INSTITUTO SUPERIOR TECNOLÓGICO

JAPÓN

Amor al conocimiento

GUÍA METODOLÓGICA

BASE DE DATOS II

DESARROLLO DE SOFTWARE

AUTOR: LCDA. PAULINA JARAMILLO

2020

1. IDENTIFICACIÓN DE

Nombre de la Asignatura: **BASE DE DATOS II**	**Componentes del Aprendizaje**	Componente docencia: 54 Componente de prácticas de aprendizaje: 53 Componente de aprendizaje autónomo: 35
Resultado del Aprendizaje: **COMPETENCIAS Y OBJETIVOS** • Planificar, implementar y gestionar el uso de las Tecnologías de Información y Comunicación de una organización, a partir del análisis de sus requerimientos, teniendo en cuenta los criterios de calidad, seguridad y ética profesional propiciando el trabajo en equipo. • Dominar los conceptos para la creación y manipulación de una Base de Datos considerando aspectos avanzados para el diseño e implementación de un sistema de Base de Datos, como son el control de la concurrencia y las técnicas de recuperación, mediante el análisis de modelos avanzados de bases de datos tales como las bases de datos orientada a objetos y base de datos distribuida. • Diseña Bases de Datos Relacionales. • Crea Bases de Datos Relacionales.		
Docente de Implementación:		
PAULINA JARAMILLO FLORES	**Duración:** 142 horas	

Unidades	Competencia	Resultados de Aprendizaje	Actividades	Tiempo de Ejecución

ORGANIZACIÓN FÍSICA DE BASE DE DATOS	Enlistar conceptos básicos de la Organización Y almacenamiento Física de la Base de Datos.	**COGNITIVO:** Reconoce conceptos relacionados a la organización física de una BDD **PROCEDIMENTAL**: Implementa modelos de base de datos relacionales con estándares de calidad **ACTITUDINAL**: Describe las características de los componentes y funciones de una BDD	Exposición visual con diapositivas de conceptos básicos de organización física de las bdd Lluvia de ideas Estimulación multimedios con presentación de metodología de diseño de bases de datos relacionales	**8**
SISTEMAS DE GESTIÓN DE BASES DE DATOS	Analizar las características y funcionamiento de los sistemas de gestión de bases de datos	**COGNITIVO:** Distingue mecanismos de reconstrucción de BDD **PROCEDIMENTAL:** Examina el procesamiento de transacciones y acceso concurrentes en una BDD **ACTITUDINAL:** Describe las características de los componentes y funciones de una BDD.	Estimulación verbal, diálogo sobre conceptos y funciones de un SGBD Exposición multimedios sobre transacciones, accesos concurrentes y reconstrucción de una BDD Mapa mental Generación modelo físico BDD del caso de estudio	**8**

TIPOS DE BASES DE DATOS	Describir los diferentes tipos de bases de datos	**COGNITIVO:** Reconoce fundamentos de los modelos de BDD tradicional y jerárquico **PROCEDIMENTAL:** Compara funcionamiento e implementación de los modelos de BDD de red y relacional **ACTITUDINAL:** Analiza consideraciones para implementación de los modelos de BDD avanzados, OO y Declarativo	Estimulación multimedios, presentación visual sobre modelos tradicionales de BDD Argumentación con los estudiantes sobre criterios de selección del modelo de la BDD	**8**
LABORATORIO DE BASES DE DATOS	Experimentar en laboratorio la implantación de un sistema de base de datos	**PROCEDIMENTAL:** Gestiona información desde una BDD distribuida **ACTITUDINAL:** Diseña y construye una bdd relacional PostgreSql	Lluvia de ideas Exposición individual BDD NoSql, estudio de casos Observación directa en computador Ejemplificación caso de estudio	**8**

2. CONOCIMIENTOS PREVIOS Y RELACIONADOS

Co-requisitos

3. UNIDADES TEÓRICAS

- **Desarrollo de las Unidades de Aprendizaje (contenidos)**

A. Base Teórica

UNIDAD 1: ORGANIZACIÓN FÍSICA DE BASE DE DATOS

La colección de datos que constituye una base de datos debe estar almacenada en un medio de almacenamiento del ordenador. De esta manera, el SGBD será capaz de recuperar, actualizar y procesar dichos datos siempre que sea necesario.

Según la velocidad de acceso a los datos, el coste y la fiabilidad hay tipos de almacenamiento:

- Almacenamiento primario: memoria volátil (caché y RAM)

- Memoria caché:
  - rápida y costosa.
  - Tamaño pequeño.
  - Gestionada por el SO.
  - No para usuario.
- Memoria principal:
  - es rápida (<100ns)
  - barata
  - volátil
  - para datos en proceso.

- Almacenamiento secundario: discos magnéticos, memoria no volátil (flash),discos RAID, discos ópticos

- Memoria flash:
  - sólo de lectura
  - programable y borrable eléctricamente
  - No volátil.
  - Velocidad lectura: igual que la memoria principal
  - Sucesivas escrituras: hay que borrar antes lo que había; por bancos completo.
  - Inconveniente: número limitado de ciclos de borrado (10 M).
  - Para dispositivos móviles, tarjetas inteligentes.

Discos magnéticos:

- Discos magnéticos: La unidad principal más usada (ver sección siguiente).

- Discos RAID. Disposición redundante de discos independientes (RAIDs - Redundant Array of Independent Disks).
    - o Mejora de la fiabilidad:
    - • redundancia. Creación de imágenes o sombras (mirrow).
    - • cada proceso de escritura se lleva a cabo en dos discos físicos
    - o Mejora del rendimiento:
    - • distribuir la carga entre todos los discos: las cabezas lectoras se reparten el trabajo
    - • accesos paralelos en peticiones de datos masivas: recuperan bits desde discos diferentes

- Discos ópticos y cintas se extraen del dispositivo. Usos:
    - o copias de seguridad (backup)
    - o almacenamiento de datos poco usados y/o estables. ej.: datos históricos
    - o transferir información sin conexión ej.: programas
    - o Almacenamiento óptico.
    - • Memoria sólo de lectura en disco compacto (Compact-Disk Read- Only Memory,CD-ROM).
    - • Memoria de escritura única y lectura múltiple (Write-Once, Read- only Memory, WORM).
- • Dispositivos de almacenamiento combinados magnetoópticos que utilizan medios ópticos para leer datos codificados magnéticamente: Jukebox, DVD. (15GB)
- o Cintas. Son DAT, graban datos en formato digital. Son muy baratas y fiables.

HISTORIA BASE DE DATOS: EVOLUCIÓN HISTÓRICA.

El uso de sistemas de bases de datos automatizadas, se desarrolló a partir de la necesidad de almacenar grandes cantidades de datos, para su posterior consulta, producidas por las nuevas industrias que creaban gran cantidad de información.

Herman Hollerit (1860-1929) fue denominado el primer ingeniero estadístico de la historia, ya que invento una computadora llamada “Máquina Automática Perforadora de Tarjetas”. Para hacer el censo de Estados Unidos en 1880 se tardaron 7 años para obtener resultados, pero Herman Hollerit en 1884 creó la máquina perforadora, con la cual, en el censo de 1890 dio resultados en 2 años y medio, donde se podía obtener datos importantes como número de nacimientos, población infantil y número de familias. La máquina uso sistemas mecánicos para procesar la información de las tarjetas y para tabular los resultados.

A diferencia con la máquina de Babbage, que utilizaba unas tarjetas similares, estas se centraban en dar instrucciones a la máquina. En el invento de Herman Hollerit, cada perforación en las tarjetas representaba un número y cada dos perforaciones una letra, cada tarjeta tenía capacidad para 80 variables. La máquina estaba compuesta por una perforadora automática y una lectora, la cual por medio de un sistema eléctrico leía los orificios de las tarjetas, esta tenía unas agujas que buscaban los orificios y al tocar el plano inferior de mercurio enviaba por medio del contacto eléctrico los datos a la unidad.

Este invento disparo el desarrollo de la tecnología, la industria de los computadores, abriendo así nuevas perspectivas y posibilidades hacia el futuro.

Década de 1950

En este lapso de tiempo se da origen a las cintas magnéticas, las cuales sirvieron para suplir las necesidades de información de las nuevas industrias. Por medio de este mecanismo se empezó a automatizar la información de las nóminas, como por ejemplo el aumento de salario. Consistía en leer una cinta o más y pasar los datos a otra, y también se podían pasar desde las tarjetas perforadas. Simulando un sistema de Backup, que consiste en hacer una copia de seguridad o copia de respaldo, para guardar en un medio extraíble la información importante. La nueva cinta a la que se transfiere la información pasa a ser una cinta maestra. Estas cintas solo se podían leer secuencial y ordenadamente.

Década de 1960

El uso de los discos en ese momento fue un adelanto muy efectivo, ya que por medio de este soporte se podía consultar la información directamente, esto ayudo a ahorrar tiempo. No era

necesario saber exactamente donde estaban los datos en los discos, ya que en milisegundos era recuperable la información. A diferencia de las cintas magnéticas, ya no era necesaria la secuencialidad, y este tipo de soporte empieza a ser ambiguo.

Los discos dieron inicio a las Bases de Datos, de red y jerárquicas, pues los programadores con su habilidad de manipulación de estructuras junto con las ventajas de los discos era posible guardar estructuras de datos como listas y árboles.

LENGUAJE COBOL – Reunión del Pentágono 1959

El lenguaje COBOL (acrónimo de COmmon Business -Oriented Language, Lenguaje Común Orientado a Negocios) fue creado en el año 1960 con el objetivo de crear un lenguaje de programación universal que pudiera ser usado en cualquier ordenador, ya que en los años 1960 existían numerosos modelos de ordenadores incompatibles entre sí, y que estuviera orientado principalmente a los negocios, es decir, a la llamada informática de gestión.

En la creación de este lenguaje participó la comisión CODASYL, compuesta por fabricantes de ordenadores, usuarios y el Departamento de Defensa de Estados Unidos, en mayo de 1959. La definición del lenguaje se completó en poco más de seis meses, siendo aprobada por la comisión en enero de 1960. El lenguaje COBOL fue diseñado inspirándose en el lenguaje Flow-Matic de Grace Hopper y el IBM COMTRAN de Bob Bemer, ya que ambos formaron parte de la comisión.

Primeras versiones de BD

A mediados de los sesenta, IBM se unió a NAA para desarrollar GUAM en lo que ahora se conoce como IMS (Information Management System). El motivo por el cual IBM restringió IMS al manejo de jerarquías de registros fue el de permitir el uso de dispositivos de almacenamiento serie, más exactamente las cintas magnéticas, ya que era un requisito del mercado por aquella época.

A mitad de los sesenta, se desarrolló IDS (Integrated Data Store), de General Electric. Este trabajo fue dirigido por uno de los pioneros en los sistemas de bases de datos, Charles Bachmann. IDS era un nuevo tipo de sistema de bases de datos conocido como sistema de red,

que produjo un gran efecto sobre los sistemas de información de aquella generación. El sistema de red se desarrolló, en parte, para satisfacer la necesidad de representar relaciones entre datos más complejos que las que se podían modelar con los sistemas jerárquicos, y, en parte, para imponer un estándar de bases de datos. Para ayudar a establecer dicho estándar, CODASYL (Conference on Data Systems Languages), formado por representantes del gobierno de EEUU y representantes del mundo empresarial, formaron un grupo denominado DBTG (Data Base Task Group), cuyo objetivo era definir unas especificaciones estándar que permitieran la creación de bases de datos y el manejo de los datos. El DBTG presentó su informe final en 1971 y aunque éste no fue formalmente aceptado por ANSI (American National Standards Institute), muchos sistemas se desarrollaron siguiendo la propuesta del DBTG. Estos sistemas son los que se conocen como sistemas de red, o sistemas CODASYL o DBTG.

Década de 1970

Edgar Frank Codd (23 de agosto de 1923 – 18 de abril de 2003), en un artículo "Un modelo relacional de datos para grandes bancos de datos compartidos" ("A Relational Model of Data for Large Shared Data Banks") en 1970 (CODASYL), definió el modelo relacional y publicó una serie de reglas para la evaluación de administradores de sistemas de datos relacionales y así nacieron las bases de datos relacionales.

A partir de los aportes de Codd el multimillonario Larry Ellison desarrollo la base de datos Oracle, el cual es un sistema de administración de base de datos, que se destaca por sus transacciones, estabilidad, escalabilidad y multiplataforma.

Inicialmente no se usó el modelo relacional debido a que tenía inconvenientes por el rendimiento, ya que no podían ser competitivas con las bases de datos jerárquicas y de red. Ésta tendencia cambio por un proyecto de IBM el cual desarrolló técnicas para la construcción de un sistema de bases de datos relacionales eficientes, llamado System R.

ANSI SPARC/DB

En 1975, el comité ANSI-SPARC (American National Standard Institute - Standards Planning and Requirements Committee) propuso una arquitectura de tres niveles para los sistemas de bases de datos, que resulta muy útil a la hora de conseguir estas tres características.

El objetivo de la arquitectura de tres niveles es el de separar los programas de aplicación de la base de datos física. En esta arquitectura, el esquema de una base de datos se define en tres niveles de abstracción distintos:

1. En el nivel interno se describe la estructura física de la base de datos mediante un esquema interno. Este esquema se especifica mediante un modelo físico y describe todos los detalles para el almacenamiento de la base de datos, así como los métodos de acceso.

2. En el nivel conceptual se describe la estructura de toda la base de datos para una comunidad de usuarios (todos los de una empresa u organización), mediante un esquema conceptual. Este esquema oculta los detalles de las estructuras de almacenamiento y se concentra en describir entidades, atributos, relaciones, operaciones de los usuarios y restricciones. En este nivel se puede utilizar un modelo conceptual o un modelo lógico para especificar el esquema.

3. En el nivel externo se describen varios esquemas externos o vistas de usuario. Cada esquema externo describe la parte de la base de datos que interesa a un grupo de usuarios determinados y ocultos a ese grupo el resto de la base de datos. En este nivel se puede utilizar un modelo conceptual o un modelo lógico para especificar los esquemas.

Década de 1980

Las bases de datos relacionales con su sistema de tablas, filas y columnas, pudieron competir con las bases de datos jerárquicas y de red, ya que su nivel de programación era bajo y su uso muy sencillo.

En esta década el modelo relacional ha conseguido posicionarse del mercado de las bases de datos. Y también en este tiempo se iniciaron grandes investigaciones paralelas y distribuidas, como las bases de datos orientadas a objetos.

Principios década de los 90

Para la toma de decisiones se crea el lenguaje SQL, que es un lenguaje programado para consultas. El programa de alto nivel SQL es un lenguaje de consulta estructurado que analiza grandes cantidades de información el cual permite especificar diversos tipos de operaciones frente a la misma información, a diferencia de las bases de datos de los 80 que eran diseñadas para las aplicaciones de procesamiento de transacciones. Los grandes distribuidores de bases de datos incursionaron con la venta de bases de datos orientada a objetos.

Finales de la década de los 90

El boom de esta década fue la aparición de la WWW "Word Wide Web" ya que por éste medio se facilitaba la consulta de las bases de datos. Actualmente tienen una amplia capacidad de almacenamiento de información, también una de las ventajas es el servicio de siete días a la semana las veinticuatro horas del día, sin interrupciones a menos que haya planificaciones de mantenimiento de las plataformas o el software.

Siglo XXI

En la actualidad existe gran cantidad de alternativas en línea que permiten hacer búsquedas orientadas a necesidades específicas de los usuarios, una de las tendencias más amplias son las bases de datos que cumplan con el protocolo Open Archives Initiative – Protocol for Metadata Harvesting (OAI-PMH) los cuales permiten el almacenamiento de gran cantidad de artículos que permiten una mayor visibilidad y acceso en el ámbito científico y general.

DISCO MAGNÉTICO

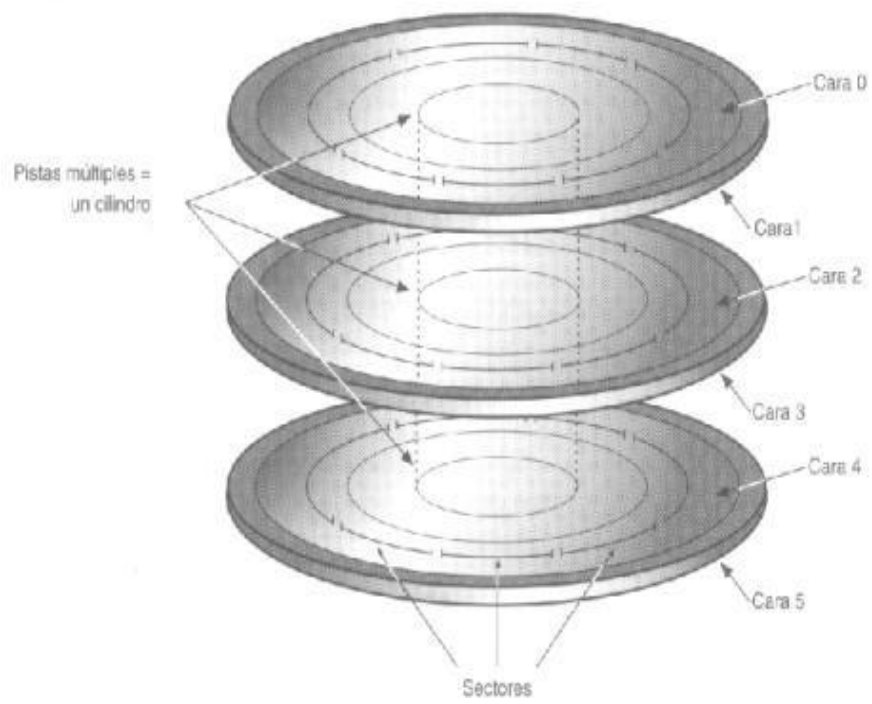
El disco magnético sirve para almacenar grandes cantidades de datos. Las características de estos discos magnéticos son:

- Cada plato /disco tiene dos caras, capaces de albergar información en cada una de ellas, e incluso formando paquetes de discos donde los datos se organizan en varias superficies.
- Pista: división circular concéntrica de la superficie de un disco. Número fijo por disco.
- Cilindro: conjunto de pistas del mismo diámetro de todas las caras que pueden accederse sin mover la cabeza.
- La superficie del disco se divide en pistas, que se subdividen en sectores.

disco.

- Sector: subdivisión de información dentro de una pista. Contienen bloques enteros.
- Un sector es la unidad mínima de información que puede leerse o escribirse en el
- Bloque: Unidad mínima de manejo por el S. De Ficheros. En un bloque caben n registros.
- Cada sector puede tener más de un bloque, se crean al formatear el disco.
- Cabezas de lectura/escritura: una por cada cara de cada disco. Optimizar su recorrido.

Esta estructura junto con el disco, y el motor del disco, forman la unidad de disco.

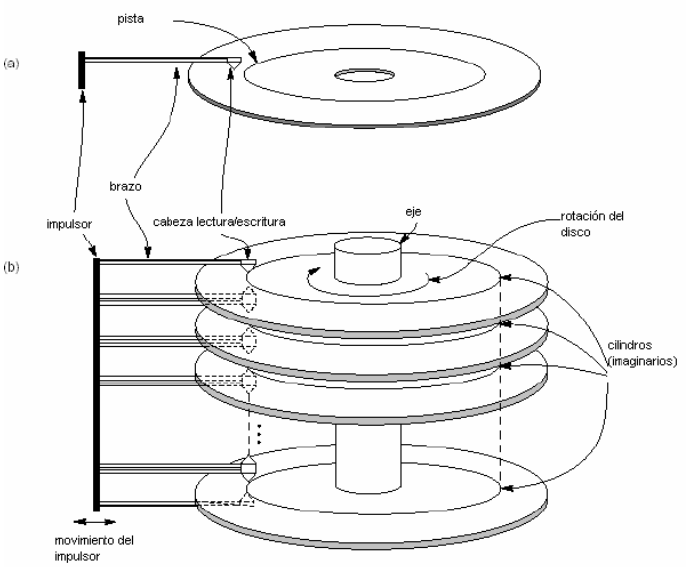


Funcionamiento de lectura/escritura en disco

El mecanismo de hardware que se encarga de escribir o leer los bloques es la cabeza de lectura/escritura del disco que se maneja gracias a un brazo mecánico.

Además, el controlador de disco controla y hace de interfaz entre ésta y el ordenador. El controlador acepta instrucciones de E/S de alto nivel y realiza las acciones oportunas para colocar el brazo y dar la orden de lectura o escritura. Una vez conocida la dirección del bloque a la que se desea acceder, el controlador de disco necesita un tiempo para colocar la cabeza sobre la pista correcta. Este tiempo se conoce como tiempo de búsqueda. Una vez situada la cabeza sobre la pista, es necesario que el disco gire hasta situar el inicio del bloque bajo la cabeza generándose otro retardo llamado retardo rotacional o latencia. Por último deberá transcurrir un tiempo adicional para la transferencia de los datos, conocido como tiempo de

transferencia de bloque. Para mejorar la eficiencia en la transferencia de bloques, se suele transferir varios bloques consecutivos de la misma pista, e incluso del mismo cilindro.



El tiempo necesario para localizar y transferir un bloque es la suma de: Tiempo de búsqueda + latencia + tiempo de transferencia de bloque

Que suele variar entre 12 y 60 mseg.

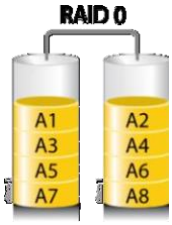
TECNOLOGIA RAID.

RAID: El acrónimo RAID, proviene del inglés Redundant Array of Independient Disks, es decir, conjunto de discos independientes redundantes, aunque originalmente la 'I' significaba Inexpensive (baratos). Este sistema de almacenamiento de datos se basa en el uso de múltiples discos para guardar información, repitiendo partes de información en diferentes discos, para asegurarse de que, en caso de que un disco deje de funcionar, ya sea por error, ruptura o mantenimiento, la información continué siendo accesible. En un principio un RAID, contaba con una sola unidad lógica que contenía a su vez los distintos discos duros. Así, el ordenador solo reconocía una unidad, que internamente se estructuraba siguiendo el sistema ya explicado. Este método se suele utilizar principalmente en servidores, que almacenan mucha información, que puede ser de mucha importancia, siendo imprescindible la fiabilidad de estos discos duros. Gracias al abaratamiento de los discos duros, y de los componentes informáticos en general, los RAID se encuentran más fácilmente en ordenadores personales avanzados, principalmente en aquellos destinados a tareas de almacenamiento, como edición de audio o vídeo.

A lo largo de la historia se ha evolucionado desde el llamado RAID 0, cuestionado por muchos como RAID, hasta el RAID 6:

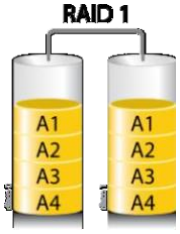
RAID 0:

El RAID 0, tal y como en un principio se concibió, no es redundante, pese a lo cual se continua llamando RAID. Por explicarlo de alguna manera, en el raid 0, la información se divide en partes y cada parte se guarda en un disco distinto, de manera que no es redundante, pero si uno de los discos cae, no se pierde toda la información. Aseguramos además de este modo un acceso más rápido, ya que podemos extraer información de varios discos duros a la vez, recuperando el archivo original de manera más rápida.



RAID 1:

En RAID 1, se crea una copia exactamente igual en otro disco duro llamada espejo. Este tipo de almacenamiento es realmente útil siempre que el rendimiento de la lectura sea mas importante que la capacidad, ya que tener toda la información por duplicado supone un gran uso de discos. Gracias a RAID 1, podemos leer de los dos discos simultáneamente, por lo que la velocidad de lectura también es muy alta. Gracias a la detección y corrección de errores de los discos duros actuales, RAID 1 no se usa prácticamente, ya que esta era la principal ventaja de RAID 1.

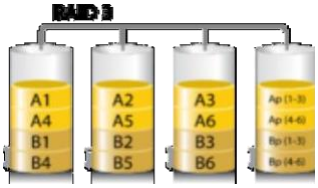


RAID2:

Este RAID prácticamente no se usa en ningún sitio. La implementación seria bit a bit, por lo que (gracias al código de Hamming que aportaría 7 bits para corrección de errores) tendríamos un total de 39 discos duros, uno para cada bit de cada palabra (de 32 bits).

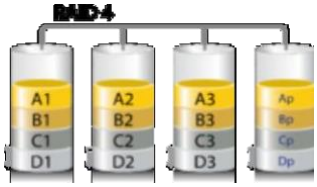
RAID 3

Este RAID también usa la separación por bits pero con un disco duro de comprobación de bits de paridad. Este sistema solo permite una petición simultánea, por lo que es poco efectivo y no se usa en la actualidad.

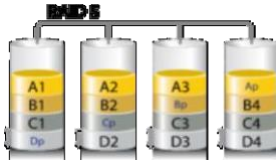


RAID 4:

RAID 4 viene siendo lo mismo que el 3 solo que separa por bloques en lugar de por bits.



RAID 5:



Una vez más, RAID 5 separa por bloques de bits, pero en esta ocasión, el bloque de paridad se reparte por todos los discos duros de manera que conseguimos poder realizar lecturas simultaneas, si bien no en el mismo disco duro, si en alguno de los otros.

En este RAID, no se accederá al bit de paridad hasta que obtengamos un CRC o error de redundancia cíclica, en el que entonces se comprobara la información gracias a dicho bit.

RAID 6:

Este RAID es una ampliación del 5, añadiendo un segundo bloque de bits de paridad.

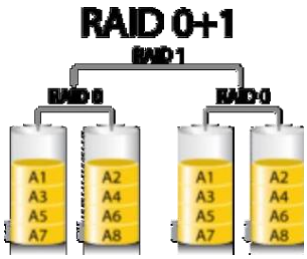
Aquí entramos en difíciles formulas acerca del código Reed-Solomon, del cual es este raid un caso especial.

RAID 5E y 6E:

Estos RAIDs son iguales a los anteriores pero añadiendo discos duros de reserva por si uno de los principales falla. Estos discos duros pueden ser hot spare, siempre conectado y preparado, o standby spare, en espera. No supone una mejora del rendimiento, pero si un menor tiempo de reconstrucción en caso de fallo.

Se suelen utilizar raids anidados para obtener una verdadera aplicación de estos tipos. Es decir, se usa raid 01 queriendo decir que primeramente se tiene un sistema raid 1 compuesto de dos raids 0.

Los más usados son el 01, el 10, el 30 y el 100



FICHEROS:

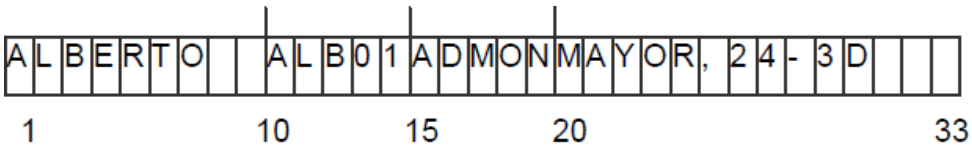
El fichero es la organización básica de almacenamiento en disco, y por tanto es la organización empleada para la implementación de una base de datos. Los ficheros a su vez son una secuencia de registros.

Los Registros son una colección de valores o elementos de datos relacionados donde los valores están formados por bits (uno o más), el conjunto de estos datos corresponde a un determinado campo de registro. Normalmente cada registro de las Bases de datos representan a una entidad, y cada valor de campo de ese registro a un atributo de la entidad. La lista de nombres asociados a cada campo y los tipos de datos que ellos almacenan constituyen el formato de registro.

Actualmente las bases de datos pueden almacenar datos de mayor tamaño adicionales al registro, como imágenes, audio, video,… lo que se conocen como BLOB (Binary Large Objects), estos tipos de datos se almacenan aparte de su registro y en distintas áreas del disco, para relacionarlos en los registros se incluye un puntero hacia ellos.

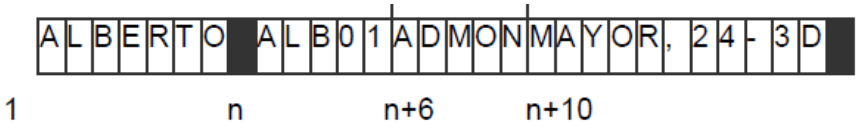
Los registros pueden ser de diferentes maneras según su registro:

- Longitud fija: La longitud del registro es conocida y la misma para todos, además cada campo ocupa una posición conocida dentro del registro. Para acceder a cada uno de ellos únicamente es necesario conocer su posición de inicio y fin en el registro.



Campo	*Posición inicio*	*Posición fin*
NOMBRE	1	9
CÓDIGO	10	14
DEPARTAMENTO	15	19
DIRECCIÓN	20	33

- Longitud variable: Si la longitud de los registros es diferente hablaremos de que son registros variables. Para delimitar el final de un campo utilizaremos caracteres especiales.



Campo	*Posición inicio*	*Posición fin*
NOMBRE	1	n-1
Carácter fin ■	N	n
CÓDIGO	n+1	n+5
DEPARTAMENTO	n+6	n+10
DIRECCIÓN	n+11	m-1
Carácter fin ■	M	m

- Registros con campos adicionales: Estos son los BLOP.

Organización e índices.

El diseño a la hora de la organización es algo que para el usuario debe ser completamente transparente, ya que es algo que el programador será el encargado de su organización dependiendo del SGBD utilizado, y en algunas ocasiones hasta interactuando con el Sistema Operativos cuando sea necesario.

Los pasos a seguir para el diseño de un SGBD son:

– Analizar transacciones.
– Seleccionar organización de archivos.
– Seleccionar índices.

Estimar el espacio en disco necesario

Montículo:

Es el método más simple de organización ya que todo registro nuevo se coloca en el último sitio, y en el primero se guarda la posición del último, con lo que para realizar una nueva inserción solo tendremos que mirar donde está el último elemento e insertarlo.

Por el contrario, la búsqueda resulta ardua, ya que tenemos que mirar todos los elementos para comprobar si el elemento buscado esta en el montículo.

En el borrado, ya que se encuentra implícita una búsqueda, tenemos el mismo problema que antes, aunque una vez encontrado el elemento, el propio borrado es muy sencillo, aunque desperdiciaremos mucho espacio en disco ya que el hueco vacío no será rellenado.

La actualización presenta el mismo problema de la búsqueda, y la misma facilidad en la propia acción que el borrado.

Ficheros ordenados:

Estos ficheros tienen un orden físico dentro del disco, gracias a un campo de ordenación. Esto conlleva ciertas ventajas e inconvenientes.

En la inserción, el trabajo se complica mucho, ya que para insertar un elemento deberemos desplazar todos los posteriores.

La búsqueda será infinitamente más eficiente siempre que busquemos por el campo de ordenación, si no es así nos encontramos con un problema de complejidad lineal, igual que en el montículo.

Igualmente encontramos el mismo problema de búsqueda tanto en el borrado como en la actualización matizando que si se realiza un borrado, tendremos el mismo problema de desperdicio de espacio que en el montículo.
Si al realizar una actualización, el campo a modificar es el campo de ordenación, sera necesario un borrado y una inserción, con lo costosos que pueden llegar a ser estos procesos.

-Direccionamiento calculado (hash):
Los ficheros de direccionamiento calculado tienen la gran ventaja de que, ya que cada elemento se inserta en un sitio especifico, gracias a las funciones de dispersión y redispersión, la búsqueda, la inserción y el borrado se hacen fácilmente, aunque para ello se debe reservar mucha memoria que, quizás, nunca llegue a usarse.

ÍNDICES

El índice de una base de datos es una estructura de datos que mejora la velocidad de las operaciones, permitiendo un rápido acceso a los registros de una tabla. Al aumentar drásticamente la velocidad de acceso, se suelen usar sobre aquellos campos sobre los cuales se hagan frecuentes búsquedas.

El índice tiene un funcionamiento similar al índice de un libro, guardando parejas de elementos: el elemento que se desea indexar (campo de indexación) y su posición en la base de datos (puntero al bloque de disco). Para buscar un elemento que esté indexado, sólo hay que buscar en el índice dicho elemento para, una vez encontrado, devolver el registro que se encuentre en la posición marcada por el índice.

Los índices pueden ser creados usando una o más columnas, proporcionando la base tanto para búsquedas rápidas al azar como de un ordenado acceso a registros eficiente.
Los índices son construidos sobre árboles B, B+, B* o sobre una mezcla de ellos, funciones de cálculo u otros métodos.

El espacio en disco requerido para almacenar el índice es típicamente menor que el espacio de almacenamiento de la tabla (puesto que los índices generalmente contienen solamente los campos clave de acuerdo con los que la tabla será ordenada, y excluyen el resto de los detalles

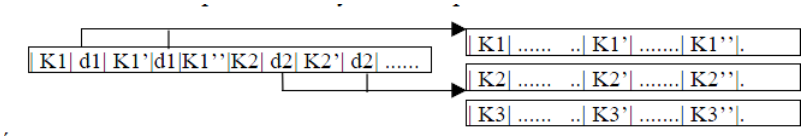
de la tabla), lo que da la posibilidad de almacenar en memoria los índices de tablas que no cabrían en ella. En una base de datos relacional un índice es una copia de parte de una tabla.

Tipos de índices según su estructura

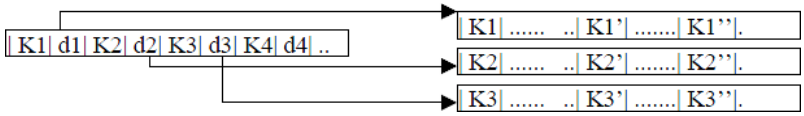
- Índice denso o Exahustivo (dense): Aparece una entrada en el índice por cada valor de la clave de búsqueda. Para buscar un registro, se ubica en el primer registro con el valor del campo clave buscado y se procesa secuencialmente (puede haber más de un registro con el valor de la clave de búsqueda, sólo si coincide con una clave candidata habrá un único registro).

Ventaja: Rapidez en la consulta.

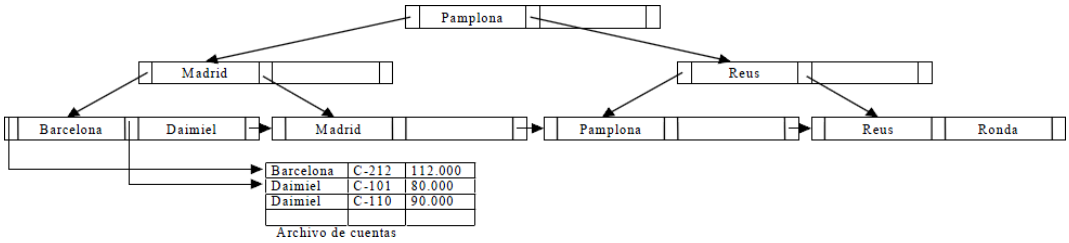
Inconvenientes: Espacio usado y menor rapidez en las actualizaciones.



- Índice disperso (sparse): No aparecen todos los valores de la clave de búsqueda. El acceso en lectura será en general más lento porque probablemente en el índice no se encuentre el valor del campo clave (hay que acceder al valor anterior y procesar secuencialmente en el orden de la clave). Esto se puede usar sólo si los registros están ordenados en esta clave de búsqueda.

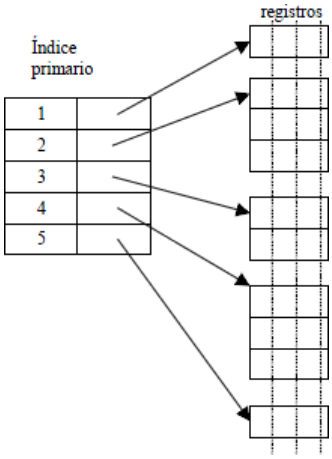


- Índices multinivel: Aparecen como consecuencia de los índices de gran tamaño que no pueden almacenarse en memoria y que implican un gran tiempo de acceso (incluso efectuando una búsqueda binaria en el fichero índice). Se crea un índice disperso sobre el índice denso. Se puede repetir el proceso tantas veces como sea necesario, creando una capa nueva con un índice disperso sobre el último índice creado. Un caso particular es el índice en árbol equilibrado B+, que se ve después.



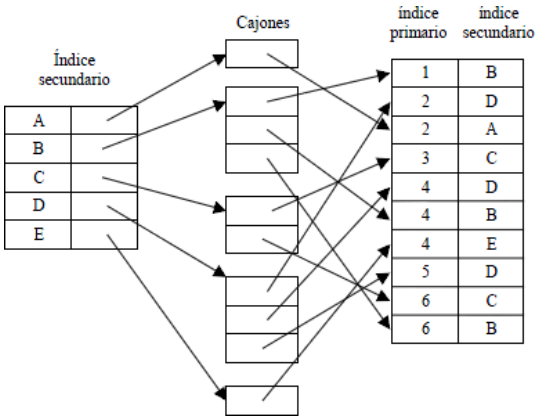
Tipos de índices según el uso y el orden de sus registros

- Índices primarios
  - o Primario si asume una ordenación de registros en el disco por un campo o varios.
  - o Estos componen la clave de búsqueda
  - o Puede haber varios registros con el mismo valor de la clave (NO clave candidata).
  - o Los registros del fichero índice tienen 2 campos:
    - El valor de la clave
    - La dirección del bloque de datos que contiene el primer registro de datos con esa clave.



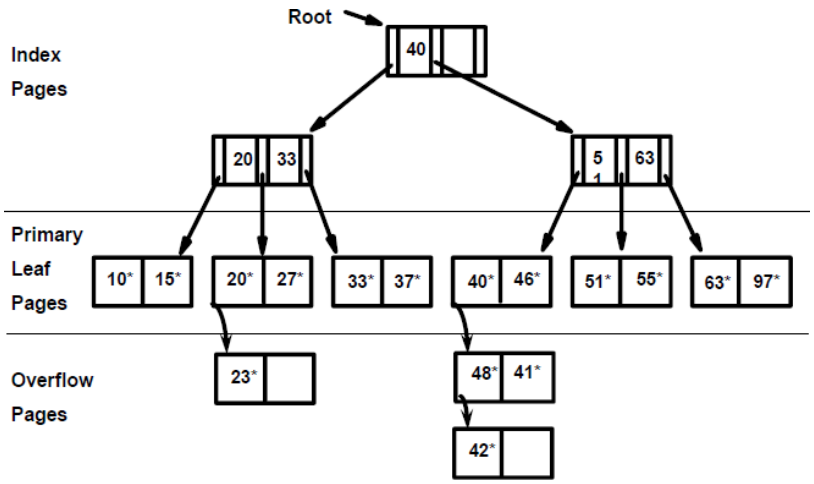
- Índices secundarios
  - o Para acceder a los registros por otros campos.
  - o Los registros no están ordenados por estos índices.

- Deben ser índices densos, con entradas a todos los registros, usando doble indirección: cada valor del índice apunta a un cajón (bucket) con la dirección de todos los registros con ese valor de índice.



- ISAM: Es un método para almacenar información a la que se pueda acceder rápidamente. Este método fue desarrollado por IBM y Actualmente forma parte del almacenamiento básico de muchos sistemas de BD.

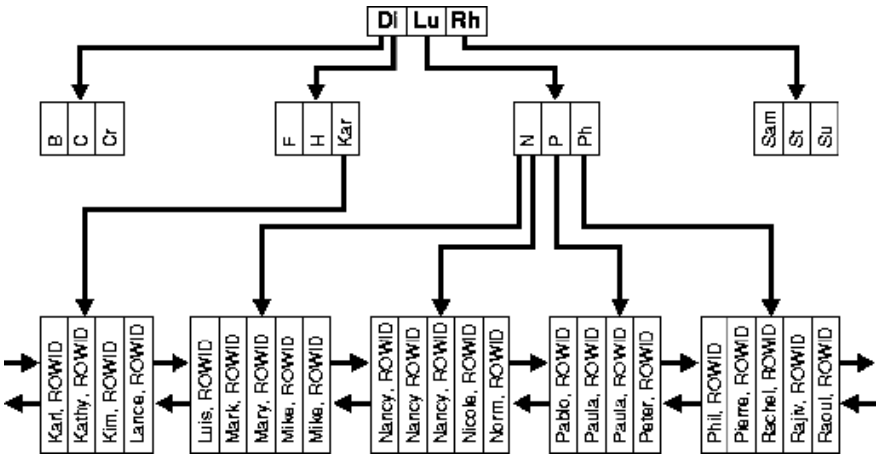
- Este sistema es más versátil que las tablas HASH
- En un sistema ISAM la información está organizada en registros de longitud fija y se almacenan secuencialmente para acelerar el acceso a ellos, aunque este se degrada a medida que se actualiza.
- El avance clave que posee ISAM es que los índices son pequeños y pueden ser buscados rápidamente, permitiendo a la base de datos acceder sólo a los registros que necesita, además que las modificaciones adicionales a la información no requieren cambios a otra información, sólo a la tabla y los índices.



- Arboles B+: Un árbol B es un tipo de estructura de datos ordenados de manera que permiten la inserción y el borrado eficiente de los elementos.

o Un árbol B+ es como un árbol binario, el cual almacena toda la información en sus nodos hojas, los nodos internos sólo contienen claves y punteros, además los nodos hojas se encuentran unidos entre sí como una lista enlazada para permitir la búsqueda secuencial.

o Es más eficiente que las tablas HASH e ISAM.



- Clúster: o Tablas agrupadas, en este caso nos referimos a un clúster como un conjunto de tablas almacenadas físicamente juntas, estas comparten columnas comunes y muchas veces se usan de forma conjunta.

o Estos son eficientes en consultas que utilizan frecuentemente datos relacionados.

nomvend	nombrecomer	telefono	calle	Ciudad	provincia	numvend	numpedido	fecha
AGAPITO LAFUENTE DEL CORRAL	MECEMSA	96-5782401	Avda. Valencia 3205	ALICANTE	ALICANTE	1	1	05/05/1992
							2	11/10/1992
							5	22/10/1992
LUCIANO BLAZQUEZ VAZQUEZ	HARW S.A.	96-3232321	GENERAL LACY, 15 2 B	ALICANTE	ALICANTE	2	3	15/10/1992
							4	16/10/1992
JUANITO REINA PRINCESA	LA DEAQUI	96-5363636	S. FRANCISCO DE ASIS, 10 1	GIJON	ASTURIAS	5	6	22/08/1995
JUAN MARTINEZ GARCIA	HARW S.A.	3334455	OSCAR, 5	VALENCIA	VALENCIA	8002	7	02/10/1992

TABLA VENDEDOR **TABLA PEDIDO**

CLAVE DE CLUSTER

- Índices de mapa de bits: Estos se usan cuando un dominio contiene un número relativamente bajo de valores posibles, se construyen a partir de vectores de bits, donde cada bit representa un registro que contiene el valor especificado.

o Una de sus características es que los índices son más compactos, por lo que es muy recomendable para grandes volúmenes de datos.

Datos

21	Pedro		Director
22	Juan		Ventas
23	Manuel		Ventas
24	Arturo		Almacén
25	Pedro		Vental
26	José		Almacén
27	Ana		Director

Índice

Director	1000001
Ventas	0110100
Almacén	0001010

Seleccionar índices:

Seleccionar los índices sirve para mejorar las prestaciones del sistema. Tenemos que tener en cuenta que actualizar o insertar una tupla genera una operación en el índice, esto incrementa el espacio en disco y las consultas pueden verse ralentizadas. En relaciones pequeñas puede ser mejor una búsqueda binaria.

En las claves ajenas pueden tener su propio índice para ayudar a la concatenación entre las tuplas e incrementar su eficiencia.

El principal objetivo de la selección de índices es evitar los atributos frecuentemente actualizados o cadenas de caracteres de gran longitud.

IMPLEMENTACION DE BASES DE DATOS RELACIONALES

El diseño de una base de datos consiste en definir la estructura de los datos que debe tener un sistema de información determinado. Para ello se suelen seguir por regla general unas fases en el proceso de diseño, definiendo para ello el modelo conceptual, el lógico y el físico.

En el diseño conceptual se hace una descripción de alto nivel de la estructura de la base de datos, independientemente del SGBD (Sistema Gestor de Bases de Datos) que se vaya a utilizar para manipularla. Su objetivo es describir el contenido de información de la base de datos y no las estructuras de almacenamiento que se necesitarán para manejar dicha información.

El diseño lógico parte del resultado del diseño conceptual y da como resultado una descripción de la estructura de la base de datos en términos de las estructuras de datos que puede procesar un tipo de SGBD. El diseño lógico depende del tipo de SGBD que se vaya a utilizar, se adapta a la tecnología que se debe emplear, pero no depende del producto concreto. En el caso de bases de datos convencionales relacionales (basadas en SQL para entendernos), el diseño lógico consiste en definir las tablas que existirán, las relaciones entre ellas, normalizarlas, etc...

El diseño físico parte del lógico y da como resultado una descripción de la implementación de una base de datos en memoria secundaria: las estructuras de almacenamiento y los métodos utilizados para tener un acceso eficiente a los datos. Aquí el objetivo es conseguir una mayor eficiencia, y se tienen en cuenta aspectos concretos del SGBD sobre el que se vaya a implementar. Por regla general esto es transparente para el usuario, aunque conocer cómo se implementa ayuda a optimizar el rendimiento y la escalabilidad del sistema.

El modelo relacional

En el modelo relacional las dos capas de diseño conceptual y lógico, se parecen mucho. Generalmente se implementan mediante diagramas de Entidad/Relación (modelo conceptual)

y tablas y relaciones entre éstas (modelo lógico). Este es el modelo utilizado por los sistemas gestores de datos más habituales (SQL Server, Oracle, MySQL...).

***Nota**: A las bases de datos relaciones se les denomina así porque almacenan los datos en forma de "Relaciones" o listas de datos, es decir, en lo que llamamos habitualmente "Tablas". Se piensa muchas veces que el nombre viene porque además las tablas se relacionan entre sí utilizando claves externas. No es así, y es un concepto que hay que tener claro. (Tabla = Relación).*

El modelo relacional de bases de datos se rige por algunas normas sencillas:

Todos los datos se representan en forma de tablas (también llamadas "relaciones", ver nota anterior). Incluso los resultados de consultar otras tablas. La tabla es además la unidad de almacenamiento principal.

Las tablas están compuestas por filas (o registros) y columnas (o campos) que almacenan cada uno de los registros (la información sobre una entidad concreta, considerados una unidad).

Las filas y las columnas, en principio, carecen de orden a la hora de ser almacenadas. Aunque en la implementación del diseño físico de cada SGBD esto no suele ser así. Por ejemplo, en SQL Server si añadimos una clave de tipo "Clustered" a una tabla haremos que los datos se ordenen físicamente por el campo correspondiente.

El orden de las columnas lo determina cada consulta (que se realizan usando SQL).

Cada tabla debe poseer una clave primaria, esto es, un identificador único de cada registro compuesto por una o más columnas.

Para establecer una relación entre dos tablas es necesario incluir, en forma de columna, en una de ellas la clave primaria de la otra. A esta columna se le llama clave externa. Ambos conceptos de clave son extremadamente importantes en el diseño de bases de datos.

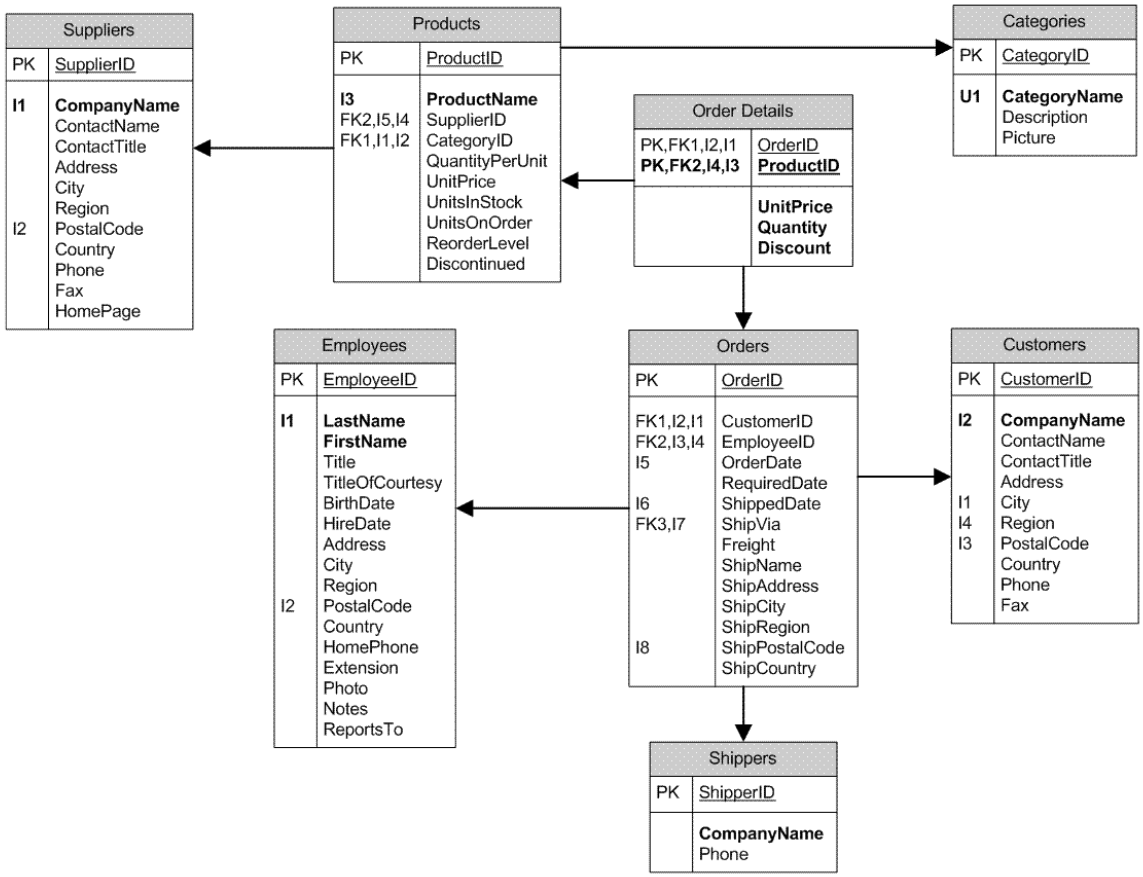
Basándose en estos principios se diseñan las diferentes bases de datos relacionales, definiendo un diseño conceptual y un diseño lógico, que luego se implementa en el diseño físico usando para ello el gestor de bases de datos de nuestra elección (por ejemplo SQL Server).

Por ejemplo, consideremos la conocida base de datos Northwind de Microsoft.

Esta base de datos representa un sistema sencillo de gestión de pedidos para una empresa ficticia. Existen conceptos que hay que manejar como: proveedores, empleados, clientes, empresas de transporte, regiones geográficas, y por supuesto pedidos y productos.

El diseño conceptual de la base de datos para manejar toda esta información se puede ver en la siguiente figura, denominada diagrama Entidad/Relación o simplemente diagrama E-R:

Northwind_EF



Como vemos existen tablas para representar cada una de estas entidades del mundo real: Proveedores (Suppliers), Productos, Categorías de productos, Empleados, Clientes, Transportistas (Shippers), y Pedidos (Orders) con sus correspondientes líneas de detalle (Order Details).

Además están relacionadas entre ellas de modo que, por ejemplo, un producto pertenece a una determinada categoría (se relacionan por el campo CategoryID) y un proveedor (SupplierID), y lo mismo con las demás tablas.

Cada tabla posee una serie de campos que representan valores que queremos almacenar para cada entidad. Por ejemplo, un producto posee los siguientes atributos que se traducen en los campos correspondientes para almacenar su información: Nombre (ProductName), Proveedor (SupplierID, que identifica al proveedor), Categoría a la que pertenece (CategoryID), Cantidad de producto por cada unidad a la venta (QuantityPerUnit), Precio unitario (UnitPrice), Unidades que quedan en stock (UnitsInStock), Unidades de ese producto que están actualmente en pedidos (UnitsOnOrder), qué cantidad debe haber para que se vuelva a solicitar más producto al proveedor (ReorderLevel) y si está descatalogado o no (Discontinued).

Los campos marcados con "PK" indican aquellos que son claves primarias, es decir, que identifican de manera única a cada entidad. Por ejemplo, ProductID es el identificador único del producto, que será por regla general un número entero que se va incrementando cada vez que introducimos un nuevo producto (1, 2, 3, etc..).

Los campos marcados como "FK" son claves foráneas o claves externas. Indican campos que van a almacenar claves primarias de otras tablas de modo que se puedan relacionar con la tabla actual. Por ejemplo, en la tabla de productos el campo CategoryID está marcado como "FK" porque en él se guardará el identificador único de la categoría asociada al producto actual. En otras palabras: ese campo almacenará el valor de la clave primaria (PK) de la tabla de categorías que identifica a la categoría en la que está ese producto.

Los campos marcados con indicadores que empiezan por "I" (ej: "I1") se refieren a índices. Los índices generan información adicional para facilitar la localización más rápida de registros basándose en esos campos. Por ejemplo, en la tabla de empleados (Employees) existe un índice "I1" del que forman parte los campos Nombre y Apellidos (en negrita además porque serán también valores únicos) y que indica que se va a facilitar la locación de los clientes mediante esos datos. También tiene otro índice "I2" en el campo del código postal para localizar más rápidamente a todos los clientes de una determinada zona.

Los campos marcados con indicadores que empiezan con "U" (por ejemplo U1) se refieren a campo que deben ser únicos. Por ejemplo, en la tabla de categorías el nombre de ésta (CategoryName) debe ser único, es decir, no puede haber -lógicamente- dos categorías con el mismo nombre.

Como vemos, un diseño conceptual no es más que una representación formal y acotada de entidades que existen en el mundo real, así como de sus restricciones, y que están relacionadas con el dominio del problema que queremos resolver.

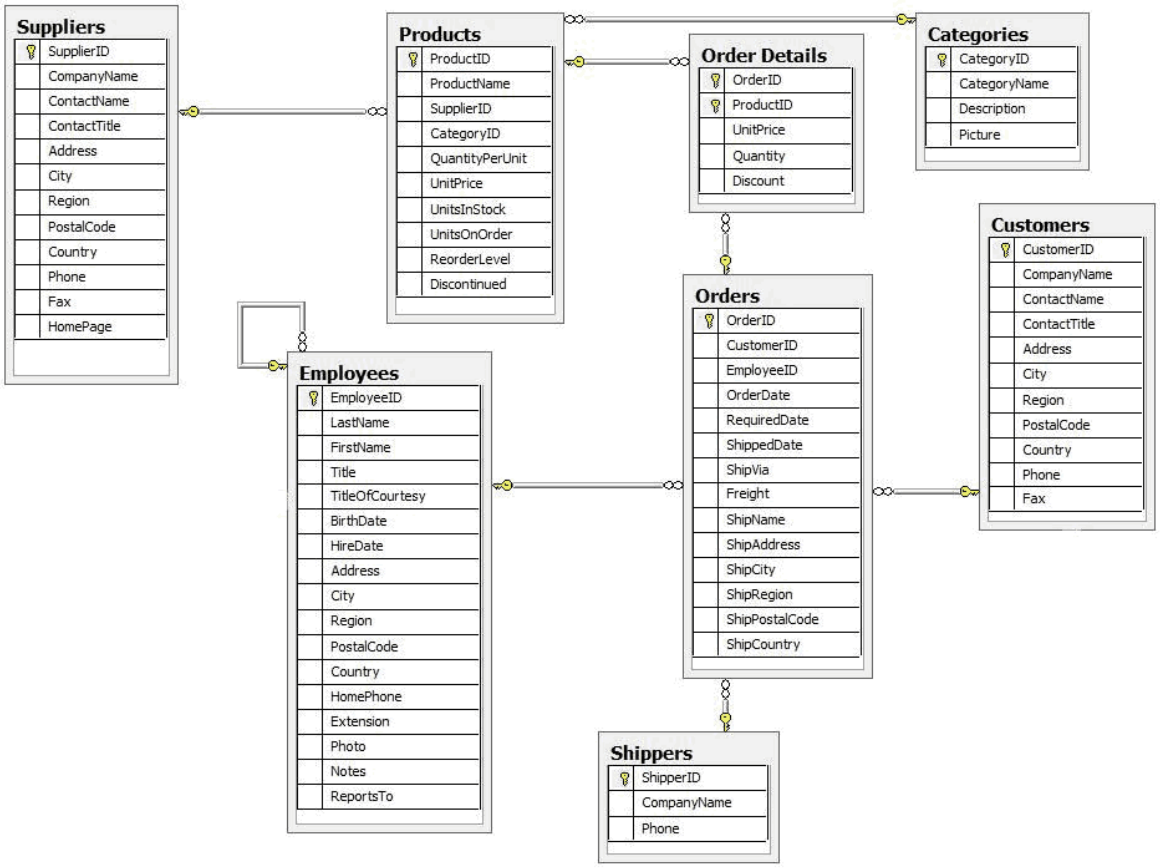
Modelo lógico

Una vez tenemos claro el modelo E-R debemos traducirlo a un modelo lógico directamente en el propio sistema gestor de bases de datos (Oracle, MySQL, SQL Server...). Si hemos utilizado alguna herramienta profesional para crear el diagrama E-R, seguramente podremos generar automáticamente las instrucciones necesarias para crear la base de datos.

La mayoría de los generadores de diagramas E-R (por ejemplo Microsoft Visio) ofrecen la capacidad de exportar el modelo directamente a los SGBD más populares.

Entonces, todo este modelo conceptual se traduce en un modelo lógico que trasladaremos a la base de datos concreta que estemos utilizando y que generalmente será muy parecido. Por ejemplo, este es el mismo modelo anterior, mostrado ya como tablas en un diagrama de SQL Server:

Northwind_Tablas



En este caso hemos creado cada tabla, una a una, siguiendo lo identificado en el diagrama E-R y estableciendo índices y demás elementos según las indicaciones de cada uno de los campos. Además hemos decidido el mejor tipo de datos que podemos aplicar a cada campo (texto, números, fechas... que se almacenan para cada registro).

Su representación gráfica en la base de datos es muy similar, sin embargo el modelo físico (cómo se almacena esto físicamente), puede variar mucho de un SGBD a otro y según la configuración que le demos.
En resumen

Según Thomas H. Grayson, un buen diseño de base de datos debe poseer siempre las siguientes cualidades, aunque algunas pueden llegar a ser contradictorias entre sí:

Reflejar la estructura del problema en el mundo real.
Ser capaz de representar todos los datos esperados, incluso con el paso del tiempo.
Evitar el almacenamiento de información redundante.
Proporcionar un acceso eficaz a los datos.

Mantener la integridad de los datos a lo largo del tiempo.

Ser claro, coherente y de fácil comprensión.

Como hemos visto el diseño de una base de datos parte de un problema real que queremos resolver y se traduce en una serie de modelos, conceptual, lógico y físico, que debemos implementar.

El primero, el diseño conceptual, es el que más tiempo nos va a llevar pues debemos pensar muy bien cómo vamos a representar las entidades del mundo real que queremos representar, qué datos almacenaremos, cómo los relacionaremos entre sí, etc...

El diseño lógico es mucho más sencillo puesto que no es más que pasar el diseño anterior a una base de datos concreta. De hecho muchas herramientas profesionales nos ofrecen la generación automática del modelo, por lo que suele ser muy rápido.

El diseño físico por regla general recae en la propia base de datos, a partir del diseño lógico, aunque si dominamos bien esa parte elegiremos cuidadosamente índices, restricciones o particiones así como configuraciones para determinar cómo se almacenará físicamente esa información, en qué orden, cómo se repartirá físicamente en el almacenamiento, etc.

UNIDAD 2: SISTEMAS DE GESTIÓN DE BASES DE DATOS

También llamado DBMS (Data Base Managment System) como una colección de datos relacionas entre sí, estructurados y organizados, y un conjunto de programas que acceden y gestionan esos datos. La colección de esos datos se denomina Base de Datos o BD, (DB Data Base).

Como objetivos principales de los SGBD constan los siguientes:

-**Independencia de datos**:

oLos programas de aplicación deben verse afectados lo menos posible por cambios efectuados en datos que no usan. -Integridad de los datos: oLa información almacenada en la BD debe cumplir ciertos requisitos de calidad, para ello hace falta, en el momento de introducirse los valores de los datos, que éstos se almacenen debidamente, y que posteriormente no se deterioren.

-Seguridad de los datos:

oA la información almacenada en la BD sólo pueden acceder las personas autorizadas y de la forma autorizada.

Es entonces deber del SGBD ofrecer los servicios típicos:

-Creación y definición de la base de datos.

-Manipulación de los datos

-Acceso a los datos.

-Mantener la integridad y consistencia de los datos.

-Mecanismos de copias de respaldo y de recuperación

Arquitectura

Fue en 1975, en el comité ANSI-SPARC (American National Standard Institute – Standards Planning and Requeriments Committee), cuando se propuso una arquitectura de tres niveles para los DBMS cuyo objetivo principal era separar la BD física de los programa de aplicación.

Tres niveles

Los tres niveles son:

-Nivel interno o físico

oEs el más cercano al almacenamiento físico, es decir, tal y como están almacenados en el ordenador. Describe la estructura física de la BD mediante un esquema interno. Este esquema se especifica con un modelo físico y describe los detalles de cómo se almacenan físicamente los datos: los archivos que contienen la información, su organización, los métodos de acceso a los registros, los tipos de registros, la longitud, los campos que los componen.

-Nivel externo o de visión

oEs el más cercano a los usuarios, es decir, es donde se describen varios esquemas externos o vistas de usuarios. Cada esquema describe la parte de la BD que

interesa a un grupo de usuarios en este nivel se representa la visión individual de un usuario o de un grupo de usuarios.

-Nivel conceptual

oDescribe la estructura de toda la BD para un grupo de usuarios mediante un esquema conceptual. Este esquema describe las entidades, atributos, relaciones, operaciones de los usuarios y restricciones, ocultando los detalles de las estructuras físicas de almacenamiento. Representa la información contenida en la BD. En la Figura siguiente se representan los niveles de abstracción de la arquitectura de un DBMS.
Esta división de niveles se mantuvo hasta que se vio que no había una descripción general del campo conceptual por lo que se decidió optar por añadir una nueva capa.

Cuatro niveles

Los cuatro niveles ahora serían:

-Nivel conceptual

oVisión desde un punto de vista organizativo, independiente del SGBD que se utilice, e incluso de la utilización o no de sistemas de bases de datos. En este nivel se describe la información de la organización (objetos y relaciones) desde un punto de vista no informático.

-Nivel lógico

oVisión expresada en términos de un SGBD concreto, o mejor dicho, de un modelo de datos soportado por un SGBD. En este esquema lógico se representan las entidades y relaciones de acuerdo a las características de dicho modelo sin entrar todavía en detalles de representación física.

-Nivel interno

oDescripción de la representación en la memoria externa del ordenador de los datos del esquema lógico, sus interrelaciones y los instrumentos para acceder a ellos. -

Niveles externos

oCada uno de ellos describe los datos y relaciones entre ellos de interés para una aplicación dada, estos esquemas pueden verse como subconjuntos de Modelo Lógico de la BD

Transformaciones

El SGBD debe de transformar cualquier petición de usuario (esquema externo) a una petición expresada en términos de esquema conceptual, para finalmente ser una petición expresada en el esquema interno que se procesará sobre la BD almacenada. El proceso de transformar peticiones y resultados de un nivel a otro se denomina correspondencia o transformación, el SGBD es capaz de interpretar una solicitud de datos y realiza los siguientes pasos:

-El usuario solicita unos datos y crea una consulta.

-El SGBD verifica y acepta el esquema externo para ese usuario.

-Transforma la solicitud al esquema conceptual.

-Verifica y acepta el esquema conceptual.

-Transforma la solicitud al esquema físico o interno.

-Selecciona la o las tablas implicadas en la consulta y ejecuta la consulta.

-Transforma del esquema interno al conceptual, y del conceptual al externo.

-Finalmente, el usuario ve los datos solicitados.

Independencias

Con la arquitectura a tres niveles también se introduce el concepto de independencia de datos, se definen dos tipos de independencia:

-Independencia lógica oLa capacidad de modificar el esquema conceptual sin tener que alterar los esquemas externos ni los programas de aplicación.

-Independencia física oLa capacidad de modificar el esquema interno sin tener que alterar ni el esquema conceptual, ni los externos.

Componentes

Un SGBD debe dar las funciones antes comentadas al usuario y para ofrecerlas hace uso de una serie de herramientas generales las cuales servirán para mantener el SGBD

.Lenguajes de los SGBD

Los lenguajes son herramientas que permiten a todo tipo de usuarios realizar ciertas operaciones sobre el SGBD. Fundamentalmente hay dos tipos de lenguajes:

-Lenguajes de definición de datos (DDL)

oSe utilizan para especificar el esquema de la BD, las vistas de los usuarios y las estructuras de almacenamiento. Es el que define el esquema conceptual y el esquema interno. Lo utilizan los diseñadores y los administradores de la BD.

-Lenguajes de manipulación de datos (DML)

oSe utilizan para leer y actualizar los datos de la BD. Es el utilizado por los usuarios para realizar consultas, inserciones, eliminaciones y modificaciones. Prácticamente hoy en día todos los lenguajes ya cuentan con la mayoría de estructuras de control (bucles, if-then-else, etc.) por lo que no habría que preocuparse por ello. Aquellos que ya incorporan esas estructuras son llamados lenguajes autosuficientes, mientras que los que no cuentan con ellas son llamados lenguajes huésped. Generalmente se suele usar SQL en los SGBD que ya contiene ambos lenguajes incorporados.

Diccionario de datos

Esta herramienta nos ofrece información adicional sobre la definición de datos (datos sobre datos), también conocida como metadatos. Proporciona información acerca de:

-La estructura lógica y física de la BD.

-Las definiciones de todos los objetos de la BD: tablas, vistas, índices ...

-El espacio asignado y utilizado por los objetos.

-Los valores por defecto de las columnas de las tablas.

-Información acerca de las restricciones de integridad

-Los privilegios y roles otorgados a los usuarios.

Además debe cumplir con las siguientes características:

-Debe soportar las descripciones de los modelos conceptual, lógico, interno y externo de la BD.

-Debe ser integrado dentro del SGBD

-Debe apoyar la transferencia eficiente de la información al SGBD.

-Debe reflejar los cambios en la descripción de la BD.

-Debe estar almacenado en un medio de almacenamiento con acceso directo para la fácil recuperación de información.

Seguridad e integridad

El SGBD debe vigilar las peticiones del usuario y rechazar todo intento de violar las restricciones de seguridad y de integridad definidas por el DBA (Administrador). Estas tareas pueden realizarse durante el tiempo de compilación, de ejecución o entre ambos. El sistema de seguridad debe garantizar:

-La protección de los datos contra accesos no autorizados, tanto intencionados como accidentales. Debe controlar que solo los usuarios autorizados accedan a la BD.

-Ser capaz de recuperar la BD llevándola a un estado consistente en caso de ocurrir algún suceso que la dañe.

-Ofrecer mecanismos para conservar la consistencia de los datos en el caso de que varios usuarios actualicen la BD de forma concurrente.

Transacciones y control de la concurrencia

Los SGBDs son sistemas concurrentes, i.e., admiten la ejecución concurrente de consultas. Ejemplo: Sistema de venta de billetes de avión.

Por tanto, es necesario:

Modelo de procesos concurrentes para admitir operaciones concurrentes que preserven la integridad de los datos.

Conceptos básicos

Transacción

Una transacción es una unidad de la ejecución de un programa. Puede consistir en varias operaciones de acceso a la base de datos. Está delimitada por constructoras como begin-transaction y end-transaction.

Propiedades ACID

Atomicidad

Es la propiedad de las transacciones que permite observarlas como operaciones atómicas: ocurren totalmente o no ocurren.

Casos a considerar:

- Consultas unitarias. Incluso para consultas unitarias hay que preservar la atomicidad: en un sistema operativo de tiempo compartido, la ejecución concurrente de dos consultas SQL puede ser incorrecta si no se toman las precauciones adecuadas.
- Operación abortada. Por ejemplo, debido a una división por cero; por privilegios de acceso; o para evitar bloqueos

Consistencia

La ejecución aislada de la transacción conserva la consistencia de la base de datos.

Aislamiento

Para cada par de transacciones que puedan ejecutarse concurrentemente Ti y Tj, se cumple que para los efectos de Ti:

- Tj ha terminado antes de que comience Ti
- Tj ha comenzado después de que termine

Ti Las transacciones son independientes

entre sí.

Niveles de aislamiento

Se puede ajustar el nivel de aislamiento entre las transacciones y determinar para una transacción el grado de aceptación de datos inconsistentes.

A mayor grado de aislamiento, mayor precisión, pero a costa de menor concurrencia.

El nivel de aislamiento para una sesión SQL establece el comportamiento de los bloqueos para las instrucciones SQL.

Niveles de aislamiento:

- Lectura no comprometida. Menor nivel. Asegura que no se lean datos corruptos físicamente.
- Lectura comprometida. Sólo se permiten lecturas de datos comprometidos.
- Lectura repetible. Las lecturas repetidas de la misma fila para la misma transacción dan los mismos resultados.
- Secuenciable. Mayor nivel de aislamiento. Las transacciones se aíslan completamente.

Comportamiento concurrente de las transacciones.

- Lectura sucia. Lectura de datos no comprometidos. (Retrocesos)
- Lectura no repetible. Se obtienen resultados inconsistentes en lecturas repetidas.
- Lectura fantasma. Una lectura de una fila que no existía cuando se inició la transacción.

	Comportamiento permitido		
Nivel de aislamiento	**Lectura sucia**	**Lectura no repetible**	**Lectura fantasma**
Lectura no comprometida	Sí	Sí	Sí
Lectura comprometida	No	Sí	Sí
Lectura repetible	No	No	Sí
Secuenciable	No	No	No

SQL Server permite todos estos niveles, Oracle sólo permite la lectura comprometida y secuenciable. Los niveles se pueden establecer en ambos SGBD para cada transacción.

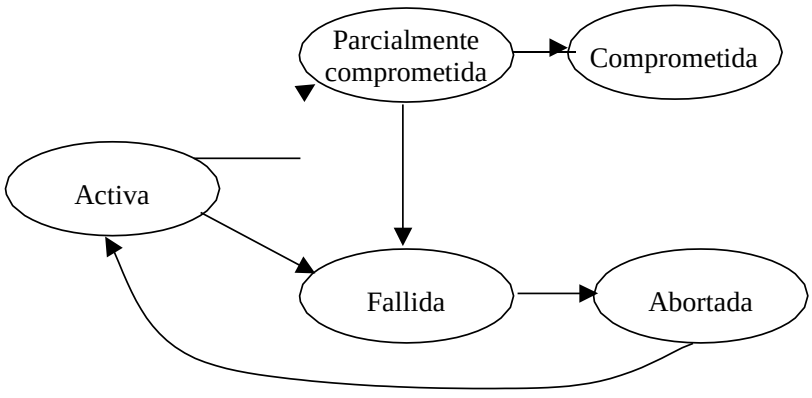
Durabilidad

El sistema gestor de bases de datos asegura que perduren los cambios realizados por una transacción que termina con éxito.

Estados de una transacción

- Activa: Durante su ejecución
- Parcialmente comprometida: Después de ejecutar su última instrucción.
- Fallida: Imposible de continuar su ejecución normal.
- Abortada: Transacción retrocedida y base de datos restaurada al estado anterior a su ejecución. Se puede reiniciar o cancelar.

Diagrama de estados de una transacción:



Retroceso de transacciones: Automáticos y programados.

Programación de transacciones

Tipos de transacciones: implícitas (modo de autocompromiso) y explícitas (delimitadas). Transacciones explícitas:

Oracle	SQL Server
Instrucción 1 ... SAVEPOINT *sp* ... ROLLBACK [TO SAVEPOINT *sp*] ... Instrucción n COMMIT [WORK]	BEGIN TRAN[SACTION] Instrucción 1 ... SAVE TRAN[SACTION] *sp* ... ROLLBACK [TRAN[SACTION] *sp*] ... Instrucción n COMMIT [TRAN[SACTION]]

Ejemplo (SQL Server):

Incremento de un 1% de las comisiones 15% y 16% de la tabla de comisiones *roysched*. Si no existen estos porcentajes entonces no se ejecutará la instrucción de actualización. En este ejemplo se deben incrementar ambos; si uno de ellos no existe, se debe dejar sin modificar.

```
BEGIN TRAN actualiza_comisiones    -- Inicio de la transacción USE
   pubs
IF EXISTS (SELECT titles.title, roysched.royalty FROM
                titles, roysched
                WHERE titles.title_id=roysched.title_id AND
                        roysched.royalty=16)
      UPDATE roysched SET royalty=17 WHERE royalty=16
   ELSE
      ROLLBACK TRAN actualiza_comisiones


IF EXISTS (SELECT titles.title, roysched.royalty FROM
                titles, roysched
                WHERE titles.title_id=roysched.title_id AND
                        roysched.royalty=15)
      BEGIN
         UPDATE roysched SET royalty=16 WHERE royalty=15
         COMMIT TRAN actualiza_comisiones
      END
   ELSE
      ROLLBACK TRAN actualiza_comisiones
```

Transacciones anidadas

Una transacción anidada o multinivel T consiste en un conjunto $T = \{t_1, t_2, \ldots, t_n\}$ de subtransacciones y en un orden parcial P sobre T.

Cada t_i de T puede abortar sin obligar a que T aborte. Puede que T reinicie t_i o simplemente no ejecute t_i. Si se compromete t_i, esa acción no hace que t_i sea permanente, sino que t_i *se compromete con* T, y puede que aborte si T aborta. La ejecución de T no debe violar el orden

parcial *P*. Es decir, si aparece un arco t_i □ t_j en el grafo de precedencia, t_j □ t_i no debe estar en el cierre transitivo de *P*.

Ejemplo (SQL Server):

```
USE
MyDB GO
CREATE PROCEDURE Formular_pedido AS --Crea un procedimiento
   almacenado BEGIN TRAN Tran_formular_pedido
   -- Instrucciones SQL para la formulación del pedido
   COMMIT TRAN Tran_formular_pedido

GO

BEGIN TRAN Tran_pedidos
   -- Formular un pedido
   EXEC Formular_pedido
COMMIT TRAN

Tran_pedidos GO
```

Oracle no admite transacciones anidadas, SQL Server sí.

Elementos

Los elementos son las unidades de datos para los que se controla el acceso. Por ejemplo: relación, tupla, campos, bloques, ...

La granularidad es el tamaño de los elementos. Así, se habla de sistemas de grano fino o de grano grueso, para denotar elementos pequeños o grandes, respectivamente.

A mayor granularidad, menor concurrencia.

No obstante, para determinadas operaciones es interesante bloquear relaciones enteras, como con la reunión de relaciones (join).

Bloqueos

Un bloqueo es una información del tipo de acceso que se permite a un elemento. El SGBD impone los bloqueos necesarios en cada momento. El gestor de acceso a los datos implementa las restricciones de acceso. En algunos sistemas se permite que el usuario pueda indicar el bloqueo más adecuado (locking hints).

Tipos de bloqueo con respecto a la operación:

read-locks: sólo permite lectura

write-locks: permite lectura y escritura

El gestor de bloqueos almacena los bloqueos en una tabla de bloqueos: (<elemento>, <tipo de bloqueo>, <transacción>)=(E,B,T)
La transacción T tiene un tipo de bloqueo B sobre el elemento E.
Normalmente, E es clave, aunque no siempre, porque varias transacciones pueden bloquear el mismo elemento de forma diferente.

Niveles de bloqueo
Especifica la granularidad del bloqueo

- Fila: Fila individual
- Clave: Fila de un índice
- Página: Páginas (8KB)
- Extent: Extensión (grupo de 8 páginas contiguas de datos o índices)
- Table: Tabla completa
- Database: Base de datos completa

Modos de bloqueo

Especifica el modo en que se bloquea un elemento

- Compartido: para operaciones sólo de lectura. Se permiten lecturas concurrentes, pero ninguna actualización.
- Actualización: para operaciones que *pueden* escribir. Sólo se permite que una transacción adquiera este bloqueo. Si la transacción modifica datos, se convierte en exclusivo, en caso contrario en compartido.
- Exclusivo. para operaciones que *escriben* datos. Sólo se permite que una transacción adquiera este bloqueo.
- Intención: se usan para establecer una jerarquía de bloqueo. Por ejemplo, si una transacción necesita bloqueo exclusivo y varias transacciones tienen bloqueo de intención, no se concede el exclusivo.
  - Intención compartido. Bloqueo compartido.
  - Intención exclusivo. Bloqueo exclusivo.
  - Compartido con intención exclusivo. Algunos bloqueos compartidos y otros exclusivos.
- Esquema. para operaciones del DDL.

- Actualización masiva. En operaciones de actualización masiva

Control de concurrencia

P: READ A; A:=A+1; WRITE A;

A en la BD	5	5	5	5	6	6
T1	READ A		A:=A+1			WRITE A
T2		READ A		A:=A+1	WRITE A	
A en T1	5	5	6	6	6	6
A en T2		5	5	6	6	

P: LOCK A; READ A; A:=A+1; WRITE A; UNLOCK A;

A en la BD		5	5	6	6	6	6	7	7
T1	LOCK A	READ A	A:=A + 1	WRITE A	UNLOC K A				
T2			LOCK A	LOCK A	LOCK A	READ A	A:=A + 1	WRITE A	UNLOCK A
A en T1		5	6	6	6				
A en T2						6	7	7	7

Livelock

Espera indefinida de una transacción por un bloqueo que no se llega a conceder porque se cede a otras transacciones.

Una solución (sistemas operativos): estrategia first-come-first-served (se atiende al primero que llega).

Deadlock

T1: LOCK A; LOCK B; UNLOCK A; UNLOCK B;

T2: LOCK B; LOCK A; UNLOCK B; UNLOCK A;

T1 y T2 bloquean A y B => Espera indefinida de T1 y T2.

Soluciones (sistemas operativos):

1- Concesión simultánea de *todos* los bloqueos de una transacción.

2- Asignar un orden lineal arbitrario a los elementos y requerir que las transacciones pidan los bloqueos en este orden.

3- Permitir los deadlocks y analizar cada cierto tiempo si existen.

Secuencialidad de planificaciones

La ejecución concurrente de varias transacciones es correcta <=> su efecto es el mismo si se ejecutan secuencialmente en cualquier orden.

Una *planificación* para un conjunto de transacciones es el orden en el que se realizan los pasos elementales de las transacciones.

Una planificación es *secuencial* si todos los pasos de cada transacción ocurren consecutivamente.

Una planificación es *secuenciable* si su efecto es equivalente al de la planificación secuencial. Ej: Se transfieren 10 unidades de A a B y 20 de B a C.

T1: READ A; A:=A-10; WRITE A; READ B; B:=B+10;WRITE B; T2: READ B; B:=B-20; WRITE B; READ C; C=C+20; WRITE C;

Cualquier planificación secuencial preserva A+B+C (Sólo hay dos, T1 antes de T2 o viceversa)

T1	T2	T1	T2	T1	T2
READ A		READ A		READ A	
A:=A-10			READ B	A:=A-10	
WRITE A		A:=A-10			READ B
READ B			B:=B-20	WRITE A	
B:=B+10		WRITE A			B:=B-20
WRITE B			WRITE B	READ B	
	READ B	READ B			WRITE B
	B:=B-20		READ C	B:=B+10	
	WRITE B	B:=B+10			READ C
	READ C		C=C+20	WRITE B	
	C=C+20	WRITE B			C=C+20
	WRITE C		WRITE C		WRITE C
Planificación secuencial		Planificación secuenciable pero no secuencial		Planificación no secuenciable (B se incrementa en 10 unidades en lugar de decrementarse; problema que se resuelve con bloqueos)	

Planificadores y protocolos

Un *planificador* arbitra los conflictos de acceso. Resuelve los livelocks con first-come-first-served. Puede también manejar deadlocks y no secuencialidad con:

- Espera de transacciones por liberación de bloqueos
- Abortos y reinicios de transacciones.

La espera produce en general más bloqueos pendientes. Los bloqueos pendientes provocan deadlocks.

Los deadlocks se resuelven generalmente abortando al menos una transacción.

Un *protocolo* es una restricción sobre la secuencia de pasos atómicos de una transacción. Por ejemplo, el orden impuesto para prevenir deadlocks.

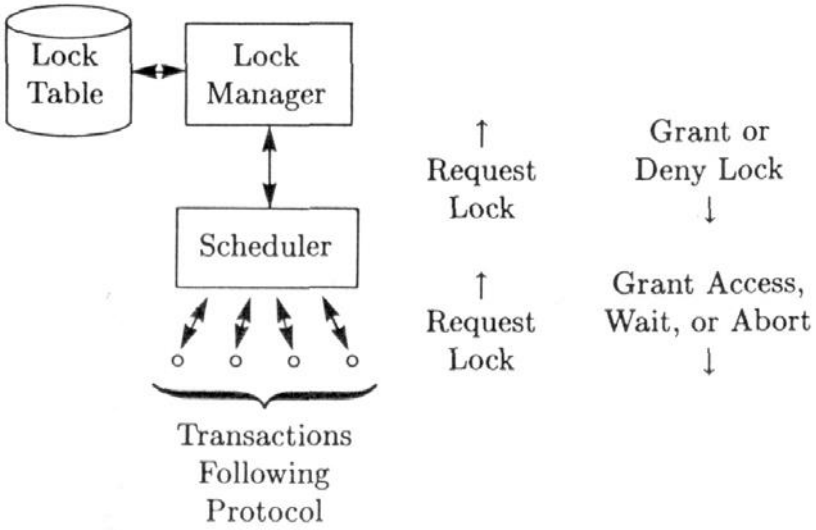


Figure 9.3 Protocol, scheduler, and lock manager.

Modelo transaccional simple

Una transacción es una secuencia de instrucciones LOCK y UNLOCK. LOCK asume una lectura de un elemento y UNLOCK una escritura.

La secuencialidad en este modelo simple implica secuencialidad en modelos más complejos.

Semántica de transacciones

Es el significado de la ejecución de la transacción (qué hace).

Para diseñar protocolos y planificadores es necesario relacionar la semántica (informal) de las transacciones con un test que determine si una secuencia de pasos de transacciones entrelazadas es secuenciable.

Primero: ver que la semántica es apropiada (conservadora: puede prohibir planificaciones que sean secuenciables pero no permite las que no lo sean).

Segundo: se hace corresponder la semántica con un grafo de ejecución que permite decidir si una planificación es secuenciable.

Se asocia una función f al par LOCK A y UNLOCK A. Argumentos: todos los elementos bloqueados anteriormente a UNLOCK A.

A_0 es el valor inicial de A.

Los *valores* de A son fórmulas que se construyen aplicando estas funciones a los valores iniciales.

Se asume que fórmulas diferentes tienen valores diferentes.

Dos planificaciones son *equivalentes* si sus fórmulas para el valor final de cada elemento son iguales.

T1		T2		T3	
LOCK A		LOCK B		LOCK A	
LOCK B		LOCK C		LOCK C	
UNLOCK A	$f_1(A, B)$	UNLOCK B	$f_3(B, C)$	UNLOCK C	$f_6(A, C)$
UNLOCK B	$f_2(A, B)$	LOCK A		UNLOCK A	$f_7(A, C)$
		UNLOCK C	$f_4(A, B, C)$		
		UNLOCK A	$f_5(A, B, C)$		

	Step	A	B	C
(1)	T_1: LOCK A	A_0	B_0	C_0
(2)	T_2: LOCK B	A_0	B_0	C_0
(3)	T_2: LOCK C	A_0	B_0	C_0
(4)	T_2: UNLOCK B	A_0	$f_3(B_0, C_0)$	C_0
(5)	T_1: LOCK B	A_0	$f_3(B_0, C_0)$	C_0
(6)	T_1: UNLOCK A	$f_1(A_0, f_3(B_0, C_0))$	$f_3(B_0, C_0)$	C_0
(7)	T_2: LOCK A	$f_1(A_0, f_3(B_0, C_0))$	$f_3(B_0, C_0)$	C_0
(8)	T_2: UNLOCK C	$f_1(A_0, f_3(B_0, C_0))$	$f_3(B_0, C_0)$	(i)
(9)	T_2: UNLOCK A	(ii)	$f_3(B_0, C_0)$	(i)
(10)	T_3: LOCK A	(ii)	$f_3(B_0, C_0)$	(i)
(11)	T_3: LOCK C	(ii)	$f_3(B_0, C_0)$	(i)
(12)	T_1: UNLOCK B	(ii)	$f_2(A_0, f_3(B_0, C_0))$	(i)
(13)	T_3: UNLOCK C	(ii)	$f_2(A_0, f_3(B_0, C_0))$	(iii)
(14)	T_3: UNLOCK A	(iv)	$f_2(A_0, f_3(B_0, C_0))$	(iii)

Key:

(i): $f_4(f_1(A_0, f_3(B_0, C_0)), B_0, C_0)$ (iii): $f_6((ii), (i))$

(ii): $f_5(f_1(A_0, f_3(B_0, C_0)), B_0, C_0)$ (iv): $f_7((ii), (i))$

Figure 9.5 A schedule.

La planificación no es secuenciable:

Si T1 precede a T2 en la planificación secuencial, el valor final de B es

$f_3(f_2(A_0, B_0), C_0)$ en lugar de

$f_2(A_0, f_3(B_0, C_0))$.

Si T2 precede a T1, el valor final de A aplicaría

f_1 a una expresión con $f5$.

En la figura anterior esto no sucede, por tanto, T2 no puede preceder a T1 en una planificación secuencial equivalente.

T1 no puede preceder a T2 ni T2 puede preceder a T1 en una planificación secuencial equivalente => no existe dicha planificación secuencial equivalente.

Test de secuencialidad

Para determinar si un planificador es correcto, se demuestra que cada planificación que admite es secuenciable.

Se examina la planificación con respecto al orden en que se bloquean elementos. Este orden debe ser consistente con la planificación secuencial equivalente. Si hay dos secuencias con órdenes diferentes de transacciones, estos dos órdenes no son consistentes con una planificación secuencial.

El problema se reduce a la búsqueda de ciclos en un grafo dirigido.

Algoritmo:

Entrada: Una planificación S para las transacciones T1,..,Tk.

Salida: Determina si S es secuenciable y, si lo es, produce una planificación secuencial equivalente a S.

Método:

Creación de un grafo de secuencialización G cuyos nodos son transacciones. S=a1;a2;..;an ai es Tj: LOCK Am o Tj: UNLOCK Am

Dado ai= Tj: UNLOCK Am se busca ap= Ts: LOCK Am en el orden de S tal que s<>j.

Entonces, el arco <Tj,Ts> pertenece a G.

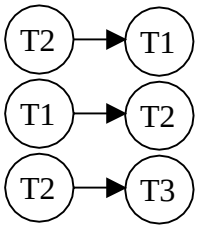
Intuitivamente: en cualquier planificación secuencial equivalente a S, Tj debe preceder a Ts. Si G tiene un ciclo, S no es secuenciable.

Si G no tiene ciclos, se produce una secuencia de transacciones mediante la ordenación topológica (también determina si no hay ciclos):

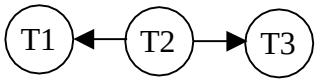
1. Encontrar un nodo Ti sin arcos de entrada (si no se encuentra, hay un ciclo).
2. Listar Ti y eliminarlo.
3. Si quedan nodos, ir a 1.

Ej. del caso anterior:

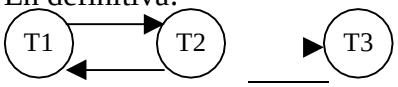
(4) T2: UNLOCK B
(5) T1: LOCK B
(6) T1: UNLOCK A
(7) T2: LOCK A
(8) T2: UNLOCK C
(11) T3: LOCK C



(9) T2: UNLOCK A
(1) T1: LOCK A
(10) T3: LOCK A



En definitiva:



Teorema 6.1: El algoritmo determina correctamente si una planificación es secuenciable.

Demostración: Ejercicio.

Protocolo de bloqueo de dos fases

Requisito: todos los bloqueos preceden a los desbloqueos. Primera fase: bloqueos. Segunda: desbloqueos.

Propiedad: según este requisito no existen planificaciones no secuenciables legales.

Teorema 9.2: Si S es cualquier planificación de transacciones de dos fases, S es secuenciable.

Demostración: Supongamos que no sea secuenciable. Entonces, por el teorema 9.1, el grafo de secuencialización de G para S tiene un ciclo:

T_{i1} □$T_{i\,2}$ □**L** □T_{ip} □T_{i1}

Por tanto, algún bloqueo de $T_{i\,2}$ sigue a un desbloqueo de T_{i1} ; algún bloqueo de $T_{i\,3}$ sigue a undesbloqueo de

$T_{i\,2}$, ..., algún bloqueo de

T_{i1}

sigue a un desbloqueo de

T_{ip} . Por tanto, algún

bloqueo de T_{i1}

sigue a un desbloqueo de T_{i1} , contradiciendo la suposición de que

T_{i1}

es una transacción de dos fases.

El protocolo de dos fases no asegura ausencia de interbloqueos:

T1= LOCK B; LOCK A; UNLOCK A; UNLOCK B;

T2= LOCK A; LOCK B; UNLOCK B; UNLOCK A;

T1	T2
LOCK B	
	LOCK A
	LOCK B
LOCK A	
¡Interbloqueo!	

Protocolos basados en grafos

A menudo es útil observar el conjunto de elementos de datos de la base de datos como una estructura de grafo. Por ejemplo:

1. Organización lógica o física de los elementos.
2. Definición de elementos de varios tamaños, donde los grandes engloban a los pequeños. Ej: relacional: tupla bloque relación base de datos.
3. Control de concurrencia efectivo.

Se pueden diseñar protocolos que no sean de dos fases pero que aseguren la secuencialidad.

En general, sea

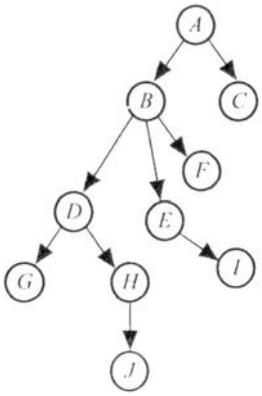
D $\{d_1, d_2, \text{K}, d_n\}$ el conjunto de todos los elementos de datos de la base dedatos dotado de un orden parcial . Si en el grafo existe un arco d_i d_j, entonces la transacción que acceda tanto a d_i como a d_j debe acceder primero a d_i y después a d_j.

Protocolo de árbol

Caso particular de protocolo basado en grafos, grafos que sean árboles con raíz. Reglas:Cada transacción T_i bloquea al menos un elemento.El primer bloqueo de T_i puede ser sobre cualquier elemento.

1. Sucesivos bloqueos de T_i sólo pueden ser sobre elementos cuyo padre haya bloqueado T_i.
2. Los elementos se pueden desbloquear en cualquier momento.T_i no puede bloquear de nuevo un elemento que haya bloqueado y desbloqueado anteriormente.

Ej:



T_1: LOCK *B*; LOCK *E*; LOCK *D*; UNLOCK *B*; UNLOCK *E*; LOCK *G*; UNLOCK

D; UNLOCK *G*;

T_2: LOCK *D*; LOCK *H*; UNLOCK *D*; UNLOCK *H*; T_3: LOCK *B*; LOCK *E*; UNLOCK *E*; UNLOCK *B*; T_4: LOCK *D*; LOCK *H*; UNLOCK *D*; UNLOCK *H*;

Planificación secuenciable:

T_1	T_2	T_3	T_4
LOCK *B*			
	LOCK *D* LOCK *H* UNLOCK *D*		
LOCK *E* LOCK *D* UNLOCK *B*			

UNLOCK *E*			
		LOCK *B* LOCK *E*	
	UNLOCK *H*		
LOCK *G* UNLOCK *D*			
			LOCK *D* LOCK *H* UNLOCK *D* UNLOCK *H*
		UNLOCK *E* UNLOCK *B*	
UNLOCK *G*			

Se puede demostrar que el protocolo de árbol no sólo asegura la secuencialidad en cuanto a conflictos, sino que también asegura la ausencia de interbloqueos.

El protocolo de bloqueo de árbol tiene la ventaja sobre el protocolo de bloqueo de dos fases de que los desbloqueos se pueden dar antes. El hecho de desbloquear antes puede llevar a unos tiempos de espera menores y a un aumento de la concurrencia. Adicionalmente, debido a que el protocolo está libre de interbloqueos, no se necesitan retrocesos.

Sin embargo, el protocolo tiene el inconveniente de que, en algunos casos, una transacción puede que tenga que bloquear elementos de datos a los que no accede. Por ejemplo, una transacción que tenga que acceder a los elementos de datos A y J en el ejemplo anterior, debe bloquear no sólo A y J sino también los elementos de datos B, D y H. Estos bloqueos

adicionales producen un aumento del coste de los bloqueos, la posibilidad de tiempos de espera adicionales y un descenso potencial de la concurrencia.

Además, sin un conocimiento previo de los elementos de datos que es necesario bloquear, las transacciones tienen que bloquear la raíz del árbol y esto puede reducir considerablemente la concurrencia.

Pueden existir planificaciones secuenciables que no se pueden obtener por medio del protocolo de árbol.

Hay planificaciones que son posibles por medio del protocolo de bloqueo de dos fases que no son posibles por medio del protocolo de árbol y viceversa.

Algoritmo para construir un orden secuencial de las transacciones:

1. Crear un nodo por cada transacción.Sean T_i y T_j transacciones que bloquean el mismo elemento, y FIRST(T) el primer elemento bloqueado por T.
2. Si FIRST(T_i) es independiente de FIRST(T_j) (no son "parientes"=pertenecen a árboles disjuntos), el protocolo garantiza que T_i y T_j no bloquearán un nodo en común, por lo que no se traza un arco entre ellas.
3. Si FIRST(T_i) es antepasado de FIRST(T_j) entonces si T_i bloquea FIRST(T_j) antes que T_j, se dibuja un arco $T_i \rightarrow T_j$; si sucede al contrario se dibuja un arco $T_j \rightarrow T_i$.

Se puede demostrar que el grafo resultante es acíclico y que cualquier ordenación topológica del grafo es un orden secuencial para las transacciones. [Ullman]

Protocolos basados en marcas temporales

Se usan cuando no se imponen bloqueos pero se sigue asegurando secuencialidad.

Marcas temporales

Transacciones:

Cada T_i lleva asociada una marca temporal fijada $MT(T_i)$.

Si T_i se selecciona antes que T_j, entonces $MT(T_i) < MT(T_j)$.El valor de transacciones.

Elementos:

$MT(T_i)$ puede extraerse del reloj del sistema o con contadores lógicos de cada elemento de datos D lleva asociado dos marcas temporales:

MTR(D): mayor marca temporal de todas las transacciones que ejecutan con éxito READ D;
MTW(D): mayor marca temporal de todas las transacciones que ejecutan con éxito WRITE D;

Protocolo de ordenación por marcas temporales

Asegura que todas las operaciones leer y escribir conflictivas se ejecutan en el orden de las marcas temporales.

1. Supóngase que la transacción T_i ejecuta READ(D).
   a. Si MT(T_i) < MTW(D) entonces T_i necesita leer un valor de D que ya se ha sobrescrito. Por tanto se rechaza la operación READ y T_i se retrocede.
   b. Si MT(T_i) □ MTW(D) entonces se ejecuta la operación READ y MTR(D) se asigna al máximo de MTR(D) y de MT(T_i).
2. Supóngase que la transacción T_i ejecuta WRITE(D).
   a. Si MT(T_i) < MTR(D) entonces el valor de D que produce T_i se necesita previamente y el sistema asume que dicho valor no se puede producir nunca. Por tanto, se rechaza la operación WRITE y T_i se retrocede.
   b. Si MT(T_i) < MTW(D) entonces T_i está intentando escribir un valor de D obsoleto. Por tanto, se rechaza la operación WRITE y T_i se retrocede.

   c. En otro caso se ejecuta la operación WRITE y MT(T_i) se asigna a MTW(D). Ejemplo:

T_1: READ B;
READ A;
PRINT $A + B$.

La transacción T_2 transfiere 10.000 pts. de la cuenta A a la B y muestra después el contenido de ambas:

T_2: READ B;
$B := B - 10.000$;
WRITE B;
READ A;
$A := A + 10.000$;
WRITE A;
PRINT $A + B$.

T_1	T_2
READ B	
	READ B $B := B -$ 10.000 WRITE B
READ A	
	READ A
PRINT $A + B$	
	$A := A +$ 10.000 WRITE A PRINT $A + B$

El protocolo de ordenación por marcas temporales asegura la secuencialidad. Esta afirmación se deduce del hecho de que las operaciones conflictivas se procesan durante la ordenación de las marcas temporales.

El protocolo asegura la ausencia de interbloqueos, ya que ninguna transacción tiene que esperar.

Protocolos optimistas

Cuando los conflictos entre las transacciones son raros se pueden aplicar técnicas optimistas, evitando los protocolos anteriores (más costosos). Estas técnicas asumen que no habrá conflictos en las planificaciones y evitan los costosos bloqueos. Cuando se intente comprometer una transacción se determinará si ha existido conflicto o no. En su caso, la transacción se retrocederá y volverá a iniciarse.

Si los conflictos son raros también lo serán los retrocesos, con lo que el nivel de concurrencia aumentará, aunque también hay que tener en cuenta el coste de los posibles retrocesos en términos de tiempo (todas las actualizaciones se hacen en memoria, no hay retrocesos en cascada porque los cambios no se hacen en la base de datos).

Fases de un protocolo de control de concurrencia optimista:

- Fase de lectura. Desde el inicio de la transacción hasta justo antes de su compromiso. Las actualizaciones se realizan en memoria, no en la propia base de datos.
- Fase de validación. Es la siguiente a la de lectura. Se comprueba que no se viola la secuencialidad. Casos:
  - Transacciones sólo de lectura. Se reduce a comprobar que los valores actuales en la base de datos son los mismos que se leyeron inicialmente.
  - Transacciones con actualizaciones. Se determina si la transacción deja a la base de datos en un estado consistente.

  En ambos casos, si la secuencialidad no se conserva, la transacción se retrocede y se reinicia.
- Fase de escritura. Es posterior a la fase de validación cuando se mantiene la secuencialidad. Consiste en escribir en la base de datos los cambios producidos en la copia local.

Fase de validación:

Cada transacción T recibe una marca temporal (*Inicio(T)*) al iniciar su ejecución, otra al iniciar su fase de validación (*Validación(T)*) y otra al terminar (*Fin(T)*).

Para pasar el test de validación debe ser cierto uno de:

1) Para todas las transacciones S con marcas temporales anteriores a T => *Fin(S)* <

Inicio(T).

2) Si la transacción T empieza antes de que acabe otra S anterior, entonces:

a) Los elementos escritos por S no son los leídos por T, y

b) La transacción S completa su fase de escritura antes de que T comience su fase de validación, es decir, *Inicio(T) < Fin(S) < Validación(T)*

Gestión de fallos de transacciones

Causas de aborto:

1. Fallo de la transacción: interrupción por el usuario, fallo aritmético, privilegios de acceso...
2. Deadlock->aborto de una transacción
3. Algoritmos de secuencialidad.
4. Error software o hardware

Fácil: 1, 2 y 3. Difícil: 4. Puntos de recuperación por copias de seguridad.

Compromiso de transacciones

Transacciones activas. En ejecución

Transacciones completadas. Sólo pueden abortar por causa grave: 4.

Punto de compromiso: COMMIT. Momento a partir del cual se entienden completadas. Las transacciones comprometidas ni se retroceden ni se rehacen.

Datos inseguros

	T1	T2
1	LOCK A	
2	READ A	
3	A:=A-1	
4	WRITE A	
5	LOCK B	
6	UNLOCK A	
7		LOCK A
8		READ A
9		A:=A*2
10	READ B	
11		WRITE A
12		COMMIT
13		UNLOCK A
14	B:=B/A	

15	*¡Fallo!* (división por 0)-> abortar T1	

Acciones después del fallo de T1:

1. UNLOCK B
2. UNDO (4) (de un registro que se verá posteriormente)
3. ROLLBACK T2, porque maneja valores de A inseguros
4. ROLLBACK Ti, para todo Ti que haya usado el dato A inseguro a partir de 14: Retroceso (rollback) en cascada.

Bloqueo de dos fases estricto

Se usa para solucionar el problema anterior.

1. Una transacción no puede escribir en la base de datos hasta que se haya alcanzado su punto de compromiso. (Evita los retrocesos en cascada)
2. Una transacción no puede liberar ningún bloqueo hasta que haya finalizado de escribir en la base de datos, i.e., los bloqueos no se liberan hasta después del punto de compromiso.

Recuperación de caídas

Tipos de caídas:

- Error de memoria volátil.
- Error de memoria permanente.

Problema: asegurar la atomicidad de las escrituras de las transacciones. Puede haber una caída del sistema antes de que se hayan escrito todos los datos modificados por una transacción.

Recuperación basada en el registro (log)

Es la técnica más habitual.

Almacena consecutivamente los cambios de la base de datos y el estado de cada transacción. Las tuplas de cuatro elementos significan: (Transacción, Elemento, Valor nuevo, Valor anterior)

Acción	**Entrada del registro**
begin-transaction	(T,begin)

T: WRITE A (A=v)	(T,A,v,A_0)
T: COMMIT	(T, commit)
Abortar T	(T, abort)

Ejemplo anterior:

Paso	**Entrada del registro**
Antes de 1	(T1,begin)
4	(T1,A,9,10)
Antes de 7	(T2,begin)
11	(T2,A,18,9)
12	(T2, commit)
Después de 14	(T1, abort)

Es fundamental que el registro de escritura se cree antes de modificar la base de datos. Generalmente el registro histórico se implementa en almacenamiento estable.

Hay dos técnicas principales de implementar este tipo de recuperación: modificación diferida e inmediata de la base de datos.

Modificación diferida de la base de datos

Pasos:

- Todas las operaciones de escritura se anotan en el registro histórico.
- Cuando la transacción está comprometida parcialmente (se han realizado todas sus operaciones, pero aún no se ha modificado la base de datos) se llevan a cabo las escrituras pendientes examinando el registro histórico.
- A continuación, se puede anotar la transacción como comprometida. Si ocurre una caída entre las escrituras, la transacción se puede deshacer.

Modificación inmediata de la base de datos

Pasos:

- Todas las operaciones de escritura se anotan en el registro histórico.
- Después de cada operación de escritura se realiza la escritura (modificaciones no comprometidas).
- Finalmente, se anota la transacción como comprometida.

Si ocurre una caída entre las escrituras, la transacción se puede deshacer.

Fallos de memoria permanente

Soluciones:

1) Salvados periódicos de la instalación.
2) Salvados periódicos de la base de datos.

UNIDAD 3: TIPOS DE BASES DE DATOS

Dependiendo de los requerimientos de la base de datos, el diseño puede ser algo complejo, pero con algunas reglas simples que tengamos en la cabeza será mucho más fácil crear una base de datos perfecta para nuestro siguiente proyecto.

Requerimientos para el Diseño de Bases de Datos

Son muchas las consideraciones a tomar en cuenta al momento de hacer el diseño de la base de datos, quizá las más fuertes sean:

• La velocidad de acceso,
• El tamaño de la DB,
• El tipo de los DATOS,
• Facilidad de acceso a los datos,
• Facilidad para extraer los datos requeridos,
• El comportamiento del manejador de bases de datos con cada tipo de datos.

No obstante que pueden desarrollarse sistemas de procesamiento de archivo e incluso manejadores de bases de datos basándose en la experiencia del equipo de desarrollo de software logrando resultados altamente aceptables, siempre es recomendable la utilización de determinados estándares de diseño que garantizan el nivel de eficiencia mas alto en lo que se refiere a almacenamiento y recuperación de los datos.

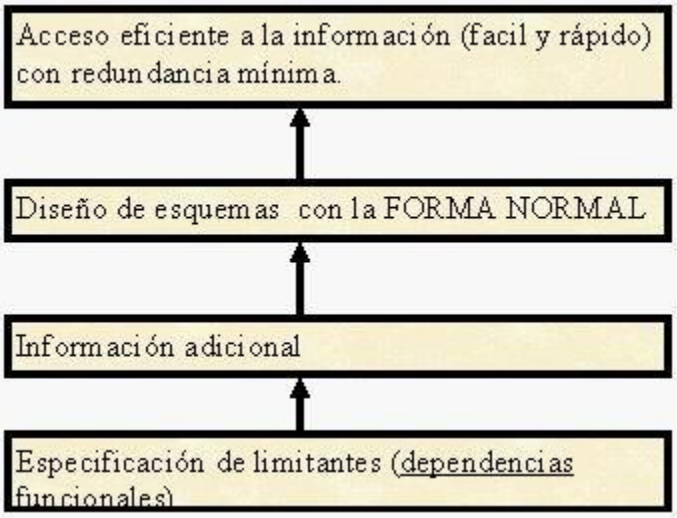
De igual manera se obtiene modelos que optimizan el aprovechamiento secundario y la sencillez y flexibilidad en las consultas que pueden proporcionarse al usuario.

Un modelo de datos es un conjunto de conceptos utilizados para organizar los datos de interés y describir su estructura en forma comprensible para un sistema informático.

Cada modelo de datos provee mecanismos de estructuración, que permiten definir nuevos tipos de datos a partir de tipos elementales predefinidos.

OBJETIVOS DEL DISEÑO DE BASES DE DATOS

Entre las metas más importantes que se persiguen al diseñar un modelo de bases de datos, se encuentran las siguientes que pueden observarse en esta figura.



Almacenar Solo DATOS Necesarios.

A menudo pensamos en todo lo que quisiéramos que estuviera almacenado en una base de datos y diseñamos la base de datos para guardar dichos datos. Debemos de ser realistas acerca de nuestras necesidades y decidir qué dato es realmente necesario.

Frecuentemente podemos generar algunos datos sobre la marcha sin tener que almacenarlos en una tabla de una base de datos. En estos casos también tiene sentido hacer esto desde el punto de vista del desarrollo de la aplicación.

Un modelo de datos es básicamente una "descripción" de algo conocido como *contenedor de datos* (algo en donde se guarda los datos), así como de los métodos para almacenar y recuperar dato de esos contenedores. Los modelos de datos no son cosas físicas: son abstracciones que permiten la implementación de un sistema eficiente de

base de datos; por lo general se refieren a algoritmos, y conceptos matemáticos.

El diseño de bases de datos es el proceso por el que se determina la organización de una base de datos, incluidos su estructura, contenido y las aplicaciones que se han de desarrollar. Durante mucho tiempo, el diseño de bases de datos fue considerado una tarea para expertos: más un arte que una ciencia. Sin embargo, se ha progresado mucho en el diseño de bases de datos y éste se considera ahora una disciplina estable, con métodos y técnicas propios. Debido a la creciente aceptación de las bases de datos por parte de la industria y el gobierno en el plano comercial, y a una variedad de aplicaciones científicas y técnicas, el diseño de bases de datos desempeña un papel central en el empleo de los recursos de datos en la mayoría de las organizaciones. El diseño de bases de datos ha pasado a constituir parte de la formación

general de los informáticos, en el mismo nivel que la capacidad de construir algoritmos usando un lenguaje de programación convencional.

Las últimas dos décadas se han caracterizado por un fuerte crecimiento en el número e importancia de las aplicaciones de bases de datos. Las bases de datos son componentes esenciales de los sistemas de datos, usadas rutinariamente en todos los computadores [...]. El diseño de bases de datos se ha convertido en una actividad popular, desarrollada no sólo por profesionales sino también por no especialistas.

A finales de la década de 1960, cuando las bases de datos entraron por primera vez en el mercado del software, los diseñadores de bases de datos actuaban como artesanos, con herramientas muy primitivas: diagramas de bloques y estructuras de registros eran los formatos comunes para las especificaciones, y el diseño de bases de datos se confundía frecuentemente con la implantación de las bases de datos.

Esta situación ahora ha cambiado: los métodos y modelos de diseño de bases de datos han evolucionado paralelamente con el progreso de la tecnología en los sistemas de bases de datos. Se ha entrado en la era de los sistemas relacionales de bases de datos, que ofrecen poderosos lenguajes de consulta, herramientas para el desarrollo de aplicaciones e interfaces amables con los usuarios. La tecnología de bases de datos cuenta ya con un marco teórico, que incluye la teoría relacional de datos, procesamiento y optimización de consultas, control de concurrencia, gestión de transacciones y recuperación, etc.

Según ha avanzado la tecnología de bases de datos, así se han desarrollado las metodologías y técnicas de diseño. Se ha alcanzado un consenso, por ejemplo, sobre la descomposición del proceso de diseño en fases, sobre los principales objetivos de cada fase y sobre las técnicas para conseguir estos objetivos.

Desafortunadamente, las metodologías de diseño de bases de datos no son muy populares; la mayoría de las organizaciones y de los diseñadores individuales confía muy poco en las metodologías para llevar a cabo el diseño y esto se considera, con frecuencia, una de las principales causas de fracaso en el desarrollo de los sistemas de datos.

Debido a la falta de enfoques estructurados para el diseño de bases de datos, a menudo se subestiman el tiempo o los recursos necesarios para un proyecto de bases de datos, las bases de datos son inadecuadas o ineficientes en relación a las demandas de la aplicación, la documentación es limitada y el mantenimiento es difícil.

Muchos de estos problemas se deben a la falta de una claridad que permita entender la naturaleza exacta de los datos, a un nivel conceptual y abstracto. En muchos casos, los datos se describen desde el comienzo del proyecto en términos de las estructuras finales de

almacenamiento; no se da peso a un entendimiento de las propiedades estructurales de los datos que sea independiente de los detalles de la realización. Ventajas del uso de la base de datos en la organización:

1. Independencia de datos y tratamiento.

Cambio en datos no implica cambio en programas y viceversa (Menor coste de mantenimiento).

2. Coherencia de resultados.

• Reduce redundancia:

• Acciones lógicamente únicas.

• Se evita inconsistencia.

3. Mejora en la disponibilidad de datos

• No hay dueño de datos (No igual a ser públicos).

• Ni aplicaciones ni usuarios.

• Guardamos descripción (Idea de catálogos).

4. Cumplimiento de ciertas normas.

• Restricciones de seguridad.

• Accesos (Usuarios a datos).

• Operaciones (Operaciones sobre datos).

5. Otras ventajas:

Más eficiente gestión de almacenamiento.

Efecto sinérgico.

Metodología de diseño de bases de datos

El diseño de una base de datos es un proceso complejo que abarca decisiones a muy distintos niveles. La complejidad se controla mejor si se descompone el problema en subproblemas y se resuelve cada uno de estos subproblemas independientemente, utilizando técnicas específicas. Así, el diseño de una base de datos se descompone en diseño conceptual, diseño lógico y diseño físico.

El diseño conceptual parte de las especificaciones de requisitos de usuario y su resultado es el esquema conceptual de la base de datos. Un *esquema conceptual* es una descripción de alto nivel de la estructura de la base de datos, independientemente del DBMS que se vaya a utilizar para manipularla. Un *modelo conceptual* es un lenguaje que se utiliza para describir esquemas conceptuales. El objetivo del diseño conceptual es describir los datos de la base de datos y no las estructuras de almacenamiento que se necesitarán para manejar estos datos.

El diseño lógico parte del esquema conceptual y da como resultado un esquema lógico. Un *esquema lógico* es una descripción de la estructura de la base de datos en términos de las estructuras de datos que puede procesar un tipo de DBMS. Un *modelo lógico* es un lenguaje usado para especificar esquemas lógicos (modelo relacional, modelo de red, etc.). El diseño lógico depende del tipo de DBMS que se vaya a utilizar, no depende del producto concreto.

El diseño físico parte del esquema lógico y da como resultado un esquema físico. Un *esquema físico* es una descripción de la implementación de una base de datos en memoria secundaria: las estructuras de almacenamiento y los métodos utilizados para tener un acceso eficiente a los datos. Por ello, el diseño físico depende del DBMS concreto y el esquema físico se expresa mediante su lenguaje de definición de datos.

Modelos de datos

Un *modelo de datos* es entonces una serie de conceptos que puede utilizarse para describir un conjunto de datos y las operaciones para manipularlos. Hay dos tipos de modelos de datos: los modelos conceptuales y los modelos lógicos. Los *modelos conceptuales* se utilizan para representar la realidad a un alto nivel de abstracción.

Mediante los modelos conceptuales se puede construir una descripción de la realidad fácil de entender. En los *modelos lógicos*, las descripciones de los datos tienen una correspondencia sencilla con la estructura física de la base de datos.

En el diseño de bases de datos se usan primero los modelos conceptuales para lograr una descripción de alto nivel de la realidad, y luego se transforma el esquema conceptual en un esquema lógico. El motivo de realizar estas dos etapas es la dificultad de abstraer la estructura de una base de datos que presente cierta complejidad. Un *esquema* es un conjunto de representaciones lingüísticas o gráficas que describen la estructura de los datos de interés.

Los modelos conceptuales deben ser buenas herramientas para representar la realidad, por lo que deben poseer las siguientes cualidades:

• *Expresividad*: deben tener suficientes conceptos para expresar perfectamente la realidad.

• *Simplicidad*: deben ser simples para que los esquemas sean fáciles de entender.

• *Minimalidad*: cada concepto debe tener un significado distinto.

• *Formalidad*: todos los conceptos deben tener una interpretación única, precisa y bien definida.

En general, un modelo no es capaz de expresar todas las propiedades de una realidad determinada, por lo que hay que añadir aserciones que complementen el esquema.

Algunos modelos con frecuencia utilizados en las bases de datos son:

Bases de datos jerárquicas

Éstas son bases de datos que, como su nombre indica, almacenan sus datos en una estructura jerárquica. En este modelo los datos se organizan en una forma similar a un árbol (visto al revés), en donde un *nodo padre* de datos puede tener varios *hijos*. El nodo que no tiene padres es llamado *raíz*, y a los nodos que no tienen hijos se los conoce como *hojas*.

Las bases de datos jerárquicas son especialmente útiles en el caso de aplicaciones que manejan un gran volumen de datos y datos muy compartidos permitiendo crear estructuras estables y de gran rendimiento.

Una de las principales limitaciones de este modelo es su incapacidad de representar eficientemente la redundancia de datos.

Base de datos de red

Éste es un modelo ligeramente distinto del jerárquico; su diferencia fundamental es la modificación del concepto de *nodo*: se permite que un mismo nodo tenga varios padres (posibilidad no permitida en el modelo jerárquico).

Fue una gran mejora con respecto al modelo jerárquico, ya que ofrecía una solución eficiente al problema de redundancia de datos; pero, aun así, la dificultad que significa administrar los datos en una base de datos de red ha significado que sea un modelo utilizado en su mayoría por programadores más que por usuarios finales.

Base de datos relacional

Éste es el modelo más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente. Tras ser postulados sus fundamentos en 1970 por Edgar Frank Codd, de los laboratorios IBM en San José (California), no tardó en consolidarse como un nuevo paradigma en los modelos de base de datos. Su idea fundamental es el uso de "relaciones". Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados "tuplas". Pese a que ésta es la teoría de las bases de datos relacionales creadas por Edgar Frank Codd, la mayoría de las veces se conceptualiza de una manera más fácil de imaginar. Esto es pensando en cada relación como si fuese una tabla que está compuesta por *registros* (las filas de una tabla), que representarían las tuplas, y *campos* (las columnas de una tabla).

En este modelo, el lugar y la forma en que se almacenen los datos no tienen relevancia (a diferencia de otros modelos como el jerárquico y el de red). Esto tiene la considerable ventaja de que es más fácil de entender y de utilizar para un usuario esporádico de la base de datos.
Los datos pueden ser recuperados o almacenada mediante "consultas" que ofrecen una amplia flexibilidad y poder para administrar los datos.
El lenguaje más habitual para construir las consultas a bases de datos relacionales es SQL, *Structured Query Language* o *Lenguaje Estructurado de Consultas*, un estándar implementado por los principales motores o sistemas de gestión de bases de datos relacionales.
Durante su diseño, una base de datos relacional pasa por un proceso al que se le conoce como normalización de una base de datos.
Durante los años '80 (1980-1989) la aparición de dBASE produjo una revolución en los lenguajes de programación y sistemas de administración de datos. Aunque nunca debe olvidarse que dBase no utilizaba SQL como lenguaje base para su gestión.

Bases de datos multidimensionales

Son bases de datos ideadas para desarrollar aplicaciones muy concretas, como creación de **Cubos OLAP**. Básicamente no se diferencian demasiado de las bases de datos relacionales (una tabla en una base de datos multidimensional podría serlo también en una base de datos multidimensional), la diferencia está más bien a nivel conceptual; en las bases de datos multidimensionales los campos o atributos de una tabla pueden ser de dos tipos, o bien representan dimensiones de la tabla, o bien representan métricas que se desean estudiar.

Bases de datos orientadas a objetos

Este modelo, bastante reciente, y propio de los modelos informáticos orientados a objetos, trata de almacenar en la base de datos los *objetos* completos (estado y comportamiento).
Una base de datos orientada a objetos es una base de datos que incorpora todos los conceptos importantes del paradigma de objetos:
• Encapsulación - Propiedad que permite ocultar los datos al resto de los objetos, impidiendo así accesos incorrectos o conflictos.
• Herencia - Propiedad a través de la cual los objetos heredan comportamiento dentro de una jerarquía de clases.

• Polimorfismo - Propiedad de una operación mediante la cual puede ser aplicada a distintos tipos de objetos.

En bases de datos orientadas a objetos, los usuarios pueden definir operaciones sobre los datos como parte de la definición de la base de datos. Una operación (llamada función) se especifica en dos partes. La interfaz (o signatura) de una operación incluye el nombre de la operación y los tipos de datos de sus argumentos (o parámetros). La implementación (o método) de la operación se especifica separadamente y puede modificarse sin afectar la interfaz. Los programas de aplicación de los usuarios pueden operar sobre los datos invocando a dichas operaciones a través de sus nombres y argumentos, sea cual sea la
forma en la que se han implementado. Esto podría denominarse independencia entre programas y operaciones.

Se está trabajando en **SQL3,** que es el estándar de SQL92 ampliado, que soportará los nuevos conceptos orientados a objetos y mantendría compatibilidad con SQL92.

Bases de datos documentales

Permiten la indexación a texto completo, y en líneas generales realizar búsquedas más potentes. Tesauro es un sistema de índices optimizado para este tipo de bases de datos.

Base de datos deductivas

Un sistema de **base de datos deductivas**, es un sistema de base de datos pero con la diferencia de que permite hacer deducciones a través de inferencias. Se basa principalmente en reglas y hechos que son almacenados en la base de datos. También las bases de datos deductivas son llamadas **base de datos lógica**, a raíz de que se basan en lógica matemática.

Bases de datos distribuida

La base de datos está almacenada en varias computadoras conectadas en red. Surgen debido a la existencia física de organismos descentralizados. Esto les da la capacidad de unir las bases de datos de cada localidad y acceder así a distintas universidades, sucursales de tiendas, etcétera.

Resumiendo:

Un modelo de datos para las bases de datos es una colección de conceptos que se emplean para describir la estructura de una base de datos. Esa colección de conceptos incluye: entidades, atributos y relaciones.

Las bases de datos almacenan datos, permitiendo manipularlos fácilmente y mostrarlos de diversas formas.

El proceso de construir una base de datos es llamado diseño de base de datos.

La mayoría de los modelos de datos poseen un conjunto de operaciones básicas para especificar consultas y actualizaciones de la base de datos, donde los modelos de datos pueden clasificarse en:

* Modelos de datos de alto nivel o conceptuales: disponen de conceptos cercanos a la forma en que los usuarios finales perciben una base de datos.
* Modelos de datos de bajo nivel o físicos: disponen de conceptos que describen detalles sobre el almacenamiento de los datos en la computadora.
* Modelos de datos de representación (o de implementación): disponen de conceptos que pueden entender los usuarios finales, pero que no están alejados de la forma en que se almacenan los datos en la computadora.

Los modelos de datos sirven para clasificar los distintos tipos de SGBD.

Los modelos más conocidos y utilizados son:

* Modelo entidad-relación
* Modelo jerárquico
* Modelo de red
* Modelo relacional

Entidad – relación

Denominado por sus siglas como: E-R; Este modelo representa a la realidad a través de ***entidades***, que son objetos que existen y que se distinguen de otros por sus características, por ejemplo: un alumno se distingue de otro por sus características particulares como lo es el nombre, o el número de control asignado al entrar a una institución educativa, así mismo, un empleado, una materia, etc. Las entidades pueden ser de dos tipos:

Tangibles:

Son todos aquellos objetos físicos que podemos ver, tocar o sentir.

Intangibles:

Todos aquellos eventos u objetos conceptuales que no podemos ver, aun sabiendo que existen, por ejemplo:

la entidad materia, sabemos que existe, sin embargo, no la podemos visualizar o tocar.

Las características de las entidades en base de datos se llaman ***atributos,*** por ejemplo el nombre, dirección teléfono, grado, grupo, etc. son atributos de la entidad alumno; Clave, número de seguro social, departamento, etc., son atributos de la entidad empleado. A su vez una entidad se puede asociar o relacionar con más entidades a través de ***relaciones.***

Pero para entender mejor esto, veamos un ejemplo:

Consideremos una empresa que requiere controlar a los vendedores y las ventas que ellos realizan; de este problema determinamos que los objetos o entidades principales a estudiar son el empleado (vendedor) y el artículo (que es el producto en venta), y las características que los identifican son:

Empleado: Artículo:

Nombre Descripción

Puesto Costo

Salario Clave

R.F.C.

La relación entre ambas entidades la podemos establecer como Venta.

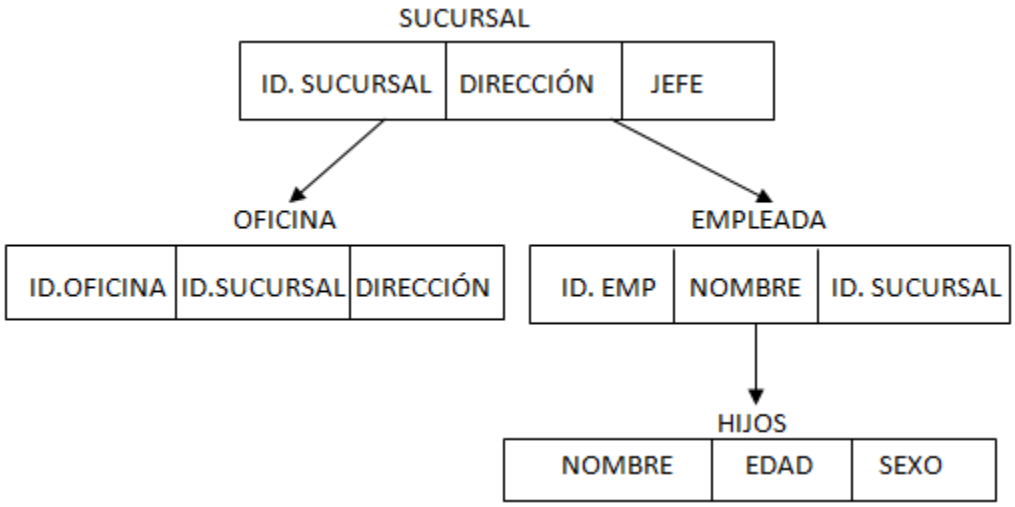
Existen más aspectos a considerar con respecto a los modelos entidad relación, estos serán considerados en el Modelo Entidad Relación.

Jerárquico

Este modelo utiliza árboles para la representación lógica de los datos.

Este árbol esta compuesto de unos elementos llamados nodos. El nivel más alto del árbol se denomina raíz. Cada nodo representa un registro con sus correspondientes campos.

La representación gráfica de este modelo se realiza mediante la creación de un árbol invertido, los diferentes niveles quedan unidos mediante relaciones.



En este modelo solo se pueden representar relaciones 1:M, por lo que presenta varios inconvenientes:

• No se admiten relaciones N:M

• Un segmento hijo no puede tener más de un padre.

• No se permiten más de una relación entre dos segmentos.

• Para acceder a cualquier segmento es necesario comenzar por el segmento raíz

• El árbol se debe de recorrer en el orden designado.

De red

En este modelo las entidades se representan como nodos y sus relaciones son las líneas que los unen. En esta estructura cualquier componente puede relacionarse con cualquier otro.

A diferencia del modelo jerárquico, en este modelo, un hijo puede tener varios padres.

Los conceptos básicos en el modelo en red son:

• El tipo de registro, que representa un nodo.

• Elemento, que es un campo de datos.

• Agregado de datos, que define un conjunto de datos con nombre.

Este modelo de datos permite representar relaciones N:M

Aquí se representa los datos mediante colecciones de registros y sus relaciones se representan por medio de ligas o enlaces, los cuales pueden verse como punteros. Los registros se organizan en un conjunto de gráficas arbitrarias.

Relacional

Este modelo es el más utilizado actualmente ya que utiliza tablas bidimensionales para la representación lógica de los datos y sus relaciones.

Algunas de sus principales características son:

• Puede ser entendido y usado por cualquier usuario.

• Permite ampliar el esquema conceptual sin modificar las aplicaciones de gestión.

• Los usuarios no necesitan saber donde se encuentran los datos físicamente.

El elemento principal de este modelo es la relación que se representa mediante una tabla.

En este modelo se representan los datos y las relaciones entre estos, a través de una colección de tablas, en las cuales los renglones (tuplas) equivalen a los cada uno de los registros que contendrá la base de datos y las columnas corresponden a las características(atributos) de cada registro localizado en la tupla;

Existen más aspectos a considerar con respecto a los modelos entidad relación, estos serán considerados en el tema de Modelo Relacional.

UNIDAD 4: LABORATORIO DE BASES DE DATOS

Para el desarrollo de esta unidad se cargará los gestores de bases de datos MySql y Postgres en arquitectura CLIENTE-SERVIDOR y DISTRIBUIDA, el caso de estudio seleccionado para la implementación de la base de datos y sobre el cual se realizará la gestión de la información, es el siguiente:

> La UNIDAD DE INVESTIGACION de la UCE dispone de un sistema de ficheros en el que almacena la información sobre los proyectos financiados que llevan a cabo los grupos de investigación de la universidad. A continuación se describe la información que contienen los ficheros que tienen que ver solamente con las convocatorias de ayudas públicas. El fichero de convocatorias mantiene información sobre las convocatorias de ayudas para la realización de proyectos de investigación. De éstas se guarda la fecha de publicación, el organismo que la promueve, el programa en que se enmarca el proyecto, la fecha límite de presentación de solicitudes, el número de la convocatoria (es único dentro de cada programa), la dirección de la web en donde obtener información sobre ella y el número del BOLETIN OFICIAL en donde se ha publicado. También se guarda la fecha de resolución, que es el día en que se ha publicado la lista de solicitudes que han sido aprobadas. De cada organismo se guarda, en otro fichero, el nombre, la

dirección, la población, el código postal y el teléfono. El fichero de solicitudes almacena los datos de las solicitudes que los grupos de investigación presentan para las distintas convocatorias de ayudas para proyectos. De cada solicitud se guarda información sobre la convocatoria a la que corresponde: organismo, programa, número y fecha. Además, se guarda la fecha en que se ha presentado esta solicitud, el título del proyecto (que será único), el nombre del investigador principal y su departamento. Cuando se publica la resolución, también se guarda la fecha de ésta y, en caso de ser aprobada la solicitud, se señala. Otros datos que aparecen en este fichero son: el importe económico que se solicita para llevar a cabo el proyecto, los nombres de los miembros del grupo de investigación que van a participar en el proyecto y las horas por semana que cada uno va a dedicar al mismo, que pueden ser distintas para cada investigador ya que pueden estar participando a la vez en otros proyectos. Además, se guardan las fechas previstas de inicio y finalización del proyecto, su duración en meses y por último, el número de entrada que ha dado el registro general a la solicitud. En la futura base de datos se desea reflejar también los grupos de investigación de la universidad, con su nombre, el investigador responsable y los investigadores que lo integran. De éstos se conoce el nombre, departamento y área de conocimiento dentro del departamento. Se considera que un grupo de investigación pertenece al departamento de su investigador responsable, aunque algunos de sus miembros pueden pertenecer a otro departamento. De los departamentos también se desea conocer el nombre de su director.

B. Base de Consulta

TÍTULO	AUTOR	EDICIÓN	AÑO	IDIOMA	EDITORIAL
BASE DE DATOS, DISEÑO, IMPLEMENTACIÓN Y ADMINISTRACIÓN	Carlos Coronel, Steven Morris y Peter Rob	1ra Edición	2011	Español	Cengage Learning
FUNDAMENTOS DE SISTEMAS DE BASES DE DATOS	Elmasri, R.A & Navathe, S.B.	2da. Edición	2002	Español	Addison Wesley
SISTEMAS DE BASES DE DATOS	Conolly, Begg.		2005	Español	

C. Base práctica con ilustraciones

4. ESTRATEGIAS DE APRENDIZAJE

ESTRATEGIA DE APRENDIZAJE 1: Análisis y Planeación
Descripción: Discusión de lecturas y artículos sobre las metodologías de diseño de bases de datos relacionales. Exposición conceptos y fundamentos de sistemas de gestión de bases de datos Estudio de casos, implementación de bases de datos relacionales. en MySql para gestión de transacciones y control de concurrencia. Estudio y trabajo en grupo, para el desarrollo de talleres de aplicación con reconstrucción de datos y seguridad (control de usuarios y control de accesos) en un el SGBD ejemplificado Estudio y trabajo autónomo para complementar los temas revisados en el aula referentes a la modelos de bases de datos avanzados
Ambiente(s) requerido: Aula amplia con buena iluminación.
Material (es) requerido: Proyector Pizarrón, marcadores. Computador Programas informáticos: Windows o Linux , Xampp, PostgreSql, entorno IDE y Plataforma virtual de aprendizaje
Docente: Profesional con título en Ingeniería de Sistemas o afines, y experiencia en administración y construcción de Bases de Datos

5. ACTIVIDADES

- Exposiciones, diálogos o discusiones
- Lecturas comentadas
- Taller práctico individual

- Talleres en grupo
- Evaluación escrita y práctica

Se presenta evidencia física y digital con el fin de evidenciar en el portafolio de cada aprendiz su resultado de aprendizaje. Este será evaluable y socializable

6. EVIDENCIAS Y EVALUACIÓN

Tipo de Evidencia	**Descripción (de la evidencia)**
De conocimiento:	Ejercicios prácticos en laboratorio para la administración de una base de datos
Desempeño:	Prácticas individuales para el diseño y construcción de bases de datos
De Producto:	Gestión de información almacenada en una base de datos cliente-servidor y en bases de datos distribuidas
Criterios de Evaluación (Mínimo 5 Actividades por asignatura)	Ensayos Conversatorios Talleres individuales Evaluaciones prácticas Evaluación escrita

Elaborado por: **Ing. Paulina Jaramillo**	**Revisado Por:** **(Coordinador)**	**Reportado Por:** **(Vicerrector)**

INSTITUTO SUPERIOR TECNOLÓGICO

JAPÓN

Amor al conocimiento

Guía Metodológica de Base de datos II
Carrera de Desarrollo de Software
Lcda. Paulina Jaramillo
2020

Coordinación editorial general:
Mgs. Milton Altamirano Pazmiño
Ing. Alexis Benavides Vinueza
Mgs. Lucia Begnini Dominguez

Diagramación:
Sebastián Gallardo Ramírez

Corrección de Estilo:
Mgs. Lucia Begnini Dominguez

Diseño:
Sebastián Gallardo Ramírez

Instituro superior tecnológico Japón
AMOR AL CONOCIMIENTO