Unidad 1 – Conceptos Básicos

Definición de Base de Datos:

- Colección de datos interrelacionados
- Colección de archivos diseñados para servir a múltiples aplicaciones
- Repositorio de información (puede ser un conjunto de archivos)

Instancias y esquemas

Instancia: la colección de información almacenada en la BD en un instante (en un momento determinado). (Equivalente al concepto de r-valor de una variable).

Esquemas: es el diseño global de la BD. (Equivalente al concepto de tipo de dato).

Independencia de Datos

Es la **capacidad de modificar la definición de un esquema sin afectar la definición de un esquema en otro nivel**. Dos niveles: ***independencia física:*** *cuando lo que se modifica es el esquema físico, sin que se vuelvan a escribir los programas de aplicación;* ***y lógica de datos:*** *cuando lo que se modifica es el esquema lógico.* Es más difícil de lograr la segunda por la dependencia de los programas a la estructura lógica de los datos a la que acceden.

Componentes

Un sistema de base de datos esta diseñado para gestionar grandes bloques de información. Para esto necesita la definición de estructuras para el almacenamiento de información, y mecanismos para la gestión de información.

DBMS

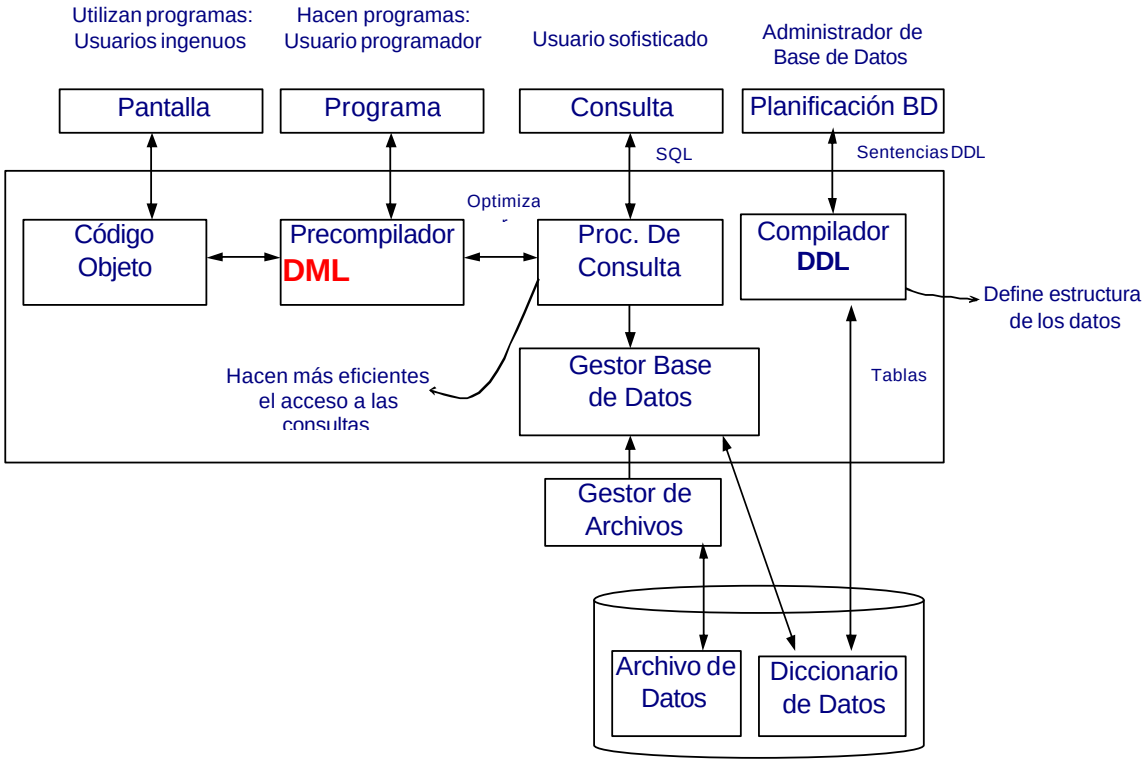
- Colección de datos y programas para acceder a ellos.
- Un DBMS consiste de un conjunto de datos interrelacionados y un conjunto de programas para acceder a estos datos, también debe proveer seguridad sobre los datos almacenados
- Colección de programas que permiten crear y mantener una base de datos.

Objetivos de un DBMS

Proporcionar un entorno que sea conveniente y eficiente para ser usado al extraer y almacenar información en la BD. Cuando no se cuenta con un DBMS se deben manejar los datos manualmente lo que lleva a que los sistemas puedan llegar a tener los siguientes inconvenientes: (el obj. es evitarlos)

1. **Redundancia e inconsistencia de los datos:** como los módulos se programan de manera independiente, puede llegar a pasar que los datos que se mantienen sean redundantes o sea que estén repetidos, por lo cual se ocupa mas lugar y peor aun puede pasar que sean inconsistentes (no concuerden entre sí).
2. **Dificultad de acceso a los datos:** Cada vez que se desea hacer una consulta hay que hacer un nuevo programa para satisfacer el pedido.
3. **Aislamiento de los datos:** Como los datos están repetidos en distintos archivos y con distintos formatos, es difícil escribir programas que accedan a los mismos.
4. **Anomalías en el acceso concurrente:** en los sistemas que permiten el acceso concurrente a las bases de datos, hay que proveer mecanismos que controlen el acceso compartido.
5. **Problemas de seguridad:** como no todos los usuarios no pueden acceder a todos los datos en el DBMS se deben proveer mecanismos para controlar el acceso a los datos.
6. **Problemas de integridad:** Los valores almacenados en la BD deben satisfacer ciertas propiedades (restricciones de consistencia). El DBMS debe hacer que se cumplan estas restricciones.

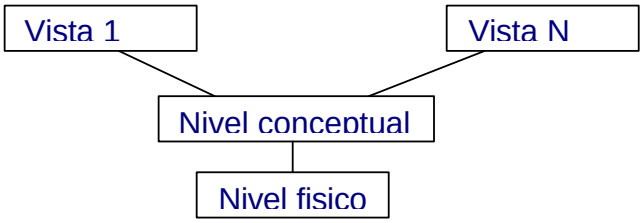
Esquema de un DBMS



Abstracciones

Uno de los objetivos importantes del DBMS es proveer una visión abstracta de los datos, y además debe permitir acceder a los datos de manera eficiente.
Para ocultar la complejidad de estos sistemas se dividen en niveles:

- **Nivel Físico:** establece como se almacenan los datos realmente. Esto de logra con estructuras complejas. (archivos, hard).
- **Nivel Conceptual:** que datos son almacenados en la BD y loas relaciones entre ellos. Se describe la BD completa a través de estructuras relativamente sencillas. También se indican las relaciones entre los datos almacenados. Este nivel lo usan los administradores de las BD que deben decir que información se guarda en la BD (datos y relaciones). entre ellos).
- **Nivel de Visión:** Algunos usuarios necesitan ver solo parte de las BD. El sistema. puede proveer muchas visiones para la misma BD. Describe parte de la BD completa. Genera un nivel de seguridad porque cada uno ve lo que necesita.



Categorías de soft de procesamiento de datos:

- **Sin independencia de datos (SO)**➔ los programas de aplicación interactuan con el disco rígido.
- **Independencia física** (leer un registro de un archivo S.O)➔ los programas interactuan con el SO y el SO con el disco rígido.
- **Independencia lógica parcial** (leer siguiente registro del archivo)➔permite recuperar un dato y automáticamente el siguiente.
- **Independencia lógica** (puedo leer el siguiente dato y este es correcto) **y física**(los programas interactuan con el DBMS que es el que manipula la información) (leer siguiente registro de un tipo particular, DBMS)
- **Independencia geográfica** (Bases de Datos Distribuidas)

Componentes del DBMS

1) Lenguaje de Definición de Datos (DDL)

Un esquema de BD se especifica por medio de un conjunto de definiciones que se expresan mediante un lenguaje especial llamado lenguaje de definición de datos (DDL). El resultado de la compilación de sentencias de DDL es un conjunto de tablas que se almacenan en un archivo especial llamado diccionario de datos. Este contiene metadatos, es decir datos sobre datos. Este archivo se consulta antes de leer o modificar los datos reales en el sistema de BD.

La estructura de almacenamiento y métodos de acceso usados por los sistemas de BD se especifican mediante un conjunto de definiciones de un tipo especial de DDL (lenguaje de almacenamiento y definición de datos).

2) Lenguaje de Manipulación de Datos (DML)

Es un lenguaje que capacita a los usuarios a acceder o manipular datos según estén organizados por el modelo de datos adecuado.

Objetivo➔ proporcionar una interacción eficiente entre personas y el sistema.

Manipulación: recuperar información (almacenamiento en la BD), agregar información nueva en la BD, quitar información almacenada en la BD, modificar información almacenada en la BD.

Existen dos tipos:

- **Procedurales**: requieren que el usuario especifique que datos se necesitan y como obtenerlos. QUE Y COMO. **SQL**
- **No Procedurales**: requieren que el usuario especifique que datos se necesitan y no como obtenerlos. QUE solamente. **QVE**

Lenguaje de consulta➔ trozo de DML que implica recuperación de información.

3) Gestor de Base de datos

Debe organizar los datos de manera de minimizar el acceso a memoria secundaria.

Módulo de programa que provee una interfaz entre los datos de bajo nivel de la base de datos y los programas de aplicación y consulta.

- Debe realizar:
  - Interacción con el gestor de archivos: debe traducir las sentencias DML a comandos del sistema de archivos de bajo nivel. Es el responsable del verdadero almacenamiento, recuperación y actualización de los datos de la BD.
  - Implantación de la integridad: debe especificar las condiciones de consistencia, determinando si las actualizaciones a la BD dan como resultado alguna violación a las restricciones.
  - Implantación de la seguridad: hace que se cumplan los requisitos de seguridad.
  - Copia de seguridad y recuperación: debe detectar fallos y recuperarse de los mismos (bitácora, backup) dejando la BD en el estado que tenia antes del fallo, iniciando un proceso de copia de seguridad y recuperarlo.
  - Control de concurrencia: controlando la interacción entre los usuarios concurrentes (bloqueos, etc.)

Administrador de Base de Datos

Es la persona que tiene el control central de BD y de los programas para acceder a los datos.

- Funciones:
  - **Definición de esquema***:* el esquema original de la BD se crea escribiendo un conjunto de definiciones que son traducidas por el compilador de DDL a un conjunto de tablas que son almacenadas en el diccionario de datos.
  - **Definición de la estructura de almacenamiento y del método de acceso**: índices, etc.
  - **Modificación del esquema y de la organización física** escribiendo definiciones usadas por el compilador de DDL o entre el lenguaje de almacenamiento y definición de datos.
  - **Autorización para acceder a los datos***:* regula que partes de los datos van a poder ser accedidas por que usuarios.
  - ***Especificación de las restricciones de integridad*:** se mantienen en una estructura especial en un archivo que el gestor consulta cuando necesita hacer modificaciones.

Usuarios de Bases de Datos

Se diferencian por la forma de interaccionar con el sistema.

- **Programadores de aplicaciones**: interactúan con el sistema por medio de llamadas en DML. El precompilador DML, convierte las sentencias en DML a llamadas normales a procedimientos en el lenguaje principal.
- **Usuarios sofisticados:** interaccionan sin escribir programas, lo hacen a través de preguntas en un lenguaje de consultas de BD (cada uno se somete a un procesador de consultas que toma una sentencia DML y la descompone en instrucciones que entiende el gestor de BD).
- **Usuarios especializados:** realizan operaciones no tradicionales (almacenamiento de datos complejos, sistema de modelacion de entorno).
- **Usuarios ingenuos:** interactúan mediante los programas de aplicación ya escritos.

Estructura del Sistema Global

El SO del computador proporciona únicamente los servicios mas básicos y el sistema de BD debe partir de esa base. Los componentes funcionales de un Sistema de BD incluyen:

- **Lógico**:
  - **Gestor de archivos:** gestiona la asignación de espacio en la memoria del disco y de las estructuras de datos usadas para representar información almacenada en disco.
  - **Gestor de BD:** interfaz entre los datos de bajo nivel almacenado en la BD y los programas de aplicación y las consultas.
  - **Procesador de consultas**: traduce sentencia en un lenguaje de consulta a instrucciones de bajo nivel que entiende el gestor de BD. Además optimiza las consultas.
  - **Precompilador DML**: convierte las sentencias en DML incorporadas en un programa de aplicación en llamadas normales a procedimientos en el lenguaje principal
  - **Compilador DDL**: convierte sentencias en DDL en un conjunto de tablas que contienen metadatos.
- **Físico**:
  - **Archivo de datos**: almacenan la BD
  - **Diccionario de datos**: almacenan metadatos sobre la estructura de la BD
  - **Índices**: proporcionan acceso rápido a los elementos de datos.

Modelo de Datos

Colección de herramientas conceptuales para describir datos, relaciones entre ellos, semántica asociada a los datos y restricciones de consistencia.

Los modelos de datos se dividen en tres grupos:

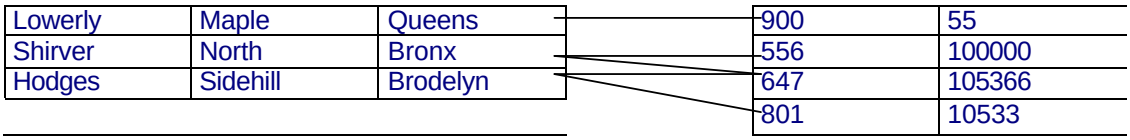
- **Basado en objetos**: describe datos en los niveles conceptual y de visión. Proporcionan capacidad de estructuración flexible y permiten especificar restricciones de datos explícitamente. Ej: Modelo ***Entidad Relación***, Modelo orientado a objetos, Modelo binario infológico, Modelo semántico de datos.

- **Basado en registros**: describe datos en los niveles conceptual y físico. Se usan para especificar la estructura lógica global de la BD y para proporcionar una descripción a nivel más alto de la implementación.
  Lenguaje asociado para expresar consultas.
  La BD está estructurada en registros de formato fijo de varios tipos. Cada tipo de registro define un número fijo de campos o atributos, y cada campo, normalmente, es de longitud fija. Ej: ***Modelo Relacional***, Modelo de red, Modelo Jerárquico.
  Los tres modelos mas ampliamente aceptados:
  Modelo relacional: los datos y las relaciones entre ellos son representados mediante una colección de tablas formadas por columnas.

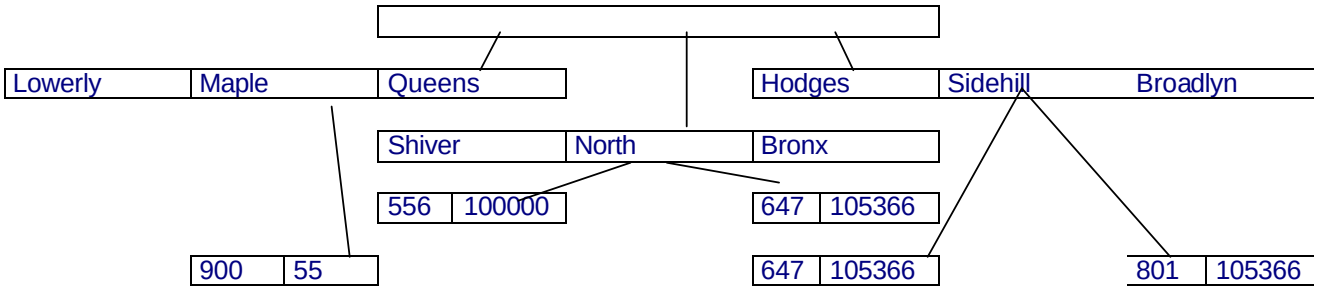
Nombre	Calle	ciudad	nro.
Lowerly	Maple	Queens	900
Shirver	North	Bronx	556
Shiver	North	Bronx	647
Hodges	Sidehill	Brodelyn	801
Hodges	sidehill	Brodelyn	647

nro.	Saldo
900	55
556	100000
647	105366
801	10533

Modelo de red: los datos se representan mediante colecciones de registros y las relaciones entre los datos se representan mediante enlaces, los cuales pueden verse como punteros. Los registros en las BD se organizan como colecciones de grafos arbitrarios.



Modelo jerárquico: similar al modelo de red en el sentido de que los datos y las relaciones entre los datos se representan mediante registros y enlaces, respectivamente. Se diferencia del modelo de red en que los registros están organizados como colecciones de árboles en vez de grafos arbitrarios.



- **Físico de datos**: describen datos en el nivel más bajo (físico). Ej: Modelo unificador, Memoria de elementos.

Unidad 2 – Modelo Entidad-Relación

Fases de diseño

El diseño de una BD se descompone en tres etapas *conceptual, lógico, físico*.
Diseño conceptual: se parte de una especificación y se obtiene un esquema conceptual. El esquema conceptual es una descripción de alto nivel de la estructura de la BD, para dicha descripción se utiliza un lenguaje llamado **modelo conceptual (modelo primario de los datos, genérico. No importa el DBMS practico ni conceptual).** El objetivo es describir el contenido de la información de la BD.
Diseño lógico: Se parte de un esquema conceptual y se obtiene un **esquema lógico.** Un esquema lógico es una descripción de la estructura de la BD que puede ser procesada por un DBMS. Para describir un esquema lógico se utiliza un modelo lógico. El modelo lógico no depende del DBMS particular sino de la clase de modelo de datos usados por el DBMS.
Diseño físico: parte del esquema lógico y da como resultado el **esquema físico**, el cual es una descripción de la implementación de la BD en memoria secundaria, describe las estructuras de almacenamiento y la forma de acceso. Este diseño depende del DBMS. Puede ser que durante esta etapa de deba modificar el esquema lógico.

Dependencias

	tipo de DBMS (DBMS conceptual)	**DBMS específico(practico)**
conceptual	NO	NO
lógico	SI	NO
físico	SI	SI

Diseño conceptual

Mecanismos de abstracción en el diseño conceptual

La abstracción es un proceso mental que permite eliminar características de un objeto que no son relevantes, en cierta aplicación.

Abstracción por clasificación (es_miembro_de)➔ La clasificación se usa para definir clases de objetos que tiene características en común

Mesa silla mueble blanco mueble negro

Silla blanca silla negra

Abstracción por agregación (es_parte_de)➔La agregación permite definir una nueva clase a partir de clases definidas que representan partes componentes de la nueva clase.

Persona Nombre sexo
H m

Abstracción por generalización (es_un)➔La generalización define la relación de subconjunto entre una o mas clases.

Persona
Hombre mujer

Coberturas en generalizaciones

1. ***Cobertura total o parcial*:** es total si cada elemento de la clase genérica corresponde al menos a un elemento de las clases subconjunto, es parcial si algún elemento de la clase genérica no corresponde a ningún elemento de las clases subconjunto.
2. ***Cobertura exclusiva o superpuesta*:** es exclusiva si cada elemento de la clase genérica corresponde a lo sumo a un elemento de la clase subconjunto; es superpuesta si pertenece a mas de una clase subconjunto.

Modelo de datos

Conceptos que pueden utilizarse para describir conjuntos de datos yoperaciones para manipularlos.
Hay dos tipos de **modelos de datos**:

Conceptuales:(usados en el diseño de BD) se hace a un nivel de abstracción alto.
Lógico: (apoyados por los DBMS)tienen una correspondencia sencilla con la implementación física (Ej: de redes, relacional, jerárquico).

Cualidades del modelo

- Expresividad : rico en conceptos, representa los requerimientos de manera natural y se puede entender con facilidad.
- Simplicidad (fáciles de entender)
- Minimalidad (c/concepto tiene significado distinto), representa cada concepto una sola vez en el diagrama, sin ambigüedad.
- Formalidad (c/concepto tiene una sola interpretación)

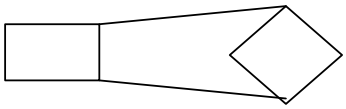
No siempre es posible satisfacer todas las propiedades a la vez.
El modelo de datos **Entidad Relación (ER)** se basa en una percepción de un mundo real que consiste en un conjunto de objetos básicos (entidades) y relaciones entre esos objetos y ambos pueden tener atributos.
Representa la estructura lógica global de la BD.
Se presentan los datos y las relaciones entre los datos mediante una colección de tablas, cada una de las cuales tiene un número de columnas con nombres únicos.

Elementos del modelo

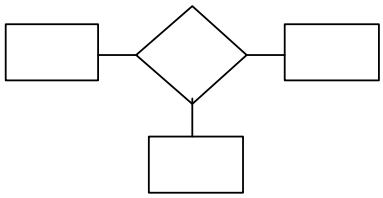
Entidad➔ Las **entidades** representan clases de objetos de la realidad que existen y son distinguibles de otros objetos, como por ej: Persona, Hombre, Mujer, Ciudad, etc. Está representada por un conjunto de atributos, los cuales definen la entidad.
Una entidad puede ser ***concreta***, tal como una persona o un libro o puede ser ***abstracta***, como un día festivo o un concepto.
Un ***conjunto de entidades*** es un conjunto de entidades del mismo tipo. El conjunto de todas las personas que tienen una cuenta en un banco, por ej., puede definirse como el conjunto de entidades cliente.
Los conjuntos de entidades no necesitan ser disjuntos, es decir, distintos conjuntos de entidades pueden tener elementos comunes.

Relaciones➔ Las relaciones son asociaciones entre dos o más entidades. Por ejemplo, podemos definir una relación que asocia al cliente Juan con la cuenta 401. Esto significa que Juan es un cliente con número de cuenta bancaria 401.
Un conjunto de relaciones es un conjunto de relaciones del mismo tipo.
Cuando la relación involucra sólo dos entidades se denomina ***relación binaria*** y siempre es posible sustituir un conjunto de relaciones no binarias (***relaciones n-arias***) por varios conjuntos de relaciones binarias distintas.
Una relación también puede tener atributos descriptivos Ej. fecha.
Los ***anillos*** son relaciones binarias que conectan una entidad consigo misma. Se conocen también como interrelaciones ***recursivas.***
Cada relación tiene un significado específico: de ahí la necesidad de seleccionar ***nombres significativos*** para ellas.
Las relaciones se caracterizan en términos de ***cardinalidad mínima y máxima***.

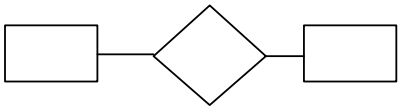
Recursiva



terciaria



Binaria



Atributos➔ Son los elementos que definen o representan a una entidad. Un atributo es una ***función*** que asigna un conjunto de entidades a un dominio. Cada entidad se describe por medio de un conjunto de pares (atributo, valor del dato), un par para cada atributo del conjunto de entidades.
El ***dominio*** es el conjunto de valores permitidos para cada atributo.
Los atributos tienen cardinalidad max y min si la cardinalidad min es 0 entonces el atributo es opcional, si es >0 entonces es obligatorio; si max es =1 entonces es monovalente y si max es >1 es polivalente.

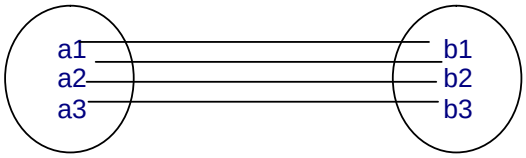
Restricciones de asignación (mapping)

La planificación de una ER puede definir ciertas restricciones a las cuales deben ajustarse los contenidos de una base de datos. Una de ellas es la cardinalidad de asignación, que es el número de entidades con las que puede asociarse otra entidad mediante un conjunto de relaciones.

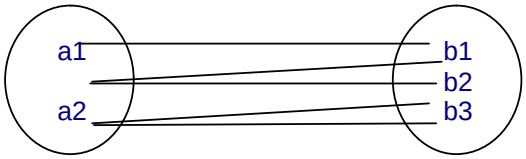
Cardinalidad

La cardinalidad de asignación es útil cuando describe conjuntos de relaciones:
Dadas A y B entidades y R relación, la cardinalidad puede ser así:

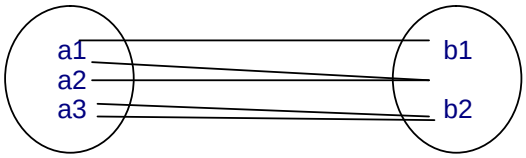
- Una a una: una entidad en A está asociada a los sumo con una entidad en B, y una entidad en B está asociada a lo sumo con una entidad en A.



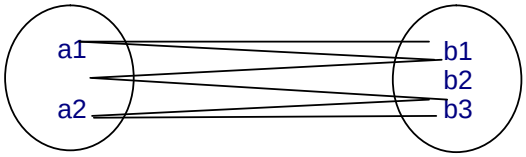
- Una a muchas: una entidad en A está asociada con un número cualquiera de entidades en B. Una entidad en B puede estar asociada a lo sumo con una entidad en A.



- Muchas a una: una entidad en A está asociada a los sumo con una entidad en B. Una entidad en B puede estar asociada con un número cualquiera de entidades en A.



- Muchas a Muchas: una entidad en A está asociada con un número cualquiera de una entidad en B. Una entidad en B puede estar asociada con un número cualquiera de entidades en A.



Al igual que la cardinalidad, las dependencias de existencia constituyen otra clase importante de restricciones. Si una entidad **x** depende de la existencia de un entidad **y**, entonces se dice que es dependiente por existencia de **y**. Esto significa que si se suprime **y**, también se suprime **x**. La entidad **y** se dice que es una ***entidad dominante*** y **x** se dice que es una ***entidad subordinada***. Ej: cuentas y transacciones de esas cuentas.

Claves

Es importante poder especificar como se distinguen las entidades y las relaciones, la diferencia entre ellas debe expresarse en términos de sus atributos.
Una **superclave** es un conjunto de uno o más atributos que considerados juntos permiten identificar en forma unívoca a una entidad en el conjunto de entidades. Si K es una superclave, entonces también lo será cualquier superconjunto de K.
Se denomina **clave candidata**, a aquella superclave para cuales ningún subconjunto propio es superclave, es decir son superclaves mínimas. Supóngase que una combinación *de nombre_cliente* y *calle* es suficiente para distinguir entre los miembros del conjunto de entidades *cliente*. Entonces *{seguridad_social}* y *{nombre_cliente, calle}* son claves candidatas. Aunque los atributos *seguridad_social* y *nombre_cliente* juntos pueden distinguir entidades *cliente*, sus combinaciones no forman una clave candidata, ya que el atributo *seguridad_social* por sí solo es una clave candidata.

Denominamos **clave primaria** a aquella clave candidata que se elige para identificar entidades dentro de un conjunto de entidades y una relación en un conjunto de relaciones.
Entidad débil➔ entidades que no tiene atributos suficientes para formar una clave primaria
Entidad fuerte➔ entidad que posee atributos para formar una clave primaria.
Considérese el conjunto de entidades transacción que tiene 3 atributos: número de transacción, fecha y cuenta. Aunque cada entidad transacción es distinta, las transacciones en cuentas diferentes pueden compartir el mismo número de transacción. Así, este conjunto de entidades no tiene clave primaria y por lo tanto es un conjunto de entidades débil.
Para que un conjunto de entidades débil sea significativo, debe ser parte de un conjunto de relaciones una a muchas. Este conjunto de relaciones no debe tener atributos descriptivos, ya que cualquier atributo que se necesite puede estar asociado con el conjunto de entidades débil.
Un miembro de un conjunto de entidades fuerte es una entidad dominante, mientras que un miembro de un conjunto de entidades débil es una entidad subordinada.
El **identificador** de un conjunto de entidades débil es un conjunto de atributos que permite que se haga esta distinción. (Por ejemplo, el identificador del conjunto de entidades débil transacción es el atributo número de transacción, ya que para cada cuenta un número de transacción identifica de forma única una transacción).
La clave primaria de un conjunto de entidades débil está formada por la clave primaria del conjunto de entidades fuerte de la que dependen su existencia y su identificador (en el caso del conjunto de entidades transacción, su clave primaria es *{numero_cuenta, numero_transacción},* donde *numero_cuenta* identifica la entidad dominante de una transacción, y *numero_transacción* distingue entidades transacción dentro de la misma cuenta).
Sea R un conjunto de relaciones que implica a los conjuntos de entidades E1, E2,. . . .,En. Sea (Ei) la clave primaria que denota como el conjunto de atributos que forma la clave primaria para el conjunto de entidades Ei. Supóngase que los nombres de atributos de todas las claves primaria son únicos y que R no tiene atributos. Entonces los atributos que describen las relaciones individuales del conjunto R denotadas por el atributo (R), son:

clave_primaria(E1) U clave_primaria(E2) U .....U clave_primaria(En)

En el caso de que R tenga atributos descriptivos, digamos *{a1, a2, ..., an}* entonces el conjunto de atributos (R) consta de:

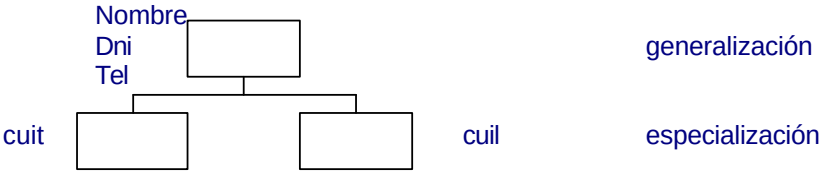
clave_primaria(E1) U clave_primaria(E2) U .....U clave_primaria(En) U *{a1,.....,an}*

La composición de la clave primaria depende de la cardinalidad de asignación y de la estructura de los atributos asociados con el conjunto de relaciones R.
Si el conjunto de relaciones R no tiene atributos asociados, entonces el conjunto atributo (R) forma una superclave. Esta superclave es una clave primaria si la cardinalidad de asignación es muchas a muchas.
Si R tiene varios atributos asociados con el, entonces una superclave está formada como antes, con la posible adición de uno o más de estos atributos.

Otros elementos de la ER

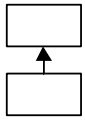
Jerarquías de generalización y especialización

Una entidad E es una generalización de $E_1..E_n$ si cada objeto de las clases $E_1..E_n$ es también un objeto de la clase E. Se representa mediante una flecha que apunta a la clase generalizada. Se debe indicar el tipo de cobertura mediante un par (T/P,E/S) (Total/Parcial,Exclusiva/Superpuesta)se deben poner los atributos en común de las entidades $E_1..E_n$ en la entidad E.



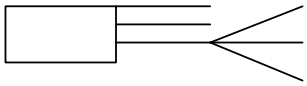
Subconjunto

Es un caso particular de generalización con una sola entidad subconjunto. La cobertura siempre es parcial exclusiva. Pueden tener atributos adicionales.



Atributos compuestos

Son atributos que están definidos en función de otros atributos y funcionan como un conjunto. No representa aportes al modelo .



Agregación

Es una abstracción a través de la cual las relaciones se tratan como entidades de nivel mas alto.

Ej: ((Empleado <trabaja> Proyecto) <usa> Maquinaria)

Mecanismos de abstracción en ER

La clasificación se puede convertir en una entidad, una relación o un atributo.

La agregación se puede convertir en una entidad, una relación o un atributo.

La generalización en general se convierte en jerarquía de entidades (aun también puede hacerlo con jerarquías de relaciones o entidades).

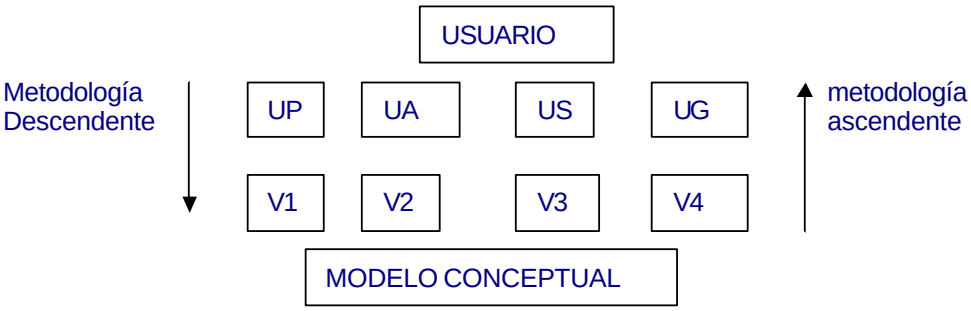
Críticas

A *favor*: Riqueza de conceptos, muy expresivo, diagramas fáciles de leer. Todo puede llevarse a binario con costo de procesamiento.

En *contra*: No es muy sencillo de usar, atenta contra la simplicidad y se complica con relaciones n-arias. No es sencillo de usar. Los problemas se resuelven de distintas formas → afecta la minimalidad.

Metodología de diseño conceptual

- Ascendente: se va de lo particular a lo general. Agregan conceptos nuevos.
- Descendente: se va de lo abstracto a lo concreto. Refinan conceptos.



Características de las transformaciones

1- Se parte de un esquema inicial y se obtiene uno final luego de aplicar la transformación.
2- Sigue habiendo correspondencia entre los datos anteriores y los nuevos.
3- El esquema nuevo debe mantener las mismas relaciones que tenia el original.

Primitivas descendentes

1- Un concepto único da como resultado un pequeño número de conceptos.
2- Los nombres se refinan y dan lugar a nuevos nombres que describen el concepto original en un nivel de abstracción mas bajo.
3- Las conexiones lógicas deben ser heredadas por un solo concepto del esquema resultante.

Primitivas ascendentes

Se van agregando nuevos conceptos que antes no existían.

1- nada → una entidad
2- dos entidades → dos entidades relacionadas
3- subconjunto de entidades → jerarquía
4- atributos sueltos → entidad con atributos
5- una entidad con atributos → una entidad con atributos compuestos

Diseño de vistas

El objetivo principal del diseño de vistas es crear un esquema conceptual partiendo de una descripción informal de los requerimientos del usuario. Se utiliza el término vista para referirse a la percepción de una base de datos o de los requerimientos de datos de una aplicación, tal como lo ve el usuario o un grupo de usuarios. El diseño de vistas abarca dos actividades:

1- **El *análisis de los requerimientos***, para captar los objetos de interés de la aplicación, su agrupación en clases, sus propiedades, etc. Los requerimientos pueden venir en:
   - ❑ Descripción en lenguaje natural – por escrito (pueden ser ambiguos, incompletos o contradictorios)
   - ❑ Formularios – de procedimientos existentes (pueden perder información)
   - ❑ Declaración de registros o formato de registros – de sist. de info. Existentes

2- ***Representación de los objetos***, clases y propiedades, usando los conceptos del modelo ER.

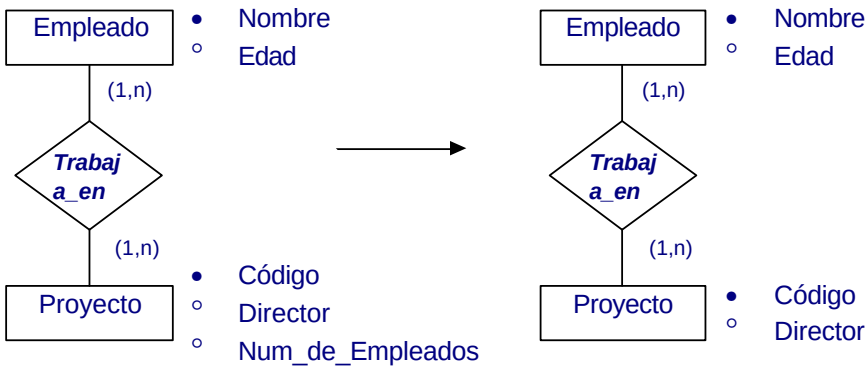
Validación del diagrama

Luego de armar un primer diagrama ER, puede ocurrir que debamos hacer una validación por si se puede escribir mejor. El modelo debe representar al mundo real (se logra respetando el análisis de requerimientos).
(Cualidades del esquema de BD)

- ❑ ***Complexión***: representa todas las características pertenecientes al dominio de aplicación (analisis de requerimientos).
- ❑ ***Corrección:*** un esquema es correcto cuando se usa con propiedad los conceptos del modelo ER. Dos tipos de corrección:
  - ✓ ***Sintáctica***: cuando los conceptos se definen con propiedad en el esquema. Cada concepto del modelo ER se usa correctamente.
  - ✓ ***Semántica***: cuando los conceptos (entidades, interrelaciones, etc) se usan de acuerdo con sus definiciones. Errores semánticos más frecuentes:
    - Usar un atributo en lugar de una entidad
    - Olvidar una generalización (o un subconjunto)
    - Olvidar la propiedad de herencia de las generalizaciones
    - Usar una interrelación con un número erróneo de entidades (ej: binaria en vez de ternaria)
    - Usar una entidad en lugar de una interrelación.
    - Olvidar algún identificador de una entidad, especialmente identificadores compuestos externos. Toda entidad necesita un identificador.
    - Omitir alguna especificación de cardinalidad mínima o máxima.

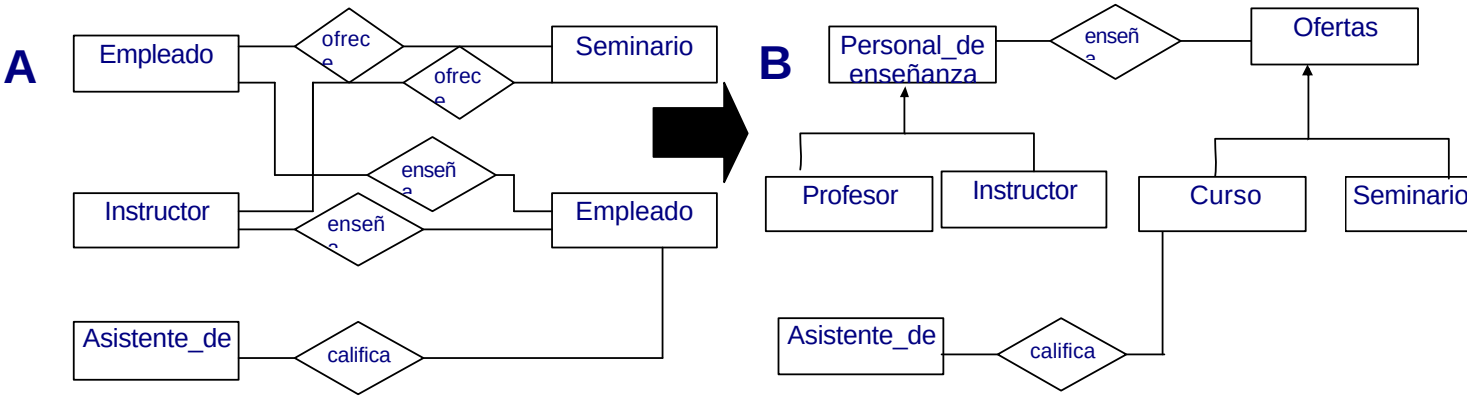
Herramientas para un buen modelo conceptua

- ❑ ***Minimalidad:*** un esquema es mínimo cuando cada aspecto de los requerimientos aparece sólo una vez en el esquema. Ej:



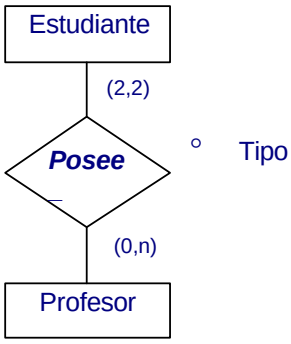
En este caso la cantidad de empleados se deriva simplemente contando los empleados relacionados con el proyecto, por lo tanto está de más el atributo Num_de_Empleados de la entidad Proyecto.
Es mínimo cuando no se puede borrar un concepto sin perder alguna info.

- ***Legibilidad:*** un diagrama tiene buena legibilidad cuando respeta gráficamente al esquema:
  - Diagramas en hoja cuadriculada.
  - Estructuras simétricas.
  - Minimizar número de cruces.
  - Generalización sobre los hijos.
- ***Expresividad:*** cuando representa los requerimientos de manera natural y se puede entender con facilidad. Ej:

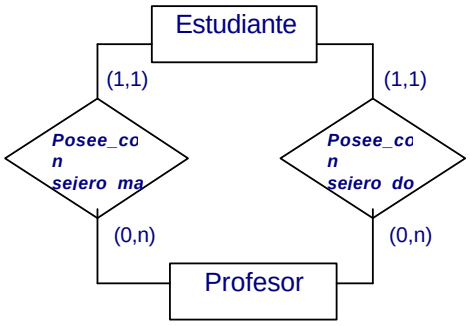


Aunque representan lo mismo, B es más expresivo que A.

- ***Autoexplicacion:*** Un esquema se explica a si mismo cuando puede representarse un gran número de propiedades usando el modelo conceptual por si mismo, sin otros formalismos. Ej:



Tipo



Este diagrama no es autoexplicativo pues no se que

Este diagrama sí se explica lo que representa. Es mejor, es

- ***Extensibilidad:*** un esquema se adapta fácilmente a requerimientos cambiantes cuando puede descomponerse en partes, a fin de aplicar los cambios dentro de cada parte. Debo tratar de que sea extensible para que lo pueda modificar si hay cambios. Aunque en la etapa de diseño no tendría porque haber modificaciones.

- ***Normalidad:*** las formas normales purifican anomalías de diseño. Existen 6 formas normales. Las formas normales pretenden mantener la estructura lógica de los datos en una forma normal limpia, "purificada", minimizando los problemas de las anomalías de inserción, borrado y actualización que ocasionan trabajo innecesario.

Transformación de esquemas

Las transformaciones de esquemas se aplican a un esquema de entrada S1 y producen un esquema resultante S2. Dos esquemas son equivalentes si pueden contestar las mismas consultas.
Se pueden clasificar en:

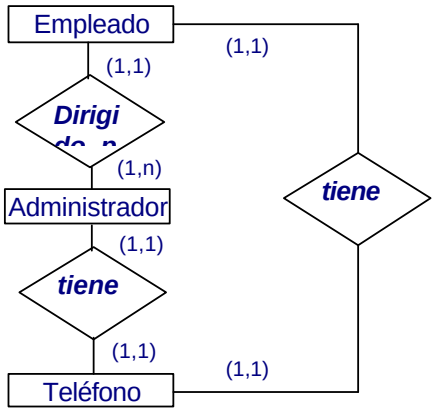
- ***Las que preservan la información***: el contenido de información del esquema no es alterado por la transformación.
- ***Las que cambian la información***:
  a) Transformaciones de aumento: el contenido de información del esquema resultante es mayor que el del esquema de entrada.
  b) Transformaciones de reducción: el contenido de información del esquema resultante es menor que el del esquema de entrada.

c) Transformaciones no comparables: en cualquier otro caso.

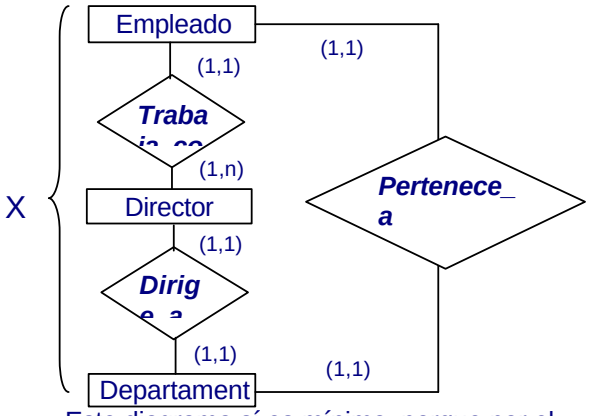
1. Transformaciones para lograr minimalidad

Depende del significado. La redundancia en los esquemas ER pueden surgir por varias razones:

- **Ciclos de interrelaciones**: la redundancia se da cuando una interrelación R1 entre dos entidades posee el mismo contenido de información que una ruta de interrelaciones R2, R3,....., Rn que conecta exactamente los mismos pares de casos de entidades que R1.



En este caso que el teléfono es único, lo puedo hallar por dos lados distintos. Como estoy repitiendo información: no es mínimo



Este diagrama sí es mínimo, porque por el camino X no se a que departamento pertenece el empleado

- **Atributos derivados**: la redundancia puede deberse a la existencia de un algoritmo para calcular los valores de datos derivados de los otros datos: es por eso que los datos derivados pueden omitirse de un esquema ER mínimo. Se deberá decidir la conveniencia de mantener o no los datos derivados.

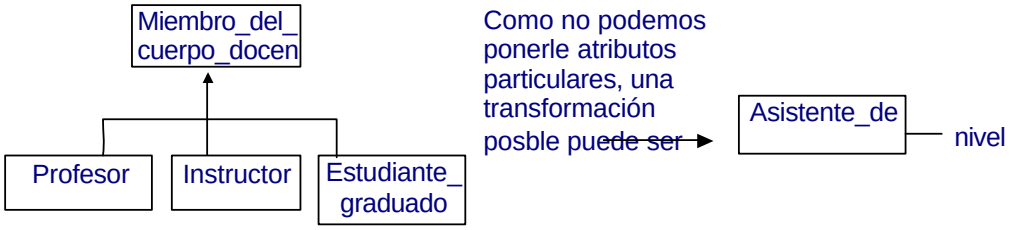
Ventajas: no se necesita calcular el valor, se reduce el número de accesos a la BD.
Desventajas: procesamiento adicional para mantener los datos derivados, mayor cantidad de almacenamiento.

Resumen: es tarea del diseñador decidir si acepta la redundancia en el esquema conceptual o la elimina. En cualquier caso, la redundancia puede ser fuente de anomalías en la administración de datos; para ello, debe estar claramente indicada en el esquema.

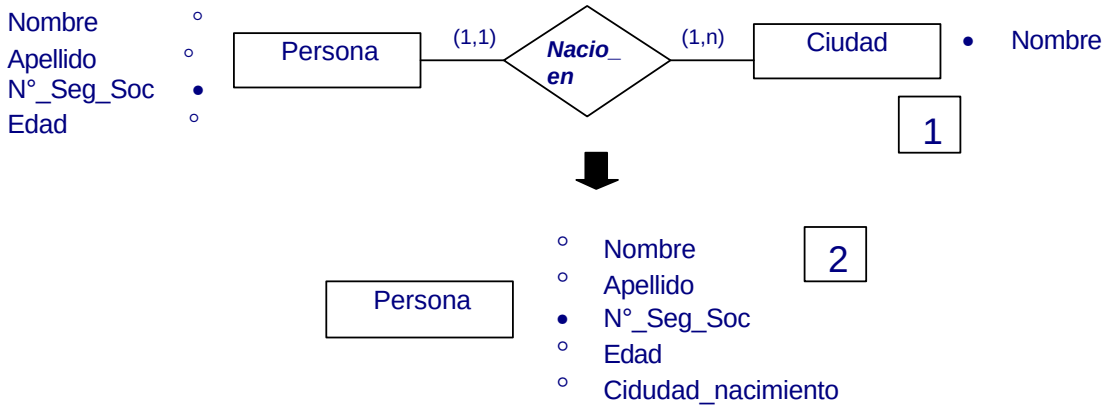
2. Transformaciones para expresividad y autoexplicación

La ***autoexplicación*** se logra cuando las propiedades de los datos se expresan usando solamente el propio modelo conceptual, en vez de anotaciones adicionales; la ***expresividad*** se realza *simplificando* el esquema.

- **Eliminación de subentidades colgantes en las jerarquías de generalización**: puede suceder que el diseñador cree una generalización en el proceso de asignar diferentes propiedades a entidades de la jerarquía. Si, al final del proceso de diseño, las subentidades no se distinguen por ninguna propiedad específica, pueden reducirse a la superentidad. La diferencia entre las distintas entidades se expresa entonces mediante un atributo. Ej:



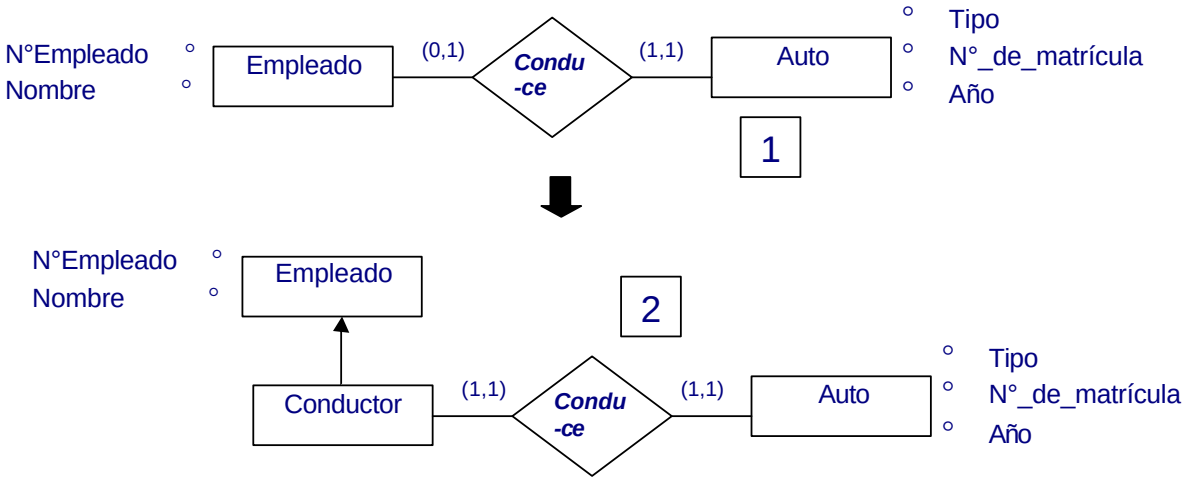
- **Eliminación de entidades colgantes**: Considérese colgante una entidad E si posee pocos (posiblemente uno) atributos Ai y una conexión con otra entidad (la entidad principal) a través de una relación R; en este caso, puede ser conveniente simplificar el esquema eliminando la entidad colgante y la interrelación de conexión, pasando los atributos Ai de la entidad colgante a la entidad principal. Las cardinalidades mínima y máxima de un atributo A nuevo (respecto de la entidad principal) pueden calcularse fácilmente como una combinación de las cardinalidades anteriores de la entidad colgante E en la interrelación R y del atributo original A dentro de la entidad. Por ejemplo, si la cardinalidad máxima de E es 1 y la de A es n, la cardinalidad máxima del nuevo atributo es n. Ej:



En este caso es difícil ver cual de los dos es mejor. Debe observarse si se utiliza ciudad para algo importante o si ciudad es un dato no muy relevante. En el primer caso, será más expresivo y autoexplicativo 1, en el segundo caso será mejor 2 (donde ciudad es un atributo, en este caso un string).

- **Creación de una generalización**: Esta transformación se aplica cuando se descubren dos entidades distintas con propiedades similares, que pertenecen en realidad a la misma jerarquía de generalización (tienen atributos iguales).

- **Creación de un nuevo subconjunto**: Esta transformación destaca el papel de una entidad. Se puede aplicar a entidades con cardinalidad mínima de cero en una interrelación; esto significa que la interrelación se aplica a un subconjunto de los casos de la entidad. Esta transformación debe aplicarse cada vez que semejante subconjunto posea una identidad clara y sea significativo para el diseño. Ej:



1 es más expresivo que 2, pues la entidad Empleado puede o no ser un conductor. Por eso en 2 queda mejor modelado al introducir un subtipo especial de los empleados llamado Conductor.

3. Transformaciones para normalidad (se ven en el diagrama lógico)

La **normalización** de relaciones es un proceso de aplicación de transformaciones progresivas para lograr la **forma normal** deseada y está guiado por las *dependencias funcionales*.

Dependencia funcional

Existe una **dependencia funcional (DF)** entre dos atributos monovalentes, A1, A2, de una entidad E o de una interrelación R, si cada valor de A1 corresponde exactamente a un valor de A2. Las dependencias funcionales son una restriccion al conjunto de relaciones legales, nos permiten expresar restricciones que no pueden expresarse por medio de superclaves

Sean A1, A2 dos atributos de una entidad E; supóngase que existe un caso de entidad e1 de E, que tiene los valores a1 para A1 y a2 para A2:

<e1:...a1...a2...>

La dependencia funcional entre A1 y A2 implica que si existe otro caso de entidad, e2, en el que A1 adopta el valor a1, entonces A2 debe adoptar el valor a2:

<e2: ...a1...a2...>

Decimos que A1 ***determina funcionalmente*** a A2, lo cual se denota también como A1 → A2; el atributo a la izquierda de la DF se llama el ***determinante***. Las DF se establecen de forma similar entre conjuntos de atributos. Cuando el lado derecho de una DF es un conjunto S de n atributos, la DF original es equivalente a n dependencias funcionales, cada una con un atributo sencillo de S como lado derecho. Las dependencia funcionales se establecen, así mismo, entre atributos de interrelaciones.

Todos los identificadores internos de las entidades determinan funcionalmente a los otros atributos monovalentes.

Las dependencias dicen que si se asigna el valor de los atributos determinantes, se encontrará en la base de datos un valor de los atributos determinados.

Dependencia funcional➔ entre atributo no clave y atributos claves.

Dependencia parcial➔ un atributo no clave depende de un subconjunto clave.

Anomalías de actualización

Si bien las dependencias funcionales que corresponden a identificadores no causan problemas, otras dependencias que pueden existir en una entidad pueden causar las llamadas anomalías de actualización. Más específicamente, se pueden producir anomalías de ***inserción***, de ***eliminación*** y de ***actualización***.

Todas estas anomalías se relacionan con la presencia de una dependencia funcional indeseable.

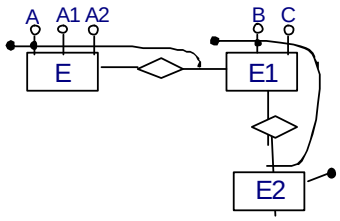
Atributos pertinentes de las entidades e interrelaciones

Los identificadores externos de una entidad proveen dependencias funcionales entre atributos de la entidad. De forma similar, los identificadores de las entidades proveen dependencias funcionales entre atributos de la interrelación.

Es necesario definir el conjunto de atributos **pertinentes** para el proceso de normalización:

- **Atributos pertinentes de una entidad:**

Considérese un entidad E; para cada identificación externa a partir de una entidad E1, se incorpora en el conjunto de atributos pertinentes de E un identificador interno de E1. Este proceso es iterativo si a su vez E1 posee identificadores externos:

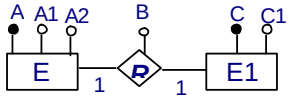


- Atributos pertinentes = {A, A1, A2, B, D}
- Identificador = (A, B, D)

- **Atributos pertinentes de una interrelación:**

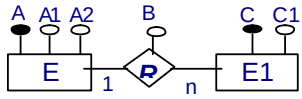
Sea R una interrelación binaria entre las entidades E1 y E2; m1 y m2 denotan las cardinalidades máximas de E1 y E2, respectivamente, en R. Se consideran tres casos:

1. ***Interrelación de uno a uno (m1=1 y m2=1).*** Se incorpora en los atributos pertinentes de R un identificador elegido arbitrariamente, sea de E1 **o** de E2.



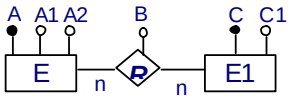
- Atributos pertinentes ={A, B, C}

2. ***Interrelación de uno a muchos (m1=1 y m2=n).*** Se incorpora en los atributos pertinentes de R un identificador de E1.



- Atributos pertinentes ={A, B}

3. ***Interrelación de muchos a muchos(m1=n y m2=n)*.** Se incorpora en los atributos pertinentes de R un identificador de E1 **y** uno de E2.



- Atributos pertinentes = {A, B} o{C, B}

- ✓ Estas reglas se extienden con facilidad a las relaciones n-arias.
- ✓ Cada entidad o interrelación, ampliada adecuadamente con los atributos pertinentes y considerada como aislada del resto del esquema, es equivalente estructuralmente a una relación *plana* y puede normalizarse.

Formas normales

La **primera forma normal** la logra automáticamente cualquier relación ***plana***; es decir, una sin entradas anidadas y en la cual todos los valores sean atómicos, lo que implica que los valores repetitivos o las lista no están permitidas.
Todos los identificadores son simples.

Segunda forma Normal (sin dependencias parciales)

Un atributo ***no es primo*** (es no primo) si no pertenece a ningún identificador. Es ***primo*** en caso contrario.
Una entidad o interrelación está en segunda forma normal si no existe ninguna DF cuyo determinante esté propiamente contenido en un identificador y cuyo atributo del lado derecho sea no primo.

Num-empleado➔nombre, num-div, adm
Num-dep➔ num-div
Num-div➔adm

Tercera Forma Normal

Una DF, A → C, es **transitiva** si existen dos dependencias A → B, B → C, tales que A, B y C sean grupos diferentes de atributos. Entonces, la dependencia A → C se puede inferir como una combinación de A → B, B → C; como tal, esta dependencia es redundante y una causa de anomalías, genera repetición de información.
(dependencia transitiva = un atributo no clave depende de otro no clave)
Una entidad o interrelación está en tercera forma normal si se encuentra en segunda forma normal y no tiene ninguna DF Transitiva.

Forma Normal Boyce –Cood

Una entidad o interrelación está en la forma normal de Boyce-Cood (BCNF), si cada determinante de sus DF es un identificador.
Un atributo B (clave o parte de clave) depende de un a tributo que no es clave

Cuarta Forma Normal

Dada una relación R, con atributos A, B, C, decimos que existe una dependencia entre A y B ***multivaluada*** en R, sii el conjunto de valores de B que concuerdan con el par (A,C) en R depende sólo del valor de A, pero no del de C.
Una ***dependencia multivaluada*** existe cuando un atributo puede determinar más de un valor para otro atributo.
Las dependencias multivaluadas no excluyen la existencia de ciertas tuplas, exigen que otras tuplas de una forma determinada estén presentes en la relación, por esto se conocen como "dependencias generadoras de tuplas".
Sea R un esquema de relaciones y sea aC = R y bC = R , la dependencia Multivaluada a→b, se cumple en R si en cualquier relación legal r(R), para todos los pares de tuplas de t1 y t2 en r tales que t1[a]=t2[a], existen las tuplas t3 y t4 en r tales que:
T1[a]=t2[a]=t3[a]=t4[a]
T3[b]=t1[b]
T3[R-b]=t2[R-b]
T4[b]=t2[b]
T4[R-b]=t1[R-b]
Una relación está en Cuarta Forma Normal sii está en FNBC y no existen dependencias multivaluadas.

Quinta Forma Normal

La quinta forma normal tiene en cuata restricciones de domino del problema que se trata.
Habiendo trabajado correctamente, esta forma se da sola.

Diseño lógico

Diseño lógico ➔Entrada de información = diagrama conceptual + información de carga de BD (saber la cantidad de datos)+ descripción del modelo lógico + criterio de rendimiento (para saber si el sistema va a funcionar bien).

Convertir el esquema conceptual en un esquema lógico de alto nivel

En gral. los diseños lógicos no poseen generalizaciones, ni relaciones ,ni restricciones de cardinalidad. En esta etapa se seleccionan claves, se dividen las entidades, se sacan jerarquías, etc.

Eliminación de jerarquías

Hay 3 alternativas.

1- Agregar a las superentidades todos los atributos de las subentidades y poner un flag para distinguir a las subentidades.

Desventajas: genera valores nulos, todas las consultas que se hacían a las subentidades van a tener que filtrar la superentidades.

Ventajas: es la solución mas simple, se puede aplicar a cualquier jerarquía (tipo de cobertura) .

2- Llevar todos los atributos de las superentidades a las subentidades.

Desventajas: solo sirve para totales y exclusivas, se pierde el concepto de que las subentidades son subconjuntos, se duplican muchos atributos.

Ventaja: sirve cuando la superentidad no debe ser utilizada.

3- Mantener todas las entidades y relacionarlas mediante relaciones especificas.

Ventajas: siempre es posible.

Desventajas: el esquema es complejo, hay redundancia entre las relaciones.

REGLAS:

(?,S) ➔ conviene dejar todo.

(P,E) ➔ conviene dejar al padre.

(T,E) ➔ conviene dejar los hijos y matar el padre.

Selección de clave primaria

Los DBMS necesitan una clave primaria. Por lo tanto debe elegir alguno de los identificadores.

En gral. se prefieren las claves simples e internas.

El ER no es muy diferente al relacional, no incluye generalizaciones.

Decisiones sobre datos derivados

Los datos derivados son datos que se pueden calcular mediante otros datos.

Ventajas: no deben calcularse.

Desventajas: hay que mantenerlos actualizados, ocupan lugar.

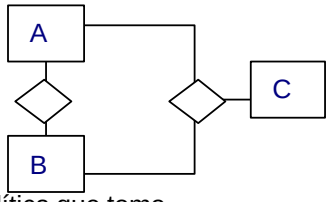
Eliminar atributos con cardinalidad > 1.

No se puede representar generalización y especialización.

Uno o mas atributos de los hijos aparecen en el padre con cardinalidad o

Todos los atributos del padre van a los hijos. Si son muchos hay problema

B y C tienen muchos atributos



En función de la cobertura es la política que tomo

Partición de entidades

Partición vertical: se reorganizan los atributos en entidades separadas, de modo que queden entidades mas pequeñas y en c\u un atributo que considere identificador se va a repetir.

Partición horizontal: se separa la entidad en dos o mas entidades, en la cual se tienen los campos originales. La entidad engloba casos que se pueden trabajar por separado, entonces hay que ver como dividirlo

Para particiones h y v son útiles en las BD distribuidas porque se tratan de mantener los datos pertinentes localmente hasta donde se pueda.

Partición de relaciones

Partición horizontal: se puede dividir por razones de peso, es decir para tener menos entradas en la relación.

Fusión de entidades

Es lo contrario a la partición. Esto se usa cuando los datos de dos entidades se usan conjuntamente, con frecuencia pueden generar dependencias.

Unidad 3 – Modelo Relacional

Definición

El modelo relacional consiste en un conjunto de definiciones de relaciones, en la cual cada relación tiene una clave primaria. Las relaciones producidas por la transformación de esquemas corresponden a entidades o bien a interrelaciones, y mantienen la misma forma normal.

Se espera que los diseñadores de bases de datos ejecutarán primero un diseño conceptual en un modelo de más alto nivel como el modelo de entidades-interrelaciones (ER) y harán corresponder los esquemas conceptuales con los esquemas relaciones.
El esquema ER lógico no es muy diferente de un esquema relacional: no incluye generalizaciones ni subconjuntos. Se necesita realizar otras simplificaciones: la eliminación de atributos compuestos y polivalentes, el modelado de indentificadores externos como internos y la eliminación de interrelaciones.
El elemento básico del modelo es la **tabla (relación)**, y un esquema de base de datos relacional es una colección de definiciones de **tablas(relación)**. El esquema de cada **tabla(relación)** es una agregación de atributos; el conjunto de todos los valores que puede adoptar un atributo en particular se denomina **dominio** de ese atributo.
Un caso de relación (también llamado extensión de la relación) es una tabla con filas y columnas. Las columnas de las relaciones corresponden a los atributos: las filas, denominadas **tuplas**, son colecciones de valores tomados de cada atributo, y desempeñan la misma función que los casos individuales de entidades en el modelo ER. El **grado** de una relación es el número de columnas; la **cardinalidad** de una relación es el número de tuplas.
En el modelo relacional, el concepto de **clave** está definido de manera similar al concepto de identificador en el modelo ER: una **clave** o superclave de una relación es un conjunto de atributos de la relación que identifica de manera única cada tupla de cada extensión de esa relación. Así, la única diferencia en el uso de identificadores y claves es que en el modelo relacional sólo se acepta la identificación interna.
En general, una relación puede tener más de una clave, y cada clave se denomina **clave candidata**. Es común designar una de las claves como **clave primaria** de la relación.
La sencillez del modelo relacional proviene del hecho de que todas las relaciones se definen de manera independiente; no hay conceptos como el de jerarquía, conexión o enlace entre las relaciones en el modelo.

Restricciones

- **De clave:** especifican las claves candidatas de cada esquema de relación: los valores de las claves candidatas deben ser únicos para cada tupla en cualquier caso de esquema de relación.
- **De integridad de entidades**: establecen que ningún valor de clave primaria puede ser nulo.
- **De integridad referencial**: se especifica entre dos relaciones, y se usa para mantener la congruencia entre las tuplas de dos relaciones. Informalmente, la restricción de integridad referencial establece que una tupla de una relación que haga referencia a otra relación debe referirse a una tupla existente en esa relación. Para definir la integridad referencial de manera más formal, primero se debe definir el concepto de clave ajena. En otras palabras la integridad referencial es asegurar que un valor que aparece en una relación también aparezca en un atributo de otra.

Conversión del modelo ER lógico al relacional

La conversión de un esquema ER para que se corresponda con el modelo relacional abarca las siguientes acciones: eliminación de identificadores externos, eliminación de atributos compuestos y polivalentes, transformación de entidades y transformación de interrelaciones (uno a uno, uno a muchos y muchos a muchos)

Eliminación de identificadores externos

Como no se puede usar identificadores externos en el modelo relacional, se deben transformar en identificadores internos. Supongamos que la clave primaria de una entidad E1 es un identificador externo o mixto, y sea la entidad E2 la que suministra la identificación externa a través de la interrelación R a E1; supondremos además que E2 tiene un identificador interno como su clave primaria. Para eliminar el identificador externo de E1, se debe importar a la entidad E1 la clave primaria de E2. Después de esta operación, puede eliminarse la interrelación R; de hecho, el vínculo entre E1 y E2 se modela al insertar en E1 la clave primaria de E2. Este procedimiento no sería posible si E2 a su vez estuviera identificada externamente por alguna otra entidad E3. Este proceso debe realizarse empezando por las entidades que tienen como clave primaria un identificador interno (entidades fuertes) y luego continuando con las entidades vecinas. Los identificadores pueden propagarse según se necesite para la identificación externa.

Eliminación de atributos compuestos y polivalentes

El modelo relacional en su forma básica contiene sólo atributos simples, monovalentes; por ello, los atributos compuestos y polivalentes deben ser modelados en términos de atributos simples monovalentes.

Con cada atributo compuesto, se tienen básicamente dos alternativas:

1. Eliminar el atributo compuesto considerando todos sus componentes como atributos individuales o

2. Eliminar los componentes individuales y considerar el atributo compuesto entero como un solo atributo.

Los atributos polivalentes requieren la introducción de entidades nuevas; la nueva entidad contiene el atributo polivalente más el identificador de la entidad original; el identificador de la nueva entidad es el conjunto de todos sus atributos.

Atributos polivalentes de las interrelaciones

Si el atributo polivalente pertenece a una interralación R entre entidades E1 y E2, se necesita crear una entidad nueva NE para representarlo. La nueva entidad NE incluye uno o dos atributos tomados de E1, E2, o ambos, dependiendo del tipo de la interrelación:

1. Si la interrelación es de uno a uno, NE incluye la clave primaria de E1 o bien de E2.
2. Si la interrelación entre E1 y E2 es de uno a muchos, NE incluye la clave primaria de E2 (suponiendo que E2 está en el lado de "muchos")
3. Si la interrelación entre E1 y E2 es de muchos a muchos, NE incluye las claves primarias tanto de E1 como de E2.

La clave primaria de NE está constituida por todos sus atributos: esto incluye aquellos que se "tomaron prestados" de E1 y E2, y el atributo polivalente. Cualquier atributo no polivalente que tenga R seguirá siendo atributo de R.

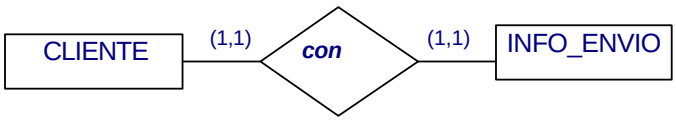
Transformación de entidades

Se transforma cada entidad del esquema en una relación. Los atributos y la clave primaria de la entidad se convierten en los atributos y la clave primaria de la relación.

Transformación de interrelaciones uno a uno

☞ ***Integración en una relación***

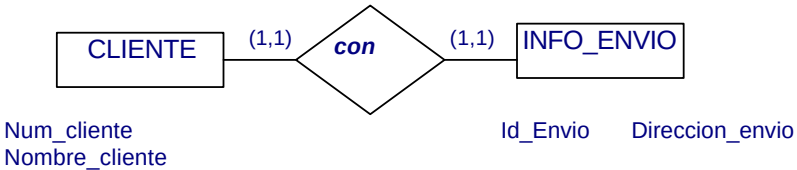
Caso 1: Las dos entidades tienen las mismas claves primarias



ENVIO_CLIENTE (Num_Cliente, Nombre_Cliente, Dirección_envio)

Las dos relaciones se integran en una sola tabla combinando todos los atributos e incluyendo la clave primaria solo de una.

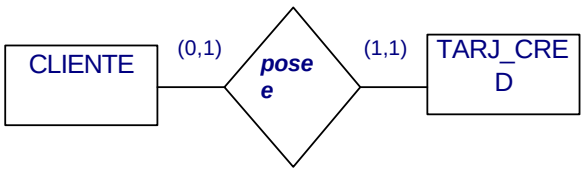
Caso 2: Las dos entidades tienen las claves primarias distintas



ENVIO_CLIENTE (Num_Cliente, Nombre_Cliente, Id_Envio, Dirección_envio)

]

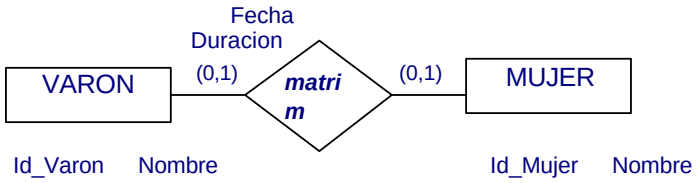
☞ ***Definición de una relación aparte***
Caso 1: una entidad tiene participación parcial



CLIENTE (Num_Cliente, Nombre_Cliente)
TARJ_CRED (Id_Tarj, Credito)
POSEE(Id_Tarj, Num_Cliente)

Quedan tres tablas, la nueva tiene la clave de la partición total (tarj_c

Caso 2: las dos entidades tienen participación parcial



VARON (Id_Varon, Nombre)
MUJER (Id_Mujer, Nombre) *las dos claves de las entidades*
MATRIM(Id_Varon, Id_Mujer, Fecha, Duracion)

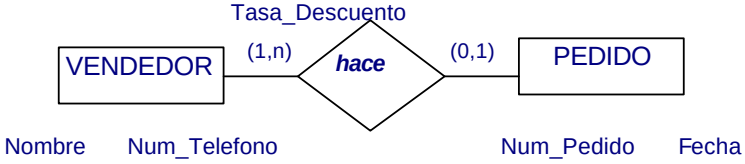
Transformación de interrelaciones uno a muchos (se mira la rel. del lado del uno)
1

☞ ***Participación parcial o total***
Caso 1: La entidad del lado de "muchos" tiene participación obligatoria

CIUDAD (Nombre_Ciud, Nom_Estado, Habitantes)
ESTADO(Nom_Estado, Gobernador, Habitantes)

Se pueden hacer dos tablas, la clave de "mucho" se pega en el lado de "uno".

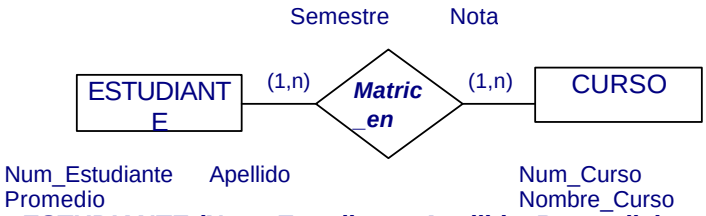
Caso 2: La entidad en el lado de "muchos" tiene participación parcial.



VENDEDOR (Nombre, Num_Telefono)
PEDIDO (Num_Pedido, Fecha)
HACE (Num_Pedido, Nombre_Vendedor, Tasa_Descuento)

Se hacen 3 relaciones a la nueva se llevan las claves primarias y los atributos de la interrelación.

Transformación de interrelaciones muchos a muchos

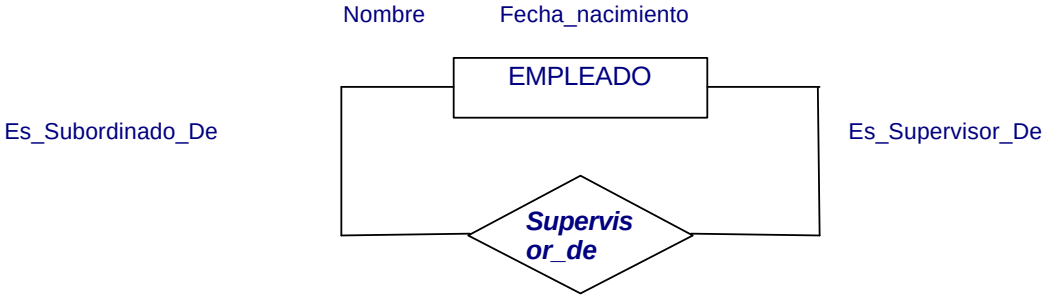


ESTUDIANTE (Num_Estudiante, Apellido, Promedio)
CURSO (Num_Curso, Nombre_Curso)
MATRIC_EN (Num_Estudiante, Num_Curso, Semestre, Nota)

Se hacen 3 tablas

Transformación de interrelaciones n-arias y recursivas

Sigue las mismas reglas de transformación que las binarias n-a-n. La re. hereda todas las claves de las entidades. En algunos casos la clave no es mínima.(si alguna tuviera participación (1,1) se podría fusionar con la interrelación).



Interrelación de muchos a muchos
EMPLEADO (Nombre, Fecha_Nacimiento)
SUPERVISOR_DE (Nombre_Supervisor, Nombre_Subordinado)

Interrelación de uno a muchos
EMPLEADO (Nombre, Fecha_Nacimiento)
SUPERVISOR_DE (Nombre_Subordinado, Nombre_Supervisor)

o bien
EMPLEADO (Nombre, Fecha_Nacimiento, Nombre_Supervisor)

UNIDAD 4➔LENGUAJES DE CONSULTA

Un **lenguaje de consulta** es un lenguaje en el que un usuario solicita información de la base de datos. Pueden clasificarse en lenguajes ***procedimentales*** o ***no procedimentales***. En un lenguaje procedimental, el usuario da instrucciones al sistema para que realice una secuencia de operaciones en la base de datos para obtener el resultado deseado. En un lenguaje no procedimental, el usuario describe la información sin indicar un procedimiento específico para obtenerla.
El **álgebra relacional** es procedimental, mientras que el **cálculo relacional de tuplas** y **el cálculo relacional de dominios** son no procedimentales (son declarativos).
Un lenguaje de ***manipulación de datos (DML)*** completo incluye no sólo un lenguaje de consulta, sino también un lenguaje para la modificación de la base de datos. Dichos lenguajes incluyen órdenes para insertar y borrar tuplas así como órdenes para modificar partes de tuplas existentes.

El modelo relacional consiste en un conjunto de definiciones de relaciones, en la cual cada relación tiene una clave primaria. Las relaciones producidas por la transformación de esquemas corresponden a entidades o bien a interrelaciones, y mantienen la misma forma normal.

Algebra relacional

Definición

Es un lenguaje de consulta procedimental. Consta de un conjunto de operaciones que toman una o dos relaciones como entrada y producen una nueva relación como resultado.

Definición formal del álgebra relacional

Una expresión básica en el álgebra relacional consta de cualquiera de las siguientes:
Una relación en la base de datos
Una relación constante
Una expresión general en el álgebra relacional se construye a partir de subexpresiones. Sean E1 y E2 expresiones del álgebra relacional. Entonces las siguientes son todas expresiones del álgebra relacional:

- E1 U E2
- E1 – E2
- E1 x E2
- σ_P(E1), donde P es un predicado con atributos de E1.
- π_S(E1), donde S es una lista que consta de alguno de los atributos de E1
- δ_x(E1), donde x es el nuevo nombre de la relación E1.

Operaciones fundamentales

Son: seleccionar, proyectar, producto cartesiano, renombrar, unión y diferencia de conjuntos.
Existen otras operaciones, a saber, intersección de conjuntos, producto natural, división y asignación.

Operaciones unarias ➔ operan sobre una relación

Seleccionar

- Selecciona tuplas que satisfacen un predicado dado
- Notación: s
- El predicado aparece como subíndice de σ.

Ej**: *Seleccionar aquellas tuplas de la relación préstamo en la que la sucursal es Perryridge*.**

$$\sigma_{\text{nombre sucursal = "Perryridge}}\ (\text{Préstamo})$$

Encontrar todas las tuplas en las que la cantidad prestada es más de 1200 pesos

$$\sigma_{\text{cantidad > 1200}}\ (\text{Préstamo})$$

Para las comparaciones se pueden usar **=, ¹ , >, <, <=, >=.** Además se pueden combinar varios predicados en un predicado mas complejo usando los conectores (Ù) y (Ú).

Encontrar las tuplas pertenecientes a prestamos de mas de 1200 pesos hechas por la sucursal Perryridge.

$$\sigma_{\text{nombre\_sucursal = 'Perryridge'}\ \wedge\ \text{cantidad > 1200}}\ (\text{Préstamo})$$

El predicado de selección puede incluir comparaciones entre dos atributos.

Consideremos el esquema de relaciones:
Esquema_Servicio(nombre_cliente, nombre_banquero)

Podemos encontrar a todos aquellos clientes que tienen el mismo nombre que su banquero personal.

$\sigma_{nombre\_cliente = nombre\_banquero}$ (Servicio)

Proyectar

- Devuelve la relación argumento con columnas omitidas.
- Notación: Õ
- La relación argumento se escribe a continuación de Π entre paréntesis.

Ej: ***queremos una relación que muestre los clientes y las sucursales en las que tienen préstamos.***

$\Pi_{nombre\_sucursal,\ nombreCliente}$ (Préstamo)

Renombrar

- Se utiliza cuando hay nombres repetidos en la misma consulta
- Notación: d

Ej: **Supongamos que queremos saber todas las persona que viven en la misma calle que un determinado cliente. Primero debemos saber en que calle vive ese cliente, y luego si vemos los que viven ella.**

Persona (N°_Cliente, Calle, Número, Te)
No se puede hacer Persona X Persona ya que tienen el mismo nombre. Para eso se renombra una de las Tablas
Persona X δ_P (Persona)

Ahora si vemos la consulta:

$\pi_{P.N°\_Cliente}(\sigma_{P.Calle = Persona.Calle}((\sigma_{N°\_Cliente = 'Garcia'}(Persona) \times \delta_P(Persona)))$

Operaciones binarias. operaciones que operan sobre pares de relaciones.
Producto Cartesiano

- Nos permite combinar información de varias relaciones
- Notación: **X**
- El producto cartesiano de las relaciones r1 y r2 es r1X r2.

$Tabla_1$	$Tabla_2$	$Tabla_1$ X $Tabla_2$
T_{11} T_{12} T_{13} . .	T_{21} T_{22} T_{23} . . .	T_{11} T_{21} T_{11} T_{22} T_{11} T_{23} T_{12} T_{21} T_{12} T_{22} T_{12} T_{23} T_{13} T_{21} .

$Tabla_1$	$Tabla_2$	$Tabla_1$ X $Tabla_2$
A, B, C	D, E, F	A, B, C, D, E, F

$Tabla_1$	$Tabla_2$	$Tabla_1$ X $Tabla_2$
A, B, C	D, A, F	Tabla1.A, B, C, D, Tabla2.A, F

Ej: Cliente (N°_Cliente, N°_OpBanc)
Persona (N°_Cliente, Direcc, Te)

Queremos determinar la dirección de todos los clientes del operador bancario Garcia

$\Pi_{Direcc}(\sigma_{N°\_OpBancN°\_Cliente = Persona.N°\_Cliente}(Cliente\ X\ Persona))$

Unión

- Devuelve las tuplas comunes a dos relaciones. Unión matemática, quita repetidos
- Notación: **U**
- Debemos asegurarnos que las uniones se toman entre relaciones compatibles. Para que una operación de unión r U s sea válida exigimos que se cumplan:
  1. Las relaciones r y s deben tener el mismo número de atributos
  2. Los dominios del atributo i-ésimo de r y del atributo i-ésimo de s deben ser los mismos.

Diferencia de conjuntos

- *Nos permite encontrar tuplas que están en una relación pero no en otra*.
- Notación: **-**

✓ La expresión r – s da como resultado una relación que contiene aquellas tuplas que están en r pero no en s.

Ej: Depósitos (Nombre_Cliente, N°_Sucursal, Saldo)
Prestamos (Nombre_Cliente, N°_Sucursal, Saldo)

Queremos saber todos aquellos clientes que tienen un depósito efectuado y que no tienen préstamos otorgados.

$\pi_{N°\_Cuenta}$ (Deposito) - $\pi_{N°\_Cuenta}$ (Préstamo)

Queremos determinar la cuenta que tiene saldo mayor en Depósitos

π_{Saldo}(Deposito) - $\pi_{Deposito.saldo}$($\sigma_{Deposito.Saldo < D.Saldo}$ (Deposito x δ_{D}(Deposito)))

Otras operaciones:

Las operaciones fundamentales del álgebra relacional son suficientes para expresar cualquier consulta en álgebra relacional. Pero se definen operaciones adicionales que no añaden ninguna potencia al álgebra, pero que simplifican consultas comunes.

Intersección de conjuntos

- Cualquier expresión del álgebra relacional que use la intersección de conjuntos puede reescribirse sustituyendo la operación intersección por un par de operaciones de diferencia de conjunto así:

$$r \cap s = r - (r - s)$$

- Notación: Ç

Producto natural

- Es una operación binaria que nos permite combinar Selección y Producto Cartesiano en una sola operación.
- Notación: **|X|**
- La operación producto natural forma un producto cartesiano de sus dos argumentos, realiza una selección forzando la igualdad en aquellos atributos que aparezcan en ambas planificaciones de relaciones y, finalmente, quita las columnas duplicadas.
- En otras palabras, hace el producto cartesiano y elimina las duplicaciones
- Hay que tener cuidado con los que tienen el mismo nombre pero son distintas.
- Se aplica sobre los atributos que nos interesa.
- Si se hace |X| entre T y S, donde r y s no tienen atributos en común => |X| = X

Ej: **Encontrar a todos los clientes que tienen un préstamo en la sucursal X.**

Cliente (cod_Cliente, Nombre, Dirección)
Préstamo (cod_Cliente, Saldo, Sucursal)

Con producto Cartesiano:

π_{Nombre}($\sigma_{Sucursal = 'x'}$ ($\sigma_{Prestamo.cod\_Cliente}$(Préstamo x Cliente)))

Con producto Natural:

π_{Nombre}($\sigma_{Sucursal = 'x'}$ (Préstamo |x| Cliente))

Nombre sucursal y activo de sucursales con clientes con cuentas y que viven en La Plata

Cliente (cod_Cliente, Nombre, Dirección)
Préstamo (cod_Cliente, Saldo, Sucursal)

Sucursal (Nombre_Sucursal, Activo)

$\pi_{Nombre\_Sucursal, Activo}(\sigma_{Direccion = \text{'La Plata'}}$ (Sucursal |X| Préstamo |X| Cliente))

División

- Involucra consultas con ∀
- Notación: ¸
- En términos de operaciones fundamentales, tenemos:

$r \div s = \pi_{R-S}(r) - \pi_{R-S}((\pi_{R-S}(r) \times s)) - r)$

$\pi_{R-S}(r)$ nos da todas las tuplas que satisfacen la segunda condición de la definición de división.

Asignación

- Expresión que asigna a una variable temporal
- Notación: Ü

Variables temporales de relación
Ej: división podría ser
temp ¬ P R-S (r)
temp – Π_{R-S} ((temp × s) – r)

Modificación de bases de datos
La veremos sólo en álgebra relacional.
Se usa la notación de asignación sobre tablas ya existentes.

Eliminación	**r ¬ r – E**
Inserción	**r ¬ r È E , E puede ser una tupla constante**
Actualización	d **A ¬ E (r)** **Ej.** d **saldo ¬ saldo * 0.95 (saldo)**

Calculo Relacional de Tuplas

Definición
Es un lenguaje de consultas no procedimental. Describe la información deseada sin dar un procedimiento específico para obtener esa información.
Una consulta en el cálculo relacional de tuplas se expresa como: **{t / P(t)}**
es decir, el conjunto de todas las tuplas t, tal que el predicado P (formula), es verdadero para t. Usamos t[A] para representar el valor de la tupla t en el atributo A, y t e r para indicar que la tupla t está en la relación r.
Se dice que una variable de tupla es una *variable libre* si sale de una tabla y se instancia con todos los atributos de la tabla (no esta condicionada a nada). Una variable es limita si está cuantificada por un ∃ (existencial) o un ∀ (universal).

Una formula se compone de átomos y un átomo tiene la siguiente forma:

- s ∈ r, s es una variable de tupla y r es una relación.
- s[x] Θ u[y] (s y r var. de tuplas y Θ el operador =, <,<=, >,>=) se quiere que todos los atributos x e y tengan dominios cuyos miembros puedan compararse por medio de Θ
- s[x] Θ c, donde s es una variable de tupla, x un atributo sobre el que s esta definida, Θes un operador de comparación y c constante en el dominio del atributo x

Las formulas se construyen a partir de átomos usando las siguientes reglas:

- Un átomo es una formula
- Si P1 es una formula, entonces también lo son ¬P1 y P1
- Si P1 yP2 son formulas entonces también lo son P1VP2, P1^P2 y ¬P1VP2
- Si P1(s) es una formula que contiene la variable de tupla libre s, entonces ∃ s E r(P1(s)) y V s E r(P1(s)) también son formulas.

Reglas de equivalencia

- P1^P2 == ¬(¬P1 V¬P2)
- ∀t E r(P1(t)) == ¬∃t Er(¬ P_1 (t))
- ¬P1 VP2 == p1➔ P2

SEGURIDAD DE LAS EXPRESIONES

Una expresión en calculo relacional de tuplas puede generar una relación infinita. Para evitar esto definimos dominio de una formula P, dom(P) que es el conjunto de los valores referenciados por P, valores que aparecen en una o mas relaciones cuyos nombres aparecen en P.

Una expresión {t | P(t) } es segura si todos los valores que aparecen en el resultado son valores de dom(P).

PODER EXPRESIVO DE LOS LENGUAJES

El calculo relacional de tuplas, restringido a expresiones seguras, tiene el mismo poder expresivo que el álgebra relacional (para cada expresión en el álgebra relacional existe una expresión equivalente en el calculo relacional de tuplas y viceversa.

OPERACIONES

SELECCION

Encontrar el nombre-sucursal,numero-prestamo, nombre-cliente y cantidad para prestamos mayores de 1200:

{t\ t E préstamo ^ t[cantidad]>1200}

PROYECCION

la misma consulta anterior pero buscando únicamente los clientes: (el conjunto de todas las tuplas t tales que existe una tupla s en la relación préstamo para la cual los valores de t y s para el atributo nombre-cliente son iguales y el valor s es mayor de 1200)

{t\ ∃s ∈ préstamo (t[nombre_cliente] = s[nombre_cliente] ^ s[cantidad]>1200)}

Encontrar todos los clientes que tienen un préstamo de la sucursal Pueyrredon y las ciudades en las que viven (requiere dos tablas por lo que dos cláusulas existe conectadas por^):

{t\∃ s E préstamo (t[nombre-cliente] = s[nombre-cliente] ^ s[nombre sucursal] = "pueyrredon" ^

∃ u E cliente (u[nombre-cliente] = s[nombre-cliente] ^ t[ciudad-cliente]=u[ciudad-cliente]))}

UNION

Encontrar todos los clientes que tienen un préstamo, una cuenta o las dos cosas en la sucursal pueyrredon (usamos dos existe conectados con o)

{t\∃ s E préstamo (t[nombre-cliente] = s[nombre-cliente] ^ s[nombre sucursal] = "pueyrredon")

V ∃ u E deposito (t[nombre-cliente]=u[nombre-cliente] ^ u[nombre-sucursal]="pueyrredon")}

Encontrar todos los clientes que tienen un préstamo y una cuenta en la sucursal pueyrredon :

{t\∃ s E préstamo (t[nombre-cliente] = s[nombre-cliente] ^ s[nombre sucursal] = "pueyrredon") ^ ∃ u E

deposito (t[nombre-cliente]=u[nombre-cliente] ^ u[nombre-sucursal]="pueyrredon")}

Encontrar todos los clientes que tienen un préstamo, una cuenta o las dos cosas en la sucursal pueyrredon (usamos dos existe conectados con o)

{t\∃ s E préstamo (t[nombre-cliente] = s[nombre-cliente] ^ s[nombre sucursal] = "pueyrredon") V ∃ u E

deposito (t[nombre-cliente]=u[nombre-cliente] ^ u[nombre-sucursal]="pueyrredon")}

DIFERENCIA DE CONJUNTOS➔ NO ¬

Encontrar todos los clientes que tienen una cuenta pero no un préstamo en la sucursal Pueyrredon:

{t\∃ u E deposito (t[nombre-cliente]=u[nombre-cliente] ^ u[nombre-sucursal]="pueyrredon") ^ ¬∃ s E

préstamo (t[nombre-cliente] = s[nombre-cliente] ^ s[nombre sucursal] = "pueyrredon")}

Encontrar todos los clientes que tienen una cuenta en todas las sucursales de brooklyn:

{t\¬V u E sucursal (u[ciudad-sucursal] = "brooklyn" V ∃ sE deposito (t[nombre_cliente] = s[nombre-

cliente] ^ u[nombre-cliente] = s[nombre-sucursal]))}

Calculo Relacional de Dominio

Esta forma usa variables de dominio que toman valores del dominio de un atributo más que valores de una tupla completa. Esta íntimamente relacionado con el calculo racional de tuplas.

DEFINICION FORMAL

Una expresión es de la forma: $\{ <x_1, ..., x_n> \mid P(x_1, ..., x_n) \}$, donde x1,...xn representan variables de dominio.
P representa una formula compuesta por átomos.

Un átomo tiene una de las siguientes formas:

- $< x_1, ..., x_n > \in r$, donde r es una relación en n atributos y x1,...xn son variables de dominio o constantes de dominio.
- $x \Theta y$, donde x e y son variables de dominio y Θ es un operador de comparación (<=,<,=,>,>=). X e y deben tener dominios que puedan compararse con Θ.
- $x \Theta c$, donde x es una variable de dominio, V es un operador de comparación y c es una constante en el dominio del atributo para el cual x es una variable de dominio.

Reglas para construir las formulas a partir de átomos:

- Un átomo es una formula.
- Si P1 es una formula entonces $\neg P1$ y (P1) también lo son.
- Si P1 y P2 son formulas entonces $P_1 \wedge P_2$, $P_1 \vee P_2$, $P_1 \Rightarrow P_2$ también lo son.
- Si P1(x) es formula en x, donde x es una variable de dominio, entonces $\exists x(P1(x))$ y $\forall x(P1(x))$ también son formulas. Notación abreviada: $\exists a,b,c,(P(a,b,c))$ para $\exists$ a($\exists$ b ($\exists$ c (P(a,b,c))))

EJEMPLOS

Encontrar el nombre de sucursal, numero de préstamo, nombre de cliente y cantidad de prestamos mayores de 1200:

$\{ <b,l,c,a> \mid <b,l,c,a> \in$ préstamo $\wedge$ a > 1200 $\}$

encontrar todos los clientes que tienen un préstamo por una cantidad mayor de 1200

$\{ <c> \exists b,l,a(<b,l,c,a> \in$ préstamo $\wedge$ a > 1200) $\}$

Encontrar todos los clientes que tienen un préstamo de la sucursal pueyrredon y la ciudad en la que viven:

$\{ <c,x> \mid \exists b,l,a(<b,l,c,a> \in$ préstamo $\wedge$ b="pueyrredon") ^ $\exists$ y(<c,y,x> E cliente)$\}$

Encontrar todos los clientes que tienen un préstamo, una cuenta o ambos en la sucursal pueyrredon

$\{ <c,x> \mid \exists b,l,a(<b,l,c,a> \in$ préstamo $\wedge$ b="pueyrredon") V $\exists$ b,a,n(<b,a,c,n> E deposito ^ b="pueyrredon")$\}\}$

Encontrar a todos los clientes que tienen una cuenta en todas las sucursales situadas en brooklyn:

$\{<c> \mid \forall$ x,y,z($\neg$(<x,,y,z>E sucursal(V z $\neq$ "brooklyn" v ($\exists$ a,n(<x,a,c,n> E deposito)))$\}$

interpretacion de la expresión anterior: el conjunto de la tuplas (nombre-cliente)<c> tal que para todas las tu[las (nombre-sucursal, activo, ciudad-sucursal), <x,y,z> al menos una de las siguientes declaraciones es verdadera:

- <x,y,z> no es una tupla de la tabla sucursal
- z no tiene el valor Brooklyn
- el cliente c tiene una cuenta (numero a y saldo n) en la sucursal x

SEGURIDAD DE EXPRESIONES:

Para el calculo relacional de dominios debemos preocuparnos también por las formulas dentro de las cláusulas existe y para todos.
En el calculo relacional de tuplas restringimos cualquier variable cuantificada existencialmente a moverse sobre relación especifica. Añadimos reglas a la definición de seguridad, decimos que **la expresión {<x1,...xn>| P(x1,...xn)} es segura si se cumplen todas las condiciones siguientes:**

- **Todos los valores que aparecen en tuplas de la expresión son valores de dom(P)**
- **Para cada subformula "existe" de la forma $ x(P1(x1)), la subformula es verdadera sii existe un valor x en dom(P1) talque P1(x) es verdadero.**
- **Para cada subformula "para todos" de la forma " x (P1(x)), la subformula es verdadera sii P1(x) es verdadera para todos los valores x de dom(P1)**

El propósito de estas reglas adicionales es asegurar que podemos probar las subformulas para todos y existe sin tener que probar infinitas posibilidades.
Para afirmar que $\forall x(P1(x))$ es verdadera, debemos probar todos los posibles valores. Esto requiere que examinemos infinitos valores pero si sabemos que la expresión es segura es suficiente probar P1(x) para aquellos valores tomados de dom(P1).

La expresión {<b,c,l,a> | Ø (<b,c,l,a> Î préstamo)} no es segura porque permite valores en el resultado que no están en el dominio de la expresión.

Procesamiento de Consultas

El costo de las consultas se basa en los accesos a disco. Hay que tratar de minimizar estos accesos, para lo cual se debe optimizar las consultas.

Las consultas que hacen los usuarios no son siempre eficientes, por lo cual el sistema debe optimizar dichas consultas. Para ello primero debe transformar la consulta a la forma interna mientras va verificando en este proceso la sintaxis. Si la consulta se realizó en base a vistas, se reemplazan las llamadas a la vista por las expresiones que calculan la vista. Luego comienza la optimización, esto es, se trata de buscar una expresión en álgebra relacional equivalente, que sea más eficiente. Luego se especifica la forma de concreta de acceso tratando de minimizar el número de accesos a disco.

Un DBMS a partir de una consulta de usuario mas entrada del sistema crea una consulta depurada (el sistema trata de minimizarla, hacerla mas eficiente usando álgebra relacional)
Para hacer mas eficiente una consulta tener en cuenta los índices (elemento para acceder a la tabla mas eficientemente y en forma ordenada). Algunos DBMS pueden crear índices temporales durante una consulta para hacerlas mas eficientes (minimizar el numero de accesos).

Equivalencia entre Expresiones

Ya se vio que en AR se pueden expresar consultas de diferentes formas. Ahora buscaremos una expresión equivalente mas eficiente.
Forma de optimización:
Selección➔ Realizar las operaciones de selección lo antes posible, de modo de achicar las subtablas. Cambiar: $s_{p1}(s_p$ **2(e))**

Si se tiene una selección con doble condición se transforma en dos selecciones sucesivas.

Proyección➔ Realizar las proyecciones lo antes posible.

$$p\textbf{sucursal, archivo}(p\textbf{ suc }(p\textbf{ lp(cliente) |X| deposito)) |X| sucursal}$$

Producto Natural➔ Hay que elegir un orden en la resolución (asociar y conmutar las relaciones) del producto natural de modo que las relaciones intermedias que se generan sean lo más pequeñas posibles. El producto natural es mejor usarlo entre pares: **r1 |X|r2 |X|r3= r1 |X| (r2|X| r3) y r1|X|r2 = r2|X|r1**

Otras equivalencias: no necesariamente tienen la misma eficiencia pero el resultado es el mismo.

$\sigma_p(r1 \cup r2) = \sigma_p(r1) \cup \sigma_p(r2)$
$\sigma_p(r1 - r2) = \sigma_p(r1) - r2 = \sigma_p(r1) - \sigma_p(r2)$
$(r1 \cup r2) \cup r3 = r1 \cup (r2 \cup r3)$
$r1 \cup r2 = r2 \cup r1$

Estimación de costo de consultas

La estrategia depende del tamaño de la relación y de la distribución de los valores dentro de las columnas. Para la elección de la estrategia se utilizan estadísticas, así cada relación guarda información sobre:

nr: número de tuplas en la relación
sr : tamaño de cada tupla en bytes.
V(A,r): número de valores distintos que aparecen en la relación r para el atributo A.

Producto cartesiano➔ rXt tendra nr*nt tuplas y cada tupla tendrá sr+st bytes.

Selección➔ para estimar cuantas tuplas satisfacen un predicado de selección:

#tuplas = nr / V(A,r) si se supone distribución uniforme
#bytes = sr

Producto natural → r 1|x| r2 (r minúscula = tabla, R mayúscula = atributo)

Tres casos: r1(R1), r2(R2) →

R1 ∩ R2 = ∅ (no tienen nada en común) → r1 |x| r2 = r1 x r2

R1 ∩ R2 , clave en R1 o en R2 , entonces cada tupla de r2 se une a una tupla de r1
→ r1 |x| r2 < = r2 (o <=r1)

R1 ∩ R2 no es clave ni de R1 ni R2 → R1 ∩ R2 = { A } (atributo que no forma clave en ninguno)

Si t ∈ r1 produce nr2 / V(A,r2) tuplas

Entonces #tuplas de r1|x| r2 = (nr1 * nr2) / V(A, r2)

Si t ∈ r2 → (nr2 * nr1) / V(A, r1)

V(A,r) varia si pongo r1 o r2 por lo que tengo que elegir el que de valor mas grande → el menor de los dos..

Caso adicional mantener actualizado sr (trivial), nr y V(A,r2) que es el mas propenso a cambiar y el mas dificil de calcular → uso estadísticas (usadas para tratar las consultas) recientes para actualizaciones.

Cuándo actualizar los datos estadísticos vistos anteriormente:

On line: agregan alto costo de mantenimiento, no es imprescindible

Batch: períodos entre actualizaciones. Las estadísticas no son exactas pero están bastante aproximadas al resultado correcto.

Costo de acceso por índices

La estrategia detallada para procesar una consulta se denomina plan de acceso de la consulta. Un plan incluye las operaciones que se van a usar, los índices que se van a usar y el orden en que se van a realizar las operaciones. El uso de índices agrega un tiempo adicional.

Si tengo varios procesadores en paralelo y cada uno atiende una subconsulta, la consulta final es mucho mas eficiente. Se necesita un soporte de manejo de red y sistemas distribuidos mas fuertes de una maquina para una consulta.

Unidad 5 - SQL

SQL: Lenguaje de Consulta Estructurado

Lenguaje de BD relacional estándar. Partes:

1. **DDL➔** proporciona ordenes para definición de esquemas, eliminar esquemas, creación de índices y modificar esquemas

2. **DML interactivo➔** incluye un lenguaje de consultas basado en el álgebra relacional y el calculo relacional de tuplas (CRT). También tiene ordenes para inserción, modificación y eliminación de tuplas.

3. **DML embebido➔** la forma inmersa de SQL esta diseñada para usar dentro de los lenguajes de programacion de propósito general (PL/I, Cobol, Pascal, Fortran y C).

4. **Definición de Vistas➔** El SQL DDL incluye ordenes para definir vistas.

5. **Autorización➔** El SQL DDL incluye ordenes para definir derechos de acceso a relaciones y vistas

6. **Integridad➔** incluye ordenes para especificar restricciones de integridad complejas.

7. **Control de transacciones➔** incluye ordenes para especificar el comienzo y final de transacciones.

Estructura básica

Consta de tres cláusulas:
Select → proyección del álgebra relacional. Para listar atributos que se desean en el resultado de una consulta.
From → producto cartesiano en el A. R. Lista las relaciones que se van a examinar en la evaluación de la expresión.
Where → selección en el A.R. consta de un predicado que implica atributos de las relaciones que aparecen en la clausula from.

FORMA DE UNA CONSULTA TIPICA:

Select $A_1, A_2, \ldots, A_n$
From $r_1, r_2, \ldots, r_m$
Where P

Ai son atributos, ri relaciones y P predicado. Equivalencia con AR: $\Pi_{A1..An}(\sigma_P(r_1 \times \ldots \times r_n))$
Si se omite la clausula where y el predicado es verdadero la lista de atributos puede sustituirse por *.
SQL forma el producto cartesiano de las relaciones nombradas en la clausula from, realiza una selección del AR usando el predicado de la clausula where y despues proyecta el resultado a los atributos de la clausula select.

Operaciones

- **Operaciones de conjuntos y tuplas duplicadas**
  SELECT ALL para especificar explicitamente que no se eliminan los duplicados.
  SELECT DISTINCT para forzar la eliminacion de duplicados

- **Operaciones de conjuntos**
  UNION➔ Clientes que tienen un prestamo, una cuenta o las dos cosas en la sucursal pueyrredon:

```
(select distinct nombre-cliente
from deposito
where nombre sucursal ="pueyrredon")
union
(select nombre-cliente
from préstamo
where nombre-sucursal="pueyrredon")
```

La operación unión elimina las tupla duplicadas, para retener duplicados debemos usar unión all.

INTERSECT➔ Clientes que tienen un préstamo y una cuenta en la sucursal pueyrredon:

```
(select distinct nombre-cliente
 from deposito
where nombre-sucursal="pueyredon")
 intersect
```

(**select distinct** nombre-cliente
from prestamo
where nombre-sucursal="pueyrredon")

MINUS➔ clientes de la sucursal pueyrredon que tiene una cuenta allí pero no un prestamo:

(**select distinct** nombre-cliente
from deposito
where nombre-sucursal="pueyrredon")
MINUS (select distinct nombre-cliente
From prestamo
Where nombre-sucursal="pueyrredon")

La resta e intersección no eliminan repetidos. La unión lo hace, salvo que se especifique ALL.

- **Predicados y conectores (joins)**

El producto Natural se define en términos de un producto cartesiano, una selección y una proyección. **Encontrar el nombre y la ciudad de todos los clientes que tienen un prestamo en alguna sucursal:**

SELECT cliente.nombre-cliente, ciudad-cliente
FROM prestamo, cliente
WHERE prestamo.nombre-cliente=cliente.nombre-cliente

SQL usa los conectores lógicos and, or y not en la clausula where, en vez de los símbolos matemáticos ^,v, ¬.
Muchas implementaciones de SQL incluyen funciones aritméticas especiales para tipos de datos particulares.
El operador de comparación ***BETWEEN*** simplifica las cláusulas where que especifican que un valor sea menor o igual que un valor dado y mayor o igual que otro. También podemos usar el operador **not between.**
Ejemplo: encontrar el numero de cuenta de as cuentas con saldos entre 90000 y 100000 :

Select numero-cuenta ***from*** deposito ***WHERE*** saldo ***BETWEEN*** 90000 ***AND*** 100000 en vez de
Select numero-cuenta ***from*** deposito ***WHERE*** saldo **>=** 90000 ***AND*** saldo<=100000

- **Operadores de selección para comparaciones de cadenas de caracteres ➔ String matching:** *LIKE '_' y '%'*

Los modelos son sensibles al tipo de letra, mayúscula no es igual que minúscula
Ejemplo: encontrar los nombres de todos los clientes cuya calle incluye la subcadena Main:

select nombre-cliente **from** cliente **WHERE** calle LIKE "%Main%"
... WHERE abcd LIKE "_ _ _%"
Carácter de escape: ... LIKE "ab\%cd%" matchea con las que empiezan con ab%cd
El carácter "%" igual a cualquier subcadena y el carácter "_" igual a cualquier caracter.

- **Pertenencia a conjuntos:** *IN*

El conector in prueba si se es miembro de un conjunto, donde el conjunto es una colección de valores producidos por una clausula select.
El conector notin prueba la no pertenencia al conjunto.
Ejemplo: encontrar todos los clientes que tiennen un prestamo y una cuenta en la sucursal pueyrredon:
En el select mas interno buscamos todos los clientes que tienene cuenta en pueyrredon, y en el mas esterno los que tienen prestamos:

SELECT DISTINCT nombre-cliente FROM préstamo WHERE nombre-sucursal = "pueyrredon" AND
Nombre-cliente IN (SELECT nombre-cliente FROM deposito WHERE nombre-sucursal = "pueyrredon")

Clientes que tienen una cuenta en pueyrredon pero no tienen un prestamo en la sucursal pueyrredon:

SELECT DISTINCT nombre-cliente FROM deposito WHERE nombre-sucursal = "pueyrredon" AND
Nombre-cliente NOT IN (SELECT nombre-cliente FROM prestamo WHERE nombre-sucursal = "pueyrredon")

- **Variables de tupla**

Una variable de tupla debe estar asociada con una tabla determinada, se definen en la cláusula from colocándola después del nombre de la tabla con la cual esta asociada por uno o más espacios.
Son útiles para comparar tuplas de la misma tabla.
Ejemplo: clientes que tienen una cuenta en la misma sucursal en la Jones tiene una cuenta:

SELECT DISTINCT T.nombre-cliente FROM deposito S, deposito T
WHERE S.nombre-cliente = "Jones" AND S.nombre-cliente = T.nombre-sucursal

O

SELECT DISTINCT T.nombre-cliente FROM deposito WHERE nombre-sucursal IN
(SELECT nombre-cliente FROM deposito WHERE nombre-cliente = "Jones"

- **Comparación de conjuntos**

Las construcciones in, ...some, ...all permiten compara un unico valor con los miembros de un conjunto completo.

*** Alguno:** *SOME (ANY)*➔ podemos reemplazar la expresion mayor que algun por **>some**

Ejemplo: encontrar los nombres de todas las sucursales que tienen un activo mayor que alguna sucursal situada en brooklyn:

Primera opcion: SELECT DISTINCT T.nombre-sucursal FROM sucursalT, sucursal S
WHERE T,activo > S. Activo AND S.ciudad-sucursal = "broolyn"

Usando some: SELECT nombre-sucursal FROM sucursal WHERE activo > SOME
(SELECT activo FROM sucursal WHERE xiudad-sucursal = "brooklyn")

la comparacion >SOME es verdadera si el valor activo de la tupla es mayor que al menos un miembto del conjunto de todos los valores del activo de las sucursales de brooklyn. Tambien se permiten las comparaciones <SOME, ≤SOME, ≥SOME, ≠SOME, =SOME(in).

*** Todos:** *ALL*➔ la frase mayor que todos corresponde a >ALL.

Ejemplo: encontrar los nombres de todas las sucursales que tienen un activo mayor que todas las sucursales de brooklyn:

SELECT nombre-sucursal FROM sucursal WHERE activo >ALL (SELECT activo FROM sucursal WHERE ciudad-sucursal = "Brooklyn")
Tambien se permiten las comparaciones <ALL,≤ALL, ≥ALL, ≠ALL, =ALL.

- **Inclusion***: CONTAINS y NOT CONTAINS*➔ permiten ver si un conjunto contiene, o no, todos los miembro de algún otro conjunto.

Ejemplo: encontrar todos los clientes que tienen una cuenta en todas la sucursales situadas en brooklyn. Para cada cliente tenemos que ver si el conjunto de todas las sucursales en las que el cliente tiene una cuenta contiene el conjunto de todas las sucursales de brooklyn:
SELECT DISTINCT S.nombre-cliente FROM deposito S WHERE (SELECT T.nombre-sucursal FROM deposito T WHERE S.nombre-cliente = T.nombre-cliente) CONTAINS (SELECT nombre-sucursal FROM sucursal WHERE ciudad-sucursal = "brooklyn")

- **Testear por relaciones vacias:** *EXISTS*

Sirve para probar si la subconsulta tiene alguna tupla. La construccion EXISTS devuelve true si la subconsulta no esta vacia.

Ejemplo: encontrar a todos los clientes que tienen una cuenta y un prestamo en la sucursal pueyrredon

SELECT nombre-cliente FROM cliente WHERE EXISTS (SELECT * FROM deposito
WHERE deposito. Nombre-cliente = cliente.nombre-cliente AND nombre-sucursal = "pueyrredon")
AND EXISTS (SELECT * FROM prestamo WHERE prestamo. Nombre-cliente = cliente.nombre-cliente AND nombre-sucursal =" pueyrredon")

La primer subconsulta exists prueba si el cliente tiene una cuenta en la sucursal pueyrredon, y la segunda si el cliente tiene un prestamo en esa sucursal.

Ejemplo 2: encontrar a todos los clientes que tienen una cuenta pero no un prestamo en la sucursal pueyrredon:

SELECT nombre-cliente FROM deposito WHERE EXISTS (SELECT * FROM deposito
WHERE deposito. Nombre-cliente = cliente.nombre-cliente AND nombre-sucursal = "pueyrredon")
AND NOT EXISTS (SELECT * FROM prestamo WHERE préstamo. Nombre-cliente = cliente.nombre-cliente AND nombre-sucursal =" pueyrredon")

- ***Ordenacion de tuplas***➔ se ofrece un cierto control de cómo van a ser presentadas las tuplas de una tabla, mediante la clausula ORDER BY (operación costosa). Por omisión los elementos se listan en orden ascendente , pero podemos especificar DESC para orden descendente.

Ejemplo: listar los prestamos en orden descendente de cantidad y si varios tienen la misma cabtidad, orden ascendente por numero de préstamo:
SELECT * FROM préstamo ORDER BY cantidad DESC, numero-prestamo ASC.

La cláusula GROUP BY posibilita calcular funciones en grupos de tuplas, el/los atributo/s dados en esta clausula se usan para formar grupos (las tupolas con igual valor en el atributo forman un grupo).

- **Comprobación de tuplas duplicadas.**
  Unique: devuelve verdadero si la subconsulta que se para como argumento no produce tuplas duplicadas.
  Ej: clientes que tienen una sola cuenta en la sucursal llamada XXX.

Ordenación de las tuplas*: ORDER BY, ASC, DESC*
SELECT * FROM prestamo WHERE monto > 100 ORDER BY monto DESC, numerocuenta ASC

Por omission sql ordena de manera ascendente.

- **Funciones de agregación**→ operan sobre grupos de tuplas y su resultado es un unico valor.
  Promedio (AVG), mínimo (MIN), máximo (MAX), total (SUM), cantidad (COUNT).
  COUNT puede usar la palabra clave DISTINCT para contar las tuplas eliminando las repeticiones.
  Podemos aplicar condiciones a grupos formados por group by mas que a las tuplas mediante la clausula HAVING. Los predicados de having se aplican luego de formar los grupos por lo que podemos usar funciones de agregacion.
NO se pueden componer funciones de agregación: MAX(AVG(...)) es inválido. Se soluciona con HAVING

Ejemplo1: encontrar el saldo promedio de las cuentas en todas las sucursales.
SELECT nombre-sucursal, AVG (saldo) FROM deposito GROUP BY nombre-sucursal

Ejemplo2: Encontrar el número de clientes con deposito para cada sucursal (contamos el cliente una sola vez, sin considerar la cantidad de cuentas que tenga).
SELECT nom_sucursal, COUNT (DISTINCT nombre_cliente) FROM deposito
GROUP BY nom_sucursal

Ejemplo3: listar sucursales con saldo promedio de las cuentas > que 1200.
SELECT nombre-sucursal, AVG (saldo) FROM deposito GROUP BY nombre-sucursal
HAViNG AVG (saldo) > 1200

Ejemplo4: encontrar aquellas sucursales con saldo promedio mayor.
SELECT nombre-sucursal, AVG (saldo) FROM deposito GROUP BY nombre-sucursal
HAViNG AVG (saldo) >= ALL(SELECT AVG (saldo) FROM deposito GROUP BY (nombre-sucursal))

La palabra clave ALL especifica retencion de duplicados.
La funcion COUNT permite contar numero de tuplas (COUNT(*) para contar todas las tuplas.
Si en una misma consulta aparecen WHERE y HAVING, primero aplicamos el predicado de WHERE, las clausulas que lo satisfacen son colocadas en grupos por GROUP BY y luego se aplica HAVING a cada grupo. Los grupos que satisfacen la clausula HAVING son usados por la clausula SELECT para generar tuplas del resultado de la consulta.
Si no hay HAVING el conjuntoo que satisface el WHERE se trata como un grupo unico.

Ejemplo5: Encontrar el saldo promedio de todos los clientes con depositos que viven en Harrison y tienen por lo menos 3 cuentas".
SELECT AVG (saldo) FROM deposito, cliente
WHERE deposito.nombre_cliente = cliente.nombre_cliente AND ciudad_cliente = "Harrison"
GROUP BY deposito.nombre_cliente HAVING COUNT (DISTINT numero_cuenta) >= 3

- **Modificación de la base de datos**

Eliminación: *DELETE*→ podemos suprimir tuplas completas. Opera sobre una sola relación.
DELETE relacion [WHERE P]
Durante la ejecución de una solicitud de DELETE solamente marcamos las tuplas que se van a suprimir, no las suprimimos realmente y cuando hallamos terminado de procesar la solicitud entonces suprimimos las tuplas marcadas. Esta regla garantiza la interpretación consistente de la supresión.

Inserción*:* *INSERT INTO*➔ para insertar datos especificamos una tupla, o una consulta cuyo resultado es un conjunto de tuplas que se van a insertar. Los valores de la tupla se listan en el orden en que se listan los atributos correspondientes en la tabla. Tambien podemos especificar los atributos como parte de la sentencia.

Ejemplo1: insertar el hecho de que SMITH tiene 1200 en la cuenta 9732 en pueyrredon:

INSERT INTO deposito VALUES ("pueyrredon",9732, "smith",1200)

O

INSERT INTO deposito (nombre-sucursal, numero-cuenta, nombre-cliente, saldo) VALUES ("pueyrredon",9732, "smith",1200)

Usar select para especificar un conjunto de tuplas.

Ejemplo2: dar a todos los clientes con prestamos en la sucursal pueyrredon una cuenta de 200 dolares.

INSERT INTO deposito SELECT nombre-sucursal, numero-prestamo, nombre-cliente, 200 FROM prestamo WHERE nombre-sucursal = "pueyrredon"

Actualización*:* *UPDATE*➔sirve para cambiar un valor en la tupla sin cambiar todos los valores en la tupla. Podemos seleccionar las tuplas que van a actualizarse usando una consulta.

Ejemplo1: incrementar todos los saldos en un 5 por 100:

UPDATE deposito SET saldo = saldo * 1.05

Ejemplo2: las cuentas con saldos mayores a 10000 reciben el 6 por 100 de interes y las demas un 5 por 100

UPDATE deposito SET saldo = saldo * 1,06 WHERE saldo > 10000
UPDATE deposito SET saldo = saldo * 1.05 WHERE saldo <= 10000

- **Valores nulos**

Es posible asignar valores a algunos atributos del esquema, el resto son asignados a valores nulos representados por NULL.
Todas las comparaciones que implican NULL son falsas.
Para probar si hay un valor nulo usamos IS NULL e IS NOT NULL para probar la ausencia.
Las funciones de agregación (excepto COUNT) ignoran valores nulos en los argumentos.

Ejemplo: encontrar todos los clientes que aparecen en la tabla prestamo con valores nulos para cantidad:

SELECT DISTINCT nombre-cliente FROM deposito WHERE cantidad IS NULL

- **Vistas**

Creación de vistas➔ se definen usando la orden **CREATE VIEW**, dando un nombre y declarando la consulta que calcula la vista.

CREATE VIEW nombre-vista AS (expresion-consulta)

Donde <expresion-consulta> es cualquier expresion de consulta permitida.

Ejemplo: hacer una vista con todos los nombres de sucursales y de los nombres de los clientes, y se llama todos-clientes:

CREATE VIEW todos-clientes AS (SELECT nombre-sucursal, nombre-cliente FROM deposito)
UNION (SELECT nombre-sucursal, nombre-cliente FROM prestamo)

Podemos usar el nombre de la vista como una tabla.

Ejemplo: encontrar todos los clientes de pueyrredon:

SELECT nombre-cliente FROM todos-clientes WHERE nombre-sucursal = " pueyrredon"

Pueden darse anomalías al modificar vistas: en un update se completaría con NULLs.
Sólo se permite UPDATE, INSERT o DELETE sobre vistas si la vista esta dedinida en terminos de una relación de la BD relacional real (BD del nivel conceptual). Estas operaciones estarian prohibidas en una vista por ejemplo en todos-clientes.

Definición de datos: CREATE TABLE➔ el conjunto de relaciones de una BD debe ser especificado al sistema por medio de un lenguaje de definicion de datos DDL. El SQL DDL permite la especificacion de un conjunto de relaciones y tambien la informacion sobre cada relación incluyendo:

- El esquema de cada relcion
- El dominio de valores asociado con cada atributo
- El conjunto de indices que se va a mantener para cada relación
- La informacion de seguridad y autorizacion para cada relación
- Limites de integridad
- La estructura fisica de almacenamiento de cada relación del disco.

Definicion de una relación➔ CREATE TABLE nombre-relacion (A_1 D_1, ..., A_n D_n)

A_i= nombre del atributo, D_i = el dominio del atributo). Tambien incluye opciones para dar restricciones de integridad.

Para eliminar toda la informacion sobre la relación sacada de la DB usamos la orden DROP TABLE r , que elimina todas las tuplas en r y el esquema de r.

La orden DELETE r elimina las tuplas de r pero mantiene el esquema de r.

ALTER TABLE se usa para añadir atributos a una relación existente y a todas las tuplas en la relación se les asigna null como valor. Su forma es: ALTER TABLE r ADD nom-atributo-nuevo dominio-atributo-nuevo

Unión de relaciones

Inner Join: producto natural entre atributos que se indican, queda el atributo en común repetido

Left outer Join: primero se calcula el inner join (idem anterior) y luego cada tupla t perteneciente a la relación de la izquierda que no encontró par aparece en el resultado con valores nulos en los atributos del segundo lado.

Right outer Join: idem anterior pero aparecen las tuplas t de la relación de la derecha

Full outer join: aparecen las tuplas colgadas de ambos lados.

Natural: evita que el atributo común (por el que se hace la unión aparezca dos veces.

Query-by-example (QBE)

Tiene una sintaxis bidimensional para expresar una consulta.
Las consultas se expresan dando un ejemplo de lo que se desea obtener como respuesta, el sistema generaliza este ejemplo para calcular la respuesta a la consulta.

Estructura basica

Las consultas se expresan usando tablas de esqueletos que muestran el esquema de la relación. El usuario toa el esqueleto que necesita para la consulta y los rellena con "filas ejemplo" (formada por constantes y "elementos ejemplo" ,variables de dominio que van precedidas por _).

Consultas sencillas

1.Encontrar los clientes que tienen una cuenta en la sucursal pueyrredon ➔ El sistema busca en deposito las tuplas que tengan el valor pueyrredon en el atributo nombre-sucursal, para cada una de esas tuplas el valor del atributo nombre-cliente se asigna a la variable x y este valor se imprime porque la orden P aparece en la columna nombre-cliente al lado de la variable x.
2. QBE elimina duplicados automaticamente ➔ para suprimir esta eliminacion insertamos la orden ALL despues de la orden P
3. Listar la relación deposito completa
4. Encontrar el numero de cuenta de todas las cuentas con un saldo mayor de 1200 ➔ Las comparaciones pueden implicar solo una expresion aritmetica en la parte derecha de la operación de comparacion (>_x + _y - 20. La expresion puede incluir varias variables y constantes.
5.Encontrar los nombres de las sucursales que no sean de brooklyn:

Sucursal	Nombre-sucursal	activo	Ciudad-sucursal
	p.		¬brooklyn

6.Encontrar todos los clientes que tienen una cuenta en pueyrredon y en redwood
7.Encontrar todos los clientes que tienen una cuenta en pueyrredon, en redwood o en ambas.
8.Encontrar todos los clientes que tienen cuenta en la misma sucursal que Jones

	Deposito	Nombre-sucursal	Numero-cuenta	Nombre-cliente	Saldo
1		Pueyrredon		P . _x	
2				P.ALL	
3	P.				
4			P.		>1200
6		Pueyrredon redwood		P._x _.x	
7		Pueyrredon redwood		p._x P._y	
8		_x _y		Jones	

Consulta sobre varias relaciones

Las conexiones entre las diversas relaciones se logran a traves de variables que fuerzan a determinadas tuplas a tener el mismo valor en determinados atributos.

1. Encontrar el nombre y la ciudad de todos los clientes que tiene un prestamo de la sucursal Pueyrredon

el sistema encuentra en prestamo las tuplas con el valor pueyrredon y para cada una de ellas encuentra en cliente las que tienen el mismo atributo nombre de cliente que la tupla ptrestamo. Presenta los atributos nombre-cliente y ciudad-cliente.

Prestamo	Nombre-sucursal	Nombre-prestamo	Nombre-cliente	Cantidad
	pueyrredon		_x	

Cliente	Nombre-cliente	calle	Ciudad-cliente
	P._x		P._y

2.Encontrar los nombres de todos los clientes que tiene una cuenta y un prestamo en la sucursal pueyredon

Deposito	Nombre-sucursal	Numero-cuenta	Nombre-cliente	saldo
	Pueyredon		P._x	

Prestamo	Nombre-sucursal	Numero-prestamo	Nombre-cliente	cantidad
	pueyrredon		_.x	

3.Encontrar los monbres de todos lod clientes con cuenta en pueyrredon pero no un prestamo alli. Las consultas que implican negacion se expresan colocando un signo ¬ debajo del nombre de la relación.

Deposito	Nombre-sucursal	Numero-cuenta	Nombre-cliente	saldo
	Pueyredon		P._x	

Prestamo	Nombre-sucursal	Numero-prestamo	Nombre-cliente	cantidad
¬	pueyrredon		_.x	

4.Encontrar los nombres de los clienetes que tiene una cuenta y un prestamo en pueyrredon.¬ubicado debajo del atributo significa distinto, y debajo del nombre de la relación significa no existe (presentar todos los valores de nombre-cliente que aparecen por lo menos en dos tuplas, con la segunda tupla que tenga un nombre-sucursal diferente del primero)

Deposito	Nombre-sucursal	Numero-cuenta	Nombre-cliente	saldo
	_y ¬_y		P._x _x	

Cuadro de condiciones

Las comparaciones no pueden impicar dos variables distintas por lo que usamos cuadro de condiciones para expresar los limites.

Restricciones ejemplo:

	Condiciones
Clientes con nombre distinto de jones	X¬=jones
Todas las cuentas con saldo entre 1300 y 1500	_x>=1300 _x<=1500
Sucursales con activos mayores que alhuna sucursal situada en brooklyn	_Y>_z
Numeros de cuenta con saldo entre 1300 y 200 pero no 1500	_x=(>=1300 and 200 and ¬1500)
Sucursales situadas en brooklyn o queens	*_x=(brooklyn or queens)*
Sucursales con activos por lo menos el doble de los de alguna sucursal de brooklyn	*y>=2*z*

La relación resultado

Si queremos presentar atributos de varios esquemas de relación tenemos que declarar la relación temporal resultado que incluye los atributos del resultado de la consulta.

1. Crear la tabla resultado con los atributos necesarios
2. Escribimos la consulta.

Ordenacion de la presentacion de tuplas

Para controlar el orden en el que se presentaran las tuplas colocamos la orden AO (orden ascendente) DO(orden descendente) en la columna apropiada.
Para aplicar dos ordenes debemos especificarlos (P>AO(1) P.DO(2))

Funciones de agregacion

Incluye los operadores AVG, MAX, MIN, SUM, CNT que deben estar seguidos de ALL para asegurar que todos los valores son considerados. Ejemplo P.SUM.ALL, si queremos eliminar los duplicados debemos agregar UNQ P.CNT.UNQ.ALL.
Para calcular funciones en grupos de tuplas usamos el operador G analogo a GROUP BY (P.G). tambien podemos armar condiciones: AVG.ALL._x >1200 o CNT.ALL._x=0 o CNT.UNQ._y = CNT.UNQ._z (cuenta y luego compara).

Modificacion de la BD

SUPRESION➔ se realiza igual que una consulta pero ponemos D (opera sobre una relación unicamente) en vez de P.
Eliminar los registros de cuentas de smith

Deposito	Nombre-sucursal	Numero-cuenta	Nombre-cliente	Saldo
D			smith	

Eliminar el valor ciudad-sucursal con nombre pueyrredon

Sucursal	Nombre-sucursal	activo	Ciudad-sucursal
	pueyrredon		D

Eliminar prestamos con numero entre 1300 y 1500

Prestamo	Nombre-sucursal	Numero-prestamo	Nombre-cliente	Cantidad
D.		_x		

Condiciones: *x=(>=1300 y <=1500*)
Eliminas todas las cuentas de las sucursales de brooklyn

deposito	Nombre-sucursal	Numero-cuenta	Nombre-cliente	saldo
D.	_x			

Sucursal	Nombre-sucursal	activo	Ciudad-sucursal
	_x		Brooklyn

INSERTAR➔ podemos especificar la tupla a insertar o escribir una consulta para calcularlas. La insercion se hace colocando el operador I
Insertar el hecho de que smith tiene 1200 en la cuenta 9732 en pueyrredon

Deposito	Nombre-sucursal	Numero-cuenta	Nombre-cliente	Saldo
I	Pueyrredon	9732	smith	1200

Para inertar basandome en una consulta: dar a los clientes de prestamos en pueyrredon una cuenta de 200 y el numero de prestamo sera el numero de cuenta.

Deposito	Nombre-sucursal	Numero-cuenta	Nombre-cliente	Saldo
I	pueyrredon	_x	_y	200

prestamo	Nombresucursal	Numero-prestamo	Nombre-cliente	Cantidad
	Pueyredon	_x	_y	

Actualizaciones➔ usamos el operador U. Podemos elegir las tuplas a actualizar por medio de una consulta. No se permite actualizar los campos clave primaria.

Actualizar el acivo de pueyrredon a 10000000

Sucursal	Nombre-sucursal	Activo	Ciudad-sucursal
	pueyrredon	U.10000000	

Aumentar los saldos en un 5 % (cada vez que recuperamos una tupla de deposito, determinamos el saldo _x y lo actualizamos _x*1,05

Deposito	Nombre-sucursal	Numero-cuenta	Nombre-cliente	saldo
U.			_x*1,05 _x	

Unidad 6➔ control y seguridad de los datos

RECUPERACIÓN Y ATOMICIDAD

Un sistema de computadores, está sujeto a fallos. Existen varias causas de las fallas: roturas de disco, corte de energía, sabotaje, etc. En cada uno de estos casos se pierde información referente al sistema de base de datos.
Una parte integral de un sistema de BD es un ***esquema de recuperación*** que es responsable de ***la eliminación de fallos y de la restauración de la BD a un estado consistente*** que existía antes de que ocurriera el fallo.
Es fundamental para la consistencia de la BD que las operaciones que formen una unidad de trabajo ocurran por completo o no ocurran. Este requisito de ***"todo o nada"*** se denomina ***atomicidad***.

Clasificacion de fallos

Mas sencillo→aquel que no resulta en perdida de informacion en el sistema.
Mas dificil→ resultan en perdida de informacion
Para determinar como se debe recuperar el sistema de los fallos necesitamos identiicar los mmodos de fallo de los dipositovos usados para almacenar los datos y luego analizar como afectan estos modos de fallos a los contenidos de la BD y proponer algoritmos para asegurar la consistencia de la BD y la atomicidad de las transacciones a pesar de fallos. Los algoritmos tienen dos partes:

- Acciones tomadas durante el procesamiento normal de transacción para asegurar información suficiente para la recuperación.
- Acciones tomadas luego de un fallo para asegurar la consistencia de la BD y la atomicidad de las transacciones.

Tipos de almacenamiento

Los medios de almacenamiento se distinguen pos su velocidad relativa, capacidad y resistencia a los fallos.

- ✓ Volátil: la información que reside en memoria volátil no sobrevive a las caídas del sistema (Ej: memoria principal y caché). El acceso a memoria volatil es rapido.
- ✓ No Volátil: aunque sobreviven a las caídas del sistema, está sujeta a fallos que pueden resultar en pérdida de información. (ej: discos y cintas)
- ✓ Estable: nunca se pierde información.

Tipos de fallos

Existen varios tipos de fallos, algunos que pierden datos y que son los más difíciles de tratar, y otros que no:

- ✓ Errores lógicos: la transacción no puede continuar con su ejecución normal debido a alguna condición externa, como puede ser una entrada inválida, overflow, informacion no localizada, exceder el limite de los recursos.
- ✓ Errores de sistema: El sistema ha entrado en un estado no deseable (bloqueo) y la transaccion no puede continuar con su ejecucion normal pero si puede volverse a ejecutar mas tarde.
- ✓ Caída del sistema: el hardware funciona mal y causa la perdida del contenido del almacenamiento volatil el no volatil permanece intacto.
- ✓ Fallo de disco: rotura de disco. Se pierde información en el disco.

Algunos sistemas proporcionan recuperacion de bateria de forma que parte de la memoria principal pueda sobrevivir a la caida del sistema y cortes del suministro de energia. El almacenamiento no volatil, como medios opticos, es una alternativa con mayor grado de fiabilidad.
Para determinar como debe recuperarse el sistema de los fallos podemos proponer algoritmos para asegurar la consistencia de la BD y la atomicidad de las transacciones a pesar de los fallos. Estos algoritmos tienen dos partes:

- Acciones tomadas ***durante*** el procesamiento normal
- Acciones tomadas ***a continuación*** de un fallo

Jerarquía de almacenamiento

El sistema de BD reside en almacenamiento no volátil (disco). La BD se divide en unidades de longitud fija llamadas **bloques** que son las unidades de asignación de almacenamiento y de transferencia de datos. Las transacciones introducen información del disco en la memoria principal y luego la retornan al disco.
Las operaciones de entrada y salida se hacen en unidades de bloques. Los bloques que residen en el disco se denominan ***bloques físicos***, mientras que los bloques que residen temporalmente en la memoria principal se denominan ***bloques de registros intermedios (buffer)***.
Un bloque puede contener varios datos. Suponemos que ningun dato se extiende sobre dos o mas bloques
Los movimientos de bloques entre el disco y la memoria principal se realizan por medio de dos operaciones:

- Input(x), que transfiere el bloque físico en el que reside el elemento de información x a la memoria principal.
- Output(x), que transfiere el bloque de registro intermedio en el que reside x al disco y sustituye el bloque físico apropiado que allí se encuentra.

Un bloque de registro intermedio se graba en disco porque se necesita el espacio de emoria o porque el sistema de BD quiere reflejar el cambio hecho en X en el disco. El sistema de BD fuerza la salida del bloque en el registro intermedio que contiene a X si se ejecuta la operación OUTPUT(X)
Las transacciones interactúan con el sistema de BD transfiriendo los datos desde variables del programa a la BD y desde esta a variables del programa. Para realizar estas transferencia se utilizan:

- **Read (X, xi)**, que asigna el valor del elemento de información x a la variable local xi. Esta operación se ejecuta como sigue:
    1. Si el bloque en el que reside X no está en la memoria principal, se ejecuta input(X)
    2. Se asigna a xi el valor de X del bloque en el buffer
- **Write(X, xi)**, que asigna el valor de la variable local xi al elemento de información X del bloque que está en el buffer. Esta operación se ejecuta como sigue:
    1. Si el bloque en el que reside X no está en la memoria principal, se ejecuta un input(X).
    2. Se asigna el valor de xi a X en el bloque del buffer que contiene a X.

EjecutamosREAD(X,xi) cuando queremos acceder a un elemento de informacion de X por primera vez, y luego todas las actualizaciones se realizan sobre xi. Luego del ultimo acceso a X se ejecuta WRITE(X,xi) para reflejar el cambio en la BD.
La operación OUTPUT(X) no necesita ejecutarse inmediatamente después de WRITE(X,xi), ya que el bloque en que reside X puede contener otros elementos de información a los que se está accediendo todavía. Si el sistema se cae después de WRITE(X,xi) pero antes de OUTPUT(X), el nuevo valor de X nunca se escribe en el disco y por lo tanto se pierde.

Modelo de transacción

DEFINICION → colección de operaciones que forman una única unidad lógica de trabajo.Las transacciones deben ser atómicas, es decir, se llevan a cabo por completo o no se ejecutan.

La Base de Datos debe ser consistente antes y después de una transacción, aunque durante su ejecución puede ser necesario permitir inconsistencia temporalmente. Esta inconsistencia temporal puede llevar a dificultades si ocurre un fallo.
Ejemplo Bancario:
Sea T una transacción que transfiere 50 dólares de la cuenta A a la cuenta B

```
T: read (A, a1)          A = 1000
   a1:= a1 – 50          B = 2000
   write (A,a1)
   read (B, b1)
   b1:= b1 + 50
   write (B,b1)
```

La restricción de consistencia es que la ejecución no cambie la suma de A y B.
Antes de ejecutar T los valores son A=1000 y B=2000. Durante la ejecución de T, los valores de a1 y b1 se cambian a 950 y 2050 dólares respectivamente.
Después de que se ejecutan las operaciones write pero antes que se ejecutan las operaciones output los valores de A y B en el bloque físico y en el bloque del registro intermedio son diferentes.
Supongamos que durante la ejecución de T ha ocurrido una falla que impidió que T completara su ejecución con éxito, y que esto ocurrió después de que se ejecutó output(A). En este caso, los valores de las cuentas A y B reflejadas en la BD del disco son 950 y 2000 dólares, perdimos 50 como resultado de este fallo y la suma A + B no se conserva.
Como resultado del fallo, el estado del sistema ya no reflejará un estado real del mundo que se supone que capte la BD, y a esto se le llama *estado inconsistente*. Aún cuando la transacción T se ejecute completamente, existe un momento en el que el valor de A es 950 dólares y el valor de B es 2000 dólares, lo que es claramente un estado inconsistente.
El programador es el responsable de definir adecuadamente las diversas transacciones de forma que cada una conserve la consistencia de la BD.

Correccion y atomicidad

Requisitos para las transacciones:

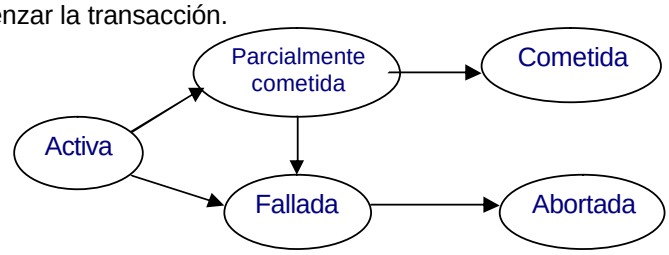
- **Ejecucion correscta→** cada transaccion debe ser un porgrama que conserve la consistencia de la BD.
- **Atomicidad→** todas las operaciones asociadas a una transaccion deben ejecutarse por completo o no ejecutarse
- **Atomicidad →** todas las operaciones de la transacción se ejecutan o no lo hacen ninguna de ellas.

- **Consistencia** ➔ la ejecución aislada de la transacción conserva la consistencia de la BD.
- **Aislamiento (isolation)** ➔ cada transacción ignora el resto de las transacciones que se ejecutan concurrentemente en el sistema, actúa c/u como única.
- **Durabilidad** ➔ una transacción terminada con éxito realiza cambios permanentes en la BD, incluso si hay fallos en el sistema.

La ejecución correcta es responsabilidad del programador y la atomicidad es responsabilidad de la BD.
Si una transacción no termina exitosamente se dice que se ABORTO. Para garantizar la atomicidad una transacción abortada no debe afectar al estado de la BD. El estado de la BD debe restaurarse al estado en que estaba antes de que comenzara a ejecutarse la transacción abortada. Por esto decimos que la transacción retrocede.
Una transacción que termina su ejecución con éxito se llama COMETIDA (deja la BD en un nuevo estado consistente. Su efecto no puede deshacerse abortando una transacción, sino que hay que ejecutar una transacción COMPENSADORA.

Estado de una transacción:

Una transaccion debe estar en uno de los siguientes estados:
- ***Activa***: estado inicial, estado normal durante la ejecución.
- ***Parcialmente cometida***: después que se haya ejecutado la última sentencia.
- ***Cometida***: después de la terminación con éxito.
- ***Fallada***: después de descubrir que la ejecución normal no puede proceder.
- ***Abortada***: después que la transacción haya retrocedido y se haya restaurado el estado de la BD al que tenía antes de comenzar la transacción.



Una vez que una transaccion esta terminada puede procesarse una nueva.
Una transacción empieza en el estado **activo**, cuando llega a su última sentencia entra en el estado **parcialmente cometido**. En ese momento, ha terminado su ejecución, pero todavía es posible que tenga que abortarse, ya que puede pasar que la salida real aún no se haya escrito en el disco y un fallo de hardware puede impedir la terminación con éxito. Por estose dee tener cuidado con las escrituras externas observables (no pueden borrarse= terminal o impresoras), por lo que solo se permite realizarlas despues que una transaccion haya entrado en ele estado cometido.
Una transacción entra en el estado **fallado** después que se determina que no puede proceder con su ejecución normal. En este caso debe retroceder y pasa al estado abortado y ahora el sistema tiene dos opciones:

Reiniciar la transacción ➔ se hace sólo si se abortó por un error de hard o soft y no por error de lógica del programa. Una transaccion reiniciada se considera nueva.

Eliminar la transacción➔ se hace cuando hubo error de lógica del programa, que se corrige con la reescritura del mismo, o hubo una mala entrada o los datos no se encontraron en la BD.

Una transacción entra en estado cometido si se garantiza que nunca se abortara.

Recuperación basada en Bitácora

Para logara atomicidad debemos sacar informacion describiendo las modificaciones que hizo la transaccion cometida a pesar de fallos.

La bitacora de la BD

BIATCORA ➔ estructura usada para grabar las modificaciones de la BD.Cada registro de la bitácora describe una única operación de escritura de la BD y tiene los siguientes campos:

- ✓ Nombre de la transacción: el nombre único de la transacción que realizó la operación write.
- ✓ Nombre del dato: el nombre único del dato escrito.
- ✓ Valor antiguo: el valor del dato anterior a la escritura
- ✓ Valor nuevo: el valor que tendrá el dato después de la escritura.

Hoay registros especiales para grabar sucesos importantes durante el procesamiento de la transaccion (, cometido o aborto comienzo de la transaccion. Los distintos tipos de registros tienen la siguiente representación:

- ***<Ti start>*** : la transacción Ti a empezado
- ***<Ti, xj, v1, v2>*** : la transacción Ti ha realizado una escritura en el dato xj. xj tenía el valor v1 antes de la escritura y v2 después de la escritura.
- ***<Ti commit>*** : la transacción Ti ha terminado. Se pone cuando está en estado de parcialmente cometida.

Siempre que una transacción realice una escritura se debe crear el registro de bitácora para esa escritura antes de que se modifique la BD.
También tenemos la posibilidad de deshacer una modificación que ya se a escrito en la BD, tomando el campo del valor antiguo de los registros de la bitácora.
Cuando el DBMS advierte que alguna transacción no terminó, va a la bitácora para recuperarla.

- Para que los registros de bitácora sean útiles en la recuperación de fallas del sistema y del disco, el registro debe residir en memoria estable. Además, entes de guardar la información del buffer al disco, el SO debe guardar antes el buffer de la bitácora.
- La bitacora contiene un registro completo de toda la actividad de la BD por lo que el volumen de datos almacenados en la bitacora puede llegar a ser exageradamente grande.

Dos políticas para asegurar la atomicidad.

✵ Modificación diferida de la base de datos (grabo después del commit)

Consiste en demorar las escrituras de la BD hasta llegar al estado de parcialmente cometida (graba las modificaciones de la BD en la bitácora pero aplaza la ejecución de todas las operaciones WRITE de una transacción hasta que la transacción este parcialmente cometida).
Esta técnica garantiza la atomicidad de la transacción. Si el sistema se cae antes de que la transacción termine su ejecución, o si la transacción aborta, entonces se ignora la información de la bitácora.

La ejecución de la transacción Ti se lleva a cabo de la siguiente manera:
Antes de que Ti comience a ejecutarse, se graba un registro <Ti START> en la bitácora. Durante la ejecución, cualquier operación WRITE(X,xi) que lleve a cabo Ti resultará en la grabación de un registro más en la bitácora, cada uno de ellos conteniendo los campos nombre de transacción, nombre de dato y nuevo valor.
Finalmente, cuando Ti esté parcialmente cometida, se grabará un registro <Ti COMMIT> en la bitácora.
Cuando la transacción Ti esté parcialmente cometida, los registros que le corresponden en la bitácora se utilizan para ejecutar las operaciones de grabación postergadas. Como puede ocurrir una falla mientras se lleva a cabo esta actualización, debe garantizarse que, antes de comenzar dicha actualización, todos los registros de la bitácora se hayan grabado en el almacenamiento estable. Una vez hecho esto, puede llevarse a cabo la actualización en sí. En este momento, la transacción entrará en el estado cometido.
La técnica de modificación diferida solamente requiere el nuevo valor del dato, por lo que podemos simplificar la estructura de la bitácora omitiendo el campo del valor antiguo.

Con el ejemplo del banco: sea To una transaccion que transfiere 50 dolares de A a B, y se define asi:

```
To:   READ(A,a1)
      A1:=a1-50
      WRITE(A,a1)
      READ(B,b1)
      B1:=b1+50
      WRITE(B,b1)
```

Sea T1 una transacion que retira 100 dolares de la cuenta C, y se define:

```
T1:   READ(C,c1)
      C1:=c1-100
      WRITE(C,c1)
```

Se ejecuta T0 y luego T1, los valores de A=1000, B=2000 y C=700, la bitacora quedaria

```
<To STARTS>
<To A, 950>
<To B, 2050>
<To COMMITS>
<T1 STARTS>
<T1 C, 600>
<T1 COMMITS>
```

Las salidas reales al sistema de BD y al registro de bitacora:

Bitacora	BD
<To STARTS>	
<To A,950>	
<To B, 2050>	
<T0 commits>	
	A=950
	B=2050
<t1 STARTS>	
<T1 C, 600>	
<T1 COMMITS>	
	C=600

Empleando la bitácora, el sistema puede manejar cualquier tipo de falla que no resulte en la pérdida de información en almacenamiento no volátil. El procedimiento de recuperación utiliza la siguiente operación:

- ***Redo(Ti)***, que asigna los nuevos valores a todos los datos que actualiza la transacción Ti.

El conjunto de los datos que Ti actualiza y sus valores nuevos respectivos pueden encontrarse en la bitácora.
La operación REDO debe ser idempotente, es decir, que ejecutarla varias veces debe ser equivalente a ejecutarla una vez. Esto es necesario para garantizar un comportamiento correcto aunque ocurra una falla durante el proceso de recuperación.
Después de ocurrir una falla, el subsistema de recuperación consulta la bitácora para determinar cuáles transacciones hace falta repetir. La transacción Ti debe repetirse si la bitácora contiene tanto el registro <Ti START> como el registro <Ti COMMIT>. Así, si el sistema se cae después que la transacción terminó su ejecución, la información de la bitácora sirve para volver el sistema a un estado consistente previo.
Finalmente, se considerará el caso en que ocurre una segunda caída del sistema durante la recuperación de la primera caida. Es posible que se hayan hecho algunos cambios a la base de datos como resultado de las operaciones REDO, pero también puede ser que no se hayan efectuado todos los cambios. Cuando el sistema vuelve a la normalidad después de la segunda caída, la recuperación se lleva a cabo de la misma manera. Por cada registro del tipo: <Ti COMMIT> que se encuentre en la bitácora, se realiza la operación REDO(Ti). Dicho de otro modo, el proceso de recuperación se reinicia desde el principio. Puesto que REDO graba valores en la base de datos independientemente de los valores que existan en ella, el resultado del segundo intento de REDO es el mismo que si REDO se hubiera logrado la primera vez.

❋ Modificación inmediata de la base de datos:

Esta técnica permite que las modificaciones se graben en la BD mientras la transacción está todavía en el estado activo (las modificaciones de datos que escriben las transacciones activas se denominan modificaciones no cometidas)**. En el caso de que ocurra una caída o un fallo de una transacción, el campo del valor antiguo de los registros de la bitácora debe utilizarse para restaurar los datos modificados al valor que tenían antes de que comenzara la transacción.**
Esto se realiza mediante la operación UNDO. Antes de que una transacción Ti comience su ejecución, el registro <Ti START> se escribe en la bitácora. Durante su ejecución, cualquier operación WRITE que haga Ti irá precedida de la escritura del registro nuevo en la bitácora. Cuando Ti esté parcialmente cometida, el registro <Ti COMMIT> se escribirá en la bitácora.
No podemos permitir que tenga lugar la actualización real de la BD antes de que se escriba el registro de bitácora correspondiente en almacenamiento estable porque la información de la bitácora se usa para reconstruir el estado de la BD. Por esto se exige que antes de ejecutar una operación OUTPUT(X) se graben en almacenamiento estable los registros de bitácora correspondientes a X.

En el ejemplo del banco, se ejecuta To y luego T1, la parte de la bitacora con informacion reelevante

<To STARTS>
<To A,1000, 950>
<To B, 2000, 2050>
<To COMMITS>
<T1 STARTS>
<T1 C, 700, 600>
<T1 COMMITS>

posible orde en que pudieron hacerse las salidas:

Bitacora	BD
<To STARTS>	
<To A,1000,950>	
<To B, 2000,2050>	
	A=950
	B=2050
<T0 commits>	
<t1 STARTS>	
<T1 C, 700,600>	
	C=600
<T1 COMMITS>	

El esquema de recuperación utiliza dos procedimientos:
Undo(Ti), que restaura todos los datos que Ti actualiza a los valores que tenían anteriormente.
Redo(Ti), que asigna los nuevos valores a todos los datos que actualiza la transacción Ti.

Las operaciones UNDO y REDO deben ser idempotentes, para garantizar su comportamiento correcto aún cuando ocurra un fallo durante el proceso de recuperación.
Después de ocurrir un fallo, el esquema de recuperación consulta la bitácora para determinar que transacciones necesitan volverse a hacer y cuales deshacerse. Esta clasificación se realiza de la siguiente manera:

- ✓ La transacción **Ti debe deshacerse** si la bitácora contiene el registro <Ti START>, pero no el <Ti COMMIT>
- ✓ La transacción **Ti debe volverse a hacer** si la bitácora contiene tanto el registro <Ti START> como <Ti COMMIT>.

Puntos de verificación

Cuando ocurre un fallo del sistema, es necesario consultar la bitácora para determinar cuáles son las transacciones que necesitan rehacerse o deshacerse. Para no tener que revisar toda la bitácora, se introducen ***puntos de verificación (checkPoints),*** que señalan puntos en los cuales puede decirse que lo realizado hasta los mismos es correcto. Es decir, un checkpoint indica que "de acá para atrás está todo bien".
Priodicamente el sistema realiza puntos de verificacion que requieren que se leve a cabo la siguiente secuencia de acciones:

1. Grabar en memoria estable todos los registros de bitaora que actualmente residen en memoria principal.
2. Grabar en disco todos los bloques modificados de los registros intermedios (buffer)
3. Grabar un registro de bitacora <checkpoint> en memoria estable

Despues de un fallo el esquema de recuperación examina la bitácora hacia atrás para determinar cual fue la ultima transacción Ti (busco <Ti START>que comenzó a ejecutarse antes del ultimo punto de verificación
Una vez identificada la transacción Ti, sólo será necesario aplicar las operaciones REDO y UNDO a la transacción Ti y a todas las transacciones Tj que comenzaron a ejecutarse después de Ti.
Opeaciones de recuperacion necesarias si se emplea la tecnica de modificacion inmediata:

- Ejecutar REDO(Tk) para todas las transacciones Tk en T cuyo registro <Tk COMMITS> correspondiente aparezca en la bitacora.
- Ejecutar UNDO Tk) para todas las transacciones Tk en T cuyo registro <Tk COMMITS> correspondiente no aparezca en la bitacora.

Cuando se emplea la tecnica de modificacion diferida no se necesita aplicar la operación UNDO.

✵ Buffers para bitácora

La estrategia de grabar bloques en disco que emplea la generalidad de los sistemas operativos no es compatible con los requisitos del procedimiento de recuperación de caídas. Se establece el requisito adicional de que todos los registros de la bitácora que correspondan a un bloque se graben en almacenamiento estable antes de grabar ese bloque en disco.
Es necesario expresar la siguiente regla relativa al manejo del buffer en sistemas de bases de datos:
Antes de ejecutar una operación de salida de un bloque que está en memoria principal, debe forzarse la salida a almacenamiento estable de todos los registros de bitácora que correspondan a la información que contiene el bloque; si no están ya en almacenamiento estable.

Cuando se graban en almacenamiento estable registros de bitácora, conviene grabar bloques completos de registros de bitácora, ya que casi siempre cuesta lo mismo grabar un bloque que grabar parte de él.

❋ Técnica alternativa: doble paginación (shadow paging):

Es una técnica de recuperación de caídas basada en bitacoras. Bajo ciertas circunstancias, es posible que la doble paginación requiera menos accesos a disco que los métodos de bitácora.
La BD se divide en un cierto numero de bloques de longitud fija que se denominan paginas. Para encontrar la pagina i en la BD usamos la tabla de paginacion.
La idea clave en que se basa la técnica de doble paginación es mantener dos tablas de paginación mientras dura una transacción, la tabla de paginación actual y la de paginación doble.
Cuando se inicia la transacción, las dos tablas de paginación son idénticas. La tabla de paginacion doble no se modifica en ningun momento durante la transaccion. La tabla de paginacion actual puede cambiarse cuando la transaccion realiza una operación WRITE. Todas las operaciones INPUT y OUTPUT utilizan la tabla de paginacion actual para localizar las paginas de la BD en el disco. Suponemos que las transacciones realizan una operación WRITE(X,xi) y que X reside en la pagina i-esima. La operación WRITE se ejecuta de la siguiente manera:

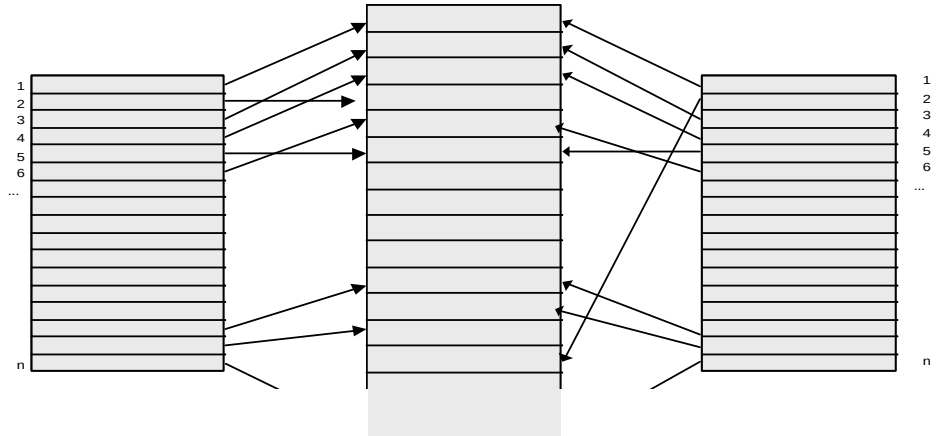
1. Si la pagina i-esima (contiene a X) no esta en memoroa ppal se ejecuta INPUT(X)
2. Si es la primera escritura de esta transaccion en la pagina se modifica la tabla de paginacion actual:
   2.1. Se localiza una pagina libre en disco
   2.2. Se pone esta pagina como ocupada
   2.3. Se modifica la tabla de paginacion actual para que la entrada i-esima apunte a la pagina encontrada.
3. Se asigna el valor de xj a X en la pagina que esta en e registro intermedio (buffer)

Esta técnica consiste en almacenar la tabla de paginación doble en memoria no volátil, de manera que el estado de la base de datos antes de ejecutarse la transacción pueda recuperarse en caso de una caída o aborto de la transacción. Cuando la transacción está cometida, la tabla de paginación actual se graba en memoria no volátil. La tabla de paginación actual se convierte entonces en la nueva tabla de paginación doble, y se permite que comience a ejecutarse una nueva transacción. La tabla de paginación actual puede conservarse en memoria principal, no nos importa si la perdemos porque el sistema se recupera usando la tabla de paginacion doble.
Para que la recuperación tenga éxito necesitamos encontrar la tabla de paginacion doble en el disco. Cuando el sistema vuelve a la normalidad después de la caída, se copia la tabla de paginación doble en memoria principal y se utiliza para procesar las siguientes transacciones.
Se garantiza que la tabla de paginación doble apunta a las paginas de la BD que corresponden a su estado anterior a cualquier transacción que estuviera activa en el momento de la caida.
No se necesita invocar operaciones UNDO.
Operaciones para cometer una transacción:

1. Asegurar de que todas las paginas del registro intermedio (buffer) en memoria ppal que haya modificado la transacción se graben en disco.
2. Grabar en disco la tabla de paginación actual. No debe escribirse sobre la tabla de paginación doble porque podemos necesitarla para la recuperación de una caida.
3. Grabar la dirección en disco de la pagina actual en la posición fija de memoria estable que contiene la dirección de la tabla de paginación doble. Esto se escribe sobre la dirección de la tabla de paginación doble antigua, la tabla actual se convierte en la doble y la transacción esta cometida.

Una caída antes de 3 hace que se vuelva al estado anterior al de ejecutar la transacción y si ocurre después de 3 los efectos de la transacción se conservan.

La doble paginación tiene varias *ventajas* con respecto a la bitácora. Se elimina el tiempo extra requerido para grabar registro de bitácora, y la recuperación de las caídas es bastante más rápida, (**menos accesos a disco**). Sin embargo, tiene otras *desventajas* como son la fragmentación de la información (las paginas de la BD cambian la posicion cuando se actualizan y se pierde la propiedad de localizacion de las paginas o debemos recurrir a esquemas de gestion de almacenamiento complejos y lentos) y la necesidad de "recolectar basura", (periodicamente hay que localizar las paginas basura - contienen informacion inaccesible luego que una transaccion se comete- y anadirlas a la lista de paginas libres. Esto exige tiempo extra y complejidad en el sistema)



❋ Fallo con pérdida de memoria no volátil

Para recuperarse de las fallas que resulten en pérdida de almacenamiento no volátil, es preciso volcamos periodicamente todo el contenido de la base de datos en memoria estable (mejor si es todos los días). Si se presenta una falla que resulte en la pérdida de bloques físicos de la base de datos, se utiliza el volcado más reciente para restaurar la base de datos a un estado consistente previo. Una vez hecho esto, se usa la bitácora para llevar el sistema de base de datos al estado consistente más reciente. No es necesario ejecutar operaciones UNDO
Ninguna transaccion puede estar activa durante el procedimiento de volcado y debe llevarse a cabo un procedimiento similar al de puntos de verificacion:

1. Grabar en memoria estable todos los registros de la bitacora que residen en memoria ppal.
2. Grabar todos los bloques del registro intermedio (buffer) en el disco
3. Copiar los contenidos de la BD en memoria estable
4. Grabar un registro de bitacora <DUMP> en memoria estable.

Desventajas➔ costoso porque la BD debe copiarse en memoria estable (transferencia de datos) y el procedimiento de transacciones se para durante el volcado y se desperdician ciclos de CPU.

❋ Implementación en memoria estable

La información que reside en memoria estable nunca se pierde. Para implementar este tipo de almacenamiento necesitamos repetir la información requerida en varios medios de almacenamiento no volátil (disco) con modos de fallo independientes y actualizar la informacion de manera controlada para que un fallo durante la transferencia no dañe la informacion.
La transferencia de bloques entre memoria y el disco puede resultar en:

- ⏭ Terminacion con éxito→ la informacion llega integra a su destino.
- ⏭ Fallo parciasl→ ocurre un fallo durante la transferencia
- ⏭ Fallo total→ ocurre un fallo durante la transferencia y el bloque destino permanece intacto.

Es necesario que si ocurre un fallo en la transferencia de datos, el sistema lo detecte e invoque a un procedimiento de recuperación que restaure el bloque a un estado consistente. Para hacerlo, el sistema debe mantener dos bloques físicos por cada bloque lógico de la BD. Una operación de salida se ejecuta así:

1. Escribir la información en el primer bloque físico
2. Una vez que se completa con éxito la primera escritura, escribir la misma información en el segundo bloque físico
3. La salida está completa sólo después de terminar con éxito la segunda escritura.

Durante la recuperacion se examina cada par de bloques fisicos, si los dos son iguales no hay errores detectables.
Si un bloque contiene un error detectable, o ambos pero tienen contenido distinto, se sustituye el contenido del primero por el valor del segundo bloque.
Este procedimiento garantiza que una estructura de almacenamiento estable termina con éxito y no produce ningun cambio.

Recuperación

Las solución más común es retroceder una o varias transacciones. Hay que tener en cuenta tres factores:
Elección de la víctima (con menos tiempo de ejec,.....)
Retroceso (hasta donde se hace retroceder a la víctima, en peor caso retroceso total)
Inanición (Evitar elegir siempre la misma víctima)

Seguridad e Integridad

La información debe estar protegida contra accesos no autorizados, alteración, etc.

La pérdida de datos puede ser por:
1-Caída del sistema durante procesamiento de transacción..
2-Problemas en el acceso concurrente a la BD.
3-Problemas por tener datos distribuidos en varios computadores.

4-un error logico que viola la suposicion de que las transacciones respetan las protecciones de consistencia de la BD.

Formas de acceso indebido:

- Lectura de datos sin autorizacion
- Modificacion sin autorizacion
- Destruccion sin autorizacion

Seguridad, generalmente se refiere a acceso no autorizado mientras que integridad se refiere a la posible pérdida de consistencia.

Medidas de protección: se deben asegurar todos los siguientes niveles para asegurar la seguridad de la BD)

Físico

Protección del equipo ante problemas naturales, fallo de energía, etc.
Protección del disco rígido contra robos, borrados, daños físicos
Protección de la red contra daños físicos
Soluciones
Replicar el hardware (discos espejos, múltiples accesos a la red (varios cables))
Seguridad física (policía)
Técnicas de software que aseguren posibles brechas de seguridad

Humano

Tener cuidado al conceder autorizacion a los usuarios para reducir la probabilidad de que un usuario autorizado permita el acceso a un intruso a cambio de sobornos u otros factores. Protegerse ante robo de password.

Nivel de seguridad de SO

Se debe proteger en forma adecuada el SO porque puede servir para obtener acceso sin autorizacion a la BD

Protección contra logins inválidos

Protección de acceso a nivel de archivos

Sistemas de BD

El sistema de BD tiene la responsabilidad de garantizar que no se violen las restricciones de acceso.

Nivel de seguridad de Red

Cada sitio debe asegurar que se comunica con sitios autorizados
Los links deben protegerse contra robos y modificación de mensajes
Mecanismos de identificación y cifrado de mensajes.

Autorizaciones y vistas

Las vistas son una forma de proporcionar a usuario un modelo personalizado de la BD, de este modo se puede ocultar cierta parte de la BD que el usuario no debe ver.
Las bases de datos relacionales cuentan con dos niveles de seguridad:

- relación ➔ permitir o impedir que el usuario tenga acceso a la relación
- Vista ➔ impedirse o permitirse que el usuario tenga acceso a la información de una vista.

Puede permitirse tener acceso a una parte de una relación mediante una vista.
Un usuario puede tener una , varias o todas las siguientes formas de autorizacion sobre partes de la BD:
Sobre los datos

- Autorizacion de lectura ➔ leer pero no modificar la BD.
- Autorizacion de Insercion ➔ insertar nuevos datos pro no modificar los existentes.
- Autorizacion de actualizacion ➔ modificar la info pero no eliminar datos.
- Autorizacion de borrado ➔ puede eliminar datos.

Sobre el esquema de la BD

- Autorización para la creación de índices ➔ crear y eliminar indices.
- Autorización de recursos ➔ creación de relaciones nueva
- Autorización de alteracion ➔ permite agregar o eliminar atributos de una relación.
- Autorización de eliminacion ➔ permite eliminar relaciones.

El encargado de proveer los permisos es el administrador de la BD.

Cifrado

Muchas veces es necesario proteger los datos de manera más estricta, para lo cual se deben cifrar los datos y sólo se pueden leer si se conoce la forma de descifrado.
Propiedades de un buen sistemas de cifrado :

1-Para los usuarios autorizados debe ser fácil de cifrar/descifrar.

2-El cifrado no debe depender de mantener el secreto del algoritmo de cifrado, sino de un parámetro llamado clave de cifrado

3-dificil de determinar la clave de cifrado para un intruso

Otros sistemas se basan en tener dos claves, una pública y otra privada. Todos los usuarios tienen una clave pública (conocida) y otra privada. Cuando un usuario desea enviar datos, codifica con la clave publica del usuario a que se lo quiere mandar, dicho usuario decodifica la información con su clave privada. Debe existir un esquema de cifrado de clave publica, que pueda hacerse publico sin facilitar la deduccion del esquema de cifrado.

PROTECION DE DATOS ESTADISTICOS (consultas) ➔ mantenemos la seguridad mediante una "contaminacion de los datos". Se basa en una falsificacion al azar de los datos que se proporcionan en respuesta a una consulta, sin destruir el significado estadistico de la respuesta.

Seguridad e integridad de los datos

La información que se almacena en la base de datos debe protegerse contra el acceso no autorizado, la destrucción o alteración con fines indebidos y la introducción accidental de inconsistencias.

- © Una **vista** puede ocultar los datos que el usuario no necesita conocer. La seguridad puede lograrse si se cuenta con un mecanismo que restrinja al usuario a su vista o vistas personales.
- © Un usuario puede contar con varios tipos de **autorización** sobre partes de la base de datos. La autorización es un modo de proteger al sistema contra el acceso no autorizado, o mal intencionado.
- © Las **limitantes de integridad** son una forma de garantizar que los cambios que efectúan los usuarios autorizados, sobre la base de datos, no resulten en pérdidas de la consistencia de la información. Una limitante de integridad puede ser un predicado arbitrario referente a la base de datos.
- © En ocasiones se requiere un esquema alterno para la conservación de la integridad, **llamado esquema de disparadores**. Un disparador es una proposición que el sistema ejecuta en forma automática cada vez que se modifica la base de datos.
- © Para proteger datos se puede **cifrar** la información. Para leer información cifrada es necesario que el lector conozca la forma de descifrar los datos.

Otra técnica para mantener la seguridad es la **contaminación de los datos**, que se basa en la falsificación al azar de los datos que se proporcionan en respuesta a una consulta. Una técnica similar se basa en la modificación aleatoria de la consulta misma. En estas dos técnicas el objetivo es lograr un equilibrio entre la exactitud y la seguridad.

Unidad 7: Control de concurrencia

En un entorno de multiprogramación es posible ejecutar varias transacciones de manera concurrente. Es necesario que el sistema controle la interacción entre las transacciones concurrentes para evitar que destruyan la consistencia de la BD. El control se logra por medio de varios mecanismos a los que nos referimos como ***esquema de control de concurrencia***.
Una **transacción** es una unidad que conserva la consistencia. Por tanto, es requisito indispensable que cualquier cuadro que se produzca al procesar un conjunto de transacciones de manera concurrente sea computacionalmente equivalente a un cuadro que se produzca al ejecutar estas transacciones en serie en algún orden. Se dice que un sistema que garantiza esta propiedad garantiza la **serialibilidad.**

Planificaciones (secuencia de ejecución de transacciones)

Cuando se ejecutan varias transacciones de manera concurrente, la consistencia de la BD puede destruirse aún cuando cada una de las transacciones individuales sea correcta.

♦ Planificaciones en serie y paralelo

Tenemos varias cuentas y transacciones que acceden y actualizan esas cuentas
Sean T0 y T1 dos transacciones que transfieren fondos de una cuenta a otra (To de A a B 50 dolares y T1 de A a B el 10 por ciento del saldo:.

```
To:  Read (A)
     A:= A – 50
     Write (A)
     Read (B)
     B:= B + 50
     Write (B)
```

```
To:  Read (A)
     Temp:= A *
0.1
     A:= A - Temp
     Write (A)
     Read (B)
     B:= B + Temp
```

Supóngase que las dos transacciones se ejecutan de una en una en el orden T0 seguida de T1. La suma A + B se conserva después de la ejecución de ambas transacciones. Lo mismo ocurre si se ejecutan de una en una en el orden T1 seguida de T0.
Estas secuencias se llaman **planificaciones**. ***Representan el orden cronológico en que se ejecutan las instrucciones en el sistema***. Es evidente que una planificación para un conjunto de transacciones debe constar de todas las instrucciones de esas transacciones y debe conservar el orden en que aparecen las instrucciones en cada transacción individual.
Las planificaciones descriptas arriba se denominan **planificaciones en serie**. Cada una de ellas consta de una secuencia de instrucciones en varias transacciones en las que las instrucciones pertenecientes a una única transacción aparecen juntas en esa planificación. Así, para un conjunto de **n** transacciones existen **n!** planificaciones en serie diferentes.
Cuando se ejecutan varias transacciones concurrentemente, no es preciso que la planificación correspondiente esté en serie. Así, el número de planificaciones posibles para un conjunto de n transacciones es mucho mayor que n!. Existen varias secuencias de ejecución posibles, ya que pueden irse cambiando varias instrucciones de las transacciones.

T0	T1
Read(A)	
A:= A – 50	
Write(A)	
Read(B)	
B:= B + 50	
Write(B)	
	Read(A)
	Temp:= A * 0.1
	A:= A – Temp
	Write(A)
	Read(B)
	B:= B + Temp
	Write(B)

Después de llevarse a cabo esta ejecución, llegamos al mismo estado que al que se llega en el caso en que las transacciones se ejecuten en serie. La suma A + B se conserva.
No todas las ejecuciones concurrentes resultan en un estado correcto:

T0	T1
Read(A) A:= A – 50	
	Read(A) Temp:= A * 0.1 A:= A – Temp Write(A) Read(B)
Write(A) Read(B) B:= B + 50 Write(B)	
	B:= B + Temp Write(B)

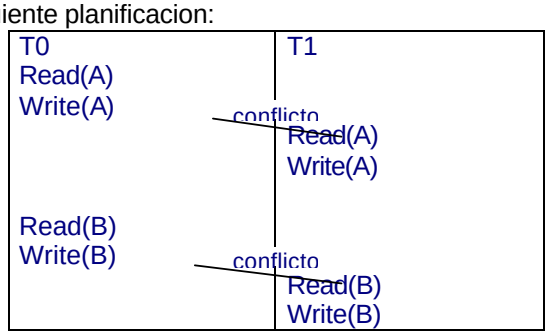
Después de la ejecución de esta planificación llegamos a un estado inconsistente, pues no se conserva la suma A + B.
Es necesario que una transacción sea un programa que conserve la consistencia. Es decir, cada transacción, al ejecutarse sola, transfiere el sistema de un estado consistente a un nuevo estado consistente. Sin embargo, durante la ejecución de una transacción puede que el sistema entre temporalmente en un estado inconsistente. La inconsistencia temporal es la que causa la inconsistencia en las planificaciones en paralelo.
La planificación debe ser, de alguna forma, equivalente a una planificación en serie.

♦ Conflicto en planificaciones serializables

Consideremos una planificación S en las que hay dos instrucciones consecutivas Ii e Ij de las transacciones Ti y Tj respectivamente. Si Ii e Ij se refieren a diferentes datos, entonces podemos intercambiar Ii e Ij sin que afecte a los resultados de las instrucciones de la planificación. Sin embargo, si Ii e Ij se refieren al mismo dato Q, entonces puede que importe el orden de los pasos. Como tratamos instrucciones READ y WRITE unicamente es necesario considerar cuatro casos:

- Ii = READ(Q), Ij = READ (Q). El orden de Ii e Ij no importa.
- Ii = READ (Q), Ij = WRITE(Q). Si Ii viene antes de Ij, entonces Ti no lee el valor de Q que escribe Tj en la instrucción Ij. Si Ij viene antes de Ii, entonces Ti lee el valor de Q que escribe Tj. Por lo tanto, el orden de Ii e Ij si importa.
- Ii = WRITE (Q), Ij = READ (Q). El orden de Ii e Ij sí importa por razones similares al caso anterior.
- Ii = WRITE (Q), Ij = WRITE (Q). El orden sí importa porque no se afecta alterando el orden de los WRITE, pero si se afecta el valor obtenido por la siguiente READ porque en la BD solo se conserva el resultado de la ultima WRITE.

Solo en el caso en que Ii e Ij sean READ no importa el orden relativo de su ejecución.
Decimos que Ii e Ij están en ***conflicto*** si son operaciones de transacciones distintas sobre el mismo dato, y por lo menos una de estas instrucciones es una operación WRITE.
Consideramos la siguiente planificacion:



WRITE(A) de To esta en conflicto con READ(A) de T1, pero WRITE(B) de T1 no esta en conflicto con READ(B) de To porque acceden a datos diferentes.
Sean Ii e Ij instrucciones consecutivas de una planificación S. Si Ii e Ij son instrucciones de diferentes transacciones e Ii e Ij no están en conflicto, entonces podemos intercambiar el orden de Ii e Ij para producir una nueva planificaicón S'. (S y S' deben ser equivalentes y generar el mismo estado final del sistema).

T0	T1
Read(A)	

Write(A) Read(B) Write(B)	 Read(A) Write(A) Read(B) Write(B)

Si una planificación S puede transformarse en una S' mediante una serie de intercambios de instrucciones no conflictivas, decimos que S y S' son **equivalentes en conflictos**.
Decimos que una planificación S es **serializable en cuanto a conflictos** si es equivalente en conflictos a una planificación en serie (que una planificación sea equivalente a una planificación en serie implica que sin tener en cuenta el estado inicial del sistema la planificación producirá el mismo estado final que alguna planificación en serie).
Es posible tener dos planificaciones que produzcan la misma salida y que no sean equivalentes en cuanto a conflictos
Sea la planificación:

T0	T4
Read(A) A:=A - 50 Write(A)	
	Read(B) B:=B - 10 WRITE(B)
READ(B) B:=B+50 WRITE(B)	
	READ(A) A:=A+10 WRITE(A)

No es equivalente en conflictos a <To, T4> porque WRITE(B) de T4 esta en conflicto con la instrucción READ(B) de To, por lo que no podemos mover todas las instrucciones de To antes de las de T4 intercambiando instrucciones no conflictivas consecutivas, aunque los valores de A y B son los mismos luego de la ejecución de ambas planificaciones.

♦ Seriabilidad de vistas

Sea S y S' el mismo conjunto de transacciones, son equivalentes en cuanto a vistas si se cumple las siguientes condiciones:

1. Para cada dato Q si la transacción T1 lee el valor inicial de Q en la planificación S ➔T1 también debe leer el valor inicial de Q en S'.
2. Para cada dato Q si la transacción T1 ejecuta READ(A) en la planificación S y ese valor fue producido por la transacción T2 ➔ T1 también debe leer en S' el valor de Q producido por T2.
3. Para cada dato Q la transacción que ejecuta la operación WRITE(Q) final en la planificación S debe ejecutar la operación final WRITE(Q) en la planificación S'.

Las condiciones 1 y 2 aseguran que cada transacción lee los mismos valores.
Las tres condiciones juntas aseguran que ambas resultan en el mismo estado final del sistema.

Toda planificación seriabilizable en confllictos es seriabilizable en vistas, pero no toda seriabilizable en vista es seriabilizable en conflictos.
Llamamos **escrituras ciegas** a aquellas que se hacen sin un READ previo, aparecen en planificaciones seriabilizables en vistas y no en conflictos.

Pruebas de serializabilidad

Supongase que nos dan una planificación S determinada y queremos determinar si es serializable. Metodos:

Pruebas de serializabilidad de conflictos (Grafo de precedencia)

Sea **S** una planificación. Construimos un grafo dirigido llamado **grafo de precedencia de S**. Este grafo consta de un par G=(V,E), donde V es un conjunto de vértices y E es un conjunto de aristas. V consta de todas las transacciones que participan en la planificación. E consta de todas las aristas Ti → Tj para las cuales se cumple una de las tres condiciones siguientes:

- Ti ejecuta WRITE(Q) antes de que Tj ejecute READ(Q)
- Ti ejecuta READ(Q) antes de que Tj ejecute WRITE(Q)
- Ti ejecuta WRITE(Q) antes de que Tj ejecute WRITE(Q)

Si existe una arista Ti → Tj en el grafo de precedencia, esto implica que en cualquier planificación en serie S' equivalente a S, Ti debe aparecer antes de Tj.

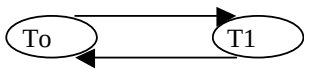
- Si el grafo de precedencia S tiene un ciclo, entonces la planificación S no es serializable en conflictos.
- Si el grafo no contiene ciclos, entonces la planificación S es serializable en conflictos.

El orden de serializabilidad puede obtenerse mediante una ordenación topológica, que determina un orden lineal consistente con el orden parcial del grafo de precedencia. En general, existen varios órdenes lineales posible que pueden obtenerse mediante una ordenación topológica.

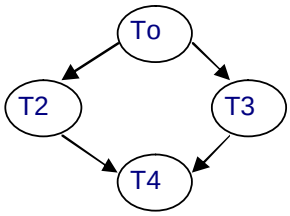
Sea la planificacion:

T0	T1
Read(A) A:= A – 50 Write(A) Read(B) B:= B + 50 Write(B)	
	Read(A) Temp:= A * 0.1 A:= A – Temp Write(A) Read(B) B:= B + Temp Write(B)

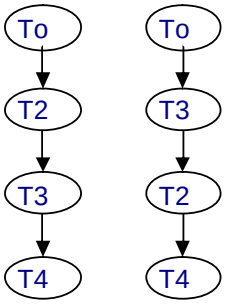
Su grafo de precedencia:



Sea el siguiente grafo:



Dos ordenaciones topologicas posibles:



Prueba de seriabilidad de vistas

No existe un algoritmo eficiente para probar la seriabilidad de vistas.
Necesitamos un esquema para decidir cuando es necesario insertar una arista en el grafo de precedencia.
Un grafo que contiene aristas etiquetadas se llama **grafo de precedencia etiquetado.** Los nodos son las transacciones que participan de la planificacion. Reglas para insertar las aristas etiquetadas.:

Sea S una planificacion que tine las transacciones {To,...,Tn} y Tb, Tf dos variables ficticias tales que Tb ejecuta WRITE(Q) por cada Q accedido en S y Tf ejecuta READ(Q) por cada Q accedido en S. Construimos una nueva planificacion S' a partir de S insertando Tb al principio de S y Tf al final de S. Construimos el grafo de precedencia etiquetado de la planificacion S':

Se añade una arista To→Tj si Tj lee el dato Q escrito por To

Se eliminan todas las aristas que inciden en transaciones inutiles (Ti es inutil cuando no existe un camino en el grafo dede Ti hasta Tf)
Por cada dato Q tal que :
Tjmlee Q escrito por Ti
Tk ejecuta WRITE(Q) y Tk≠Tb
Se hace lo siguiente:
Si Ti=Tb y Tj≠Tf entonces se inserta la arista To→Tk
Si Ti≠Tb y Tj=Tf se inserta la arista To→Ti en el grafo de precedencia etiquetado.
Si Ti≠Tb y Tj≠Tf se inserta el par de aristas Tpk→Ti t Tpj→Tk , p entero unico mayor que 0 que no se uso para etiquetar aristas.

(Ejemplo pag 385)

Protocolos basados en bloqueos

Consiste en exigir que mientras una transacción accede a un dato, ninguna otra puede modificarlo. Esto se hace permitiendo que una transacción acceda a un dato sólo si tiene actualmente un bloqueo en ese dato.

Metodo 1: Bloqueo

- **Compartido**: Si una transacción Ti ha obtenido un bloqueo de modo compartido (representado por S) en el dato Q, entonces Ti puede leer ese dato pero no puede escribir Q. Se ejecuta haciendo LOCK-X(Q) y se desbloquea UNLOCK(Q).

- **Exclusivo**: Si una transacción Ti ha obtenido un bloqueo de modo exclusivo (representado por X) en el dato Q, entonces Ti puede leer y escribir Q.

Función de compatibilidad: Sean A y B modos de bloqueo. Supóngase que la transacción Ti solicita un bloqueo de modo A en el dato Q en el cual la transacción Tj (Ti ≠ Tj) tiene actualmente un bloqueo de modo B. Si puede concedérsele a Ti un bloqueo en Q inmediatamente, entonces decimos que el modo A es compatible con el modo B.
Las funciones compatibles pueden representarse por medio de matrices. Un elemento de la matriz M , M(A,B) es verdadero sii el modo A es compatible con el modo B.
Un modo comparido es compatible con el modo compartido pero no con el exclusivo.
Para acceder a un dato, la transacción Ti primero debe bloquearlo. Si el dato ya está bloqueado por otra transacción en un modo incompatible, entonces Ti debe esperar hasta que se liberen todos los bloqueos incompatibles que tengan otras transacciones. La transacción Ti puede desbloquear un dato al que había bloqueado con anterioridad.
Una transacción debe tener un bloqueo en un dato mientras esté accediendo a él. No siempre es deseable que una transacción desbloquee un dato inmediatamente después de su último acceso a ese dato, ya que existe la posibilidad de que no puede garantizarse la serializabilidad.
Debemos bloquear y desbloquear los datos correctamente para que cuando dos transacciones se ejecutan concurrentemente produzcan datos correctos.

Ejemplo: T1 transfiere 50 de B a A, y T2 presenta la cantidad total de dinero en las cuenta A y B, la suma A + B:

T1	T2
LOCK-X(B) READ(B) B:=B-50 WRITE(B) LOCK-X(A) READ(A) A:=A+50 WRITE(A) UNLOCK(B) UNLOCK(A)	LOCK-S(A) READ(A) LOCK-S(B) READ(B) DISPLAY(A+B) UNLOCK(A) UNLOCK(B)

Consideramos la planificacion:

T1	T4
Lock-x(B) Read(B) B:= B – 50 Write (B)	
	Lock-s(A) Read(A) Lock-s(B)
Lock-x(A)	

Como ninguna de las transacciones puede seguir, se llega a **Deadlock**. El sistema debe retroceder una de las dos transacciones y los datos que tenía bloqueados esa transacción se liberan y la otra transaccion puede volver a disponer de los datos y continuar su ejecucion.
Si intentamos lograr un máximo de concurrencia desbloqueando datos tan pronto como sea posible, podemos obtener estados inconsistentes. Si no desbloqueamos un dato antes de solicitar un bloqueo en otro dato, pueden ocurrir situaciones de Deadlock.
Por eso exigimos que todas las transacciones del sistema sigan un conjunto de reglas, llamado **protocolo de bloqueo**, ***que indica cuando una transacción puede bloquear y desbloquear los datos***. Restringen el numero de planificaciones seriabilizables posibles.
Decimos que una planificación S es **legal** bajo un protocolo de bloqueo dado si S es una planificación posible para un conjunto de transacciones que sigue las reglas del protocolo de bloqueo. Decimos que un protocolo de bloqueo garantiza la serializabilidad en conflictos sólo si para todas las planificaciones legales la relación asociada es acíclica.

Metodo 2: El protocolo de bloqueo de dos fases

Este protocolo garantiza la serializabilidad y requiere que todas las transacciones hagan sus solicitudes de bloqueo y desbloqueo en dos fases:

- ✓ Fase de crecimiento: una transacción puede obtener bloqueos pero no puede liberar ningún bloqueo.
- ✓ Fase de encogimiento: una transacción puede liberar bloqueo pero no puede obtener ningún bloqueo nuevo.

Inicialmente, una transacción está en la fase de crecimiento. La transacción adquiere los bloqueos que necesita. Una vez que libere un bloqueo, entra en la fase de decrecimiento y no podrá solicitar más bloqueos.
Este protocolo garantiza la seriabilidad en conflictos pero no garantiza que no se produzca Deadlock.
Si Ti no es una transacción de dos fases, siempre es posible encontrar otra transacción Tj que sea de dos fases, tal que exista una posible planificación no serializable en conflictos para Ti y Tj.
Es posible un refinamiento del protocolo, donde se provee un mecanismo para "elevar" un bloqueo compartido a exclusivo (**upgrade**) y para "bajar" un bloqueo exclusivo a compartido (**downgrade**), logrando, en ciertos casos, mejorar la concurrencia. Upgrade puede hacerse sólo en la fase de crecimiento, mientras que downgrade sólo en la de decrecimiento.
Cuando una transacción Ti ejecuta una operación READ(Q), el sistema ejecuta una instrucción LOCK-S(Q) seguida de la instrucción READ(Q). Cuando Ti ejecuta una operación WRITE(Q), el sistema comprueba si Ti ya tiene un bloqueo compartido en Q. En ese caso, ejecuta una instrucción UPGRADE(Q) seguida de WRITE(Q). En caso contrario, ejecuta LOCK_X(Q) seguida de WRITE(Q). Si esto es posible para un conjunto de transacciones existen planificaciones seriabilizables en conflictos que no pueden obtenerse mediante un protocolo de bloqueo de dos fases.

Merodo 3:Protocolos basados en grafos.

Consiste en tener conocimiento previo del orden en que se va acceder a la BD pero que garanticen seriabilidad. Para adquirir este cnocimiento, imponemos una ordenación parcial en el conjunto D={d1, d1, ..., dn} de todos los datos. Si di → dj, entonces se debe acceder a di antes que a dj. Esta ordenacion puede roponerse unicamente para conrolar concurrencia. El conjunto **D** puede verse como un ***grafo acíclico dirigido***, llamado **grafo de la BD**. Se usarán sólo los grafos que sean árboles con raíz.

Se utilizará un PROTOCOLO DE ARBOL restringido sólo a bloqueos exclusivos:
La unica instrucción de bloqueo permitida es LOCK-X. Cada transacción Ti puede bloquear un dato a lo sumo una vez y debe observar las siguientes **reglas**:

1. El primer bloqueo de Ti puede ser en cualquier dato.
2. A partir de ese momento, Ti puede bloquear un dato Q sólo si Ti bloquea actualmente al padre de Q.
3. Los datos pueden desbloquearse en cualquier momento.
4. Un dato que Ti haya bloqueado y desbloqueado, no puede volver a bloquearlo posteriormente.

Todas las planificaciones que sean legales bajo el protocolo de árbol son serializables en conflictos.

- ✓ Ventajas con respecto al de dos fases: el desbloqueo puede ocurrir antes que en protocolo de dos fases, por lo tanto menos tiempo de espera y mas concurrencia. Hay libertad de bloqueos por lo que no hay retrocesos.
- ✓ Desventajas con respecto al de dos fases: en algunos casos es necesario que una transacción bloquee datos a los que no accede, aumentando el tiempo extra para el bloqueo, tiempo de espera adicional y disminucion del potencial de la concurrencia.

Protocolos basados en hora de entrada

Es seleccionar por adelantado una ordenación entre transacciones. Para esto, usando el reloj del sistema o un contador lógico, el sistema asigna una hora de entrada a Ti, representada por **TS(Ti),** tal que ***TS(Ti) < TS(Tj) si Tj entró después de Ti***.
Las horas de entrada de las transacciones determinan el orden de seriabilidad, entonces si TS(Ti) < TS(Tj), entonces el sistema debe asegurarse de que la planificación producida es equivalente a una planificación en serie en la que Ti aparece antes de Tj.
Asociamos dos valores de hora de entrada a cada dato Q que se actualizan cada vez que se ejecuta un nuevo READ(Q) o WRITE(Q):

- ⦚ ***W hora de entrada(Q)*** = que representa la mayor hora de entrada de cualquier transacción que ejecutó con éxito write(Q). => **HW(Q).**
- ⦚ ***R hora de entrada(Q)*** = que representa la mayor hora de entrada de cualquier transacción que ejecutó con éxito read(Q). => **HR(Q).**

Protocolo de ordenación por orden de entrada

El protocolo de ordenación por hora de entrada garantiza que todas las operaciones READ y WRITE que pudieran entrar en conflicto se ejecuten en el orden de hora de entrada.
Modo de operación del protocolo:

- Supóngase que Ti solicita READ(Q)
  - Si TS(Ti) < HW(Q) → Ti necesita leer un valor de Q que ya fue sobreescrito. Se rechaza READ y Ti retrocede.
  - Si TS(Ti) ≥ HW(Q) → se ejecuta la operación READ y HR(Q) = max(HR(Q), TS(Ti)).
- Supóngase que Ti solicita WRITE(Q)
  - Si TS(Ti) < HR(Q) → el valor de Q que Ti está produciendo se necesitó con anterioridad y se supuso que nunca se produciría. Se rechaza WRITE y Ti retrocede.
  - Si TS(Ti) < HW(Q) → Ti está intentando escribir un valor obsoleto de Q. La operación WRITE se rechaza y Ti retrocede. (Roolback).
  - En los demás caso, la operación WRITE se ejecuta y HW(Q) = max(HW(Q), TS(Ti))

A cada transacción Ti que retrocede por solicitar un READ o WRITE se le asigna una nueva hora de entrada y se reinicia.
Este protocolo garantiza la seriabilidad en conflictos y las operaciones que pudieran entrar en conflicto se procesan en el orden de entrada.
Asegura la libertad de bloqueo porque las transacciones nunca esperan.

Regla de escritura de Thomas

Es una modificacion del protocolo de ordenacion por hora de entrada. Funciona eliminando las operaciones WRITE obsoletas de las transacciones que las solicitaron. En los casos en que TS(Ti)>= R-hora- de-entrada(Q), ignoramos la operación WRITE obsoletas.

Granularidad múltiple

Se dan circunstancias en las que sería conveniente agrupar varios datos y tratarlos como una unidad de sincronización individual.
(Algunas transacciones acceden a registros y otras a la tabla entera.)
Lo que se necesita es un mecanismo que permita al sistema definir múltiples niveles de **granularidad** permitiendo que los datos tengan distintos tamaños y se defina una jerarquía de granularidades de los datosgranularidades peque.as se anidan en las mas grandes.
El término granularidad se utiliza para referir al nivel de objeto de datos que se bloquea. Se define una ***jerarquía de granularidad*** que puede ser representada por medio de un ***árbol***. Se puede bloquear cada nodo del árbol individualmente (compartido o exclusivo), pero hay que tener en cuenta que se verificarán bloqueos implícitos en esa acción sobre los descendientes de ese nodo (una transaccion bloquea un nodo en modo compartido o exclusivo, entonces habra bloqueado implicitamente a todos sus descendientes en el mismo modo).
Cuando una transacción solicita un bloqueo contra uno de los niveles, debe recorrer el árbol para ver si otra transacción ya tiene un bloqueo sobre un nivel superior que sea compatible con el solicitado. Para minimizar los recorridos sobre el árbol, conviene introducir un ***bloqueo de intención***, el cual se aplica a los niveles superiores del que se está efectivamente bloqueando. Habrá que distinguir entre los bloqueos de intención (se hace un bloquep explicito en un nivel mas bajo con bloqueos de modo exclusivo o compartido) y compartidos (implica que se esta haciendo un bloqueo explicito en un nivel mas bajo del arbol pero solo con bloqueos de modo compartido). Entonces, cuando una transacción quiere bloquear un dato, en su recorrido hacia la raíz para analizar qué ha sucedido antes, va marcando cada nodo con el correspondiente bloqueo de intención.

El protocolo de bloqueo de granularidad multiple que se presenta a continuacion garantiza la seriabilidad. Cada transaccion Ti puede bloquear un nodo Q utilizando las siguientes reglas:
1. Debe respetarse la funcion de compartibilidad de bloqueo de

	IS	IX	S	SIX	X
IS	verdadero	verdadero	verdadero	verdadero	falso
IX	verdadero	verdadero	falso	falso	falso
S	verdadero	falso	verdadero	falso	falso
SIX	verdadero	falso	falso	falso	falso
X	falso	falso	falso	falso	Falso

2. Bloquear la raiz del arbol en cualquier modo
3. Ti puede bloquear al nodo Q en el modo S o IS solo si Ti bloqueo al padre de Q en modo IX o IS
4. Ti puede bloquear al nodo Q en el modo X, SIX o IX solo si Ti bloqueo al padre de Q en modo Ix o SIX
5. Ti puede bloquear un nodo solo si no desbloqueo anteriormente ningun nodo (Ti es de dos fases.
6. Ti puede desbloquear un nodo Q solo si ninguno de los hijos de Q esta bloqueado por Ti.

Los bloqueos se hacen de arriba abajo y se liberan de abajo a arriba.
Este protocolo intensifica la concurrencia y reduce el gasto de bloqueo.

Esquemas multiversion

Cada versio de WRITE crea una nueva version de Q. Cuando se solicita un READ(Q) se selecciona una de las versiones de Q para leerse, garantizando seriabilidad, usando la hora de entrada.
A cada dato Q se asocia una secuencia de versiones <Q1,...,Qm>. Cada version Qk contiene tres campos de informacion:

- Contenido ➔el valor de la version Qk
- W-hora-de-entrada(Qk) ➔ se representa la hora de entrada de la transaccion que creo la version Qk
- R-hora-de-entrada(Qk) ➔ representa la hora de entrada mas grande de cualuier transaccion que haya leido con éxito la version Qk.

Operación del esquema: la transaccion Ti solocita un READ(Q) o WRITE(Q). Sea Qk una version de Q con hora de entrada de escritura es la hora de entrada mas grande, menor que TS(Ti)

1. Si Ti solicita READ(Q) el valor devuelto es el del contenido de la version Q
2. Si la transaccion Ti solicita WRITE(Q) y TS(Ti) < R-hora-de-entrada(Qk) entonces la transaccion Ti retrocede, en caso contrario se crea una nueva version de Qk.

Este esquema tiene la propiedad de que una solicitu de lectura nunca falla y nunca tiene que esperar.
Desventaja: la lectura de un dato requiere actualizacion de R-hora-de-entrada (dos accesos a disco) y los conlictos entre transacciones se resuelven por medio de retrocesos en vez de esperas (costoso).

OTRAS OPERACIONES

Delete(Q): Hace conflicto con cualquier otra. Sobre bloqueo de dos fases, por ej., se requiere un lock completo del registro. En timestamps, si TS(Ti)>=max(W(Q),R(Q)) se acepta.
Insert(Q): En dos fases, se le da un lock exclusivo al que lo crea. En timestamp, se setea W(Q)=R(Q)=TS(Ti).

PROCESAMIENTO DE TRANSACCIONES

Modelo de almacenamiento.

Existe un unico registro intermedio (buffer) de la BD compartido y uno de bitácora compartido.
Cada transaccion tiene su propia área de trabajo en la que almacena la copia de los datos a los que accede.
La memoria principal tiene tres registros intermedios:

- REGISTRO INTERMEDIO DEL SISTEMA: contiene paginas de codigo objeto del sistema y las areas de trabajo locales de las transacciones activas. Los datos que contiene estan bajo el control del gestor de la memoria virtual del SO.
- REGISTRO INTERMEDIO DE BITACORA: contiene paginas de registros de bitacora hasta que se graban en el almacenamiento estable.
- REGISTRO INTTERMEDIO DE LA BD: contiene paginas de la BD, lo gestiona el SO.

El almacenamiento secundario se divide en categorias:

- CODIGO OBJETO DEL SISTEMA: codigo para el sistema
- ÁREA DE RECOPIA DE MEMORIA VIRTUAL: el área del disco que se usa para almacenar las paginas del área de trabajo de las transacciones locales que no se guarda en la memoria ppal.
- ALMACENAMIENTO ESTABLE EN LINEA: aproximacion de almacenamiento estable, contiene los registros requeridos para recuperacion de fallos de almacenamiento volatil. (recupera inmediatamente de caidas del sistema y fallos de transacciones)
- ALMACENAMIENTO ESTABLE EN ARCHIVOS: aproximacion fuera de linea del almacenamiento estable, contiene solo los datos necesarios para recuperacion de fallos de almacenamiento no volatil.

Todas las transacciones comparten el registro intermedio de bitacora y el de BD, pero cada una tiene su área de trabajo.

Sistemas concurrentes. Recuperación de fallos en transacciones.

Si una transaccion T falla debemos deshacer los cambios que esta hizo en la BD y cualquier tansaccion T' que sea dependiente de T tambien se aborta.

Retroceso en cascada.

Cuando el fallo de una transaccion conduce a una serie se retrocesos de transacciones (leen datos que escribio T).
No es deseable porque lleva a deshacer una importante cantidad de trabajo.

Planificacion recuperable

Una planificacion es **no recuperables** si la transaccion Ti que depende de Tj ya termino, cuando falle Tj y se la retroceda no va a poder retrocederse Ti (porque ya termino). Dejando una inconsistencia. Tratamos de evitar estas planificaciones.
Los sitemas de procesamiento de transacciones deben garantizar que es posible recuperarse del fallo de cualquier transaccion activa.
Timestamp protocolo: un bit 'commit' para cada transacción. Se hace esperar a Ti si quiere leer un dato que modifico Tj y bj=false. Este protocolo introduce espera pero no deadlock.
Dos fases: todos los bloqueos exclusivos se conservan hasta el commit.

Exploracion de bitacora

En un sistema de procesamiento de transacciones concurrentes los puntos de verificacion del registro de la bitacora son de la forma <CHECKPOINT L> donde L es una lista de transacciones activas en el momento del punto de verificacion.
Cuando el sistema se recupera de una caida contruye dos listas en la recuperacion: LIASTA DE DEAHACER (transacciones a deshacer) y LISTA DE VOLVER A HACER (transacciones a repetir), que inicialmente estan vacias y recorremos la bitacora hacia atrás en busca de checkpoint:

- Por cada registro <Ti COMMITS> añadimos Ti a la lista de volver a hacer.
- Por cada registro <Ti STARTS> si Ti no esta an la lista volver a hacer añadimos Ti a la lista de deshacer.

Terminada la recuperacion reanudamos el procesamiento de las transacciones.

Deadlock

Definición ➔ existe un conjunto de transacciones tal que cada una del conjunto esta esperando a otra del conjunto.
Metodos para tratar el problema:

PREVENIR ➔ Para garantizar que el sistema nunca entrara en estado de bloqueo.
Hay 3 formas
1-Cada transacción bloquea todos los datos que necesita ántes de comenzar a ejecutarse. Problema, baja utilización de datos y posibilidad de inanición.

2-Imponer una ordenación parcial de los datos y exigir que una transacción pueda bloquear los datos en el órden especificado

3-Expropiación y retroceso, en esta estrategia se asigna una hora de ingreso a cada transacción la cual NO se cambia cuando una transacción es reseteada. Esquemas para prevencion empleando hora de entrada:

a- **Esperar – Morir:** Ti solicita un dato que tiene Tj. Si Ts(Ti)<Ts(Tj) entonces Ti espera de lo contrario Ti retrocede.
b- **Herir – Esperar:** Ti solicita un dato que tiene Tj, si Ts(Ti)>Ts(Tj) entonces espera Ti, si no expropia los recursos de Tj y se reinicia Tj. Este tiene menos retrocesos que el anterior.

Estos esquemas no tienen inanicion.

DETECCIÓN Y RECUPERACIÓN➔ permitimos que se entre en estado de bloqueo y luego tratamos de recuperarlo
Detección

Cada cierto tiempo se ejecuta un algoritmo que determina si el sistema está en bloqueo o no.

Se construye el grafo de espera (G(V,A) V transacciones del sistema y A existe una arista Ti➔Tj si Ti espera que Tj libere un dato que ella necesita(, para esto se requiere información adicional (asignaciones actuales, solicitudes pendientes), dicho grafo hay que mantenerlo actualizado.

Si hay un ciclo en el grafo, entonces hay bloqueo y las transacciones del ciclo son las bloqueadas. Para detectar bloqueos se invoca un algoritmo que busca ciclos en el grafo.

La frecuencia con la que se invoca el algoritmo de detección depende de la probabilidad de que ocurra bloqueo y a cuantas transacciones afectará.

Recuperacion

La solucion mas comun es retroceder una o varias transacciones teniendo en cuenta 3 factores:
Eleccion de la victima➔ la que representa menor costo(con menos tiempo de ejecucion, datos usados, datos necesarios para terminar, ...)
Retroceso (hasta donde se necesita retroceder la victima)
Inanicion (evitar elegir siempre la misma victima).

Unidad 8 ➔bases de datos distribuidas

Arquitecturas de Sistemas de Bases de Datos

Sistemas Centralizados:

- Corren en una sola computadora sin interactuar con otros sistemas (monousuarios)
- Sistemas de cómputo de propósito general: algunas CPU y un número de dispositivos que están conectados a través de un bus común que proveen acceso a memoria compartida
- Sistemas multiusuario: sistemas mainframes.

Sistemas paralelos

- Múltiples procesadores y múltiples discos conectados a través de una red de alta velocidad.

Sistemas distribuidos vs. Paralelos

- Procesadores poco acoplados
- No comparten componentes físicos
- Independencia de DB de cada nodo o localidad

BASES DE DATOS DISTRIBUIDAS

Los datos se almacenan en varios computadores que se comunican entre si
Un sistema distribuido de bases de datos consiste en un conjunto de localidades (procesadores), cada una con una base de datos local. Cada localidad maneja transacciones locales y puede participar de transacciones globales.

Ventaja de la utilizacion
Capacidad de compartir y acceder a la información de una forma fiable y eficaz.

- **Autonomía:** cada sitio tiene el control de su información
- **Disponibilidad:** los datos pueden ser replicados en varios lugares y el sistema puede funcionar aún un sitio falla.
- **Utilización compartida de los datos y distribución del control:** una localidad puede acceder a los datos de otra. El administrador global de la base de datos se encarga de todo el sistema, delegando responsabilidades a los locales.
- **Fiabilidad y disponibilidad:** si una localidad se cae, es posible que las demás localidades sigan trabajando (mayor disponibilidad). El sistema detecta cuando falla una localidad y toma medidas necesariaspara recuperarse dejando de usar la localidad que fallo. Cuando esta se recupera debe haber mecanismos para reintegrarla al sistema.

Agilización del procesamiento de consultas: si una consulta comprende datos de varias localidades se puede dividir las consultas en subconsultas que se ejecuten en paralelo en distintas localidades

Desventajas
mayor complejidad para coordinar las localidades, esto es:

- mayor costo de soft
- mayor posibilidad de errores: como las localidades operan en paralelo es mas dificil garantizar que los algoritmos sean correctos.
- mayor tiempo extra de procesamiento (intercambio de mensajes para coordinar las localidades)

Diseño de bases de datos distribuidas:

Hay varios factores a tomar en cuenta:

- **Repetición:** se mantienen varias copias de la relación en distintas localidades. Mejora rendimiento en lectura, aumenta disponibilidad, pero agrega overhead en escritura.
- **Fragmentación:** la relación se divide en varios fragmentos en localidades diferentes.
- **Fragmentación horizontal**: partición horizontal (eq. a la selección).
- **Fragmentacion vertical:** partición vertical. (eq. a proyección + un id-tupla para |X|).
- **Repeticion y fragmentacion:** combinacion de ambas. La relación se divide en fragmentos y el sistema mantiene varias copias identicas de cada uno de ellos

TRASNPARENCIA Y AUTONOMIA

Transparencia: grado hasta el cual los usuarios del sistema pueden ignorar los detalles del diseño distribuido (distribucion de la informacion).
Autonomía: grado de independencia de un administrador local del resto del sistema distribuido.

Puntos a considerar:

Asignación de nombres y autonomia local: Las localidades deben asegurarse de no usar el mismo nombre para datos diferentes.

Una solución es usar un asignador central de nombres. (cuello de botella). Además si el asignador de cae, es posible que nadie pueda seguir trabajando. Se reduce la autonomía local, porque la asignación es centralizada.

Otra solución: cada localidad pone como prefijo su identificador de localidad a cualquier nombre que genere. Más autonomía local, pero no transparencia de la red ya que se agregan identificadores de localidad a los nombres. Cada copia y fragmento deben tener el mismo nombre que el elemento original, pero además hay que poder determinar que es un fragmenta o copia. Por lo tanto se le agregan sufijos indicando qué copia es y/o qué fragmento es.

Transparencia de la repetición y la fragmentación: el sitstema oculta al usuario gran parte del esquema de la BD distribuida.
Cuando se solicita un dato, no es necesario especificar qué copia se quiere. Hay una tabla para determinar las copias de un dato. El sistema determina que copia acceder cuando se le solicite una lectura y modifica todas las copias cuando ocurre una escritura.

La fragmentación debe ser transparente a los usuarios. Se tiene que poder trabajar con toda la relación y el sistema debe encargarse de reconstruirla a partir de los fragmentos.

Transparencia de la localización: Cada localidad tiene una lista alias para cada usuario, que ese usan para referirse los datos y el sistema traduce en nombres. Asi el usuario no necesita conocer la ubicación fisica del dato y el administrados de BD podra cambiarlo de localidad.

PROCESAMIENTO DISTRIBUIDO DE CONSULTAS

En el procesamiento de consultas se debe tomar en cuenta:

- Accesos al disco
- El costo de transmisión de los datos en la red.
- El beneficio de ejecutar partes de la consulta en localidades distintas en paralelo.

Repeticion y fragmentacion ➔Al hacer una consulta, hay que ver si la relación esta repetida y/o fragmentada. El sistema elige qué copia usar y/o une los fragmentos que sean necesarios.

Recuperacion en sistemas distribuidos

Estructura del sistema

Cada localidad cuenta con los siguientes dos subsistemas:
Gestor de transacciones: gestiona la ejecución de las transacciones que acceden a datos almacenados en esa localidad. Estas pueden ser transacciones locales o globales. Se encarga de mantener la bitácora para la recuperación y coordina la ejecucion en paralelo de las transacciones que se ejecutan en esa localidad
Coordinador de transacciones: coordina la ejecución de varias transacciones iniciadas en esa localidad. Se encarga de: iniciar la ejecución; dividirla en varias subtransacciones que debe distribuir en las localidades apropiadas para su ejecucion, y coordinar la terminación.

Manejo de fallos

Tipos de fallos : de memoria, de disco, de una localidad, interrupcion de una linea de comunicación, perdida de mensajes, fragmentacion de la red.

Si una localidad falla, el sistema debe iniciar un procedimiento de reconfiguración del sistema que permita continuar con las operaciones normales.

- Si una localidad falla, hay que actualizar el catálogo para que las consultas no accedan a dicha localidad.
- Si existían transacciones activas en la localidad que fallo, deben abortarse.
- Si la localidad que quedo fuera de servicio es el distribuidor central de un subsistema, es necesario elegir un nuevo distribuidor.

Hay que tener cuidado al reintegrar una localidad al sistema. Cuando se la reintegra, hay que ejecutar un procedimiento de actualización de sus tablas de sistema para que reflejen los cambios ocurridos mientras estaba inactiva.

PROTOCOLOS DE COMPROMISO

Para garantizar la atomicidad, todas las localidades en las que se haya ejecutado una transacción T deben coincidir en el resultado final ya sea “ejecutada” o “abortada”. El coordinador ejecuta un protocolo de compromiso:

Sea T una transacción que se inicio en la localidad Li y sea Ci el coordinador de esa localidad. Ci inicia el protocolo cuando T termina de ejecutarse (parcialmente cometida).

Compromiso de dos fases

Fase 1: Ci añade el registro <PREPARAR T> a la bitácora y la graba en memoria estable. Luego envía un mensaje "preparar T" a las localidades en donde se ejecutó T.
Cuando una localidad recibe este mensaje, su gestor de transacciones determina si esta dispuesto a ejecutar la parte de T que le tocó:

- Si no esta dispuesto, agrega un registro <NO T> a su bitácora y luego envía un mensaje "ABORTAR T" a Ci.
- Si esta dispuesto, agrega <LISTA T> a la bitácora y graba todos los registros de bitácora correspondientes a T en memoria estable. Una vez hecho eso envié un mensaje "lista T" a Ci.

Fase 2: se inicia cuando todas las localidades respondieron al mensaje "PREPARAR T".
Si Ci recibió todos "LISTA T" entonces T puede ejecutarse. En caso contrario, T debe abortarse.
Se agrega a la bitácora <EJECUTAR T> o <ABORTAR T> según sea el caso y luego se graba en memoria estable. A continuación Ci envía un mensaje "EJECUTAR T" o "ABORTAR T" a todas las localidades involucradas y cada una de las localidades lo registra en su bitácora.

En algunas implementaciones de este protocolo, una localidad envía un mensaje de "RACONOCIMIENTO" a T al coordinador al final de la segunda fase. Cuando el coordinador recibe este mensaje de todas las localidades, agrega el registro <T COMPLETA> en la bitácora.

MANEJO DE FALLOS (forma en la que el compromiso de dos fases responde a los fallos)

Fallo de una localidad participante: en el momento de recuperarse, la localidad debe fijarse en su bitácora par ver que paso con las transacciones que se estaban ejecutando cuando fallo. Casos:

- Si la bitácora tiene <EJECUTAR T>, la localidad ejecuta **redo(T)**.
- Si tiene <ABORTAR T>, se ejecuta **undo(T)**.
- Si tiene <LISTA T>, consulta a Ci para determinar el destino de T. Luego hará **redo(T)** o **undo(T)** según sea el caso. Si Ci esta inactiva, la localidad deberá consultar con el resto de las localidades para ver que paso con T.
- Si la bitácora no contiene registros de control, significa que la localidad tuvo un fallo antes de responder el mensaje "Preparar T" del coordinador. Por lo tanto no le envió respuesta al coordinador y este tuvo que abortar T. Entonces ahora la localidad debe ejecutar **undo(T)**.

Fallo del coordinador: si falla el coordinador, las localidades deben definir el destino de T o esperar que se recupere el cordinador si es necesario.
Si una localidad activa tiene <EJECUTAR T> en su bitácora, entonces T debe ser ejecutada.
Si una localidad tiene <ABORTAR T> entonces T debe ser abortada.
Si alguna localidad no tiene <lista T> entonces el coordinador no pudo haber decidido ejecutar T. Es preferible abortar T antes que esperar a que el coordinador se recupere.
Si todas tienen <lista T> pero ninguna <ejecutar T> o <abortar T>, entonces es imposible determinar hasta que el coordinador se recupere si hubo una decisión y si la hubo cuál fue. Por lo que T se queda bloqueada (y puede estar bloqueando datos). A esta situación se la denomina problema de bloqueo.

Fallo de una linea de comunicación: cuando pasa esto todos los mensajes que enrutaba no llegan intactos a su destino

Fragmentación de la red: posibilidades cuando se fragmenta la red:

- Si el coord. y los participantes quedan en un mismo fragmento, no pasa nada.
- Si no se perderan los mensajes; y esta situación es equivalente a el fallo en las líneas de comunicación.

Cuando la transacción esta parcialmente cometida

Compromiso de tres fases

Esta diseñado para impedir el problema de bloqueo que hay en el protocolo de dos fases. El protocolo requiere que:

- No pueda ocurrir una fragmentación en la red.
- Debe haber al menos una localidad funcionando en cualquier momento.
- Como máximo, un número K de localidades se pueden caer simultáneamente.
-

Fase 1: idéntica a la del protocolo de dos fases.
Fase 2: si Ci recibe un mensaje "abortar T", o si Ci no recibe respuesta dentro de un intervalo de tiempo, entonces Ci decide abortar T.

Si Ci recibe “lista T” de todas las localidades participantes, Ci agrega un registro <preejecutar T> a la bitácora. Luego envía un mensaje “preejecutar T” a las localidades. La decisión de preejecutar le permite al coordinador informar que todas las localidades participantes están listas.

Cuando una localidad recibe abortar T o preejecutar T, lo pone en su bitácora; y luego envía en mensaje de “reconocimiento a T” al coordinador.

Fase 3: sólo se ejecuta si la decisión de la fase 2 fue la de preejecutar. En este momento el coordinador debe esperar recibir por lo menos K de mensajes de “reconocimiento a T”. Después de esto, el coord. agrega <ejecutar T> a la bitácora y envía el mensaje “ejecutar T” a todas las localidades participantes. Cuando una localidad recibe este mensaje lo registra en su bitácora.

En algunas implementaciones, al recibir una localidad el mensaje “ejecutar T” ésta envía un mensaje “recibo T” al coordinador. Cuando el coordinador recibe este mensaje de todas las localidades, agrega <T completa> a la bitácora.

MANEJO DE FALLOS:

Fallo de una localidad participante: cuando una localidad falla tiene que revisar su bitacora para determinar el destino de las transacciones que se ejecutaban cuando fallo:

-si tiene <ejecutar T> se ejecuta **redo(T)**.

-Si tiene <abortar T> se ejecuta **undo(T)**.

-Si tiene <lista T> y ningún <abortar T> o <preejecutar T>, la localidad debe consultar con Ci para determinar el destino de T. Si Ci responde con un mensaje de abortada entonces hay que ejecutar **undo(T)**. Si responde con un mensaje de preejecutar, la localidad lo registra en su bitácora y continua el protocolo enviando un mensaje de reconocimiento a Ci y se sigue con los pasos del protocolo.

-Tiene <preejecutar T> y ningún registro <abortar T> o <ejecutar T>. La localidad debe consultar con Ci. El coord. va a responder si hay que ejecutarla o abortarla (se pone el registro en la bitácora y se hace redo o undo de la transacción). Si Ci responde que T esta en estado de preejecución, la localidad continuara el protocolo en ese punto.

Si el coordinadoe no responde dentro de un intervalo de tiempo establecido, la localidad que esperaba la respuestaejecutará el protocolo de fallo del coordinador para elegir uno nuevo

PROTOCOLO DE FALLO DEL COORDINADOR:

este protocolo selecciona un nuevo coordinador. Cuando el coord. que fallo se recupera, toma el papel de una localidad común.

1. las localidades activas eligen un nuevo coord.
2. el nuevo coord. envía un mensaje a cada localidad pidiendo el estado de T.
3. Cada localidad incluyendo Cnuevo determina el estado local de T: ejecutada, abortada, lista (la bitácora no tiene <abortar> ni <preejecutar> pero si tiene <lista>), preejecutada (no tiene <abortar> ni <ejecutar>), no-lista (no tiene <lista> ni ningun <abortar>)
4. Cada localidad participante envía su estado local a Cnuevo.
5. Dependiendo de la respuesta recibida, Cnuevo decide si ejecutar o abortar o reiniciar el protocolo de compromiso:

-si al menos una localidad tiene <ejecutada> entonces se ejecuta T.

-Si al menos una localidad tiene <abortada> se aborta T.

-Si ninguna tiene <abortada> o <ejecutada> pero alguna tiene <preejecutada> entonces Cnuevo reanuda el protocolo de compromiso enviando mensajes “preejecutar” nuevos.

-En los demás casos, se aborta T.

Si ninguna localidad ha recibido “preejecutar” desde Ci, es posible que Ci decidiera abortar T antes del fallo. El protocolo aborta T antes de causar un bloqueo.

No se permite que la red este fragmentada porque eso puede conducir a la eleccion de 2 coordinadores. Además, es muy importante la eleccion del parametro K puesto que si estuvieran activos menos de K participantes, se produciria un bloqueo.

Igualmente, se utiliza más el protocolo de dos fases .

CONTROL DE LA CONCURRENCIA

Protocolos de bloqueo

Se supone que existen el bloqueo compartido y el exclusivo.

1-Esquema sin repetición: Cada localidad tiene un gestor de bloqueo localencargado de administrar el bloqueo y desbloqueo para sus datos. Una localidad solicita bloquear datos a otra localidad enviando un mensaje al gestor de bloqueos de esta solicitando un bloqueo, cuando se logro el bloqueo el gestor envía un mensaje a la localidad que lo origino. Es fácil de implementar. (requiere el envio de 2 mensajes para bloqueo y 1 para desbloqueo).

2-Enfoque de coordinador único: el sistema tiene un único gestor de bloqueos que reside en una unica localidad L. Todas las solicitudes de bloqueo y desbloqueo van a la localidad L y el gestor de bloqueos determinara si puede hacerse de inmediato. Si esto es posible envia un mensaje a la localidad que lo origino , sino se posterga la solicitud hasta que pueda atenderse. Una transaccion puede leer el dato en cualquier localidad donde halla una copia, pero se debe escribir en todas las localidades donde halla una copia.
Ventajas➔ sencillez de inplementacion (para el bloqueo también se necesitan 2 mensajes y uno para el desbloqueo) sencillez del manejo de bloqueos (el bloqueo y desbloqueo se hace en cada localidad).

Desventajas ➔ Se centraliza el bloqueo (cuello de botella) y es vulnerable a caídas del gestor.

Una alternativa es usar el *enfoque de coordinadores múltiples*, en el cual el manejo de bloqueos se distribuye entre varias localidades.

3-Protocolo de mayoría: hay un gestor de bloqueos p[ara cada localidad, que administra los bloqueos de los datos de esa localidad. Cuando se quiere bloquear un dato que esta en n localidades se debe enviar una solicitud de bloqueo a n/2 de ellas,. Cada uno de los gestores de bloqueo determina si la solicitud se puede atender, sino se posterga. La transacción no operará sobre el dato hasta que haya logrado el bloqueo de N/2+1 datos

La ventaja es que los datos repetidos se manejan en forma descentralizada.

Las desventajas de este protocolo son: hay que enviar muchos mensajes (2(n/2+1)) para realizar bloqueo y desbloqueo. Puede haber deadlocks.

4-Protocolo preferencial: es como el protocolo de mayoría con la diferencia de que se le da un tratamiento especial a las solicitudes de bloqueos compartidos que a los exclusivos cda localidad tiene un gestor de bloqueos que administra los datos almacenados en ella. Los bloqueos compartidos y exclusivos son manejados de la siguiente forma:

- Bloqueos compartidos: alcanza con bloquear una copia de los datos
- Bloqueos exclusivos: hay que bloquear todos las copias del dato.

La ventaja es que las operaciones de lectura consumen menos tiempo extra que el protocolo de mayoría.
La desventaja es el tiempo extra de las operaciones de escritura. Tiene deadlock y es dificil manejarlos.

5-Copia primaria: se elige una de las copias como la *copia primaria* la cual reside en la localidad primaria a la que se le pide el bloqueo cuando se necesita. Si la localidad primaria cae no se podrá tener acceso al dato aunque existan otras localidades con copias de este dato. El control de concurrencia se maneja como si no hubiera copias de los datos.

Asignación de horas de entrada

Se pueden aplicar los protocolos de hora de entrada a un esquema distribuido asignando a cada transaccion una hora de entrada unica. Pero primero hay que desarrollar un esquema para generar horas de entrada únicas.

Hay dos métodos para la generación de entradas únicas, uno centralizado y el otro distribuido. En el centralizado se elige una sola localidad para que se encargue de distribuir las horas de entrada.

En el distribuido cada una de las localidades genera una hora de entrada única local. Las horas de entrada global única se obtienen concatenando las locales con el identificador de localidad, que debe ser único. El identificador se pone en la parte menos significativa.

Manejo de bloqueos

Si se permite que ocurran bloqueos en un sistema distribuido, el principal problema es como se sva a mantener el grafo de espera. Cada localidad mantiene un grafo de espera local donde los nodos son las transacciones que usan los datos de esa localidad.
Si en el grafo local hay un ciclo hay bloqueo, pero que no halla ciclos no significa que no halla bloqueos. La union de grafos aciclicos puede dar como resultado un grafo ciclico. Formas para organizar el grafo del sistema distribuido:

1 ENFOQUE CENTRALIZADO
se construye y se mantiene un grafo de espera global (union de los locales) en una sola localidad que sera en coordinador de deteccion de bloqueos. Este grafo puede construirse:

- cada vez que se inserta o elimina una arista en alguno de los grafos de espera locales.
- Periodicamente, luego de una cierta cantidad de cambios en el grafo local
- Siempre que el coordinador invoque el algoritmo de deteccion de ciclos.

Siempre que se invoque el algoritmo de deteccion de ciclos el coordinador examina su grafo global y si encuentra un ciclo elige una victima para retroceder decisión que debe informar a todas las localidades, y las localidades jhacen retrocederlas.
Puede ser que se produzcan retrocesos innecesarios.

2.ENFOQUE DE DISTRIBUCION TOTAL
todos los controladores comparten la responsabilidad de detectar los bloqueos. Cada localidad construye un grafo de espera local. Si existe un bloqueo aparece un ciclo en por lo menos un d los grafos parciales.

Selección del coordinador

Si el coordinador falla, se puede continuar la ejecucion si se activa otro

COORDINADORES DE COPIA DE SEGURIDAD (BACK-UP)
Es una localidad que ademas de otras tareas mantiene la informacion que le permite asumir el papel de coordinador (ejecuta los mismos algoritmos y mantiene la misma informacion de estado que el coordinador real).
Ventaja==: el proceso puede reanudarse inmediatamente.
Desventaja ➔ tiempo extra que requiere la doble ejecucion de la tareas del coordinador.

ALGORITMOS DE ELECCION
Asignamos un numero de identificacion a cada localidad activa del sistema elige la localidad del nuevo coordinador cuando el real queda fuera de servicio, eligiendo la localidad activa con ID mas alto.
Algoritmo de prepotencia ➔ cuando una localidad envia un mensaje al coordinador y este no responde dentro de un intervalo de tiempo, este esta fuera de servicio. La localidad manda un mensaje de eleccion a todas las que tengan ID mas alto y si no recibe respuesta de ninguna se elige a si misma como coordinador.

SISTEMAS DE BASES DE DATOS MULTIPLES

Un sistema de bases de datos múltiple (MDBS) crea la ilusión de una integración lógica de bases de datos sin requerir la integración física.

Los DBMS locales pueden emplear diferentes modelos lógicos, definición de datos y lenguajes de manipulación de datos, etc.

Cada DBMS local puede estar usando un modelo de datos diferente. El MDBS debe utilizar un modelo común de datos. Se utiliza el modelo relacional con SQL como el lenguaje común de consultas.

Un MDBS mantiene dos tipos de transacciones:

- Transacciones locales: que se ejecutan por cada DBMS, fuera del control del MDBS.
- Transacciones globales: que se ejecutan bajo el control del MDBS.

El criterio de corrección para la concurrencia es la serializabilidad. Una condición suficiente para la serializabilidad global es que en todas las planificaciones locales el orden de serializabilidad de la transacción global es la misma.

Reglas para distribucion de datos de Date

Las debe cumplir un DBMS para estar en un entorno distribuido

1- Autonomía local: cada localidad debe funcionar sin importar el contexto, cada una es independiente entre ellos cada BD local es manejada por su DBMS.
2- No se necesita un sitio central (ya que si lo hay se pueden producir fallos y cuello de botella).
3- Operación continua (un sistema de BD no debe quedar fuera de servicio)
   Se debe asegurar:
   - soportar backup online o incremental.
   - soportar recuperación rápida.
4- Independencia de la ubicación de los datos
   - El diccionario de datos debe mantener una tabla con los elementos de datos, sus alias y sus ubicaciones.
   - El diccionario debe mantener y utiliza los datos aun cuando están en movimiento.
5- Independencia de la fragmentación de los datos (se debe poder encontrar los datos a pesar de estar fragmentados). El DBMS debe funcionar sin importar donde estan los datos.
6- Independencia de la replicación de datos: (el DBMS usa la mejor replica para resolver la consulta)
   - Los datos se pueden replicar transparentemente
   - El sistema debe actualizar los datos
   - Lockeo
7- Procesamiento de consultas distribuidas
   - La performance de una consulta debe ser independiente de la localidad donde se realiza
8- Manejo de transacción distribuida
   - Soportar transacciones atómicas

- Protocolo de 2 o 3 fases

9- Independiente de Hardware ➔ un DBMS debe correr independientemente del hardware que tenga disponible.
10- Independiente del SO➔ un DBMS debe correr independientemente SO que tenga disponible.
11- Independiente de la red ➔ un DBMS debe correr independientemente del protocolo de comunicacion disponible..
12- Independiente del DBMS (El modelo de datos debe ser unico homogéneo, se debe independizar del DBMS a usar)

Cuando el modelo de datos no es homogéneo se llaman modelos FEDERATIVOS (BD Federales). Se solucionan poniendo conversores de modelos.

Unidad 9.warehousing

Bases de datos estadísticas

Aseguran intimidad de los datos, solo pueden accederse con fines estadísticos
Soluciones

- El sistema rechaza cualquier consulta que involucre menos de un número predeterminado de individuos
- *Polución* de datos: falsificación aleatoria de datos en la respuesta.
- Moficación aleatoria de la consulta en si misma

Modelo de Red

Consta de una colección de registros los cuales están conectados mediante enlaces (solo entre dos registros). Los registros son similares a las entidades en el modelo ER. Cada registro esta formado por campos los cuales contienen un solo valor.

La estructura de un modelo de red se representa mediante un diagrama de estructura de datos, que posee dos componentes: Cajas (corresponde a los registros). Líneas (corresponde a enlaces). Estos diagramas tiene el mismo propósito que los diagramas ER (especificar la relación entre los datos)

La cardinalidad en los enlaces se puede expresar poniendo punta de flechas a los enlaces. Así si el enlace no tiene puntas (-----) entonces es de muchos a muchos. Si el enlace es A<- B entonces la relación de uno a muchos de A a B. Si el enlace es A <-> B entonces es de uno a uno.

Los diagramas tienen estructura de grafo.

Modelo Jerárquico

Consta de una colección de registros los cuales están conectados mediante enlaces. A diferencia del de red la estructura de los diagramas es en forma de árbol. La información de un registro particular puede tener que repetirse (posibilidad de inconsistencia y desperdicio de lugar).

El modelo de red se representa mediante un diagrama de estructura de árbol. Las los enlaces solo se pueden hacer de uno a muchos o de uno a uno, no se permiten las relaciones n a n . El hijo siempre debe tener una flecha (con punta ->) al padre, el padre puede o no tener fecha al hijo (.->)

Nuevas aplicaciones en BD

Hasta ahora Modelos de datos con:

- Uniformidad de datos (estructura y tamaño)
- Orientación de Registros (long. Fija)
- Elementos de datos de pequeño tamaño
- Campos atómicos (1 FN)

Nuevas características que debemos reflejar: CAD, CASE, BD multimediales, Hipertextos,…
Se Necesita:

- Tipos de datos complejos
- Encapsulación de datos y TAD
- Nuevos métodos de Indexación y consulta

Modelo OO (BD relacionales Orientadas a Objetos)

Algunos conceptos

- Un objeto se corresponde a una entidad en el modelo E-R.
- El paradigma OO está basado en *encapsulación* de código y datos relacionados a un objeto en una unidad simple
- El Modelo de datos OO es un modelo lógico (como ER)
- Se adapta el paradigma de programación OO a sistema de BD.**Conceptos de OO**: objeto, clase, métodos, mensajes, herencia simple, herencia múltiple, etc.

Identidad de Objetos

- Un objeto retiene su identidad aún si el valor de variables o definición de métodos cambia a lo largo del tiempo.
- Identidad de objetos es un concepto de identidad más potente que de los lenguajes de programación o modelo de datos no OO
- Un identificador de objeto puede ser almacenado como un campo de otro objeto, o referir a otro objetos

Aplicaciones nuevas

Sistemas de ayuda a las decisiones

- Data marts
- Data warehouse
- Data mining

Bases de datos especiales (GIS)
Bases de datos multimedia
Bases de datos portátiles
Recuperación de la información
Recuperación de la información distribuida.

Data warehouse: (almacén, repositorio de datos)

- Un lugar donde las personas pueden acceder a sus datos
- Una BD que contiene los datos históricos producidos por los sistemas informáticos de las empresas
- Son datos administrados que están situados fuera y después de los sistemas operacionales.

Warehousing

Se remite al adecuado manejo de la información histórica generada por una organización, a fin que esta pueda otorgarle a sus ejecutivos las herramientas adecuadas para el proceso de toma de decisiones
DW: un tema de negocios
Cuando las necesidades del mercado y los progresos tecnológicos convergen, se dan los cambios más importantes en las prácticas de negocio.

Antecedentes

- Evolución de las clases de aplicaciones en las organizaciones
- Evolución de la tecnología de software y hardware
- Cambios en la naturaleza de los negocios (globalización)

Casos que se presentan

Tenemos montañas de datos pero no tenemos acceso a ellos.
Todos saben que algunos datos no son confiables.
Solo mostrarme lo que es importante.

Nada enloquece más a los gerentes que dos personas presentando el mismo resultado de negocios pero con número distintos.

Objetivos

- Brindar acceso a los datos (eficiente, fácil, consistente, confiable)
- Brindar soporte a la toma de decisiones, por ejemplo permitiendo realizar predicciones de operaciones futuras de un modo racional
- Encontrar relaciones entre los datos producidos por diferentes áreas, determinando patrones que permitan predecir futuros sucesos
- Identificar nuevas oportunidades de negocio
- Monitorear las operaciones actuales de negocio y compararlas con las operaciones efectuadas previamente
- Reducción de los costos de gestión de la información Aumento de la productividad
- Posibilidad de separar y combinar los datos por medio de toda posible medida en el negocio. Por ejemplo para análisis del problema en términos de dimensiones.
- Reducción de los costos de gestión de información.

Si el DW no logra transformar los datos en información para hacerla accesible a los usuarios en un tiempo lo suficientemente rápido como para marcar una diferencia, no sirve

Algunas definiciones

- Es un ambiente estructurado y extensible diseñado para el análisis de datos no volátiles, transformados lógica y físicamente desde varias aplicaciones fuentes para alinearse con la estructura de negocio, actualizado y mantenido por un período de tiempo largo, expresado en términos de negocio simples y resumido para un análisis rápido.
- Es una colección de datos orientada a sujetos, integrada, variante en el tiempo, no volátil para soportar el proceso de toma de decisiones.
- Es un proceso y no un producto. Es una técnica de ensamblar y gestionar adecuadamente los datos procedentes de distintas fuentes con el objeto de tener una visión única de los mismos para toda la empresa.

Componentes

- BD de datos fuentes (producción)
- BD con datos resumidos (DW)
- Interfases de acceso (consultas, reportes, análisis multidimensional, Data Mining)

Elementos que participan en la construcción de un DW

- Recursos Humanos
- Metodología de trabajo
- Herramientas de Hardware y software
- Infraestructura de la organización en general

Etapas de diseño de un DW

- Análisis
- Diseño del modelo de datos
- Selección y extracción de datos
- Limpieza y Transformación
- Creación de los Metadatos
- Carga
- Actualización

❋ **Análisis**

- Necesidades de la organización
- Áreas involucradas y el personal interviniente
- Evaluación de herramientas y arquitectura tecnológica

❋ **Diseño del modelo de datos**

- Forma de manipular la información por cada área de la organización
- Compatibilización de modelos
- Generación de la estructura que sirva como modelo de datos

❋ **Selección y extracción de datos**

- Información transaccional (operatividad)
- Actualizacón de datos vs. extracción
- Performance (transacciones en línea)
- Creación de vistas (espacio adicional vs. mejor tiempo de acceso)

❋ **Limpieza y transformación**

- Calidad de datos de fuentes diferentes
- Técnicas de limpieza (errores inherentes en los sistemas, errores de validación de dominios, tipeos incorrectos)Data scrubbing: proceso de filtrado, decodificación y traducción de datos fuentes para crear datos validados
- Homogeneizar tipos de datos
- Valores por defecto inteligentes
- Períodos de tiempo completos.

❋ **Creación de metadatos**

- Proveen información sobre los datos almacenados en el DW

❋ **Carga**

- Preprocesamiento adicional
- Checkpoint
- Secuencias o paralelo

❋ **Actualización**

- Periodicidad
- Cumplimiento de etapas con los nuevos datos

Data Marts

- Satisfacen necesidades específicas de un área. Pueden ser un subconjunto de un DW.
- Pueden cohexistir varios DM dentro de una empresa, cada área de la misma puede tener uno.

Data Mining

- Extracción no trivial de información implícita previamente desconocida y potencialmente útil de la BD
- Los datos ocultan desviaciones, tendencias, anomalías, que se descubren con:
- Reglas de inducción
- Redes neuronales, etc.

Ventajas

- Ayuda a soportar el proceso de toma de decisiones

- Identificación de nuevas oportunidades en el mercado
- Estructuras cooporativas flexibles: distintos departamentos en una organización comparten la misma información
- Generación de informes críticos sin costo de tiempo
- Formato de datos consistente
- Servir de plataforma efectiva de mezclado de datos desde varias aplicaciones corrientes
- Posibilidad de encontrar respuestas más rápidas y más eficientes a las necesidades del momento
- Reducción de costos globales para la organización

Conclusiones

- La necesidad de sistemas de información que permitan generar consultas, reportes y análisis para la toma de decisiones es cada día más apremiante para las empresas que quieren competir exitosamente en el mercado. La implantación de un DW surge como la solución a las necesidades antes mencionadas
- Hoy en día desarrollar un DW es raramente rápido y fácil. Pocas instalaciones toman menos de seis meses (la mayoría toma dos años o más)
- No siempre en las organizaciones existe una conciencia real de la potencialidad que tienen encubierta en los datos transaccionales generados cotidianamente.
    - Las organizaciones de tecnología informática pueden o no tener todas las cualidades técnicas necesarias, pero no implementarán un proyecto de DW exitoso sin que logren que la unidad de negocios se involucre.
- Se debe tener en cuenta que un DW no es un producto que se compra, debe ser planificado cuidadosamente de acuerdo a cada organización y construido a tal efecto.
- Cuando se diseña un DW hay que tener en claro que se está convirtiendo la información generada en forma cotidiana en conocimiento para la organización.
- Pensar en un DW como una simple liberación de los datos corporativos sería un error. El valor real de un DW recién se descubre cuando lo utiliza alguien que puede encontrar detalles importantes en los datos y marcar las diferencias.

Unidad 10. Archivos y sistemas de archivos

Almacenamiento primario (ram): Tiene acceso rápido, su costo es mayor que la memoria secundaria y su capacidad es menor que ésta. En general son memorias volátiles.

Almacenamiento secundario: Las capacidades de estas memorias están en le orden de los giga.Hay que tratar de reducir los accesos a este tipo de memoria. Estas memorias tienen la característica de ser no volátilies.

Archivos

Los datos colocados en el almacenamiento secundario se reúnen en archivos.

Un archivo se define como una colección de información relacionada, de acuerdo con las siguientes definiciones:

- ☞ **Una colección de registros que abarca un conjunto de entidades con ciertos aspectos en común y organizados para algún propósito particular.**
- ☞ **Una colección de registros, guardados en dispositivos de almacenamiento secundario o dispositivos de memoria secundaria.**

Cuando se diseñan los archivos hay que tratarlo de hacer de modo tal que cuando se quieran acceder a los datos se pueda hacerlo utilizando la menor cantidad de accesos posibles.

Archivos Físicos y Lógicos

⇨ **Archivo Físico**: Archivo que en realidad existe en el almacenamiento secundario. Es el archivo tal como lo conoce el S.O y que aparece en su directorio de archivos.

⇨ **Archivo Lógico**: Es el archivo visto por el programa. El uso de archivos lógicos permite a un programa describir las operaciones que van a efectuarse en un archivo sin saber cual archivo físico real se usará. El programa puede entonces usarse para procesar cualquiera de diversos archivos que comparten la misma estructura.

Dispositivos de almacenamiento secundario

Discos

Las unidades de disco consisten en un conjunto de cabezas de lectura y escritura, interpuestas entre uno o más platos. Cada plato contribuye con una o dos superficies, donde cada una contiene un conjunto de pistas concéntricas, y cada pista está dividida en sectores o bloques o buffers. Al conjunto de pistas concéntricas que pueden leerse sin mover las cabezas de lectura y escritura se le llama cilindro.
Hay dos formas de hacer referencia a datos en los discos:

× ***Organización por bloques***

El término **bloque** se refiere a un grupo de registros que se almacenan juntos en un disco y son tratados como una unidad, para propósitos de E/S. Al usar bloques, es más facil para el usuario hacer que la organización física de los datos corresponda con su organización lógica y, en consecuencia, se puede lograr mejor desempeño.
Algunas veces también la organización de las unidades de disco por bloques permite a la unidad de disco buscar entre los bloques de una pista un registro con cierta llave sin tener que transmitir primero los bloques no deseados a la memoria RAM.

✓ Ventajas:

- Los bloques pueden tener longitud fija o variable, por lo tanto esta organización no presenta los problemas de distribución de sectores y de fragmentación, porque el tamaño de los bloques puede variar para ajustarse a la organización lógica de datos.
- Los bloques son superiores a los sectores cuando se pretende que la asignación física del espacio para registros corresponda con su organización lógica

✓ Desventajas:

- Peligro de fragmentación interna de la pista.
- Algo mas complejo para el trabajo del usuario.
- Pérdida de oportunidades para lograr algunos tipos de sincronización que si proporciona la organización por sectores.

× ***Organización por sectores***

⇨ ***Disposición física de los sectores*****:➔** hay diversos puntos de vista respecto a la organización de los sectores en una pista. El más sencillo propone que los sectores sean segmentos de pista adyacentes de tamaño fijo, capaces de contener un archivo. Como el computador no puede leer los sectores físicos adyacentes, se **intercalan** los sectores dejando un intervalo de varios sectores físicos entre los sectores lógicamente adyacentes.

⇨ ***Cúmulos*****: ➔**cuando un programa accede a un archivo, el administrador de archivos del SO hace la correspondencia entre partes lógicas del archivo y sus posiciones físicas. Para ello considera el archivo como una serie de cúmulos de sectores. Un cúmulo es un número fijo de sectores contiguos y todos los cúmulos de un disco son del mismo tamaño.

⇨ ***Extensiones*****: ➔**si un disco cuenta con mucho espacio disponible, sería posible que se constituyera un archivo completo de cúmulos contiguos. Cuando esto sucede, se dice que el archivo se compone de una extensión.

⇨ ***Fragmentación*****:➔** en general, todos los sectores de una unidad de disco determinada deben contener el mismo número de bytes.

Si no hay correspondencia entre los tamaños de registros y sectores, se puede:

- Almacenar sólo un registro por sector (cualquier registro puede extraerse con sólo recuperar un sector pero produce **fragmentación interna.**
- Permitir que los registros se trasladen entre sectores, de modo que el principio de un registro pueda encontrarse en un sector y el final de otro (no produce fragmentación interna pero algunos registros sólo pueden extraerse mediante el acceso a dos sectores)

× ***Sobrecarga de datos usados para control***

Tanto los bloques como los sectores ocupan una cierta cantidad de espacio en disco en forma de sobrecarga por datos de control, que consiste en información que se almacena en el disco al dar formato previo, el cual se realiza antes de usar el disco.

× ***El costo de un acceso al disco***

El costo de un acceso al disco puede medirse en términos del tiempo que consume el desplazamiento, el retraso por rotación y el tiempo de transferencia. Si se usa la intercalación de sectores, es posible acceder a sectores lógicamente adyacentes separándolos físicamente con uno o más sectores. Aunque el tiempo de acceso a un solo registro en forma directa es mucho menor que en forma secuencial, el tiempo de desplazamiento adicional requerido para efectuar el acceso directo hace que éste sea mucho más lento que el secuencial cuando se accede a una serie de registros.

Cintas

No son tan importantes como los discos y tienen un lugar destacado en el procesamiento de archivos, además de ser baratas, rápidas para el acceso secuencial, resistentes y fáciles de guardar y transportar.
Los datos almacenados en cinta normalmente se organizan en pistas paralelas de un bit como uno o más bytes. Para estimar la velocidad de procesamiento y la utilización del espacio, es fundamental reconocer el papel que desempeña el hueco entre bloques.
Las diferencias de desempeño ente las unidades se miden en términos de:

- Densidad de la cinta
- Velocidad de la cinta
- Tamaño del hueco entre bloques.

El viaje de un Byte

PROCESADOR DE E/S→ dispositivo usado para la transmision de informacion desde o hacia el almacenamiento externo. Independencia de la CPU.

CONTROLADOR DE DISCO→ encargado de controlar la operación de disco (colocarse en la pista→colocarse en el sector→transferencia al disco).

CAPAS DEL PROTOCOLO DE TRANSMISION.

Se puede describir la participación de los diversos programas y dispositivos que intervienen en el viaje que realiza un byte cuando se envía desde la memoria RAM hacia el disco:

- El **programa del usuario**, que hace la llamada inicial al sistema operativo;
- El **administrador de archivos del SO**, el cual mantiene y maneja tablas cuya información emplea para pasar del punto de vista lógico que el programa tiene del archivo hasta el archivo físico en donde se almacenará el byte;
- Un **procesador de E/S y su software**, que sincronizan la transmisión de un byte entre un buffer de E/S en memoria RAM y disco;
- El **controlador del disco y su software**, que dan instrucciones a la unidad acerca de cómo encontrar la pista y el sector apropiados, para después enviar el byte (obtetener la FAT del ultimo sector del disco , lugar a escribir) y
- La **unidad de disco**, que recibe el byte y lo deposita en la superficie del disco.

Manejo de Buffers

Cuando la entrada y la salida se guardan, en lugar de enviarse inmediatamente a su destino, se dice que se hace un manejo de buffers.
Manejar buffers implica trabajar con grandes grupos de datos en memoria RAM para que el número de accesos al almacenamiento secundario se reduzca.
Es común que los administradores de archivos asignen varios buffers para realizar E/S.
Existen problemas cuando un programa efectúa al mismo tiempo la entrada y la salida de un dato, y sólo hay un buffer de E/S disponible (Cuello de botella). Por esta razón, los sistemas de E/S casi siempre usan al menos dos buffers: uno para la entrada y otro para la salida.
Algunas técnicas de administración para mejorar el desempeño comprenden:

✓ El manejo doble de buffers

- ✓ El manejo de buffers en un depósito común
- ✓ El manejo de buffers empleando modo de direcciones.

Conceptos fundamentales de estructuras de archivos

Un archivo como secuencia de bytes

El nivel más bajo de organización que por lo general se impone a un archivo es el de una secuencia de bytes. Por desgracia, al almacenar los datos en un archivo sólo como una secuencia de byte, se pierde la capacidad de distinguir entre las unidades fundamentales de información de los datos (comienzo y fin de cada dato).
Campo➔ unidad fundamentales de .Los campos se agrupan para formar registros. Existen varias formas de separar un campo de otro y un registro de otro. **Estructura de campos:**

- Fijar la longitud de cada campo **(Campos fijos)**: si se fuerza que los campos tengan longitudes predecibles, entonces se pueden recuperar del archivo con sólo contar hasta el final del campo. *Desventajas*: el archivo crece mucho y además, algunos datos muy grandes pueden no caber en el espacio asignado.
- Comenzar cada campo con un indicador de longitud **(Indicador de longitud al inicio)**: puede que alcance con un byte.
- Separar los campos con delimitadores **(Delimitador de fin de campo)**: se elige un carácter especial que no aparezca como carácter legítimo dentro de un campo y se inserta dentro del archivo luego de escribir cada campo.

Registro➔conjunto de campos agrupados que definen un elemento del archivo. **Estructuras de registros**
Igual que con los campos:

- Fijar la longitud de cada campo
- Comenzar cada campo con un indicador de longitud
- Separar los campos con delimitadores

Y además

- Usar un segundo archivo para mantener información sobre las direcciones: se puede emplear un segundo archivo de índice para mantener información sobre la distancia en bytes de cada registro en el archivo original. Se busca la posición de un registro en el índice y después se alcanza el registro dentro del archivo de datos.

ARCHIVO SERIE ➔ los registros son accesibles solo luego de procesar su anterior.
ARCHIVO SECUECIAL➔ los registros son accesobles en orden de alguna clave.

Extracción de registros por clave: forma canónicas para llaves

Cuando se examina un registro individual, es conveniente identificarlo con una ***Llave*** que se base en el contenido del registro.
Se debe definir una forma estandar para llaves, paralelamente con las reglas y procedimientos asociadas para convertilas a esta forma. Esto se llama ***forma canónica de la llave***, que es la representación única para esa llave que se ajusta a la regla.
Este requisito de unicidad se aplica sólo a las llaves primarias. Una ***llave primaria*** es la llave que se usa para identificar unívocamente un registro. También es posible buscar con *llaves secundarias*, pero estas no identifican un único registro.

Acceso secuencial

Significa leer el archivo desde el principio y continuar hasta que se halla leido todo lo que se necesita.

Busqueda secuencial – Evaluación del desempeño

Ahora que se tiene acceso al concepto de llave canónica, se está preparado para escribir un programa que, registro por registro, busque una llave en particular.
Para desarrollar una medida de desempeño (secuencia de puntos de comparacion para medir las mejoras que se hacen) se requiere elegir una unidad de trabajo que represente de manera útil las restricciones del desempeño del proceso total: dado que el costo de una comparación en RAM es muy pequeño en relación con el costo de un acceso a disco, se cuentan las llamadas a la función READ() de bajo nivel, suponiendo que cada llamada requiere un desplazamiento del brazo del disco y que todas tienen el mismo costo.

En general, el trabajo requerido para buscar en forma secuencial un registro en un archivo con n registros es proporcional a n; implica, a lo sumo, n comparaciones y, en promedio, alrededor de n/2 comparaciones. Se dice que una búsqueda secuencial es de orden O(n).
La búsqueda secuencial es demasiado costosa para la mayoría de las situaciones de estracción de información, aunque puede ser útiles en archivos en los que se busca poco o en archivos con pocos registros.

Mejora del desempeño de la búsqueda secuencial

El manejo de **registros en bloques** puede usarse para mejorar considerablemente el tiempo de E/S en una búsqueda secuencial.
La idea es leer un bloque de varios registros a la vez y después procesar ese bloque de registros en memoria RAM.
Aunque se mejora el desempeño, no cambia el orden de operación de búsqueda secuencial: no cambia el número de comparaciones que debe hacerse en memoria RAM, solo se mejora el acceso al disco.

Acceso directo

Se tiene **acceso directo** a un registro cuando es posible colocarse directamente en el inicio del registro y leerlo. El acceso directo es **O(1).**
Este acceso implica saber donde está el comienzo del registro requerido, para ello, cada registro posee un **número relativo de registro (NRR).** Si un archivo es una secuencia de registros, entonces el NRR de un registro proporciona su posición relativa con respecto al principio del archivo.
Si el archivo consiste en registros de *longitud variable*, el NRR indica la posición relativa del registro que se quiere en la secuencia de registros, pero se debe leer secuencialmente a lo largo del archivo, contando los registros que se llevan para tener el registro que se quiere.
Para que sea posible el acceso directo por NRR se necesita trabajar con registros de *longitud fija* y conocida para calcular la distancia en bytes del inicio del registro en relación con el inicio del archivo.
Dado un archivo de registros de longitud fija de ***tamaño r***, la ***distancia en bytes*** de un registro con un ***NRR de n*** es: **n x r.**

Registros de encabezado

Se usa al principio del archivo para mantener información general sobre ese archivo que se usara en un futuro..
Tiene una estructura diferente a la que tienen los registros de datos del archivo. Incluyen información como longitud de c/registro, fecha y hora de actualización, etc

Acceso y organización de archivos

Es importante ser cuidadoso con las diferencias entre el ***acceso al archivo*** y la ***organización del archivo***.
Se intenta organizar los archivos de manera que permitan los tipos de accesos necesarios para una aplicación en particular. Por ejemplo, una de las ventajas de la organización de registros de longitud fija es que permite tanto acceso secuencial como directo.

Mantenimiento de archivos y elimincion de registros

Archivos volátiles→ (ABM frecuentes, sujetos a muchos cambios
Archivos fuera de línea =estatico → pocas modificaciones, ABM poco o nada frecuentes

****

Llamadas de bajo nivel
READ(Archivo_fuente,Dir_destino,Tamaño)
WRITE(Archivo_destino,Dir_fuente,Tamaño)
****

La organización original de un archivo influye en cómo puede ser alterado, y también cómo puede responder al deterioro.
Un archivo ***volátil***, es decir, que esté sometido a muchos cambios, puede deteriorarse muy rápido, a menos que se tomen medidas para ajustar su organización a los cambios. En el otro extremo están los archivos **fuera de línea o estáticos**, que son sometidos a pocos cambios y no necesitan mantenerse actualizados.
En general, las modificaciones que pueden hacerse sobre un archivo son:

- ✓ Agregar un registro
- ✓ Actualizar un registro
- ✓ Eliminar un registro

Si la única clase de cambios posible en un archivo es agregar registros, no hay deterioro. Sólo cuando se actualizan registros de longitud variable, o cuando se eliminan registros de longitud fija o variable es que los aspectos de mantenimiento se vuelven complejos.
La ***fragmentación interna*** ocurre cuando existe espacio desperdiciado dentro de un registro.
La ***fragmentación externa*** ocurre cuando se crean espacios sin utilizar entre registros, normalmente debido a eliminaciones.
Existen varias formas de combatir la fragmentación. La más sencilla es la ***compactación*** del almacenamiento, la cual elimina el espacio no utilizado que ocasionó la fragmentación externa, agrupando todos los registros no eliminados.

Compactación de almacenamiento

Cualquier estrategia de eliminación de registros debe proporcionar una forma de reconocer los registros que se han eliminado. Un enfoque sencillo es colocar una marca especial en cada registro eliminado. Los programas que usan el archivo deben incluir cierta lógica para que se ignoren estos registros marcados.
Después de que los registros eliminados se han acumulado por algún lapso, se emplea un programa especial de ***compactación de almacenamiento*** que reconstruye el archivo ya sin los registros eliminados. Esto puede hacerse construyendo una copia del archivo o en el mismo archivo.

Panorama de la eliminación de registros de longitud fija y variable

La fragmentación puede tratarse ***dinámicamente*** reutilizando el espacio eliminado cuando se agregan registros. Debido a la necesidad de mantener información respecto al espacio que puede reutilizarse, este enfoque resulta más complejo que la compactación. Además, si un archivo tiene registros de longitud variable, no basta mantener una lista de entradas o posiciones de registros disponibles. Se necesita también desarrollar una estrategia de colocación para elegir un espacio del tamaño correcto para el registro nuevo.
Para proporcionar un mecanismo de eliminación de registros con la reutilización del espacio liberado, es necesario garantizar dos cosas:

- Que los registros eliminados se marquen de alguna forma especial, y
- Que se pueda encontrar el espacio que los registros eliminados ocupaban, para reutilizarlo cuando se agreguen registros.

Para tener una rapida localizacion de los espacios disponibles podemos usar las siguientes estructuras:
Listas ligadas ➔ tenemos una referencia al primer nodo y luego la recorremos secuencialmente usando un apuntador.
Pilas ➔ se extraen y ponen los elementos por un mismo extremo = tope.

Eliminación de registros de longitud fija

En un archivo de registros de longitud fija, todas las entradas son igualmente útiles; son intercambiables. Por consiguiente, la forma más simple de mantener la lista ligada de disponibles es tratarla como una pila Detalles a considerar:

- Se necesita un lugar adecuado para el apuntador al primer registro dosponible de la lista de diponibles.
- Tener una forma de marcar los registros eliminados y colocarlos en la lista de disponibles.
- Poder reutilizar el espacio de la lista de diponibles cuando se agrega un registro.
- Verificar si un registro fue eliminado antes de usarlo.

Puesto que es muy fácil encontrar el primer campo de un registro de longitud fija, la eliminación de un registro puede llevarse a cabo colocando una marca especial en el primer campo. Se usa un asterisco (“*****”) para marcar que un registro ha sido eliminado.
Como todos los registros de un archivo de longitud fija son del mismo tamaño, no es difícil reutilizar los registro eliminados. La solución consiste en agrupar todos los espacios disponibles en una lista de disponible, que se crea reuniendo todos los registros eliminados en forma de lista enlazada. Esta se forma acomodando campos de enlace en los espacios de registros eliminados (justo después del campo “*”) de tal forma que cada registro de la lista contenga un campo que proporcione el número relativo del registro del que sea el siguiente de la lista.

Eliminación de registros de longitud variable

Al igual que como se hace en los registros de longitud fija, se pueden colocar campos de “*” y enlazar los campos en los registros de longitud variable, aunque existe una diferencia importante. Con los registros de longitud fija no es necesario asegurarse de que una entrada sea del tamaño correcto para almacenar el nuevo registro. Por lo tanto, a través de funciones, puede desarrollarse un sistema para eliminar y reutilizar registros de longitud variable.
Una de las funciones, maneja la lista de disponibles como una ***pila***, colocando los registros recién eliminados al frente. Puesto que los registros se colocan en la lista de disponibles según un orden donde el recién eliminado es el primero, no están ordenados por tamaño.
La otra función, se usa cuando se necesita una entrada para guardar un nuevo registro, busca desde el inicio de la lista hasta que encuentra una entrada lo ***suficientemente grande*** para guardarlo o bien hasta que llegue al final.

Fragmentación del almacenamiento

La *eliminación* y *reutilización* de registros produce ***fragmentación*** dentro de los archivos.
Fragmentacion Interna: ocurre cuando se desperdicia espacio en un registro, se le asigna el lugar pero no lo ocupa totalmente. En registros de longitud fija debemos usar caracteres de relleno para completar el registro desde el ultimo campo hasta el final del mismo, esto es espacio desperdiciado. Para minimizarla usamos registros con una longitud lo mas aproximada posible a la que realmente se necesita.
Si usamos registros de longitud variable no tendremos fragmentacion interna porque el tamaño del registro sera el que realmente necesitamos.

- **Externa**: ocurre cuando el espacio que no se usa está entre los registros individuales. El espacio esta disponible no esta encerrado en ningun registro pero esta demasiado fragmentado como para reutilizarse. Existen varios caminos para ***minimizar la fragmentación externa:***

✯ ***Compactar*** el archivo mediante procesamiento por lotes, cuando el nivel de fragmentación se vuelve excesivo; es una forma de liberarse por completo de la fragmentacion externa recorriendo todos los registros para juntarlos de modo que no existan segmentos desperdiciados entre ellos.

✯ ***Union de huecos*** ➔ las entradas adyacentes (hueco) en la lista de disponibles pueden juntarse para formar segmentos más grandes y más útiles en general, y

✯ Adoptar una ***estrategia de colocación*** para seleccionar las entradas que se reutilizarán, de modo que se minimice la fragmentación antes que suceda.

Estrategias de colocación

Las siguientes son las estrategias de colocación para seleccionar el espacio de la lista de disponibles y almacenar un registro nuevo:

- ➔ Primer Ajuste: selecciona la primer entrada disponible que pueda almacenar el registro.
- ➔ Mejor Ajuste: Elige la entrada que más se aproxime al tamaño del registro.
- ➔ Peor Ajuste: selecciona la entrada mas grande, sin importar cuán pequeño sea el nuevo registro. Como esto deja la mayor entrada posible para reutilización, en algunos casos, el peor ajuste ayuda a minimizar la fragmentación externa.

Localización de datos en un archivo: introducción a la clasificación y a la búsqueda binaria

Llave: expresión derivada de uno o más campos dentro de un registro, que puede usarse para ubicar ese registro. A los campos usados para construir una llave se les denomina *campos de llave*. El acceso por llave proporciona una forma de recuperar información que se basa en el contenido de los registros, y no en su posición.

Llave primaria: llave que identifica *unívocamente* cada registro y que se usa como el método primario de acceso a los registros.

Forma canónica: Forma estándar para una llave que puede derivarse, con la aplicación de reglas bien definidas, a partir de la forma no estándar particular de los datos encontrados en un campo de llave del registro, o que puede ser proporcionada en una solicitud de búsqueda por parte del usuario.

La única forma de extraer o encontrar un registro con algo de rapidez, hasta el momento, es buscar por su ***NRR***. Pero es mucho mas probable que se conozca la identidad de un registro por su ***llave*** y no por su NRR.

El acceso por llave implica una *búsqueda secuencial*, examinando un registro tras otro hasta que se encuentra el que contiene la llave. Para mejorar esto, se utiliza un método para encontrar registros por llave: la ***búsqueda binaria.*** La búsqueda binaria requiere *O(log n)* comparaciones para encontrar un registro en un archivo con n registros y, por lo tanto, es muy superior a la búsqueda secuencial.

Problemas de la búsqueda binaria.

1- La búsqueda binaria es más eficiente que la secuencial, pero igual los accesos a disco son considerables.

2- El uso de búsqueda binaria tiene el costo adicional de mantener el archivo ordenado. Si al archivo se le insertan registros frecuentemente entonces se tendrá tiempo extra de procesamiento.

3- El uso de la búsqueda binaria esta limitada a que el archivo se pueda ordenar.

Como la búsqueda binaria funciona sólo en un archivo ordenado, es absolutamente necesario un procedimiento de ordenación. El problema de la clasificación se dificulta debido a tres factores:

⇨ Se desea clasificar un archivo con base en llaves formadas canónicamente, que corresponden a cada registro, y no con base en el contenido de los registros mismos.

⇨ Se desea que la clasificación se haga rápidamente, lo que implica no efectuar mucho movimiento físico de los datos. Más específicamente, no se quiere estar moviendo y copiando campos completos de la llave

⇨ Se está clasificando archivos en el almacenamiento secundario en lugar de hacerlo en vectores en memoria RAM. Se necesita desarrollar un procedimiento de clasificación que no requiera múltiples desplazamientos sobre el archivo.

Para responder a estos requisitos, se desarrolla un programa de ***clasificación*** en memoria RAM que emplea *indirección*. Se leen los registros del archivo secuencialmente en un arreglo, formando al mismo tiempo un *vector de llaves*, que están relacionadas por su posición con el registro del que son imagen. También se forma un *vector de subíndices* relacionándose con las posiciones dentro del vector de llaves. Aunque la clasificación es por llave, el vector que se reordena es el de subíndices.

Uno de los problemas que ocasiona la clasificación y búsqueda binaria, es que en memoria RAM puede almacenarse en forma completa sólo archivos pequeños.

Esto puede resolverse con el método que se conoce como ***clasificación por llave***. Este se parece a la clasificación en memoria RAM pero no usa la memoria para almacenar todo el archivo, sino que lee las llaves de los registros y después clasifica las llaves. Es decir, **la clasificación por llave usa la lista de llaves clasificada para reacomodar**

los registros del almacenamiento secundario, de modo que estén en orden. La desventaja de esto es que el reacomodo (reordenamiento) de un archivo con n registros requiere n desplazamientos aleatorios en el archivo original.
No se ordena en archivo en disco ya que sería muy ineficiente.

Diferentes Métodos para clasificar archivos:

1. Ordenamiento del archivo sobre disco

Requiere muchos accesos a disco. Es malo.

2. Clasificación del archivo en RAM

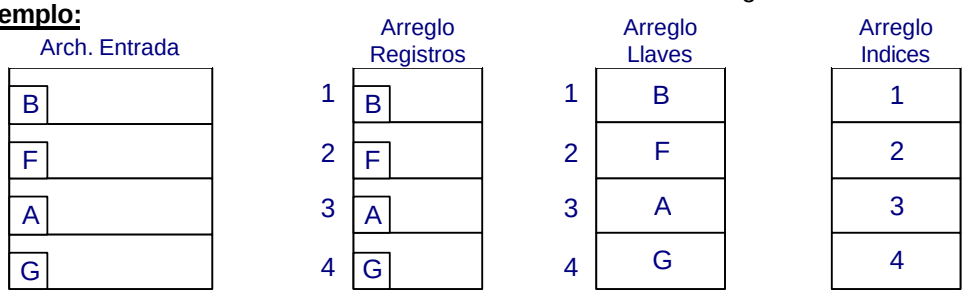
Sólo se puede utilizar si el archivo completo entra en RAM. Se basa en lo siguiente:

- Se leen secuencialmente los registros del archivo y se los almacena en RAM mediante un arreglo de registros (REGISTROS).
- Se obtiene un segundo arreglo LLAVES con sólo las llaves en forma canónica, y un tercer arreglo INDICES con los subíndices del arreglo LLAVES.
- Escribir los registros del arreglo REGISTROS en el orden de clasificación.

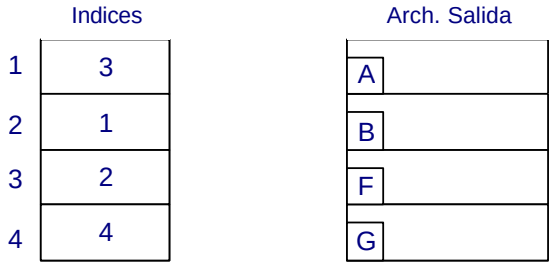
Desventajas:

- ✓ Sólo se puede usar cuando todo el archivo entra en RAM.
- ✓ En cada inserción o modificación de una llave se debe reorganizar el archivo.

Ejemplo:



Luego de clasificar el arreglo indica de acuerdo a las llaves:



3. Clasificación por llaves

Se trasladan a RAM sólo las llaves de los registros del archivo.
El proceso es similar al anterior.

Diferencias:

- ✓ Se lee cada registro en un buffer temporal que luego se tira
- ✓ Cuando se escribe el registro ordenado tiene que leerse por segunda vez, pues no se tiene todos almacenados en RAM.

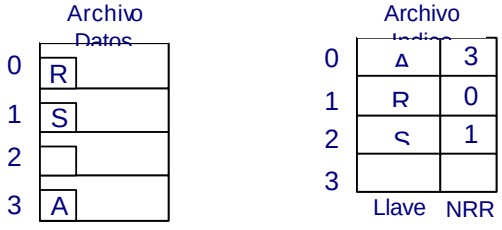
Ventajas:

- ✓ Pueden clasificarse archivos muy grandes, pues las Llaves ocupan menos espacio que los registros.
- ✓ Conduce a la idea de indización.

4. Archivo índice

En lugar de volver a escribir todos los registros clasificados se creará un archivo copia del vector llaves (c/registro tendrá (LLAVE, NRR), en el archivo índice). Para buscar un registro en particular se emplea la búsqueda binaria en el archivo índice.

Ejemplo:



Cuando la clave no cabe en memoria, partimos en n archivos (ordenables en memoria) los ordenamos y juntamos para lograr un archivo ordenado.
Tengo que introducir en memoria tantos registros del archivo como puedo. Logro particiones del tamano maximo como para abarcar toda la memoria disponible (tengo un buffer por particion)
La lectura de cada particion es secuencial y una vez ordenadas a medida que leo los registros puedo armar el nuevo registro ordenado.

5. Procesos secuenciales coordinados

Proceso de dos o más estructuras secuenciales (lista) ordenadas por una clave para producir una única estructura secuencial de salida también ordenada por llave, utilizando intercalación (unión / merge) o correspondencia (intersección)

Ejemplo:

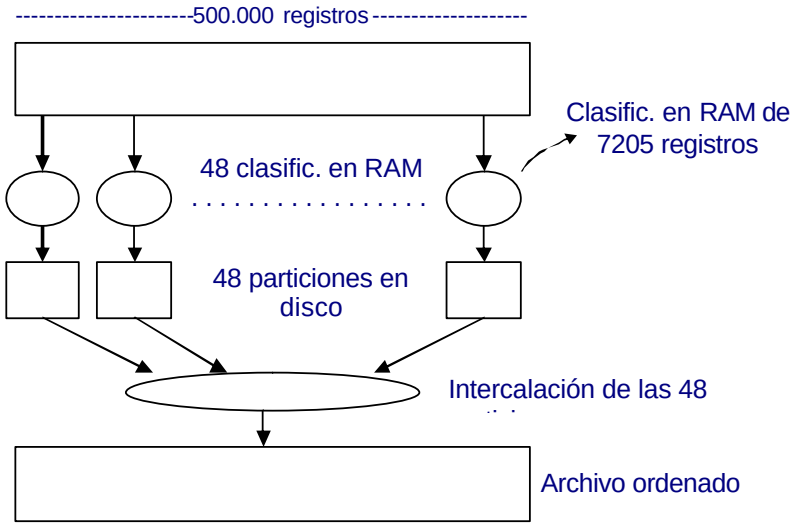
Dos listas: L1 = (1,7,8,9,11)
L1 = (3,5,6,9,10)
➔ L3 = (1,3,5,6,7,8,9,10,11)

6. Método de intercalación

Sirve para clasificar archivos muy grandes que no entran completamente en RAM ni tampoco entran sus claves.
Se transfiere un conjunto de registros, subarchivo = particion, en RAM, se los clasifica y se los escribe a disco como un subarchivo=particion clasificado. Una vez que se tienen todas las particiones en el disco, se puede realizar una intercalación (merge) múltiple para obtener un archivo completamente ordenado.

Ejemplo:

Supongo un archivo de 500.000 registros de 100 bytes cada uno. La Llave primaria es de 5 bytes. Disponemos de una RAM de 1 MB: el archivo completo no entra en RAM, ni tampoco sus llaves → dividimos el archivo en 48 particiones de la siguiente manera:



Ventajas:

- ✓ La lectura de cada partición es secuencial, por lo tanto, la cantidad de desplazamientos es mínima
- ✓ La escritura del nuevo archivo también es secuencial, por lo tanto la cantidad de desplazamientos es mínima

Desventajas:

- ✓ En el momento de hacer el merge, tendremos el doble de registros en el disco ➔ este algoritmo requiere espacio suficiente en disco.

- ✓ Para realizar el merge de n particiones, donde cada partición es del tamaño de la RAM disponible → tamaño del buffer = (1/n) * RAM disponible = (1/n) * Tamaño de cada partición. Es decir, se requieren n accesos a disco para leer todos los registros de una partición. Como hay n particiones → la operación merge requerirá n^2 accesos, lo cual es mucho.

7. Método de selección por reemplazo

Para reducir la cantidad de particiones del método anterior, se puede usar este:
Toma M registros de la particion, se selecciona de RAM la llave con valor mas bajo, se envía a la salida y se la reemplaza luego por una llave nueva de la lista de entrada. La selección de la llave puede realizarse por medio de un árbol de selección. Si la nueva llave menor elegida es menor que la última salidar menor → la nueva llave se duerme cuando están todas dormidas → se comienza con una nueva partición.

Ventajas:

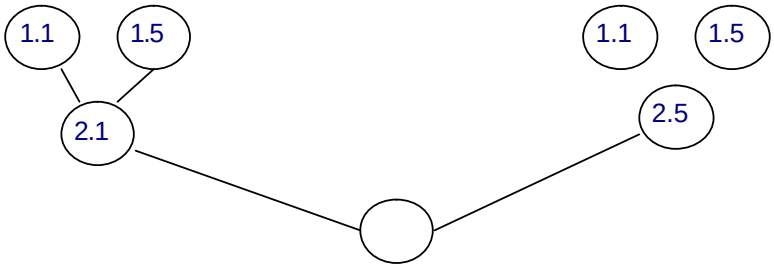
- ✓ Aumenta el tamaño de las particiones (2*M registros, M reg que caben en memoria)→ habrá menos particiones por lo que decrece el trabajo durante la intercalacion, el merge será de orden menor, los buffers serán más grandes y habrá menos accesos a disco.
- ✓ Mejora el trabajo en disco (porque tengo particiones mas grandes) pero empeora la performance en RAM.

8.Selección natural

Se añade un lugar especial en RAM (archivo temporario) para poner los dormidos, registros de reemplazo con clave de salida menor, ya que en la selección por reemplazo están ocupando espacio innecesario en el buffer donde estamos trabajando.
Si un registro dse duerme es porque no hace falta (se escribe en un archivo temporal).
Genero menos particiones mas largas pero tiene mucho costo asociado, tengo el espacio dividido en tres buffers: uno de edntrada, uno de salida (para el archivo ordenado) y uno intermedio.
Ventaja: aprovecha mejor el espacio de memoria, necesita algún buffer mas pero ahorra memoria

9. Intercalacion por pasos

Es una intercalación fuera de RAM, no hay problemas de intercalación pero aumentan los acceso. Solución→ intercalaciones parciales generando archivos temporales. Tomo particiones desordenadas e intercalando en diversos pasos generando una ordenada.



Mejora si hay cierto orden, y mejora la performance si disponemos de varias discos para archivos intermedios o con varias cabezas lectoras grabadoras.

MEJORAS DEL MERGE=INTERCALACION

10. MERGE BALANCEADO DE N CAMINOS

Tengo dos conjuntos uno de entrada y otro de salida. La cantidad de buffers las divide proporcionalmente el SO.
Método:

1) Cada partición (M elementos) se acomoda en un archivo de entrada.
2) Mezcla archivos de entrada produciendo la salida en un archivo de salida (cada partición tendrá M*nro de archivos de entrada registros ordenados)

3) Continuar 2 hasta terminar los archivos de entrada.
4) Convertir el archivo de entrada en archivos de salida y viceversa.
5) Repetir desde 2 hasta formar una partición con todos los elementos ordenados.

11. MERGE OPTIMO

Hace merge entre las particiones de entrada y genera una de salida. Esto se repite hasta generar una particion con todos los datos.
Asigna n-1 buffers para entrada y una para salida.

12. MERGE MULTIFASE

Igual esquema de buffer que optimo pero el de salida cambia.
Mas complejidad para distribuir particiones.
Método:
1. Producir particiones con registros de las particiones de archivo de entrada tomando tantos como los que tengo en el archivo menor.
2. Repetir hasta tener una partición con todos los registros fibonacci, particiones dummy, numero de particiones no se adecua con numero de fibonacci).

Indices

INDICE ➔ es una herramienta para encontrar registros en un archivo. Consiste en un campo de llave mediante el cual se busca el índice y un campo de referencia que indica donde encontrar el registro del archivo de datos asociado con una llave en particular. *Permite imponer un orden en un archivo sin que realmente se reacomode.*
INDICE SELECCTIVO ➔ contiene llaves solo para una porción de los registros del archivo de datos. Dicho índice permite al usuario ver un conjunto especifico de los registros del archivo.
INDICE SIMPLE➔ están construidos con la idea de una secuencia lineal y ordenada de registros índice. Debilidad de los registros índices simples: Agregar registros al índice es costoso.

- **Definiciones ➔** estructuras de datos auxiliares para acceder a la informacion

- Un índice es una herramienta para encontrar registros en un archivo. Consiste de un campo de llave mediante el cual se busca en el índice, y un campo de referencia que indica donde encontrar el registro del archivo de datos asociado con una llave en particular.

Es equivalente a un índice de un libro. La clave esta ordenada y por medio de ella tengo el NRR para acceder el archivo de datos que esta desordenado.

- Tabla que opera con un procedimiento que acepta información acerca de ciertos valores de atributos como entrada (llave) y provee, como salida , una información que permite la rápida localización del o los registros con esos atributos.
  Este índice actúa como un procedimiento. Accedo a la clave y obtengo algo que me permite acceder en forma directa al archivo de datos.
  Manifiesta la posibilidad de manejar índices en memoria: en memoria ordeno y busco rápido, y accedo una sola vez a disco.

- Un índice es una estructura de datos de la forma (Clave, dirección) usada para decrementar el tiempo de acceso a un archivo.
  IndiceArchivo (clave, NRR/distancia en bYtes) ≅ (tema, #hoja)
  Los datos se ponen siempre al final o de acruerdo a la política que se use para las bajas (dejando espacio libre y reaprobechar o no). Se actualiza solamente el índice.
  Encontrar un registro es encontrar la posicion de la clave en el indice.
  El indice tiene tantas entradas como tenga el archivo de datos.

Además:

- Un índice es un recurso para encontrar información que permite imponer un orden en un archivo sin que realmente se reacomode.

- La adición de registros son mucho menores que en un archivo clasificado
- Proporciona varios caminos de acceso a un archivo

CARACTERISTICAS GENERALES
Tengo dos archivos: uno de datos y otro de indices que contiene registros fijos pequeños.
Puedo tener solamente la clave primaria (univoco y una cantidad determinada (limite en el tamaño).
Los indices pueden tener mas informacion (tamaño del registro de lactura).

Técnicas

(1) Archivos secuenciales indizados (son escasos no aparecen todas las entradas del archivo)

Supongamos que se tiene un archivo con registros de longitud variable, donde cada registro es precedido por un campo de tamaño que permita acceso mediante saltos secuenciales.
Para organizar el archivo para que se permita acceso rápido por llave a los registros individuales se utiliza un índice.
La estructura del archivo de índices está compuesta de registros de longitud fija, donde cada registro tiene dos campos de longitud fija: un campo llave y un campo de distancia en bytes.
Para cada registro del archivo de datos hay un registro en el archivo de índices.
El índice está clasificado (ordenado), a diferencia del archivo de datos.

ArchIndices

Llave	Campo de ref
Ang3795	167
Col31809	353
Col38358	211
Dg139201	396
Dg18807	256
Ff245	442

ArchDatos

DirReg	Registro de datos actual			
167	lon	2312	romeo	Pro....
211	col	38358	Nebra...	Sprin...
256	dg	18807	Sinf...	Beeth...
353	col	31809	Sinf...	Dvor...
396	dg	139201	Concierto...	Beeth...
442	Ff	245	Good...	Sweet...

```
Procedure Recupera_Registro(llave)
" Encontrar la posisión de la llave en el índice {usando búsqueda binaria}"
Si (encontré la llave) entonces
        Tomar la distancia en Bytes del registro
        Desplazarse en el archivo de Datos.
        Leer el Registro
    Sino
        Registro inexistente
Fin procediminto
```

Aunque esta estrategia de extracción de información es relativamente directa, tiene algunas características:

- Se trabaja con dos archivos, el archivo de datos y el archivo de índices. Este último es más fácil de manejar porque emplea registros de longitud fija, es mucho más pequeño que el de datos y permite usar búsqueda binaria.
- Al requerir que el archivo de índices tenga registros de longitud fija, se impone un límite para el tamaño de las llaves.
- Los índices pueden tener mas información, como la longitud de cada registro del archivo de datos por lo que se puede producir un desborde.

(2) Archivos no secuenciales indizados: (completo porque cada elemento del archivo aparece en el indice)

Tengo un archivo de datos desordenado y los indices ordenados que contienen los nombres del archivo que permite acceder mas rapido.
Los datos de ponen al final y se hacen los cambios necesarios en el indice.
Tiempo para encontrar la informacion = log2N (N=nro de elementos)

Operaciones basicas en un archivo indizado

Supongamos que el índice es suficientemente pequeño para almacenare en memoria en un arreglo de estructuras llamado INDICE[]. Cada elemento del arreglo tiene la estructura de un registro indice.
El manejo y mantenimiento de un archivo con entradas secuenciales acoplado a un índice simple requiere el desarrollo de procedimientos:

- **Creación de los archivos** ➔Tanto el archivo de índices como el de datos se crean como archivos vacíos, con sólo registros de encabezado.

✯ **Carga del índice en la memoria** ➔ Sólo es cuestión de leer y guardar el registro de encabezado del índice, y después leer los registros del archivo de índice en el arreglo INDICE[]. El procedimiento es una lectura secuencial y como los registros son pequeños debe escribirse de tal forma que se lea un número grande de registros a la vez.

✯ **Reescritura del archivo de índices de la memoria** ➔ Cuando ytermina el procesamiento de un archivo indizado es necesario reescribir INDICE[] en el archivo de índices. Es importante que un programa contenga al menos las dos siguientes defensas contra la posibilidad de que la reescritura del índice se efectúe en forma incompleta:

- ∑ Debe existir un mecanismo que permita al programa saber cuando el índice no está actualizado (con banderas de estado, por Ej).
- ∑ Si un programa detecta que un índice no está actualizado, debe acceder a un procedimiento que reconstruya el índice a partir del archivo de datos. Esto debe suceder automaticamente antes de intentar usarlo.

```
Procedure Reescribir_Indice()
        Revisar la bandera de estado que indica si camnbió el arreglo de memoria.
        Si hubo cambios entonces
                Abrir el archivo como vacío
                        Actualizar registro de cabecera
                Escribir el índice
        Cerrar el archivo de índices
Fin procedure
```

✯ **Adición de registros**

Agregar un registro nuevo al archivo de datos requiere que también se agrege un registro al archivo de índices. Debe conocerse la distancia en bytes de la posición física donde se agregó el nuevo registro y colocarce en el arreglo INDICE[] junto con la forma canonica de la llave del registro, el cuál deberá reacomodarse (este reacomodamiento no requiere ningun acceso al archivo porque el arreglo esta totalmente contenido en memoria).

✯ **Eliminación de registros**

Eliminar un registro nuevo al archivo de datos requiere que también se elimine el registro correspondiente del archivo de índices. Como la operación de borrado en el arreglo (eliminar el registro de indice y desplazar los demas para agrupar el espacio) puede ser no muy costosa, pueden intentarse reacomodos más rápidos usándose apuntadores a estructuras o marcar como eliminado el registro de indice y el registro de datos correspondiente.

✯ **Actualización de registros**

Hay dos categorías:

- ∑ La actualización cambia el valor del campo llave: esto puede provocar un reacomodo del archivo de índices y del de datos. Es como eliminación seguida de adición (eliminar y luego agregar).
- ∑ La actualización no afecta al campo de la llave: no requiere el reacomodo del archivo de índices pero si el de datos. Si el registro no aumenta de tamaño se puede escribir en el mismo lugar que antes, de lo contrario habra que buscar una nueva ubicación y en este caso se tendra que actualizar la direccion de inicio del registro indice correspondiente.

Indices demasiado grandes para almacenarce en memoria

Si el índice es demasiado grande para almacenarse en memoria o trabajo concurrectemente (el indice es un recurso compartido ➔ no puede estar en memoria), entonces el acceso y el mantenimiento del índice deben hacerse en el almacenamiento secundario (buscar un indice== busqueda en almacenamiento secundario)

♦ ***Desventajas***:

- La búsqueda binaria del índice requiere varios desplazamientos en disco y es costoso
- El reacomodo de índices debido a la adición o eliminación de registros requiere mover o clasificar registros en el almacenamiento secundario.

♦ ***Soluciones***:

- Cuando la velocidad de acceso tiene prioridad ➔ una organización por dispersión
- Cuando se necesita flexibilidad en el acceso por llave y en el acceso ordenado y secuencial ➔ uso un índice estructurado en forma de árbol (niveles de indices).

♦ ***Ventajas del uso de índices simples (aunque sea en almacenamiento secundario) con respecto a el uso de archivos clasificados con llave***:

- Un índice simple hace posible la búsqueda binaria para obtener acceso por llave a un registro en un archivo de registros de longitud variable. El índice proporciona el servicio de asociar un registro de longitud fija, donde se puede usar búsqueda binaria, con cada registro de datos de longitud variable.
- Si los registros de índices son considerablemente más pequeños que los registro de datos del archivo, la clasificación y el mantenimiento del índice pueden ser menos costosos de lo que sería la clasificación y el mantenimiento del archivo de datos.
- Si hay registros fijos en el archivo de datos, el uso de un índice permite reacomodar las llaves sin tener que mover los registros de datos.
- Pueden emplearse varios índices para proporcionar diversas formas de ver un archivo de datos.

Indices secundarios

Muchas veces se desean hacer búsquedas por campos que no son claves, para esto se eligen claves secundarias, las cuales no identifica unívocamente a los registros (puede haber claves repetidas).
Los índices secundarios poseen la clave secundaria y la clave primaria asociada a ella, y están ordenados por la clave secundaria + llave primaria (forma canónica)
Un índice secundario es un archivo de registros de longitud fija, formado por un campo llave y un campo de referencia, que puede ser una distancia en bytes del archivo de datos, o una referencia a la llave primaria.

Adición de registros

Agregar un registro al archivo significa agregar un registro al índice secundario. El índice secundario puede contener llaves duplicadas, estas se agrupan y dentro de cada grupo deben estar ordenadas de acuerdo con los valores de los campos de referencia.

Eliminación de registros

Eliminar un registro del archivo de datos significa la eliminación no sólo del registro correspondiente en el índice primario, sino tambien de todos los registros de los índices secundarios que hacen referencia a este registro del índice primario.

- ✓ Si el índice secundario hace referencia directamente al archivo de datos debemos eliminar todas las referencias (indice primario y los secundarios que hacen referencia a ese primario) la eliminación de un registro puede implicar el reacomodo de los registros restantes.
- ✓ Si las referencias son al índice primario, se puede solamente modificar y reacomodar el índice de llaves primarias.

Actualización de registros

- ✓ Si los índices secundarios hacen referencia al archivo de datos, entonces las actualizaciones que cambien la posición física del registro en el archivo, también implican actualizar los índices secundarios.
- ✓ Si la referencia es al índice primario, el índice secundario se ve afectado sólo cuando cambia la llave primaria o secundaria.

Existen tres situaciones posibles:

- La actualización cambia la llave secundaria: habrá que reacomodar el índice secundario de tal forma que permanezca el orden de clasificación. Operación costosa.
- La actualización cambia la llave primaria: sólo se requiere que se actualice el campo de referencia afectado en todos los índices secundarios (buscar los indices secundarios que no se cambiaron y reescribir los campos afectados). Si una llave secundaria aparece mas de una vez se debe hacer una reclasificacion local.
- La actualizacion se restringe a otros campos:las actualizaciones que no afectan los campos de la llave primaria o secundaria no afectan el indice de llaves secundarias. Si hay varios indices de llaves secundarias asociados a un archivo, la actualizacion de registros afecta solo a un subconjunto de indices secundarios.

Una variante para los índices secundarios es que estos apunten a posiciones relativas dentro del archivo (NRR o una forma de ubicar el registro), esto hace mas ineficiente las operaciones de eliminación y actualización. En la actualización si el registro de datos cambia de posición dentro del archivo (cambia de tamaño), habrá que actualizar todos los índices secundario, y en la eliminación, no se podrá usar la técnica que se usaba antes (ya que no se puede distinguir si se apuntan a datos válidos o no), por lo tanto habrá que borrarlos de todos los índices secundarios

Consulta

Como se pueden tener varios archivos de índice secundario, estos se puede combinar para realizar consultas más complejas. Para lograr esto supongamos que tenemos dos índices y queremos obtener los datos que cumplen con clavesec1 and clavesec2, para lograr esto, se seleccionan del 1er índice los que cumplen con clavesec1 y del 2do índice los que cumplen con clavesec2, luego se hace una intersección de las claves primarias resultantes. Estos datos permitirán obtener la información que deseamos

Mejora de la estructura secundaria de indices

Las estructuras secundarias de índices ocasionan dos problemas:

- El archivo de índices tiene que reacomodarse cada vez que se agrega un registro nuevo al archivo, aún cuando éste sea de una clave secundaria que ya existe.
- Si hay claves secundarias duplicadas, el campo de clave secundaria se repite para cada entrada. Esto desperdicia espacio.

⇨ Posibles soluciones:

♦ Una solución simple es cambiar la estructura del índice secundario, de tal forma que se asocie un arreglo de referencia con cada llave secundaria. No se tiene la necesidad de reacomodar el índice cada vez que se agrega un nuevo registro al archivo de datos. Esto tiene como desventaja que con el arreglo es de tamaño fijo se limita el número de referencias asociadas a cada llave secundaria.

♦ Mejor solución: listas invertidas ➔ A los archivos que son como los índices secundarios, en los que una llave secundaria lleva a un conjunto de una o más llaves primarias, se los llama listas invertidas. Se trata de una lista de referencias de llaves primarias.
Podría redefinirse el índice secundario de modo que conste de registros con dos campos: un campo de llave secundaria y un campo que contenga el NRR de la primer referencia a la llave primaria correspondiente en la lista invertida. Las referencias reales de las llaves primarias asociadas con cada llave secundaria podrían almacenarse en un archivo secuencial separado.
Indice de llaves secundarias que hacen referencia a listas ligadas de referencia a llaves primarias.

Archivo de índice secundario

0	A	3
1	B	2
2	C	7
3	D	10
4	E	6
5	F	4
6	G	9

Archivo de listas de ID

0		-1
1	A4	-1
2		-1
3	A1	8
4		-1
5	A3	1
6		-1
7		-1
8	A2	5
9		-1
10		0

Siguiendo los id hasta encontrar la marca -1, tenemos la lista: A2, A3, A1, A4.

Esta mejora proporciona algunas ventajas:

✖ El único momento en que se requiere reacomodar el archivo de índices secundarios es cuando se añade una nueva llave secundaria.

✖ En caso de que sea necesario reacomodar el archivo de índices secundarios, la tarea es más rápida, ya que existan menos registros y cada registro es mas pequeño.

✖ Puesto que hay menos necesidad de clasificaciones, es menor el precio que hay que pagar por mantener los archivos de índices secundarios en el almacenamiento secundario, por lo que queda más lugar en memoria RAM.

✖ El archivo de listas es de entradas secuenciales, por lo que nunca necesitará clasificarse

✖ Como el archivo de listas es de registros de longitud fija, es muy fácil desarrollar un mecanismo para reutilizar el espacio de los registros eliminados.

Indices selectivos

Un **índice selectivo** contiene llaves sólo para una porción de los registros del archivo de datos. Dicho índice permite al usuario ver un subconjunto específico de los registros del archivo.
Se aplica a archivos serie.
Son utiles cuando el contenido del archivo de datos entran varias categorias generales. Soluciona problemas de vistas (acceder solo a una porcion de elementos) genero una vista ➔ genero un indice selectivo para esto.

Enlace (Binding)

El **enlace** se realiza cuando una llave se asocia con un determinado registro físico del archivo de datos; en general, puede tener lugar ya sea durante la preparación del archivo de datos e índices o durante la ejecución del programa. En un ***enlace fuertemente acoplado***, los índices contienen referencias explícitas a los registros físicos de datos asociados, las reorganizaciones del archivo de datos pueden provocar modificaciones en todos los archivos de indices enlazados. En el último caso, la conexión entre una llave y un registro físico en particular se posterga hasta que realmente se extrae el registro, durante la ejecución del programa.

✖ El enlace fuerte es bueno cuando:

- El archivo de datos es estático o casi estático, para que requiera poca o nula adicion, elimincion y actualizacion de registros.
- El desempeño rápido durante la extracción tiene prioridad.

✖ El enlace en el momento de la extracción de la información es bueno para las aplicaciones en las que hay adición, eliminación y actualización de registros.

✖ Postergar el enlace tanto como sea posible facilita y asegura las operaciones

Operaciones secuenciales coordinadas

- Procesamiento coordinado de dos o mas listas secuenciales para producir una única lista de salida.
- Correspondencia de nombres en dos listas (intersección de dos listas)
- Intercalación de listas (merge)
- Intercalación múltiple (K-formas): Se cuenta con k listas ordenadas de entrada y se desea tener una única lista de salida. Funciona bien con 8 listas o menos. Sino árbol de selección

Intercalación para clasificar archivos grandes

Se desea ordenar archivos que por ser demasiado grandes no entran en RAM.

- La E/S es básicamente secuencial

Kformas, cada porción es del tamaño del área de trabajo disponible en RAM, el tamaño del buffer para cada porción es:

(1/K)*tamaño del espacio de RAM = (1/K)*tamaño de cada porción

Este tipo de intercalación tiene un alto costo en desplazamientos

Para reducir el número de desplazamientos en intercalación:

- intercalación en más de un paso
- incrementar el tamaño de las porciones clasificadas

Patrones de intercalación de varios pasos

Se intercala en etapas; seleccionando conjuntos reducidos e intercalándolos entre si. Hay porciones intermedias. Los datos son leídos más de una vez. *(Ver la forma de calcular la cantidad de desplazamientos)*

Incremento del tamaño de partición.

Si se aumenta de alguna manera el tamaño de las porciones iniciales, hay una ganancia, ya que el merge se hace sobre menos particiones por lo tanto cuando se lee del disco se puede llevar mas información a la memoria de una. Para esto se usa el algoritmo de selección por reemplazo.

Se selecciona de la memoria la llave con valor mas bajo, se envía a la salida y se lee una llave nueva. Puede usarse un árbol de selección. Si la nueva llave tiene un valor inferior a la ultima que se envío a la salida, se pone en la heap 'lockeado' y se sigue hasta que la heap no tiene lugar (todos trabados) y se inicia una nueva partición.

Longitud promedio de las porciones para selección por reemplazo

Es 2*cantidad de claves en memoria. Generalmente es mucho mayor, porque porciones parcialmente ordenadas mejora mucho esta técnica.

Indices en almacenamiento secundario

El problema de mantener un índice en almacenamiento secundario es que éste es muy lento. Según hemos visto hasta ahora, éste problema se puede dividir en dos más específicos:

- La búsqueda binaria requiere muchos desplazamientos. Si el índice esta en disco, el ir y venir de la búsqueda binaria se hace muy costoso.

Mantener un índice ordenado para hacer la búsqueda es carísimo: Insertar una clave puede requerir reordenar todo el archivo, lo que es inaceptable.

Unidad 13. ARBOLES

El problema de mantener un índice en almacenamiento secundario ordenado es que éste es muy lento. Podemos armar arboles de indices ➔estructuras de datos para encontrar mas rapido la informacion de un archivo.

Arboles Binarios

Implementar árboles binarios, donde cada nodo es un registro fijo (llave, Hijo Izq, Hijo Der) de archivo es una solución.

Tengo que tener informacion sel primer elemento del arbol (archivo) para poder recorrerlo.

Escribo siempreal final del archivo y actualizo los nros relativos de registro de su padre.

El dato puede ser:

- clave + hd+hi
- registro completo + hd + hi.

La ventaja de esta implementación es que no se requiere reordenar todo el archivo al insertar una clave nueva, el mantenimiento es muy sencillo.

Las desventajas son: Si el árbol no esta balanceado, el peor caso puede transformarse en una búsqueda secuencial.

BALANCEADOS➔ La altura de la trayectoria mínima hacia una hoja no difiere en más de uno a la altura máxima(trayectoria máxima a una hoja). Además, si se requiere un acceso por cada lectura de nodo, entonces son demasiados accesos (peor caso $\log_2(N+1)$).

Inconvenientes de los arboles binarios ➔ se desbalancean facilmente.

Arboles AVL

Para solucionar el problema de que los árboles binarios pueden convertirse en listas (árboles degenerados!!) si las claves se insertan de una manera indeseable, se introducen los arboles AVL.

Los arboles AVL tienen la particularidad de estar balanceados en altura (para cada nodo existe un limite en la diferencia que se permite entre las alturas de cualquiera de los subarboles del nodo). Un árbol es k-balanceado en altura(BA(k)) si la altura de sus hijos no difiere en más de k y ellos son BA(k). Los AVL son BA(1) donde las inserciones y eliminaciones se efectuan con un minimo de accesos

Mantener un AVL es un poco más complicado que un binario, pero no requiere mas de elegir una de cuatro rotaciones restringidas a un área local, no afecta todo el arbol, hay un reacomodamiento de todo el arbol (de no más de 5 reasignaciones de punteros c/u). Tienen una estructura que hay que respetar.

En un árbol completamente balanceado, el peor caso en una búsqueda es: $\log_2(N+1)$.

En un AVL es 1.44 $\log_2(N+2)$.

Hasta ahora tenemos resuelto, al menos parcialmente, el problema de mantener el índice ordenado, y garantizar un poco de balanceo. Sin embargo, la cantidad de accesos (peor caso $1.44\log_2(N+2)$) sigue siendo mala.

Arboles Binarios Paginados

Ahora tenemos que solucionar el problema de la cantidad de accesos. Para ello se crearon los árboles binarios paginados. Estos árboles consisten en un árbol binario común, el cual se ha dividido en paginas. Cada pagina contiene un árbol cuyas hojas pueden apuntar a otras paginas. (El archivo queda organizado como un árbol de páginas). Por ejemplo, si cada pagina almacena 7 nodos, esta pagina puede ramificarse en 8 paginas más.

Estos árboles son los más rápidos que se han analizado hasta este momento. (Peor caso de accesos en una búsqueda es de $\log_{k+1}(N+1)$, donde k es el nro de nodos por pagina). [NT: Estos árboles no son más que árboles n-arios de búsqueda donde cada nodo tiene k elementos.]. el numero de accesos se mantienen chico con grandes cantidades de registros. Manejando buffers el numero de acceso baja logaritmicamente.

Problema al construirlos ascendentemente: El problema que tienen estos árboles es que su construcción ascendente produce árboles desbalanceados muy fácilmente. Es muy complicado hacer un algoritmo que reacomode, para cada inserción, todo el árbol completo. Solución: Construcción ascendente.

A partir de un acceso puedo acceder a distintos subarboles.

Grafico hoja 333

Arboles multicamino

Son una generalizacion de arboles binarios, cada nodo tiene K punteros y K-1 claves (o registros) = orden del arbol, disminuye la profundidad del arbol. La cantidad de punteros es igual al orden del arbol.

Dibujo pag 339

Disminuye la profundidad del arbol.
Tiene orden = la cantidad máxima de hijos que voy a tener (binario → orden 2).

Arboles B

Son arboles multicamino con una construccion especial en forma ascendente que permite mantenerlo balanceado a bajo costo.

Estos árboles B se construyen ascendentemente. En un árbol B, cada nodo contiene una secuencia ordenada de llaves e hijos entre cada llave. [NT: Es como un árbol n-ario de búsqueda, donde cada nodo tiene una cantidad variable de hijos].
Los NRR (o la info asociada con cada llave) van acompañando la llave en cada nodo. Podemos definir formalmente un árbol B de orden m, de la siguiente manera:

- Cada nodo tiene máximo m hijos.
- Cada nodo, excepto raíz y hojas, tiene por lo menos $\lceil m/2 \rceil$ hijos
- La raíz tiene mínimo dos hijos o ninguno.
- Todas las hojas están en el mismo nivel. Arbol de altura constante. (busqueda = (altura+1)/2)
- Un nodo, no hoja, con k hijos tiene k-1 claves (registros)= orden dl arbol.
- Un nodo hoja tiene un minimo de $\lceil m/2 \rceil$-1 claves(registros) y no más de m-1 registros.

Los árboles B se construyen hacia arriba a partir del nivel hoja, de modo que la creación de las páginas nuevas comienzan en el nivel de las hojas.
El poder de los árboles B reside en que están balanceados (no hay ramas demasiado largas); tiene poca profundidad (requieren pocos desplazamientos); permiten eliminaciones e inserciones aleatorias a un costo relativamente bajo mientras se mantiene el balance, y garantizan, al menos, un 50 por ciento de utilización de almacenamiento.

Operaciones sobre árboles B:

Estructura de la pagina:
Cada llave es un solo carácter , MAXLLAves es la máxima cantidad de llaves permitidas y MAXHIJOS la máxima cantidad de hijos permitidas. Cont llaves sirve para determinar si la pagina esta llena o no.

```
TYPE
PAGINAAB= RECORD
        Contllaves : integer;
        Llaves : ARRAY [1..MAXLLAVES] of llave;
        Hijo : array [1..MAXHIJOS] of NRRde pag
        End;
VAR PAGINA: PAGINAAB;
```

Busqueda de elementos:
El procedimiento de busqueda se llama a si mismo recursivamente; localiza una pagina y despues busca en ella para encontrar la lleve en niveles sucesivamente mas bajos del arbol hasta que la encuentra o no puede descender mas.

```
FUNCTION busca (NRR, LLAVE, NRR_ENCONTRADO, POS_ENCONTRADA
Si NRR = NULO entonces    {condicion de detccion de la recursion}
        Devuelve no encontrada
Otro
        Lee la pagina NRR en PAGINA
        Busca LLAVE en PAGINA, haciendo POS igal a la posicion donde LLAVE este o deberia estar.
Si se encontro la llave entonces
        NRR_ENCONTRADO := NRR {el NRR actual contiene la lave}
        POS_ENCONTRADA := POS
        Devuelve ENCONTRADA
        Otro {sigue la referencia HIJO al siguiente nivel inferior }
                Devuelve  (busca( PAGINA, HIJO[POS], LLAVE, NRR_ENCONTRADO, POS_ENCONTRADA))
        Fin si
Fin si
Fin FUNCION;
```

Inserción, division y promocion de un elemento:

- Se busca la hoja en donde hay que insertar la clave. Busqueda que llega hasta el nivel de hoja.
- Despues de encontrar el lugar de insercion en la hoja el trabajo de insersion, division y promosion continua en forma ascendente desde abajo:
  - Si hay lugar se pone ahí y listo.
  - Cuando tenemos que insertar una clave y la hoja esta llena→ se divide en dos la hoja y se distribuyen las llaves tan equitativamente como sea posible, promobiendo el medio como padre y se inserta en la raiz

(cuando este nodo se llena tambien debe dividirse, repartir las llaves y con la del medio creo una nueva raiz).

Nota: Si se redistribuye antes de partir, se retrasa la partición y el árbol queda más bajo más tiempo.

En el procedimiento recursivo tenemos tres fases:

1. busqueda en paginas que ocurre antes de la llamada recursiva.
2. La llamada recursiva mueve la operación hacia abajo en el arbol conforme busca la llave o el lugar para insertarla.
3. En la trayectoria de vuelta luego del descenso recursivo lego de la llamada recursiva se ejecuta la insercion, division y promocion.

La funcion **inserta()** realiza la division y promocion. Usa ,los argumentos NRR_ACTUAL (NRR de la pagina que se usa actualmente), LLAVE (llave a insertar), LLAVE_PROMO(para el valos devuelto - llave promovida-), HIJO_D_PROMO (valor devuelto-apuntador al hijo derecho de la llave promovida), tambien devuelve el valor PROMOCION si hay una promocion y NOPROMOCION si se inserta y no se promueve nada y ERROR sino se puede hacer la insercion.

La funcion inserta usa la funcion **divide()** que crea una pagina nueva distribuye las llaves entre la original y la nueva y determina la llave y el NRR que hay que promover (solo se promueve la llave a partir de la pagina de trabajo: todos los HIJO_D_NRR son transferodos de regreso a PAGINA y PAGINANUEVA, el NRR que se promueve es el NRR de PAGINANUEVA porque PAGINANUEVA es el descendiente derecho de la llave que se promueve.

```
Function inserta (NRR_ACTUAL, LLAVE, HIJO_D_PROMO, LLAVE_PROMO)
    Si NRR_ACTUAL = NULO entonces
        LLAVE_PROMO :=LLAVE
        HIJO_D_PROMO:=NULO
        Devuelve PROMOCION  {promueve la llave original y NULO}
    Otro
        Leer la pagina de NRR_ACTUAL en PAGINA
        Buscar la LLAVE en PAGINA
        POS:=la posicion en donde esta deberia estar LLAVE
    Si se encuentra LLAVE entonces
        Emitir mensaje de error indicando llave duplicada
        Devuelve ERROR
    VALOR_DEVUELTO:=inserta )PAGINA.HIJO[POS], LLAVE, NRR_P_A, LLAVE_P_A)
    Si VALOR_DEVUELTO ==NOPROMOCION o ERROR entonces
        Devuelve VALOR_DEVUELTO
    Otro si hay espacio en PAGINA para LLAVE_P_A entonces
        Insertar LLAVE_P_A y NRR_P_A (promovida desde abajo, en PAGINA devuelve NO PROMOCION)
    Otro
        Divide (LLAVE_P_A, NRR_P_A, PAGINA, LLAVE_PROMO, HIJO_D_PROMO, PAGINA_NUEVA)
        Escribe PAGINA al archivo en el NRR_ACTUAL
        Escribe PAGINA_NUEVA al archivo en el nrr HIJO_D_PROMO
        Devuelve PROMOCION {promocion de LLAVE_PROMO e HIJO_D_PROMO)
    Fin si
Fin funcion
PROCEDIMIENTO divide (LLAVE_I, NRR_I, PAGINA, LLAVE_PROMO, HIJO_D_PROMO, PAGINANUEVA)
    Copiar las llaves y apuntadores de pagina en una pagina de trabajo que puede almacenar una llave adicional y un hijo.
    Insertar LLAVE_I y NRR_I en sus lugares correctos en la pagina de trabajo.
    Asignar e iniciar una pagina nueva en el archivo del arbol V para que contenga a PAGINANUEVA
    Asignar a LAVEPROMO el valor de la llave de l medio que sera promovida luego de la division.
    Asignar a HIJO_D_PROMO el NRR de PAGINANUEVA
Copiar las llaves y apuntadores a los hijos que preceden a LLAVE_PROMO de la pagina de trabajo a PAGINA
    Copiar las llaves y apuntadores a los hijos que siguen a LLAVE_PROMO de la pagina de trabajo a PAGINANUEVA.
Fin procedimiento
```

Se necesita una rutina para unir los procedimientos inserta() y divide(), que debe ser capaz de:

- Abrir o crear el archivo del arbol B e identificar o crear la pagina raiz
- Leer las llaves que van a almacenarse en el arbol B y llamar a inserta para colocarlas().
- Crear un nuevo nodo raiz cuando inserta() divide la pagina que en ese momento es la pagina raiz.

PROCEDIMIENTO manejador

```
    Si el archivo del arbol B existe, entonces
        Abrir el archivo de arbol B y colocar la primera llave en la raiz
    Tomar el NRR de la pagina raiz del archivo y almacenarla en raiz
```

```
        Tomar una llave y almacenarla en LLAVE
        Mientras existan llaves
                Si (inserta (RAIZ, LLAVE, HIJO_D_PROMO, LLAVE_PROMO==PROMOCION)
                Entonces
                Crear una nueva pagina raiz con llave := HIJO_D_PROMO
                        hijo izquierdo := RAIZ e hijo derecho :=HIJO_D_PROMO
                asignar RAIZ el NRR de la nueva pagina raiz
                tomar la siguiente llave y almacenarla en llave
        finn mientras
        escribe el NRR almacenado en RAIZ de regreso al archivo del arbol B
        cerrar el archivo del arbol
fin procedimiento manejador.
```

Eliminación de un elemento:

La posibilidad de que los arboles B sean amplios y bajos esta ligada a las siguientes reglas que establecen que:

- Cada pagina, excepto la raiz y las hojas, tiene por lo menos [m/2] descendientes
- Una pagina que no es hoja con k descendientes contiene k-1 llaves y
- Una pagina hoja contiene por lo menos [m/2]-1 y no mas de m-1 llaves.

Es necsario desarrollar algun tipo de garantia igualmente confiable para que se mantengan estas propiedades cuando se eliminan llaves del arbol.

Siempre se debe eliminar de paginas hoja, por lo que si una llave no esta en una hoja, la ponemos en una y eliminamos de la hoja. Si la pagina con la que intercambiamos se excede en el numero de llaves debemos redistribuir entre las paginas y encontrar una llave para poner en la pagina padre que actue como separador entre las paginas de nivel inferior.

Si no hay llaves suficientes para repartir tenemos que concatenar combinando dos paginas y la llave de la pagina padre para hacer una sola completa. Esta accion debe inplicar el descenso de llaves y esto puede causar ineficiencia en el padre.

Pasos para eliminar una llave:

- Se busca el nodo donde esta el elemento a borrar.
- Si el elemento no esta en una hoja, se intercambia con su sucesor inmediato en el árbol (que seguro esta en una hoja)
- Se borra el elemento de la hoja.
- Si el nodo queda insuficiente, se miran los hermanos.
- Si alguno le puede prestar, hacen redistribución (tipo trencito incluyendo a la clave del padre).
- Si no le pueden prestar, elige a uno y se fusionan en una sola hoja (hay que concatenar). Esta hoja tiene las claves de las dos hojas que se juntan más la clave padre. Dicha clave es eliminada del padre pudiendo pasar lo mismo hasta la raíz (si se elimina la ultima llave de la raiz la altura del arbol decrece).

Algoritmo iterativo

```
{fin indica fin del borrado
N_G : nodo del tamaño mayor al normal
ADY: nodo adyacente hermano
REG: registro a borrar}
{buscar reg en el arbol}
if REG no esta en una hoja
        then buscar el sucesor de REG en una hoja
             intercalar REG con su sucesor
{eliminacion de REG}
FIN:=falso
Repeat
        Remover REG y el puntero asociado
```

```
If nodo_corriente > [M/2]-1 elementos
        Then FIN:=TRUE
        Else if s posible redistribuir
                Then {redistribuir} {ADY > MIN}
                        Coiar adyacente y nodo actual en N_G
                        Dividir N_G
                        Promover nueva llave al padre
                        Armar el nodo actual y adyacente con la mitad de N_G cada uno
                        FIN:= true
                Else {concatenar}
                        Elegir el mejor adyacente para concatenar
                        Poner adyacente y nodo actual juntos y agregarle el elemento del padre
                        Eliminar nodo que sobra y producir enganches
                        Nuevo REG elemento que baja nodo padre y debemos eliminar
Until FIN
If no hay registro en la raiz
        Then nueva raiz es el nodo actual
                Liberar vieja raiz
```

REDISTRIBUCION
Es la division inversa, con la diferencia que no se propaga tiene efectos locales (no implica creacion o eliminacion de nodos).
La redistribucion durante la insercion es una forma de posponer la creacion de paginas nuevas. Cuando un nodo cae en insuficiencia (menos del 50% lleno) podemos trasladar llaves desde un hermano para mantener la propiedad de 50% lleno. Cunado distribuimos las llaves tambien tenemos que alterar el contenido del ppadre.
En lugar de dividir una pagina completa y crear dos medio llenas, la redistribucion permite reacomodar dentro de otra pagina algunas de las llaves que provocan saturacion. Usando redistribucion como una alternativa de la division en el caso que sea posible y dividir solo cuando ambos hermanos esten completos, la utilizacion del espacio s incremente

Arboles B*

Son una variación de los B. Estos árboles al menos 2/3 llenos (en lugar de ½ como estaban los B) . La modificación principal para lograr esto, está en la división. Antes de dividir, primero se fija si puede distribuir con algún hermano (cualquier pagina que no sea la raiz tendra al menos un hermano completo con la cual redistribuir, si no hay hermanos no se puede dividir de dos a tres por lo que se permite que la raiz crezca un tamamo mayor hasta que pueda producir dos paginas dos tercios llenas). Si no puede entonces hay dos nodos llenos antes de partir,que se parten en tres nodos, quedando c/u 2/3 llenos. La modificación de los algoritmos, queda a cargo del lector. Formalmente es de orden m cuando:

- Cada pag tiene máx m hijos.
- Cada pagina, salvo raíz y hojas, tiene al menos (2m-1) /3 hijos
- La raíz tiene al menos dos hijos.
- Todas las hojas aparecen al mismo nivel.
- Pagina que no sea hoja con k hijos tiene k-1 claves.
- Una pagina tiene por lo menos $\lfloor (2m-1)/3 \rfloor$ llaves y no mas de. m-1

OPERACIONES:
BUAQUEDA➔ igual a B
INSERCION ➔ tres casos:

- derecha : redistribuir con un nodo adyacente hermano derecho o izquierdo si es el ultimo.
- Izquierdo o derecha : si el nodo tiene dos adyacentes y el derecho esta lleno entonces redistribuyo con el izquierdo.
- Izquierdo y derecho : busca llenar los tres nodos (cuando uno se llena lo junto con el de la izquierda, el de arriba y el de la derecha y rdistribuyo entre los tres, cada uno 3/4 llenos (la altura no aumenta y se mejora la busqueda).

ELIMINACION ➔ similar (concatenacion , redistribucion).

Arboles B Virtuales

Indice de un arbol donde se guardan varias paginas en memoria RAM, para un mejor acceso a una o varias paginasSon árboles B con un caché de páginas (nodos). Entre las técnicas de reemplazo de paginas están:

- LRU: Se reemplaza la que menos recientemente se usó.

- Ponderacion de altura: Se mantienen si o si los primeros niveles del árbol, y si entran más se usa por ej: LRU.

Archivo Secuencial Indizado:

Son archivos que se pueden recorrer secuencialmente (acceder por orden fisico) o acceder con un indice (ordenado por llave).

CONJUNTO DE SECUENCIAS ➔ conjunto de registros que mantiene un orden fisico por llave mientras se agregan o quitan datos, si podemos mantenerlos podemos indizarlos. Es imposible reclasificarlos cuando se eliminan registros.

BLOQUES ➔ unidad basica de entrada y salida. Conjunto de registros. Logro restringir los efectos de una insercion o eliminacion solo a una parte del conjunyto de secuencias.
Esta relacionado con el tamaño del buffer (debe permitir almacenar un bloque entero= despues de leer un bloque todos los registros deben estar en memoria RAM donde se puede manejar y reacomodar mas rapidamente).

OPERACIONES
La insercion de nuevos registros puede provocar saturacion por lo que tenemos que dividir los registros entre dos bloques y se racomodan los enlaces para que aun sea posible moverse a lo largo del archivo en el orden de las llaves bloque tras bloque.
La eliminacion de registros puede provocar insuficiencia. Tenemos dos soluciones: concatenar (movemos los registros entre bloques logicamente adyacentes, liberando uno para su posterior uso) o redistribuir (entre bloques adyacentes).

COSTOS
- Con las inserciones el archivo aumenta de tamaño por la fragmentacion interna.
- Se mantiene el orden fisico solo en el bloque.

TAMAÑO DEL BLOQUE. CONSIDERACIONES:
- Debe ser suficiente como para almacenar varios bloques en memoria RAM a la vez.
- Las lecturas y escrituras de un bloque deben ser rapidas.
- Se debe permitir acceder a un bloque sin necesidad de pagar un costo de desplazamiento en el disco en una lectura o escritura.

CUMULO = CLUSTER ➔ minimo numero de sectores asignados a la vez. Un cluster garantiza una cantidad minima de secuencialidad fisica. Dentro de un cluster accedo a los datos sin movimiento de disco.

INDICES ➔ cada bloque contiene un grupo de registros (intervalo) que es lo bastante informativo para ayudar como para elegir el bloque que quiza tiene el registro. Podemos tener un indice de registros de longitud fija que contiene la clave del ultimo registro de cada bloque. Esto proporciona un acceso secuencial indizado completo (consultamos el indice, extraemos el bloque).
El indice debe guiar hacia el bloque del conjunto de secuencias que contiene el registro, entonces el contenido del indice ayuda a obtener el bloque correcto en el conjunto de secuencias, solo contiene informacion acerca de a donde ir para obtener respuestas.

Arboles B+

DEFINICION ➔ Consiste en un conjunto de secuencias de registros ordenados por llave en forma secuencial, junto cin un conjunto indice que proporciona acceso indizado a los registros. Toma caracteristicas de los arbol B y de los archivos ordenados fisicamente.
Todos los registros se almacenan en un conjunto de secuencias.
Se usan cuendo necesito aplicar orden por algun criterio.
Los datos estan en los nodos terminales enganchados entre si, y arriba tengo elementos que me permiten el acceso rapido.
Cada puntero entre dos separadores A1 y A2 apunta a un bloque donde todas las claves de los regs de ese bloque están entre A1 y A2.

PROPIEDADES (orden M):
1. Cada pagina tiene maximo M descendientes.
2. Cada pagina, menos la raiz y las hojas, tienen entre [M/2} y M hijos
3. La raiz tiene al menos dos descendientes (o nnguno)
4. Todos las hojas aparecen en igual nivel
5. Una pagina que no sea hoja sitiene K descedientes contiene K-1 llaves

6. Los nodos terminales rapresentan un conjunto de datos y son linkeados juntos. Los nodos no terminales tiene punteros a datos.

OPERACIONES

- **Busqueda** ➔ en la parte superior tengo los elementos necesarios como para llegar al ultimo nivel donde tengo la informacion (encuentro un puntero a partir del cual llegare a la informacion)

- **Inserciones y eliminaciones de registros**➔ (similares a arbol B) se manejan por medio de division y concatenacion y redistribucion del bloque en el conjunto de secuencias. El conjunto indice que se usa solo como una ayuda para encontrar los bloques en el conjunto de secuencias se maneja como un arbol B.

COMPAR ACIONES ARBOL B Y ARBOL B+

	Arbol B	Arbol B+
Ubicación de datos	En cualquier nodo	En los nodos terminales
Tiempo de busqueda	=	=
Procesamiento secuencial	Lento, complejo	Rapido, uso punteros
Insercion / eliminacion		Puede requerir mas tiempo

Arboles B+ de prefijos simples

Son arboles B+ donde el conjunto indice esta constituido por separadores mas cortos (prefijos simples).
Separador mas corto: separadores que ocupan el espacio minimo para una estrategia de compresion en particular (sigue sirviendo como separador a pesar de modificaciones en los datos (eliminacion, insercion, etc).
Son árboles B+ donde un separador de dos bloques A y B, es el prefijo de la menor clave de B que los separa. Son útiles porque el tamaño de los separadores se hace chiquito. Naturalmente se utilizan claves de tamaño variable en los nodos del árbol, quedando una cantidad de claves variables en cada nodo.

OPERACIONES

La INSERCION y ELIMINACION de registros tiene lugar en el conjunto de secuencias (lugar donde estan los registros) y los cambios en el conjunto de indices son secundarios, son una consecuencia de las operaciones fundamentales sobre el conjunto de secuencias. Si es necesario division, concatenacion o redistribucion se realizan como si el conjunto de secuencias no existiera➔ luego de hacer los cambios en los registros del conjunto de secuencias se hacen los ambios en el conjunto indice:

- Si los bloques se dividen en el conj de secuencias hay que insertar un nuevo separador dentro del conj indice
- Si se concatenan bloques , debemos eliminar un separador del indice
- Si los registros se redistribuyen en el conj de secuencias se debe cambiar un valor en el conjunto de indices.

TAMAÑO DEL BLOQUE DEL CONJ DE INDICES

- Es recomendable que tenga el mismo tamaño que el de secuencias.
- Con un tamaño de bloque comun se facilita la implementacion de un esquema de manejo de buffers para crear un arbol B+ de prefijos simples virtual.
- Los bloques del conjunto indice y de secuencia se ponen dentro de un mismo archivo para evitar desplazamientos entre archivos. Esto es mas sencillo si los tamaños de los bloques son iguales.

ESRUCTURA INTERNA DEL BLOQUE DEL CONJUNTO DE INDICES

Cont de separad.	Long. Total de los separad.	Separadores	Indices de los separadores	Nro. Relativo de bloque

- ♦ Contador de separadores: numero de separadores, sirve para ayudar a encontrar el elemento del medio en el indice de los separadores, para que pueda enpezarse una busqueda binaria.
- ♦ Longitud total de los separadores: se necesita conocer el tamaño de la lista de separadores para encontrar el inicio del indice.
- ♦ Indice de los separadores: posicion donde comienza cada separador.
- ♦ Numero relativo de bloque: tanto de indice como de secuencia.

Descripción general sobre árboles

Los índices simples y lineales funcionan bien si se almacenan en memoria RAM electrónica, pero es caro mantenerlos y buscarlos cuando son tan grandes que deben guardarse en almacenamiento secundario. Lo costoso del uso del almacenamiento secundario es más evidente en dos áreas:

- Clasificación de índice, y
- Búsqueda, porque aún la búsqueda binaria requiere de dos o tres accesos al disco.

Para poder estructurar un índice de tal modo que pueda mantenerse en orden sin tener que clasificar, se utilizan estructuras de árboles y es necesario tener un árbol balanceado para asegurarse de que el árbol no se vuelva demasiado profundo luego de repetidas inserciones aleatorias. Los **árboles AVL** proporcionan una forma de balancear un árbol binario con sólo una pequeña cantidad de gastos adicional.

Pero la necesidad de reducir el número de accesos a disco requeridos para buscar en un árbol, conduce a la idea de dividir el árbol en *páginas*, de tal forma que una parte considerable del árbol pueda extraerse con sólo un acceso al disco. Los *índices paginados* permiten buscar a través de muchas llaves con sólo unos pocos accesos al disco.

Por desgracia, resulta difícil combinar la idea de *paginación* de estructuras de árboles con la de *balanceo* de estos árboles por el método AVL. El problema se presenta al seleccionar miembros de la página raíz de un árbol o subárbol cuando el árbol se construye en la forma descendente convencional.

Como consecuencia de lo anterior, es necesaria la utilización de los **árboles B**. Estos árboles, resuelven el dilema de la paginación y balanceo iniciado desde el nivel de las hojas y promoviendo llaves hacia arriba conforme el árbol crece.

Una vez entendidas las operaciones básicas sobre los árboles B, es una buena opción incluir refinamientos y mejoras a dichos árboles. El primer conjunto de mejoras, implica incrementar la utilización del almacenamiento dentro de los árboles B, lo cual puede dar como resultado una disminución de la altura del árbol y, por lo mismo, mejoras en el desempeño.

Se puede observar que algunas veces *redistribuir* llaves durante la *inserción*, en vez de *dividir* páginas, puede mejorar la utilización de *espacio* en los árboles B, de tal manera que sea de alrededor del 85 por ciento. Si se lleva aún más lejos la búsqueda de incrementos de eficiencia en almacenamiento, se encuentra que puede combinarse la redistribución durante la inserción con un tipo deferente de división, para asegurar que alrededor de dos terceras partes y no sólo la mitad de las páginas estén llenas después de una división. Los árboles que usan esta combinación de redistribución y división se llaman **árboles B*.**

Si bien los árboles B pueden ser estructuras de almacenamiento muy eficientes y flexibles, que mantienen su balanceo luego de las operaciones, es necesario encontrar una forma de hacer uso eficiente de los índices que son demasiado grandes para almacenarse en su totalidad en memoria RAM.

Teniendo en cuenta esto y desechando la idea fija de todo o nada en cuanto a almacenar todo el índice en la memoria o todo en almacenamiento secundario, es posible considerar almacenar parte de él, a través del manejo de *páginas en buffers* creando un **árbol B virtual**, donde se guardan varias páginas en memoria RAM anticipando la posibilidad de un acceso posterior a una o más de ellas.

Se desarrollan dos métodos para determinar las páginas del árbol que se van a utilizar. Un método usa la *altura de la página* en el árbol para decidir cuáles conviene guardar: mantener la raíz tiene la más alta prioridad, los descendientes de la raíz tiene la siguiente propiedad, y así sucesivamente. El segundo método para la selección de páginas por guardar en memoria RAM se basa en lo *reciente del uso*: siempre se reemplaza la página cuyo uso es menos reciente. *(LRU)*.

Una cuestión a tener en cuenta radica en ver dónde colocar la información asociada con una llave del índice del árbol B. Almacenarla con la llave es atractivo porque, así, encontrar la llave es lo mismo que encontrar la información; no se requieren accesos adicionales al disco.

Sin embargo, si la información asociada ocupa mucho espacio, se puede reducir el orden del árbol, incrementando por consiguiente su altura. En tales casos con frecuencia resulta ventajoso almacenar la información asociada en un archivo separado.

Hasta aquí se ha visto el acceso indexado o secuencial por el orden de la llave, pero sin encontrar una forma eficiente de proporcionar ambos tipos de accesos. La solución a este problema, se basa en el uso de un conjunto de *secuencia en bloques* y un *conjunto índice* asociado.

El conjunto de secuencias almacena todos los registros de datos del archivo ordenados por llave. En cuanto a la construcción del índice asociado, si el índice es lo suficientemente pequeño para caber en memoria RAM, una solución bastante satisfactoria es usar un índice sencillo que contenga, por ejemplo, la llave del último registro en cada bloque del conjunto de secuencias. Si el índice se vuelve demasiado grande para entrar en la memoria RAM, es recomendable el uso de la misma estrategia usada cuando el uso de un índice simple sobrepasa el espacio disponible de memoria RAM: se convierte el índice en un árbol B. Esta combinación de un conjunto índice en forma de árbol B es el primer encuentro con la estructura conocida com **arbol B+.**

Un árbol B+ consiste en un conjunto de secuencias de registros ordenados por llave en forma secuencial, junto con un conjunto índice que proporciona acceso indexado a los registros. Las principales *ventajas* que los árboles B+ ofrecen sobre los árboles B son:

- Permiten un verdadero acceso secuencial indexado, y

- El conjunto índice contiene sólo separadores en vez de llaves y registros completos, de modo que con frecuencia es posible crear un árbol B+ de menor altura que un árbol B.

Unidad 14: Dispersion = Hashing

Existen tres modos principales de acceder a los archivos: en *forma secuencial*, a través de un *índice*, y de manera *directa*.

La ***dispersión*** representa la principal forma de organización de archivos para permitir el acceso directo. Puede proporcionar un acceso más rápido que la mayoría de las otras organizaciones y con muy poco desperdicio de almacenamiento; además es adaptable a casi todo tipo de llaves primarias**. En teoría, la dispersión hace posible encontrar cualquier registro con sólo una acceso al disco, pero este ideal se logra rara vez**.

La desventaja principal de la dispersión es que los archivos con dispersión no pueden clasificarse por llave.

DEFINICIONES:

1. **Tecnica para generar una direccion base unica para una llave dada. Se usa cuando se requiere un acceso rapido a una llave.**
2. **Tecnica que convierte la llave de un registro en un numero casi aleatorio, que despues sirve para determinar donde se almacena el registro.**
3. **Tecnica de almacenamiento y recuperacion que usa una funcion de hash para mapear registros en direccion de almacenamiento.**

ATRIBUTOS DEL HASH

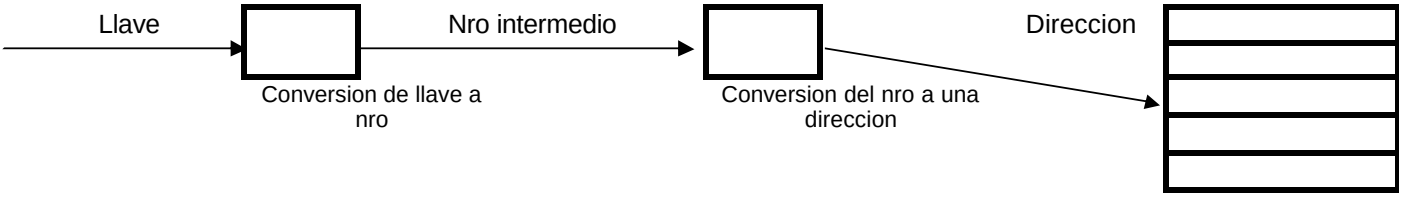
- ♦ No requiere almacenamiento adicional (indice).
- ♦ Facilita la insercion y eliminacion rapida de registros.
- ♦ Encuentra registros con pocos accesos a disco.

COSTOS

- ♦ No se puede usar registros de longitud variable.
- ♦ No puede haber orden fisico de los datos
- ♦ No permite llaves duplicadas.

DETERMINACION DE UNA DIRECCION:

1. la clave se convierte en un numero aleatorio.
2. El numero se convierte en una direccion de memoria.
3. El registro se guarda en esa direccion
4. Si la direccion esta ocupada hay overflow que tiene un tratamiento especial.



PARAMETROS QUE AFECTAN LA EFICIENCIA:

- ♦ Tamaño de las cubetas
- ♦ Densidad de empaquetamiento
- ♦ Funcion de hash
- ♦ Metodo de tratamiento de desborde

Funcion de dispersion

La dispersión implica la aplicación de una ***función de dispersión h(K)*** a una llave de *registro K* para producir una dirección. La dirección se toma como la dirección base del registro cuya llave es K, y forma la base para la búsqueda del registro. Las direcciones producidas por las funciones de dispersión por lo general parecen ser aleatorias.

Cuando dos o más llaves se dispersan a la misma dirección, se les llama ***sinónimos***. Si en una dirección no caben todos sus sinónimos sobrevienen las ***colisiones (*****situacion en la que un registro es asignado a una direccion que no tiene sifuciente espacio para almacenarlo)**, y esto hace que algunos de los sinónimos no puedan almacenarse en la dirección base y deban estar en algún otro lugar.

Como las búsquedas de registros comienzan en sus direcciones base, las búsquedas de registros que no están en sus direcciones base por lo general implican accesos adicionales al disco.

El término ***longitud de media de búsqueda*** describe el número promedio de accesos a disco que se requieren para extraer un registro. La longitud de búsqueda ideal es 1.

EJEMPLOS DE FUNCIONES DE DISPERSION

- ✯ **Centros cuadrados**: la llave se multiplica por si misma y tomando los dígitos centrales al cuadrado, posteriormente se ajusta al espacio diponible
- ✯ **División**: la clave se divide por un # aproximadamente igual al # de direcciones (número primo pues tiende a distribuir residuos en forma más eficiente)
- ✯ **Desplazamiento**: los dígitos externos de ambos extremos se corren hacia adentro, se suman y se ajusta al espacio disponible
- ✯ **Plegado**: los dígitos externos se pliegan, suman y adaptan al espacio de direcciones.
- ✯ **Análisis de dígitos**: analizan las claves para eliminar posibles repeticiones en la misma.
- ✯ **Conversión de raíz**: la base del número se modifica y en la serie de dígitos resultante se suprimen dígitos de orden mayor.
- ✯ **División polinómica**: cada dígito clave toma como coeficiente de polinomio, se divide por polinomio fijo, coeficinete del resto se toma como dirección.

Existen tres *métodos generales* para reducir el número de colisiones:

1.Esparcir los registros

Esparcir los registros implica elegir una función de dispersión que distribuya los registros en el espacio de direcciones al menos en forma aleatoria. Una *distribución uniforme* esparce los registros en forma equitativa, evitando así la ocurrencia de colisiones. Una *distribución aleatoria* o casi aleatoria es mucho más fácil de lograr, y usualmente se considera aceptable.

En un algoritmo sencillo de dispersión los pasos a seguir son:

- Representar la llave en forma numérica.
- Desglosar y sumar.
- Dividir entre el tamaño del espacio de direcciones, produciendo una dirección válida.

Cuando se examinan varios tipos diferentes de algoritmos de dispersión, se observa que a veces pueden encontrarse algunos que produzcan distribuciones mejor que aleatorias. Pero, en su defecto, se sugieren algunos algoritmos que por lo general producen distribuciones que son aproximadamente aleatorias.

La ***distribución de Poisson*** proporciona una herramienta matemática para examinar en detalle los efectos de una distribución aleatoria. Las funciones de Poisson pueden usarse para predecir los números de direcciones a las que es probable asignar 0, 1, 2, o más registros, considerando el número de registros que van a dispersarse y el número de direcciones disponibles. Esto permite predecir el número de colisiones que pueden ocurrir cuando se disperse el archivo, el número probable de registros en saturación, y algunas veces la longitud media de búsqueda.

2. Usar memoria adicional

Distribuye pocos registros en muchas direcciones, lo que hace que haya menos sinonimos que si se asugnan pocas direcciones adicionales.
Ventaja➔ se disminuye el overflow.
Desventaja➔se desperdicia espacio.

El ***término densidad de empaquetamiento*** describe la proporción del espacio de direcciones disponible que realmente está almacenando registros. La función de Poisson se usa para determinar cómo influyen las diferencias en la densidad de empaquetamiento sobre el porcentaje de registros que probablemente serán sinónimos.

3.Usar compartimientos (colocar mas de un registro por direccion)

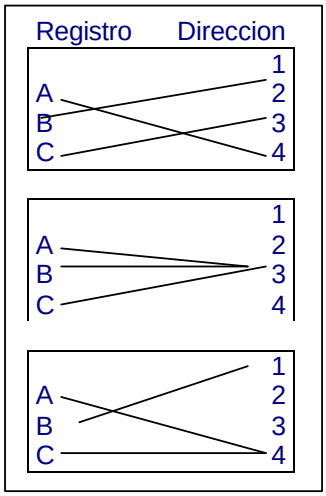
Compartimiento ➔direcciones del archivo que pueden almacenar varios registros.
Tamaño de compartimiento➔ El número de registros que pueden almacenarse en una dirección dada, determina el punto en el cual los registros asignados a la dirección provocarán saturación.
Puede usarse la función de Poisson para explorar los efectos de las variaciones en los tamaños de los compartimientos y las densidades de empaquetamiento. Los compartimientos grandes, combinados con una densidad de empaquetamiento baja, pueden dar como resultado longitudes medias de búsqueda pequeñas.

Algoritmos simple de dispersion

OBJETIVO➔ debe esparcir los registros tan uniformemente como sea posible en el intervalo de direcciones disponible.

3 PASOS:

1. Representar la llave en forma numerica (en caso de que no sea un numero)
2. Desglosar y sumar ➔ tomar pedazos del numero y sumarlos.
3. Dividir entre el tamaño del espacio de direcciones ➔ recortar el numero producido para que este dentro del intervalo de direcciones de registros en el archivo.



Clave ➔ numero ➔ conversion ➔ direccion

```
FUNCTION dispersion  (LLAVE, CANTMAXDIR)
        SUM:=0;
        J:=0;
        Mientras (J < 12)
                SUM :=SUM + 100*LLAVE[J] + LLAVE[J+1] mod 20000 ;
```

```
            J :=J+ 2;
        Fin mientras;
        Retorna (SUM mod CANTMAXDIR)
Fin FUNCTION
```

Funciones de dispersion y distribuciones de registros

Distribucion uniforme (mejor) = sin sinonimos ➔funcion de dispersion que distribuir los registros en un archivo de modo que no existan colisiones porue se distribuyen en forma uniforme entre las direcciones.

Peror forma de distribucion = todos son sinonimos➔Todos los registros comparten la misma direccion base, y da como resultado el numero maximo de colisiones.

Aleatoria (aceptable) = pocos sinonimos ➔ los registros estan algo esparcidos, con pocas colisiones: para una determinada llave, cualquier direccion tiene la misma probabilidad que las demas de ser elegida.

Metodos de dispersion

Analisis de digitos = examinar la llave para buscar un patron ➔ si alguna parte de la llave muestra un patron util, puede usarse una funcion de dispersion que extraiga esa parte de la llave.
Desglosar partes de la llave ➔ extraer digitos de parte de una llave y sumarlas. Destruye los patrones originales de la llave.
Division ➔ dividir la llave por un numero aproximadamente igual al tamaño de la direccion y usar el residuo.
Centro de cuadrados ➔ elevamos la llave al cuadrado y extraemos de la mitad del resultado el numero de digitos que sea necesario.
Transformacion de la base o raiz ➔ convertir la llave a otra base numerica y en el resultado se suprimen los de orden mayor.
Desplazamiento ➔ los digitos externos de ambos extremos, se corren hacia adentro se suman los digitos y multiplica por ajuste.
Pegado ➔ digitos externos de una clave se pliegan, suman y ajustan.
Division polinomica ➔ cada digito clave se toma como coeficiente de polinomios, se dvide por un polinomio fijo y se toman los coeficientes del resto como direccion.

Prediccion de la distribucion de los registros

Se desea predecir el numero probable de colisiones en un archivo que puede guardar solo un registro por direccion.

Tenemos N direcciones en un archivo. Cuando dispersamos una llave existen dos posibles resultados para la direccion: A = la direccion no se elige o B = la direccion se elige.

P(B) = b = 1/N → probabilidad de que la direccion sea elegida.
P(A) = a = (N - 1)/N = N - 1/N → probabilidad de no ser elegida.

Se dispersan dos llaves, la probabilidad de que ambas produzcan la direccion dada: p(BB) = b x b = 1/N x 1/N y la probabilidad de que las llaves se dispersen en otra direccion diferente a la dada: p(BA)=bxa = 1/N x (1 - 1/N).

En general para calcular la probabilidad de una secuencia de resultados (BABBA) puede reemplazarse cada A y B po a y b respectivamente y calcular el producto indicado: p(BABBA) = bxaxbxbxa = a^2 b^3 .

FUNCION DE POISSON APLICAD A LA DISPERSION

$$P(x) = ((r/N)^x e^{-(r/N)})/ x!$$

N = numero de direcciones disponibles, r = nro de registros que se van a almacenar y x = nro de registros asignados a una direccion dada. Entonces **p(x) indica la probabilidad de que a una direccion determinada se hayan asignado x registros luego de haber aplicado la funcion de dispersion a los n registros**.

En general si hay N direcciones el numero esperado de direcciones con x registros asignados a ella es

Np(x)

Densidad de empaquetamiento

Definicion ➔ proporcion entre el numero de registros por almacenar (r) y el numero de espacios dispponibles (N). Es una medida de la cantidad del espacio que se usa en realidad en un archivo. Es el valor necesario para evaluar el desempeño en un ambiente de dispersion.

Densidad de empaquetamiento = r/N = nro de registros/nro de espacios

Cuantos mas registros esten empaquetados en un espacio del archivo mas probable es qoe ocurra una colision al agregar un registro nuevo.

Resolucion de colisiones mediante saturacion progresiva

Aunque el número de colisiones puede reducirse, se necesita algún medio para enfrentarse a ellas cuando ocurran. Una técnica sencilla de solución de colisiones es la ***saturación progresiva***. Si el intento de almacenar un registro nuevo da como resultado una colisión, la saturación progresiva implica la búsqueda ordenada en las direcciones que siguen a la dirección base del registro, hasta que se encuentre una direccion libre para almacenar el registro nuevo.

La busqueda de un registro comienza en su dirección base y despues se continua buscando en direcciones sucesivas. Pueden suceder dos cosas:

- Se encuentra una direccion vacia ➔ la rutina de busqueda puede suponer que el registro no esta en el archivo.
- El archivo esta lleno ➔ la busqueda vuelve a donde comenzo para estar seguros que el registro no esta en el archivo.

La saturación progresiva es sencilla y algunas veces funciona muy bien; sin embargo, crea longitudes de búsqueda grandes cuando la densidad de empaquetamiento es alta y el tamaño del compartimiento es reducido.

Si hay muchas colisiones habre bastantes registros en saturacion que ocupan espacios que no deben, tambien pueden formarse cúmulos de registros lo que hace que estos se coloquen a gran distancia de su direcion base, requiriedo muchos accesos a disco para extraerlos.

Longitud de busqueda➔ numero de accesos requeridos para extraer un registro de memoria secundaria. Se incrementa cada vez que ocurre una colision.

Se crean longitudes de búsqueda muy grandes para registros que se encuentran lejos de su direccion base.

Longitud promedio de busqueda ➔ numero promedio de veces que se espera acceder al disco. Aumenta cuando se incrementa la densidad de empaquetamiento

Long media de busqueda = long total de busqueda / nro total de reg

Una longitud de busqueda media mayor de 2 se considera inaceptable por lo que tenemos que usar menos del 40% del almacenamiento para tener un desempeño tolerable.

Esta situacion puede mejorarse colocando mas de un registro en una sola direccion.

Compartimiento ➔ uno o mas sectores de disco que se extraen en un acceso al disco.
Cuando un registro se almacena o extrae su direccion de compartimiento se determina por dispersion.
Un compartimiento completo se carga en RAM y se busca en forma secuencial en el mismo hasta encontrar el registro deseado.
Si el compartimiento se llena hay saturacion.
El tamaño de los compartimientos puede ser el de una pista (puede ser demasiado grande cuando se considera el tiempo que lleva transmitirla completa comparado con lo que se tarda en transmitir varios sectores). Otro tamaño mejor puede ser un cumulo.
Cuando mas grande es el compartimiento menor es la longitud de busqueda

ESTRUCTURA DEL COMPARTIMIENTO

Todos los registros que se guardan en un compartimiento comparten la misma direccion

Cont=2	reg	Reg			

Cada compartimiento tiene un contador que mantiene la informacion sobre cuantos registros se han almacenado y se usa una marca especial para indicar que un registro esta vacio.
Se produce una colicion cuando al agregar un registro nuevo el contador excede la cantidad de registros que puede almacenar.

INICIALIZACION DE UN ARCHIVO PARA DISPERSION

El tamaño logico de un archivo debe permanecer fijo, por lo que le asignamos el espacio fisico antes de empezar a almacenar registros. (creamos un archivo con espacios vacios que luego se llenaran con los registros de datos según sea necesario

CARGA DE UN ARCHIVO CON DISPERSION

El programa usa la funcion de dispersion para producir una direccion base para cada llave. La busqueda de espacios libres comienza con el compartimiento almacenado en su direccion base y si este esta lleno se continua con los sucesivos hasta encontra uno vacio., se inserta el registro nuevo y se lo transcribe al archivo en el lugar desde el que se carga. La busqueda de espacio puede resultar un ciclo infinito si el archivo esta lleno.

ELIMINACION

Hay tres problemas asociados con la eliminación de registros en los archivos con dispersión, y son:

1. La posibilidad de que los espacios vacíos creados por las eliminaciones impidan búsquedas posteriores de registros en saturación.
2. La necesidad de recuperar el espacio que queda disponible cuando se eliminan los registros.
3. El deterioro de las longitudes medias de búsqueda causado por los espacios vacíos, los cuales mantienen a los registros más delante de lo necesario de su dirección base.

Los primeros dos problemas pueden resolverse usando *marcas de inutilización* a fin de identificar registros eliminados, reemplazando el registro por un marcador que indique que hubo pero ya no un registro en ese lugar.
Las soluciones al problema del deterioro de las longitudes medias de busqueda, implica hacer reorganización local cada vez que sucede una eliminacion (el algoritmo de eliminacion puede examinar los registros que siguen a una marca de inutilizacion para sabre si se puede acortar la longitud de busqueda moviendo el registro hacia la direccion base). Otra solucion implica la reorganización completa del archivo luego de que la longitud media de busqueda alcanza un valor inaceptable. Otra posibilidad es elegir un algoritmo de resolución de colisiones que no provoque deterioro.

Otras tecnicas de resolucion de colisiones

1. Dispersion doble

Reduce el acumulamiento local, pero puede colocar algunos registros en saturación a gran distancia de su dirección base (no quedan locales), por lo que necesitan desplazamientos adicionales. Los programas con dispersion doble pueden solucionar este problema si son capaces de generar direcciones en saturacion de tal modo que los registros en saturacion se mantengan en el mismo cilindro que los registros con direcion base.
Se resuelven colisiones aplicando una segunda función a la llave para producir un numero c que se suma a la direccion original tantas veces como sea necesario hasta encontrar una direccion vacia.

2. Saturacion progresiva encadenada

Similar a progresiva. Los registros de saturación se encadenan y no están necesariamente en posiciones contiguas. Cada uno de los registros posee un puntero al siguiente en la lista de colisionados (campo liga).
Ventaja ➔ en cualquier busqueda solo se necesita acceder a los registros con llaves que son sinonimos
Se debe procurar que toda direccion que se clasifique como direccion base para algun registro del archivo realmente se almacena un registro. Una forma de asegurar esto es que se haga la carga del archivo en dos pasos: en el primer paso sólo se cargan los registros con direcciones bases y en una segunda fase se almacenan los registros en saturación en cada una de las direcciones libres.

3. Encaminamiento con un área de saturacion separada

Llamamos área principal de datos al conjunto de direcciones base y al conjunto de direcciones de saturacion se le llama área de saturación (que está separada) , estas claves están ligadas mediante punteros. Este tipo de disposición hace que se requieran mas movimientos de la cabeza del disco.
Simplifica considerablemente el encadenamiento, y tiene la ventaja de que el área de saturación puede ser organizada en formas más apropiadas para el manejo de los registros en saturación.
Un peligro de este método es que se puede perder localidad.

TABLAS DE DISPERSION ➔ en lugar de tener registros en el archivo tenemos solamente apuntadores a registros, el archivo es solamente un indice donde se busaca por dispersion. Es una tabla que puede estar en memoria y que tiene una entrada por cubeta en el archivo.
Las tablas de dispersion ofrecen la ventaja de que un solo acceso encontramos un indice por lo que es bueno cuando hay mas operaciones de recuperacio que inserciones. Tambien manejan convenientemente el uso de registros de longitud variable.
Desventaja: las inserciones son lentas.
El archivo de datos puede ser de diferentes formas:

- Conjunto de listas ligadas de sinonimos
- Un archivo clasificado
- Un archivo de entradas secuenciales

4. Dispersion indizada

En lugar de usar los resultados de una dispersión para producir la dirección de un registro, la dispersión puede usarse para identificar una localidad en un índice que a su vez apunte a la dirección del registro.
A pesar de que en este método cada búsqueda requiere un acceso adicional, permite organizar los registros de datos reales de un modo que facilita otros tipos de procesamiento, como el procesamiento secuencial.

5.Dispersion extensible

Implica doblar el tamaño de la tabla de dispersion cuando sea necesario redispersar el archivo. Si se cumplen las siguientes restricciones la tecnica funciona con razonable eficiencia:

- la funcion de dispersion debe generar un entero mucho mayor que el numero maximo de direcciones que se usaran, luego dividir este nro entre el tamaño actual de direcciones y tomar el residuo.
- Cuando se reorganiza la tabla , el nro de direccions debe multiplicarse por un entero fijo (2, la tabla duplica su tamaño).

Combina la indización con la dispersión, pero permite que el tamaño del espacio de direcciones crezca dinámicamente en tanto crece el archivo sin tener que reorganizar la tabla entera cada vez que este crece. En esta técnica la función de hash genera un número (generalmente de 32 bits).
En cada momento se tiene una cantidad i de bits (que queda determinada de acuerdo al tamaño del archivo). Si se intenta insertar en un cubeta que está llena se debe agregar un bit y reubicar todos los registros que estaban en esa cubeta. Para identificar la cubeta en la que se debe ubicar la clave, se toman los primeros i bits.

Como en muchos casos se accede a algunos registros con más frecuencia que a otros, a menudo vale la pena tomar en cuenta los patrones de acceso. Si pueden identificarse aquellos registros a los cuales se accede con mayor probabilidad, pueden tomarse precauciones para asegurarse que estén almacenados más cerca de la dirección base que aquellos a los cuales se accede con menor frecuencia; esto disminuye la longitud media de búsqueda ***efectiva***. Una de tales medida es cargar los registros a los cuales se accede con mayor frecuencia antes que los otros.

❑ **Definiciones**

Colisión: Situación en la que un registro es asignado a una dirección que no tiene suficiente lugar para almacenarlo. Cuando ocurre una colisión deben encontrarse medios para resolverla.

Compartimiento: Área de espacio en el archivo que se trata como un registro físico para propósitos de almacenamiento y extracción, pero que puede almacenar varios registros *lógicos*. Al almacenar y extraer registros lógicos en compartimientos en vez de hacerlo individualmente, los tiempos de acceso pueden mejorar considerablemente en muchos casos.

Densidad de empaquetamiento: proporción de espacio del archivo asignado que en realidad almacena registros. Si un archivo está medio lleno, su densidad de empaquetamiento es 50 por ciento. La densidad de empaquetamiento y el tamaño del compartimiento son las dos medidas más importantes en la determinación de la probabilidad de que ocurra una colisión cuando se busca un registro en un archivo.

Dirección base: dirección generada por una función de dispersión para una llave dada. Si un registro está almacenado en su dirección base, entonces la longitud de búsqueda del registro es uno, porque sólo se requiere un acceso para extraerlo. Para extraer o almacenar un registro que no está en su dirección base se requiere más de un acceso.

Dispersión: Técnica para generar una dirección base única para una llave dada. La dispersión se usa cuando se requiere acceso rápido a una llave (o a su registro correspondiente).

Dispersión mínima: esquema de dispersión donde el número de direcciones es exactamente igual al número de registros. No se desperdicia espacio de almacenamiento.

Función de dispersión perfecta: Función de dispersión que distribuye registros en forma uniforme, minimizando el número de colisiones. Las funciones de dispersión perfecta son muy deseables, pero extremadamente difíciles de encontrar para conjunto grandes de llaves.

Longitud media de búsqueda: se define la longitud media de búsqueda como la suma *del número de accesos requeridos para cada registro en el el archivo*, dividida entre el *número de registros del archivo*. Esta definición no toma en cuante el número de accesos requeridos por búsquedas sin éxito, ni tampoco el hecho de que es más probable que se acceda a algunos registros con más frecuencia que a otros.

Marca de inutilización: Marcador especial colocado en el campo llave de un registro para indicar que ya no es válido. El uso de marcas de inutilización resuelve dos problemas asociados con la eliminación de registros: que el espacio liberado no rompa la búsqueda secuencial de un registro, y que el espacio liberado sea fácilmente reconocido como disponible y pueda ser utilizado para adiciones posteriores.

Saturación (desbordamiento): Situación que ocurre cuando un registro no puede almacenarse en su dirección base.

Saturación progresiva: Técnica de manejo de saturación en la que las colisiones se resuelven almacenando un registro en la siguiente dirección disponible después de su dirección base. La saturación progresiva no es la técnica de manejo de saturación más eficiente, pero es una de las más simples y es adecuada para muchas aplicaciones.

Sinónimos: Dos o más llaves diferentes que se asignan a la misma dirección base. Cuando cada dirección del archivo puede almacenar un solo registro, los sinónimos siempre ocasionan colisiones. Si se usan compartimientos, pueden almacenarse sin colisiones varios registros cuyas llaves sean sinónimos.