del índice y de la tabla, si está activada la supervisión, si se están utilizando los índices y las horas a las que se activó y/o detuvo la supervisión. Cada vez que activa la supervisión de la utilización de índices, la vista se reinicializa para el índice especificado y la información de utilización anterior se elimina o reinicializa. La información de la vista permanece inalterada una vez que finaliza la supervisión de un índice hasta que se activa la supervisión de la utilización de dicho índice de nuevo o bien se borra el índice. La vista V$OBJECT_USAGE es *dinámica;* es decir, el contenido mostrado depende de la identidad y de los privilegios del usuario que consulta la vista, pero la vista se basa en una tabla de diccionario de datos real de forma que el contenido se conserva y se mantiene coherente incluso tras una caída de la base de datos.

Digamos que ha activado previamente la supervisión en los índices PK_EMPL y PK_DEP. En este ejemplo, la supervisión sigue estando activada en el índice PK_DEP pero se ha desactivado en el índice PK_EMPL. Puede utilizar la siguiente consulta para ver los resultados almacenados en la vista V$OBJECT_USAGE:

```
select
 Index_Name, /* El nombre del índice */
 Table_Name, /* El nombre de la tabla */
 Monitoring, /* Si está activado (SÍ/NO) */
 Used,       /* Si el índice se ha utilizado (SÍ/NO) */
 Start Monitoring, /* Hora de inicio de la supervisión */
 End_Monitoring    /* Hora de parada de la supervisión */
from V$OBJECT_USAGE;
```

INDEX_NAME	TABLE_NAME	MON	USE	START_MON	END MON
PK_DEP	DEPARTMENT	YES	YES	08/21/2001	
PK_EMPL	EMPLOYEES	NO	NO	08/21/2001	08/21/2001

Como puede ver, durante el período de tiempo en el que se ha activado la supervisión, el índice PK_DEP se ha utilizado pero el índice PK_EMPL no.

Solamente podrá ver la información almacenada en V$OBJECT_USAGE de la sesión en la que active la supervisión. Si activa la supervisión de la utilización de índices desde el esquema HR, no podrá ver los resultados del esquema SYSTEM.

4.3 Espacios de tablas gestionados localmente

A partir de Oracle8i, están disponibles los espacios de tablas gestionados localmente y pueden encargarse de la gestión de extensiones dentro de los propios espacios de tablas. En los espacios de tablas gestionados localmente, el espacio de tablas se encarga de gestionar su propio espacio manteniendo un mapa de bits en cada archivo de datos de los bloques o conjuntos de bloques libres y ocupados del archivo de datos. Cada vez que una extensión se asigna o libera para su reutilización, Oracle actualiza el mapa de bits para mostrar el nuevo estado.

Cuando utilice espacios de tablas gestionados localmente, el diccionario no se actualizará y no se generará actividad de anulación. Los espacios de tablas gestionados localmente llevan a cabo automáticamente el seguimiento del espacio libre adyacente, de manera que no es necesario agrupar manualmente las extensiones. Dentro de un espacio de tablas gestionado localmente, todas las extensiones pueden tener el mismo tamaño, o el sistema puede determinar automáticamente el tamaño de las extensiones.

Para utilizar la gestión local del espacio, especifique la opción **local** en la cláusula **extent management** en el comando **create tablespace** (en Oracle9i es éste el tipo predeterminado de gestión de extensiones). A continuación se muestra un ejemplo del comando **create tablespace** declarando un espacio de tablas gestionado localmente:

```
create tablespace CODESJTABLES
datafile   '/u01/oracle/VLDB/codes_tables.dbf'
size 10M
extent management local uniform size 256K;
```

Suponiendo que el tamaño de bloque para la base de datos en la que se ha creado este espacio de tablas sea de 8 KB, en este ejemplo, el espacio de tablas se crea con la gestión de extensiones declarada como **local** y con un tamaño uniforme de 256 KB. Cada bit del mapa de bits describe 32 bloques (256/8). Si se omite la cláusula **uniform size,** el valor predeterminado es **autoallocate.** El tamaño predeterminado para **uniform es 1 MB.**

Si especifica **local** en un comando **create tablespace,** no puede especificar una cláusula **default storage, minextents o temporary.** Si utiliza el comando **createtemporary tablespace** para crear el espacio de tablas, debe especificar **extent management local.**

A medida que los objetos de su base de datos crezcan en extensiones, los espacios de tablas gestionados localmente se hacen más importantes. Como ya hemos observado anteriormente, las operaciones de gestión del espacio asociadas a comandos DDL en tablas con muchos miles de extensiones pueden efectuarse mal debido a la gestión de diccionario de datos asociada. Si utiliza espacios de tablas gestionados localmente, dicha penalización en el rendimiento se reduce de manera notable.

4.4 Desfragmentación de extensiones libres

Una extensión libre en un espacio de tablas es una colección de bloques libres contiguos del espacio de tablas. Cuando se borra un segmento, sus extensiones dejan de estar asignadas y se marcan como libres. No obstante, estos segmentos libres no siempre se combinan de nuevo con extensiones libres contiguas, porque la separación entre estas extensiones libres puede mantenerse.

Para evitar la mayoría de los problemas de fragmentación del espacio, utilice espacios de tablas gestionados localmente en bases de datos Oracle8i y Oracle9.

El proceso en segundo plano SMON agrupa periódicamente extensiones libres contiguas si el valor predeterminado del parámetro **pctincrease** para el espacio de tablas es distinto de cero. Si este parámetro es cero, entonces la base de datos no agrupará automáticamente el espacio libre del espacio de tablas. Puede utilizar la opción **coalesce** del comando **alter tahlespace** para hacer que las extensiones libres contiguas se agrupen, con independencia del valor predeterminado del parámetro **pctincrease** para el espacio de tablas.

El proceso en segundo plano SMON solamente agrupa espacios de tablas cuyo valor predeterminado **pctincrease** no es cero. Un **pctincrease** de 1 obligará a SMON a agrupar el espacio libre adyacente en un espacio de tablas pero solamente realizará ocho agrupamientos al mismo tiempo para cada espacio de tablas. Para una mejor utilización del espacio, emplee espacios de tablas gestionados localmente de forma que el espacio de tablas nunca tenga que ser agrupado de esta manera.

No forzar el agrupamiento de extensiones libres afecta a la asignación del espacio dentro del espacio de tablas durante la siguiente petición de espacio (como por ejemplo debido a la creación o expansión de una tabla). En su búsqueda de una extensión libre lo

bastante grande, la base de datos no fusionará extensiones libres contiguas a menos que no haya otra alternativa. De este modo, se tiende a utilizar la extensión libre más amplia que hay al final del espacio de tablas, mientras que las extensiones libres más pequeñas situadas hacia el principio del espacio de tablas están relativamente desocupadas, convirtiéndose en un factor que «limita la velocidad>> del espacio de tablas porque por sí solos no tienen el tamaño adecuado para poder ser ocupados. A medida que progresa este patrón de uso, la base de datos se aleja cada vez más de la asignación ideal de espacio. La fragmentación de espacio libre predomina especialmente en entornos en los que las tablas y los índices de base de datos se borran y crean de nuevo con frecuencia en espacios de tablas gestionados por diccionario, especialmente si sus parámetros de almacenamiento se modifican en el proceso. La utilización de espacios de tablas gestionados localmente elimina este problema por completo.

Puede obligar a la base de datos a recombinar las extensiones libres contiguas, emulando así la funcionalidad SMON. Agrupar las extensiones libres aumentará la probabilidad de que las extensiones libres situadas cerca del principio del archivo se reutilicen, conservando así el espacio libre cerca del final del archivo del espacio de tablas. Como resultado de ello, es más probable que las nuevas peticiones de extensiones culminen con éxito.

Antes de intentar desfragmentar un espacio de tablas, primero debe obtener información detallada sobre la utilización del espacio en el espacio de tablas. El siguiente script mostrará todo el espacio marcado como libre u ocupado por los objetos de base de datos. Esto resulta útil para mostrar la distribución y el tamaño de las extensiones libres y para determinar los objetos de base de datos que se han situado como barreras entre las extensiones libres. Este script es útil principalmente para espacios de tablas gestionados por diccionario, ya que los espacios de tablas gestionados localmente no mostrarán los problemas de asignación de espacio ilustrados en el siguiente ejemplo.

Archivo: mapper.sql
Parámetros: el nombre del espacio de tablas que se está analizando
Ejemplo de invocación: @mapper DEMODATA

Este script genera la distribución de la utilización del espacio (espacio libre frente a ocupado) en un espacio de tablas. Muestra gráficamente la fragmentación de segmentos y del espacio libre.

```
set pagesize  60  linesize  132  verify off
column file_id heading  "File|Id"

select
 'free space' Owner, /*<<propietario<< del espacio libre*/
 ' ' Object,         /*nombre del objeto vacío*/
 File_ID,  /*ID de archivo para la cabecera de extensión*/
 Block_ID, /*ID de bloque para la cabecera de extensión*/
 Blocks    /*longitud de la extensión, en bloques*/
from DBA_FREE_SPACE
```

```
where Tablespace_Name = UPPER('&&1')
union
select
 SUBSTR(Owner,1,20),  /*nombre del propietario (primeros 20
caracteres)*/
 SUBSTR(Segment_Name,1,32) /*nombre del  segmento*/
 File_ID, /*ID de archivo para cabecera de extensión*/
 Block_ID, /*ID de bloque para cabecera de bloque*/
 Blocks /*longitud de la extensión en bloques*/
from DBA_EXTENTS
where Tablespace_Name = UPPER('&&1')
order by 3,4

spool &&1._map.lst
/
spool off
undefine 1
```

El resultado de ejemplo de esta consulta para un espacio de tablas gestionado por diccionario se muestra en el siguiente listado. El resultado de la consulta muestra el propietario y el nombre de segmento para cada extensión del espacio de tablas. Si la extensión está libre, entonces el propietario (owner) aparece como <<free space>>, y el nombre de segmento (la columna Object) se deja en blanco.

File Owner	OBJECT	Id	BLOCK_ID	Blocks
OPS$CC1	FILES	6	2	20
OPS$CC1	SPACES	6	22	20
OPS$CC1	EXTENTS	6	42	20
OPS$CC1	FILES	6	62	20
free space		6	82	5
free space		6	87	5
free space		6	92	5
OPS$CC1	SPACES	6	97	20
OPS$CC1	EXTENTS	6	117	20
free space		6	137	10
OPS$CC1	FILES	6	147	20
free space		6	167	14,833

La salida del listado anterior muestra 12 filas. Cinco de ellas son de extensiones de espacio libres, y siete son de segmentos de datos. Las primeras tres extensiones de espacio libre son contiguas. Después de dos extensiones de datos más, hay otra extensión de espacio libre. Como está separada de las demás extensiones libres del espacio de tablas, la

cuarta extensión libre no puede combinarse con ninguna de las demás extensiones libres a menos que el espacio de tablas sea desfragmentado.

4.4.1 Cómo combinar las extensiones libres

Si un espacio de tablas pudiera verse beneficiado del hecho de tener agrupadas sus extensiones libres, entonces debe agrupar dichas extensiones manualmente o bien activar el proceso SMON para que las agrupe.

Para activar el proceso SMON, debe configurar el valor predeterminado de **pctincrease** para el espacio de tablas a un valor distinto de cero. En el siguiente listado se modifican los parámetros de almacenamiento predeterminados para el espacio de tablas DEMONDX de manera que utilice un **pctincrease de 1.**

```
alter tablespace DEMONDX
default storage (pctincrease 1) ;
```

Si se crea un objeto en el espacio de tablas DEMONDX sin un valor **pctincrease** especificado, entonces el objeto empleará el valor **pctincrease** predeterminado (50 por 100) para el espacio de tablas. En general, los valores **pctincrease** bajos reflejan con precisión el crecimiento lineal normal del número de filas de la base de datos. Así, para el valor **pctincrease** se utilizó el valor distinto de cero mínimo admisible (1). Puede cambiar este valor predeterminado mediante la cláusula storage del objeto que cree.

Para agrupar manualmente las extensiones libres del espacio de tablas, utilice la opción **coalesce** del comando **alter tablespace.**

```
alter tablespace DEMONDX coalesce;
```

Las extensiones libres contiguas serán agrupadas. Se producirá un máximo de ocho agrupamientos, de forma que podrá ejecutar esta instrucción varias veces para agrupar todas las extensiones libres de un espacio de tablas. Puede ejecutar de nuevo el script mapper.sql mostrado anteriormente en este capítulo para ver la nueva distribución de extensiones ocupadas y libres en el espacio de tablas. Si hay muchas extensiones libres localizadas entre extensiones de datos, entonces tendrá que crear de nuevo el espacio de tablas (por ejemplo, exportando e importando sus datos) para poder agrupar las extensiones libres. Si es posible, utilice espacios de tablas gestionados localmente para evitar la necesidad de este tipo de operaciones de desfragmentación.

4.5 Identificación de filas encadenadas

Cuando se crea un segmento de datos, se especifica un valor **pctfree.** El parámetro **pctfree** le indica a la base de datos la cantidad de espacio que se debe mantener libre en cada bloque de datos. El espacio libre se utiliza cuando las filas que ya están almacenadas en el bloque de datos crecen en longitud debido a operaciones de tipo **update.**

Si una fila ya no cabe por completo en un bloque de datos individual debido a una operación de **update,** entonces dicha fila puede ser trasladada o migrar a otro bloque de

datos o, también, la fila puede *encadenarse* a otro bloque. Si está almacenando filas cuya longitud es mayor que el tamaño del bloque Oracle, entonces se producirá un encadenamiento de forma automática (e inevitable). Además, las tablas que contienen más de 255 columnas siempre tendrán filas encadenadas. El encadenamiento afecta al rendimiento porque obliga a que Oracle busque en varias ubicaciones físicas datos de la misma fila lógica. Eliminando el encadenamiento innecesario se reduce el número de lecturas físicas necesarias para devolver datos de un archivo de datos.

Puede evitar la migración estableciendo el valor adecuado para **pctfree** durante la creación de segmentos de datos.

Puede utilizar el comando **analyze** para reunir estadísticas sobre objetos de la base de datos. El optimizador basado en costes puede utilizar estas estadísticas para determinar la mejor ruta de ejecución que se puede utilizar. El comando **analyze** tiene una opción que detecta y registra filas encadenadas en las tablas. Su sintaxis es:

```
analyze tabla NOMBRE_TABLA
list chained rows into FILAS_ENCADENADAS;
```

El comando **analyze** almacenará la salida de esta operación en una tabla llamada CHAINED_ROWS en su esquema local. El código SQL para crear la tabla CHAINED_ROWS está en un archivo llamado utlchain.sql, en el directorio /rdbms/admin del directorio principal de software de Oracle. La siguiente consulta seleccionará las columnas más significativas de la tabla CHAINED_ROWS:

```
select
    Owner_Name,    /*Propietario del segmento de datos*/
    Table_Name,    /*Nombre de la tabla con filas encadenadas*/
    Cluster_Name,/*Nombre del cluster, si la tabla está agrupada en cluster*/
    Head_RowID     /*Identificador de fila de la primera parte de la fila*/
from CHAINED_ROWS;
```

La salida mostrará los identificadores de fila para todas las filas encadenadas, lo que le permite averiguar rápidamente cuántas filas de la tabla están encadenadas. Si el encadenamiento predomina en una tabla, entonces dicha tabla debe reconstruirse con un valor más alto para **pctfree.**

Puede ver el impacto del encadenamiento de filas consultando V$SYSSTAT. La entrada V$SYSSTAT para la estadística «continuación de fila en extracción de tabla>> se incrementará cada vez que Oracle seleccione datos de una fila encadenada. También se incrementará cada vez que Oracle seleccione datos de *una fila extendida* (una fila que está encadenada porque su longitud es mayor que la de un bloque). Las tablas con tipos de datos LONG, BLOB, CLOB y NCLOB son las que tienen mayor probabilidad de tener filas extendidas.

Además de encadenar filas, de vez en cuando Oracle trasladará filas. Si una fila supera el espacio disponible para su bloque, las filas pueden insertarse en un bloque diferente. El proceso de trasladar una fila de un bloque a otro se llama *migración de filas, y* la fila trasladada se denomina *fila migrada.* Durante la migración de filas, Oracle tiene que

gestionar dinámicamente el espacio en varios bloques y acceder a la lista de bloques libres (la lista de bloques disponibles para inserciones). Una fila migrada aparece como una fila encadenada en la tabla de filas encadenadas (CHAI-NED_ROWS) cuando la tabla se analiza para que muestre la lista de filas encadenadas. Las filas encadenadas afectan negativamente al rendimiento de sus transacciones.

4.6 Cómo aumentar el tamaño del bloque de Oracle

El efecto de aumentar el tamaño de bloque de la base de datos es importante. En la mayoría de los entornos, se permiten, al menos, cuatro tamaños de bloque (por ejemplo, 2 KB, 4 KB, 8 KB y 16 KB). La mayoría de las rutinas de instalación se configuran para utilizar tamaños de bloque de 8 KB. No obstante, utilizar el siguiente valor de mayor tamaño para el tamaño de bloque puede mejorar el rendimiento de las operaciones de consulta más críticas hasta en un 50 por 100.

El beneficio en rendimiento tiene pocos costes. Como habrá más filas por bloque de base de datos, hay una mayor probabilidad de contienda a nivel de bloque durante los comandos de manipulación de datos. Para resolver los problemas de contienda, aumente los valores para **freelists, maxtrans e initrans** para tablas e índices. En general, configurar **freelists** a un valor superior a 4 no ofrecerá demasiado beneficio adicional. Los valores **initrans y maxtrans** deben reflejar el número de transacciones simultáneas esperadas dentro de un bloque.

Para aumentar el tamaño del bloque de base de datos, debe reconstruirse la base de datos completa, y todos los archivos antiguos de la base de datos deben borrarse. Los nuevos archivos pueden crearse en la misma ubicación que los antiguos, con el mismo tamaño, pero la base de datos los gestionará con más eficacia. El ahorro en rendimiento se obtiene gracias al modo en que Oracle gestiona la información de la cabecera de bloque. Los datos utilizan más espacio, mejorando así la posibilidad de varios usuarios de acceder al mismo bloque de datos en memoria. Duplicar el tamaño de los bloques Oracle tiene poco efecto sobre la cabecera de bloque; de esta forma se utiliza un porcentaje menor de espacio para almacenar información de cabecera de bloque. Para un entorno Windows NT, un tamaño de bloque de 4 KB se alinea perfectamente con el tamaño de bloque que utiliza NTFS, por lo que se recomienda utilizar dicho tamaño.

El tamaño del bloque de base de datos se especifica durante la creación de la base de datos mediante el parámetro DB_BLOCK_SIZE. El parámetro DB_BLOCK_SIZE puede especificarse en el archivo de parámetros de inicialización. En Oracle9i, puede especificar el tamaño de bloque a nivel de espacio de tablas.

Tenga cuidado cuando configure DB_BLOCK_SIZE si sigue utilizando el parámetro DB_BLOCK_BUFFERS que se requiere para bases de datos anteriores a Oracle9i. DB_BLOCK_BUFFERS, que casi ya no se utiliza en Oracle9i, establece el número de objetos asignados a la caché de buffers de bloques de datos. Como alternativa, las bases de datos Oracle9i deben utilizar el parámetro DB_CACHE_SIZE.

4.7 Utilización de tablas organizadas por índice

Una tabla organizada por índice (IOT, Index Organized Table) es un índice en el que está almacenada una fila completa, no solamente los valores clave de la fila. En lugar de almacenar un identificador para la fila, la clave primaria de la fila se trata como su identificador lógico. Las filas de tablas IOT no tienen identificadores de fila.

Dentro de la IOT, las filas se almacenan ordenadas por sus valores clave primarios. Así, cualquier consulta por rangos basada en la clave primaria puede resultar beneficiada, ya que las filas se almacenan cerca unas de otras. Además, cualquier consulta de equivalencia basada en la clave primaria también se beneficia de esta forma de organización, ya que los datos de la tabla están todos almacenados en el índice. En la combinación tradicional tabla/índice, un acceso basado en índice requiere un acceso al índice seguido de un acceso a la tabla. En una IOT, solamente se accede a la IOT; no hay índice asociado.

No obstante, las mejoras en rendimiento obtenidas de un solo acceso al índice en lugar de un acceso normal de combinación índice/tabla pueden ser mínimas (cualquier acceso mediante índice debería ser rápido). Para que resulte sencillo mejorar aún más el rendimiento, las tablas organizadas por índice ofrecen características adicionales:

- **Un área de desbordamiento.** Configurando el parámetro **pctthreshold** cuando se crea la IOT, se pueden almacenar los datos de la clave primaria además de los datos de fila. Si los datos de fila exceden el umbral de espacio disponible en el bloque, entonces se trasladarán de manera dinámica a un área de desbordamiento. Puede designar que el área de desbordamiento esté en un espacio de tablas distinto, mejorando su capacidad para distribuir la E/S asociada a la tabla.

- **índices secundarios.** Desde Oracle8i, es posible crear índices secundarios en la tabla IOT. Oracle utilizará los valores de clave primaria como identificador lógico para cada fila.

- **Compresión de claves.** Si los mismos datos se repiten en las mismas columnas de varias filas, entonces puede configurar la IOT para que almacene los datos repetidos una vez. Mediante la compresión de claves, Oracle crea una relación de uno a muchos entre los valores de columna únicos y los que están repetidos.

- **Requisitos de almacenamiento reducidos.** En una combinación tradicional tabla/índice, los mismos valores clave se almacenan en dos lugares. En una IOT, se almacenan una vez, reduciendo los requisitos de almacenamiento.

Para crear una IOT, utilice la cláusula **organization index** del comando **create table.** Debe especificar una clave primaria cuando cree una IOT. Dentro de una IOT, puede eliminar columnas o marcarlas como inactivas mediante la cláusula **set unused** del comando **alter table.**

4.7.1 Otros aspectos de la optimización relacionados con tablas organizadas por índice

Al igual que los índices, las IOT pueden fragmentarse internamente con el tiempo a medida que se insertan, actualizan y borran valores. En el siguiente ejemplo, la tabla EMPLOYEE_IOT se ha reconstruido, junto con su área de desbordamiento:

```
alter table EMPLOYEE_IOT
move  tablespace DATA
overflow  tablespace  DATA_OVERFLOW;
```

Debe evitar almacenar filas largas de datos en las IOT. En general, debe evitar utilizar una IOT si los datos son mayores del 75 por 100 del tamaño de bloque de base de datos. Si el tamaño de bloque es 4 KB, y las filas van a tener una longitud superior a 3 KB, debe estudiar el uso de tablas e índices normales en lugar de IOT. Cuanto más largas sean las filas, y más transacciones se realicen con la IOT, más frecuentemente será necesario reconstruirla.

Como hemos mencionado antes en este capítulo, los índices afectan a las velocidades de carga de datos. Para obtener los mejores resultados, el índice de clave primaria de una tabla organizada por índice debe cargarse con valores secuenciales para minimizar los costes de la gestión de índices. Las velocidades de carga de tablas organizadas por índice tienden a ser inferiores a las de las tablas normales.

CAPÌTULO 5
OPTIMIZACIÓN DE LA MANIPULACIÓN DE DATOS

Puede mejorar el rendimiento de las escrituras en base de datos creando varios procesos DBWR (Database Writer: escritor de base de datos). Crear varios procesos DBWR evitará que las peticiones de acceso a varios discos provoquen bloqueos en el rendimiento. El número de procesos DBWR que se deben crear para una instancia se establece mediante el parámetro DB_WRITER_PROCESSES en el archivo de parámetros de inicialización de la base de datos. Si utiliza un solo proceso DBWR, puede crear varios procesos esclavos de E/S asociados a él mediante el parámetro DBWR_IO_SLAVES. Utilice DB_WRITER_PROCESSES si su sistema soporta ASYNC yo (E/S asincrona) y DBWR_IO_SLAVES si no la soporta.

Además de crear procesos esclavos de E/S para DBWR, puede crearlos para los procesos LGWR y ARCH solamente en Oracle8. Los parámetros LGWR_IÓ_SLA-VES y ARCH_IO_SLAVES ya no se soportan desde Oracle8i, por lo que ya no se utilizan. Puede utilizar el parámetro de inicialización LOG_ARCHIVE_MAX_PRO-CESSES desde Oracle8i parta establecer el número de procesos ARCH que se deben iniciar.

5.1 Inserciones masivas: uso de la opción Direct Path de SQL*Loader

Cuando se utiliza en el modo Conventional Path (modo de acceso convencional), SQL*Loader lee un conjunto de registros de un archivo, genera comandos **insert** y los pasa al kernel de Oracle. Oracle busca entonces espacio para dichos registros en los bloques libres de la tabla y actualiza los posibles índices asociados.

En el modo Direct Path (modo de acceso directo), SQL*Loader crea bloques de datos formateados dentro del proceso de servidor y los escribe directamente en los archivos de datos. Esto requiere comprobaciones ocasionales con la base de datos para obtener nuevas ubicaciones de bloques de datos, pero no son necesarias operaciones de E/S adicionales en la base de datos. El resultado es un proceso de carga de datos que es mucho más rápido que en el modo de acceso convencional.

Si la tabla está indexada, entonces los índices pasarán al estado DIRECT PATH durante la carga. Una vez completada la carga, las nuevas claves (valores de columna de índice) se ordenarán y fusionarán con las claves ya existentes del índice. Para mantener el conjunto temporal de claves, el procedimiento de carga creará un segmento de índices temporal que sea, al menos, tan grande como el índice de mayor tamaño de la tabla. Los requisitos de espacio para esta operación pueden minimizarse ordenando previamente el índice y utilizando la cláusula SORTED INDEXES en el archivo de control SQL*Loader.

Para utilizar la opción de acceso directo, se deben crear una serie de vistas en la base de datos. Estas vistas se construyen durante la creación de la base de datos mediante el script catldr.sql, ubicado en $ORACLE_HOME/rdbms/admin.

Para minimizar la cantidad de asignación dinámica de espacio necesaria durante la carga, el segmento de datos que esté cargando ya debe haberse creado, teniendo previamente asignado todo el espacio que vaya a necesitar. También debe ordenar previamente los datos de las columnas del índice de mayor tamaño de la tabla. Ordenar los datos y conservar los índices asociados a la tabla durante una carga en modo de acceso directo

producirá normalmente un mejor rendimiento que si borrara los índices antes de la carga y los volviera a crear después al terminar.

Para aprovechar la opción de acceso directo, la tabla no puede estar agrupada en cluster, y no puede haber otras transacciones activas operando sobre ella. Durante la carga, solamente se llevará el control de las restricciones NOT NULL, UNIQUE y PRIMARY KEY; una vez finalizada, las restricciones CHECK y FOREIGN KEY pueden volver a activarse automáticamente. Para hacer que esto ocurra, utilice la cláusula `en el` archivo de control SQL*Loader:

```
REENABLE  DISABLED_CONSTRAINTS
```

La única excepción de este proceso de reactivación es que los disparadores de inserción de tabla, cuando se reactivan, no se ejecutan automáticamente para cada una de las filas nuevas que se hayan insertado en la tabla. Un proceso independiente debe ejecutar manualmente los comandos que este tipo de disparador tendría que haber ejecutado.

La opción de carga en modo de acceso directo de SQL*Loader proporciona importantes mejoras de rendimiento con respecto a la carga en modo de acceso convencional de SQL*Loader en la carga de datos en tablas Oracle evitando el paso por los comandos SQL, la gestión de la caché de buffers y lecturas innecesarias de los bloques de datos. La opción Parallel Data Loading (carga de datos en paralelo) de SQL*Loader permite que varios procesos de carga trabajen en la carga de la misma tabla, utilizando recursos de reserva del sistema, y reduciendo así el tiempo total de carga necesario. Si se dispone de suficiente capacidad de proceso y recursos de E/S, se pueden reducir muy significativamente esos tiempos totales de carga.

Para utilizar la carga de datos en paralelo, inicie varias sesiones de SQL*Loader utilizando la palabra clave **parallel** (en caso contrario, SQL*Loader aplica a la tabla un bloqueo exclusivo). Cada sesión es una sesión independiente que necesita su propio archivo de control. El siguiente listado muestra tres cargas en modo de acceso directo distintas, que utilizan el parámetro PARALLEL=TRUE en la línea de comandos:

```
sqlload USERID=ME/PASS CONTROL=PART1.CTL DIRECT=TRUE
PARALLEL=TRUE

sqlload USERID=ME/PASS CONTROL=PART2.CTL DIRECT=TRUE
PARALLEL=TRUE

sqlload USERID=ME/PASS CONTROL=PART3.CTL DIRECT=TRUE
PARALLEL=TRUE
```

Cada sesión crea sus propios archivos de registro (log), de registros no válidos (bad) y de registros descartados (discard) (partl.log, part2.log, part3.log, part1.bad, part2.bad, etc.) de manera predeterminada. Como hay varias sesiones cargando datos en la misma tabla, solamente se permite la opción APPEND para carga de datos en paralelo. Las opciones REPLACE, TRUNCATE e INSERT de SQL*Loader no están permitidas para la carga de datos en paralelo. Si necesita borrar los datos de la tabla antes de iniciar la carga, debe

borrarlos manualmente (con los comandos **delete o truncate).** No puede utilizar SQL*Loader para borrar los registros automáticamente si está utilizando carga de datos en paralelo.

Si utiliza carga de datos en paralelo, la sesión SQL *Loader no hace el tratamiento de los índices. Antes de iniciar el proceso de carga, debe borrar todos los índices de la tabla y desactivar todas las restricciones de tipo PRIMARY KEY y UNIQUE. Una vez estén terminadas las cargas, puede crear de nuevo los índices de la tabla.

En la carga en modo de acceso directo en serie (PARALLEL=FALSE), SQL*Loader carga los datos en extensiones de la tabla. Si el proceso de carga falla antes de que la carga quede completada, algunos datos pueden haber quedado confirmados **(commit)** en la tabla antes del fallo del proceso. En la carga de datos en paralelo, cada proceso de carga crea segmentos temporales para cargar los datos. Los segmentos temporales se combinan después con la tabla. Si un proceso de carga de datos en paralelo falla antes de terminar la carga, los segmentos temporales no se habrán combinado con la tabla. Si los segmentos temporales no se han combinado con la tabla que se está cargando, entonces ningún dato de la carga se habrá confirmado en la tabla.

Puede utilizar el parámetro FILE de SQL*Loader para hacer que cada sesión de carga de datos utilice un archivo de datos distinto. Haciendo que cada sesión de carga utilice su propio archivo de datos, se puede equilibrar la carga de operaciones de E/S de los procesos de carga. La carga de datos hace una utilización muy intensa de los recursos de E/S y debería distribuirse entre varios discos para que la carga en paralelo logre unas mejoras significativas en rendimiento con respecto a la carga en serie.

Tras una carga de datos en paralelo, cada sesión puede intentar reactivar las restricciones de la tabla. Siempre que al menos una sesión de carga esté en progreso, el intento de reactivar las restricciones fallará. La última de las sesiones de carga en terminar debería ser la que tenga éxito en la reactivación de las restricciones. Debe revisar el estado de sus restricciones tras completar la carga. Si la tabla que se cargue tiene restricciones PRIMARY KEY y UNIQUE, puede crear los índices asociados en paralelo antes de activar las restricciones.

5.2 Inserciones masivas: problemas habituales y trucos adecuados

Si los datos no están siendo insertados partiendo de un archivo sin formato, SQL*Loader no será una solución útil. Por ejemplo, si necesita mover un conjunto de datos de gran tamaño de una tabla a otra, probablemente deseará evitar tener que escribir los datos en un archivo sin formato y después leerlos de nuevo en la base de datos. La forma más rápida de mover datos en su base de datos es moverlos de una tabla a otra sin pasar por el sistema operativo.

Cuando mueva datos de una tabla a otra, hay cuatro métodos habituales para mejorar el rendimiento de la migración de datos:

- Optimizar las estructuras (eliminando índices y disparadores).
- Desactivar las restricciones durante la migración de datos.
- Utilizar indicaciones y opciones para mejorar el rendimiento de la transacción.
- Aislar los segmentos de anulación para la transacción de gran tamaño.

El primero de los cuatro, optimizar las estructuras, implica la desactivación de cualquier disparador o índice que haya en la tabla en la que se están cargando los datos. Por ejemplo, si tiene un disparador a nivel de fila en la tabla de destino, ese disparador se ejecutará para cada fila insertada en la tabla. Si es posible, desactive los disparadores antes de la carga de datos. Si el disparador debe ejecutarse para cada fila insertada, entonces quizás pueda hacer una operación masiva una vez que las filas se hayan insertado, en lugar de repetir la operación durante cada inserción. Si está adecuadamente optimizada, esa operación masiva se realizará más rápido que las ejecuciones repetidas del disparador. Tendrá que asegurarse de que las operaciones masivas se ejecutan para todas las filas que no hayan sido procesadas por los disparadores.

Además de desactivar los disparadores, debe borrar los índices de la tabla de destino antes de iniciar la carga de datos. Si los índices se dejan en la tabla, Oracle gestionará de manera dinámica los índices a medida que cada fila se inserta. En lugar de tratar de manera continua el índice, bórrelo antes del inicio de la carga y créelo de nuevo cuando la carga haya finalizado.

Borrar índices y desactivar disparadores resuelve la mayoría de los problemas de rendimiento asociados a operaciones de migración de datos de gran tamaño entre tablas.

Además de desactivar los índices, debe tener en cuenta la desactivación de restricciones sobre la tabla. Si los datos de origen ya están en una tabla de la base de datos, puede verificar si cumplen esas restricciones (restricciones de tipo CHECK o de clave externa) antes de cargarlos en la tabla de destino. Una vez que los datos se han cargado, reactive las restricciones.

Si ninguna de esas opciones le proporciona un rendimiento adecuado, debe estudiar las opciones que Oracle ha introducido para la optimización de la migración de los datos. Dichas opciones incluyen:

- **La indicación APPEND para comandos insert.** Al igual que en la carga en modo de acceso directo, la indicación APPEND carga bloques de datos en una tabla, empezando en el límite superior de la tabla (véase el Capítulo 4). El uso de la indicación APPEND puede aumentar la ocupación del espacio.

- **La opción nologging.** Si está ejecutando un comando **create table as select,** utilice la opción **nologging** para evitar la escritura en los registros de reconstrucción durante la operación.

- **Las opciones de paralelización.** Las opciones de consulta en paralelo utilizan varios procesos para realizar una sola tarea. Para un comando **create table as select,** puede ejecutar en paralelo la parte de la creación de la tabla **(create table)** por un lado y la consulta por otro. Si utiliza las opciones de paralelización, debe emplear también la opción **nologging;** si no lo hace así, las operaciones que se ejecuten en paralelo tendrán que esperar, debido a las escrituras en serie realizadas en los archivos de los registros de reconstrucción en línea.

Antes de utilizar cualquiera de estas tres opciones avanzadas, primero debe estudiar la estructura de la tabla de destino para asegurarse de que ha evitado los problemas habituales citados previamente en esta sección.

El cuarto método para mejorar el rendimiento, aislar la actividad del segmento de anulación para la transacción, puede requerir la creación de un nuevo espacio de tablas. Por ejemplo, puede crear un nuevo espacio de tablas para segmentos de anulación y crear uno o varios segmentos de anulación de gran tamaño dentro de él. Los archivos de datos asociados a dicho espacio de tablas deben situarse en discos que están aislados del resto de la base de datos. Cree un segmento de anulación para cada una de las transacciones de migración de datos que vayan a ejecutarse simultáneamente. A continuación puede utilizar el comando **set transaction use rollback segment** para obligar a la transacción a utilizar el nuevo segmento de anulación. Si no puede utilizar este comando, entonces tendrá que desconectar el resto de los segmentos de anulación antes de iniciar las transacciones de migración de datos de gran tamaño. Para minimizar el tamaño de los segmentos de anulación requeridos, realice operaciones **commit** con frecuencia durante las transacciones.

También puede utilizar lógica de programación para hacer que el proceso de inserción se lleve a cabo por matrices en lugar de como un conjunto completo. Por ejemplo, COBOL y C soportan inserciones por matrices, reduciendo el tamaño de las transacciones necesarias para procesar un conjunto de datos de gran tamaño.

5.3 Borrados masivos: el comando Truncate

De vez en cuando, los usuarios intentan borrar todos los registros de una tabla al mismo tiempo. Cuando encuentran errores durante este proceso, se quejan de que los segmentos de anulación son demasiado pequeños cuando lo que sucede en realidad es que su transacción es demasiado grande.

Un segundo problema se produce una vez que se han borrado todos los registros. Aunque el segmento ya no tiene registros, sigue manteniendo bajo su control todo el espacio que tenía asignado. Así, borrar todos esos registros no permite ahorrar ni un solo byte del espacio asignado.

El comando **truncate** resuelve ambos problemas. Es un comando DDL, no un comando DML, de modo que *no se puede anular* mediante **rollback.** Una vez que haya utilizado el comando **truncate** en una tabla, sus registros habrán desaparecido, y ninguno de sus disparadores de borrado se habrá ejecutado en el proceso. No obstante, la tabla conserva todos los objetos asociados a ella, tales como concesiones de privilegios, índices y restricciones.

El comando **truncate** es la forma más rápida de borrar grandes volúmenes de datos. Como borrará todos los registros de una tabla, esto quizá le obligue a modificar el diseño de su aplicación, de forma que ningún registro que deba ser protegido se almacene en la misma tabla que los registros que se van a borrar. Si utiliza particiones, puede truncar una partición de una tabla sin afectar al resto de las particiones.

En este listado se muestra un comando **truncate** de ejemplo para una tabla:

```
truncate table EMPLOYEE drop storage;
```

El ejemplo anterior, en el que se han borrado los registros de la tabla EMPLOYEE, muestra una potente función de **truncate.** La cláusula **drop storage** se utiliza para liberar el espacio de la tabla no incluido en la extensión inicial (initial) (es la opción predeterminada). Así, puede borrar todas las filas de la tabla y reclamar todo el espacio reservado excepto el de la extensión inicial, sin eliminar la tabla propiamente dicha.

El comando **truncate** funciona también para las agrupaciones en cluster. En el siguiente ejemplo se utiliza la opción **reuse storage** para liberar todo el espacio asignado dentro del segmento del que lo adquirió:

```
truncate cluster EMP_DEPT reuse storage;
```

Cuando este comando de ejemplo se ejecute, todos los registros del cluster EMP_DEPT se borrarán instantáneamente.

Para truncar particiones, debe conocer el nombre de la partición. En el siguiente ejemplo, la partición llamada PART3 de la tabla EMPLOYEE se ha truncado mediante el comando **alter table:**

```
alter table EMPLOYEE
truncate partition PART3
drop storage;
```

El resto de las particiones de la tabla EMPLOYEE no se verá afectado por el truncado de la partición PART3.

Como alternativa, puede crear un programa PL/SQL que utilice SQL dinámico para dividir una gran operación **delete** en varias transacciones más pequeñas.

Puede utilizar particiones para aislar los datos físicamente. Por ejemplo, puede almacenar los datos de un departamento en una partición distinta de la tabla EMPLOYEE. Si realiza una carga o borrado de datos masivo en la tabla, puede personalizar las particiones para ajustar la operación de manipulación de datos. Por ejemplo:

- Puede truncar una partición y sus índices sin afectar al resto de la tabla.
- Puede borrar una partición mediante la cláusula **drop partition** del comando **alter table.**
- Puede borrar el índice local de una partición.
- Puede configurar una partición como **nologging,** reduciendo así el impacto de inserciones de gran tamaño.

Desde el punto de vista del rendimiento, la ventaja principal de las particiones reside en su capacidad para ser gestionadas con independencia del resto de la tabla. Por ejemplo, poder truncar una partición permite borrar una gran cantidad de datos de una tabla (pero no todos los datos de la tabla) sin generar ninguna información de reconstrucción. A corto plazo, el beneficiario de esta mejora en el rendimiento es el DBA; a largo plazo, la empresa entera se beneficia de la disponibilidad mejorada de los datos.

CAPÌTULO 6
OPTIMIZACIÓN DEL ALMACENAMIENTO

6.1 Optimización del almacenamiento físico

Las operaciones de E/S física asociadas a las bases de datos deben distribuirse por igual y gestionarse correctamente. Planificar la distribución de los discos implica la comprensión de las interacciones de los procesos en segundo plano DBWR, LGWR y ARCH. En el Capítulo 4 también se ofrece un mecanismo para verificar la validez de los diseños finales.

Además de ese nivel de Optimización del almacenamiento físico, se deben tener en cuenta otros factores. Las siguientes secciones resuelven factores que son externos a la base de datos pero que pueden tener un profundo impacto sobre su capacidad para acceder rápidamente a los datos.

6.1.1 Utilización directa de los dispositivos

Algunos sistemas operativos UNIX proporcionan mecanismos de utilización *directa de los dispositivos (raw devices).* Cuando se utilizan los dispositivos directamente, el proceso DBWR no hace uso de la caché de buffers de UNIX y elimina la sobrecarga de trabajo provocada por el uso del sistema de archivos. Para aplicaciones que hacen un uso intensivo de operaciones de E/S, la utilización directa de los dispositivos puede dar como resultado una mejora en el rendimiento de, aproximadamente, el 2 por 100 con respecto a sistemas de archivos tradicionales. Las recientes mejoras en los sistemas de archivos han superado en gran parte esta diferencia de rendimiento, y es creencia general que la carga de trabajo adicional que supone el tener que gestionar la utilización directa de los dispositivos no compensa ningún beneficio que pueda obtenerse en el rendimiento.

La utilización directa de los dispositivos no puede gestionarse con los mismos comandos que los sistemas de archivos. Por ejemplo, el comando tar no puede utilizarse para realizar la copia de seguridad de archivos individuales, para lo cual se debe emplear el comando **dd.** Se trata de un comando cuya utilización es mucho menos flexible y que limita las posibilidades de recuperación. El acceso directo a los dispositivos se utiliza habitualmente en entornos que dan soporte a Oracle Real Application Cluster (antes llamado Oracle Parallel Server), pero con los avances obtenidos en las tecnologías de acceso a disco, estos clusters pueden no requerir la utilización directa de los dispositivos en un futuro muy cercano.

Los archivos de Oracle no deben residir en los mismos dispositivos físicos que los archivos que no son de Oracle, especialmente si se hace utilización directa de los dispositivos. Mezclar un sistema de archivos UNIX activo con un dispositivo que está siendo directamente accedido por Oracle también activo provocará problemas de rendimiento de la E/S.

6.1.2 Tecnologías RAID y de duplicación en espejo

En general, RAID 0+1 producirá el mejor rendimiento, mientras que RAID-5 es el que ofrece la implemen-tación más barata. El beneficio en rendimiento derivado de una implementación RAID está directamente relacionado con el modo en que el sistema sea implementado.

6.2 Optimización del almacenamiento lógico

Desde un punto de vista lógico, los objetos similares deben almacenarse juntos. Los objetos se deben agrupar según su utilización del espacio y las características de su interacción con el usuario. Basándose en estas agrupaciones, se deben crear espacios de tablas que den servicio a determinados tipos de objetos.

Las vistas materializadas pueden utilizarse para agregar datos y mejorar el rendimiento de las consultas. Una vista materializada es idéntica en estructura a una instantánea (es una tabla física que contiene datos que normalmente se leerían mediante una vista). Al crear una vista materializada, se especifica la consulta base de la vista así como un esquema para la actualización de su contenido. A continuación se puede indexar la vista materializada para mejorar el rendimiento de las consultas efectuadas contra ella. Como resultado de todo esto, puede proporcionar datos a sus usuarios en el formato en que los necesiten, adecuadamente indexados.

CAPÌTULO 7
REDUCCIÓN DEL TRÁFICO DE RED

A medida que las bases de datos y las aplicaciones que las utilizan se van distribuyendo, la red que soporta los servidores puede convertirse en un cuello de botella en el proceso de entrega de datos al usuario. Reducir el tráfico de red disminuirá también su dependencia de la red, eliminando así una posible causa de problemas de rendimiento.

7.1 Duplicación de datos

Es posible manipular y consultar datos desde bases de datos remotas. No obstante, no resulta deseable que grandes volúmenes de datos se deban enviar constantemente de una base de datos a otra. Para reducir la cantidad de datos que se envían a través de la red, se deben tener en cuenta distintas opciones de duplicación de datos.

En un entorno puramente distribuido, cada dato existe en un solo lugar, como se muestra en la Figura 7.1. Cuando se necesitan datos, se accede a ellos desde bases de datos remotas mediante enlaces de base de datos. En el ejemplo mostrado en la Figura 8.1, los datos EMPLOYEE se han consultado desde la base de datos MÁSTER 1, y los datos DEPT se han consultado desde la base de datos REMOTE 1. Es posible acceder a ambas base de datos desde enlaces de base de datos creados dentro de la base de datos REMOTE 1.

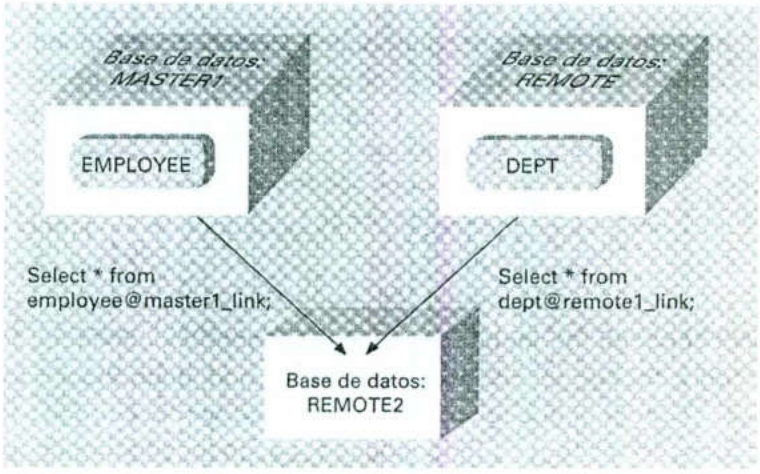


Fig. 7.1 Entorno distribuido de ejemplo.

Este enfoque purista (tener datos almacenados en un único lugar) es similar a implementar una aplicación en la tercera forma normal, y como se afirmó previamente en este capítulo, dicho enfoque no permitirá dar soporte a ninguna de aplicación de producción de cierta importancia. Modificar las tablas de la aplicación para mejorar el rendimiento de recuperación de datos implica desnormalizar los datos. El proceso de desnormalización almacena de manera deliberada datos redundantes con el fin de simplificar las rutas de acceso a los datos por parte de los usuarios.

En un entorno distribuido, la duplicación de datos cumple este objetivo. En lugar de obligar a las consultas a cruzar la red para resolver las peticiones de los usuarios, los datos seleccionados de servidores remotos se duplican en el servidor local. Esto se puede realizar utilizando distintas técnicas, tal y como se describe en las siguientes secciones.

7.1.1 El comando copy para duplicar datos

En la primera opción, los datos pueden copiarse periódicamente al servidor local. El mejor modo de realizar esto es utilizar el comando **copy** de SQL*Plus, según se describe en la Parte III. El comando **copy** permite duplicar columnas y filas seleccionadas en cada servidor. Esta opción se ilustra en la Figura 7.2.

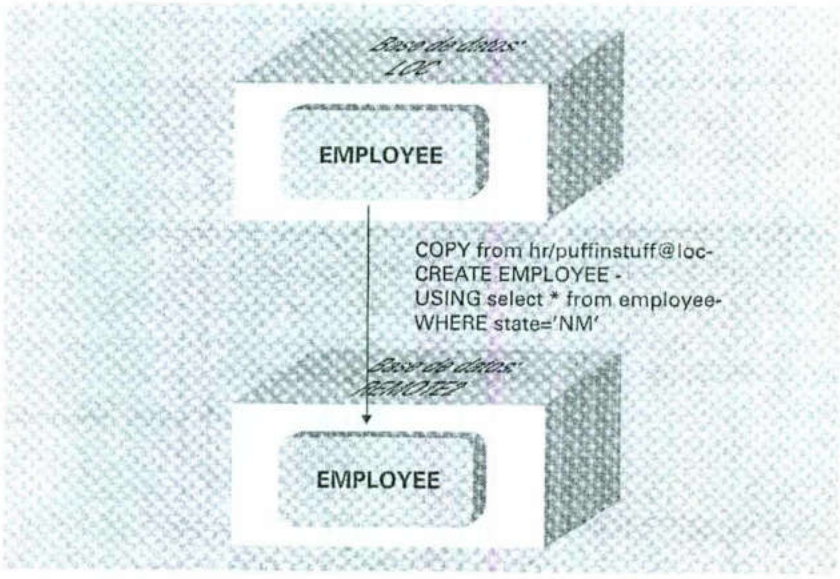


Fig. 7.2 Duplicaciòn de datos utilizando el comando copy.

Por ejemplo, el servidor remoto puede tener una tabla llamada EMPLOYEE. El servidor local podría duplicar los datos que necesita utilizando el comando **copy** para seleccionar los registros de la tabla EMPLOYEE remota. Puede utilizar el comando **copy** para almacenar dichos registros seleccionados en una tabla de la base de datos local. El comando **copy** incluye una cláusula de consulta, de modo que es posible devolver solamente las filas que cumplan los criterios especificados.

En este ejemplo, una parte de la tabla EMPLOYEE se copia de la base de datos principal a una tabla local. Se utiliza una cláusula **where** para restringir los registros que son seleccionados.

```
set copycoinmit 1
set arraysize 1000
copy from HR/PUFFINSTUFF@loc -
create EMPLOYEE -
using -
select * from EMPLOYEE -
where State = NM
```

La cláusula **copy from** de este ejemplo especifica el nombre de la base de datos remota. En este caso, se indica a la consulta que utilice la base de datos identificada por el nombre de servicio LOC. Durante la conexión se debe iniciar una sesión utilizando la cuenta hr, con la contraseña puffinstuff.

Los comandos **set copycommit y set arraysize** especifican el tamaño de la matriz de datos. Configurar el tamaño de la matriz permite al DBA obligar a la base de datos a hacer operaciones de **commit** durante la copia de los datos, reduciendo así el tamaño de las transacciones que se deben soportar. Para obtener más información sobre esta función y otras opciones de **copy,** consulte la Parte III.

En cuanto los datos se almacenen localmente, estarán accesibles para los demás usuarios. Así pueden consultarlos sin cruzar toda la red; se realiza un único acceso a la red durante el comando copy en lugar de accesos individuales para cada consulta.

El inconveniente de duplicar datos de esta forma es que los datos resultantes se quedan obsoletos en el mismo momento en que se crean. Por esta razón, duplicar datos con el objetivo de mejorar el rendimiento es más eficaz cuando los datos de origen cambian con muy poca frecuencia. El comando **copy** debe ejecutarse con la suficiente frecuencia como para que las tablas locales contengan datos útiles y tan precisos como sea necesario. La opción **replace** del comando **copy** puede utilizarse para reemplazar el contenido de las tablas locales para comandos **copy** subsiguientes.

Aunque la tabla local pueda actualizarse, ninguno de los cambios realizados en ella se reflejará en la tabla de destino. Así, esta situación solamente resulta efectiva para mejorar el rendimiento de las operaciones de consulta. Si necesita poder actualizar los datos locales y enviar los cambios en los datos de nuevo a la base de datos principal, entonces tendrá que utilizar algunas de las opciones de duplicación avanzadas de las que se dispone en Oracle. Oracle da soporte a configuraciones multimaestro, además de a vistas materializadas de sólo lectura y actualizables.

7.1.2 Uso de vistas materializadas o instantáneas en la duplicación de datos

Las funciones de procesamiento distribuido de Oracle ofrecen distintas formas de gestionar la duplicación de datos dentro de una base de datos. Las vistas materializadas, conocidas en anteriores versiones como instantáneas, duplican datos desde un origen principal a varios destinos. Oracle ofrece herramientas para refrescar los datos y actualizar los destinos a determinados intervalos de tiempo. La Figura 7.3 ilustra una opción de entorno distribuido.

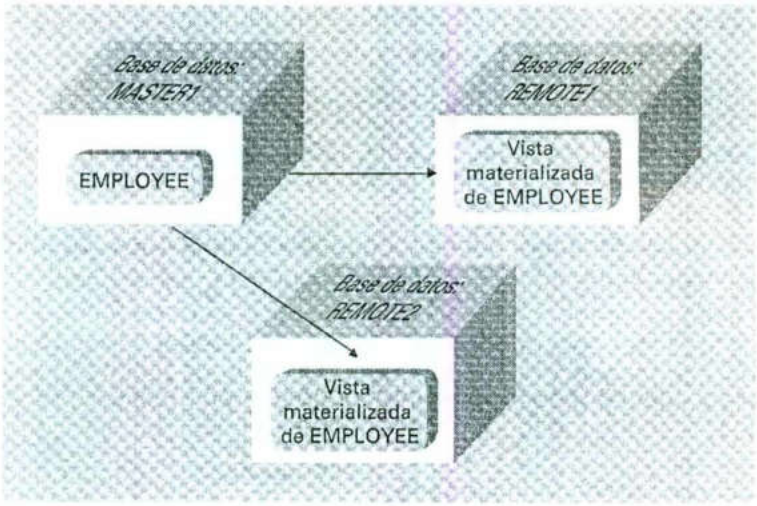


Fig. 7.3 Duplicaciòn de datos utilizando vistas materializadas.

Las vistas materializadas pueden ser de sólo lectura o actualizables. Antes de crear una vista materializada se debe establecer un enlace con la base de datos de origen. El siguiente ejemplo crea un enlace de base de datos privado llamado HR_LINK utilizando el nombre de servicio LOC:

```
Create database link HR_LINK
connect to HR identified by PUFFINSTUFF
using 'loc';
```

El comando **create database link,** tal y como se muestra en este ejemplo, tiene varios parámetros:

- El nombre del enlace (HR_LINK, en este ejemplo).
- La cuenta a la que conectarse.
- El nombre de servicio de la base de datos remota (tal y como se halla en el archivo tnsnames.ora del servidor). En este caso, el nombre de servicio es LOC.

Hay dos estilos disponibles de vistas materializadas: *simple y compleja.* El tipo adecuado que debe utilizar para su entorno depende de la cantidad de datos duplicados y del modo en que se consultan. El tipo de vistas materializadas empleado afecta a las opciones de refresco de datos que van a estar disponibles.

El tipo de vista materializada se determina mediante la consulta que la define. Una vista materializada simple se basa en una consulta que no contiene cláusulas **group by,** cláusulas **connect by,** uniones (join) u operaciones de conjuntos. Una vista materializada compleja contiene al menos una de estas opciones. Por ejemplo, una vista materializada basada en la consulta:

```
select  *   from EMPLOYEE@HR_LINK;
```

sería simple, mientras que una basada en la consulta:

```
select  DEPT,MAX(Salary)
from EMPLOYEE@HR_LINK
group by DEPT;
```

sería compleja, porque utiliza funciones de agregación.

La sintaxis empleada para crear la vista materializada en el servidor local se muestra en el siguiente listado. En este ejemplo, a la vista materializada se le asigna un nombre (LOCAL_EMP) y se especifican sus parámetros de almacenamiento. Se proporciona su consulta base, así como su intervalo de refresco. En este caso, se le indica a la vista materializada que recupere inmediatamente los datos principales, para realizar después la operación de refresco cada siete días (SysDate+7).

```
create materialized view LOCAL_EMP
pctfree 5
tablespace data_2
storage (initial 100K next 100K pctincrease 0)
refresh fast
      start with SysDate
      next SysDate+7
as select * from EMPLOYEE@HR_LINK;
```

La cláusula **refresh fast** (refresco rápido) le indica a la base de datos que utilice un registro de vista materializada para refrescar la vista materializada local. La posibilidad de utilizar registros de vistas materializadas durante los refrescos solamente está disponible con vistas materializadas simples. Cuando se utiliza un registro de vista materializada, solamente los cambios efectuados en la tabla principal son enviados a las tablas de destino. Si utiliza una vista materializada compleja, entonces se debe emplear la cláusula **refresh complete** (refresco completo) en lugar de **refresh fast.** En un refresco completo, cada operación de refresco reemplaza por completo los datos existentes en la tabla subyacente asociada a la vista materializada.

Los registros de vistas materializadas se deben crear en la base de datos principal mediante el comando **create materialized view log.** En el siguiente listado se muestra un ejemplo de este comando:

```
create materializad view log on EMPLOYEE
tablespace DATA
storage (initial 10K next 10K pctincrease 0);
```

El registro de vista materializada debe crearse en el mismo esquema que la tabla principal.

Puede utilizar vistas materializadas simples con registros de vistas materializadas para reducir el de tráfico de red implicado en el mantenimiento de los datos duplicados. Como solamente se envían los cambios efectuados en los datos mediante un registro de vista materializada, el mantenimiento de vistas materializadas simples debe utilizar menos recursos de red de los que requieren las vistas materializadas complejas, especialmente si las tablas principales son de gran tamaño y bastante estáticas. Si las tablas principales no son estáticas, entonces el volumen de las transacciones enviadas mediante el registro de vista materializada puede no ser menor de las que se enviarían para realizar un refresco completo.

La duplicación también juega un papel importante en el diseño de la aplicación. Si sus rutas de acceso a los datos requieren la combinación de información de varias tablas remotas, entonces tiene dos opciones, como muestra la Figura 7.4. La primera opción (Figura 7.4a) es crear varias vistas materializadas simples y realizar después la consulta de unión (join) en el servidor local. La segunda opción (Figura 7.4b) es crear una sola vista materializada compleja *en* el servidor local basándose en varias tablas remotas.

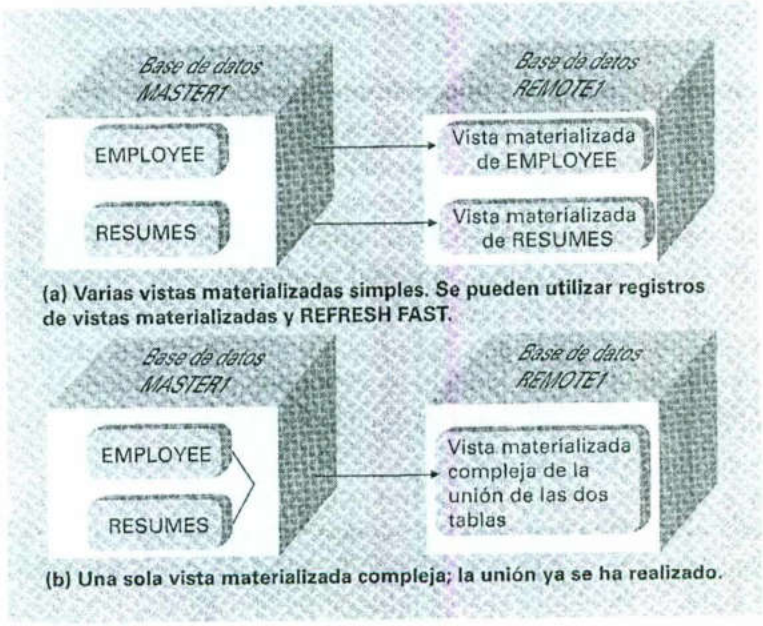


Fig. 7.4 Opciones de duplicación de datos para uniones.

¿Qué opción recuperará más rápido los datos? La respuesta depende de varios factores:

- **Tamaño de las tablas principales.** ¿Cuánto tiempo hace falta para llevar a cabo un refresco completo?

- **Volumen de las transacciones en las tablas principales.** ¿Cómo de grandes son los registros de vistas materializadas?
- **Frecuencia de los refrescos.** ¿Con qué frecuencia se duplicarán los datos?

Si los datos se refrescan con poca frecuencia, y hay pocas transacciones en las tablas principales, entonces sería más rápido utilizar una vista materializada compleja (véase la Figura 8.4b). Si los datos se actualizan y refrescan frecuentemente, entonces el ahorro en tiempo que se obtiene de utilizar refrescos rápidos debería pesar más que el coste de realizar la unión cuando se ejecuta la consulta (en lugar de hacerlo con antelación mediante la vista materializada). En ese caso, utilizar una serie de vistas materializadas simples daría como resultado un tiempo de respuesta más rápido.

En general, un refresco rápido funcionará mejor que un refresco completo si menos del 25 por 100 de las filas han cambiado. Si ha cambiado un porcentaje mayor de filas, entonces debe tener en cuenta la utilización de un refresco completo. Aunque en ese caso un refresco completo podría ser más rápido, generará un volumen mayor de tráfico de red que un refresco rápido.

La optimización que se busca con la duplicación de datos es minimizar el tiempo necesario para satisfacer la petición de datos remotos por parte del usuario. La decisión sobre el tipo adecuado de configuración de vista materializada solamente puede tomarse si se conocen las uniones más habituales con antelación. Para obtener más información sobre la gestión de vistas materializadas y su impacto sobre la optimización de consultas en almacenes de datos, consulte el Capítulo 15.

7.2 Utilización de llamadas a procedimientos remotos

Cuando utilice procedimientos en un entorno de base de datos distribuido, hay dos opciones: crear un procedimiento local que haga referencia a tablas remotas o crear un procedimiento remoto que sea llamado por una aplicación local. Estas dos opciones se ilustran en la Figura 7.5.

La ubicación adecuada del procedimiento depende de la distribución de los datos y del modo en que éstos se van a utilizar. El énfasis debe ponerse en minimizar la cantidad de datos que se deben enviar a través de la red para resolver la petición de datos. El procedimiento debe residir dentro de la base de datos que contiene la mayor parte de los datos que se utilizan durante las operaciones del procedimiento.

Por ejemplo, consideremos este procedimiento:

```
createprocedure MY_RAISE (My_Emp_No IN NUMBER, Raise IN
NUMBER) as begin
      update EMPLOYEE@HR_LINK
      set Salary = Salary+Raise
      where Empno = My_Emp_No;
end;
```

En este caso, el procedimiento solamente accede a una sola tabla (EMPLOYEE) en un nodo remoto (como indica el enlace de base de datos HR_LINK). Para reducir la cantidad de datos enviados a través de la red, lleve este procedimiento a la base de datos remota identificada por el enlace de base de datos HR_LINK y elimine la referencia a dicho enlace en la cláusula **update** del procedimiento. A continuación, llame al procedimiento desde la base de datos local utilizando el enlace de base de datos, tal y como se muestra en el siguiente listado:

```
execute MY_RAISE@HR_LINK(1234,2000) ;
```

En este caso, se pasan dos parámetros al procedimiento (My_Emp_No que toma el valor 1234 y Raise que toma 2000). El procedimiento se invoca utilizando un enlace de base de datos para indicarle a la base de datos dónde puede encontrar el procedimiento.

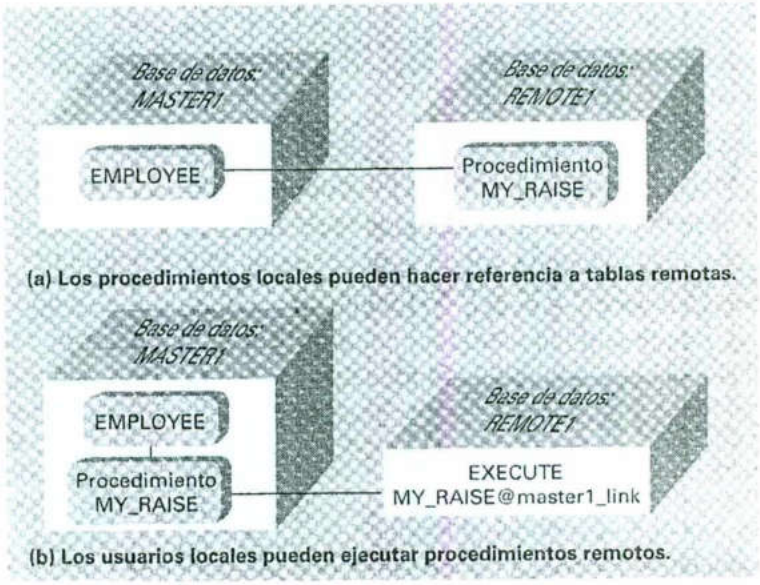


Fig. 7.5 Opciones para la ubicación de procedimientos.

El beneficio de optimización que se obtiene al ejecutar un procedimiento remoto es que todo el proceso del procedimiento se realiza en la base de datos en la que residen los

datos. La llamada al procedimiento remoto minimiza el tráfico de red necesario para completar el procedimiento.

Para mantener la transparencia de la ubicación, puede crear un sinónimo local que apunte al procedimiento remoto. El nombre del enlace de base de datos se especificará en el sinónimo, de manera que las peticiones de usuario utilizarán automáticamente la base de datos remota:

```
create synonym MY_RAISE for MY_RAISE@HR_LINK;
```

Un usuario podría introducir entonces el comando:

```
execute MY_RAISE(1234, 2000} ;
```

y ejecutaría el procedimiento remoto definido por el sinónimo MY_RAISE.

CAPÌTULO 8
USO DE OEM Y DE LOS PAQUETES DE OPTIMIZACIÓN DEL RENDIMIENTO

Oracle proporciona tres paquetes que pueden utilizarse junto con Oracle Enterprise Manager (OEM) para llevar el control de la optimización del rendimiento y la gestión de cambios, y para generar diagnósticos del sistema y de la base de datos.

Se puede conectar con la consola OEM de dos formas: como sesión autónoma o mediante el servidor OMS (Oracle Management Server, servidor de administración de Oracle). Dependiendo de la forma de conexión, la opción Tuning Pack (paquete de optimización del rendimiento) ofrece distintas posibilidades. Por ejemplo, si conecta en modo autónomo, dispone de cuatro opciones: Oracle Expert (modo experto de Oracle), Outline Management (gestión de bocetos), SQL Analyze (análisis de SQL) y Tablespace Map (mapa de espacios de tabla). En OMS puede elegir entre las cuatro opciones de la versión autónoma y además Index Tuning Wizard (asistente para optimización de índices) y Reorg Wizard (asistente de reorganización).

Examinaremos a continuación la opción Oracle Expert. Puede configurar la opción Oracle Expert para que recopile diversos tipos de información, que haga sugerencias sobre las etapas de optimización y que cree los scripts SQL para implementar las sugerencias de optimización.

Dentro del paquete de diagnósticos, se ofrecen las opciones Performance Management (administración del rendimiento) y Performance Overview (panorámica del rendimiento), como se describe en la sección <<Oracle Performance Manager>> más adelante.

8.1 El paquete Oracle Expert

Cuando accede a la opción Oracle Expert por primera vez, la única opción que se le ofrece es cargar una sesión de ejemplo de optimización, a menos que haya establecido previamente sus credenciales preferidas utilizando la opción Preferences (preferencias) de la pantalla Configuration Option (opciones de configuración) del administrador de consola de OEM. Una vez establecidas las credenciales preferidas, dispondrá de las opciones de cargar la sesión de ejemplo o crear una sesión nueva. La Figura 8.1 muestra la pantalla inicial del asistente Tuning Session Wizard con la opción <<Createa new tuning session>> (crear una nueva sesión de optimización) seleccionada.

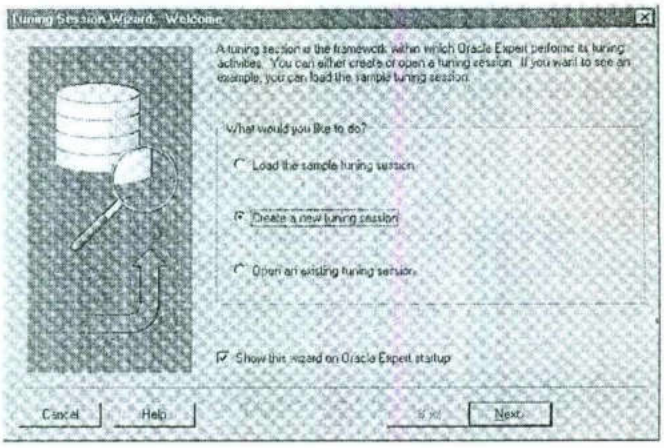


Fig. 8.1 Pantalla de bienvenida del asistente de sesiòn de optimizaciòn.

Después de haber creado al menos una sesión, también se le ofrece la opción de conectarse a una sesión existente. Para crear una nueva sesión de optimización debe introducir el nombre de la base de datos y un nombre para la sesión. El nombre empleado para este ejemplo es <<General tuning 1>> en la base de datos mydb9.world.

Las opciones asociadas a la configuración de una nueva sesión son Scope (ámbito), Collect (criterios de recopilación), Review (revisión), Recommendations (recomendaciones) y Scripts. El primer paso en la configuración de una sesión nueva es establecer el ámbito de las áreas en las que desee realizar la optimización. La Figura 8.2 muestra la pantalla inicial de Oracle Expert con una sesión de optimización nueva configurada para comprobar los siguientes elementos:

- Optimización de instancias.
- Oportunidades de reutilización de SQL.
- Gestión adecuada del espacio.
- Acceso óptimo a los datos.

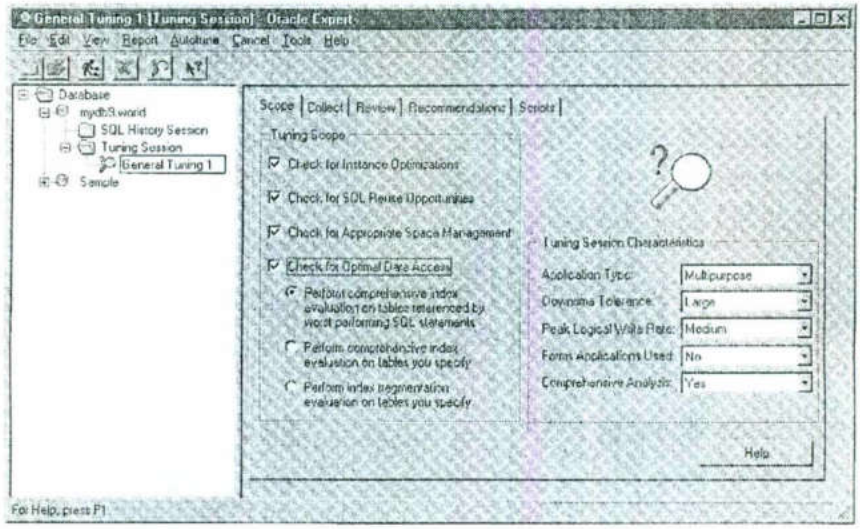


Fig. 8.2 Pantalla inicial de Oracle Expert.

Una vez establecido el ámbito de la sesión de optimización, deberá elegir los criterios de recopilación (Collect). La Figura 8.3 muestra la pantalla de criterios de recopilación con cuatro opciones seleccionadas: System (sistema), Database (base de datos), Instance (instancia) y Workload (carga de trabajo). Tras elegir los criterios de recopilación, haga clic en el botón Collect y la herramienta realizará la recopilación de datos pedida. Para ver las sugerencias y actuar en consecuencia, es necesario esperar a que la recopilación de datos termine o sea detenida manualmente. Si no se especifica un período de tiempo límite para la recopilación de datos, la herramienta seguirá recogiendo datos a los intervalos especificados hasta que se detenga manualmente el proceso.

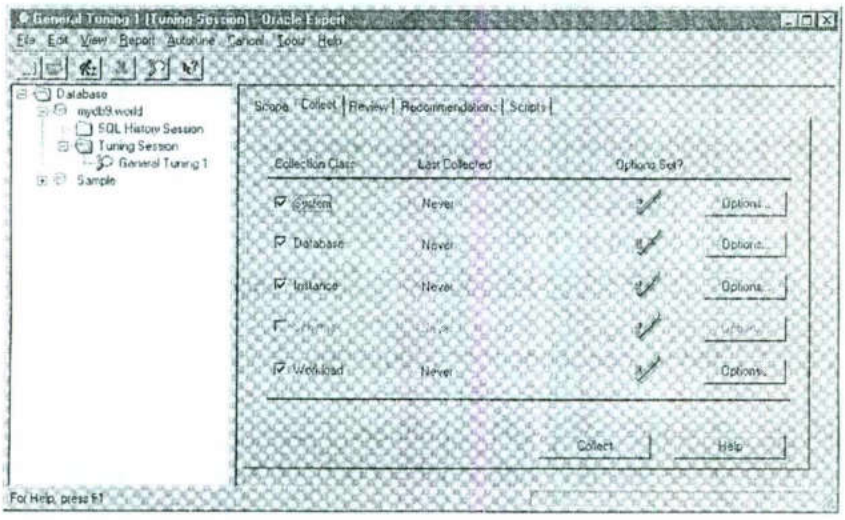


Fig. 8.3 Pantalla Collect de Oracle Expert.

Puede utilizar la opción Review para cambiar los distintos parámetros dinámicos de la base de datos. La Figura 8.4 muestra la pantalla Review con algunas de las áreas disponibles en las que pueden modificarse los parámetros.

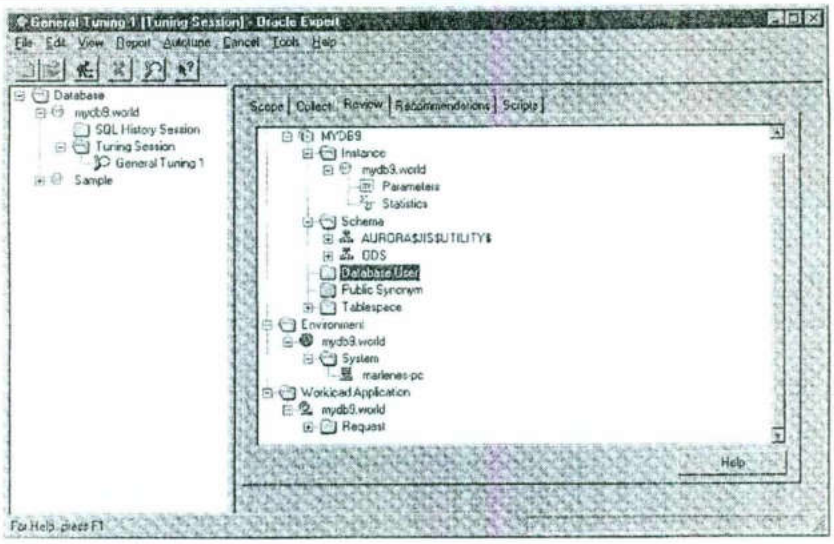


Fig. 8.4 Pantalla Review de Oracle Expert.

Una vez recopilados los datos, puede seleccionar la opción Recommendations mostrada en la Figura 8.5 para ver las conclusiones y sugerencias de optimización de la base de datos. Por último, puede utilizar la opción Scripts para generar los archivos de comandos que implementen las sugerencias de optimización de base de datos recomendadas. La Figura 8.6 muestra la pantalla Scripts. Si hace clic en el botón Generate (generar) de la pantalla Scripts, la herramienta generará los scripts.

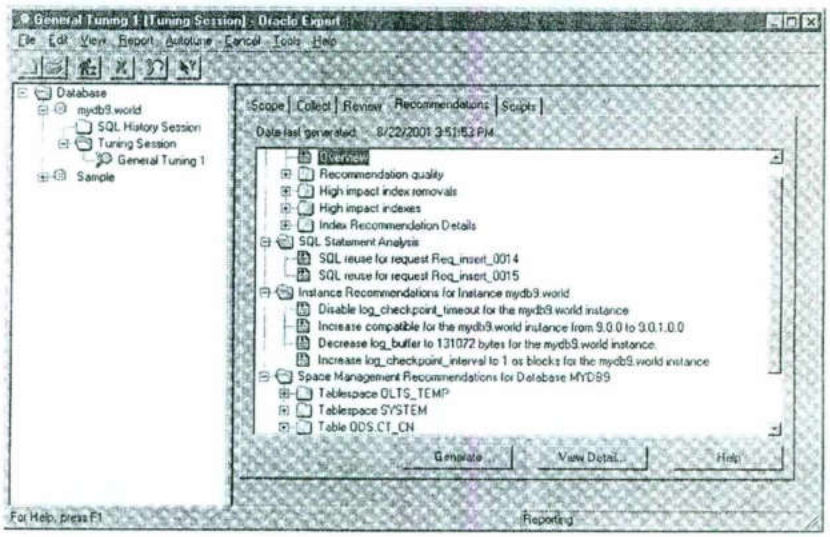


Fig. 8.5 Pantalla Recommendations de Oracle Expert.

8.2 La opción Oracle Performance Manager

La opción Oracle Performance Manager está disponible en el paquete Diagnostics (diagnósticos) y permite la visualización de uno o más gráficos de rendimiento para proporcionar información inmediata sobre el estado actual de la base de datos.

La Figura 8.7 muestra la pantalla inicial del administrador de rendimiento con la opción Memory (memoria) seleccionada. Dentro de esta opción se ha elegido el gráfico SGA Overview (panorámica de la SGA), y el gráfico de estadísticas se muestra en la Figura.

Puede ver los gráficos en varios estilos, incluyendo diagramas de tarta, de barras, de bandas, en tabla, jerárquicos, de orientación horizontal y de orientación vertical. No todos los tipos de gráficos estarán disponibles para todas las opciones de gráficos. Cuando un tipo de gráfico no esté disponible para la opción seleccionada, el icono de ese tipo aparecerá en gris e inactivo, de modo que no pueda ser seleccionado.

El intervalo que indica la frecuencia con que se actualizan las estadísticas puede especificarse utilizando el icono del cronómetro o el menú desplegable (véase la Figura 8.8). Las estadísticas pueden grabarse permanentemente para verlas posteriormente.

Esta herramienta está diseñada para proporcionar una vista rápida del estado de las distintas áreas de la base de datos. Una de las más útiles es la opción Overview of Performance (panorámica del rendimiento), que permite obtener una vista rápida resumida de varios diagramas y gráficos al mismo tiempo. La Figura 8.9 muestra una panorámica de las estadísticas de la base de datos.

Otra forma rápida de obtener solamente la panorámica del rendimiento de una base de datos entera es seleccionar la base de datos en cuestión desde la consola de OEM y elegir a continuación la opción Performance Overview de la opción Diagnostics. Se muestra la misma pantalla de estadísticas que aparece en la Figura 8.9.

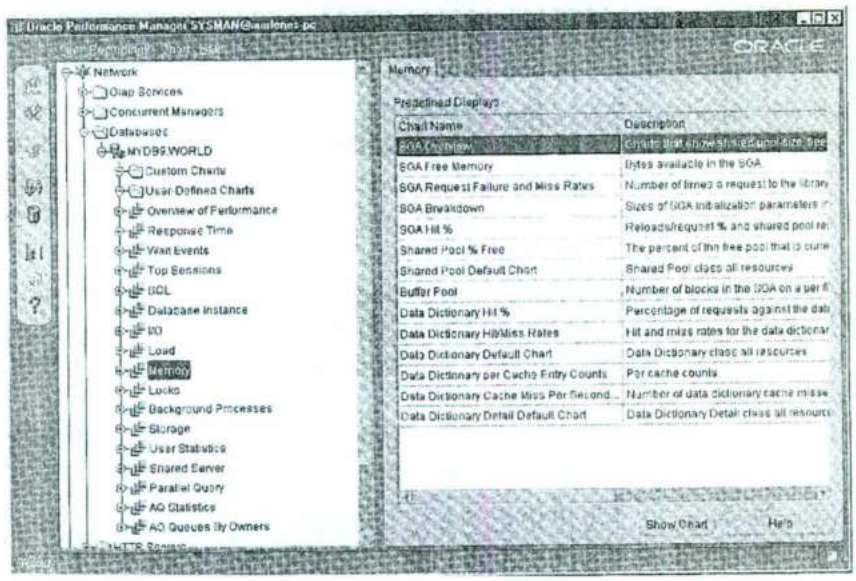


Fig. 8.7 Pantalla inicial de Performance Manager.

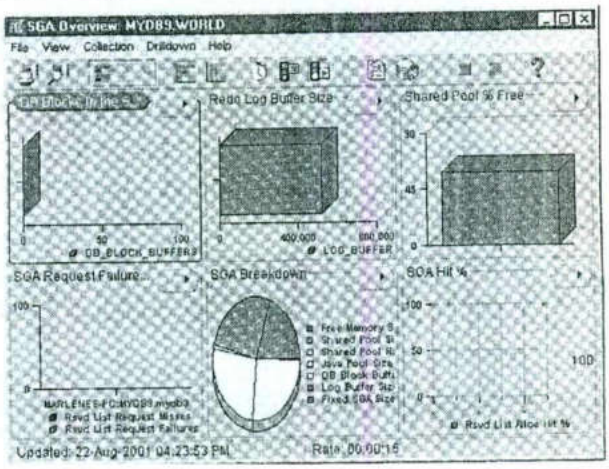


Fig. 8.8 Gràfico SGA Overview de Performance Manager.

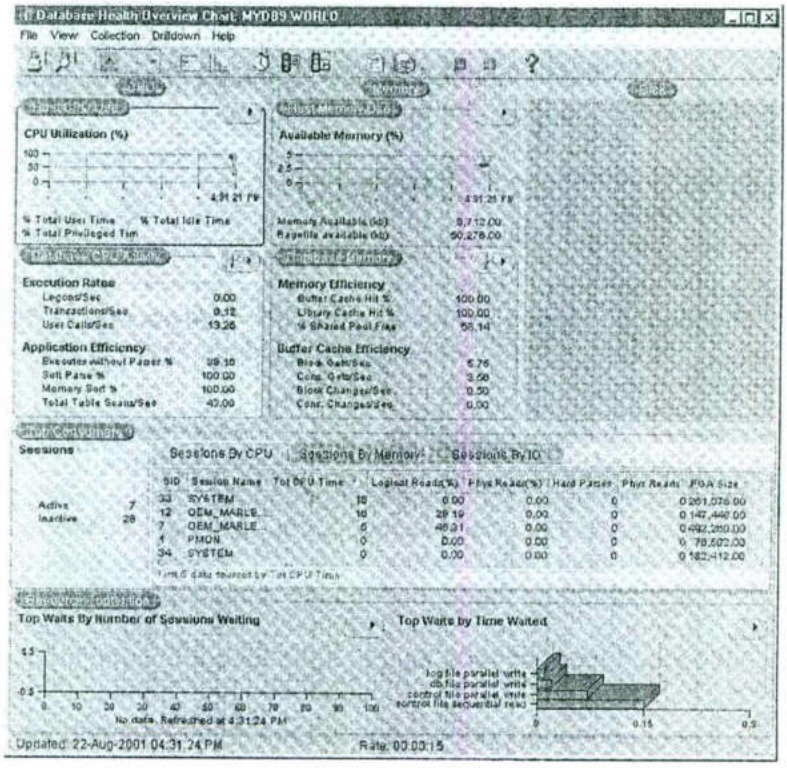


Fig. 8.9 Pantalla d ela panoràmica del rendimiento.

CONCLUSIÒN

Hay un enfoque aún más básico que las técnicas y herramientas presentadas a lo largo de esta tesina. Antes de perder su tiempo y sus recursos en la implementación de una nueva función, primero debe estabilizar su entorno y su arquitectura (el servidor. la base de datos y la aplicación). Si el entorno es estable, entonces debe poder cumplir rápidamente dos objetivos:

- Reproducir con éxito el problema de rendimiento.
- Aislar con éxito la causa del problema.

Para lograr estos objetivos, quizá necesite tener un entorno de pruebas disponible para sus pruebas de rendimiento. Una vez que el problema ha sido aislado con éxito, puede aplicar las técnicas detalladas en esta tesina. En general, el método para la solución de los problemas de optimización debe seguir el mismo orden que las secciones de este trabajo:

1. Evaluar el diseño de la aplicación.
2. Optimizar el código SQL.
3. Optimizar el uso de la memoria.
4. Optimizar el almacenamiento de los datos.
5. Optimizar el tratamiento de los datos.
6. Optimizar el almacenamiento físico y lógico.
7. Optimizar el tráfico de red.

Dependiendo de la naturaleza de su aplicación, puede elegir un orden distinto para estos pasos, o bien puede combinarlos. Por ejemplo, en una aplicación de tres niveles soportada en la Web, puede encontrar problemas de rendimiento durante la introducción de datos en línea. En dichos entornos, debe minimizar la cantidad de interacción con la base de datos. A menudo, las comprobaciones de la validez de los datos (como las consultas de tablas de códigos) que funcionan bien para aplicaciones basadas en servidor provocan problemas de rendimiento para aplicaciones de tres niveles. Las comprobaciones de la validez de los datos requieren que los datos sean enviados hacia delante y hacia atrás a lo largo de los distintos componentes de datos de la aplicación. En general, debe evitar la consulta de datos de la base de datos que la aplicación no utiliza directamente.

¿Cómo puede resolver este problema y seguir realizando comprobaciones de validez de datos? Existen varias opciones:

- Combinar distintas comprobaciones de validez de datos y realizar una sola consulta de la base de datos en lugar de varías. Asegúrese de optimizar la consulta combinada.
- Utilizar variables locales o listas desplegables de opciones en el nivel de aplicación para evitar las consultas a la base de datos. Por ejemplo, cree una lista de estados o países en el nivel de aplicación en lugar de consultar a la base de datos para obtener dicha lista.

- Confíe en los usuarios. En lugar de efectuar la comprobación de validez de datos en el momento en que se introducen los datos, realice las comprobaciones de validez durante el proceso de confirmación. Si utiliza restricciones, los datos se comprobarán de todas formas durante la operación de **commit;** las comprobaciones adicionales en el nivel de aplicación suelen llevarse a cabo para asegurar una reacción más rápida frente a los usuarios. No obstante, los usuarios expertos de una aplicación no cometen muchos errores relacionados con los datos, de modo que el coste de examinar su trabajo en el nivel de aplicación puede no añadir valor alguno a su trabajo. Si elimina las comprobaciones de datos en el nivel de aplicación, puede eliminar muchos de los problemas de rendimiento debidos a la parte de introducción de datos de la aplicación.

Si el diseño de la aplicación no se puede modificar, y el código SQL tampoco, entonces puede optimizar las áreas de memoria y de disco utilizadas por la aplicación. Cuando modifique los parámetros del área de memoria y de disco, debe asegurarse de revisar el diseño de la aplicación y la implementación SQL para asegurarse de que los cambios no afectan negativamente a la aplicación. La necesidad de revisar el proceso de diseño de la aplicación es especialmente importante si elige utilizar un método de duplicación de datos, ya que la no disponibilidad en el momento adecuado de los datos duplicados puede causar problemas dentro del proceso comercial al que da servicio la aplicación.

Monitorización

Ejemplos de consultas

Descripción	Consulta
Usuarios de la Base de datos	select * from dba_users;
Usuario con el que estamos conectados	select username from user_users; select user from dual;
SID de la Base de datos a la que estamos conectados	select name from v$database;
Máquina a la que estamos conectados	select distinct machine from v$session where type='BACKGROUND';
Sesiones activas. Usuarios que estan conectados en este momento	select distinct sid from v$sesstat; select username, sid, serial# from v$session; select username, sid, serial#, program from v$session; select spid, osuser, s.username, s.program from v$process p, v$session s where p.addr=s.paddr; select spid, osuser, s.username, s.program from v$process p, v$session s where p.addr=s.paddr and s.program like '%SQL%';
Finalizar la sesion de un usuario	select username, sid, serial# from v$session; --siendo 11,9 los campos sid, serial# alter system kill session '11,9'; --También desde MS-DOS, consultando select spid, osuser, s.username, s.program from v$process p, v$session s where p.addr=s.paddr; --siendo 000DC el campo SPID `c:\orant\bin>orakill.exe ORCL 000DC`
Estadisticas de uso de CPU para todas las sesiones activas. Por ejemplo, la estadistica "5 user rollbacks" aumentará cada rollback realizado desde scott	select * from v$sysstat;
Estadisticas de uso de CPU para una de las sesiones activas	select v$sesstat.sid, v$sysstat.name, v$sesstat.value from v$sysstat , v$sesstat where v$sysstat.STATISTIC# = v$sesstat.STATISTIC# and v$sesstat.sid=1;
Caché de sentencias sql	select sql_text from v$sqlarea;
Tamaño de todas las estructuras de memoria (en orden descendente)	select * from v$sgastat order by bytes desc;
Cálculo del porcentaje de fallos en los accesos a Row Cache (Caché del diccionario) calculado como Fallos / (Aciertos + Fallos). Si es > 15% se debería incrementar el tamaño de la Shared Area (Shared Pool) mediante shared_pool_size en initorcl.ora	select sum(gets) "(Aciertos+Fallos)", sum(getmisses) "Fallos" from v$rowcache;

Algunas recomendaciones

Recomendación	Acción
SQL idénticas (mayúsculas/minúsculas) y sin comodines	Normativa, Uso de constantes, Funciones de generación de SQL
Tablespace SYSTEM exclusivo para el diccionario	
Número de extensiones mínimo (evitar creación de extensiones)	
Máxima cantidad razonable de RAM para Oracle	shared_pool_size en initorcl.ora
Máximo número razonable de CPU para Oracle	cpu_count en initorcl.ora
Particionamiento de tablas en varios datafiles y estos distribuidos en varios discos	
Si se usa AUTOEXTEND ON siempre con MAXSIZE	
Los Rollback Segments (para consistencia en lectura y recuperaciones en caso de error) en un tablespace exclusivo debido a su uso elevado. Se recomienda un segmento de rollback por cada 4 transacciones concurrentes sobre la misma instancia	
Es preferible muchas sentencias SQL pequeñas antes que pocas grandes	
Borrar índices antes de los procesos batch, recreándolos después	

Gestión del espacio

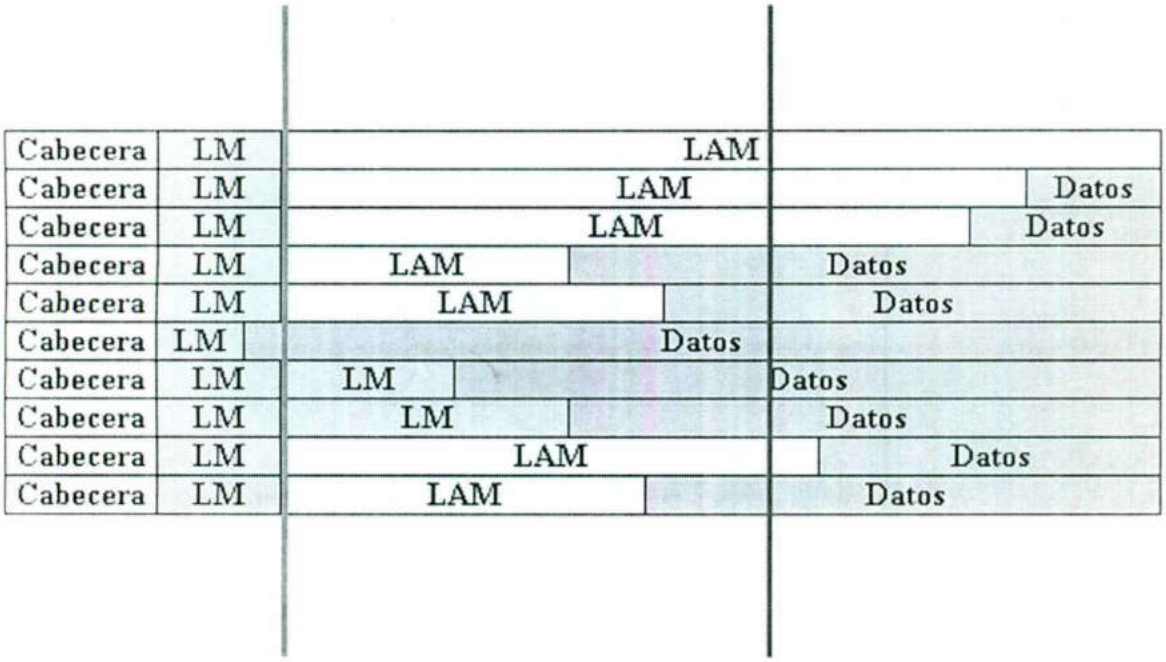
Vistas más relevantes

Significado	dba	user	all
Usuarios	dba_users	user_users	all_users
Tablespaces	dba_tablespace	user_tablespaces	-
Ficheros que componen los datafiles	dba_data_files	-	-
Segmentos	dba_segments	user_segment	all_segments
Extensiones que forman los segmentos	dba_extents	user_extents	-
Bloques libres	dba_free_spac	user_free_spac	-
Bloques libres que podrían unirse	dba_free_space_coalesce	-	-

Bloque Oracle

Una posible evolución de un bloque oracle

LM: Libre para Modificaciones
LAM: Libre para Altas y Modificaciones



A	B	M	PCTFREE	PCTUSED	Objetivo
+			-		Ahorro espacio
		+	+		Incremento de velocidad en transacciones
+	+	-		+	Ahorro espacio
		+		-	Incremento de velocidad en transacciones

REFERENCIAS BIBLIOGRÁFICAS

Bases de datos en castellano. Estructuras de Oracle
http://www.programacion.com/bbdd/tutorial/oracle/
28/05/2003

Administración y Optimización de Bases de Datos Oracle
http://www.redcientifica.com/oracle/c0002p0001.html
5/06/2003

Solocursos.net
http://www.solocursos.net/oracle-slctema118.htm
6/06/2003

Sección de Oracle
http://www.lawebdejm.com/prog/oracle/index.html
16/07/2003

Corey, Michael J.; Abbey, Michael,;Dechichio Daniel J. Jr., Puesta a punto de Oracle, Edición 1999, Oracle Press. Osborne/Mc Graw-Hill

Loney , Kevin; Theriault, Marlene, Oracle 9i Manual del Administrador, Edición especial, Oracle Press 2002. Osborne/Mc Graw-Hill.

Loney, Kevin; Theriault, Marlene, Oracle 8i DBA Handbook, Edición 1999, Oracle Press, Osborne/Mc Graw-Hill.