**ARCHIVOS CONVENCIONALES**

Registro: están compuestos por una serie de caracteres o bytes. Es una colección o conjunto de campos (atributos), donde cada uno de ellos contiene solamente un único valor almacenado.

Archivos: es un elemento de información conformado por un conjunto de registros. Los archivos, alojados en memoria secundaria, pueden almacenarse de dos formas diferentes: archivos convencionales o bases de datos. A su vez, los archivos convencionales pueden organizarse como archivos secuenciales o archivos directos y pueden estar estructurados en varios niveles y subarchivos.

Puntero: es un objeto del lenguaje de programación que contiene la dirección de memoria de un dato o de otra variable que contiene al dato. La obtención del valor almacenado en esa ubicación se la conoce como desreferenciación del puntero. Los punteros se pueden utilizar para manipular estructuras de datos, para referenciar bloques de memoria asignados dinámicamente y para proveer el paso de argumentos por referencias en las llamadas a funciones.

**ORGANIZACIÓN SECUENCIAL**

Es la forma básica de organizar un conjunto de registros, que forman un archivo, utilizando una organización secuencial. Lo registros quedan grabados consecutivamente cuando el archivo se utiliza como entrada. En la mayoría de los casos, los registros de un archivo secuencial quedan ordenados de acuerdo con el valor de algún campo de cada registro. El o los campos, cuyo valor se utiliza para determinar el ordenamiento es conocido como la llave del ordenamiento. Un archivo puede ordenarse ascendente o descendentemente con base en su llave de ordenamiento. Una característica de los archivos secuenciales es que todos los registros se almacenan por posición: primer registro, segundo registro etc.

En estos archivos, la información sólo puede leerse empezando desde el principio del archivo. El sistema siempre comienza al principio del archivo y lee un registro a la vez hasta llegar al registro deseado. La escritura de nuevos datos siempre se hace al final del archivo.

Ventajas:

* Proveen la mejor utilización de espacio y son rápidos cuando los registros son accedidos secuencialmente.
* Capacidad de acceso al siguiente registro rápidamente: mientras que el patrón de acceso a un archivo secuencial se conforme al ordenamiento de registros en el archivo, los tiempos de acceso serán muy buenos. En caso contrario, la eficiencia del programa puede ser terrible.
* Son muy sencillos de usar y aplicar.
* Los archivos con poca volatilidad, gran actividad y tamaño variable son altamente susceptibles de ser organizados secuencialmente.

Desventajas:

* La localización de un determinado registro no se puede hacer individualmente ni de manera rápida.
* La dirección de registro está implícita y son vulnerables a fallas del sistema.

**PILAS**

La forma menos complicada de organización de archivos puede denominarse pila. Los datos se recolectan en el orden en que llegan. Cada registro consiste en una ráfaga de datos. El propósito de la pila es simplemente acumular la masa de datos y guardarlo.

Como no hay estructura para el archivo de la pila el acceso a registro es por búsqueda exhaustiva. Si se quiere todos los registros que contienen un campo particular o que tienen un valor determinado para ese campo, debe buscarse en el archivo entero.

Los archivos de pilas se aplican cuando los datos se recogen y almacenan antes de procesarlos o cuando no son fáciles de organizar. Este tipo de archivo usa bien el espacio cuando los datos almacenados varían en tamaño y en estructuras. Además no se adapta a la mayoría de las aplicaciones.

Ventajas:

* Fácil de programar.
* Es la estructura que permite la inserción más rápida, ya que se limita a insertar al final.

Desventajas:

* Organización muy primitiva.
* El acceso a los datos es totalmente ineficiente ya que están desordenados.
* Requiere de mucho mantenimiento por tenerse que estar reorganizando cada poco tiempo.
* Puede ser que provoque redundancia y repetición en la información.

**ARCHIVOS DIRECTOS**

En contraste con los archivos secuenciales, permiten acceder directamente a un registro de información sin tener que buscar uno a uno por todos los registros del archivo, utilizando una llave de acceso dentro del archivo.

**ORGANIZACIÓN RELATIVA**

Consiste en un conjunto de celdas numeradas desde 0 o 1 hasta 'n', las cuales están grabadas en forma contiguas en el disco. Las celdas pueden o no contener información. Si queremos acceder a un registro determinado tenemos que especificar el número de celda en que se encuentra. En cada celda puede haber uno o más registros (en general es uno, si es más de uno se dice que el archivo tiene "buckets").

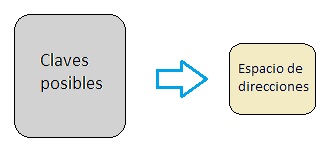
Hay dos formas de acceder:

1. *Acceso directo*: especificando el número de celda.
2. *Acceso secuencial*: a partir de una determinada celda el siguiente.

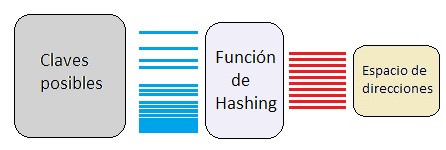
Características**:** Las celdas son de longitud fija. Los registros tienen un tamaño máximo igual al tamaño de la celda. Hay un número máximo de celdas que se establece al crear el archivo el cual puede ampliarse.

Cada celda tiene un número que la identifica (a su vez permite identificar al registro que contiene). Éste no tiene por qué figurar en el contenido del registro.

**ORGANIZACIÓN DIRECTA**

Los registros se graban en posiciones que dependen del valor de una clave. Se le llama clave a una parte del contenido del registro que permita identificarlo unívocamente. Cuando tenemos un conjunto de registros tenemos un conjunto de claves.

Pero quizás todo el rango de claves posibles no se mapea uno a uno con los datos almacenados, porque por ejemplo para el caso de los alumnos de la facultad, no hay tantos alumnos como padrones posibles. Necesitamos lo que se llama una *función de hashing* que se encargue de distribuir en forma pareja las claves en el espacio de direcciones.



Al ser el conjunto de claves mayor que el de las direcciones del disco, es lógico que ocurra el caso en que se asigne la misma posición a dos claves distintas. En ese caso se dice que hay una *colisión*. Cuando esto ocurre a las claves que provocan la colisión se las llama *sinónimos*. Hay distintos métodos para tratar colisiones.

Cada posición tiene que guardar si está llena, vacía o borrada. Si se quiere recuperar un registro, se debe leer el archivo, con la función de hashing buscamos la dirección donde debería estar y analizamos si hubo colisión o no.

El espacio asignado es, en principio, fijo, y no puede ampliarse, ya que eso implicaría cambiar la función de hashing.

**ORGANIZACIÓN INDEXADA**

Esta organización incluye índices en el almacenamiento de los archivos, facilitando la búsqueda de algún registro en particular sin necesidad de ver todo el archivo. Cada archivo tiene sus bloques de índices y de datos.

Un índice consiste en un listado de los campos claves del archivo, junto con la dirección correspondiente en el almacenamiento masivo.

Esta organización es conveniente para archivos con mediana volatilidad, actividad variable y tamaño relativamente estable.

Ventajas:

* Búsquedas más rápidas por la utilización de índices.
* Para tener acceso a los datos sólo es necesario traer a memoria el bloque de índices donde está el apuntador a los datos.

Desventajas:

* Ocupa más espacio en el disco que los archivos secuenciales, debido al uso del área de índices.
* Sólo se puede grabar en soportes direccionales.
* Los registros deben ser de longitud fija.

**ORGANIZACIÓN SECUENCIAL INDEXADA**

En esta organización, los registros se almacenan secuencialmente pero el archivo además tiene un índice, y sólo admite dispositivos de acceso directo. El diseño del registro tiene que tener un campo clave que va a servir de índice. Un mismo fichero puede tener más de un campo clave.

Se puede leer la información de dos modos:

* Acceso directo: se hace conociendo el contenido del campo clave del registro que queremos localizar.
* Acceso secuencial: los registros son leídos ordenados por el contenido del campo clave, independientemente del orden en que se fueron grabando.

El archivo consta de tres áreas:

* Área de índices: es un archivo secuencial creado por el sistema, en el que cada registro contiene el campo clave y la dirección de comienzo de cada registro.
* Área de datos: contendrá los registros de datos clasificados en orden ascendente por su campo clave.
* Área de excedentes u overflow: Se reserva esta área para añadir nuevos registros que no pueden ser colocados en el área primaria cuando se produce una actualización del archivo.

Ventajas:

* Rápido acceso por medio de la clave de registro. Permite dos modos de acceso.
* El sistema se encarga de relacionar la posición de cada registro con su contenido por medio del área de índices.
* Se pueden actualizar los registros en el mismo fichero, sin necesidad de crear un fichero nuevo de copia en el proceso de actualización.

Desventajas:

* Necesidad de espacio adicional para el área de índices y desaprovechamiento de espacios intermedios libres después de sucesivas actualizaciones.
* Las transacciones desordenadas tienden a reducir la eficiencia del proceso.
* Solo se puede utilizar soportes direccionales.

**BASES DE DATOS**

Es un sistema computarizado cuya finalidad es almacenar información y permitir a los usuarios recuperar y actualizar esa información en base a peticiones.

**Componentes de una BD**

* Datos: Es una entrada del sistema de información. Pueden ser integrados y compartidos.

*Integrados* porque se puede imaginar una BD como una unificación de varios archivos con redundancia (datos repetidos) mínima entre ellos.

*Compartidos* porque las piezas individuales de la BD son utilizadas por diferentes usuarios al mismo tiempo.

* HW: Unidades de procesamiento y unidades de almacenamiento principal y secundario.
* SW: Abarca el SO y el DBMS.

*Motor de BD*(DBMS): Es una pieza de SW ubicada entre el SO y los programas de aplicación de usuarios. Procesa todas las peticiones (agregar, eliminar, modificar y leer datos), ocultando a las aplicaciones los detalles de implementación.

* Usuarios: Hay varios tipos:

*Programadores*: Codifican aplicaciones de alto nivel para los usuarios finales. Estas aplicaciones son las que generan peticiones.

*Usuario final*: Utilizan las aplicaciones desarrolladas por los programadores que finalmente accederán en forma concurrente a la BD.

*Administrador de BD* (DBA): Personal técnico responsable del funcionamiento y disponibilidad de la BD.

*Administrador de datos* (DA): Persona no técnica responsable de las necesidades de información de la BD. Decide qué datos hay que guardar en la BD.

**Beneficios** con respecto a los archivos.

* Recuperación: La diferencia fundamental con los archivos convencionales es que al recuperar información, las BD no devuelven solamente los datos, como los archivos, sino también su estructura.
* Compactación: Los archivos de BD son administrados por el DBMS, quien resuelve las problemáticas de duplicación de datos y administración del espacio vacío.
* Velocidad: Las BD son más rápidas para obtener la información del negocio requerido, es decir, en el código se usarán pocas sentencias para acceder a los datos. Sería más rápido acceder directamente al SO pero lleva mucho más código.
* Productividad: Los trabajadores se concentran en la información de negocio requerida y no en los detalles técnicos que se necesitan para obtenerla.
* Información actualizada: Los cambios realizados sobre una BD se hacen visibles al instante para el usuario que la requiere.
* Seguridad: Las BD son más potentes que los sistemas de archivos ya que permiten mayor granularidad (nivel de detalles) en la regla de seguridad sobre los datos. Es decir que se le puede dar seguridad a cada campo de los registros.
* Manejo centralizado de la información: Los modelos de datos se crean y administran en forma centralizada.
* Integridad: Es el grado de corrección de los datos, coherente entre los datos y con la realidad.
* Transaccionabilidad: Manejar transacciones como unidades atómicas de trabajo, es decir, se hace todo o nada.
* Recuperación ante fallos: Las BD están preparadas para reponerse automáticamente ante cualquier fallo de un sistema computarizado.

**Tareas del DBA**

* *Respaldo*: Asegura que la BD tenga un mecanismo de respaldo (backup).
* *Integridad*: Ofrece reglas de integridad para la BD.
* *Seguridad*: Brinda controles de acceso físicos (servidores) y lógicos (usuarios y datos) a los datos.
* *Disponibilidad*: Cumple los contratos de servicio definidos.
* *Desempeño*: Ofrece buena performance, rapidez.
* *Soporte*: Brinda asesoramiento en materia de BD. El programador puede hacerle consultas.

**Independencia de datos** (Importante en empresas grandes y por la centralización)

Es la inmunidad o robustez que tienen las aplicaciones para soportar cambios en la representación y acceso a los datos. (que el usuario no se entere de los cambios)

Hay 3 tipos:

* Independencia de dispositivos: Capacidad de soportar cambios en los almacenamientos físicos. Lo da el HW y el SO.
* Independencia física: Las aplicaciones deben soportar cambios en los archivos. Lo brinda el DBMS.
* Independencia lógica: Las aplicaciones deben soportar cambios en el modelo de datos (tablas). Lo da el DBMS y hay que cumplir el modelo de datos.

**MODELOS DE BD TRADICIONALES**

El modelo determina el modo de organizar, manipular y almacenar los datos. Tiene 3 aspectos:

* *Estructural*: La estructura para la representación de los datos.
* *Manipulación*: Cómo se procesan los datos para obtener la información.
* *Integridad*: Garantizar la corrección de los datos.

**Bases de Datos Jerárquicas**

Es un tipo de sistema de gestión de BD que almacena la información en una estructura jerárquica que enlaza los registros en forma de estructura de árbol, en donde un nodo padre de información puede tener varios nodos hijo, y así sucesivamente, esto suele denominarse relación uno a muchos. Los nodos inferiores se subordinan a los que se hallan a su nivel inmediato superior. Un nodo que no tiene padre es llamado raíz, en tanto que los que no tienen hijos son conocidos como hojas.

Los datos se almacenan en forma de registros y cada uno consta de un conjunto de campos. Un conjunto de registros con los mismos campos se denomina fichero. Cada nodo del árbol representa un tipo de registro conceptual, es decir, una entidad. A su vez, cada registro está constituido por un número de campos que los describen (las propiedades o atributos de las entidades). Las relaciones entre entidades están representadas por las ramas.

El modelo jerárquico facilita relaciones padre-hijo y son unidireccionales. Las relaciones se establecen mediante punteros entre registros. Es decir, un registro hijo contiene la dirección física en el medio de almacenamiento de su registro padre. Esto tiene una ventaja fundamental sobre las bases de datos relacionales: el rendimiento. El acceso de un registro a otro es prácticamente inmediato sin necesidad de consultar tablas de correspondencia.

Las relaciones jerárquicas entre diferentes tipos de datos pueden hacer que sea muy sencillo responder a determinadas preguntas, pero muy difícil el contestar a otras.

Una BD de tipo jerárquico recorre los distintos nodos de un árbol en el siguiente orden:

1. Visitar la raíz.
2. Visitar el hijo más a la izquierda, si lo hubiera, que no haya sido visitado.
3. Si todos los descendientes del segmento considerado se han visitado, volver a su padre e ir al punto 1.

Desventajas:

Todos sus problemas derivan del hecho de que el sistema gestor de BD no implementa ningún control sobre los propios datos, sino que queda en manos de las aplicaciones garantizar la integridad de los datos. Dado que todas las aplicaciones están sujetas a errores y fallos, esto es imposible en la práctica.

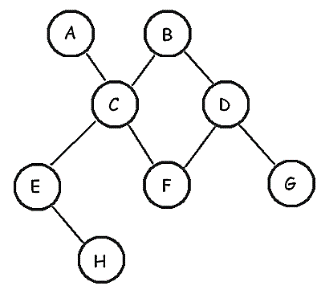
* *Duplicidad de registros:* no se garantiza la inexistencia de registros duplicados. Esto también es cierto para los campos clave. Es decir, no se garantiza que dos registros cualesquiera tengan diferentes valores en un subconjunto concreto de campos.
* *Integridad referencial:* no existe garantía de que un registro hijo esté relacionado con un registro padre válido. Por ejemplo, es posible borrar un nodo padre sin eliminar antes los nodos hijo, de manera que éstos últimos están relacionados con un registro inválido o inexistente.
* *Desnormalización:* este no es tanto un problema del modelo jerárquico como del uso que se hace de él. Sin embargo, a diferencia del modelo relacional, las bases de datos jerárquicas no tienen controles que impidan la desnormalización de una BD. Por ejemplo, no existe el concepto de campos clave o campos únicos.

**Bases de Datos de Red**

Conformado por una colección de registros, los cuales están conectados entre sí por medio de enlaces en una red. El enlace es exclusivamente la asociación entre dos registros, así que podemos verlo como una *relación estrictamente binaria.* Si bien una relación involucra a 2 registros, cada registro puede tener más de un enlace permitiendo la relación de muchos a muchos, lo cual fue un gran avance por sobre el modelo jerárquico.

Elementos del modelo:

* *Elemento de datos* (Data Item): es la unidad de datos más pequeña a la que se puede hacer referencia en el modelo de red. Un elemento de datos debe tener un nombre, y una ocurrencia del mismo contiene un valor que puede ser de distintos tipos (booleano, numérico, etc.)
* *Registro* (Record): es la unidad básica de acceso y manipulación de la BD y se corresponde con el concepto de registro (en los ficheros) y de entidad (en otros modelos).
* *Conjunto* (SET o COSET): es una colección de dos o más tipos de registros que establece una vinculación entre ellos. Constituye el elemento clave y distintivo de este modelo de datos. Los conjuntos son el modo de representar las relaciones.

Estructura

Similar a la del modelo jerárquico, es decir, un árbol de registros o nodos. Dichos nodos están relacionados mediante enlaces denominados punteros o ligas (desde un punto de vista físico). La particularidad del modelo de red es que permite a cada registro tener múltiples registros paternales y de hijos, formando una estructura de enrejado.

**MODELO DE BD RELACIONALES** (se usa ahora)

Sus 3 aspectos son:

* Estructural: Los datos se estructuran solamente en tablas. Las tablas representan una abstracción de la forma en que los datos están almacenados físicamente, es decir, las tablas son la estructura lógica de la BD, no la estructura física.
* Manipulación: Define operaciones cuyos operando son tablas y sus resultados también. Esta propiedad es fundamental para las bases de datos relacionales puesto que el *tipo* de salida de cada operación es del mismo tipo que las entradas, esto permite encadenar operaciones formando lo que se conoce como “expresiones anidadas”. (Consultas al DBMS sobre lo que quiero)
* Integridad: Define reglas de integridad implementadas como restricciones sobre las tablas. (restricciones de dominio, clave primaria)

Estos aspectos están demostrados matemáticamente:

* El estructural se demuestra con la teoría de conjuntos. Una relación representa una tabla (R = {(a,b), (c,d)} )
* El de manipulación con el álgebra relacional y el cálculo relacional. Se basa en la teoría de los números reales pero toman tablas como operando y regresan una tabla como resultado.

Terminología del modelo relacional

Si pensamos en una relación como una tabla, entonces una tupla corresponde a una fila de dicha tabla y un atributo a una columna; al número de tuplas se le llama cardinalidad y al número de atributos se le denomina grado. Un dominio es no es más que un tipo de datos, es decir, un conjunto de valores (todos los posibles para dicho tipo), de donde se toman los valores de atributos específicos de relaciones específicas.

Un tipo determinado puede ser escalar (no tienen componentes visibles para el usuario) o no escalar (tiene componentes visibles para el usuario).

No existen apuntadores que conecten una tabla con otra. Las conexiones están representadas por la aparición de un valor en dos o más tablas.

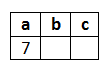
**Operaciones** que define el álgebra relacional, en donde todos los resultados son tablas nuevas:

* Proyección: Operación unaria que tiene como operando una tabla y produce como resultado la misma tabla pero con menos campos. Adicionalmente, la operación requiere un parámetro que es una lista de campos.



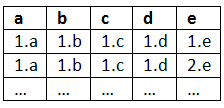
=

* Selección: Operación unaria que tiene un operando y produce como resultado la misma tabla pero con menos registros. Adicionalmente posee un parámetro que es una condición.



=

* Producto cartesiano: Operación binaria que posee dos operandos y el resultado es una tabla con la suma de los campos de los operandos y que contiene todas las combinaciones posibles entre los registros.



x =

* Reunión: Operación binaria que tiene dos operandos y un parámetro que es una lista de campos. El resultado es el producto cartesiano de los operandos seguido de una selección en donde se igualan los campos del parámetro. (Une las tablas de costado)

Filtra que sean iguales



= = (T1 x T2)

T1 T2

* Unión: Operación binaria cuyos operandos deben tener la misma cantidad y calidad de campos. El resultado es una tabla con los mismos campos que sus operandos pero con la sumatoria de sus registros. (Une las tablas una arriba de otra)



=

* Intersección: Operación binaria, con la restricción de que los operandos deben tener la misma cantidad y calidad de campos y el resultado es una tabla con los mismos campos que sus operandos pero con los registros comunes a ambos.
* Diferencia: Operación binaria con la restricción de igual cantidad y calidad de campos en sus operandos y el resultado es una tabla con los mismos campos que sus operandos pero con los registros que pertenecen al primero y no pertenecen al segundo.
* División: Es la división de conjuntos de sus operandos.

El DBMS tienen la inteligencia para simplificar las operaciones que se codifiquen, por lo tanto, se puede escribir de la forma que mejor se entienda.

**SQL** (Standard Query Language)

Es el lenguaje estándar para trabajar con BD relacionales y es soportado prácticamente por todos los productos en el mercado. Surge en los 70’ como una propuesta de lenguaje para manipular BD relacionales. Está basado en el álgebra relacional. Se estandarizó generando un ANSI – SQL en el año 1992 y se creó *SQL 92 (Estándar Internacional de Lenguaje de BD SQL 1992)*, el cual permitió programa para cualquier DBMS.

Luego surge un nuevo estándar que expandió la sintaxis de SQL 92. Dicho estándar se denominó *SQL 3* pero no gozó de demasiado uso.

En un principio, SQL fue diseñado para ser específicamente un "sublenguaje de datos". Sin embargo, con la incorporación en 1996 de la característica de PSM (Módulos Almacenados Persistentes) al estándar, SQL se convirtió en un lenguaje completo.

Las operaciones de SQL son *set at time* que trabaja sobre conjuntos sin recorrer nada, en contra de row at time que trabaja sobre cada registro).

SQL es un lenguaje *host*: está dentro de un lenguaje de programación cualquiera y general del sistema.

App

SQL

DBMS

SO

BD en disco.

C#

-

-

-

-

-

SQL

SQL

Server

Devuelve

tabla

**Sentencias**

* DDL (Data Definition Language): Define el modelo de datos.

. Create table

. Alter table: modifica estructura

. Drop table: elimina

* DML (Data Manipulation Language): Manipula los registros de la tabla.

. Insert

. Update: actualiza

. Delete

. Select: lee o consulta

**CONSULTA**

**Sentencia ‘SELECT’**

Tiene 3 cláusulas fundamentales:

* *SELECT* = Proyección
* *FROM* = Producto cartesiano x
* *WHERE* = Selección

Otras cláusulas:

* **Select DISTINCT**: Elimina duplicados (considera todos los campos, si hay al menos uno diferente ya no los considera duplicados). Usar solo cuando es necesario (es costoso).
* **Comodín**: muestra todos los campos.
* **Alias**: Para llamar por otro nombre. Mejora la visualización de los campos de la tabla
* **Constantes**: Pone un valor fijo para todos los registros

**Conectores**

*Lógicos*: Unen dos condiciones transformando la expresión en una única condición compuesta. [*And, not, or*]

*De inclusión*: Evalúan true/false. Para simplificar las sentencias con lógicos. [in - not in], [= > < ], [all / any]

**SUBCONSULTA**

Es una sentencia SELECT completa embebida en la cláusula WHERE de otra sentencia SELECT completa utilizando conectores de inclusión. La tabla que genera la subconsulta se desecha por lo que no se pueden usar sus campos para mostrar. Puede haber varios niveles de subconsulta. Tiene mejor desempeño que el producto cartesiano porque genera tablas más chicas.

Tipos de subconsulta:

* *Anidadas*: Son aquellas que no hacen referencia a ningún objeto de la consulta exterior. El orden de procesamiento lógico es: primero se ejecuta la subconsulta una vez y luego se ejecuta la consulta exterior una vez. Las subconsultas anidadas son independientes. Este tipo es más eficiente que las subconsultas correlacionadas ya que se ejecutan una sola vez.
* *Correlacionadas*: Son aquellas que hacen referencia a objetos de la consulta externa. El orden de procesamiento lógico es: primero se ejecuta la consulta externa una vez, y por cada registro a validar en su cláusula WHERE se ejecuta la subconsulta. No son independientes.

**GROUP BY**

Un grupo es un conjunto de registros que tienen valores iguales en uno o más campos. Esos campos son los que definen el grupo. El objetivo es poder sacar medidas resumen. Es más lento que el distinct.

Agrupar siempre por las claves. Los campos que se muestren el el SELECT deben ser los mismos que aparecen en el GROUP BY

Funciones de resumen:

Son funciones que se utilizan en las cláusulas posteriores en el orden del procesamiento lógico del GROUP BY y permiten extraer medidas de resumen de cada uno de los grupos. Es necesario que esté el GROUP BY. El DBMS reconoce las funciones resumen que se escriben en todas las sentencias y las calcula al momento de agrupar.

**Orden de ejecución completo:**

6 → SELECT

7 → INTO

1 → FROM

2 → WHERE

3 → GROUP BY

4 → HAVING

5 → ORDER BY

**EVOLUCIÓN DE LIBRERÍAS DE ACCESO A DATOS**

* DBLibrary: librerías que proporcionaba directamente el fabricante del DBMS. Cada librería tenía su forma particular de usarse y las aplicaciones no tenían portabilidad.
* ODBC: librería fabricada por Microsoft cuyo objetivo es la portabilidad en las aplicaciones cliente. Se basa en una capa de librerías genéricas conectadas a una serie de drivers. Estos drivers son los provistos por los fabricantes de DBMS. La programación era de muy bajo nivel.

ODBC

drivers

drivers

drivers

* OLEDB: fabricado por Microsoft con la misma arquitectura y objetivo que ODBC pero incorpora características de la programación orientada a objetos. La programación sigue siendo de bajo nivel. Los drivers se llaman providers pero son lo mismo.
* DAO (Data Access Object): fabricada por Microsoft que encapsula ODBC permitiéndole al programador programar en un nivel más alto. (se pone primero DAO, después ODBC y después las librerías) DAO se conecta directamente con BD Access.

DAO

ODBC

drivers

drivers

drivers

* RDO (Remote Data Object): evolución de DAO agregándole características como por ejemplo soporte para BD distribuidos. Usa ODBC también.
* ADO (Active Data Object): Encapsula OLEDB brindándole al programador una interface de alto nivel.

ADO

OLEDB

providers

providers

providers

* JDBC: mismas características de ODBC pero para Java y con interfaces de alto nivel. Es un lenguaje interpretado. Los de antes eran compilados.
* ADO.NET: evolución de ADO para la plataforma .NET de Microsoft. Tiene OLEDB también.
* Mapeadores objeto relacional: son librerías que funcionan por encima de las librerías tradicionales de alto nivel. Su objetivo es mapear automáticamente los objetos de negocio del programa a las tablas de la BD relacional. Los mapeadores generan automáticamente la sentencia de SQL. Ejemplos: NHibernet (para .net), Hibernet (java), Linq (Microsoft), Entity framework (Microsoft), Data nucleus.

ADO.NET

OLEDB

providers

providers

providers

NHibernate

C#

**ARQUITECTURA ANSI-SPARC**

Divide la estructura de la BD en 3 niveles:

* Externo: Administra las diferentes visiones o vistas que tienen los diferentes usuarios o roles que utilizan las aplicaciones. Se refiere a la forma en que un usuario individual ve los datos. Pueden existir varias visiones o vistas y cada una abarca solamente una porción del modelo lógico que corresponde al perfil del usuario interesado.

Cada usuario tiene a su disposición un lenguaje (o una aplicación) que incluyen un sublenguaje de datos (SLD), es decir, un subconjunto de lenguaje total que se ocupe específicamente de los objetos y operaciones de la BD. En principio, cualquier SLD es una combinación de por lo menos dos lenguajes subordinados: DDL (lenguaje de definición de datos) y DML (lenguaje de manipulación de datos).

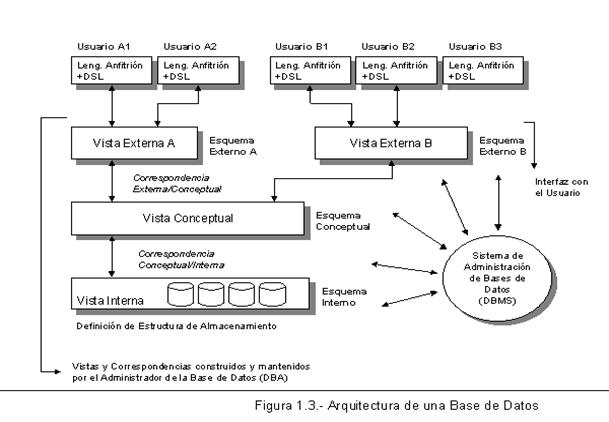
Surge entonces la siguiente clasificación:

* Fuertemente acoplados: el lenguaje anfitrión y el SLD son distintos y fácilmente distinguibles.
* Débilmente acoplados: el lenguaje anfitrión y el SLD son muy difíciles de distinguir.
  + MAPPING Externo-Conceptual: Administra la relación entre el modelo de datos conceptual y las diferentes vistas del nivel externo. Garantiza la independencia lógica de datos.
* Conceptual: Administra la estructura lógica de los datos. Mantiene la representación del *modelo de datos* lógico, manejando cuestiones de estructuras, de seguridad, de manipulación y de integridad. Este nivel esconde los detalles del almacenamiento físico y se concentra en describir entidades, atributos, tipos de datos y relaciones.
  + MAPPING Interno-Conceptual: Administra la relación entre el nivel conceptual y el interno, es decir, como se distribuyen los registros de las tablas del nivel conceptual en los diferentes archivos físicos del nivel interno. Garantiza la independencia física de datos.Si se dividen los archivos en diferentes discos, se aumenta la velocidad de búsqueda.
* Interno: Es el de mayor relación con el SO y contiene la administración de los *archivos convencionales* en MS. Administra la estructura interna de los archivos, la organización interna y los métodos de acceso.

Al definir este nivel se debe tener en cuenta el espacio de almacenamiento:

* Tamaño.
* Tasa de crecimiento.
* Velocidad de acceso: velocidad de discos, concurrencia de discos, distribución de los datos en los archivos (mapping i-c), RAID.

Es responsabilidad del DBA: definir y mantener los distintos niveles y mappings, definir restricciones de seguridad y de integridad, establecer políticas de backups, establecer un enlace con los usuarios, ofrecer asesoramiento técnico, supervisar el rendimiento y responder a los requerimientos cambiantes, entre tantas otras tareas.



**RAID**: Arreglo de disco físico.

Arreglo de discos físicos: puedo elegir usar varios discos físicos de modo que el sistema operativo vea todo como uno solo.

* Software (a través del sistema operativo).
* Hardware (placas especiales a las que se enchufan los discos).

*Tipos*:

* RAID 0: divide todo lo que se almacena en una unidad en los discos disponibles.
* Duplica la velocidad de acceso.
* No tiene redundancia. Si se rompe un disco, se pierde el archivo.
* RAID 1: todo se almacena en todos los discos (duplicado).
* Redundancia. Mejora el desempeño de lecturas.
* Pérdida de la mitad del almacenamiento físico. Pequeña penalidad en la escritura.
* RAID 5: Como mínimo tener tres discos. Guarda un componente en uno, el otro componente en otro y en el tercero guarda la paridad.
* Mejora en 2/3 la velocidad de acceso. Tiene redundancia.
* Mínimo 3 discos para armar el RAID. Pierdo 1/3 de espacio de almacenamiento.
* RAID 10: Mínimo 4 discos. Combinación entre 0 y 1. Divide los discos a la mitad: arriba usa RAID 0 y abajo 1
* Duplica la velocidad de acceso. Tiene redundancia.
* Mínimo 4 discos para armar el RAID. Se pierde el 50% del espacio de almacenamiento.

**SISTEMA DE ADMINISTRACIÓN DE BD**

El **DBMS (sistema de administración de BD)** es el software que maneja todo acceso a la BD. De manera conceptual, lo que sucede es lo siguiente:

Un usuario emite una petición de acceso, utilizando algún sublenguaje de datos específico (SQL). El DBMS intercepta esa petición y la analiza. Inspecciona el esquema externo para ese usuario, el Mapping externo/conceptual correspondiente, el esquema conceptual, el Mapping interno/conceptual y la definición de la estructura de almacenamiento. El DBMS ejecuta las operaciones necesarias sobre la BD almacenada.

**Funciones del DBMS**:

* Definición de datos: el DBMS debe ser capaz de aceptar definiciones de datos en la forma fuente y convertirlas a la forma objeto correspondiente. Debe incluir entre sus componentes un procesador o compilador DDL para cada uno de los diversos DDLs (lenguajes de definición de datos).
* Manipulación de datos: el DBMS debe ser capaz de manejar peticiones para recuperar, actualizar o eliminar datos existentes en la BD o agregar nuevos datos a ésta. Debe incluir un componente procesador o compilador DML para tratar con el DML (lenguaje de manipulación de datos).
* Optimización y ejecución: las peticiones DML deben ser procesadas por el componente *optimizador* de la BD, cuya finalidad es determinar una forma eficiente de implementar la petición. Las peticiones optimizadas se ejecutan entonces bajo el control del *administrador en tiempo de ejecución*.
* Seguridad e integridad de los datos: el DBMS debe vigilar las peticiones del usuario y rechazar todo intento de violar las restricciones de seguridad y de integridad definidas por el DBA.
* Recuperación de datos y concurrencia: el DBMS (o algún componente de software relacionado: administrador de transacciones) debe imponer ciertos controles de recuperación y concurrencia.
* Diccionario de datos: el DBMS debe proporcionar una función de diccionario de datos (contiene datos acerca de los datos, es decir, metadatos).
* Rendimiento: el DBMS debe realizar todas las tareas antes identificadas de la manera más eficiente posible.

Podemos resumir todo lo anterior diciendo que la finalidad general del DBMS consiste en proporcionar una interfaz de usuario para el sistema de BD.

**Comparación entre los sistemas de administración de BD y los sistemas de administración de archivos**

El administrador de archivos es el componente del SO subyacente que administra los archivos almacenados; por lo tanto, hablando en términos generales, está "más cerca del disco" de lo que lo está el DBMS (De hecho, el DBMS es generalmente construido sobre algún tipo de administrador de archivos). Por lo tanto, el usuario de un sistema de administración de archivos podrá crear y destruir archivos almacenados y realizar operaciones sencillas de recuperación y actualización sobre registros almacenados en dichos archivos.

Los sistemas de administración de archivos, en contraste con el DBMS:

* Los administradores de archivos no están al tanto de la estructura interna de los registros almacenados, de ahí que no puedan manejar peticiones que se basen en el conocimiento de esa estructura.
* Por lo regular ofrecen poco o ningún soporte para las restricciones de seguridad y de integridad y para los controles de recuperación y concurrencia.
* No hay un concepto real de diccionario de datos en el nivel del administrador de archivos.
* Proporcionan mucho menos independencia de datos que el DBMS.
* Por lo regular los archivos no están "integrados" o "compartidos" en el mismo sentido que en una BD (normalmente son exclusivos de cierto usuario o aplicación en particular).

**Optimización**

Se dice a menudo que los lenguajes relacionales no son de procedimientos, en el sentido de que los usuarios sólo especifican qué datos desean, no cómo obtenerlos. El proceso de "navegar" por los datos a fin de satisfacer la petición del usuario es realizado automáticamente por el sistema, en vez de ser realizado en forma manual por el usuario. Por esta razón, en ocasiones se dice que los sistemas relacionales realizan una *navegación automática*. Podemos decir entonces, que lenguajes como SQL (relacionales) están en un nivel más alto de abstracción que los lenguajes de programación, lo que aumenta la productividad de los sistemas relacionales.

Decidir cómo realizar la navegación automática es responsabilidad de un componente muy importante del DBMS denominado *optimizador*. Es trabajo del optimizador seleccionar una forma eficiente de implementar cada petición relacional del usuario.

Dada una petición en particular, el optimizador elegirá la estrategia para implementar esa petición basándose en consideraciones como las siguientes:

* A qué tablas se hace referencia en la petición.
* Qué tan grandes son actualmente esas tablas.
* Qué índices existen.
* Qué tan selectivos son esos índices.
* Cómo están agrupados físicamente los datos en el disco.
* Qué operadores relacionales están involucrados.

**Catálogo**

Todo DBMS debe proporcionar una función de *catálogo* o *diccionario*. El catálogo es el lugar donde se guardan los diversos esquemas (externo, conceptual, interno) con su respectivos mappings, entre otras cosas. En otras palabras, el catálogo contiene información detallada (*metadatos*) respecto de los distintos objetos que son de interés para el propio sistema.

Por ejemplo, el optimizador utiliza información del catálogo relacionada con los índices y otras estructuras de almacenamiento físico (así como otro tipo de información) para poder decidir cómo implementar las peticiones del usuario. En forma similar, el subsistema de seguridad utiliza información del catálogo relacionado con los usuarios y las restricciones de seguridad, principalmente para autorizar o negar dichas peticiones.

Una característica de los sistemas relacionales es que en ellos el propio catálogo consiste en tablas (tablas del sistema). Como resultado, los usuarios pueden consultar el catálogo exactamente de la misma manera en que consultan sus propios datos

**Variables de relación base y vistas**

Las expresiones relacionales nos permiten obtener otros valores de relación a partir de los ya dados. A las variables de relación originales (tablas existentes) se les denomina *variables de relación base* y a sus valores de relación se les llama *relaciones base*; a una relación que es o que puede ser obtenida a partir de dichas relaciones base por medio de alguna expresión relacional, se le denomina relación *derivada* o *derivable*.

Por lo regular los sistemas relacionales también manejan otro tipo de variables de relación con nombre, denominada *vista*, cuyo valor en cualquier momento dado es una *relación derivada*. El valor de una vista determinada en un momento dado, es cualquiera que sea el resultado de evaluar cierta expresión relacional en ese momento; dicha expresión relacional es especificada en el momento de que se crea la vista en cuestión. La vista es en efecto sólo una ventana hacia el interior de la variable de relación base.

La diferenciación entre variables de relación base y vistas se caracteriza con frecuencia de la siguiente manera:

* Las variables de relación base “existen realmente”, en el sentido de que representan datos que en realidad están almacenados en la BD.
* En contraste, las vistas “no existen realmente”, sino que sólo proporcionan diferentes formas de ver a los datos reales en un instante dado.

**TRANSACCIÓN**

Unidad de trabajo lógica que comprende por lo regular a varias sentencias SQL. Es una unidad de trabajo porque todas las operaciones dentro de la transacción se toman como un todo y también es una unidad de recuperación porque si algunas de las operaciones falla, se vuelve toda la transacción atrás. Las transacciones empiezan con un BEGIN TRANSACTION y pueden finalizar con un COMMIT si la transacción terminó correctamente o un ROLLBACK si la transacción debe volverse atrás o deshacerse.

Una sola sentencia se considera transacción porque puede que tenga que actualizar varios registros y por si hay algún fallo de energía.

BEGIN TRANSACTION

UPDATE cuenta1

UPDATE cuenta2

IF OK

COMMIT

ELSE

ROLLBACK

Propiedades de las transacciones (ACID):

* *Atomicidad*: las transacciones son atómicas, se ejecuta todo o nada.
* *Consistencia*: dejan siempre a las BD en un estado consistente. Cuando arranca una transacción, la BD está en un estado consistente. Cuando finaliza, la consistencia debe seguir estando garantizada. Esto último no ocurre en el medio del procesamiento de una transacción.
* *Aislamiento*: Las transacciones están aisladas entre sí, es decir, una transacción no interfiere en el procesamiento de otra y ambas mantienen la consistencia de la BD. (concurrencia)
* *Durabilidad*: Una vez que una transacción es confirmada (COMMIT), las modificaciones que realizó en la BD son permanentes más allá de que ocurra una caída posterior.

Recuperación de transacciones: Cuando una transacción es confirmada, se establece un COMMIT POINT o punto de confirmación, y corresponde al final de una unidad lógica de trabajo y a un estado de consistencia de la BD. En cambio, cuando una transacción de revertida (ROLLBACK) todas las modificaciones deben volverse al estado en que se encontraban en el COMMIT POINT anterior.

Log transacciones (T-Log): El DBMS, para cada BD, mantiene en disco un log de transacciones en donde va almacenando secuencialmente todas las operaciones que se realizan sobre los datos, como así también la información necesaria para revertirlas. Se utiliza el protocolo de escritura adelantada que dice que antes de realizar cualquier modificación en la BD se debe persistir la operación en el T-Log. Cada BD tiene los archivos donde están las tablas y un archivo para T-LOG (siempre tiene que estar en un disco aparte por la velocidad). Las BD primero se escriben en el LOG y después en la tabla.

La BD usa la RAM como caché (deja las tablas en RAM para ser más rápido)

Protocolo de escritura adelantada: Escribe primero en LOG que está en disco, después los pasa a RAM donde están las tablas y después por detrás se actualizan las tablas en disco (check point). Esto garantiza la recuperación transacciones (recovery).

El Log tiene una parte activa y otra pasiva. La activa contiene las operaciones de las transacciones que aún no finalizaron. En cambio, la pasiva, contiene las operaciones de aquellas transacciones que ya finalizaron. No guardar la parte pasiva o hacerle backup para que no ocupe espacio.

||||||||||||| ||||||||||||| ||||…..

T1 terminada

T2 terminada

T3 ejecutándose

Pasivas

Activa

CHECK POINT: Son momentos regulares en el tiempo en que el DBMS persiste en disco todas las modificaciones existentes en los buffers de memoria RAM. Adicionalmente se registra en el T-Log el momento en que se realizó ese check point.

m

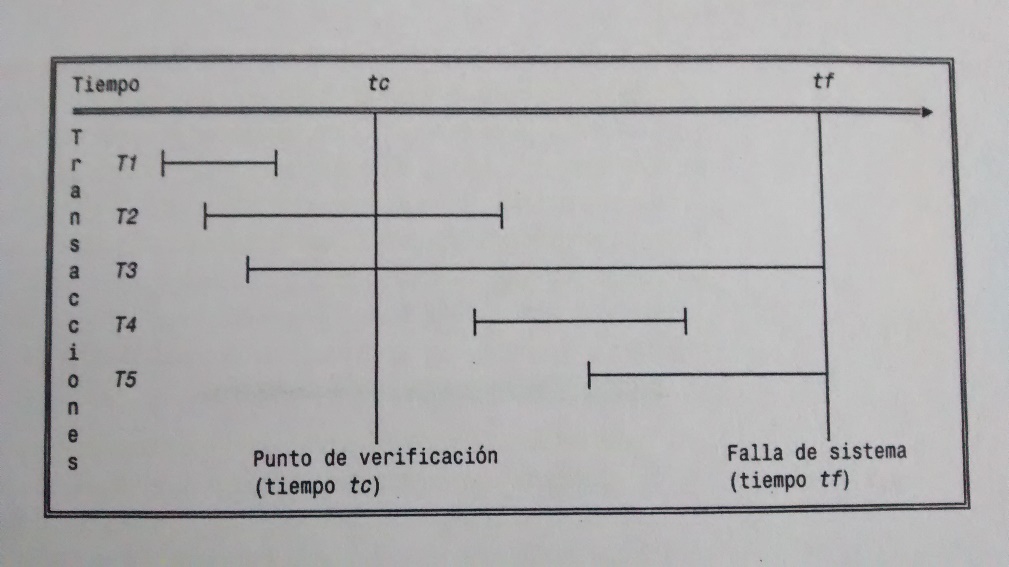
m

||||||||||||| ||||||||||||| ||||…..

**RECOVERY**

* **Falla Local**: Se ejecuta cuando una transacción en particular hizo rollback e implica deshacer los cambios únicamente de esa transacción. Para eso debe retroceder en el T-Log a partir de la ejecución del rollback e ir revirtiendo todos los comandos ejecutados por esa transacción hasta el comienzo de la misma. Deshace todos los cambios en memoria, sin importar si las transacciones están en disco o en memoria.
* **Falla Global**: Es la que afecta al DBMS completo en donde hay que deshacer los comandos de las transacciones que aún no finalizaron y rehacer los comandos de aquellas transacciones que finalizaron correctamente pero sus modificaciones no fueron persistidas en disco.
* *Soft Crash* (caídas suaves): Son caídas que no comprometen los medios de almacenamiento secundario y únicamente se pierde el contenido de la RAM.

Recuperación de la BD ante un Soft Crash:



* En el tiempo sucede una falla de sistema.
* El punto de verificación más reciente anterior al tiempo , se tomó en el tiempo .
* Las transacciones de tipo terminaron (satisfactoriamente) antes del tiempo .
* Las transacciones de tipo se iniciaron antes del tiempo , y terminaron (satisfactoria mente) después del tiempo y antes del tiempo .
* Las transacciones de tipo también se iniciaron antes del tiempo , pero no habían terminado en el tiempo .
* Las transacciones de tipo se iniciaron después del tiempo y terminaron (satisfactoriamente) antes del tiempo .
* Por último, las transacciones de tipo también se iniciaron después del tiempo , pero no habían terminado en el tiempo .

DBMS al iniciar, ejecuta un proceso de recovery general (global) para dejar la BD en un estado consistente. Durante el recovery, los datos no son accesibles. Luego se posiciona (en el T-Log) en el momento del último Check Point ejecutado y procede de la siguiente manera:

1. Comienza con dos listas de transacciones, la lista DESHACER y la lista REHACER. Iguala la lista DESHACER con la lista de todas las transacciones dadas en el registro de punto de verificación más reciente y deja vacía la lista REHACER.
2. Busca hacia delante en el T-Log comenzando en el registro del punto de verificación.
3. Si encuentra una entrada BEGIN TRANSACTION en la bitácora para la transacción T, añade T a la lista DESHACER.
4. Si encuentra una entrada COMMIT para la transacción T, mueve a T de la lista DESHACER a la lista REHACER.
5. Por último, elimina de la tabla DESHACER las transacciones que empezaron luego del CHEK POINT y no terminaron.

Ahora el sistema trabaja hacia atrás en el T-Log, deshaciendo las transacciones que están en la lista DESHACER, y luego vuelve a trabajar hacia delante volviendo a hacer las transacciones que están en la lista REHACER teniendo en cuenta la necesidad de ejecutar Rollbacks parciales (Se deshace a partir del Check Point hacia atrás y se rehace toda la transacción).

Una vez terminada toda la actividad de recuperación, es entonces (y sólo entonces) que el sistema está listo para aceptar más trabajo.

* *Hard Crash* (caídas fuertes): Son caídas en donde hay compromiso en los medios secundarios. Ej: Se rompe un disco.

Para recuperarse de esto, implica restaurar las copias de seguridad que se han realizado en otro soporte. Una vez restaurada, si el medio de almacenamiento en donde estaba el T-Log no fue comprometido se lo utiliza para ejecutar el proceso de recovery normal con la salvedad de que se inicia en el último check point anterior de la copia de seguridad. No hay necesidad de deshacer transacciones que estaban en progreso al momento de la falla, ya que por definición todas las actualizaciones de esas transacciones ya han sido "deshechas" (de hecho, se perdieron).

De esta forma no hay pérdida de datos (se recupera parte activa y pasiva, aunque deba volverse atrás una vez restauradas).

En cambio, si el medio del almacenamiento del Log sí se vio comprometido, los datos cargados en horarios sin copia de seguridad se perdieron.

**Confirmación (COMMIT) en dos fases**

La confirmación de dos fases es importante siempre que una transacción dada pueda interactuar con varios “administradores de recursos” independientes, donde cada uno administra su propio conjunto de recursos recuperables y mantiene su propio T-Log. Si la transacción termina satisfactoriamente, se deben confirmar todas sus actualizaciones, tanto para los datos de una BD (1) como para otra BD distinta (2); por el contrario, si falla, todas sus actualizaciones deben ser deshechas.

Esto nos lleva a concluir que no tiene sentido que la transacción emita, un COMMIT para 1 y un ROLLBACK para 2, o viceversa; y a que incluso si emitiera la misma instrucción para ambos, el sistema aún podría fallar entre una y otra, con resultados desafortunados. Por lo tanto, en vez de ello, la transacción emite un solo COMMIT (o ROLLBACK) a *nivel sistema*.

Ese COMMIT o ROLLBACK “global” es manejado por un componente del sistema llamado *coordinador*, cuya tarea es garantizar que ambos administradores de recursos confirmen o deshagan al unísono las actualizaciones de las que son responsables, y además proporcionar esa garantía, aunque el sistema falle a mitad del proceso. Quien permite que el coordinador proporcione esta garantía es el *protocolo de confirmación de dos fases*, el cual funciona de la siguiente manera:

Supongamos que la transacción ha terminado satisfactoriamente su procesamiento de BD y por lo tanto, la instrucción al nivel sistema que emite es COMMIT y no ROLLBACK. Al recibir la petición de COMMIT, el coordinador realiza el siguiente proceso de dos fases:

1. Primero, da instrucciones a todos los administradores de recursos a fin de que estén listos para manejar la transacción. Luego, cada administrador de recursos involucrado debe forzar todos los registros de T-Log de los recursos locales usados por la transacción, hacia su propia T-Log físico (es decir, hacia el almacenamiento no volátil); esto con el fin de que, sin importar qué pase después, el administrador de recursos tenga ahora un registro permanente del trabajo que hizo a nombre de la transacción. Suponiendo que la escritura forzada es satisfactoria, el administrador de recursos responde ahora un "OK" al coordinador, y en caso contrario responde "No OK".
2. Cuando el coordinador ha recibido las respuestas de todos los participantes, fuerza una entrada en su propia bitácora física registrando su decisión con respecto a la transacción. Si todas las respuestas fueron "OK", esa decisión es "confirmar", y si alguna respuesta fue "No OK", la decisión es "deshacer".

*De cualquier forma, el coordinador informa después su decisión a cada participante y luego cada participante debe confirmar o deshacer la transacción localmente según se le indica*. Cada participante debe hacer lo que dice el coordinador en la fase 2; éste es el protocolo. La apariencia del registro de decisión en el T-Log físico del coordinador es lo que marca la transición de la fase 1 a la 2.

Si el sistema falla durante el proceso general, el procedimiento de reinicio buscará el registro de decisión en el T-Log del coordinador. Si lo encuentra, el proceso de confirmación de dos fases puede continuar donde se quedó. Si no lo encuentra, da por hecho que la decisión fue "deshacer" y de nuevo el proceso puede terminar en forma adecuada.

**Concurrencia**

El termino concurrencia se refiere al hecho de que los DBMSs permiten que muchas transacciones accedan a una misma BD a la vez. Se necesita algún tipo de *mecanismo de control de concurrencia* para asegurar que las transacciones concurrentes no interfieran entre sí.

Tres problemas de concurrencia

* El problema de la actualización perdida

La transacción A recupera alguna tupla en el tiempo ; la transacción B recupera la misma tupla en el tiempo ; la transacción A actualiza la tupla en el tiempo (con base en los valores vistos en el tiempo ), y la transacción B actualiza la misma tupla en el tiempo (con base en los valores vistos en el tiempo , que son los mismos vistos en el tiempo . **La actualización de la transacción A se pierde en el tiempo , ya que la transacción B la sobrescribe.**

|  |  |  |
| --- | --- | --- |
| Transacción A | Tiempo | Transacción B |
| - | | | - |
| Recuperar |  | - |
| - | | | - |
| - |  | Recuperar |
| - | | | - |
| Actualizar |  | - |
| - | | |  |
| - |  | Actualizar |

* El problema de la dependencia no confirmada

Este problema ocurre cuando la BD permite que una transacción recupere o actualice una tupla que ha sido actualizada por otra transacción pero que aún no ha sido confirmada por esa misma transacción. **Puede suceder que esta última transacción (encargada de actualizar la información en un primer momento) que no ha terminado, finalice de manera no satisfactoria ejecutando un ROLLBACK y por lo tanto la transacción A está operando sobre una suposición falsa o, en el caso de que intente actualizar , dicho cambio simplemente se pierde.**

|  |  |  |
| --- | --- | --- |
| Transacción A | Tiempo | Transacción B |
| - | | | - |
| - |  | Actualizar |
| - | | | - |
| Recuperar (o actualizar) |  | - |
| - | | | - |
| - |  | ROLLBACK |
| - | | | - |

* El problema del análisis inconsistente

Considere dos transacciones, A y B, operando sobre tuplas de una cuenta (): la transacción A está sumando saldos de cuenta y la transacción B está transfiriendo una cantidad de 10 de la cuenta 3 a la cuenta 1. **El resultado de 110 producido por A es obviamente incorrecto; y si A continuara y escribiera ese resultado en la BD, dejaría en efecto a la BD en un estado inconsistente. Decimos que A ha visto un estado inconsistente de la BD y por lo tanto, ha realizado un *análisis inconsistente*.**

|  |  |  |
| --- | --- | --- |
| Transacción A | Tiempo | Transacción B |
| Recuperar |  | - |
| Recuperar |  | - |
| - |  | Recuperar |
| - |  | Actualizar |
| - |  | Recuperar |
| - |  | Actualizar |
| - |  | COMMIT |
| Recuperar  (**y no** ) |  | - |

**Bloqueo**

Es una *técnica de control de concurrencia*. La idea básica es simple: cuando una transacción debe asegurarse de que algún objeto en el que está interesada no cambiará de ninguna forma mientras lo esté usando, adquiere un bloqueo sobre ese objeto. El efecto del bloqueo es *“inhibir todas las demás transacciones” en ese objeto y, por lo tanto, impedir que lo cambien* (estado estable).

Suponemos entonces que el sistema soporta dos tipos de bloqueos, ***bloqueo exclusivo*** *o* ***de escritura (X)*** y ***bloqueo compartido*** *o* ***de lectura (S)***.

Si la transacción A pone un bloqueo exclusivo (X) sobre la tupla , entonces se rechazará una petición de cualquier otra transacción B para un *bloqueo de cualquier tipo* sobre la tupla t.

Si la transacción A pone un bloqueo compartido sobre la tupla entonces:

* Se rechazará una petición de cualquier otra transacción B para un bloqueo exclusivo (X) sobre .
* Se otorgará una petición de cualquier otra transacción B para un bloqueo compartido (S) sobre .

**Bloqueo Mortal**

Es una situación en la que dos o más transacciones se encuentran en estados simultáneos de espera, cada una de ellas esperando que alguna de las demás libere un bloqueo para poder continuar. Como la espera entre las transacciones es mutua (se esperan entre sí) se torna infinita.

Si ocurre un bloqueo mortal es preferible que el sistema lo detecte y lo rompa. La detección implica la detección de un ciclo en el *grafo de espera*. La ruptura implica seleccionar una de las transacciones bloqueadas mortalmente como víctima y entonces deshacerla liberando por lo tanto sus bloqueos y permitiendo que continúen las demás transacciones (no todos los sistemas *detectan* los bloqueos mortales, sino que algunos usan un mecanismo de tiempo y asumen simplemente que una transacción que no ha realizado algún trabajo durante cierto periodo preestablecido, está bloqueada mortalmente).

Algunos sistemas volverán a iniciar esta transacción desde el principio, bajo la suposición de que las condiciones que causaron el bloqueo mortal probablemente ya no existirán. Otros sistemas simplemente regresan un código de excepción "víctima de bloqueo mortal" hacia la aplicación, y es asunto del programa manejar la situación de forma adecuada (no se recomienda).

**Seriabilidad**

Un conjunto de transacciones es seriable si el resultado de su ejecución concurrente es igual al resultado de su ejecución en serie. Se cumple mediante el protocolo de bloqueo en dos fases, el cual consiste en que una transacción debe adquirir un bloqueo sobre el objeto antes de utilizarlo y después debe liberar ese objeto.

**Niveles de aislamiento**

Es el *grado de interferencia* que una transacción dada es capaz de tolerar por parte de las transacciones concurrentes.

Si queremos garantizar la seriabilidad, no podemos aceptar ninguna cantidad de interferencia. Entre mayor sea el nivel de aislamiento, menor será la interferencia (y la concurrencia); y entre más bajo sea el nivel de aislamiento, mayor será la interferencia (y la concurrencia).

**Bloqueo por aproximación**

Si una transacción está solicitando un bloqueo (X) sobre una tupla, se dice que trabaja con una granularidad fina. Si, en cambio, la transacción pide bloqueo sobre un conjunto grande de tuplas o una tabla, nos encontramos frente a una granularidad gruesa. Cuanto más fina sea la granularidad, mayor cantidad de procesos concurrentes pueden ejecutarse.

Supongamos que una transacción solicita un bloqueo sobre una tupla de una tabla, y luego otra solicita un bloqueo sobre esa tabla completa. El gestor tendría que recorrer internamente las tablas, para ver si alguna tupla de la tabla se encuentra ya tomada. Esto claramente supone un trabajo excesivo para el gestor. En su lugar, es posible trabajar con el ***protocolo de bloqueos por aproximación***, de acuerdo con el cual ninguna transacción puede adquirir un bloqueo sobre una tupla sin antes poseer un bloqueo por aproximación sobre la tabla que lo contiene. Aparecen entonces tres nuevos estados de bloqueo, que se agregan a los vistos anteriormente (S y X). Los mismos son:

IS (intento o aproximación de bloqueo compartido): pretende poner bloqueos S sobre tuplas individuales de la tabla para garantizar su estabilidad mientras están en proceso.

IX (intento o aproximación de bloqueo exclusivo): igual que IS, pero además A puede actualizar tuplas individuales en la tabla y por lo tanto pondrá bloqueos X sobre esas tuplas.

SIX (combinación de las anteriores - intento o aproximación de bloqueo compartido exclusivo): puede tolerar lectores concurrentes, pero no actualizadores concurrentes en la tabla. Además, A puede actualizar las tuplas individuales en la tabla y por lo tanto pondrá bloqueos X sobre esas tuplas.

Protocolo de bloqueo por aproximación:

1. Antes de que una transacción dada pueda adquirir un bloqueo S sobre un tupla dada, primero debe adquirir un bloqueo IS o uno más fuerte sobre la tabla que contiene a esa tupla.
2. Antes de que una transacción dada pueda adquirir un bloqueo X sobre un tupla dada, primero debe adquirir un bloqueo IX o uno más fuerte sobre la tabla que contiene a esa tupla.

**Integridad**

El término integridad se refiere a la exactitud o corrección de los datos en la BD. Una BD determinada podría estar sujeta a cualquier cantidad de restricciones de integridad de una complejidad arbitraria.

Restricciones de Tipo

Es una sola enumeración de los valores válidos del tipo. Ejemplo, la restricción del tipo PESO: limitar los pesos de tal forma que puedan ser representados mediante un número racional que sea mayor que cero.

Restricciones de Atributo

Es sólo una declaración para que un atributo especificado sea de un tipo en particular.

Las restricciones de atributo son parte de la definición del atributo en cuestión y pueden ser identificadas por medio del nombre de atributo correspondiente. De aquí que una restricción de atributo sólo pueda ser eliminada mediante la eliminación del propio atributo.

Restricciones de Tablas (Variables Relacionales)

Es la que es impuesta a una tabla individual. Siempre son verificadas de inmediato. Por lo tanto, cualquier instrucción que intente asignar un valor a una tabla dada que viole cualquier restricción para esa tabla, será en efecto rechazada.

Ejemplo: se puede crear una restricción para que la ciudad del proveedor no pueda ser “Londres”.

Restricciones de BD

Es aquella que relaciona dos o más tablas distintas dentro de la misma BD. La verificación de restricciones de BD no puede hacerse de inmediato, sino que debe diferir hasta el final de la transacción, es decir, al momento del COMMIT. Si se viola una restricción de BD al momento del COMMIT, la transacción deshecha.

Restricciones de Estado frente a restricciones de transición

* Las *restricciones de estado* son aquellas que se ocupan de los estados correctos de la BD.
* Las *restricciones de transición* determinan que transiciones de un estado a otro son válidas.

**Claves**

Claves candidatas

Sea un conjunto de atributos de la tabla . Entonces es una clave candidata de si, y solamente si, posee las dos propiedades siguientes:

* *Unicidad:* jamás, ningún valor válido de contiene dos tuplas distintas con el mismo valor de .
* *Irreductibilidad:* ningún subconjunto propio de tiene la propiedad de unicidad.

Toda tabla tiene por lo menos una clave candidata. Además, las claves candidatas ofrecen el ***mecanismo de direccionamiento en el nivel de tupla*** básico dentro del modelo relacional, lo que significa que la única forma garantizada por el sistema para señalar una tupla específica es por medio de algún valor de clave candidata.

Claves primarias y claves alternas

Cuando una tabla tiene más de una clave candidata, el modelo relacional requiere que sólo una de esas claves se elija como ***clave primaria****.* A las otras se las llama entonces ***claves alternas.***

Claves externas (o foránea)

Una clave externa representa una referenciaa la tupla que contiene el valor de clave candidata coincidente.

Una clave externa es un conjunto de atributos de una tabla cuyos valores tienen que coincidir con los valores de cierta clave candidata para cierta tabla .

Sea una tabla. Entonces, una clave externa en es un conjunto de atributos de (los llamaremos ) tal que:

1. Existe una tabla con una clave candidata o primaria
2. En todo momento, cada valor de en el valor actual de debe ser idéntico al valor de en *alguna* tupla del valor actual de .

*Características de las claves externas:*

* La definición requiere que cada valor de una clave externa dada aparezca como un valor de la clave candidata correspondiente. Sin embargo, observe que lo opuesto no es obligatorio, es decir, la clave candidata correspondiente a una determinada clave externa puede contener un valor que no aparece actualmente como un valor de esa clave externa.
* Una clave externa es *simple* o *compuesta* dependiendo del hecho de que la clave candidata con la que coincide sea simple o compuesta.
* Cada atributo de una clave externa dada debe tener el mismo nombre y tipo que los componentes correspondientes de la clave candidata con la que coincide.

**Vistas**

Crea una tabla virtual cuyo contenido se define mediante una consulta. Se puede utilizar para lo siguiente:

* Para centrar, simplificar y personalizar la percepción de la BD para cada usuario.
* Como mecanismo de seguridad: los usuarios tienen permiso de acceder directamente a las tablas base.
* Para proporcionar una interfaz compatible.

**Procedimientos almacenados**

Es un grupo de una o varias transacciones, los cuales pueden:

* Aceptar parámetros de entrada y devolver parámetros de salida.
* Contener instrucciones de programación que realicen operaciones en la BD y llamadas a otros procedimientos.
* Devolver un valor de estado a un programa que realiza una llamada para indicar si la operación se ha realizado correctamente o se han producido errores.

Ventajas:

* Tráfico de red reducido entre el cliente y el servidor
* Mayor seguridad: controla qué procesos y actividades se llevan a cabo y protege los objetos de BD subyacentes. Esto elimina la necesidad de conceder permisos en cada nivel de objetos y simplifica los niveles de seguridad.
* Reutilización del código
* Mantenimiento más sencillo
* Rendimiento mejorado: de forma predeterminada, un procedimiento se compila la primera vez que se ejecuta y crea un plan de ejecución que vuelve a usarse en posteriores ejecuciones. Como el procesador de consultas no tiene que crear un nuevo plan, normalmente necesita menos tiempo para procesar el procedimiento.

**Procedimientos activados (triggers)**

Son procedimientos *definidos por el usuario*. Un procedimiento activado es un procedimiento que es invocado "automáticamente" al ocurrir algún evento o condición de activación. Esta condición es típicamente la ejecución de alguna operación de actualización de la BD, pero podría ser, por ejemplo, la ocurrencia de una excepción especificada o el transcurso de un intervalo de tiempo especificado.

**Check**

Es una restricción para especificar los valores de datos aceptables en una o más columnas.