Administración de Bases de Datos

Configuración y Administración del espacio en disco

Administración de tablas

Las tablas son la unidad básica de almacenamiento de datos en una base de datos Oracle. Los datos se almacenan en filas y columnas.

Seguir estas pautas puede facilitar la administración de sus tablas y puede mejorar el rendimiento al crear la tabla, así como al cargar, actualizar y consultar los datos de la tabla:

- **Diseñar tablas antes de crearlas.** Por lo general, el desarrollador de aplicaciones es responsable de indicar los elementos de una aplicación, incluidas las tablas. Los administradores de bases de datos son los responsables de establecer los atributos del *tablespace* que contendrá las tablas de la aplicación.
  - Normalice cada tabla.
  - Seleccione el tipo de datos adecuado para cada columna.
  - Considere la conveniencia de agregar una o más columnas virtuales.
  - Ubique las columnas que permitan valores nulos al final, para ahorrar espacio de almacenamiento.
  - Agrupe las tablas cuando sea apropiado, para conservar el espacio de almacenamiento y optimizar el rendimiento de sentencias SQL.
  - Antes de crear una tabla, también debe determinar si desea utilizar restricciones de integridad.
- **Especificar el tipo de tabla que se va a crear.** Puede crear diferentes tipos de tablas con Oracle Database.
  - Tablas Ordinarias (montículo). Este es el tipo de tabla básico y de propósito general. Sus datos se almacenan como una colección desordenada.
  - Tabla agrupada. Una tabla agrupada es una tabla que forma parte de un clúster. Un clúster es un grupo de tablas que comparten los mismos bloques de datos, porque comparten columnas y a menudo se usan juntas.
  - Tabla organizada por un índice. A diferencia de una tabla ordinaria, los datos de una tabla organizada por índice se almacenan en una estructura de índice de árbol B de forma ordenada por clave primaria. Además de almacenar los valores de la columna de la llave primaria, cada entrada del índice en el árbol B también almacena los valores de las columnas no clave.

- Tabla particionada. Las tablas particionadas permiten que los datos se descompongan en partes más pequeñas y manejables, llamadas particiones y subparticiones. Cada partición puede tener atributos físicos independientes, proporcionando así una estructura que se puede ajustar para mejorar la disponibilidad y el rendimiento. Además, cada partición puede ser administrada individualmente, lo que puede reducir el tiempo requerido para la copia de seguridad y su administración.
- Tabla externa. Una tabla externa es una tabla que no reside en la base de datos, sino que reside en archivos externos, como archivos del sistema operativo o como archivos del sistema de archivos distribuido de Hadoop (HDFS).
- Tabla híbrida particionada. Una tabla híbrida particionada es una tabla particionada en la que algunas particiones residen en la base de datos y algunas particiones residen en archivos externos, como archivos del sistema operativo o archivos del Hadoop Distributed File System (HDFS)

- **Especificar la ubicación de cada tabla.** Es aconsejable especificar la cláusula TABLESPACE en una instrucción CREATE TABLE para identificar el espacio de tabla que va a almacenar la nueva tabla. En el caso de las tablas particionadas, opcionalmente puede identificar el espacio de tabla que almacenará cada partición.
  - Asegúrese de que dispone de los privilegios y la cuota adecuados en los espacios de tablas que utilice.
  - Si no especifica un espacio de tablas en una instrucción CREATE TABLE, la tabla se crea en su Espacio de tabla predeterminado.
  - Al especificar el espacio de tablas que contendrá una nueva tabla, asegúrese de comprender las implicaciones de su selección. Una especificación correcta del espacio de tabla durante la creación de cada tabla, puede aumentar el rendimiento del sistema de base de datos y disminuir el tiempo necesario para la administración de bases de datos.
  - Las siguientes situaciones ilustran algunos aspectos que pueden afectar el rendimiento:
    - Los objetos de los usuarios no deben almacenarse en el espacio de tablas SYSTEM. Si los objetos de los usuarios se crean en el espacio de tablas SYSTEM, el rendimiento de la base de datos puede verse afectado, ya que tanto los objetos del diccionario de datos como los objetos de usuario deben competir por los mismos recursos. Para evitar esto, Asegúrese de que cuando se crean los usuarios, se les asigne un *Tablespace* predeterminado.
    - Si las tablas asociadas a una aplicación se almacenan arbitrariamente en varios *Tablespaces*, el tiempo necesario para completar las operaciones

administrativas (como la copia de seguridad y la recuperación) se pueden incrementar notoriamente.

- **Crear tablas a partir de otras.** Se puede utilizar la opción AS SELECT en la sentencia CREATE TABLE, debido a que es una manera eficiente de creación de tablas.
- **Crear tablas usando la opción NOLOGGING.** Para crear una tabla de la manera más eficiente, utilice la cláusula NOLOGGING en la sentencia CREATE TABLE ... AS SELECT. La cláusula NOLOGGING hace que se genere la mínima información en el archivo REDO durante la creación de la tabla.
  - El uso de la cláusula NOLOGGING tiene los siguientes beneficios:
    - Se ahorra espacio en los archivos de REDO.
    - Mejora el rendimiento para la creación de tablas grandes.
  - La cláusula NOLOGGING también especifica que las cargas directas posteriores mediante SQL*Loader y las operaciones directas de INSERT no se registren.
  - Sin embargo, las instrucciones UPDATE, DELETE y la inserción convencional posteriores, no se verán afectadas por el atributo NOLOGGING de la tabla y generan REDO.
  - Si la tabla es muy importante, se recomienda realizar una copia de seguridad inmediatamente después de que se crea la tabla. En cambio, si las tablas se crean para uso temporal, esta precaución no es necesaria.
  - En general, la mejora del rendimiento al especificar NOLOGGING es mayor para las tablas grandes. En el caso de las tablas pequeñas, el uso de NOLOGGING no es relevante. Sin embargo, para tablas más grandes, la mejora del rendimiento puede ser significativa.
- **Considere la posibilidad de usar la compresión de tablas**. A medida que su base de datos crezca en tamaño, considere la posibilidad de usar la compresión de tablas para ahorrar espacio y mejorar rendimiento.
  - La compresión ahorra espacio en disco, reduce el uso de memoria en la memoria caché del búfer de la base de datos y puede acelerar significativamente la ejecución de consultas durante las lecturas.
  - La compresión tiene un costo en la sobrecarga de la CPU al efectuar la carga de datos y las operaciones DML. Sin embargo, este costo es compensado por la reducción de los requisitos de E/S. Dado que los datos de la tabla comprimidos permanecen comprimidos en la memoria, lo que también puede mejorar el desempeño de las operaciones DML, ya que caben más filas en el búfer de la base de datos.

  - La compresión de tablas es completamente transparente para las aplicaciones. Es útil en sistemas de soporte para la toma de decisiones, sistemas de procesamiento de transacciones en línea y los sistemas de archivado.
  - La compresión se puede especificar para un espacio de tablas, una tabla o una partición. Si se especifica en el nivel de espacio de tabla, todas las tablas creadas en ese espacio de tabla se comprimen de forma predeterminada.
- **Considere la posibilidad de utilizar tablas agrupadas en clústeres**. Una tabla agrupada en clústeres de atributos es una tabla organizada en montículo que almacena los datos en el disco de forma muy cercana, basados en directivas de agrupación especificadas por el usuario.
  - En la directiva de CLUSTERING BY LINEAR ORDER, ordena los datos de una tabla por columnas específicas.
  - Una tabla agrupada en clústeres de atributos tiene las siguientes ventajas:
    - Es posible una E/S de un solo bloque más optimizada para las búsquedas de tablas cuando la agrupación en clústeres de atributos es lineal con acceso a un índice común.
    - El ordenamiento de datos permite una mejor poda de los índices de almacenamiento de externo.
    - Puede agrupar tablas de hechos en función de atributos unidos de otras tablas.
- **Estime el tamaño de la tabla.** Calcule el tamaño de las tablas antes de crearlas. Preferiblemente, haga esto como parte de la planeación de la base de datos. Conocer los tamaños y usos de las tablas de la base de datos es una parte importante de una buena planeación.
- Puede utilizar el tamaño estimado combinado de las tablas, junto con las estimaciones de los índices, para determinar la cantidad de espacio en disco que se requiere para contener la base de datos prevista. A partir de estas estimaciones, puede prever los requerimientos de hardware.

Administración de índices

Los índices son estructuras opcionales asociadas a tablas y clústeres que permiten que las consultas SQL se ejecuten más rápidamente.

Al igual que el índice de un libro le ayuda a localizar la información más rápido, un índice de una tabla proporciona un acceso más rápido a los datos. Puede usar índices sin tener que volver a escribir ninguna consulta. Los resultados son los mismos, pero se ven más rápidamente.

Oracle proporciona varios esquemas de indexación:

- **Índices de árbol B**: los predeterminados y los más comunes en tablas.

- **Índices de clúster de árbol B**: definidos específicamente para clústeres
- **Índices de clúster hash**: definidos específicamente para un clúster hash
- **Índices globales y locales**: se relacionan con tablas e índices particionados.

Los índices son lógica y físicamente independientes de los datos de la tabla asociada. Al ser estructuras independientes, requieren su propio espacio de almacenamiento. Puede crear o quitar un índice sin afectar a las tablas, las aplicaciones de base de datos u otros índices.

La base de datos automáticamente mantiene los índices al insertar, actualizar y eliminar filas de la tabla asociada. Si usted elimina un índice, todas las aplicaciones continúan funcionando. Sin embargo, el acceso a los datos previamente indexados podría ser más lento.

Directrices para administrar índices.

- **Crear índices después de insertar datos de tabla.** A menudo, los datos se insertan o cargan en una tabla mediante SQL*Loader o una importación utilidad. Es más eficaz crear un índice para una tabla después de insertar o cargar los datos. Si crea uno o más índices antes de cargar los datos, la base de datos debe actualizar cada índice a medida que se inserta cada fila.
- **Indexar las tablas y columnas correctas.** Siga las recomendaciones sobre las tablas y columnas que son adecuadas para la indexación. Las columnas candidatas para la indexación son:
  - Donde los valores son relativamente únicos en la columna.
  - Donde hay una amplia gama de valores.
  - Si hay un pequeño rango de valores se recomiendan los índices de mapa de bits.
  - Si la columna contiene muchos valores nulos, pero las consultas a menudo seleccionan todas las filas que tienen un valor, se desaconseja usar la frase IS NOT NULL.
- **Ordenar las columnas del índice para mejorar el rendimiento.** El orden de las columnas de la instrucción CREATE INDEX puede afectar al rendimiento de las consultas. En general, especifique primero las columnas que se usan con más frecuencia.
- **Limitar el número de índices de cada tabla.** Una tabla puede tener cualquier número de índices. Sin embargo, cuantos más índices haya, más sobrecarga se genera cuando se modifica la tabla.
- **Eliminar índices que ya no son necesarios.** Se recomienda quitar los índices que ya no son necesarios.
- **Estimar el tamaño del índice y establecer los parámetros de almacenamiento.** Estimar el tamaño de un índice antes de crearlo puede facilitar una mejor planificación y gestión del espacio en disco.

- **Especificar el espacio de tabla para cada índice.** Los índices se pueden crear en cualquier espacio de tablas. Un índice se puede crear en el mismo o espacio de tabla diferente al de la tabla que indexa.

Bitácoras

Son estructuras ampliamente utilizadas para grabar las modificaciones de la base de datos.

Cada registro de la bitácora escribe una única escritura de base de datos y tiene lo siguiente:

- **Nombre de la Transacción:** Nombre de la transacción que realizó la operación de escritura.
- **Nombre del Dato:** El nombre único del dato escrito.
- **Valor Antiguo:** El valor del dato antes de la escritura.
- **Valor Nuevo:** El valor que tendrá el dato después de la escritura.

Es fundamental que siempre se cree un registro en la bitácora cuando se realice una escritura antes de que se modifique la base de datos.

También tenemos la posibilidad de deshacer una modificación que ya se ha escrito en la base de datos, esto se realizará usando el campo del valor antiguo de los registros de la bitácora.

Los registros de la bitácora deben residir en memoria estable como resultado el volumen de datos en la bitácora puede ser exageradamente grande.

La instrucción en MySQL para crear una bitácora en .txt se crea antes de acceder a la base de datos con la instrucción:

```
"xampp>mysql>bin>mysql -hlocalhost -uroot --tee=C:bitacora.txt"
```

La bitácora debe registrar todos los movimientos (insertar, eliminar y modificar) que se realicen en las tablas de la base de datos. Para lograr lo anterior es necesario crear un disparador para que se ejecute después de la operación de insertar, otro para después de eliminar y el último para después de modificar para cada una de las 3 tablas de la base de datos.

PARTICIONES

Cuando alguna de las tablas de una base de datos llega a crecer tanto que el rendimiento empieza a ser un problema, es hora de empezar a conocer algo sobre optimización. Una característica de MySQL son las particiones.

Particionar tablas en MySQL nos permite rotar la información de nuestras tablas en diferentes particiones, consiguiendo así realizar consultas más rápidas y recuperar espacio en disco al borrar los registros. El uso más común de particionado es según la fecha.

Para ver si nuestra base de datos soporta particionado simplemente ejecutamos:

```
SHOW VARIABLES LIKE '%partition%';
```

Se puede particionar una tabla de 5 maneras diferentes:

- **Por Rango:** para construir las particiones se especifican rangos de valores.

```
ALTER TABLE contratos
PARTITION BY RANGE (YEAR (fechaInicio)) (
PARTITION partDecada80 VALUES LESS THAN (1990),
PARTITION partDecada90 VALUES LESS THAN (2000),
PARTITION partDecada00 VALUES LESS THAN (2010),
PARTITION partDefault VALUES LESS THAN MAXVALUE );
```

  La última partición (partDefault) tendrá todos los registros que no entren en las particiones anteriores. De esta manera nos aseguramos que la información nunca dejará de insertarse en la tabla.

- **Por Listas:** para construir nuestras particiones especificamos listas de valores concretos.

```
ALTER TABLE contratos
PARTITION BY LIST (YEAR (fechaInicio)) (
    PARTITION partDecada80 VALUES IN (1980, 1981, 1982, 1983,
1984, 1985, 1986, 1987, 1988, 1989),
    PARTITION partDecada90 VALUES IN (1990, 1991, 1992, 1993,
1994, 1995, 1996, 1997, 1998, 1999),
    PARTITION partDecada00 VALUES IN (2000, 2001, 2002, 2003,
2004, 2005, 2006,
2007, 2008, 2009),
    PARTITION partDecada10 VALUES IN (2010, 2011, 2012, 2013,
2014, 2015, 2016,
2017, 2018, 2019) );
```

- **Por Hash:** MySQL se encarga de distribuir las tuplas automáticamente usando una operación de módulo. Sólo hay que pasarle una columna o expresión que resulte en un entero (el hash) y el número de particiones que queramos crear.

```
ALTER TABLE contratos
PARTITION BY HASH (YEAR (fechaInicio))
PARTITIONS 7;
```

- **Por Clave:** similar a la partición por hash, pero en este caso no necesitamos pasarle un entero; MySQL utilizará su propia función de hash para generarlo. Si no se indica ninguna columna a partir de la que generar el hash, se utiliza la clave primaria por defecto.

```
ALTER TABLE contratos
PARTITION BY KEY ()
PARTITIONS 7;
```

- **Compuesta:** podemos combinar los distintos métodos de particionado y crear particiones de particiones

Borrar Particiones

Lo bueno de trabajar con particiones es que podemos borrar rápidamente registros sin tener que recorrer toda la tabla e inmediatamente recuperar el espacio en disco utilizado por la tabla.

Por ejemplo, si queremos borrar la partición más antigua simplemente ejecutamos:

```
ALTER TABLE reports DROP PARTITION p201111;
```

Añadir particiones

En el ejemplo anterior las 2 últimas particiones creadas han sido:

```
PARTITION p201205 VALUES LESS THAN (TO_DAYS ("2012-06-01")),
PARTITION pDefault VALUES LESS THAN MAXVALUE
```

El problema es que todos los INSERT que se hagan después de mayo de 2012 se insertarán en pDefault. La solución sería añadir particiones nuevas para cubrir los próximos meses:

```
ALTER TABLE reports REORGANIZE PARTITION pDefault INTO (
PARTITION p201206 VALUES LESS THAN (TO_DAYS ("2012-07-01")),
PARTITION pDefault VALUES LESS THAN MAXVALUE);
```

En el caso que no tuviéramos una partición del tipo pDefault simplemente ejecutamos:

```
ALTER TABLE reports ADD PARTITION (PARTITION p201206 VALUES LESS
THAN (TO_DAYS ("2012-07-01")));
```

Consultar Particiones

Para consultar información de particiones creadas en una tabla así como también los registros que contiene cada una ejecutamos:

```
SELECT PARTITION_NAME, TABLE_ROWS FROM
information_schema.PARTITIONS WHERE TABLE_NAME='reports';
```