Data Base Administration

1. FUNCIONES DE UN DBA (Data Base Administrator)

El rol de DBA involucra las siguientes tareas:

- Creación de Bases y Tablas
  - Base de Datos
    - Verificar el lugar de creación, en los espacios de disco adecuados.
    - El tipo de Base de Datos. Ver la necesidad de creación con o sin Logs transaccionales (dependiendo del motor)
    - Crear la BD.
    - Borrar la BD.

  - Tablas
    - Seleccionar los Data Types que serán usados en las tablas.
    - Evaluar la creación o no de tablas fragmentadas (tablas con grandes volumenes de datos).
    - Verificar el Lugar de Creación dentro de las estructuras de disco asignadas al Mortor de BD.
    - Definir la Alocación de espacio de disco por cada tabla, y su futuro crecimiento.
    - Crear las Tablas
    - Alterar la definición de las Tablas.
    - Borrar las Tablas.

- Asegurar la Integridad de los Datos: Existen algunos métodos para asegurar la integridad de los datos:

  - El administrador DSA, puede crear views, puede otorgar o revocar permisos sobre la base de datos, tablas o columnas.
    - Base de Datos
    - Tablas
    - Columnas (depende del motor de BD)
  - Creación de Views
  - Otorgar o revocar permisos sobre la ejecución de Stored Procedures y Functions.

  - La integridad de entidades es usada para forzar la integridad en una tabla mediante la creación de claves primarias que unívocamente identifican a una tabla.
  - La integridad referencial es usada para forzar la integridad entre tablas.
  - La integridad semántica es usada para asegurar la coherencia de los nombres de los data types y sus valores defaults.

- Manejo de Concurrencia:

  - Concurrencia Sobre la Lectura: Se resuelve con los llamados Niveles de Aislamiento.
  - Concurrencia de Update : Según el modo de lockeo de la aplicación y de la tabla a la cual se esta actualizando. Pudiendo usar también los niveles de aislamiento.

- Optimización de Accesos

  Otro de los objetivos de la administración de base de datos es que los usuarios puedan acceder y actualizar los datos de la manera más rápida.
  El módulo optimizador del Motor de BD es el responsable de seleccionar el camino menos costoso y más eficiente para llegar a recuperar y/o actualizar los datos. Examina los índices y la distribución de datos sobre las tablas a ser accedidas, y evalúa todos los caminos posibles y mediante un algoritmo establece un costo para cada camino eligiendo luego el menos costoso.

Entonces el rol del administrador de base de datos es asegurar que las tablas se encuentren indexadas adecuadamente de acuerdo a los requerimientos de las aplicaciones (eliminando índices innecesarios) y tener actualizado permanentemente las estadísticas que el Módulo optimizador necesita para elaborar los costos mencionados anteriormente.

- Creación de Indices
- Optimización de acceso a los datos
    - Actualización de estadísticas (Depende del motor de BD)
    - Distribución de Datos (Depende del motor de BD)

2. AMBIENTES DE TRABAJO

Características de los DSS Queries (DataWarehouse):

- Muchas filas son leídas y el resultado no esta en una transacción
- Los datos son leídos secuencialmente
- SQL complejos son ejecutados
- Grandes archivos temporarios son creados
- El tiempo de respuesta es medido en horas y minutos
- Hay relativamente poca concurrencia de queries

CARACTERISTICAS DEL Ambiente OLTP: On-Line Transaction Processing

- Relativamente pocas filas leídas
- Alta frecuencia de transacciones
- Acceso a los datos a través de índices
- Simple Operaciones SQL
- Respuesta de scan medida en segundos
- Muchos queries concurrentes

3. FRAGMENTACION

Se denomina fragmentación al proceso por el cual una tabla es dividida en diferentes partes (dbspaces) con una condición determinada. La fragmentación provee la habilidad de distribuir datos de una tabla en discos separados, también llamada *Fragmentación Horinzontal o Particionamiento.*

Hay dos tipos:

Fragmentación por Expresión
- Normal
- Hash functions

por Round Robin

El objetivo de la fragmentación es balancear la entrada/salida de datos y maximizar el rendimiento entre múltiples discos.

Ventajas de la Fragmentación:

√ Scans Paralelos : En ambientes DSS (Decision Support), el sistema puede leer múltiples fragmentos en paralelo. En este tipo de ambientes se utilizan queries donde es necesario leer gran cantidad de filas. Este es el principal beneficio de la fragmentación. Las tareas que se pueden paralelizar son: scans, joins, sorts, funciones agregadas, agrupaciones (group by) e inserts. Es realmente efectivo cuando el equipo cuenta con mas de un procesador para atender a los múltiples threads lanzados en paralelo.

- √ Balanceo de I/O Mediante el balanceo de I/O ud. puede reducir la contencion de disco y elimina cuellos de botella. Esto es ventajoso para los ambientes OLTP donde el alto grado de rendimiento es crítico.
- √ Archive y Restore Tanto el archive como el restore pueden ser realizados a nivel de dbspace. Por lo tanto es posible mantener y/o restaurar solo una parte de la tabla si esta fragmentada en diferentes dbspaces.
- √ Alta disponibilidad Es posible especificar que salte algún fragmento no disponible. Esto es ventajoso en DSS donde gran cantidad de datos son leídos y el procesamiento no debería ser interrumpido si un fragmento en particular no está disponible.

4. ESTRATEGIA DE ÍNDICES

Existen cuatro características o tipos de índices:

1. Unique
2. Cluster
    - Este tipo de índice provoca al momento de su creación que físicamente los datos de la tabla sean ordenados por el mismo.
3. Compuesto
    - Las principales funciones de un índice compuesto son:
        - Facilitar múltiples joins entre columnas
        - Incrementar la unicidad del valor de los índices
4. Duplicado

Ejemplo de índice compuesto:

customer_nun, lname, fname

Este índice se usará en o para los siguientes casos:

- Joins sobre customer_num, o customer_num y lname o customer_num, lname y fname
- Filtros sobre customer_num, o customer_num y lname o customer_num, lname y fname
- ORDERS BY sobre o customer_num y lname o customer_num, lname y fname
- Joins sobre customer_num y Filtros sobre lname y fname
- Joins sobre customer_num y lname y Filtros sobre fname

BENEFICIOS DE INDEXAR:

1. Se le provee al sistema mejor perfomance al equipo ya que no debe hacer lecturas secuenciales sino accede a través de los índices, solo en los casos que las columnas del Select no formen parte del índice.
2. Mejor perfomance en el ordenamiento de filas
3. Asegura únicos valores para las filas almacenadas
4. Cuando las columnas que intervienen en un JOIN tienen índices se le da mejor perfomance si el sistema logra recuperar los datos a través de ellas

COSTO DE INDEXAR:

1. El primer costo asociado es el espacio que ocupa en disco, que en algunos casos suele ser mayor al que ocupan los datos.
2. El segundo costo es el de procesamiento, hay que tener en cuenta que cada vez que una fila es insertada o modificada o borrada, el índice debe estar bloqueado, con lo cual el sistema deberá recorrer el árbol de índices B+.

<u>GUIA RESUMEN CUANDO DEBERIAMOS INDEXAR</u>

- Indexar columnas que intervienen en Joins
- Indexar las columnas donde se realizan filtros

- Indexar columnas que son frecuentemente usadas en orders by
- Evitar duplicación de índices
    - ◊ Sobre todo en columnas con pocos valores diferentes Ej: Sexo, Estado Civil, Etc.
- Limitar la cantidad de índices en tablas que son actualizadas frecuentemente
    - ◊ Porque sobre estas tablas se estarán ejecutando Selects extras
- Verificar que el tamaño de índice debería ser pequeño comparado con la fila
    - ◊ Tratar sobre todo en crear índices sobre columnas cuya longitud de atributo sea pequeña
    - ◊ No crear índices sobre tablas con poca cantidad de filas, no olvidar que siempre se recupera de a páginas. De esta manera evitaríamos que el sistema lea el árbol de índices
- Tratar de usar índices compuestos para incrementar los valores únicos
    - ◊ Tener en cuenta que si una o más columnas intervienen en un índice compuesto el optimizador podría decidir acceder a través de ese índice aunque sea solo para la búsqueda de los datos de una columna, esto se denomina "*partial key search"*
- Usando cluster index se agiliza la recuperación de filas
    - ◊ Uno de los principales objetivos de la Optimización de bases de datos es reducir la entrada/salida de disco. Reorganizando aquellas tablas que lo necesiten, se obtendría como resultado que las filas serían almacenadas en bloques contiguos, con lo cual facilitaría el acceso y reduciría la cantidad de accesos ya que recuperaría en menos páginas los mismos datos.

5. OPTIMIZADOR BASADO EN COSTOS (Motor Informix)

Existen 3 Tipos de estrategias de JOINS:

- **Nesteed Loop Joins**
    - Scanea la primer tabla en algún orden, con el dato de la columna busca en la segunda tabla, y forma la tupla
    - Proceso:
        - ◊ Va leyendo la tabla "A" por orden específico (puede entrar por índice), realiza el filtro del where si fuera necesario. Crea una key con la columna del join con la que debe acceder a la tabla "B". Si el campo de la tabla es declarado único, entonces leerá una sola vez, sino puede llegar a generar una key en la tabla "B" para acceder a la misma tantas veces como sea necesario y así va generando las tuplas necesarias.
- **Sort Merge Joins**
    - Ordena las filas de la primer tabla por la columna del join con el resultado del escaneo, luego ordena la segunda tabla con el resultado del escaneo, y entonces mezcla ambas para obtener las tuplas resultantes
- **Hash Joins**
    - Scanea la primer secuencialmente para construir la tablita de Hash (puntero+columna del join) y busca su correspondiente en la segunda tabla para formar la tupla
    - Proceso:
        - ◊ Este método tiene especial uso cuando se trata de tablas voluminosas y también cuando esta seteado el ambiente como DSS. Este proceso es más rápido que el Sort Merge Joins pero puede necesitar mas espacio de memoria y almacenamiento.
        - ◊ Lee las filas de la primer tabla realizando el filtro del where si fuera necesario, con la columna que interviene en el Join calcula el Puntero para la tabla Hash y arma la misma.
        - ◊ Luego cada fila de la tabla Hash es buscada en la Tabla "B" y arma de esta manera las tuplas resultantes

EJEMPLO DE COMO SE CALCULA EL COSTO:

COSTO = (CANTIDAD DE I/O DE DISCO) + (#CANTIDAD DE FILAS RETORNADAS * W)

W = Factor de conversión

PROCESO DE OPTIMIZACION: En general

- ◊ Examina todas las tablas , filtros e índices
  - √ Examina selectivamente cada filtro
  - √ Determina si los índices pueden ser usados para: Filtros, Orders By, Groups By
  - √ Busca la mejor manera de buscar en una tabla: Sequencial o por Índice
- ◊ Estima costo por cada par de join encontrado y posible
  - √ Busca la mejor manera de Juntas las dos tablas (nesteed Loop, Hash o sort merge)
  - √ Decide cual de los índices puede ser el mejor para el join
  - √ Calcula el costo del join
  - √ Elimina pares redundantes
- ◊ Repite cada estimación para cada tabla adicional en el join

6. SEGURIDAD SOBRE LOS DATOS

Existen 3 niveles de Seguridad:

a) Base de datos
b) Tabla
c) Columna (depende del motor de bd)

Existen 3 niveles de privilegios:

Privilegios a Nivel de Tabla:

Alter Insert Delete Update Select Create Index All

Otorgar Privilegios (GRANT)

Ejemplos para Otorgar privilegios a nivel de Base de Datos (BD Informix)

Sintaxis del Grant:

GRANT db_privilegio TO {User | PUBLIC} {WITH GRANT OPTION} {AS grantor}

Ej.:

GRANT DBA TO Jose;

GRANT RESOURCE TO PUBLIC;

GRANT RESOURCE TO Maria, Jose, Pedro;

WITH GRANT OPTION

Permite al usuario que recibe también otorgar privilegios a otros (Solo el mismo que se otorgó).

EJ:

GRANT RESOURCE TO José WITH GRANT OPTION
José puede otorgar a otros privilegios de RESOURCE

AS GRANTOR = (GRANTOR = Usuario)

Hace otorgador de permisos a otro usuario, pero renunciando su habilidad mas tarde de revocar los privilegios otorgados.

EJ:

GRANT RESOURCE TO Jose AS Maria

Ejemplos – Otorgar privilegios a nivel de Tablas (BD Informix)

Sintaxis:

GRANT {tb_privilegio, tb_privilegio,.... } ON {Tbl_name}
TO {User list|PUBLIC} {WITH GRANT OPTION} {AS usuario}

Ejemplos:

GRANT Insert, Delete ON Items TO pedro, juan;
GRANT all ON Orders TO PUBLIC;
GRANT all ON customers TO Luis AS Maria

{WITH GRANT OPTION} y AS usuario -→ Idem Anterior

Revocar Privilegios

REVOKE

Otorgando privilegios a nivel de Tablas (BD Informix)

Sintaxis:

GRANT {tb_privilegio, tb_privilegio,.... } ON {Tbl_name}
TO {User list|PUBLIC} {WITH GRANT OPTION} {AS usuario}

Ejemplos:

GRANT Insert, Delete ON Items TO pedro, juan;

GRANT all ON Orders TO PUBLIC;

GRANT all ON customers TO Luis AS Maria

{WITH GRANT OPTION} y AS usuario -→ Idem Anterior

Otorgando privilegios a nivel de Columnas:

Sintaxis:
GRANT {tb_privilegio(COL1,COL2), tb_privilegio(COL3,COL4),.... } ON {Tbl_name}
TO {User list|PUBLIC} {WITH GRANT OPTION} {AS usuario}

Ej.:
GRANT INSERT, UPDATE(cantidad), DELETE ON Items TO Pedro

Otorgando privilegios a nivel de Store Procedures:

GRANT EXECUTE ON Del_proce TO PUBLIC

Nota: Es posible quitar algún tb_privilegio a un usuario sobre alguna tabla, pero también es posible dar el permiso de ejecución de un store procedure que realice ese evento quitado sobre la tabla y esto es correcto (según manual training).

Revocar Privilegios (REVOKE)

Ejemplos para revocar privilegios a nivel de Base de Datos (BD Informix)

Sintaxis:

REVOKE {tb_privilegies ON Table_name|Db_privilegies } FROM {User List|PUBLIC}

Ej:

```
REVOKE RESOURCE FROM Pedro;
REVOKE CONNECT FROM Juan;
```

Ejemplos para evocar privilegios a nivel de Tabla (BD Infomrix)

Ej.:

```
REVOKE Insert, Delete, Update ON items FROM Pablo, Jose;
REVOKE all ON Orders FROM PUBLIC;
```

7. CREACIÓN DE VIEWS

Una view es también llamada una tabla virtual o ventana dinámica.
Una view es un conjunto de columnas, ya sea reales o virtuales, de una misma tabla o no, con algún filtro determinado o no.

Sintaxis: (BD Informix)

```
CREATE VIEW Nombre AS
        Select .....;
```

Ej.:

```
CREATE VIEW dat_cli AS
        Select nombli, direccion FROM clientes
                        WHERE cod_postal = 1002;

CREATE VIEW dat_cli (num_orden, Pre_total) AS
        Select order_num, sum(total_price)  FROM items GROUP BY 1;
```

Nota: Tener en cuenta que ciertas Views no tienen permisos de insert, delete, update... Como en el ejemplo anterior .

Restricciones:

1. No se pueden crear índices en las Views
2. Una view depende de las tablas a las que se haga referencia en ella, si se elimina una tabla todas las views que dependen de ella se borraran. Lo mismo para el caso de borrar una view de la cual depende otra view.
3. Algunas views tienen restringido los: Inserts, Deletes, Updates.

- Aquellas que tengan joins
- Una función agregada
- Tener en cuenta ciertas restricciones para el caso de Actualizaciones:
    - Si en la tabla existieran campos que no permiten nulos y en la view no aparecen, los inserts fallarían
    - Si en la view no aparece la primary key los inserts podrían fallar

4. Se puede borrar filas desde una view que tenga una columna virtual
5. Con la opcion WITH CHEK OPTION, se puede actualizar siempre y cuando el checkeo de la opción en el where sea verdadero
6. Al crear la view el usuario debe tener permiso de select sobre las columnas de las tablas involucradas
7. No es posible adicionar a una View las cláusulas de: ORDER BY y UNION