Base de datos

Una **base de datos** tiene algún origen del que se derivan los datos, algún grado de interacción con eventos del mundo real y un público que está activamente interesado en su contenido. Los usuarios finales de una base de datos pueden efectuar transacciones comerciales o se pueden producir unos eventos que provoquen un cambio en la información almacenada en la base de datos. Al objeto de que una base de datos sea en todo momento precisa y fiable, debe ser un reflejo exacto de lo que representa; por consiguiente, en la base de datos deben reflejarse los cambios tan pronto como sea posible. Una base de datos puede ser de cualquier tamaño y complejidad.

Un **sistema de administración de datos** (DBMS, database management system) es una colección de programas que permite a los usuarios crear y mantener una base de datos. El DBMS es un sistema de software de propósito general que facilita los procesos de definición, construcción, manipulación y compartición de bases de datos entre varios usuarios y aplicaciones.

Definir una base de datos implica especificar los tipos de datos, estructuras y restricciones de los datos que se almacenarán en la base de datos. La definición o información descriptiva de una base de datos también se almacena en esta última en forma de catálogo o diccionario de la base de datos; es lo que se conoce como metadatos.

La **construcción** de la base de datos es el proceso consistente en almacenar los datos en algún medio de almacenamiento controlado por el DBMS. La **manipulación** de una base de datos incluye funciones como consultas para recuperar datos específicos, actualizar la base de datos para reflejar los cambios y generar informes a partir de los datos. **Compartir** una base de datos permite que varios usuarios y programas accedan a la base de datos de forma simultánea.

Una **aplicación** accede a la base de datos enviando consultas o solicitudes de datos al DBMS. Una **consulta** normalmente provoca la recuperación de algunos datos; una **transacción** puede provocar la lectura o la escritura de algunos datos en la base de datos.

Otras funciones importantes ofrecidas por el DBMS son la protección de la base de datos y su mantenimiento durante un largo periodo de tiempo. La **protección** incluye la protección del sistema contra el funcionamiento defectuoso del hardware o el software (caídas) y la protección de la seguridad contra el acceso no autorizado o malintencionado. Una gran base de datos típica puede tener un ciclo de vida de muchos años, por lo que el DBMS debe ser capaz de mantener el sistema de bases de datos permitiendo que el sistema evolucione según cambian los requisitos con el tiempo.

Una característica fundamental de la metodología de bases de datos es que el sistema de bases de datos no sólo contiene la propia base de datos, sino también una completa definición o descripción de la estructura de la base de datos y sus restricciones. Esta definición se almacena en el **catálogo DBMS**, que contiene información como la estructura de cada archivo, el tipo y el formato de almacenamiento de cada elemento de datos, y distintas restricciones de los datos. La información almacenada en el catálogo se denomina metadatos y describe la estructura de la base de datos principal. Debe referirse al catálogo para conocer la estructura de los archivos de una base de datos específica, como el tipo y el formato de los datos a los que accederá. El software DBMS debe funcionar igual de bien con cualquier cantidad de aplicaciones de bases de datos, siempre y cuando la definición de la base de datos esté almacenada en el catálogo

Ventajas de utilizar una metodología DBMS:

- Control de la redundancia
- Restricción del acceso no autorizado
- Almacenamiento persistente para los objetos del programa
- Suministro de estructuras de almacenamiento para un procesamiento eficaz de las consultas
- Copia de seguridad y recuperación
- Suministro de varias interfaces de usuario
- Representación de relaciones complejas entre los datos
- Implementación de las restricciones de integridad

En una **arquitectura DBMS** cliente/servidor básica, la funcionalidad del sistema se distribuye entre dos tipos de módulos. El **módulo cliente** manipula la interacción del usuario y proporciona interfaces amigables para el usuario, como formularios o GUIs basadas en menús. El **módulo servidor**, manipula normalmente el almacenamiento de los datos, el acceso, la búsqueda y otras funciones.

Arquitectura de tres niveles

El objetivo es separar las aplicaciones de usuario y las bases de datos físicas. En esta arquitectura se pueden definir esquemas en los siguientes tres niveles:

1. El **nivel interno** utiliza un modelo de datos físico y describe todos los detalles del almacenamiento de datos y las rutas de acceso a la base de datos.
2. El **nivel conceptual** oculta los detalles de las estructuras de almacenamiento físico y se concentra en describir las entidades, los tipos de datos, las relaciones, las operaciones de los usuarios y las restricciones.
3. El **nivel de vista o externo** un esquema externo describe la parte de la base de datos en la que un grupo de usuarios en particular está interesado y le oculta el resto de la base de datos.

Independencia de los datos

La capacidad de cambiar el esquema en un nivel de un sistema de bases de datos sin tener que cambiar el esquema en el siguiente nivel más alto. Se pueden definir dos tipos de independencia de datos:

1. **Independencia lógica de datos**: es la capacidad de cambiar el esquema conceptual sin tener que cambiar los esquemas externos o los programas de aplicación. Es posible cambiar el esquema conceptual para expandir la base de datos (añadiendo un tipo de registro o un elemento de datos), para cambiar las restricciones o para reducir la base de datos (eliminando un tipo de registro o un elemento de datos). En el último caso, no deben verse afectados los esquemas externos que sólo se refieren a los datos restantes.

La independencia lógica de datos es muy difícil de conseguir porque permite los cambios estructurales y restrictivos sin afectar a los programas de aplicación (un requisito mucho más estricto).

2. **Independencia física de datos**: es la capacidad de cambiar el esquema interno sin que haya que cambiar el esquema conceptual. Por tanto, tampoco es necesario cambiar los esquemas externos. Puede que haya que realizar cambios en el esquema interno porque algunos archivos físicos fueron reorganizados (por ejemplo, por la creación de estructuras de acceso adicionales) de cara a mejorar el rendimiento de las recuperaciones o las actualizaciones. Si en la base de datos permanecen los mismos datos que antes, no hay necesidad de cambiar el esquema conceptual.

Por regla general, la independencia física de datos existe en la mayoría de las bases de datos y de los entornos de archivos en los que al usuario se le ocultan la ubicación exacta de los datos en el disco, los detalles hardware de la codificación del almacenamiento, etcétera.

Lenguajes DBMS

En muchos DBMS donde no se mantiene una separación estricta de niveles se utiliza el **lenguaje de definición de datos** (**DDL**, data definition language), para definir los dos esquemas. El DBMS tendrá un compilador DDL cuya función es procesar las sentencias DDL a fin de identificar las descripciones de las estructuras del esquema y almacenar la descripción del mismo en el catálogo del DBMS.

Una vez compilados los esquemas de la base de datos y rellenada ésta con datos, los usuarios deben disponer de algunos medios para manipularla. Entre las manipulaciones típicas podemos citar la recuperación, la inserción, el borrado y la modificación de datos. El DBMS proporciona un conjunto de operaciones o **lenguaje de manipulación de datos** (**DML**, data manipulation language) para todas estas tareas.

Bases de datos relacionales

Las bases de datos relacionales se basan en el modelo relacional y usan un conjunto de tablas para representar tanto los datos como las relaciones entre ellos.

1. **Tablas**: Cada tabla tiene varias columnas, y cada columna tiene un nombre único. Cada tabla contiene registros de un tipo dado. Cada tipo de registro define un número fijo de campos, o atributos. Las columnas de la tabla se corresponden con los atributos del tipo de registro.
2. **Claves**: Una *superclave* es un conjunto de uno o varios atributos que, considerados conjuntamente, permiten identificar de manera unívoca una tupla de la relación. Si C es una superclave, entonces también lo es cualquier superconjunto de C. Las superclaves mínimas se denominan *claves candidatas*, es posible que varios conjuntos diferentes de atributos puedan ejercer como claves candidatas.

   La *clave primaria* denota una clave candidata que ha elegido el diseñador de la base de datos como medio principal para la identificación de las tuplas de una relación. La clave primaria debe escogerse de manera que los valores de sus atributos no se modifiquen nunca, o muy rara vez.

   Las claves (sean primarias, candidatas o superclaves) son propiedades de toda la relación, no de cada una de las tuplas. Ninguna pareja de tuplas de la relación puede tener simultáneamente el mismo valor de los atributos de la clave. La selección de una clave representa una restricción.

   La *clave foránea* o foreign key (clave externa) especifica una restricción de integridad referencial entre dos esquemas de relación R1 y R2. Un conjunto de atributos FK en una relación R1 es una foreign key de R1 que referencia a la relación R2 si satisface las siguientes reglas:

   1. Los atributos en FK tienen el mismo dominio, o dominios, que los atributos de clave principal PK de R2; se dice que los atributos FK referencian o hacen referencia a la relación R2.
   2. Un valor de FK en una tupla t1 del estado actual r1(R1) tampoco aparece como valor de PK en alguna tupla t2 del estado actual r2(R2) o es NULL. En el caso anterior, tenemos que t1[FK] = t2[PK], y decimos que la tupla t1 referencia o hace referencia a la tupla t2.

3. **Valores nulos**: Se permite que los atributos de las claves foráneas sean valores nulos, siempre que no hayan sido declarados no nulos previamente. Si todas las columnas de una clave foránea son no nulas en una tupla dada, se utiliza para esa tupla la definición habitual de las restricciones de clave foránea.
   Las restricciones de integridad declaran que el valor de ninguna clave principal puede ser NULL. Si se permitiera este valor, significa que no se podrían identificar ciertas tuplas. Las restricciones de clave y las de integridad de entidad se especifican en relaciones individuales.
4. **Restricciones de integridad referencial**: la *integridad referencial* garantiza que el valor que aparece en una relación para un conjunto dado de atributos aparezca también para un conjunto determinado de atributos en otra relación.
   Se refieren a que se debe hacer con las claves foráneas cuando se actualiza o se elimina la clave primaria y/o candidata a la que hace referencia:
   - *on delete/update cascade* : se eliminan/actualizan en cascada todas las claves foráneas que son iguales a la clave primaria/candidata
   - *on delete/update set to null* : se ponen en nulo todas las claves foráneas que son iguales a la clave primaria/candidata (en la definición la clave foránea debe aceptar valores nulos)
   - *on delete/update set defaul*t : se ponen en el valor por omisión definidos todas las claves foráneas que son iguales a la clave primaria/candidata

Restricciones de integridad

Los nombres de todas las restricciones dentro de un esquema particular deben ser únicos, se utilizan para identificar una restricción particular en caso de que la restricción tenga que eliminarse más tarde y sustituirse por otra restricción. La asignación de nombres a las restricciones es opcional.

Las restricciones de integridad garantizan que las modificaciones realizadas en la base de datos por los usuarios autorizados no den lugar a una pérdida de la consistencia de los datos. Por tanto, las restricciones de integridad protegen contra daños accidentales a las bases de datos.

Por tanto, la mayor parte de los sistemas de bases de datos permiten especificar restricciones de integridad que puedan probarse con una sobrecarga mínima.

1. **Restricción not null:** prohíbe la inserción de valores nulos para ese atributo. Cualquier modificación de la base de datos que haga que se inserte un valor nulo en un atributo declarado como not null genera un diagnóstico de error. También puede aplicarse a declaraciones de dominio definidas por los usuarios; en consecuencia, no se permitirá a los atributos de ese tipo de dominio tomar valores nulos.
2. **Restricción unique**: indica que los atributos forman una clave candidata; es decir, ningún par de tuplas de la relación puede ser igual en todos los atributos de la clave primaria. Sin embargo, se permite que los atributos de la clave candidata tengan valores nulos, a menos que se hayan declarado de manera explícita como not null.
3. **Restricción check**: cuando se aplica a declaraciones de relaciones, la cláusula check (P) especifica un predicado P que deben cumplir todas las tuplas de la relación. Cuando se aplica a un dominio, la cláusula check permite a los diseñadores de esquemas especificar un predicado que debe cumplir cualquier valor asignado a las variables cuyo tipo de datos sea el dominio.
4. **Integridad referencial**: a menudo se desea garantizar que el valor que aparece en una relación para un conjunto dado de atributos aparezca también para un conjunto

determinado de atributos en otra relación. Las claves foráneas pueden especificarse como parte de la instrucción create table mediante la cláusula foreign key.
Cuando se *viola una restricción* de integridad referencial, el procedimiento normal es rechazar la acción que ha causado esa violación (es decir, la transacción que lleva a cabo la acción de actualización se retrocede). Sin embargo, la cláusula foreign key puede especificar que si una acción de borrado o de actualización de la relación a la que hace referencia viola la restricción, entonces, en lugar de rechazar la acción, el sistema realice los pasos necesarios para modificar la tupla de la relación que hace la referencia para que se restaure la restricción.
Sentencias DML permitidas por las acciones de integridad referencial cuando se ejecutan sobre claves primarias/únicas de la tabla padre, o sobre claves foráneas de la tabla hijo.

Sentencia DML	**Ejecutada en la tabla padre**	**Ejecutada en la tabla hijo**
INSERT	Sin problemas si los valores de la clave son únicos	Sin problemas si el valor de la clave foránea existe en la tabla padre o es parcial o totalmente nula
UPDATE	Permitida si la sentencia no deja alguna fila hijo sin su correspondiente valor en la tabla padre	Permitida si la nueva clave en la tabla foránea tiene su valor correspondiente en la tabla padre
DELETE Restrict	Permitida si ninguna fila de la tabla hijo referencia el valor de la clave en la tabla padre	Sin problemas
DELETE Cascade	Sin problemas	Sin problemas

Modelo Entidad-Relación

Objetos básicos

1- **Entidad** es una “cosa” u “objeto” del mundo real que es distinguible de todos los demás objetos, tiene un conjunto de propiedades, y los valores de algún conjunto de propiedades pueden identificar cada entidad de forma unívoca. Las entidades pueden ser concretas, como las personas, o abstractas, como los conceptos. Un conjunto de entidades es un conjunto de entidades del mismo tipo que comparten las mismas propiedades, o atributos.

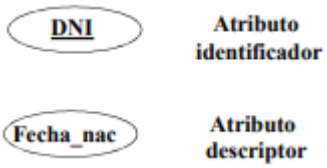
Un conjunto de entidades que no tiene suficientes atributos para formar una clave primaria se denomina conjunto de entidades débiles. Un conjunto de entidades que tiene una clave primaria se denomina conjunto de entidades fuertes.

Los conjuntos de entidades de nivel superior e inferior también se pueden denominar con los términos superclase y subclase, respectivamente. El conjunto de entidades persona es la superclase de las subclases cliente y empleado. Los conjuntos de entidades de nivel inferior (o subclases) también heredan la participación en los conjuntos de relaciones en los que participa su entidad de nivel superior (o superclase).

2- **Atributos** son propiedades descriptivas que posee cada miembro de un conjunto de entidades. La designación de un atributo para un conjunto de entidades expresa que la base de datos almacena información parecida relativa a cada entidad del conjunto de entidades; sin embargo, cada entidad puede tener su propio valor para cada atributo.

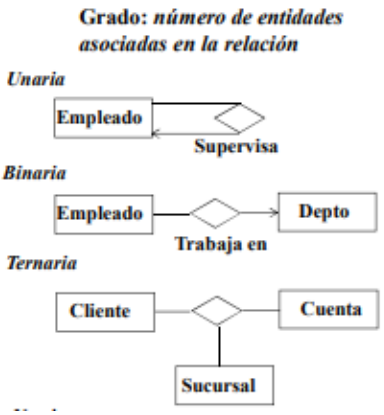
Para cada atributo hay un conjunto de valores permitidos, denominados dominio o conjunto de valores de ese atributo.

Un identificador se emplea para determinar de manera única una instancia de una entidad o relación de otra. Un descriptor se emplea para especificar una característica no-única de una instancia de una entidad



3- **Relación** es una asociación entre varias entidades. Un conjunto de relaciones es un conjunto de relaciones del mismo tipo.

La función que desempeña una entidad en una relación se denomina **rol** de esa entidad. Como los conjuntos de entidades que participan en un conjunto de relaciones, generalmente, son distintos, los roles están implícitos y no se suelen especificar. Sin embargo, resultan útiles cuando el significado de una relación necesita aclaración. Tal es el caso cuando los conjuntos de entidades de una relación no son distintos; es decir, el mismo conjunto de entidades participa en un conjunto de relaciones más de una vez, con diferentes roles.



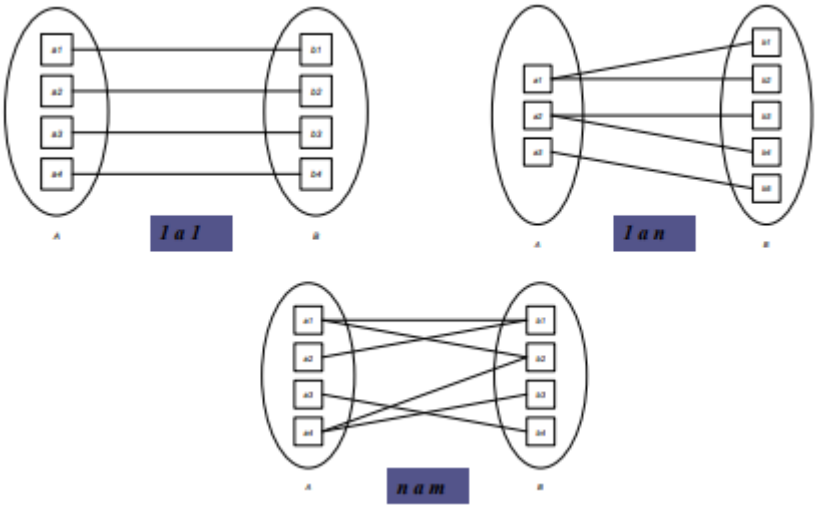
Relaciones binarias: Esta relación existe cuando solo dos entidades se relacionan para compartir datos a través de los atributos, puede ser entre una entidad fuerte y otra débil o entre una entidad fuerte y otra fuerte. Esta relación es la más común entre las entidades y es la que mayormente se usa para relacionar las entidades.

Relaciones redundantes: aquellas que se emplean para representar el mismo concepto, se pueden establecer más de una relación entre las dos mismas entidades siempre y cuando tengan diferentes significados. En este caso no se consideran redundantes.

Relaciones ternarias y n-arias: se deben definir cuando no sea posible representar el mismo concepto por medio de relaciones binarias entre las entidades. También es posible representar la relación ternaria como un tipo de entidad regular introduciendo una clave artificial o sustituta.
La especialización y la generalización definen una relación de inclusión entre un conjunto de entidades de nivel superior y uno o más conjuntos de entidades de nivel inferior. La **especialización** es el resultado de tomar un subconjunto de un conjunto de entidades de nivel superior para formar un conjunto de entidades de inferior. La **generalización** es el resultado de tomar la unión de dos o más conjuntos distintos de entidades (de nivel inferior) para producir un conjunto de entidades de nivel superior. Los atributos de los conjuntos de entidades de nivel superior los heredan los conjuntos de entidades de nivel inferior.

Correspondencia de cardinalidades

Expresa el número de entidades a las que otra entidad se puede asociar mediante un conjunto de relaciones. La correspondencia de cardinalidades resulta muy útil para describir conjuntos de relaciones binarias, aunque pueda contribuir a la descripción de conjuntos de relaciones que impliquen más de dos conjuntos de entidades.
Para un conjunto de relaciones binarias R entre los conjuntos de entidades A y B, la correspondencia de cardinalidades debe ser una de las siguientes:
•**Uno a uno**: cada entidad de A se asocia, a lo sumo, con una entidad de B, y cada entidad en B se asocia, a lo sumo, con una entidad de A.
•**Uno a varios:** cada entidad de A se asocia con cualquier número (cero o más) de entidades de B. Cada entidad de B, sin embargo, se puede asociar, a lo sumo, con una entidad de A.
•**Varios a uno:** cada entidad de A se asocia, a lo sumo, con una entidad de B. Cada entidad de B, sin embargo, se puede asociar con cualquier número (cero o más) de entidades de A.
•**Varios a varios**: cada entidad de A se asocia con cualquier número (cero o más) de entidades de B, y cada entidad de B se asocia con cualquier número (cero o más) de entidades de A.



Restricciones a las generalizaciones
Un tipo de restricción implica la determinación de las entidades que pueden formar parte de un conjunto de entidades de nivel inferior dado. Esa pertenencia puede ser una de las siguientes:

1. **Definida por la condición**: en los conjuntos de entidades de nivel inferior definidos por la condición, la pertenencia se evalúa en función del cumplimiento de una condición o predicado explícito por la entidad.
2. **Definida por el usuario**: los conjuntos de entidades de nivel inferior definidos por el usuario no están restringidos por una condición de pertenencia; más bien, el usuario de la base de datos asigna las entidades a un conjunto de entidades dado.
3. **Disjuntos**: la restricción sobre la condición de disyunción exige que cada entidad no pertenezca a más de un conjunto de entidades de nivel inferior.
4. **Solapados**: en las generalizaciones solapadas la misma entidad puede pertenecer a más de un conjunto de entidades de nivel inferior de la generalización.
5. R**estricción de completitud** sobre una generalización o especialización, especifica si una entidad del conjunto de entidades de nivel superior debe pertenecer, al menos, a uno de los conjuntos de entidades de nivel inferior de la generalización o especialización. Esta restricción puede ser de uno de los tipos siguientes:

• **Generalización o especialización total**: cada entidad de nivel superior debe pertenecer a un conjunto de entidades de nivel inferior.
• **Generalización o especialización parcial:** puede que alguna entidad de nivel superior no pertenezca a ningún conjunto de entidades de nivel inferior.

Transformación de modelo E-R a tablas de una BD relacional

- Las **entidades fuertes** del modelo se transforman en una relación (tabla) que incluye todos los atributos de la entidad. Uno de sus atributos debe ser la clave primaria de la tabla.
- Las **entidades débiles** del modelo se transforman en una relación (tabla) que incluye todos los atributos de la entidad, más la clave de la entidad fuerte de la que depende, que debe incluirse como clave foránea. La clave primaria de la tabla es la clave de la entidad fuerte más los atributos de la entidad débil y la clave parcial (discriminador) de la entidad débil.
- Las **asociaciones binarias con cardinalidad 1:1** donde participan las entidades S y T, se debe elegir una de las entidades, S por ejemplo, e incluir como clave foránea en S, la clave primaria de T. Es preferible elegir como S a la entidad que participa en la asociación con existencia obligatoria (existencia 1). Los atributos propios de la asociación también se deben incluir en S. De la transformación resultan dos tablas correspondientes a cada una de las entidades que participan de la relación con el agregado de la clave foránea y los atributos de la asociación correspondientes.
- Las **asociaciones binarias con cardinalidad 1:N** donde participan las entidades S y T, se debe identificar la relación S del lado N de la asociación, e incluir como clave foránea de S la clave primaria de T. Los atributos propios de la asociación también se deben incluir en S. El resultado final es siempre dos tablas correspondientes a cada una de las entidades S y T que participan de la asociación, con el agregado de la clave foránea y los atributos propios correspondientes a la asociación.
- Las **asociaciones binarias con cardinalidad N:M** y para asociaciones con grado mayor a 2 (>2) se genera una nueva tabla para la relación donde se incluyen como claves foráneas las claves primarias de las entidades que participan de la asociación, su combinación constituirá la clave primaria de la tabla generada. Incluya también los atributos propios de la relación.
- Por cada **atributo multivaluado** A se debe crear una nueva tabla. A esta tabla se le debe agregar como clave foránea la clave primaria (K) de la entidad que posee el

atributo multivaluado A. La clave primaria de la tabla generada la constituirá la combinación de A y K.

Transformación en tablas de una generalización

1- **Vertical**: Crear una tabla para la entidad de nivel más alto, crear una tabla para c/u de las entidades de nivel más bajo incluyendo la clave de la entidad de nivel más alto como clave foránea.

2- **Horizontal**: No crear una tabla para la entidad de nivel más alto, en cambio crear una tabla para cada una de las entidades de nivel más bajo donde además de los atributos propios se incluya una columna por cada uno de los atributos de la entidad de nivel más alto

3- **Plano**: Crear una sola tabla que incluya a la entidad de nivel más alto y todas las entidades de nivel más bajo con todos los atributos de unas y otras, más un atributo t del tipo discriminador que me indique el tipo de entidad al que nos estamos refiriendo, la clave será la clave de la entidad del nivel más alto).

Archivos-Índices

Organización de archivos secuenciales

Los **archivos** se organizan lógicamente como secuencias de registros. Esos registros se corresponden con los bloques del disco. Los **archivos secuenciales** están diseñados para el procesamiento eficiente de los registros de acuerdo con un orden basado en alguna clave de búsqueda. La clave de búsqueda es cualquier atributo o conjunto de atributos; no tiene por qué ser la clave primaria, ni siquiera una superclave. Para permitir la recuperación rápida de los registros según el orden de la clave de búsqueda, éstos se vinculan mediante punteros. El puntero de cada registro señala al siguiente registro según el orden indicado por la clave de búsqueda. Además, para minimizar el número de accesos a los bloques en el procesamiento de los archivos secuenciales, los registros se guardan físicamente en el orden indicado por la clave de búsqueda, o lo más cercano posible.

Archivos secuenciales desordenados

Es el tipo de organización más sencillo y básico, según el cual los registros se guardan en el fichero en el mismo orden en que se insertan; es decir, los registros se insertan al final del fichero.

La **inserción** de un registro nuevo es muy eficaz. El último bloque de disco del fichero se copia en el búfer, se añade el registro nuevo y se reescribe el bloque de nuevo en el disco. En la cabecera del fichero se guarda la dirección del último bloque del fichero.

La **búsqueda** de un registro utilizando cualquier condición de búsqueda implica una búsqueda lineal, bloque a bloque, por todo el fichero (procedimiento muy costoso). Si sólo un registro satisface la condición de búsqueda, entonces, por término medio, un programa leerá en memoria y buscará en la mitad de los bloques antes de encontrar el registro. En el caso de un fichero de b bloques, esto requiere buscar de media en (b/2) bloques. Si ningún registro satisface la condición de búsqueda, el programa debe buscar en los b bloques del fichero.

Para **eliminar** un registro, un programa primero debe buscar su bloque, copiar el bloque en un búfer, eliminar el registro del búfer y, por último, reescribir el bloque en el disco. Esto deja espacio inutilizado en el bloque de disco. La eliminación de una gran cantidad de registros de este modo supone un derroche de espacio de almacenamiento. Otra técnica que se utiliza para borrar registros es contar con un byte o un bit extra, denominado marcador de borrado, en cada registro. Un registro se borra estableciendo el marcador de borrado a un determinado valor. Un valor diferente del marcador indica un registro válido (no borrado).

Estas dos técnicas de borrado requieren una **reorganización** periódica del fichero para reclamar el espacio inutilizado correspondiente a los registros borrados.

Para leer todos los registros en orden por los valores de algún campo, debemos crear una copia ordenada del fichero. La ordenación es una operación costosa si el fichero es grande, y para ella se utilizan técnicas especiales de **ordenación externa.**

Archivos secuenciales ordenados

Los registros de un fichero se pueden ordenar físicamente en el disco en función de los valores de uno de sus campos, denominado **campo de ordenación,** si el campo de ordenación también es un campo clave (campo que garantiza un valor exclusivo en cada registro) del fichero, entonces el campo se denomina **clave de ordenación** del fichero.

La **lectura** de los registros en el orden marcado por los valores de la clave de ordenación es extremadamente eficaz porque no se necesita una ordenación, encontrar el siguiente

registro al actual según el orden de la clave de ordenación normalmente no requiere acceder a bloques adicionales porque el siguiente registro se encuentra en el mismo bloque que el actual (a menos que el registro actual sea el último del bloque). El uso de una **condición de búsqueda** basándose en el valor de un campo clave de ordenación ofrece un acceso más rápido cuando se utiliza la técnica de búsqueda binaria, una mejora respecto a las búsquedas lineales, aunque no se utiliza habitualmente para los ficheros de disco. Una búsqueda binaria en ficheros de disco puede realizarse en los bloques en lugar de en los registros. Una búsqueda binaria normalmente accede a log2(b) bloques, se encuentre o no el registro.
La **ordenación** no ofrece ninguna ventaja para el acceso aleatorio u ordenado de los registros basándose en los valores de otros campos no ordenados del fichero. En estos casos, realizamos una búsqueda lineal para el acceso aleatorio. Para acceder a los registros en orden basándose en un campo desordenado, es necesario crear otra copia ordenada (con un orden diferente) del fichero.
La inserción y eliminación de registros son operaciones costosas para un archivo ordenado, porque los registros deben permanecer físicamente ordenados. Para **insertar** un registro debemos encontrar su posición correcta en el fichero, basándonos en el valor de su campo de ordenación, y después hacer espacio en el fichero para insertar el registro en esa posición. Si se trata de un fichero grande, esta operación puede consumir mucho tiempo porque, por término medio, han de moverse la mitad de los registros del fichero para hacer espacio al nuevo registro. Esto significa que deben leerse y reescribirse la mitad de los bloques del fichero después de haber movido los bloques entre sí. En la **eliminación** de un registro, el problema es menos grave si se utilizan marcadores de borrado y una reorganización periódica.

Archivos Hash o de acceso directo
Otro tipo de organización de ficheros está basado en la dispersión, que proporciona un acceso muy rápido a los registros bajo ciertas condiciones de búsqueda. Esta organización se denomina **fichero disperso** o **fichero hash**. La condición de búsqueda debe ser una condición de igualdad sobre un solo campo, denominado **campo de dispersión** o **campo hash**. En la mayoría de los casos, el campo de dispersión también es un campo clave del fichero, en cuyo caso se denomina **clave de dispersión** (o clave hash).
La idea que hay detrás de la dispersión es proporcionar una **función h**, denominada **función de dispersión** (o hash), que se aplica al valor del campo de dispersión de un registro y produce la dirección del bloque de disco en el que está almacenado el registro. Una **búsqueda** del registro dentro del bloque puede llevarse a cabo en un búfer de la memoria principal. En la mayoría de los registros, sólo necesitamos acceder a un bloque para recuperar el registro.
El problema con la mayoría de las funciones de dispersión es que no garantizan que valores distintos se dispersen a direcciones distintas, porque el espacio del campo de dispersión (el número de valores posibles que un campo de dispersión puede tomar) es normalmente mucho más grande que el espacio de direcciones (el número de direcciones disponibles para los registros). La función de dispersión mapea el espacio del campo de dispersión al espacio de direcciones.
Una **colisión** se produce cuando el valor del campo de dispersión de un registro que se está insertando se dispersa a una dirección que ya contiene un registro diferente. En esta situación, debemos insertar el registro nuevo en alguna otra posición, puesto que su

dirección de dispersión está ocupada. El proceso de encontrar otra posición se denomina resolución de colisiones. Hay varios métodos para resolver una colisión:

- **Direccionamiento abierto**: a partir de la posición ocupada especificada por la dirección de dispersión, el programa comprueba las posiciones subsiguientes en orden hasta encontrar una posición sin utilizar (vacía).
- **Encadenamiento**: para este método, se conservan varias ubicaciones de dispersión, normalmente extendiendo el array con algunas posiciones de desbordamiento. Adicionalmente, se añade un campo puntero a cada ubicación de registro. Una colisión se resuelve colocando el registro nuevo en una ubicación de desbordamiento sin utilizar y estableciendo el puntero de la ubicación de la dirección de dispersión ocupada a la dirección de esa ubicación de desbordamiento. Se conserva entonces una lista enlazada de registros de desbordamiento por cada dirección de dispersión.
- **Dispersión múltiple**: el programa aplica una segunda función de dispersión si la primera desemboca en una colisión. Si se produce otra colisión, el programa utiliza el desbordamiento abierto o aplica una tercera función de dispersión y utiliza después el direccionamiento abierto si es necesario.

Todo método de resolución de colisiones requiere sus propios algoritmos para insertar, recuperar y borrar registros. Los algoritmos de encadenamiento son los más sencillos. Los de borrado para el direccionamiento abierto son bastante difíciles. El objetivo de una buena función de dispersión es distribuir uniformemente los registros sobre el espacio de direcciones para minimizar las colisiones sin dejar demasiadas ubicaciones sin utilizar.

Archivos indexados

Es la organización de archivos que incluye índices en el almacenamiento de los registros; de esta forma nos será más fácil buscar un registro determinado sin necesidad de recorrer todo el archivo. Un campo (o un grupo de campos) del registro denominado clave es utilizado como campo de índice.

Implicancia del uso de índices:

1- La colocación de un listado al inicio del archivo: para la identificación del contenido.

2-La presentación de un segundo índice: para reflejar la información de cada punto principal del índice anterior.

3- La actualización de los índices: Cuando se insertan y eliminan archivos, es preciso actualizar los índices para evitar contratiempos actualizando un archivo.

4- La organización de un índice: Nos evita examinar archivo por archivo para recuperar algún registro buscado; por lo tanto ahorraríamos tiempo si tenemos una adecuado organización de los índices

Hay dos tipos básicos de índices:

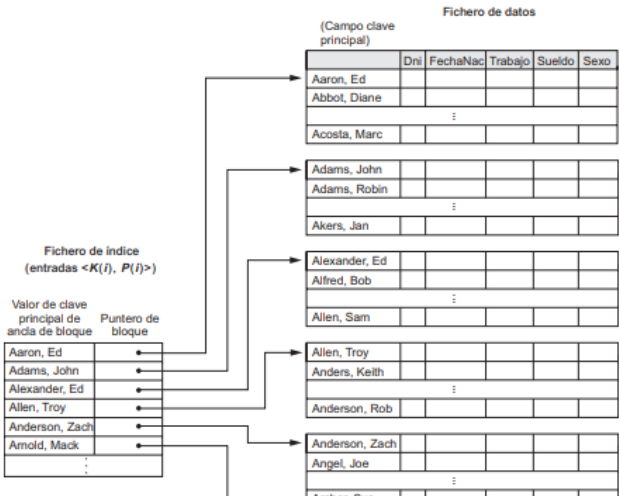
- **Índices ordenados:** están basados en una disposición ordenada de los valores.
- **Índices asociativos:** están basados en una distribución uniforme de los valores a través de una serie de cajones (buckets). El valor asignado a cada cajón está determinado por una función, llamada función de asociación (hash function).

Los atributos o conjunto de atributos usados para buscar en un archivo se llaman **claves de búsqueda**, usando el concepto de clave de búsqueda se determina que, si existen varios índices para un archivo, existirán varias claves de búsqueda.

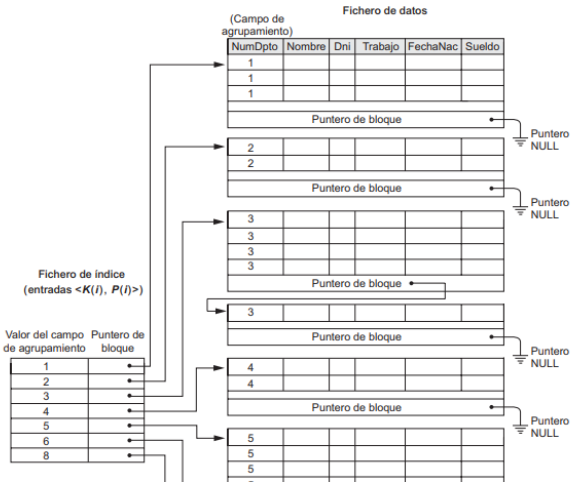
Índices de un sólo nivel

En un fichero con una estructura de registro dada compuesta por varios campos (o atributos), normalmente se define una estructura de índice con un solo campo del fichero,

que se conoce como **campo de indexación** (o atributo de indexación). Normalmente, el índice almacena todos los valores del campo de índice, junto con una lista de punteros a todos los bloques de disco que contienen los registros con ese valor de campo. Los valores del índice están ordenados, de modo que podemos hacer una búsqueda binaria en el índice. El fichero de índice es mucho más pequeño que el fichero de datos, por lo que la búsqueda en el índice mediante una búsqueda binaria es razonablemente eficaz. La indexación multinivel anula la necesidad de una búsqueda binaria a expensas de crear índices al propio índice.



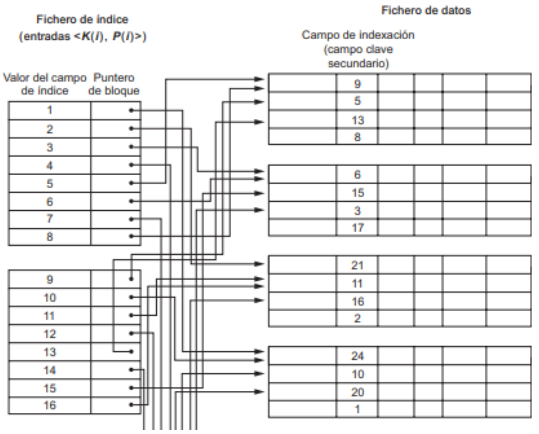
1- Índices principales: es un fichero ordenado cuyos registros son de longitud fija con dos campos. El primer campo es del mismo tipo de datos que el campo clave de ordenación (denominado clave principal) del fichero de datos, y el segundo campo es un puntero a un bloque del disco (una dirección de bloque). En el fichero índice hay una entrada de índice (o registro de índice) por cada bloque del fichero de datos. Cada entrada del índice tiene dos valores: el valor del campo clave principal para el primer registro de un bloque, y un puntero a ese bloque.
El número total de entradas del índice coincide con el número de bloques de disco del fichero de datos ordenado. El primer registro de cada bloque del fichero de datos es el registro ancla del bloque.
Los índices se pueden clasificar como densos o escasos. Un **índice denso** tiene una entrada de índice por cada valor de la clave de búsqueda (y, por tanto, cada registro) del fichero de datos. Un **índice escaso** (o no denso) sólo tiene entradas para algunos de los valores de búsqueda. Por consiguiente, un índice principal es un índice no denso, porque incluye una entrada por cada bloque de disco del fichero de datos y las claves de su registro ancla, en lugar de una entrada por cada valor de búsqueda (es decir, por cada registro). El fichero de índice para un índice principal necesita menos bloques que el fichero de datos, por dos razones. En primer lugar, hay menos entradas de índice que registros en el fichero de datos. En segundo lugar, cada entrada del índice tiene normalmente un tamaño más pequeño que un registro de datos, porque sólo tiene dos campos; en consecuencia, en un bloque entran más entradas de índice que registros de datos. Por tanto, una búsqueda binaria en el fichero de índice requiere menos accesos a bloques que una búsqueda binaria en el fichero de datos, si el fichero de índice principal contiene bi bloques, la localización de un registro con un valor de clave de búsqueda requiere una búsqueda binaria de ese índice y acceder al bloque que contiene ese registro: un total de log2b+1 accesos.



2- Índices agrupados: Si los registros del fichero están ordenados físicamente por un campo no clave (es decir, un campo que no tiene un valor distinto para cada registro), este campo se denomina campo agrupado. Un índice agrupado también es un fichero ordenado con dos campos; el primero es del mismo tipo que el campo agrupado del fichero de datos, y el segundo es un puntero a un bloque. En el índice agrupado hay una entrada por cada valor distinto del campo agrupado, que contiene el valor y un puntero al

primer bloque del fichero de datos que tiene un registro con ese valor para su campo agrupado.
La inserción y el borrado de un registro todavía provoca problemas, porque los registros de datos están ordenados físicamente. Para aliviar el problema de la inserción, se suele reservar un bloque entero (o un grupo de bloques contiguos) para cada valor del campo agrupado; todos los registros con ese valor se colocan en el bloque (o grupo de bloques). Esto hace que la inserción y el borrado sean relativamente directos. Un índice agrupado es un índice no denso porque tiene una entrada por cada valor distinto del campo de indexación, que no es una clave por definición y, por tanto, tiene valores duplicados en lugar de un valor único por cada registro del fichero.

3- Índices secundarios: proporciona un medio secundario de acceso a un fichero para el que ya existe algún acceso principal, puede ser un campo que es una clave candidata y tiene un valor único en cada registro, o puede ser una "no clave" con valores duplicados. El índice es un fichero ordenado con dos campos, el primero es del mismo tipo de datos que algún campo no ordenado del fichero de datos que es un campo de indexación. El segundo campo es un puntero de bloque o un puntero de registro.

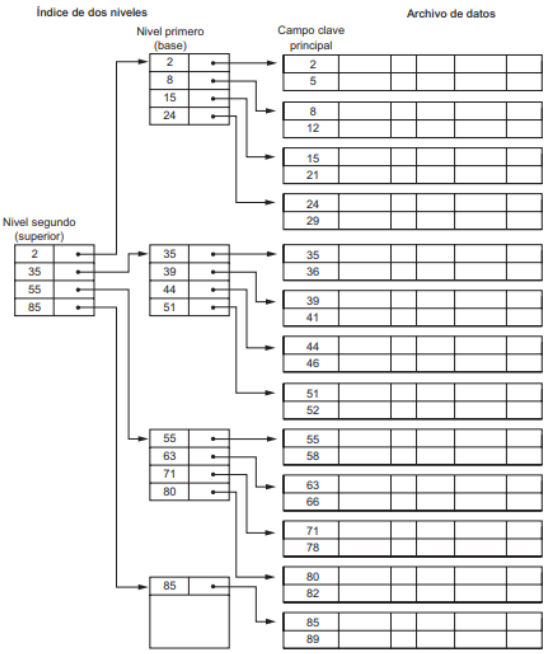


Puede haber muchos índices secundarios (y, por tanto, campos de indexación) para el mismo fichero. Primero consideramos que la estructura de acceso de un índice secundario en un campo clave tiene un valor distinto por cada registro. Dicho campo recibe a veces el nombre de clave secundaria. En este caso, hay una entrada de índice por cada registro del fichero de datos, que contiene el valor de la clave secundaria para el registro y un puntero al bloque en el que está almacenado el registro o al propio registro. Por tanto, dicho índice es denso. Como los registros del fichero de datos no están físicamente ordenados por los valores del campo clave secundario, no podemos utilizar las anclas de bloque. Por eso creamos una entrada de índice por cada registro del fichero de datos, en lugar de hacerlo por cada bloque, como en el caso de un índice principal. Normalmente necesita más espacio de almacenamiento y un tiempo de búsqueda mayor que un índice principal, debido a su mayor cantidad de entradas.
Un índice secundario proporciona una ordenación lógica de los registros por el campo de indexación. Si accedemos a los registros según el orden de las entradas del índice secundario, los obtendremos ordenados por el campo de indexación.

	Campo de índice utilizado para ordenar el fichero	**Campo de índice no usado para ordenar el fichero**
El campo de indexación es una clave	Índice principal	Índice secundario (clave)
El campo de indexación no es una clave	Índice agrupado	Índice secundario (no clave)

Tipo de índice	**Número de entradas de índice (primer nivel)**	**Denso no denso**
Principal	Número de bloques en el fichero de datos	No denso
Agrupado	Número de valores del campo de índice distintos	No denso
Secundario (clave)	Número de registros del fichero de datos	Denso
Secundario (no clave)	Número de registros[b] o número de valores del campo de índice distintos[c]	Denso o no denso

Índices multinivel



Un índice multinivel considera el fichero de índice, **primer nivel** (o base) de un índice multinivel, como un fichero ordenado con un valor distinto. Por consiguiente, podemos crear un índice principal para el primer nivel; este índice al primer nivel se denomina **segundo nivel** del índice multinivel. Como el segundo nivel es un índice principal, podemos utilizar anclas de bloque de modo que el segundo nivel tenga una sola entrada por cada bloque del primer nivel. El **factor de bloqueo** (bfri o fo) para el segundo nivel (y para todos los niveles subsiguientes) es el mismo que para el índice de primer nivel porque todas las entradas del índice tienen el mismo tamaño; cada una con un valor de campo y una dirección de bloque.

Cantidad de bloques:

- 1er. Nivel: r1/fo = b1, r1: valores de indexación diferentes
- 2do. Nivel: b1/fo = b2
- nivel n: (nro. de bloques del nivel n-1)/fo

Necesitamos un segundo nivel sólo si el primer nivel necesita más de un bloque de almacenamiento en disco y, de forma parecida, requerimos un tercer nivel sólo si el segundo nivel necesita más de un bloque. Repetimos el proceso hasta que todas las entradas del mismo nivel del índice “t” encajen en un solo bloque. Este bloque en el nivel t se denomina nivel de índice superior.

Cada nivel reduce el número de entradas en el nivel previo por un factor de fo, por tanto, un índice multinivel con r1 entradas de primer nivel tendrá aproximadamente t niveles, donde $t = log\ _{fo} r1$. El esquema multinivel puede utilizarse en cualquier tipo de índice, sea principal, agrupado o secundario (siempre y cuando el índice de primer nivel tenga valores distintos para K(i) y entradas de longitud fija). Acceder a los datos asociados con un valor de indexación tendrá tantos accesos como niveles tenga el índice+1. Un índice primario multinivel se denomina archivo secuencial-indexado. Se emplea un algoritmo de búsqueda que es similar ya sea que el índice sea denso como no denso.