Bases de Datos 2

Aquiles Loranca Sánchez

CAPITULO I – CONCEPTOS BASICOS.
. Base de Datos.
. Base de Datos Relacional.
. Bases de Datos Relacionales vs. Archivos.
. Sistema de Administracion de Bases de Datos Relacionales (RDBMS)
. Niveles de una Base de Datos.

CAPITULO II – REPASO DE SQL, UN LENGUAJE RELACIONAL.
. Historia y evolución.
. Estructura general de SQL.
  – Data Definition Language (DDL)
  – Data Manipulation Language (DML)
  – Data Control Language (DCL)
. Convivencia entre diferentes DBMS (database management systems).

CAPITULO III – ARQUITECTURA DE UN RDBMS.
. Componentes de un RDBMS:
  – Memoria
  – Disco
  – CPU
  – Red

CAPITULO IV – LA ARQUITECTURA DEL DISCO.
. Tipos de Acceso a Disco.
. Alamecenamientos en Disco
  – Datos.
  – Indices.
. Buffers de memoria.

CAPITULO V – BASES DE DATOS DISTRIBUIDAS.
. Arquitectura Cliente Servidor.
. ODBC Drivers.

CAPITULO VI – CONTROL DE CONCURRENCIA.
. Concepto de LOCK.
. Modos de bloqueo.
. Niveles de Aislamiento.
. Dead Lock

CAPITULO VII – TRANSACCIONES.
. Concepto de Transacción.
. Bitácora de Transacciones.
. Archivos de Bitácoras de Transacciones.
. Long Transactions.
. Transacciones implícitas.

CAPITULO VIII – INTEGRIDAD DE DATOS.
. Concepto de integridad.
. Integridad Física.
  Basados en timestamps.
. Integridad Relacional.
  Integridad Referencial.
  Integridad de Entidad.
  Integridad Semántica.
    Basados en Constraints.
. Integridad Transaccional.
    Basados en Checkpoint

CAPITULO IX – MECANISMOS DE TOLERANCIA A FALLAS.
. Fast Recovery.
. Transacciones.
. Espejeo (Mirroring).
. Arreglos Redundante de Discos (RAID).
. Replicación de Datos (DR).
. Esquemas de Alta Disponiblidad por HW (HA)
. Redundacia completa de Servidores (Tandem)

CAPITULO X – INFORMACIÓN CORPORATIVA Y TIPOS DE BASES DE DATOS.
. Ambientes Transaccionales (OLTP)
. Ambiente de Toma de Decisiones (OLAP)
. Ambientes de Web.

CAPITULO XI – APLICACIONES DE BASE DE DATOS.
. Sistemas de información y operación (transaccionales).
. CRM (administración de la relación con clientes).
. GIS (sistemas de información geográfica).
. MIS (sistemas de información gerencial).

CAPITULO XII – PERFORMANCE AND TUNNING.
. Concepto de Performance.
. Tiempos de Respuesta.
. Deteccion de Cuellos de Botella.
. Tablas temporales y de trabajo.
. Procedimientos y funciones.
. Pruebas de rendimiento.

CAPITULO XIII – TENDENCIAS EN BASES DE DATOS.
. Bases de Datos Orientadas a Objetos.
. Bases de Datos Columnares en Memoria.
. Big Data.
. Bases de Datos Non-sql.
. Datawarehousing, Business Analitics y Business Intelligence.

FORMA DE CALIFICAR:

2 EXAMENES PARCIALES	34%
LABORATORIOS	33%
TAREAS Y TRABAJOS	16%
TRABAJO FINAL	17%

BIBLIOGRAFIA

Database System Concepts
Silberschatz/Korth/Sudarshan
Mc. Graw Hill
2010

Sistemas de Bases de Datos
Conceptos Fundamentales
Elmasri/Navathe
Pearson Educación
2010 México.

Diseño de Bases de Datos.
Gio Wiederhold.
Mc. Graw Hill.
1988 México.

WWW

Informix Homepage
http://www.ibm.com/software/data/informix/

Informix Dynamic Server 11.50 Fundamentals Exam 555 certification preparation
http://www.ibm.com/developerworks/offers/lp/idscert/ids-cert555.html

DB2 10 Certification Tutorials for Fundamentals Exam 610
https://www.ibm.com/developerworks/community/blogs/SusanVisser/entry/db2_10_certificaton_tutorials1

Transaction Processing Performance Council Homepage
http://www.tpc.org/

International Informix Users Group
http://www.iiug.org/

CAPITULO I– CONCEPTOS BASICOS.

Base de Datos.

Es una colección de Datos relacionados entre sí que pueden servir para muchos propósitos y para muchos usuarios.

Base de Datos Relacional.

Es aquella colección de Datos relacionados entre sí que pueden servir para muchos propósitos y para muchos usuarios, que están organizadas en base al álgebra relacional (desarrollada por E.F. Codd). En términos prácticos implica que todos los datos son presentados en forma de tablas con filas y columnas y sus diferentes formas normales.

Bases de Datos relacionales vs Archivos.

Archivos Desventajas:

- Redundancia de Datos.
- Problemas de Integridad.
- Compartición limitada.
- Limitantes en la disponibilidad de los datos.
- Difícil administracion.

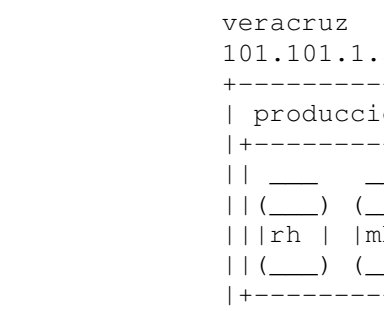
BD Relacional (Ventajas):

- Reduce la redundancia de datos.
- Asegura la integridad de los datos.
- Asegura los esquemas de seguridad
- Acceso concurrente por multiples usuarios.
- Soporta datos compartidos.
- Se ajusta a los requerimientos de cambios, crecimiento y nuevos datos.

Sistema de Administracion de Bases de Datos Relacionales (RDBMS):

Un RDBMS es un sistema que integra datos especifos en una base de datos y hace posible su acceso integral a través del álgebra relacional, funciones y condiciones organizacionales dentro de una empresa.

Se pueden tener uno mas RDBMS dentro de una máquina especifica, a cada uno de estos RDBMS se le conoce como **instancia;** a su vez cada RDBMS puede tener una o mas bases de datos.



En este ejemplo la máquina veracruz tiene dos intancias o RDBMS: una es la de desarrollo y otro es la de producción, cada uno con sus propios recursos, si cualquiera de las dos instancia tiene problema interno no afectará a la otra, como podemos ver cada instancia tiene dos B.D. en el ejemplo, es de observar que puede haber bases de datos homónimas en

+---------------------------+ diferentes instancias.

Dado que el RDBMS o instancia es la parte modular de un ambiente de base de datos es normal que se denomine Engine (o motor) y es por eso que al proceso de ajustar parametros o configuración física (cuando es posible) se le llame afinación (o tunning) ☺.

NIVELES DE UNA B.D.
(Vision Jerárquica de los Datos)

- *Externo*: Modelo de Datos. (E-R). Es la representación lógica que define nuestras unidades de datos y especifica su relación con otras unidades de datos.

- *Conceptual*: Esquema. Es la implementacion del modelo E-R en un lenguaje de Bases de Datos como SQL o Xbase.

- *Interno*: Arquitectura. La forma en la que los datos son almacenados físicamente dentro del almacenamiento primario y secundario en forma de chunks, archivos CISAM, planos, etc.

Conceptos vistos:

Base de Datos.
Base de Datos Relacional.
RDBMS.
Instancia.
Engine.
Modelo de Datos.

CAPITULO II – REPASO DE SQL, UN LENGUAJE RELACIONAL.

El lenguaje de consultas estructurado o SQL por sus siglas en inglés es un lenguaje declarativo, no navegacional, desarrollado a principios de los 70´s por Donald D. Chamberlin y Raymond F. Boyce de IBM para el proyecto Sistema-R de IBM, para así proporcionar acceso y manejo a bases de datos implementando el álgebra relacional desarrollada por Edgar F. Codd (también de IBM) en los años 60´s. Originalmente se llamó SEQUEL (Structured English QUEry Language) pero fue cambiado a SQL por que ya exisitía un producto de una compañía de aviación en Reino Unido con el nombre "SEQUEL".

Los principales objetivos de SQL son:

1. La implementación de las estructuras de datos en cualquier manejador de base de datos en plataformas específicas, es decir, de pasar del modelo conceptual al modelo físico.
2. La obtención, actualización, inserción y borrado de la información en dicha base de datos.
3. Porporcionar medios adicionales para la seguridad en la base de datos y, en el caso de varios motores de base de datos, la administración del motor de base de datos sin que sea un estandard o una necesidad el administarlos desde la interfase SQL.

En el primer caso las sentencias son fundamentalmente aquellas que pertenecen al conjunto de instrucciones que se conocen como DDL, o Data Definition Language, y tienen una relación directa con el diagrama entidad-relación. Sus principales sentencias son las siguientes:

CREATE DATABASE …
DROP DATABASE …
CREATE TABLE …
ALTER TABLE …
DROP TABLE …
CREATE VIEW …
CREATE INDEX …
DROP INDEX …

En el segundo caso, nos referimos a las sentencias que utilizamos para hacer búsquedas, actualizaciones, inserción y borrado de información, y éstas pertenecen al conjunto de instrucciones que se conocen como DML, o Data Manipulation Language, éstas tienen una relación directa con el álgebra relacional, siendo la más compleja las sentencias del tipo SELECT. Sus principales sentencias son las siguientes:

SELECT …
UPDATE …
DELETE …
INSERT …

En el tercer caso, existen sentencias que nos permiten controlar la seguridad, dando o quitando permisos a los usuarios de las bases de datos, por eso se llama a este conjunto de sentencias DCL o Data Control Language, aunque en este último rublo también se suele incluir las extensiones propias de algunos motores de SQL para la

administración de la misma, así como algunas sentencias para controlar los niveles de aislamiento, control de concurrencia y transaccionalidad, dado que estas extensiones no son estándares en todos los motores de base de datos no deben ser vistas como una generalidad. Las principales sentencias abarcadas en esta división son:

. GRANT …
. REVOKE …

Adicionalmente a estas sentencias, los motores de base de datos modernos permiten expandir la capacidad del lenguaje a través de recursos como Procedimientos Almacenados (Stored Procedures) que pueden ser implementados a través de lenguajes propios, o como rutinas en lenguajes de alto nivel como C ó Java, así mismo los motores de base de datos orientados a objetos permiten la creación de funciones definidas por el usuario (UDRs) que permiten polimorfismo (tamién llamado 'overloading' o sobrecarga), es decir, funciones que se comporten de forma diferente dependiendo del número de argumentos o tipo de argumentos que se proporcionen a la función, esto lo veremos más adelante en las bases de datos orientadas a objetos.

Así mismo la gran mayoría permiten la defínción de Triggers que son acciones ante diferentes eventos como UPDATEs, INSERT´s, DELETE´s e incluso en algunos motores de base de datos para SELECT´s, estas acciones pueden ser disparadas antes o después del evento en cuestión y que pueden ser usadas para automatizar ciertas labores, con fines de auditoría o con fines de replicación, aunque para estás últimas dos labores es mejor realizarlas con las herramientas propias de cada motor de datos donde estén disponibles por cuestiones de performance y de funcionalidad.

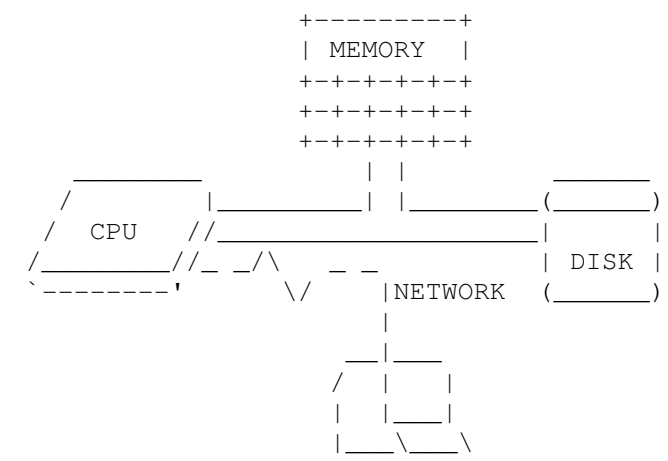
Por último que no menos importante está el mencionar que los motores de base de datos pueden comunicarse entre sí, a través de comunicaciones cliente servidor usando programas conocidos como Gateways, estos Gateways sirven como intérpretes que reescriben los queries recibidos en el "dialecto" del motor de base de datos cliente en el "dialecto" del motor de base de datos servidor, obteniendo la información de éste, y reenviando la información obtenida en un formato válido para el motor de base de datos cliente, la mayoría de los Gateways hacen esta función a través de driver´s ODBC, las comunicaciones cliente-servidor y los drivers ODBC se verán a mayor profundidas en el capítulo V de estos apuntes.

Data Definition Langauge.
Data Manipulation Language.
Data Control Language.
Stored Procedures.
Triggers.
Gateway.

CAPITULO III – ARQUITECTURA DE UN RDBMS.

Componentes de un RDBMS:

Aquí nos referiremos a los componentes de una instancia de base de datos, cada una con sus propios componentes señalados a continuación independientemente que dos instancias corriendo en un mismo equipo deban compartir recursos físicos.



MEMORIA:

Se usa para guardar informacioón relativa a la base de datos (diccionario de datos y catálogo de sistema), almacenamiento de datos para su manipulación o consulta (buffers) y usada para fines de comunicaciones que no involucran canales de comunicacion externos.

Esta divida en al menos tres porciones, cada una de estas porciones se puede ver a nivel de S.O. como un segmento de memoria compartida alojado y apartado.

Resident Portion.- Esta porción no crece, de ahí su nombre, almacena la información del diccionario de datos, catálogo de sistema y los buffers de datos, buffers de physical logs y de logical logs.

Virtual Portion.- Esta porción es dinámica, es decir, puede crecer según las necesidades de memoria, se usa para hacer joins, sorts y guardar información que no se manipula en buffers. Al referirnos al hecho de que esta porcion puede crecer queremos decir que aloja otro segmento de memoria compartida a la porción ya creada.

Message Portion.- Usado para comunicaciones vía shared memory, funciona a modo de un apartado postal, con casilleros específicos, donde el motor busca solicitudes de queries y, una vez completadas, deja sus resultados.

Buffers.- Area de memoria de caracter temporal donde se pueden manejar los datos que son reflejados en disco de forma asíncrona a fin de tener tiempos de respuesta mejores.

¿Por que usar buffers en lugar de accesos directos a disco?

DISCO.

El componente mas crítico e importante del sistema:

- 80% de los problemas de performance estan relacionados con disco.
- Aquí se guarda la información no volátil necesaria para que cualquier RDBMS pueda levantar y funcionar.
- Se guarda información sobre una y cada una de las bases de datos, tablas e índices.
- Aquí se guardan también las bitacoras de transacciones (logical logs), asi como imagenes de paginas antes de modificarse (physical logs).
- Y, por supuesto, aquí se guardan los datos, índices y datos especiales (ie: video, audio, imagenes, etc.)

Arquitectura de disco: The big picture

Fisico:

Page==> Chunk ==> Dbspace

Logico:

Page==>Extent==>tablespace

- Page: Es la unidad báasica de almacenamiento de un RDBMS, el tamaño de página está determinado por el S.O..

- Chunk: Es la unidad de espacio asignado al RDBMS. puede ser bien un disco crudo (raw device) o un archivo a nivel de S.O. (cooked file).

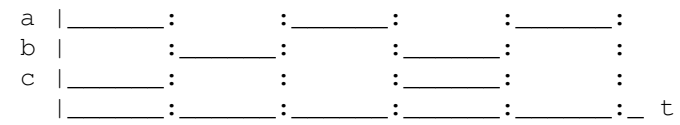
- Dbspace: Es un conjunto lógico de chunks que forma un espacio total que puede ser usado para almacenar bases de datos o tablas en un ambiente RDBMS.

- Extent: Es el conjunto de páginas FISICAMENTE contiguas.

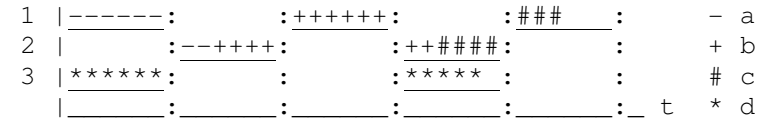
- Tablespace: Es el conjunto de extents asociado a una tabla en particular.

CPU.

Esta componente no es otra cosa que los procesos (programas en ejecucion) que nos permiten manipular y controlar los recursos a disposicion del RDBMS. Anteriormente todos los procesos que solicitaban datos corrian como procesos comunes a nivel de SO, los programas que extraian la informacion del disco, la ponian en memoria, realizaban las busquedas, etc, tambien, pero habia problemas ya que quedaban a merced del contex switch del SO:



Por esto nace el concepto de Virtual Procesor (VP): un programa que simula ser ser CPU con propiedades especificas, que acepta "programas" especificos del RDBMS llamados threads, es decir, los VP's son procesos a nivel de S.O. que simulan ser procesadores de funcion especifica y aceptan codigo virtual en forma de threads que son intercambiados en forma de parámetros.



RED.

Comprende la infraestructura de comunicaciones externas necesarias para el enlace de dos instancias remotas, bien para la compartición de datos o bien para la explotación remota de informacion.

En cuanto a este componente se refiere solamente indicaremos que se compone por el canal físico de conexión y un protocolo de comunicaciones, (ej: Ethernet y TCP/IP), tenemos otras implicaciones pero estas se tratarán en el apartado de comunicaciones Cliente-Servidor.

Solamente resta mencionar que este componente depende muchísimo del S.O. y es la segunda causa de cuellos de botella en ambientes cliente servidor.

Conceptos vistos:

Memory Component.
- Resident Portion.
- Virtual Portion.
- Message Portion.

CPU Component.
- Virtual Processor.
- Thread.
- Context Switch.

Disk Component.
- Page.
- Chunk.
- Dbspace.
- Extent.
- Tablespace

Network Component.
Engine.
Modelo de Datos.

CAPITULO IV – LA ARQUITECTURA DE DISCO.

Tipos de acceso a discos:

Aunque estos tipos de accesos actualmente están en desuso, es conveniente que los analicemos a fin de tener una mejor comprensión histórica de las bases de datos:

Archivos Secuenciales.
Archivos de Acceso Aleatorio.
Archivos Indexados.

Actualmente las bases de datos modernas usan básicamente uno de los siguientes tipos de acceso a disco:

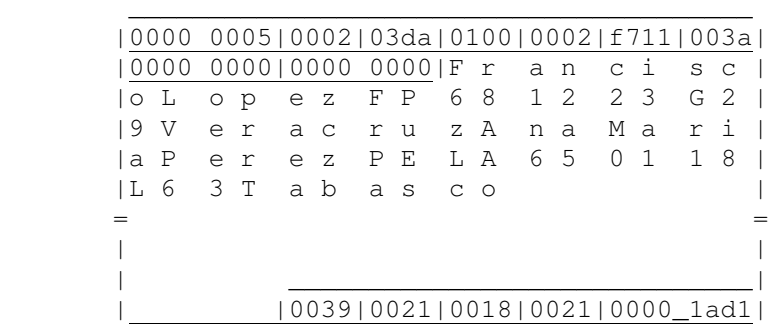
Archivos C-ISAM.
Archivos crudos de arquitectura propia.

En los primero se ofrece el acceso a archivos secuenciales con índice a través de estructuras en C. La mayoría de los manejadores relaciones que no operan con su proipa arquitectura interna hacen uso de este tipo de archivos.

Los archivos crudos de arquitectura propia nos dan la facilidad de adminsitrar a nuestro antojo los recursos de disco haciendo caso omiso de algunas estructuras de control a nivel de S.O. que podrían darnos menor rendimeinto en uso de recursos y tiempo. Independientemente de la arquitectura propietaria de la que estemos hablando, los datos son almacenados en estructuras discretas llamadas páginas, de las que ya hablamos antes, estas páginas guardan los registros físicamente y generalmente tienen una estrutura de control al inicio (header) y un pequeño índice de los registros que estan guardados al final. Por ejemplo, en el caso de informix, existe una estructura de control con una longitud de 24 bits al inicio de la página que tiene un identificador único para cada página en la instancia compuesto por dos estructuras: el indicador del chunk al cual pertence, y el offset dentro de ese chunk en el cuál se encuentra, un checksum una estructura llamada timestamp que se encuentra al final de la página, un identificador o bandera que nos indica para que está siendo usada esta página (datos, indices, logical logs, reservadas, etc.), número de registros (o slots) ocupados en la página, el espacio total disponible en la página (free counter), el apuntador dentro de la página que apunta al siguiente espacio disponible (free pointer), el apuntador a la siguiente página de índices si se trata de índices (next page) y la anterior (previous page):

Al final de la página encontramos un contador de tiempo (o timestamp), cuyo checksum es guardado en el header para fines de integridad, y a su vez, del final hacia atras, aparece el offset en bytes, dentro de la página, donde se encuentran los registros almacenados en ella.

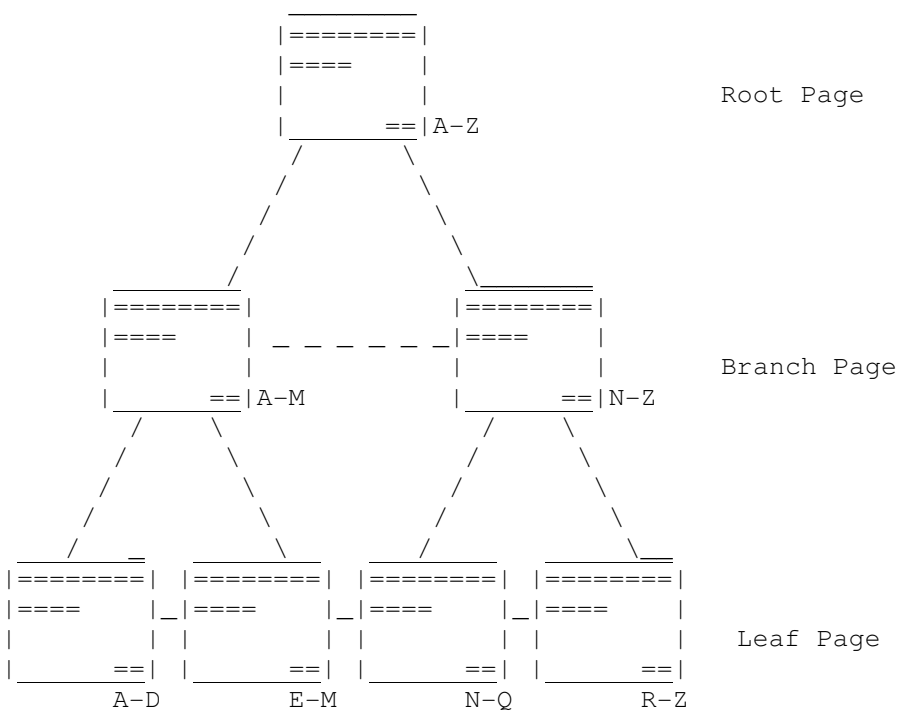
Por ejemplo para una página de datos de informix con solo dos registros:



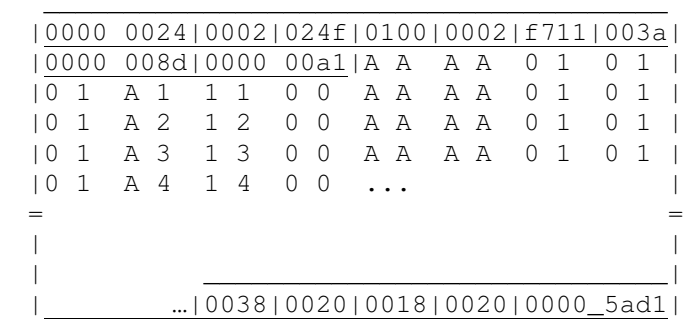
Para los índices usamos una estructura parecida, solo que aquí incluimos los campos llaves en lugar de los registros completos y algunas estructuras de control, como banderas de borrado, etc.

Así mismo los índices son almacenados en estas páginas, a diferencia de los datos en estas páginas solo se guarda el valor por el cual se genera el índice, el row_id o identificador de registro dentro de la tabla que apunta a la dirección física donde radica éste y banderas y estructuras de control adicionales.

La estructura más común para guardar los índices es en forma de arboles balanceados extendidos o B+Tree, es decir, estructuras basadas en páginas con páginas balanceadas como hijas donde el contenido total de registros debe ser más o menos homogéneo, con apuntadores entre páginas del mismo nivel. Por ejemplo supongamos que tenemos un índice por RFC y tenemos 500,000 registros, la estructura quedaría como sigue:



Por ejemplo la página 0x24 del chunk 0x2 tendría un contenido parecido a este:



Como podemos ver el acceso a través de índices es mucho más rápido para búsquedas de pocos registros ya que nos permite, en primer lugar, excluir una serie de valores ya que los datos están ordenados por la llave y, en segundo lugar, encontralos ordenados internamente para subqueries. Sin embargo, como seguramente muchos de ustedes han podido ya imaginar, el costo es alto, de hecho, no es raro encontar tablas donde los índices que se generan sobre de ellas ocupan mayor espacio que la tabla misma.

Así mismo en el disco se guarda información relativa a las bases de datos prsentes en la instancia, tablas, campos y contenido de los mismos. A las tablas que guardan registro de la estructura detallada de cada base de datos incluyendo la estructura de la tabla, constraints, stored procedures y triggers les llamamos diccionarios de datos. La forma en la que los diferentes RDBMS guardan está varia mucho, pueden ser desde bases de datos específicas hasta estructuras en disco guardadas en forma de tablas pero inaccesibles directamente para el usuario.

Manejo de Buffers

Hasta aquí todo va muy bien con el disco, estructuras óptimas, inteligentes, a la medida, pero la política de todo buen RDBMS será evitar al máximo el acceso a lecturas directas a disco. ¿Por qué? Por que es de los pocos elementos de los sistemas de cómputo en los que todavía dependemos de dispositivos mecánicos, es decir, mientras en la velocidad de proceso y manejo de memoria usamos unidades de milesimas de microsegundos (1x10^-12) en los discos, en el mejor de los casos, estamos hablando de tiempo del orden de los nanosegundos (1x10^-9), esto es, literalmente hablando, 1000 veces mayor. Es por eso que una parte importantísima es que se lleven los datos a memoria la primera vez que son leídos, si otro usuario desea leer los mismos datos primero verificará si se encuentran en memoria, de ser así accederá a memoria en lugar de hacerlo a disco. Al área de memoria asignada para tener la copia de los datos presentes en disco en memoria reciben el nombre de buffers. Si se requiere de la actualizacion de los datos (INSERTS, UPDATES o DELETES) estos se realizan primero en el respectivo buffer de memoria y luego se ve reflejado a disco. El área de buffers es variable y si no es lo suficientemente grande puede afectar al rendimiento del RDBMS. Existen otros componentes de disco que, a su vez, utilizan el

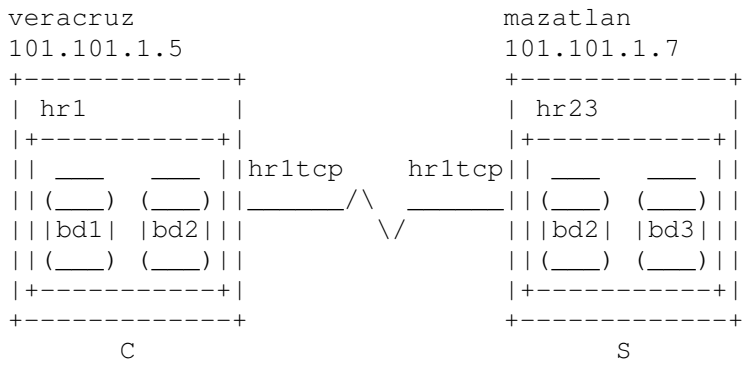
cache para fines específicos, por ejemplo el diccionario de datos, los logical logs, etc.

Conceptos vistos:

Estructura de páginas de datos
Estructura de páginas de índices
Estructuras B Tree
Estructuras B+ Tree
Diccionario de Datos.
Buffers.

CAPITULO V – BASES DE DATOS DISTRIBUIDAS.

Con el avance en materias de redes, reduciendo costos e incrementando velocidades, la idea de compartir datos remotamente fue siendo cada vez más atractiva y factible, su principal bondad es la de evitar, si no eliminar, la redundancia de datos en una organizacion con varios ambientes autónomos de bases de datos, mediante este esquema permite el acceso remoto a bases de datos en otros ambientes e incluso servidores a fin de incluirles en queries locales.




```
DATABASE bd2;
SELECT A.fname, A.lname, B.salary
   FROM customer A, bd2@hr23:customer B
   WHERE A.rfc=B.rfc;
```

Syntaxis: database@dbservername:table.field

Por otra parte este esquema permite la consulta remota desde un ambiente autónomo aunque este no posea un motor de base de datos, tal es el caso de sistemas que aprovechando sistemas operativos gráficos como Windows 9x o Mac OS, son desarrollados en forma de interfases gráficas desarrolladas en front ends como Visual Basic, Power Builder, Delphi, etc. y comunicados a través de drivers propietarios o a traves de ODBC's drivers (Open Data Bases Conection), que es un estándar de conexión para Bases de Datos en ambientes gráficos ampliamnte usado para comunicaciones cliente-servidor).

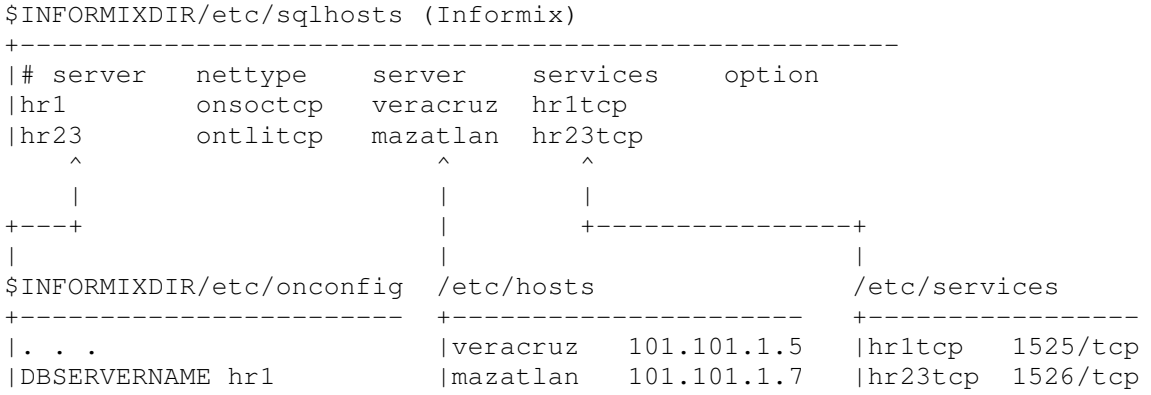
El esquema cliente-servidor toma su nombre de sus dos componentes principales: el cliente que es la aplicación o la instancia local que solicita la información y el servidor, la instancia de base de datos que tiene la información requerida y la hace llegar al cliente a través de la red.

Para el envio de la informacion, ademas de valernos de una red física y un protocolo de comunicaciones (Ej. Ethernet y TCP/IP), los manejadores de bases de datos utilizan un protocolo adicional que corre sobre el protocolo de comunicaciones propiamente dicho, recordando que un protocolo es un conjunto de reglas que deben obervarse para establecer la comunicación de dispositivos entre sí. Este protocolo es conocido como “two phase commit”, cuya función es la de asegurar que la información enviada entre cliente y servidor sea correcta y entendida. Este protocolo implica que las validaciones son bidireccionales (S->C y

C->S) y por la naturaleza del protocolo implica un esquema de operación síncrono. Así mismo necesitamos de una interface de programación en red (network programming interface) que no es otra cosa que un driver que contiene blibiotecas de rutinas para permitir la comunicación en la red. Un ejemplo de una NPI para Unix es el TLI (Transport Layer Interface), un ejemplo de una NPI para NT es WINSOC (sockets programming interface).

Para establecer una comunicación cliente-servidor necesitamos saber:

1 ¿De qué y hacia que máquina nos vamos a conectar?
  Archivo de hosts.
2 ¿Qué canal de TCP/IP vamos a usar para comunicarnos?
  Archivo de services.
3 ¿Como se llama nuestra instancia?
  Variable de ambiente que distingue a nuestra instancia
  de todas las demás.
4 ¿Como unimos todas las piezas?
  Archivo de configuración de comunicaciones C-S.



Conceptos vistos:

Cliente.
Servidor.
Arquitectura Cliente-Servidor.
Protocolo.
ODBC.
Two phase commit protocol.

CAPITULO VI – CONTROL DE CONCURRENCIA.

Como vimos en capítulos anteriores, una de las características más importantes en un RDBMS es la capacidad de poder distribuir información entre varios usuarios al mismo tiempo, a esto es a lo que llamamos concurrencia, para el manejo y control de la concurrencia hacemos uso de ciertos recursos llamados LOCK's (o candados).

Un lock es un recurso que nos permite el bloquear el acceso a una base de datos, una tabla, una página de información o un registro.

Aquí vale la pena hacer mención sobre el bloqueo a nivel de registros y a nivel de páginas; como recordaremos en nuestro capítulo de arquitectura interna la unidad básica de almacenamiento es la página, de tamaño fijo y definido a nivel de S.O., una página puede contener más de un registro (el máximo de registros por página en informix es de 256), al definir la tabla se puede elegir si se desea usar bloqueo por registro o bloqueo por página (este último es el default), al bloquear por registro se pondrá un lock por registro y solo este registro quedará restringido, por el contrario, si el modo seleccionado es por página, al momento de acceder a un registro dado se bloqueará no solamente este registro sino todos aquellos que compartan esta misma página.

Los tipos de bloqueo que podemos imponer con estos recursos son:

- Compartidos
- Exclusivos

Un bloqueo compartido es aquel que se pone cuando el registro está siendo accedido para consulta nada más.

Un bloqueo exclusivo es colocado a nivel registro o página cuando estamos haciendo actualizaciones o inserciones a una tabla determinada y a una tabla o base de datos cuando se está recreando la tabla o lo solicitemos explícitamente.

Un bloqueo exclusivo no está sujeto a negociación, es decir, una vez que el registro, página, tabla o base de datos ha sido bloqueado en modo exclusivo nadie puede acceder a esta ni siquiera para consulta (mucho menos para actualización o inserciones) hasta que ésta sea liberada.

Un bloqueo compartido puede tener varias acciones dependiendo de ciertas políticas que pueden ser definidas a nivel de sesiones usando comandos SQL. A dichas políticas les denominamos **"Niveles de Aislamiento"**.

Los niveles de aislamiento se aplican cuando alguien trata de acceder a un registro o página que esta siendo accedido para consulta por otro usuario. Los niveles de aislamiento disponibles normalmente son los siguientes:

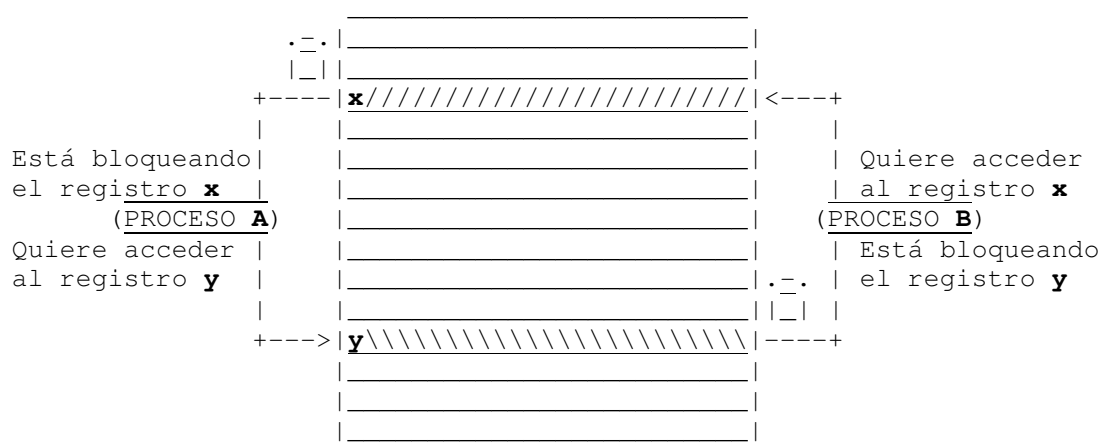
- Dirty Read: No hace validación de ningún tipo, si el registro o página no está bloqueado en modo exclusivo lo lee como si nadie más lo estuviera leyendo en ese momemnto.

- Commited Read: Si el registro o página está siendo leído por alguien más el proceso que desee leerlo debe esperar a que este registro o página sea liberado por el otro usuario (en la mayoría de los RDBMS esté es el default).

- Cursor Stability: Este nivel se usa solo en aplicaciones que hagan uso de cursores (arreglos en memoria usados por aplicaciones 4GL o lenguajes embebidos para manipular un subconjunto de datos extraídos de una base de datos), todos los registros involucrados en el cursor tendrán un bloqueo compartido, éste no se quitará hasta que el cursor haya sido liberado.

- Repetable Reads: Este nivel de aislamiento aplica a bases de datos con transacciones únicamente, este nivel colocará un bloqueo compartido a uno y cada uno de los registros o páginas que sean leídas dentro de una transacción y no serán accesibles hasta que la transaccíon se aplique (comitt) o se le dé marcha atras (rollback).

En estas condiciones nos encontramos con fenómenos muy interesantes, particularmente un riesgo muy alto es el de la presencia de dead locks (o abrasos mortales). Un deadlock es la condición que se presenta cuando dos o más procesos tratan de accesar recursos mutuamente cuando estos se encuentran bloqueados. Por ejemplo:



En este ejemplo, el proceso **A** tiene un lock en el registro **x** y quiere bloquear para lectura el registro **y**. Sin embargo el registro **y** está siendo accedido por el proceso **B**, esto significa que el proceso **A** esperará a que el registro **y** sea liberado, pero ¿qué sucede si el proceso **B** tratá, a su vez, de leeer el registro **x** y bloquearlo para lectura? Se presentará un círculo vicioso donde el proceso **A** estará esperando por el registro **y**, pero que jamás podrá acceder hasta que el

proceso **B** acceda al registro **x** que éste tiene bloqueado y termine. Un deadlock es un problema serio en el que ambos procesos tendrían que esperar por siempre, peor aún está situación podría paralizar la mayor parte de nuestro actividad en el RDBMS. Para solucionar este tipo de problemas el RDBMS toma actúa de la siguiente manera:

El RDBMS tiene una relación de recursos bloqueados en el sistema, antes de que un lock sea otorgado la lista de locks de cada usuario es examinada. Si un lock está actualmente en uso en dicho registro se identifica el usuario que posee dicho lock y se checa si este usuario está a la espera de locks pertenecientes al usuario que desea accesar al registro. Si es así el proceso que solicita el recurso es abortado y el mensaje de error de ejecución “Deadlock detected” es retornado.

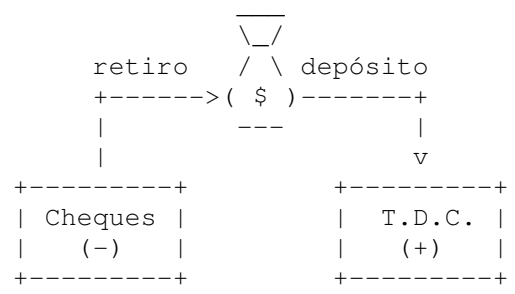
Conceptos vistos:

Concurrencia.
Lock.
Bloqueo compartido.
Bloqueo exclusivo.
Niveles de bloqueo.
Niveles de aislamiento.
- Dirty read.
- Commited read.
- Cursor stability.
- Repeteable read.

Deadlock

CAPITULO VII – TRANSACCIONES.

Una transacción es un conjunto de operaciones que deben realizarse todas o ninguna. Un ejemplo claro de esto es una transferencia bancaria, por ejemplo un pago a tarjeta de credito:



Esta operación requiere de dos pasos: un retiro y un depósito, para que dicha operación se considere exitosa deben realizarse ambas, de lo contrario ninguna debe ser aplicada, este ejemplo puede antojarse simplista, pero imagina miles de operaciones de este tipo en línea, bien por sistema o bien por internet. Si alguna operacion de este tipo quedara a medias y en firme, el resultado sería una base da datos incosistente, ya que la información almacenada en la BD no corresponderia con la realidad (una cuenta de la que se sacaron $1,000.00 pero que nunca se depositaron en la segunda cuando en la realidad la operación se llevo a cabo)

Para este fin necesitamos de un medio que nos permita mantener registro de cada actividad perteneciente a cada transacción en el ambiente de base datos, este espacio destinado a guardar las transacciones le llamamos bitácora de transacciones o logical log.

Esta bitácora es usada para cuatro cosas:

- Dar marcha atras (rollback) a la transacción en caso de errores o por código.
- Asegurar la integridad relacional de la base de datos en caso de caída total del RDBMS o del equipo (lo veremos en un capítulo de integridad de datos).
- Dado que las transacciones guardan todos los SQL que tienen que ver con actualizacion (UPDATE, INSERT o DELETE) pueden respaldarse a fin de utilizarles para llevar a un punto en el tiempo más reciente en caso de tener que hacer un restore de un respaldo (lo veremos en mecanismos de disponibilidad).
- Para fines de repliación de datos basadas en transacciones (lo veremos en mecanismos de disponibilidad).

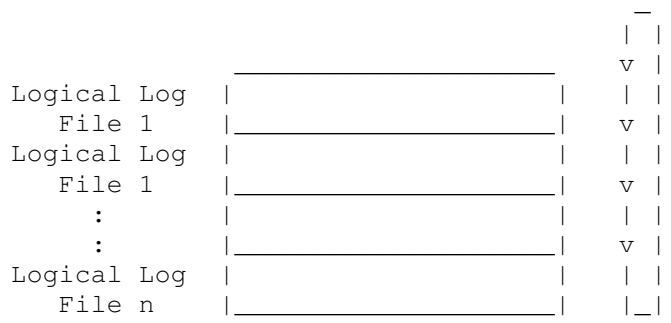
La bitácora de transacciones, de ahora en adelante logical logs, reside en un espacio en disco finito y determinado, aunque es posible que se guarden en archivos externos a nivel de S.O. lo más común es que sean administrados en una porción de uno o varios chunks del RDBMS y rara vez, por no decir nunca, radica en un solo archivo, o logical log file, de hecho, es raro y muy poco recomendable, el tener menos de tres logical logs files.

Cabe mencionar que cuando a una transacción se le da marcha atras, los pasos necesarios para regresar al principio de esta, (por ejemplo para un INSERT en la transaccion necesitamos aplicar un DELETE), son almacenados también en la bitácora de transacciones y reciben el nombre de registros de compensación.

Los logical log files son unidades discretas y que se liberan cuando:

- Todas las transacciones que tienen operaciones en dicho logical log file han terminado exitosamente (COMMIT) o han terminado de dar marcha atrás (ROLLBACK) y
- Haya sido respaldadas en caso de que se haya configurado el uso de respaldo de logical logs.

Estos logical logs files se usan de forma circular, es decir, se comienza en el logical log file 1, una vez lleno se usa el dos, que a su vez al llenarse nos forza a usar un tercero, si éste es el último y el primero ha sido liberado, procederemos a usarlo nuevamente y así sucesivamente.



Sin embargo ¿qué sucedería si el tercer logical log file se llena y el primero aún no ha sido liberado? En esta situación, logical log files llenos, la bitácora ya no tendría mas espacio para guardar las transacciones entrantes, peor aún, ni siquiera tendría la posibilidad de abortar las transacciones presentes por que no podría almacenar los registros de compensación. Esto puede llevarnos a un estado de inactividad total, más aún, algunos manejadores de bases de datos, como informix, notifican de esta circunstancia y tiran el motor de base de datos a fin de no causar mas daño en la integridad transaccional de los datos, previa alarma de que los logical logs se estan llenando.

Por si esto no fuera suficiente, el motor quedará atorado en el intento por arrancar al realizar la operación conocida como Fast Recovery. Un proceso de revisión y corrección de posibles inconsistencias transaccionales que comentaremos en un capítulo posterior.

La única manera de levantar el motor en estas condiciones es:

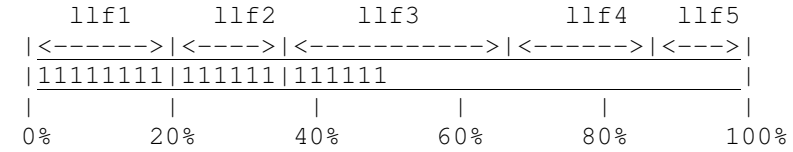
- Recuperando de un respaldo.
- Añadiendo, fuera de línea, más logical logs fies (lo que no siempre es posible).

- Cambiando los catálogos internos para que ignore las transacciones pendientes, dejando a la base de datos en un estado probablemente inconsistente.

Como podemos ver esta situacion no es en lo absoluto deseable y debe ser evitada a toda costa. De hecho esta situación es PERFECTAMENTE PREVESIBLE Y EVITABLE, las causas que llevan a esto son, generalmente, las siguientes:

- **Cuando no se han respaldado los logical logs y por ello no pueden ser liberados:** Si uno no está interesado en respaldar los logical logs uno debe configurar el RDBMS para que libere el logical log file en cuanto sus transacciones terminen. O

- **Cuando se presenta una long transaction:** Entendiéndose por long transaction a aquella transacción que puede llegar a ocupar todos los logical logs; para evitar que una posible long transaction llene los logical logs, casi todos los manejadores de bases de datos tienen niveles, o marcas de agua, de uso del espacio total de los logical log files a nivel de transacción, de llegar una transacción a este nivel (por ejemplo 50%) la transacción es abortada y el mensaje de error de long transaction es devuelto al usuario en tiempo de ejecución. Para evitar que procesos sean abortados por long transaction (LNTX) se debe incrementar el espacio disponible en logical logs files o, si las marcas de agua estan muy bajas (30% por ejemplo), se puede incrementar el umbral aunque no es lo más recomendable.



LTXHWM= 50

Por último, aunque decidamos que nuestra base de datos no sea transaccional, aun así existirán operaciones que generen necesariamente transacciones, tal es el caso de la creación de tablas, reestructura de las mismas o borrado. A este tipo de transacciones les llamamos **transacciones implícitas**.

Conceptos vistos:

Transacciones
   COMMIT
   ROLLBACK
Bitacora de transacciones
Logical Log File
Respaldo de logical logs.
Long Transaction
System Down
Transacciones implicitas.

CAPITULO VIII – INTEGRIDAD DE DATOS.

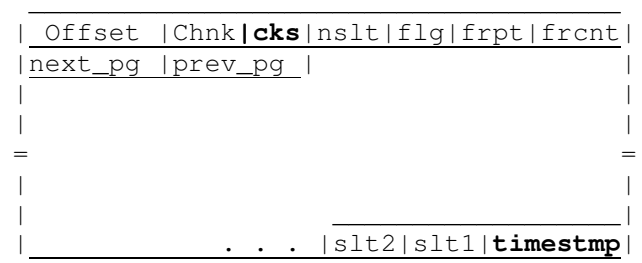
El concepto más básico y sencillo para integridad o consistencia de datos es la propiedad de los datos de reflejar fielmente la realidad exterior que representa. Esto es, si un dato dice que Almacenes la Negrita tiene 20´000,000.00 en su cuenta de cheques sea exacto y cierto, que solamente exista una sola vez este cliente en el sistema y, por supuesto, que la información pueda ser recuperada con exactitud.

Esto se antoja sencillo, pero en realidad es mucho más complejo de lo que se vé, primero tenemos que ver que los datos no hayan sido almacenado mal a consecuencia de un apagón sin UPS mientras se lleva acabo la actualización física de datos al disco, que acciones tomar cuando la llave primaria de un dato está repetida, o peor aún, cuando una transacción es abortada y sabemos que cada actualización hecha dentro de ella debe ser regresada.

Como podemos ver el concepto de integridad es demasiado amplio, es por ello que existen diferentes niveles de integridad.

INTEGRIDAD FISICA.

Consiste en que los datos esten correctamente alamcenados en el disco, que no hayan quedado a medio actualizar por culpa de un apagón al escribir datos actualizados al disco, o inclusive, el que se encime información entre dos chunks de información. Para garantizar este tipo de integridad a nivel de página usamos las marcas de tiempo o timestamps definidas en el capitulo de la estructura de disco.



Cuando se lee la página se obtiene un checksum del timestamp ubicado al final de la página y se compara con el almacenado en el header de la página, de esta manera podemos garantizar que la información actualizada o creada no quedo a medias. Para ello antes de escribir o actualizar cualquier dato a la página se actualiza el checksum del timestamp (en la cabecera) se realiza la actualización y hasta que toda la información ha sido actualizada no se actualiza el timestamp al final de la página, ¿Qué quiere decir esto? Que cuando estemos en el "salto de la muerte", es decir escribiendo a disco, y algo muy malo sucede, dejando la escritura de la página “a medias”, el RDBMS tiene una forma de detectar este tipo de corrupción si el checksum del timestamp no coincide con aquel que aparece en el header, evitando así problemas de integridad de este tipo, evitando encontrar nombres como "Pedroisco" en lugar de “Pedro” o “Francisco”.

INTEGRIDAD RELACIONAL.

Aplica a la necesidad de asegurarse de que los datos almacenados cumplan con el modelo E-R planteado para representar la realidad, que quiere decir, que los campos definidos como llaves primarias sean únicos y no nulos, que las llaves foráneas existan como primarias en otra tabla, que los valores definidos como no duplicados y no nulos realmente lo sean y que los datos definidos realmente contengan el tipo de dato definido. Para ello nos valemos de "condiciones" o CONSTRAINTS, un constraint es una condición obligatoria definida a nivel de la base de datos y que es validada automáticamente por la base de datos. Generalmente se definen al crear la tabla, por ejemplo:

```
CREATE TABLE customer(
  customer_num SERIAL,
  fname CHAR(20),
 PRIMARY KEY (customer_num)
 CONSTRAINT pk_cnum
);
```

```
CREATE TABLE orders(
  order_num SERIAL,
  customer_num INTEGER,
 FOREIGN KEY (customer_num)
 REFERENCES customer
  CONSTRAINT fk_cnum
);
```

Los tipos de integridad relacional son:

- **Integridad Referencial (Referential Integrity):** ¿Están establecidas y garantizadas las relaciones entre tablas? FOREING KEY, PRIMARY KEY CONSTRAINTS.

- **Integridad de Entidad (Entity Integrity):** ¿Tiene cada registro en la tabla un identificador único? PRIMARY KEY CONSTRAINT.

- **Integridad Semántica (Semantic Integrity):** ¿Reflejan adecuadamente los datos en las columnas respectivas el tipo de información para el cual la columna fué diseñada? CHECK CONSTRAINT.

INTEGRIDAD TRANSACCIONAL.

Se refiere a la propiedad de una transacción de realizar todas las operaciones que la componen o ninguna. En caso de incosistencias, quizá sea la más compleja y difícil de detectar de todas ya que las operaciones que componen a las transacciones pueden terminar exitosamente y sin embargo el RDBMS puede reflejar una realidad completamente falsa ¿Recuerdan el ejemplo de la transaccion bancaria?

Ahora bien, para asegurar la integredidad transaccional nos valemos de tres cosas:

- Los logical log files y
- Physical log file y
- Checkpoints.

El physical log es un espacio en disco donde guardamos la "imagen de antes" de una página que esta siendo acualizada o borrada en memoria, la idea es que si por alguna falla el motor llegara a caerse con datos

actualizados en memoria, podamos recuperar al menos su contenido en el ultimo punto de consistencia conocida (checkpoint).

Un checkpoint se define como un punto en el tiempo donde la consistencia física y transaccional estan reconocidas. Estos checkpoints son periódicos y nos dan puntos de refencia donde la integridad transaccional es conocida.

Las acciones que tienen lugar durante un checkpoint son:

- Se detiene toda actividad de escritura o actualizacion en la instancia.
- Todas las páginas modifcadas en memoria son bajadas a disco.
- Se vacia el physical log, ya que todas las paginas modificadas en memoria ya han sido escritas a disco.
- Se inserta un registro de checkpoint en el logical log file, mismo que contiene información sobre la fecha y hora del checkpoint e información sobre una y cada una de las transacciones que permanezcan abiertas al momento del checkpoint.
- Los buffers de los logical logs se vacían a disco.

Esto nos da un punto conocido de consistencia. Ya que sabemos que la informacion presente en los buffers (volátil en caso de falla) esta reflejada en el disco y que tenemos toda la informacion relacionada con transacciones para dejar la base de datos perfectamente consistente.

Conceptos vistos:

Integridad.
   Integridad Física.
   Integridad Relacional.
      Integridad referencial.
      Integridad semántica.
      Integridad de entidad.
   Integridad Transaccional.
Physical Log File.
Checkpoint.

CAPITULO IX – MECANISMOS DE TOLERANCIA A FALLAS.

FAST RECOVERY.

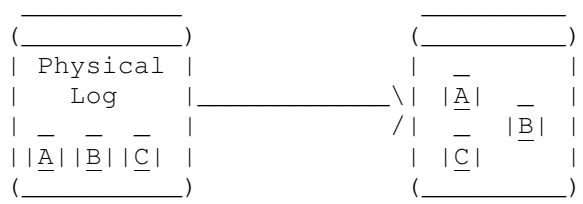
La primera preocupación al momento de ocurrir una falla que provoque una caída anormal de RDBMS es el de verificar la integridad física y transaccional (o lógica) de las diferentes bases de datos. Por ello al levantar el motor se realiza la operación denominada fast-recovery que se define como aquella operación mediante la cual se regresa al último punto de consistencia conocido (checkpoint) aplicándose todas las transacciones terminadas y dándole marcha atras a aquellas inconclusas al momento de la falla.

Este proceso consta de dos pasos:

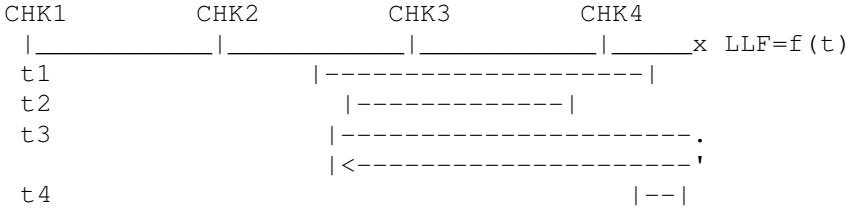
Physical Recovery: En este paso se recuperan las imagenes de antes de las páginas modificaadas a partir del último checkpoint, guardadas en el physical log, a fin de garantizar el estado físico de la información al momento de dicho checkpoint.

Logical Recovery: En este paso se aplican los pasos posteriores al checkpoint correspondientes a las transacciones activas al momento de checkpoint. Dándoseles comitt a las transacciones acabadas y rollback a las que al momento de falla aún no hayan terminado.

Physical Recovey:



Logical Recovey:



Usando el diagrama anterior: supongamos que donde esta marcada la x hubo una caida anormal del sistema: El último punto conocido de consistencia es CHK4, por lo que se recuperan las imagenes de “antes” de todas las páginas que pudieron ser modificadas desde entonces (Physical recovery), a patir de ahí se aplican los pasos de la transacción t1 hasta que se le da commit. t2 ni siquiera es conteplada, t3 quedó inconclusa al momento de falla por lo que deberemos darle marcha atrás completamente, es por esta razon que el log no se libera si tiene alguna transaccion activa, nótese el número de registros de compensacion necesarios y t4 aunque no está anotada en último checkpoint se aplica por estar en la bitácora de transacciones.

TRANSACCIONES.

El proceso de Fast Recovery tiene lugar siempre, independientemente de que las bases de datos en nuestra instancia tengan o no logging. Sin embargo, el tener logging en una base de datos tenemos la capacidad de salvar mas operaciones que aquellas que se presentan en firme. También veremos, en el siguiente apartado que el tener la base de datos transaccional y respaldar los logical log files nos puede ahorrar mucho, pero muchísimo trabajo.

RESPALDOS (BACKUP).

Un respaldo es la copia física de las estructuras guardadas en disco del RDBMS a fin de ser almacenadas y ser recuperadas en caso de falla de disco o de perderse la consistencia del sistema.

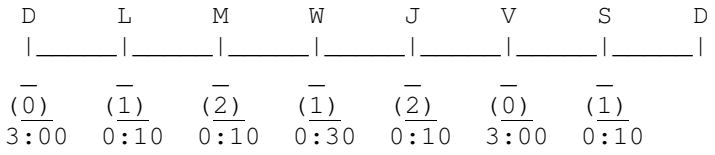
Los respaldo pueden ser de dos tipos:

- De datos o físicos.
- De transacciones.

Respaldos de datos:

Los respaldos de datos son, como su nombre lo indica, la copia física de una y cada una de las páginas que contienen datos, índices, tablas de catálogo de sistema e información elemntal para poder levantar la instancia y recuperar el respaldo.

Dado que un respaldo total de una base de datos pudiera tomar demasiado tiempo, existe la facilidad de respaldar solo las páginas que se hayan modificado desde el último respaldo total del sistema, a este tipo de respaldos les llamamos incrementales, en este contexto a los respaldos totales de sistema se le llama de nivel 0, al respaldo que se realiza sobre las páginas modificadas desde este respaldo total se les denomina como respaldo a nivel 1, algunos manejadores permiten un nivel más, el nivel 2 que es aquel respaldo incremental que se efectua sobre aquellas páginas que han cambiado desde el último respaldo a nivel 1. Si bien estos respaldos toman menos tiempo, este tiempo debe ser invertido al momento de la recuperación, por ejemplo:



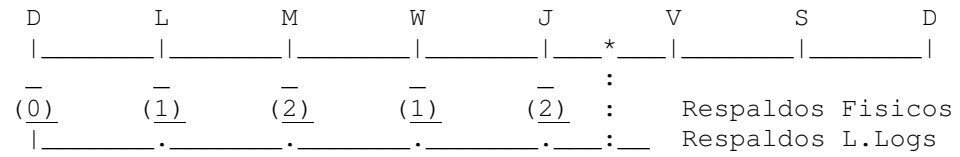
Observese la política y la periodicidad de los respaldos, observese cada cuanto se toma un respaldo a nivel 0 respecto al 1 y al 2. ¿Alguna semejanza con el algoritmo para la resolución de las torres de Hanoi? Sí, es el mismo. El respaldo a nivel 1 tomado el Miércoles suplanta al hecho el Lunes así como el respaldo a nivel 2 del Martes, al llegar el Viernes los respaldos anteriores pierden viencia, sin embargo es siempre muy recomendable el guardar al menos dos respaldos anteriores por si la media llegara a dañarse.

En caso de falla, supongamos que un disco se hecha a perder el Viernes, suponieno que el respaldo a nivel 0 del viernes jamás fue tomado, se reemplaza el disco y procedemos a bajar el respaldo a nivel 0 del Domingo (3 hr.), luego el respaldo a nivel 1 del Miércoles (30 min) y por último el respaldo a nivel 2 del Jueves (10 min), 3 horas 40 min. en total.

Respaldos de transacciones:

Los respaldos de transacciones nos pueden ayudar a ahorar trabajo al momento de llevar la base de datos al último momento en el tiempo antes de la falla, si recordamos bien la bitácora de transacciones guarda, en forma compacta, todas las operaciones que afectan el contenido de la B.D., es decir, podemos salvar horas y horas de trabajo tan solo con respaldar la bitácora de transacciones. Mejor aún, algunos manejadores ofrecen la facilidad de recuperar en un punto en el tiempo. Por ejemplo, supongamos el siguiente caso:

Se toman los siguientes respaldos y el Jueves, a las 5:00 PM, se borra por error una tabla crítica para el sistema, representada aqui por un asterisco



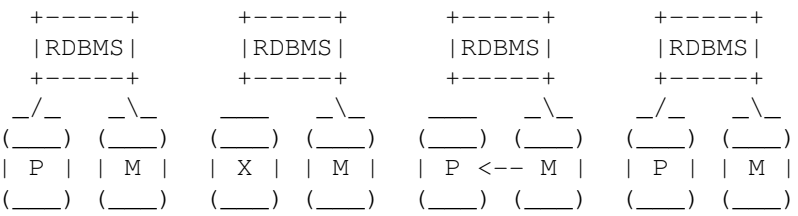
Obviamente requerimos de un restore: aplicamos el respaldo a nivel 0 del Domingo, el nivel 1 del Miércoles y el 2 del Jueves, por último aplicamos los respaldos de logical logs a partir del jueves hasta las 4:55. Notese los puntos (.) en la linea de logical logs, estos puntos, dependiendo del RDBMS y la herramienta empleda, descartan los logs anteriores o permiten su uso, así otra solución podría ser recuperar el respaldo a nivel 0 del Domingo y aplicar todas las transacciones desde el Domingo hasta el Jueves a las 4:55 PM. Generalmente este proceso suele ser mas tardado que los incrementales. Cabe mencionar que, por razones obvias, el respalo de logical logs vuelve a empezar al hacer un respaldo a nivel 0.

Si deseamos mantener un respaldo de los logical logs se recomienda que este sea continuo, es decir, que se realice con una unidad dedicada, recordemos que de no respaldar estos logical logs files si le especificamos al RDBMS que queremos respaldarlos puede derivar en un System Down.

ESPEJEO (MIRRORING).

Un respaldo puede solucionar, eventualmnte, casi cualquier contingencia, si embargo existen aplicaciones que nunca debieran detenerse, o detenerse lo mínimo solo en casos emergentes, a este tipo de sistemas se les conoce como de **Misión Crítica**, el siguiente nivel de seguridad de datos consiste en atacar la mayor causa de problemas graves: el disco, sabemos que una falla en disco significa pérdida de información, tener que dar de baja el motor y recuprar de respaldo. Una

alternativa consiste en tener duplicada la información en dos discos en lugar de uno, así si uno de los discos falla el segundo entra en su lugar y el RDBMS jamás se da de baja, mejor aún, se puede cambiar el disco dañado, actualizar la información en el disco nuevo (a esta actividad le llamamos sincronización), completamente en línea:



ARREGLOS REDUNDANTES DE DISCOS INDEPENDIENTES (RAID).

En 1988, David A. Patterson, Garth Gibson, y Randy H. Katz de la Universidad de California en Berkeley publicaron un documento llamado *"A Case for Redundant Arrays of Inexpensive Disks"* o Un caso de Arreglos de Redudantes de Discos sin Costo, en el cual se manejan cinco modelos de arreglos de disco para incrementar la seguridad de los datos en caso de problemas de hardware, o ***niveles de RAID***. Ellos catalogaron sus modelos RAID (Redundant Arrays of Inexpensive Disks) del nivel 1 al 5. Dado que "sin costo" es un término muy relativo, la industria a reemplazado el término ***Inexpensive*** original en RAID por el de ***Independent*** ya que los discos que comprenden el arreglo son unidades independientes. Los niveles RAID originales son:

- ***RAID Nivel 1***, también conocido como *disk mirroring*, nos protege contra fallas en disco replicando toda la información contenida en el disco al menos una vez en otro dispositivo. Esto ofrece una disponibilidad extremadamente alta a un costo relativamente bajo. Para algunas aplicaciones con actividad de I/O muy intensivas, un arreglo con RAID Nivel 1 puede mejorar el performance en comparación con un solo disco.

- ***RAID Nivel 2*** provee redundancia usando el código de Hamming. Los datos y un código para la detección de errores son manejados a lo largo de todos los discos del arreglo a nivel de bit. El código de detección y corrección de datos es el código de Hamming ampliamente usado para la detección y corrección de datos en la memoria RAM. Dado que el código de Hamming es usado tanto para la detección como para la corrección, el RAID Nivel 2 no hace uso completo de todas las capacidades de corrección de errores que comúnmente vienen incorporadas dentro de las controladoras de disco. Las propiedades del código de Hamming también restringen las configuraciones posibles para el RAID Nivel 2. Es más, el RAID Nivel 2 no ha sido implementado tan ampliamente en productos disponibles comercialmente.

- ***RAID Nivel 3*** usa un disco de paridad para almacenar información redundante sobre la data de los otros discos. El RAID Nivel 3 trabaja relies en estrecha coordinación del S.O. como miembro de las actividades de disco. El RAID Nivel 3 es óptimo para aplicaciones en las que grandes bloques de información secuencial deben ser transefridas rápidamente pero no es recomendable para el procesamiento de transacciones.

- ***RAID Nivel 4*** también usa un disco de paridad para almacenar información redundante sobre los datos guardados en los otros discos. La diferncia entre el nivel 3 y el nivel 4 es que el RAID Nivel 3 opera los discos pertenecientes al arreglo al unísono, mientras que el RAID Nivel 4 los opera independientemente.

- ***RAID Nivel 5*** usa la capacidad de almacenamiento equivalente al de un disco en el arreglo para almacenar la información de paridad de los datos almacenados en los discos restantes del arreglo. Difiere del RAID Nivel 3 en que los discos en el arreglo operan independientemente los unos de los otros, además deque la información redundante de los discos restantes está distribuida entre todos los discos de arreglo. RAID Nivel 5 ofrece las ventajas de confiabilidad del mirroring con el stripping. A menos de que un cache se use, puede haber un degradamiento substancial del performance en comparación al acceso a disco convencional. Un RAID Nivel 5 se recomienda para aplicaciones cuyas cargas de I/O consisten fundamentalmente en un enorme número de lecturas asíncronas.

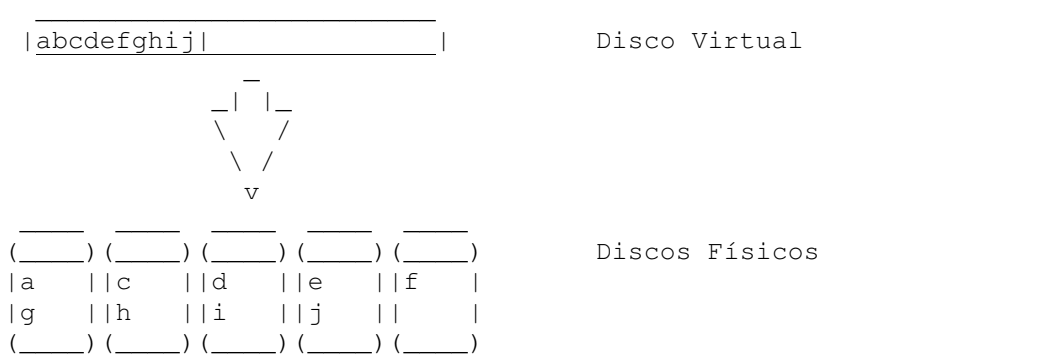
Desde la publicación del documento original en Berkeley, un sexto nível de RAID ha sido descrito por los autores originales. El RAID Nivel 6 usa un segundo disco más conteniendo, por segunda vez, información redundante para proporcionar protección contra pérdida de datos ocacionada por una doble falla en disco, obviamente también cubre contra falla en un solo disco del arreglo.

Así mismo se ha generado el término **RAID Nivel 0**, que es comunmente referido al stripping, operación mediante la cual se distribuye pedazos de información entre los diferentes discos del arreglo como si fuesen “tiras”, la razón por la que se le conoce como RAID es por que el mapeo de la data en los discos es parecido al que hacen los arreglos de disco con RAID, aunque, como podemos percatarnos, NO existe redundancia de datos en un RAID nivel 0, sin embargo el término es por demás aceptado y completamente usado por la industria y el medio.

También se encuentran en el mercado combinaciones de los Nivelels de RAID arriba descritos en muchos productos comerciales. La combinación más popular es el RAID 10 o, más comunmente 0+1, la cual combina el stripping del RAID Nivel 0 y el mirroring del RAID Nivel 1 en un solo arreglo que provee de disponibilidad de datos y mejoras en el perfrormance de la I/O a través del striping.

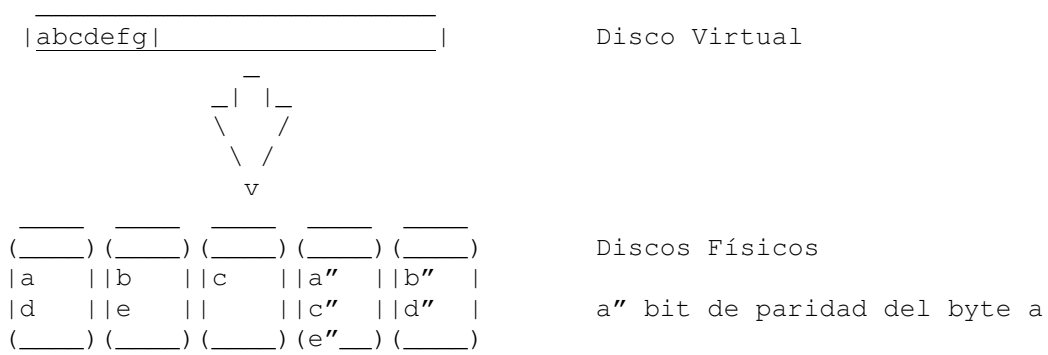
Los niveles mas comunes de RAID implementados por los fabricantes de hardware son 0,1,3,5 and 0+1.

RAID 0 (Stripping)



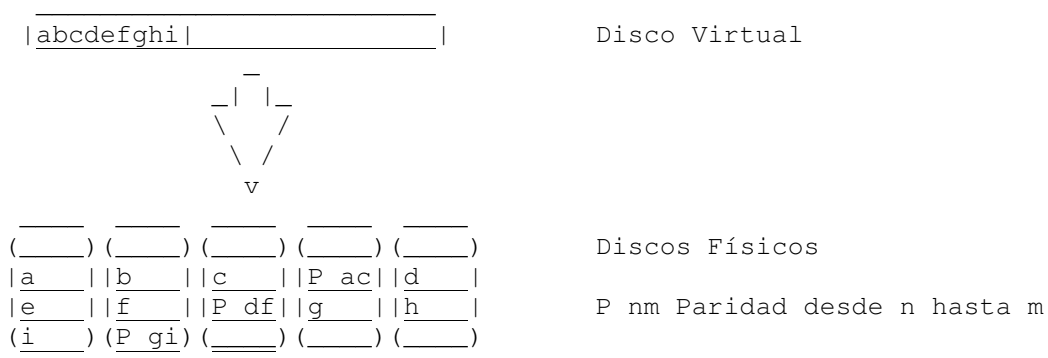
RAID 1 (Mirroring)

RAID 2 (Hamming Code)



RAID 3 y 4 (Hamming Code con Parity Disk)

RAID 5

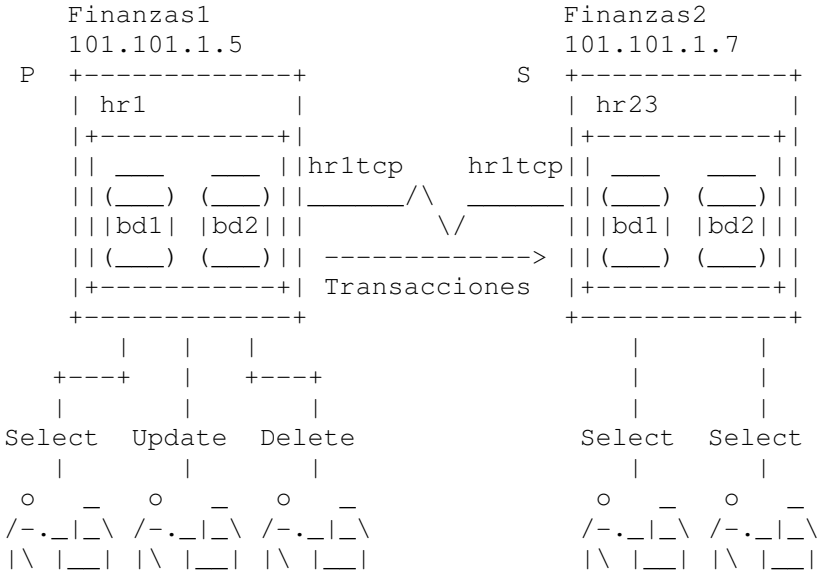


RAID 1+0 (Mirroring + Stripping)

REPLICACION DE DATOS (HDR).

Por Replicación de Datos entendemos al proceso de reproducir los datos presentes en una instancia principal, llamada primaria, en otra instancia denominada secundaria para fines de Alta Disponibilidad, de ahí el témino HDR (High-Availability Data Replication), este comentario aplica por que existe otro tipo de replicación de datos cuya función no es la de tener alta disponibilidad de datos sino la diseminación automática de información entre varias instancias, también llamada Enterprise Replication o por sus siglas ER, aunque hay que admitir que su fundamento interno es muy similar al que aplica en el HDR.

La idea es muy sencilla, tener dos instancias idénticas, con idéntica estructura de discos, es decir, mismo número de discos, mismo path de los chunks a nivel de S.O., e idéntico tamaño, precisamente por estas implicaciones la instancia primaria y la secundaria deben estar en cajas distintas.



La idea es que al primario se puedan conectar los usuarios para realizar consultas, actualizaciones o bajas, el secundario estará disponible si el equipo o la instancia primaria caen inesperadamente pero mientras esto no ocurra los recursos que estamos invirtiendo en el secundario no son desaprovechados ya que podemos acceder a esta instancia SOLO para hacer CONSULTAS.

Para realizar la replica nos valemos de la bitácora de transacciones, obviamente resulta obligatorio que todas las bases de datos en la instancia primaria hayan sido definidas con logging, cada vez que a una transaccion se le a dado COMITT, esta pasa un buffer y a un espacio temporal en disco donde se encola, existe un demonio que, através de una comunicación vía TCP-IP, envía las transacciones terminadas en el primario hacia el secundario, a su vez existe un demonio en el secundario que las recibe y las aplica conforme van llegando, de hecho la instancia secundaria se encuentra en una especie de Fast Recovery permanente a excepción de que permite consultar los datos.

Existen dos tipos de replicación de datos para Alta Disponibilidad:

- Replicación Síncrona.
- Replicación Asíncrona.

En la primera no basta con enviar la transacción y continuar la operación, en la replicación síncrona el secundario debe enviar una confirmación (aknowledge) de recibido y aplicado antes de que el primario envíe la siguiente transacción, resultado, mayor confiabilidad pero menor performance.

En la segunda el primario envía la tranascción y no espera reconocimiento (aknowledge) para enviar el siguiente paquete.

Existe una señal que se envía el primario al secundario, una especie de ping, si el secundario deja de recibir esta señal por un tiempo especifico, completamente configurable, entonces asume que el primario se ha caido y de acuerdo a la política que escojamos el secundario puede salir de su estado de Fast Recovery a un estado completamente operable, admitiendo actualizaciones, de dos formas:

- Automática
- Asistida.

En la primera se coloca como primario automáticamente, si bien las conexiones al primario original se pierden, todo lo que tendrán que hacer los usuarios es conectarse al que antes era el secundario y voila.

Esquemas de Alta Disponiblidad por HW (HA)

Una vez más pudiera darse el caso de que el esquema de replicación no fuera suficiente, en este sentido existen una amplia gama de plataformas que ofrecen esquemas de alta disponibilidad (High Availability) por ejemplo:

HP-UX	HP-HA
Sun	Solstice HA
IBM	HA-CMP
Microsoft NT	Wolfpack

Básicamente lo que hacen es configurar una IP-Address virtual entre dos equipos clusterizados o dominios, en cuanto alguno de estos se daña, la IP-Address apunta al equipo en buen estado, el contenido de memoria es almacenado a disco, y estos son, a su vez, “importados”, ¿resultado? quizá un pequeño pico en el tiempo de respuesta pero nada más, obviamente existen requerimientos muy variados en cuanto a Hardware pero casi todos comparten los siguientes:

- Las máquinas deben estar en un cluster propietario. SP/2 de IBM, Sunfire Ultra 10000 de Sun, HP-9000 series V, etc.
- Los discos, de tecnología propietaria, deben poder accesarse directamente por dos equipos A NIVEL HW, no confundir con NFS.

REDUNDANCIA COMPLETA DE SERVIDORES (TANDEM)

Y por si la paranoía llegara a niveles extremos, existen esquemas de HW replicado en forma integral, esta tecnología puesta tan en boga por Tandem también lleva su nombre, en esta arquitectura todo, y digo TODO, está duplicado, memoria, disco, CPU´s, de hecho todo requerimiento se procesa simultaneamente en los dos equipos, comparando los resultados en puntos claves, si bien su confiabilidad es la más alta del mercado generalmente lo es también su precio, actualmente los fabricantes más comunes de esta tecnología son Tandem y Sequent.

Conceptos vistos:

Fast Recovery
Respaldos
- Físicos
- De logical Logs
- Incrementales

Sistemas de Misión Crítica
Mirroring
RAID
- Nivel 0
- Nivel 1
- Nivel 2
- Nivel 3
- Nivel 4
- Nivel 5
- Nivel 0+1
- Nivel 6

Replicación de Datos (HDR)
- Síncrona
- Asíncrona

High Availability por HW.
Arquitectura tandem.

CAPITULO X – INFORMACIÓN CORPORATIVA Y TIPOS DE BASES DE DATOS.

Si bien los manejadores de bases de datos están diseñados para satidfacer los más variados escenarios y usos posibles es importante distinguir los ambientes de bases de datos usados con mayor frecuencia, ya que de una buena determinación de estos ambientes podremos realizar las configuraciones adecuadas que nos permitan tener el mejor rendimiento de nuestro RDBMS, los ambientes más comunes son:

- OLTP
- OLAP
- Web.

A continuación procederemos a hablar de cada uno de ellos:

Ambientes Transaccionales (OLTP).

Un ambiente transaccional u OLTP (OnLine Transaction Processing) es aquel dedicado a la consulta y actualizaciones concurrentes a alta escala, un ejemplo típico de este tiopo de ambientes son los sistemas de cajas registradores en tiendas de autoservicio, en estos ambientes se realiza una enorme cantidad de inserciones simultaneas (un insert por cada producto registrado) y se realizan gran número de actualizaciones simultaneas (al actualizar los inventarios, por ejemplo), estos sistemas se distinguen fundamentalmente por las siguientes caracterizticas:

- Gran número de actualizaciónes concurrentes.
- Queries de alta selectividad, es decir, cuando se consulta la B.D. solo un número muy pequeño de registros es devuelto.

A fin de garantizar la integridad transaccional en caso de falla es altamente recomendable, por no decir necesario, que la base de datos sea transaccional, en esta situación se recomienda el uso de niveles de aislamiento en COMMITED READ.

Ambiente de Toma de Decisiones (OLAP)

Un ambiente de toma de deciones u OLAP (OnLine Analitical Processing) es aquel dedicado a la alta gerencia y dirección a fin de poder tomar decisiones correctas, oportunas y rápidas a partir de sistemas de información gerencial. En estos ambientes se manejan grandes volumenes de información, pero no se realizan actualizaciones en línea, en estos sistemas de información se guarda únicamente la información necesaria para la toma de decisones y se actualiza periodicamente. Un ejemplo claro es el sistema que genera reportes cada mes para saber cuanto dinero y de que forma a perdido una compañía por los clientes que han se han cambiado a la competencia. Sus principales caracteristicas son:

- Pocas o nulas actualizaciones en línea.
- Queries que devuelven gran cantidad de información acompañada por consolidados (sumatorias) y ordenados por criterios.
- Capacidad de realizar queries no planeados por personas que no tienen necesidad conocer tecnicamente el RDBMS ni su arquitectura.

En estos ambientes se trabaja fundamentalmente con tablas estáticas, es decir, que no son actualizadas en línea, generalmente la información se carga periódicamente, en esquemas por lotes podríamos decir, y lo demás es consulta por parte de la alta dirección de la empresa. Dado este esquema el uso de logical logs es por demás innecesario, por lo que es común que en estos esquemas las bases de datos aparezcan sin logging. En toda base de datos sin logging no se pueden manejar niveles de aislamiento, siendo su situación equivalente al DIRTY READ.

Cuando se usa DIRTY READ no deben hacer actualizaciones y consultas al mismo tiempo, esto se debe a que al hacer actualizaciones y consultas al mismo tiempo con una base de datos en dirty read, informix puede devolver "registros fantasmas" es decir, uno o varios registros que estan en proceso de actualización o de inserción, este estado no se considera como un error de ejecución, por lo que se deberá hacer los SELECT sin que ningún UPDATE, INSERT o DELETE esté corriendo simultaneamente.

A esta categoría pertenecen los Sistemas de Información Gerencial (SIG), Dataware House (DWH) y DataMarts (DM).

Ambientes de Web.

Una de las grandes ventajas que tenemos en la actualidad es el Boom del uso de Internet a través del World Wide Web, WWW de aquí en adelante, aúnque esta tecnología puede ser vista como Cliente-Servidor, por su conexión y sus particularidades debe ser visto de una forma completamente aparte.

La aparición de internet como tal (gopher, ftp, archie, veronica, etc.) no fué suficiente para generar la masa crítica involucrada en la demanda de los servicios en línea, fue hasta la aparición del WWW (World Wide Web) que el boom de internet apareció, en esta generación todo se manjaba directamente a través de un protocolo llamado http (HyperText Transfer Protocol) y archivos planos o ASCII generados a partir de especificaciones conocidas como HTML (HyperText Marking Language). A este nivel no había una explotación real de bases de datos, solo intercambio de información a documento específicos contenidos como archivos en un disco.

La primera generación de programas capaces de explotar una base de datos desde el Web estaban basados en cgi () en esta tecnología basicamente lo que hacemos es pasar el control de la interfase html a un programa que corre en el servidor, lee los datos de una página anterior, hace la consulta y genera una página nueva para publicar los resultados.

La segunda generación está basada en API´s (Application Interfase) que se conectan diractamente al servidor del Web a través de un puerto, este API relaiza la interfase con la base de datos y es la que recupera la información, aún se

La siguiente generación esta basada en principios parecidos a aquella basada en API´s del web server con la diferencia de que la explotación de los datos se hace a través de Java o ActiveX y máquinas virtuales

embebidas que permiten un proceso realmente distribuido y arquitectura cliente servidor.

Las tecnologías de Web están orientadas fundamentalmente a dos fines:

- Distribución y acceso de documentos en intranets o internet.
- Comercio Electrónico y banca electrónica.

Se deben tener las mismas consideraciones que para OLTP más las consideraciones propias de una conexión TCP/IP, no dedicada, de velocidad posiblmente baja y de transacciones que no deben ser muy grandes y posiblemente numerosas.

Conceptos vistos:

OLTP
OLAP
WWW
  html
  cgi

CAPITULO XI – APLICACIONES DE BASE DE DATOS

Sistemas de información y operación (transaccionales)

Este tipo de sistemas, son sistemas hechos a la medida para solventar necesidades específicas de la operación cotidiana de la empresa hacia la cual va dirigida. Aunque puede haber casos específicos de productos de terceros, como por ejemplo, las soluciones de Terminales Punto de Venta (POS).

Normalente son del tipo OLTP, descritas en el capítulo anterior, y permiten llevar a cabo las operaciones del día a día.

CRM (administración de la relación con clientes).

Este tipo de sistemas, se basan en un modelo orientado hacia el cliente de nuestra compañía, o hacia el mercado en algunos casos. Son sistemas que ofrecen una visión de "que quiere" o "que necesita" nuestro cliente y nos permite fortalecer el vínculo que nuestro cliente tiene con la compañía que usa el sistema.

Estos sistemas normalmente se encuentran desarrollados por terceros, algunos de paga y otros Open Source, que pueden ser tropicalizados para satisfacer las necesidades de un pais o mercado particular, pero con una orientación estándar. Con módulos del tipo Ventas, Mercadeo, por mencionar los más comunes.

GIS (sistemas de información geográfica).

Este tipo de sistemas, han tenido mucho auge a partir de la necesidad de concentrar información geográfica con bases de datos tradicionales, lo que implica no sólo el almacenamiento, sino la capacidad de representar dentro de la base de datos información compleja que no es fácil de manipular con datos convencionales. Por ejemplo: Coordenadas geodésicas, Manejo de latitudes y longitudes, Mapas en formato gráfico, etc.., en industrias como telecomunicaciones, agencias espaciales y relacionadas con satélites, empresas petroleras, empresas orientadas a prospección, agropecuarias, climáticas, entre las más importantes.

Es por eso que este tipo de sistemas son especialmente atractivos para el uso de bases de datos orientados a objetos, donde se pueden manejar este tipo de datos especiales de forma nativa y de forma muy eficiente comparado con un sistema de cómputo que sólo almacenara los datos usando tipo de datos convencionales.

Si bien, este tipo de sistemas son generalmente desarrollos hechos en casa, en la mayoría de los casos existe la tecnología de base de datos para almacenar los datos específicos de base de datos geográficas de manera nativa, como sería el caso del Informix Geodetic Datablade de IBM para motores de base de datos Informix, así como las diferentes capacidades de la base de datos para generar tipos de datos nuevos ad-hoc para solventar estas necesidades.

MIS (sistemas de información gerencial).

Son aquellos sistemas de información que normalmente tienen como alacance a toda la compañía orientados a solucionar problemas a nivel de toda la empresa. En este rublo cabe destacar dos tipos de sistemas complementarios:

Ambiente de Toma de Decisiones:

También conocidos en otro tiempo como DSS (Decisión Support Systems), son sistemas de Información que permiten a la gerencia alta y media, tener la información necesaria para tomar decisiones estratégicas, estos sistemas son del tipo OLAP (explicados en el capítulo anterior) y típicamente realizan análisis de tendencias, así como correlaciones entre los datos. Si bien puede haber sistemas del tipo MIS hechos en casa, son cada vez más las compañías que usan productos y modelos de DataMart y DatawareHousing (DWH) para su toma de decisiones, analizaremos más a detalle el DWH en un capítulo posterior.

ERP (Planificación de recursos empresariales)

Un ERP permite concentrar en un sistema el manejo empresarial de los recursos de la misma (incluido el humano) y que permite tener un control centralizado del manejo de los mismos, es predominantemente OLTP, aunque puede tener fuertes componentes del tipo batch como es el caso de la nómina y algunos procesos de contabilidad. De forma similar a un CRM, estos sistemas normalmente se encuentran desarrollados por terceros y pueden ser tropicalizados para satisfacer las necesidades de un país o de una industria en partícular, entre los módulos más comunes de este tipo de ambientes se encuentran logística, producción, inventario, distribución, facturas y contabilidad; teniendo ingerencia sobre múltiples actividades de la empresa tales como ventas, pagos a proveedores, inventarios e inclusive recursos humanos.

Relación entre los diferentes tipos de Aplicaciones de la Base de Datos en la empresa

Ahora bien, sobre el tipo de sistemas descritos anteriormente, normalmete existe una relación entre ellos (a excepción de los sistemas de información geográfica que tienden a ser específicos de algunas industrias), complementándonse y auxiliandose entre ellos.

Por ejemplo los sistemas de información y operación, realizan las actividades más directas (también conocido como front-office), como por ejemplo el control de cajas de un supermercado, el levantamiento de pedido de una compra vía web, o el censado del consumo de un bien como puede ser el tiempo de teléfono o el consumo de energía eléctrica. Los datos generados, por una parte, alimentan a un sistema CRM para establecer un ciclo de relacionamiento con el cliente, así del ambiente de información y operación se concentra a nivel emprecial en un ERP que permite llevar una contabilidad centralizada óptima, pago a los proveedores que nos venden ya sea bienes de consumo final que nosotros revendemos, o insumos para producir nuestra propia mercancía, control de los inventarios a nivel corporativo, o incluso la nómina. Del ERP y el CRM podemos ir alimentando el sistema de toma de decisones para poder revisar tendencias y poder generar estrategias que nos permitan

ser más atractivos y éxitosos en el negocio, con acciones tales incrementar la eficiencia de la operación, establecer campañas de publicidad más asertivas, así como entender mejor el comportamiento del mercado para tomar acciones previas que maximicen la probabilidad de liderazgo, o que minimicen los riesgos potenciales.

Conceptos vistos:

Sistemas de información y operación (transaccionales).
CRM
GIS
MIS
ERP

CAPITULO XII – PERFORMANCE AND TUNNING

Concepto de Performance.

Entendemos por Performance de Manejo de bases de datos a la relación directa que existe entre los recursos que posee y el aprovecahmiento de éstos. No existen unidades típicas de performance, pero generalmente están asociados bien a tiempos de respuesta, bien a porcentaje en el aprovechamiento de los recursos, o a una combinación de ambos.

Quizá el concepto en español mas parecido a Performance sea rendimiento, sin embargo el término en inglés denota más que buena velocidad y aprovechamiento de los recursos, cuando usamos el verbo to perform, nos referimos a que el sistema, dispositivo o actividad realizandose está realizándose de forma adecuada.

Este rendimiento es producto del análisis a conciencia de las expectativas de nuestros usarios respecto al equipo, al RDBMS y, por supuesto, a los sistemas que lo utilizan, sin embargo nos enfocaremos en este capítulo al performance del RDBMS.

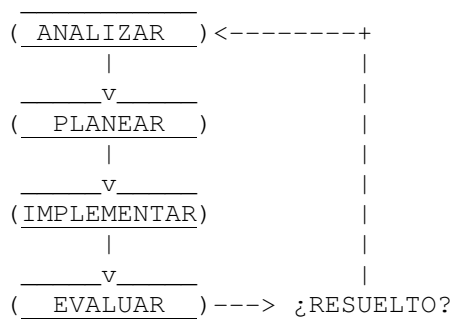
Una véz que nosotros hemos establecido las expectativas, y relizar el análisis de los diferentes puntos sujetos de mejora, se deben ajustar diferentes parámetros a nivel del RDBMS y del S.O., a esta labor de ajustar parámetros le denominamos afinación o tunning.

Metodología.

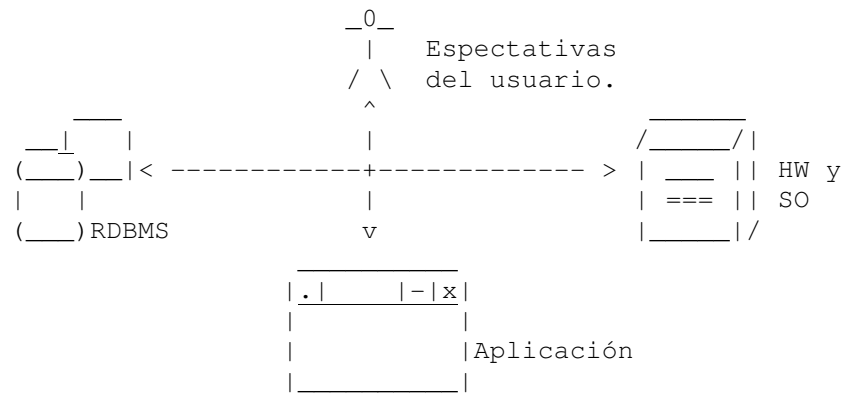
Los que conocimos los motores de automóvil con carburador, sabemos que no basta con saber como funciona el automóvil y como ajustar el tiempo del carburador y calibrar bujías para tenerlo afinado, se necesita de la experiencia y del continuo ejercicio para poder tener el resutado esperado.

En el caso de los RDBMS sucede algo parecido, se requiere experiencia y práctica para poder afinar adecuadamente un RDBMS, pero es indispensable seguir un método a fin de realizar las actividades adecuadamente y saber realmente si las acciones tomadas fueron adecuadas o no.

La metodología más sencilla se representa en el siguiente diagrama de flujo:



Aspectos a tomar en cuenta en el establecimiento de los objetivos durante el proceso de Performance and Tunning:



Espectativas de usuario: Si las espectativas de usuario no son suceptibles de ser alcanzadas bajo la situación óptima del equipo, no hay nada que el DBA pueda hacer. Es importante conscientizar al usuario a fin de que sus espectativas sean lo más cercanas a la realidad posible.

Hardware y Sistema Operativo: Esta categoría abarca disco, procesador, redes y kernel del sistema operativo. Es importantísimo el examinar el performance del equipo físico en el que corre nuestro sistema.

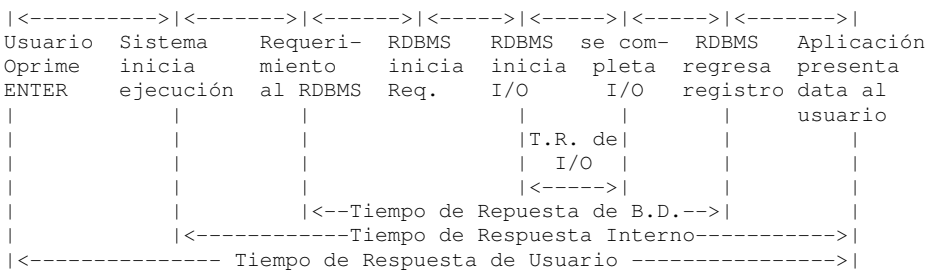
Aplicación: Esta área es comunmente ignorada, sobre todo cuado el área de Administración de Bases de Datos es diferente a la de desarrollo, es un grave error ignorar esta parte, es parte de las responsabilidades del DBA el asegurarse que la gente que escribe la aplicación lo haga aprovechando los recuros.

RDBMS: Finalmente, el RDBMS debe ser monitoreado a fin de asegurarnos que el performance sea óptimo.

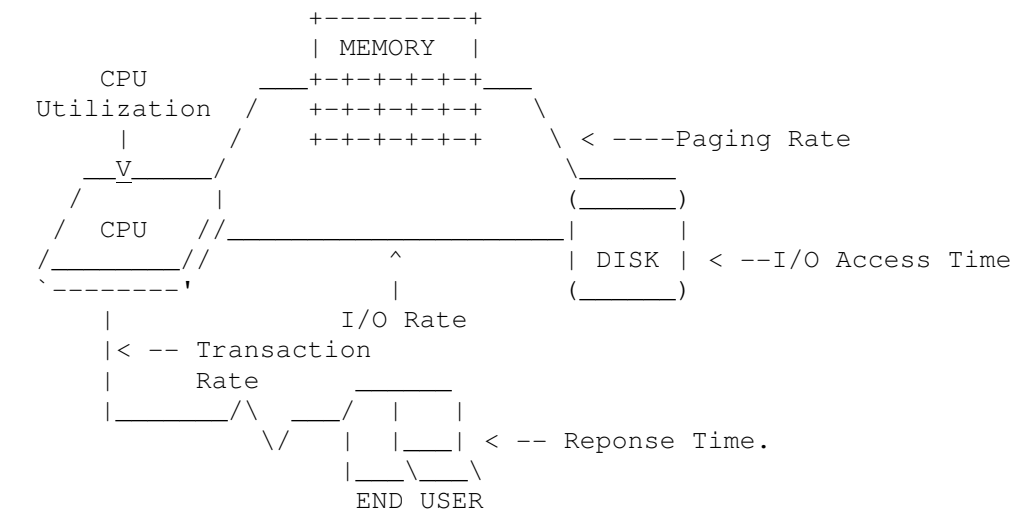
Componentes del performance:

Tiempo de Respuesta=
- Tiempo de Comunicación +
- Tiempo de Procesamiento +
- Tiempo de Disco +
- Tiempo de Paginación.

Tiempo de respuesta:



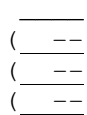
Medición del Performance:



- Response Time: Segundos por unidad de trabajo.
- Transaction rate: Transacciones completadas por segundo.
- CPU utilization: segundo de proceso efectivo de CPU sobre segundos reales por cien.
- I/O Rate: Número de Accesos a I/O por segundo.
- I/O access time: Tiempo de uso al I/O en segundos.
- Paging rate: Número de páginas subidas a memoria por segundo.

Componente de Disco:

Access Time = Seek Time + Latency + Transfer Time



Seek Time: Tiempo que le toma a la cabeza trasladarse a un cilindro específico.

Latency: Tiempo que le toma al disco el rotar sobre un cilindro específico.

Transfer Time: Que tan rápido puede transferir la data desde y hacia el disco.

El disco es el factor más crítico en cuanto a performance se referie, al ser el único dispositivo mecánico del RDBMS uno se puede imaginar como puede afectarse el performance en este componente.

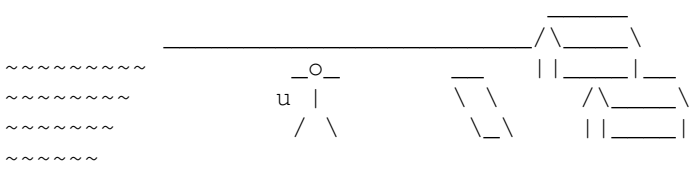
Algunas recomendaciones al respecto son:

- Procure que las tablas más accesadas queden hacia los cilindros centrales del disco.
- Procure que el tamaño de los extents sea los suficientemente grande para evitar que la cabeza tenga que trasladarse a otro sector y cilindro a menos de que sea necesario.
- Balancee las cargas en los discos disponible fragmentando cuando esta sea posible.
- Si la fragmentación no es factible, use un esquema de RAID 0 para balncear las cargas.
- Evite en medida de los posible el uso de RAID 5 (Performance Killer).
- Dimensione adecuadamente sus buffers.

Los buffers y el disco:

Imagine que tiene usted un lago repleto con agua y un poblado que require agua, usted tiene que trasladar el agua desde el lago hasta el pueblo con una cubeta :-) depositándola en una pileta común. En este ejemplo suponemos que en el traslado lo que realmente nos pesa es la distancia, no la cantidad, y dado que una vez que la ciudad requiera más agua tendremos que hacer más viajes pensemos:

¿Es mejor hacer más viajes con una cubeta chica?
¿Es mejor hacer menos viajes con una cubeta grande?



Dado que el peso del agua no es tanto problema, preferiremos hacer menos viajes con una cubeta más grande, en este ejemplo el lago son los datos en disco, el consumo de agua del pueblo es la demanda de información por parte de los usuarios y nuestra cubeta son los buffers, dado que en un RDBMS el acceso a disco es el más penalizado, es por eso que cada viaje cuesta más que el agua que se pueda cargar.

Es decir, de un buffer adecuadamente grande, dependerá en gran medida el correcto acceso a disco. Sin embargo la analogía no queda ahí, no necesariamente más es mejor, es decir, llega un punto en el que el tamaño de la cubeta satisface el abastecimiento y no por más grande que sea la cubeta vamos a tener menos viajes, por lo que un buffer exageradamente grande puede ser un desperdicio de recursos.

Componente de Memoria.

No hay gran cosa que se pueda afinar al respecto salvo el tomar las siguientes consideraciones:

- Asigne suficientes buffers, como se mencionó en el apartado anterior a fin de que el grado de paginación (número de lecturas de disco a memoria) sea el emnor posible.
- Algunos sistemas operativos permiten forzar la permanencia de los segmentos en memoria física, evitando el tener que sea páginada y bajada a espacios temporales del S.O., cuando sea posible force la residencia en memoria física.
- Si el número de segmentos crece a más de cuatro (particularmente por el segmento virtual), se corre mayor riesgo de que sea generado en memoria virtual, es preferible tener un segmento de componente virtual grande que varios pequeños.

Componente de CPU.

Dado que la arquitectura de un RDBMS se basa en el procesos que simulan ser CPU's, llamados Procesadores Virtuales, la medición y optimización del performance en esta componente debe estar orientada al monitoreo de uso de los procesadores a nivel de S.O. (con el comando sar). De los VP's los más críticos y los que más consumen recursos son los CPU VP's por lo que recomendamos su monitoreo siguiendo las siguiendo las siguientes recomendaciones:

- Habilitar el No Aging para los CPU VP's cuando el S.O. lo permita.
- Habilitar y forzar el processor affinity (asociar un proceso a un procesador físico) para los CPU VP's cuando el S.O. lo permita.
- No habilitar más de n-1 CPU VP's donde n es el número de CPU's físicos.
- Si se tinen 2 CPU's físicos, configurar el RDBMS como si estuviera corriendo en máquinas con un solo procesador.

Componente de RED.

En realidad esta es la capa donde menos control se tiene, se pueden detectar cuellos de botella usando el comando de S.O. netstat (recordar que la primera línea siempre se ignora), si observamos que las colisones son de dos dígitos (>9) se puede incorporar una tarjeta nueva, y un alias nuevo para esta tarjeta a fin de balancear la carga de red.

Por otra parte el uso de Stored Procedures en un esquema Cliente-Servidor, puede ayudar a desahogar la cantidad de datos que se transmiten por la red.

Estadísticas.

La mayoría de los RDBMS actuales tienen varias formas de resolver un mismo query, para identificar cual es la mejor manera de resolver dichos queries, los RDBMS poseen un componente llamado optimizador.

Dicho optimizador es el que toma la decisión de como resolver un query específico, obviamente, como en el mundo real, las estadísticas sobre los datos, índices y esquemas de la base de datos ayudan a nuestro optimizador a conseguir mejores tiempos de respuestas.

Para ello existen instrucciones para actualizar dichas estadísticas como por ejemplo UPDATE STATISTICS.

Se recomienda actualizar las estadísticas en las siguientes condiciones:

- Después de cargas masivas de información.
- Despues de actualizaciones másivas de información.
- Periódicamente (por ejemplo una vez por semana).

Las estadísticas son la principal herramienta para optimizar procesos y queries problemáticos y deberán ser corridos ANTES de iniciar cualquier análisis de performance.

Conceptos vistos:

Performance
Tiempo de Respuesta
Tiempo de Comunicación
Tiempo de Procesamiento
Tiempo de Disco
Tiempo de Paginación
Tiempo de Respuesta de Usuario
Tiempo de Repuesta Interno
Tiempo de Respuesta de B.D.
Tiempo de Respuesta de I/O.
Uso CPU
Paging rate
Transaction Rate
I/O Rate
Access Time
Seek Time
Latency
Transfer time
Buffers de Memory
Estadísiticas
Benchmark

CAPITULO XIII

DATAWAREHOUSING.

Elementos de un DWH.

Un datawarehouse es, en el sentido más amplio del término, un sistema de toma de decisiones (DSS) con algunas características especiales tales como:

- Las bases de datos son creadas específicamente para la toma de decisiones, a diferencia de los OLTP. Las bases de datos son configuradas en sus propios ambientes los cuales son optimizados para obtener el mejor performance.
- La Data se extrae de uno o más sistemas fuentes y se integra en una fuente única. Una vez realizada esta actividad los datos son sumarizados (Sumatorias, promedios, ordenamientos) y explotados.
- Los datos son analizados para detectar patrones y tendencias.

En un DWH se busca, a su vez, concentra5r un repositorio de información al nivel de la compañía, a fin de que los resultados que afectan a toda la compañía sean precisos y la alta dirección tenga fuentes de información consistentes y adecuadas para la toma de decisiones.

Una definición concreta de DWH podría ser, según Bill Inmon: “Es el conjunto de bases de datos diseñadas en forma integral, orientadas al sujeto para soportar las funciones de toma de decisiones, donde cada unidad es relevante en algún momento en el tiempo”, o en palabras de Ralph Kimball: “Es una copia de datos transaccionales estructurados específicamente para su consulta y análisis”.

Complemento lo anterior, y hablando de un modo más operativo, debemos contemplar el hecho de que nuestros gerentes y directores deben tener interfaces de consulta fáciles de manejar, que no requieran un conocimiento técnico elevado tales como: manejo de SQL o la arquitectura de base de datos, y que contemplen consultas no planeadas.

Para poder conseguir los puntos anteriores es necesario contemplar algunos componentes necesarios para lograr este objetivo:

- Tabla de hechos: Es la información consolidada sujeto de las consultas para la toma de deciones.
- Dimensiones: Son básicamente los cátalogos de la tablas de hechos, cabe mencionar que la normalización no es prioritaria en este esquema y, en ocaciones, es inclusive sacrificable en aras del performance, siendo necesario solamente el tener a la base de datos en primera forma normal.

Datawarehousing y Datamart.

Quizá una de las principales confusiones sea la de distinguir un datawarehose de un datamart, un datamart es, fundamentalmente, un subconjunto de un dataware house, mismo que está orientado a un departamento o unidad específica de negocios.

En un datamart la data tiene un alto grado de sumarización a nivel mensual, bimestral, trimestral, semestral o anual. No se actualiza con frecuencia pero la información se reemplaza a intervalos regulares a fin de mantener la precisión.

Estos datamart se usan normalmente para hacer decisiones a nivel d departamento o para analizar algún aspecto espécifico de la unidad de negocios más que una estrategia a nivel empresarial.

Dataminning.

El proceso de búsqueda de información dentro de un dataware house es su razón de ser, su principal inquietud y su principal poderío, a dicho proceso se le conoce como exploración de datos o Dataminning.

Este proceso hace uso no solamente de las tablas de hechos y de dimensiones, sino inclusive de información acerca de la información, de ahí el témino Metadata, esta metadata sirve tanto para encontrar la mejor manera de explotar la información, como para generar consultas no planeadas.

Conceptos vistos:

Dataware House
Datmart
Metadata
Data Minning
Tabla de hechos
Dimensiones

CAPITULO XIII B – BASES DE DATOS ORIENTADAS A OBJETOS.

Paradigma Relacional vs Objetos.

A diferencia de un RDBMS donde nosotros tenemos un conjunto de elementos específicos, una sintaxís específica no dinámica y una forma procedural y algorítmica de ver los problemas y sus soluciones, nos encontramos con otra forma de ver los problemas, relacionadas con entidades complejas, con una solución basada no solo en el problema externo sino también con la implementación de esquemas internos dentro de nuestro esquema de herramientas para la resolución de los problemas.

Es en este sentido que nacen los objetos, entendiéndose por objeto a aquella entidad poseedora de ciertos elementos ennumerables en forma de sustantivos llamados atributos, ciertos algoritmos o rutinas que le permiten manipularse a ellos mismos y a su entorno llamados métodos, y un arbol genealógico que determina si fueron generados como producto de la "evolución" de un objeto de mayor orden, denominado clase.

Con estos elementos aplicados a un manejador de base de datos podemos extender exponencialmente sus capacidades al darle el poder para realizar dinámicamente, más funciones que aquellas que tenía cuando fué creado, la capacidad de manejar una cantidad mucho más amplia de datos en cuanto a tipos se refiere, el manejo óptimo de código externo al permitr la definición parcial de la solución de los problemas de programación en la estructura de los "datos" involucrados y la economía de código interno al poder generar objetos más complicados a partir de objetos más simples sin tener que definir todo el código de la nueva entidad.

A un manejador de bases de datos relacional capaz de manejar Objetos se le conoce como Sistema Manejador de Base de Datos Relacional Orientado a Objetos u OORDBMS por sus siglas en inglés.

Herencia, Encapsulamiento y Polimorfismo.

Para que un objeto sea considerado como tal, además de tener en sus componentes a los atributos, métodos y clase, debe cumplir con ciertas características que son herencia, encapsulamiento y polimorfismo, entendiéndose como herencia a la capacidad de un objeto de poseer todos los atributos y métodos del objeto a partir del cual fue creado, así pues, el objeto nuevo llamado "imaginario", generado a partir del objeto "real", poseerá todos los elementos del objeto "real" más los propios (otro "real"), así como todos los métodos de real("+,-,*,/,sqr, sqrt") más los propios ("suma_vec, resta_vec, producto_cruz, producto_punto"), así mismo un objeto debe ser visto como una caja negra, sabiendo que párametros acepta, que métodos tiene, que resultados arroja a partir del tipo de operación realizada, todo ello viéndolo como el objeto tal cual es a este enfoque se refiere el encapsulamineto y por último el polimorfismo se refiere a la cualidad de un objeto de actuar en formas distintas dependiendo del ambiente en el que esté actuando, así pues si multiplicamos un valor "imaginario" por otro "imaginario", el objeto debe ser lo suficientemente

inteligente para distinguir que el resultado debe ser otro “imaginario” y utilizar el método “producto_cruz”, si por el contrario, el “imaginario” se está multiplicando por un “real”, el resultado podría ser otro número real, por lo que se usará en su lugar el método “producto_punto”, el encanto de esta propiedad es que nosotros solo le decimos al OORDBMS que queremos un producto, es el objeto, y no otra entidad, quien decide que método utilizar dependiendo del entorno.

Tipos de Datos vs Primitivas.

En una base de datos relacional convencional solamente tenemos acceso a ciertos tipos de datos, específicos y relativamente rígidos, tipos de datos que dificultan el manejo de entidades no convencionales tales como: números imaginarios, cualquier tipo de vectores, coordenadas geoespaciales o algunos tipos más específicos como puede ser: una página Web, un documento de Word, un archivo de video o formatos que no existían en el momento de crear la versión del manejador de base de datos que estemos usando.

A partir del concepto de objetos podemos crear nuevos tipos de datos añadiéndoles funcionalidad y complejidad permitiendo que la base de datos se encargue automáticamente de la manipulación y del proceso, liberando al desarrollo de engorrosos procedimientos, implementación de procedimientos y código de validación dejándole esta labor al manejador de base de datos.

En este contexto, los tipos de datos presentes en un RDBMS deja de ser tipos rígidos para convertirse en un objeto en sí mismo, del cual pueden salir varios tipos de datos difrenetes más, y de ahí que a este tipo de datos en partícular le llamemos “primitivas”.

Conceptos vistos:

Objeto
OORDBMS
Herencia
Encapsulamiento
Polimorfísmo
Primitiva

BASES DE DATOS Y WWW.

Esquemas de conectividad en el Web.

Una de las principales aplicaciones que han dado revuelo al uso de los manejadores de bases de datos es el uso de la World Wide Web o, en su forma familiar corta, Web a secas.

La forma en la que interactúa un RDBMS con el manejador de Web puede ser de los siguientes tipos:

Por CGI.

En este esquema, se maneja un programa ejecutable en C que llama al manejador de base de datos a través de librerías que permiten el manejo de instrucciones SQL que se conectan directamente al RDBMS y permiten la manipulación de los datos desde el progma en C, a este tipo de lenguajes embebidos le denominamos : ESQL/C.

Por API.

Otra forma de realizar este tipo de consultas es a través de interfases de usuario diseñadas por el manejador de Web Server, que incluyen módulos opcionales para comunicarse directamente a un manejador de base de datos a través de Java o ActiveX a través de ODBC drivers específicos, que en el caso de Java reciben el nombre de JDBC's.

Por Demonios propietarios.

Otro esquema consiste en expandir el lenguaje HTML para permitir el incorporar instrucciones SQL y algunas estructuras de control como extenciones al HTML estándard, teniendo programas que sirven de interfase permitiendo al manejador de base de datos encargarse tanto de la extracción de información como de la presetación de ésta dentro de un Web Server comercial, dada la naturaleza de los datos regresados y la funcionalidad extentidad requerida para el manejo de este tipo de datos, el manejador de base de datos requerido debe ser un OORBDMS.

e-commerce.

El caso más impresionante y en apogeo del uso de OORDBMS y Web es el uso de comercio electrónico o e-commerce, este ambiente debe ser considerado como un ambiente OLTP con implicaciones de objetos y el webserver.

Conceptos vistos:

e-comerce
JDBC
Webserver

APENDICE – PERFIL Y RECOMENDACIONES PARA UN DBA.

Hoy en día el papel de las bases de datos como repositorio de información electrónica es más importante que nunca, ganando terreno rápidamente sobre los manejadores de archivos tradicionales, incluyendo los C-ISAM y, de hecho, desplazando en muchas plataformas al confiable y viejo COBOL.

Sin embargo esto tiene un precio: se requiere de una función administrativa capaz no solo de realizar las actividades operativas y técnicas que involucran a la base de datos en sí, sino también de gestionar administrativamente los requerimientos necesarios para su buen desempeño administrativo, es por eso que el perfil de un DBA se vuelve más crítico y complejo cada día, ya que debe entender el diseño de los datos, colaborar en el diseño de los mismos, responsabilizarse de la creación de los objetos de las bases de datos, mantener y afinar su operación, vigilar la seguridad y al mismo tiempo gestionar administrativamente que los privilegios, recursos y modelos de la base de datos se encuentren correctamente documentados, así como establecer comunicación con otras áreas para auxiliar, y auxiliarse, en puntos frontera, por ejemplo, con el Sistema Operativo o con la plataforma de desarrollo.

Perfil técnico del DBA.

Una de las principales razones para usar Sistemas de Administración de Bases de Datos (DBMS´s) es el tener el control centralizado tanto de los datos como de los programas que acceden a dichos datos. La persona que tiene dicho control centralizado sobre el sistema es llamado Administrador de la Base de Datos o DBA, y entre sus funciones se incluyen:

- Definición de esquemas: El DBA crea el esquema original de la base de datos a través del conjunto de instrucciones de definición de datos (DDL) del SQL.
- Definición de la estructura y método de acceso al almacenamiento de los datos.
- Definición de esquemas físicos y su modificación: El DBA tiene la responsabilidad de cambiar el esquema de la base de datos, así como su organización física, ya sea por requerimientos nuevos o bien por cuestiones de performance, esto incluye tablas, índices, vistas, alias, stored procedures y triggers entre los más importantes.
- Otorgar autorizaciones para el acceso a los datos: Al otorgar diferentes tipos de autorización, el DBA puede regular cuales partes de la base de datos pueden ser accesadas por usuarios específicos. La información sobre la autorización se mantiene en estructuras especiales del sistema que el motor de base de datos consulta cuando cualquier persona intenta acceder datos en la base de datos.
- Establecer las políticas y procedimientos para obtener la alta disponibilidad de los sistemas críticos si esto es necesario así como el establecimiento de políticas y procedimientos para la recuperación de los sistemas en caso de caída anormal del equipo o desastre.

- Mantenimiento: Las principales actividades asociadas con las labores de mantenimiento de rutina de un DBA típico se enlistan a continuación, aunque en varias compañías, algunas de estas labores son delegadas a al equipo técnico de producción y/o de soporte técnico y se consulta al DBA solo en caso de duda:
  - Realización de respaldos de las bases de datos, ya sea en cintas o en servidores remotos, para prevenir la pérdida de datos en caso de desastre o corrupción de datos.
  - Realización de los respaldos de las bitácoras de transacciones en ambientes con transaccionalidad activa.
  - Revisión del comportamiento de las bitácoras de transacciones para evitar el aborto de transacciones o congelación de la instancia de base de datos por transacciones muy grandes.
  - Monitoreo del uso de recursos, tales como buffers, locks, archivos de bitácoras de transacciones y procesos configurables de la base de datos.
  - Instalación y reinstalación de los binarios de las bases de datos así como sus respectivas actualizaciones o parches.
  - Depuración de bitácoras de mensajes de error.
  - Realización de cargas y descargas de tablas.
  - Revisión periódica del buen estado de los datos detectando tempranamente corrupciones en índices o en datos productos de fallas intermitentes en disco o por terminación anormal de los procesos.
  - Asegurarse que exista suficiente espacio libre en disco para las operaciones normales, así como la de extender las capacidades de espacio según se requiera.
  - Monitorear los procesos en la base de datos y asegurarse de que su performance no sea degradado por otros procesos demasiado pesados lanzados por otros usuarios.
  - La ejecución oportuna de la actualización de estadísticas del motor de base de datos y rebind de los procesos externos en los motores de bases de datos que posean cualquiera de estas características.
  - Realización de afinaciones periódicas a la base de datos para asegurarse de que el performance no se vea degradado por ser su comportamiento demasiado distinto de aquel que predominaba en la última afinación, esto incluye la afinación de parámetros del motor de base de datos.
  - La depuración periódica de tablas que cambian con respecto al tiempo y su respectivo movimiento a tablas históricas si así procede.
  - Apoyo y/o realización de migraciones de la bases de datos entre versiones del motor de base de datos o plataformas.
  - Levantamiento y seguimiento de casos de soporte técnico por problemas en la base de datos.

Perfil administrativo del DBA.

El administrador de la base de datos es mucho más que un técnico, su labor no termina con la operación técnica del sistema de base de datos relacionales. En la actualidad, el área de base de datos no puede

existir como una entidad aislada, pues el motor de base de datos es el proveedor de datos de las aplicaciones críticas del negocio y es, al mismo tiempo, un cliente del Sistema Operativo, así mismo el DBA debe gestionar no solo los recursos físicos del sistema, sino que también debe gestionar, en forma conjunta con otras áreas, uno de los recursos más importantes de las corporaciones actuales: el tiempo. Esto es particularmente patente en las ventanas de tiempo necesarias para efectuar validaciones a las bases de datos, el tiempo en el que el motor está fuera de línea recuperándose de una falla o un desastre, e incluso el costo en tiempo o en performance que implica la realización de un respaldo.

El DBA debe ser capaz de cooperar y de negociar con las áreas de sistemas operativos, de soporte técnico y de desarrollo y debe tener cierto grado de acción autónoma que le permita establecer acciones correctivas necesarias para el correcto funcionamiento del sistema, estas son algunas de las descripciones del perfil administrativo de un DBA:

- Advertir de las necesidades futuras de recursos cuando estos se estén agotando, monitoreando el consumo de dichos recursos, entre ellos el espacio en disco, el consumo de memoria y de CPU.
- Ajustar el modelo físico y esquemas relacionados con el negocio real y su modificación: El DBA normalmente tiene la responsabilidad de cambiar el esquema de la base de datos, así como su organización física, para reflejar los cambios en las necesidades del negocio, esto en forma conjunta con el área de desarrollo.
- Debe tener habilidades de negociación y comunicación con las áreas de sistema operativo, soporte y desarrollo a fin de realizar cambios proactivos benéficos a la operación, obtención de respuestas rápidas para problemas en producción y la pronta comprensión y solución en problemas que abarquen áreas fronterizas entre ellas, por ejemplo, el usos de CPU o la ejecución de un query mal programado.
- Capacidad de autonomía y de toma de decisiones en cuestiones operativas necesarias para la correcta operación del negocio, reforzada con las políticas y los procedimientos necesarios cuando estos sean posibles.

Recomendaciones.

En base a nuestra experiencia con otros clientes podemos establecer las siguientes recomendaciones generales a los DBAs:

- Documente los procedimientos en caso de emergencia en una carpeta con los procedimientos administrativos y técnicos necesarios para responder a contingencias con la base de datos, tales como mecanismos de recuperación y habilitación de esquemas alternos. Así mismo incluya todos los datos necesarios para levantar casos de soporte técnico en caso necesario.
- Establezca procedimientos de emergencia claros y confiables relacionados con los password tanto del DBA de la base de datos como del Administrador del sistema, una política muy usada es la de colocar el password de root en un sobre, el del DBA en otro sobre, sellarlos y firmarlos en el sello y dárselos a consigna al director o subdirector del área para que sean abiertos si se requieren en

caso de emergencia y no se ha podido localizar al DBA o al administrador del sistema (Este es tan solo un ejemplo, el manejo de los passwords es responsabilidad del área de seguridad de cada cliente en particular).

- Documente la estructura física y lógica de la base de datos, incluya en una carpeta los diagramas Entidad-Relación de sus bases de datos, incluya una hoja con un mapa de discos, señalando los discos asignados a sus sistema pero no empleados a fin de saber rápidamente donde obtener más espacio en caso necesario, e incluya copia impresa de los archivos de configuración de la instancia y las bases de datos.
- Documente e implemente nomenclaturas de nombrado estándar para los diversos objetos de sus base de datos, tales como bases de datos, tablas, vistas, alias, índices, constraints, stored procedures y triggers.
- Ponga especial énfasis en el respaldo de sus bases de datos, ya que en caso de desastre o corrupción severa, este será su principal, y en ocasiones único, medio de recuperación.
- En bases de datos OLTP el respaldo de las bitácoras de transacciones debiera ser obligatorio, aunque el motor de base de datos permita el ignorar el respaldo de las bitácoras de transacciones.
- Para cargas iniciales de información en tablas vacías use las opciones para deshabilitar transaccionalidad sobre la tabla antes de la carga, esto le evitará consumo innecesario de archivos de bitácoras de transacciones, asegurándose de habilitarlo una vez que la carga haya terminado si está trabajando en un ambiente OLTP..
- Así mismo, al momento de carga de mucha información en tablas muy grandes es recomendable el bloquear toda la tabla en modo exclusivo, esto evitará un consumo excesivo de locks en una actividad que generalmente es considerada como de mantenimiento.
- El número típico promedio de instancias que un DBA puede atender por turno es de 3, este número puede ser drásticamente afectado por la complejidad, tamaño y criticidad de los sistemas involucrados.
- Establezca esquemas de reemplazo de DBAs, es decir, un DBA alterno para ciertos sistemas, esto permitirá manejar mejor las faltas tanto programadas (vacaciones) como no programadas (convalecencia, emergencias personales, despidos o renuncias) de su personal.
- Si se tiene una aplicación de misión crítica y que opera las 24 horas del día los 365 días del año, considere el contratar DBAs en varios turnos e incluya calendarios escalonados para los fines de semana y días festivos. Se recomienda esta opción sobre la alternativa de los esquemas de guardias, que son muy empleados hoy en día, pues el esquema de guardias en casos emergentes plantea un estrés excesivo para el personal que se ve cobrado al día laboral siguiente o en la moral del empleado.
- Así mismo, para aplicaciones de misión crítica que deben operar las 24 horas del día, los 7 días de la semana, es recomendable el contratar los planes de soporte técnico 24 x 7 cuando estos existan para su producto.

Conceptos vistos:

Perfil técnico de un DBA.
Perfil administrativo de un DBA.
Recomendaciones