Diseño Físico

Diseño Físico

- El diseño físico de BD forma parte importante del ciclo de vida de un sistema de BDs.
- Consiste en escoger las estructuras de almacenamiento y caminos de acceso que
  - 1) cumplan los objetivos del sistema
  - 2) proporcionen un balance óptimo entre el rendimiento (tiempos de respuesta de transacciones, número de transacciones por minuto...) y el costo (espacio utilizado, reorganizaciones de datos...).
- No existen metodologías para realizar el diseño físico. Es muy dependiente del SGBD concreto.

Diseño Físico: Recopilar información.

- Por cada op. (preg. SQL) con la BD indicar:
  - Tipo: INSERT, SELECT, UPDATE, DELETE
  - Tablas que se van a acceder (cardinalidad)
  - Condiciones de selección (selectividad de cada una)
  - Condiciones de combinación-join (selectividad)
  - Atributos a ser proyectados / modificados
  - Frecuencia esperada de que se realice la operación.
  - Restricciones importantes de ejecución (si las hay)
- Regla 80-20: El 80% del procesamiento se realiza por el 20% de las transacciones.

Reconsiderar algunas de las claves utilizadas

- Las claves escogidas deben asegurar que no haya elementos repetidos.
- A veces se asignan códigos que toman valores numéricos sucesivos: 1,2,3,...
- Problema: esto puede implicar realizar consultas con varios joins.
- Si es posible hay que intentar usar claves con significado siempre que aseguren la unicidad.

Desnormalización

- El proceso de normalización consiste en dividir una tabla R en R1 y R2 si $R=R1 \bowtie_{R1.K=R2.K} R2$
  - Evita redundancia y anomalías (ins./bor./mod.)
  - Problema: Para obtener R hay que hacer el join
- Si (casi) siempre que se recuperan los valores de R1 se utilizan también los de un mismo atributo(s) R2.Atr, entonces se puede añadir el atributo R2.Atr a la tabla R1 --> (No estaría en 3FN!!)
  - Hay que controlar que no haya anomalías
  - Habrá redundancia pero estará controlada
  - Se evitará ejecutar joins (según las frecuencias def.)

Particionamiento horizontal

- Si existe tabla R = $\sigma_{C1}(R)$ U ... U $\sigma_{Cn}(R)$ donde
  - muchas operaciones con la BD son con $\sigma_{Ci}(R)$
  - algunos atributos son inaplicables (NULL) según Ci
  - entonces cada $\sigma_{Ci}(R)$ se guarda en una tabla y se define una vista sobre R (¿diseño lógico? ¿físico?)
- En general si una operación $\sigma_{Ci}(R)$ es muy frecuente, con Ci muy selectivo y R muy grande: almacenar $\sigma_{Ci}(R)$ en una tabla S
  - hay que controlar la redundancia / integridad (triggers)
    - inconveniente: inserciones en S o R (ver frecuencias)
  - los programas deberán usar la nueva tabla S
  - si después se suprime tabla S --> crear vista para S
    - para mantener indep. física. Esto sí es diseño físico !!

Particionamiento vertical

- Si existe una tabla R (A1,...An,B1,...Bm) donde
  - muchas de las operaciones afectan sólo a atributos A1,...,An y muy pocas veces a atributos B1,...Bm
  - esas operaciones son muy frecuentes
  - R(..Ai,...,Bj...) es mucho más grande que R(..Ai...)
- Entonces almacenar R(...Ai...) en una tabla S
  - controlar redundancia / integridad. Fácil si hay mecanismo de triggers. Si no, controlar la parte de las aplicaciones que insertan / modifican R.
  - inconveniente: las inserciones en R(...Ai...). Hay que valorar su frecuencia para ver si merece la pena.

Precomputar joins en tablas

- Si existe una consulta R1 ⋈ R2 ⋈ ... ⋈ Rn que se ejecuta frecuentemente, cuyo coste es elevado (los joins son costosos) y donde cada relación Ri no se actualiza frecuentemente entonces se puede crear una tabla donde se almacene el resultado de dicha consulta.
  - Habrá que controlar recomputar dicha consulta
    - 1) Utilizando triggers cada vez que cambie algún Ri
    - o bien 2) Ejecutando periódicamente algunos scripts (ej. a las noches). Se puede si no es obligatorio que la consulta devuelva los valores más actuales.
  - Valorar: frecuencia de cambios en Ri, tamaño del resultado, tiempo de ejecución de la consulta inicial

Organización física para tablas

- Si un atributo se usa a menudo para recuperar tuplas en orden o para hacer joins entonces se define como clave primaria o como índice cluster (si no puede ser clave). ¡¡Sólo UNO!!
  - algunos SGBD permiten almacenar tablas juntas (en un mismo cluster). Útil para ejecutar joins (alternativa a desnormalizar)
- Si hay otros atributos que se usan en condiciones de selección o en joins entonces se definen como índices.
  - conveniente si se seleccionan pocas tuplas (< 15% total tuplas) y si la cardinalidad de la tabla es alta (> 100 tuplas)
- Si la tabla se actualiza con gran frecuencia hay que definir un número mínimo de índices (coste de actual.)
- Si un atributo se usa frecuentemente para selecciones del tipo A=c o en joins y no para recuperar por orden de A, entonces definirlo como hash (si SGBD permite)

Conclusiones

- Realizar el diseño físico inicial
  - Obtener información de las operaciones esperadas
  - Resolver operaciones con mayores restricciones (aplicando algunos de los métodos explicados)
  - Resolver el resto de las opers. sin perjudicar a otras
    - añadir índices para favorecer consultas perjudica a operaciones de inserción / borrado
- Replantearse continuamente dicho diseño (*Tunning*)
  - analizar/auditar el sistema actual
  - tomar nuevas decisiones (añadir/borrar índices o tablas (crear vistas y triggers si es necesario)