

BASES DE DATOS

CONCEPTO

Base de Datos: es una colección de datos interrelacionados almacenados en conjunto, sin redundancias perjudiciales o innecesarias, su finalidad es servir a una aplicación o más, de la mejor manera posible; los datos se almacenan de forma que resulten independientes de los programas que los usan; se emplean métodos bien determinados para incluir nuevos datos y para modificar o extraer los datos almacenados.

Dato: son los hechos conocidos que se pueden grabar y que tienen un significado implícito.

PROPIEDADES DE LAS BASES DE DATOS

- Una base de datos representa algún aspecto del mundo real, lo que en ocasiones se denomina minimundo o **universo de discurso**. Los cambios introducidos en el minimundo se reflejan en la base de datos.
- Una base de datos es una colección de datos lógicamente coherente con algún tipo de significado inherente. No es correcto denominar base de datos a un conjunto de datos aleatorios.
- Una base de datos se diseña, construye y rellena con datos para un propósito específico.

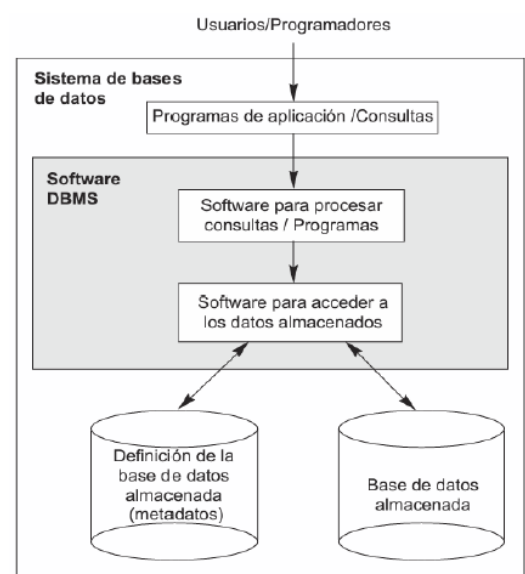
En otras palabras, una base de datos tiene algún origen del que se derivan los datos, algún grado de interacción de eventos del mundo real y un público que esta activamente interesado en su contenido. Además, puede ser de cualquier tamaño y complejidad. Se puede generar y mantener manualmente o estar computarizada. Una base de datos computarizada se puede crear y mantener con un grupo de aplicaciones escritas específicamente para esa tarea o mediante un sistema de administración de bases de datos.

SISTEMAS DE ADMINISTRACIÓN DE BASES DE DATOS (DBMS)

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 base de datos entre varios usuarios y aplicaciones.

FUNCIONES DEL DBMS

- **Definir** una base de datos implica especificar los tipos de datos, estructuras y restricciones de los datos que se almacenaran 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, llamada metadatos. (DDL)
- **Construcción** de la base de datos es el proceso consistente en almacenar los datos en algún medio de almacenamiento controlado por el DBMS.
- **Manipulación** de una base de datos incluye funciones como la consulta de la base de datos para recuperar datos específicos, actualizar la base de datos para reflejar los cambios introducidos en el minimundo y generar informes a partir de los datos. (DML)



- **Compartir**, una base de datos permite que varios usuario y programas accedan a la base de datos de forma simultánea.

OTRAS FUNCIONES

Protección: incluye la protección del sistema contra el funcionamiento defectuoso del hardware o el software, y la protección de la seguridad contra el acceso no autorizado o malintencionado.

Mantenimiento: 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 base de datos permitiendo que el sistema evolucione según cambian los requisitos con el tiempo.

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 de forma simultánea.

No es necesario utilizar software DBMS de propósito general para implementar una base de datos computarizada. Podríamos escribir nuestro propio conjunto de programas para crear y mantener la base de datos; en realidad podríamos crear nuestro propio software DBMS de propósito especial. En cualquier caso, utilicemos o no un DBMS de propósito general, normalmente tenemos que implantar una cantidad considerable de software complejo. De hecho, la mayoría de los DBMS son sistemas de software muy complejos.

En general, un DBMS se encarga de: definición de datos (DDL), manipulación de datos (DML), gestión de la seguridad, mantener la integridad de los datos, respaldo y recuperación de la base de datos, manejar transacciones, permitir el acceso concurrente a la base de datos, suministrar diferentes vistas de usuarios, permitir crear estructuras de almacenamiento para procesamiento eficaz, facilitar carga de datos desde otros archivos, mantener estadísticas y permitir el monitoreo.

SISTEMAS DE BASES DE DATOS

Un **sistema de base de datos** es la combinación de hardware (equipos físicos), software (DBMS -back end-, aplicaciones desarrolladas -front end-), usuarios (DBA, usuarios finales, diseñadores, etc.), datos (tablas, relaciones).

Es útil describir la base de datos como una parte de una tarea más amplia conocida como sistema de información dentro de cualquier organización. El departamento de Tecnología de la Información de una empresa diseña y mantiene un sistema de información compuesto por varios computadores, sistemas de almacenamiento, aplicaciones y bases de datos. El diseño de una aplicación nueva empieza con una fase denominada definición de requisitos y análisis. Estos requisitos son documentados en detalle y transformados en un **diseño conceptual**. El diseño después se convierte en un **diseño lógico** que se puede expresar en un modelo de datos implementado en un DBMS comercial. La etapa final es el **diseño físico**, durante la que se proporcionan especificaciones suplementarias para el almacenamiento y acceso a la base de datos. El diseño de base de datos se implementa y rellena con datos reales y se realiza un mantenimiento continuado a fin de reflejar el estado del minimundo.

CARACTERÍSTICAS DE LA METODOLOGÍA DE BASES DE DATOS

Unas cuantas características distinguen la metodología de bases de datos de la metodología tradicional de programación con archivos. En el procesamiento tradicional de archivos, cada usuario define e implementa archivos necesarios para una aplicación concreta como parte de la programación de esa aplicación. Esto puede disminuir la integridad de los datos y aumentar redundancia en la definición y el almacenamiento de datos dando como resultado un derroche de espacio de almacenamiento y un mayor esfuerzo para mantener la integridad ("si se actualiza en un lado, hay que actualizar en todos").

En la metodología de bases de datos se mantiene un único almacén de datos, que se define una sola vez y a la que acceden varios usuarios. En los sistemas de archivos cada aplicación tiene libertad para asignar un nombre independientemente a los elementos de datos. Por el contrario, en una base de

datos, los nombres o etiquetas de los datos se definen una vez y son utilizados por consultas, transacciones y aplicaciones.

Las principales características de la metodología de bases de datos frente a la metodología del procesamiento de archivos son las siguientes:

1. Naturaleza autodescriptiva de un sistema de bases de datos
2. Aislamiento entre programas y datos, y abstracción de datos
3. Soporte de vistas de los datos
4. Compartición de datos y procesamiento de transacciones multiusuario

NATURALEZA AUTO DESCRIPTIVA DE UN SISTEMA DE BASE DE DATOS

El sistema de base de datos no solo contiene la propia base de datos sino una completa definición 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 un catálogo se denomina **metadatos** y describe la estructura de la base de datos principal.

El software DBMS y los usuarios de la base de datos utilizan el catálogo cuando necesitan información sobre la estructura de la base de datos. Un paquete de software DBMS de propósito general no se escribe para una aplicación de base de datos específica. Por consiguiente, 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 este almacenada en el catálogo.

El software DBMS y los usuarios de la base de datos utilizan el catalogo cuando necesitan información sobre la estructura de la base de datos.

En el procesamiento de archivos tradicional, normalmente la definición de datos forma parte de los programas de aplicación. Así pues, esas aplicaciones están restringidas a trabajar sólo con una base de datos específica, cuya estructura está declarada en dichas aplicaciones. Mientras el software de procesamiento de archivos solo puede acceder a bases de datos específicas, el software DBMS puede acceder a distintas bases de datos extrayendo del catálogo las definiciones de las mismas y utilizando después esas definiciones.

AISLAMIENTO ENTRE PROGRAMAS Y DATOS, Y ABSTRACCIÓN DE DATOS

En el procesamiento de archivos tradicional, la estructura de los archivos de datos está incrustada en las aplicaciones, por lo que los cambios que se introducen en la estructura de un archivo pueden obligar a realizar cambios en todos los programas que acceden a ese archivo. Por el contrario, los programas que acceden a un DBMS no necesitan cambios en la estructura de programa, en la mayoría de los casos. La estructura de los archivos de datos se almacena en el catálogo DBMS, independientemente de los programas de acceso, esta propiedad se denomina **independencia de programa-datos**.

En algunos tipos de sistemas de bases de datos, como los sistemas orientados a objetos y los de objetos relacionales, los usuarios pueden definir operaciones sobre los datos como parte de las definiciones de la base de datos. Las aplicaciones de usuario pueden operar sobre los datos invocando estas operaciones por sus nombres y argumentos, independientemente de cómo estén implementadas las operaciones. Esto se denomina **independencia programa-operación**.

Las características que permite la independencia programa-datos y la independencia programa-operación se denomina **abstracción de datos**. Un DBMS proporciona a los usuarios una **representación conceptual** de los datos que no incluye muchos de los detalles de cómo están almacenados los datos o de cómo están implementadas las operaciones. Un **modelo de datos** es un tipo de abstracción de datos que se utiliza para proporcionar esa representación conceptual. El modelo de datos utiliza conceptos lógicos, como objetos, sus propiedades y sus relaciones, lo que para la mayoría de los usuarios es más

fácil de entender que los conceptos de almacenamiento en el computador. Por ello, el modelo de datos oculta los detalles del almacenamiento y de la implementación que no resultan interesantes a la mayoría de los usuarios de base de datos.

En la metodología de bases de datos, la estructura detallada y la organización de cada archivo se almacenan en el catálogo. Los usuarios de bases de datos y los programas de aplicación hacen referencia a la representación conceptual de los archivos y, cuando los módulos de acceso al archivo DBMS necesitan detalles sobre el almacenamiento del archivo, el DBMS los extrae del catálogo. Se pueden utilizar muchos modelos de datos para proporcionar esta abstracción de datos a los usuarios de bases de datos.

En las bases de datos orientadas a objetos y de objetos relacionales, el proceso de abstracción no solo incluye la estructura de datos sino también las operaciones sobre los datos. Estas operaciones proporcionan una abstracción de las actividades del minimundo normalmente entendidas por los usuarios. Dichas operaciones pueden ser invocadas por las consultas del usuario o por las aplicaciones, sin necesidad de conocer los detalles de cómo están implementadas esas operaciones. En este sentido, una abstracción de la actividad del minimundo queda a disposición de los usuarios como una **operación abstracta**.

SOPORTE DE VARIAS VISTAS DE LOS DATOS

Normalmente una base de datos tiene muchos usuarios, cada uno de los cuales puede necesitar una perspectiva o vista diferente de la base de datos. Una vista puede ser un subconjunto de la base de datos o puede contener datos virtuales derivados de los archivos de la base de datos pero que no están explícitamente almacenados (ej. datos calculados, como edades, importes parciales, etc.). Algunos usuarios no tienen la necesidad de preocuparse por si los datos a los que se refieren están almacenados o son derivados. Un DBMS multiusuario cuyos usuarios tienen variedad de diferentes aplicaciones debe ofrecer facilidades para definir varias vistas, que se realizan en función del rol del usuario.

COMPARTICIÓN DE DATOS Y PROCESAMIENTO DE TRANSACCIONES MULTIUSUARIO

Un DBMS multiusuario, debe permitir que varios usuarios puedan acceder a la base de datos al mismo tiempo. El DBMS debe incluir **software de control de concurrencia** para que esos varios usuarios que intentan actualizar los mismos datos, lo hagan de un modo controlado para que el resultado de la actualización sea correcto.

Estos tipos de aplicaciones se denominan, por lo general, **aplicaciones de procesamiento de transacciones en línea (OLTP, *online transaction processing*)**. Un papel fundamental es garantizar que las transacciones concurrentes operan correcta y eficazmente.

Una transacción es un programa en ejecución o un proceso que incluye uno o más accesos a la base de datos, como la lectura o la actualización de los registros de la misma. El DBMS debe implementar varias propiedades de transacción, como la de **aislamiento** que garantiza que parezca que cada transacción se ejecuta de forma aislada de otras transacciones, aunque puedan estarse ejecutando cientos de transacciones al mismo tiempo y la de **atomicidad** que garantiza que se ejecuten o todas o ninguna de las operaciones de bases de datos de una transacción.

SISTEMAS TRADICIONALES DE ARCHIVOS VS. SISTEMAS DE BASES DE DATOS

Sistemas tradicionales de ficheros

- Sistemas orientados a los procesos.
- Ficheros específicos para cada aplicación.
- Redundancia de datos.
- Inconsistencia de datos.
- Pérdida de espacio físico.
- Dependencia física de los datos.
- Dependencia lógica de los datos.

Sistemas de bases de datos

- Sistemas orientados a los datos.
- No son diseñados para una aplicación.
- Disminuyen la redundancia.
- Evitan inconsistencia.
- Mantener formato de datos.
- Independencia física de los datos.
- Independencia lógica de los datos.
- Compartir datos.
- Aplicar restricciones de seguridad.

- Mantener la integridad.

ACTORES DE LA ESCENA (USUARIOS) Y TRABAJADORES ENTRE BAMBALINAS

Los **actores de la escena** son las personas cuyos trabajos implican el uso diario de una base de datos grande. Los que trabajan en el mantenimiento del entorno del sistema de bases de datos pero que no están activamente interesados en la propia base de datos se los llamará **trabajadores entre bambalinas**.

ACTORES DE LA ESCENA

1. Administrador de las bases de datos (DBA).
2. Diseñador de bases de datos.
3. Usuarios finales.
 - 3.1. Usuarios finales casuales.
 - 3.2. Usuarios finales principiantes o paramétricos.
 - 3.3. Usuarios finales sofisticados.
 - 3.4. Usuarios finales independientes.
4. Analistas de sistemas y programadores de aplicaciones.

ADMINISTRADOR DE LAS BASES DE DATOS

En cualquier empresa donde muchas personas utilizan los mismos recursos, se necesita un administrador jefe que supervise y administre esos recursos. En un entorno de bases de datos, el recurso principal es la base de datos en sí misma, mientras que el recurso secundario es el DBMS y el software relacionado. La administración de estos recursos es responsabilidad del **administrador de la base de datos (DBA, database administrator)**, y es el responsable del acceso autorizado a la base de datos, de la coordinación y monitorización de uso, y de adquirir los recursos software y hardware necesarios. El DBA también es responsable de problemas como las brechas de seguridad o de unos tiempos de respuesta pobres. En las empresas grandes el DBA está asistido por un equipo de personas que llevan a cabo estas funciones. Define estructuras físicas de almacenamiento, especifica perfiles y normas de seguridad, monitorea rendimiento, implementa integridad, provee métodos de respaldo y recuperación.

DISEÑADORES DE BASES DE DATOS

Son los responsables de identificar que datos serán los que se almacenaran en la base de datos y de elegir las estructuras apropiadas para representar y almacenar esos datos. Es responsabilidad de los diseñadores comunicarse con todos los usuarios de la base de datos para conocer sus requisitos, a fin de crear un diseño que satisfaga sus necesidades. En muchos casos, los diseñadores forman parte de la plantilla del DBA y se les pueden asignar otras responsabilidades una vez completado el diseño de la base de datos. Deben interactuar con los grupos de usuarios potenciales y desarrollan **vistas** de la base de datos que satisfacen los requisitos de datos y procesamiento de esos grupos. El diseño final de la base de datos debe ser capaz de soportar los requisitos de todos los grupos de usuarios.

USUARIOS FINALES

Son las personas cuyos trabajos requieren acceso a la base de datos para realizar consultas, actualizaciones e informes. Se clasifican en:

- **Usuarios finales casuales:** acceden ocasionalmente a la base de datos, pero pueden necesitar una información diferente en cada momento. Usan un sofisticado lenguaje de consulta de bases de datos para especificar sus peticiones y normalmente son administradores de nivel medio o alto u otros usuarios interesados.
- **Usuarios finales principiantes o paramétricos:** su labor principal gira en torno a la consulta y actualización constantes de la base de datos, utilizando tipos de consultas y actualizaciones estándar que se han programado y probado cuidadosamente (transacciones enlatadas). Ejemplo: cajero, agente de viajes.

- **Usuarios finales sofisticados:** se encuentran los ingenieros, los científicos, los analistas comerciales y otros muchos que están familiarizados con el DBMS a fin de implementar sus aplicaciones y satisfacer sus complejos requisitos.
- **Usuarios finales independientes:** mantienen bases de datos personales utilizando paquetes de programas confeccionados que proporcionan unas interfaces fáciles de usar y basadas en menús o gráficos.

Un DBMS típico proporciona muchas formas de acceder a una base de datos. Los usuarios finales principiantes tienen que aprender muy poco sobre los servicios del DBMS; simplemente tienen que familiarizarse con las interfaces de usuario de las transacciones estándar diseñadas e implementadas para su uso. Los usuarios casuales solo se aprenden unos cuantos servicios que pueden utilizar repetidamente. Los usuarios sofisticados intentan aprender la mayoría de los servicios del DBMS para satisfacer sus complejos requisitos. Los usuarios independientes normalmente llegan a ser expertos en un paquete de software específico.

ANALISTAS DE SISTEMAS Y PROGRAMADORES DE APLICACIONES (INGENIEROS DE SOFTWARE)

Los **analistas de sistemas** determinan los requisitos de los usuarios finales, especialmente de los usuarios finales principiantes y paramétricos, así como las especificaciones de desarrollo para las transacciones enlatadas que satisfacen esos requisitos. Los **programadores de aplicaciones** implementan esas especificaciones como programas; después verifican, depuran, documentan y mantienen esas transacciones enlatadas. Dichos analistas y programadores (conocidos como **desarrolladores de software o ingenieros de software**) deben familiarizarse con todas las posibilidades proporcionadas por el DBMS al objeto de desempeñar sus tareas.

TRABAJADORES ENTRE BAMBALINAS

Son los usuarios que están asociados con el diseño, el desarrollo y el funcionamiento de un entorno de software y sistema DBMS. No están interesadas en la base de datos propiamente dicha. Se dividen en:

- **Diseñadores e implementadores de sistemas DBMS:** diseñan e implementan los módulos y las interfaces DBMS como paquetes de software. Configura el sistema de base de datos.
- **Desarrolladores e implementadores de herramientas:** diseñan e implementan herramientas. Las herramientas son paquetes de software que facilitan el modelado y el diseño de la base de datos, el diseño del sistema de base de datos y la mejora del rendimiento. Son paquetes opcionales que a menudo se compran por separado.
- **Operadores y personal de mantenimiento:** Son los responsables de la ejecución y el mantenimiento del entorno hardware y software para el sistema de base de datos.

VENTAJAS DE UTILIZAR UNA METODOLOGÍA DBMS

1. Control de la redundancia.
2. Restricción del acceso no autorizado.
3. Almacenamiento persistente para los objetos del programa.
4. Suministro de estructuras de almacenamiento para un procesamiento eficaz de las consultas.
5. Copia de seguridad y recuperación.
6. Suministro de varias interfaces de usuario.
7. Representación de relaciones complejas entre los datos.
8. Implementación de las restricciones de integridad.
9. Inferencia y acciones usando reglas.
10. Implicaciones adicionales de utilizar la metodología de bases de datos.
 - 10.1. Potencial para implementar estándares.
 - 10.2. Tiempo de desarrollo de aplicación reducido.
 - 10.3. Flexibilidad.
 - 10.4. Disponibilidad de la información actualizada.
 - 10.5. Economías de escala.

CONTROL DE LA REDUNDANCIA

La redundancia resultante de almacenar los mismos datos varias veces conduce a serios problemas. En primer lugar, las actualizaciones lógicas sencillas hay que hacerlas varias veces, una por cada archivo donde se almacena el dato a actualizar, lo que lleva a mucho esfuerzo y tiempo. En segundo lugar, se derrocha espacio de almacenamiento al guardar repetidamente los mismos datos y este problema puede llegar a ser muy serio en las bases de datos más grandes. En tercer lugar, los archivos que representan los mismos datos pueden acabar siendo incoherentes, lo que puede ocurrir cuando una determinada actualización se aplica a unos archivos y otros no. Incluso si una actualización se aplica a todos los archivos adecuados, los datos pueden ser incoherentes porque las actualizaciones han sido aplicadas por los distintos grupos de usuarios.

En la metodología de base de datos, las vistas de los diferentes grupos de usuarios se integran durante el diseño de la base de datos. Idealmente, debemos tener un diseño que almacene cada elemento de datos lógico sólo en un lugar de la base de datos. Este hecho garantiza la coherencia y ahorra espacio de almacenamiento. Sin embargo, en la práctica, a veces es necesario recurrir a una **redundancia controlada** para mejorar el rendimiento de las consultas. El DBMS debe tener la capacidad de controlar esta redundancia para evitar las incoherencias entre archivos.

RESTRICCIÓN DEL ACCESO NO AUTORIZADO

Cuando varios usuarios comparten una base de datos grande, es probable que la mayoría de los mismos no tengan autorización para acceder a toda la información de la base de datos. Además, algunos usuarios sólo pueden recuperar datos, mientras que otros pueden recuperarlos y actualizarlos. Así pues, también hay que controlar el tipo de operación de acceso. Normalmente los usuarios o grupos de usuarios tienen números de cuenta protegidos mediante contraseñas, que pueden utilizar para tener acceso a la base de datos. Un DBMS debe proporcionar **seguridad y un subsistema de autorización**, que el DBA utiliza para crear cada cuenta y especificar las restricciones de las mismas. Después el DBMS debe implementar automáticamente esas restricciones. Se pueden aplicar controles parecidos al software DBMS (por ejemplo, restricciones para quienes crean cuentas nuevas).

ALMACENAMIENTO PERSISTENTE PARA LOS OBJETOS DEL PROGRAMA

Las bases de datos se pueden utilizar para proporcionar almacenamiento persistente a los objetos de programa y las estructuras de datos, siendo este el caso de los orientados a objetos. Los valores de las variables de un programa se descartan una vez que termina ese programa, a menos que el programador los almacene explícitamente en archivos permanentes, lo que a menudo implica convertir esas estructuras complejas en un formato adecuado para el almacenamiento del archivo. Cuando surge la necesidad de leer estos datos una vez más, el programador debe convertir el formato del archivo a la estructura variable del programa. Se dice que dicho objeto es **persistente**, porque sobrevive a la terminación de la ejecución del programa y otro programa C++ lo puede recuperar más tarde.

El almacenamiento persistente de objetos de programas y estructuras de datos es una función importante de los sistemas de bases de datos. Los sistemas de bases de datos tradicionales a menudo adolecían de lo que se denominó problema de incompatibilidad de impedancia, puesto que las estructuras de datos proporcionadas por el DBMS eran incompatibles con las estructuras de datos del lenguaje de programación. Los sistemas de base de datos orientados a objeto normalmente ofrecen la compatibilidad de la estructura de datos con uno o más lenguajes de programación orientados a objetos.

SUMINISTRO DE ESTRUCTURAS DE ALMACENAMIENTO PARA UN PROCESAMIENTO EFICAZ DE CONSULTAS

Los sistemas de base de datos deben proporcionar capacidades para ejecutar eficazmente consultas y actualizaciones. Como la base de datos normalmente se almacena en el disco, el DBMS debe proporcionar estructuras de datos especializadas para acelerar la búsqueda en el disco de los registros deseados. Para ello se utilizan archivos auxiliares denominados **índices**. A fin de procesar los registros necesarios de la base de datos para una consulta en particular, estos registros deben copiarse del disco

a la memoria. Por consiguiente, el DBMS a menudo tiene un módulo de búfer que mantiene partes de la base de datos en los búferes de la memoria principal.

El **módulo de procesamiento y optimización de consultas** del DBMS es el responsable de elegir un plan eficaz de ejecución de consultas para cada consulta basándose en las estructuras de almacenamiento existentes. La elección de que índices crear y mantener es parte del diseño y refinamiento de la base de datos física, que es una de las responsabilidades del personal del DBA.

COPIA DE SEGURIDAD Y RECUPERACIÓN

Un DBMS debe ofrecer la posibilidad de recuperarse ante fallos del hardware o del software. El **subsistema de copia de seguridad y recuperación** del DBMS es el responsable de la recuperación.

SUMINISTRO DE VARIAS INTERFACES DE USUARIOS

Como una base de datos la utilizan muchos tipos de usuarios con distintos niveles de conocimiento técnico, un DBMS debe proporcionar distintas interfaces de usuario (lenguajes de consulta para los usuarios casuales, interfaces de lenguaje de programación para los programadores de aplicaciones, formularios y códigos de comandos para los usuarios paramétricos, e interfaces por menús y en el idioma nativo para los usuarios independientes). Tanto las interfaces al estilo de los formularios como las basadas en menús se conocen normalmente **como interfaces gráficas de usuario (GUI, *graphical user interfaces*)**.

REPRESENTACIÓN DE RELACIONES COMPLEJAS ENTRE DATOS

Una base de datos puede incluir numerosas variedades de datos que se interrelacionan entre sí de muchas formas. Un DBMS debe ser capaz de representar las relaciones complejas entre los datos, definir las nuevas relaciones que surgen, y recuperar y actualizar fácil y eficazmente los datos relacionados.

IMPLEMENTACIÓN DE LAS RESTRICCIONES DE INTEGRIDAD

La mayoría de las aplicaciones de bases de datos tienen ciertas restricciones de integridad que deben mantenerse para los datos. Un DBMS debe proporcionar servicios para definir e implementar estas restricciones. Ejemplos: restricción de tipo de dato; un registro de un archivo se debe relacionar con otros registros de otros archivos; unicidad en los valores de los datos. Estas restricciones se derivan del significado o la **semántica** de los datos y del minimundo que representan. Los diseñadores tienen la responsabilidad de identificar las restricciones de integridad durante el diseño de la base de datos. Algunas restricciones pueden especificarse en el DBMS e implementarse automáticamente. Otras restricciones pueden tener que ser comprobadas por los programas de actualización o en el momento de introducir los datos. En las aplicaciones grandes es costumbre denominar estas restricciones como **reglas de negocio**.

INFERENCIA Y ACCIONES USANDO REGLAS

Algunos sistemas ofrecen la posibilidad de definir reglas de deducción para inferir información nueva a partir de los hechos guardados en la base de datos. Estos sistemas se denominan **sistemas de base de datos deductivos**. En los sistemas de bases de datos relacionales actuales es posible asociar *triggers* a las tablas. Un **trigger** es una forma de regla que se activa con las actualizaciones de la tabla, lo que conlleva la ejecución de algunas operaciones adicionales sobre otras tablas, el envío de mensajes, etc. Los procedimientos más implicados en la implementación de reglas se conocen popularmente como **procedimientos almacenados**; se convierten en parte de la definición global de la base de datos y se les invoca correctamente cuando se dan ciertas condiciones.

IMPLICACIONES ADICIONALES DE UTILIZAR LA METODOLOGÍA DE BASES DE DATOS

- **Potencial para implementar estándares:** la metodología de base de datos permite al DBA definir e implementar estándares entre los usuarios de la base de datos en una empresa grande. Esto permite la comunicación y la cooperación entre varios departamentos, proyectos y usuarios dentro de la empresa.

- **Tiempo de desarrollo de aplicaciones reducido:** se necesita muy poco tiempo para desarrollar una aplicación nueva. Una vez que la base de datos esta operativa y en ejecución, por lo general se necesita mucho menos tiempo para crear aplicaciones nuevas utilizando los servicios del DBMS.
- **Flexibilidad:** puede ser necesario cambiar la estructura de una base de datos a medida que cambian los requisitos. Los DBMS modernos permiten ciertos tipos de cambios evolutivos en la estructura de la base de datos, sin que ello afecte a los datos almacenados y a los programas de aplicación existentes.
- **Disponibilidad de información actualizada:** un DBMS hace que la base de datos esté disponible para todos los usuarios. Tan pronto como se aplica la actualización de un usuario a la base de datos, todos los demás usuarios pueden ver esa actualización inmediatamente.
- **Economías de escala:** la metodología DBMS permite la consolidación de los datos y las aplicaciones, lo que reduce el derroche de superposición entre las actividades del personal de procesamiento de datos en diferentes proyectos o departamentos, así como las redundancias entre las aplicaciones. Esto permite que toda la organización invierta en procesadores más potentes, dispositivos de almacenamiento o aparatos de comunicación, en lugar de que cada departamento compre sus propios equipos (menos potentes). De este modo se reducen los costes globales de funcionamiento y administración.

CUANDO NO USAR UN DBMS

Los sobrecostes de utilizar un DBMS se deben a lo siguiente:

- Inversión inicial muy alta en hardware, software y formación.
- La generalidad de que un DBMS ofrece definición y procesamiento de datos.
- Costes derivados de las funciones de seguridad, control de concurrencia, recuperación e integridad.

Es posible que surjan otros problemas si los diseñadores y el DBA no diseñan correctamente la base de datos o si las aplicaciones de sistemas de bases de datos no se implantan correctamente.

Deseable utilizar archivos normales en las siguientes circunstancias:

- Aplicaciones de bases de datos sencillas y bien definidas que no es previsible que cambien.
- Requisitos estrictos y en tiempo real para algunos programas que no podrían satisfacerse debido al sobrecoste de un DBMS.
- Inexistencia del acceso multiusuario a los datos.

CONCEPTOS Y ARQUITECTURA DE LOS SISTEMAS DE BASES DE DATOS

La arquitectura de los paquetes DBMS ha evolucionado desde los antiguos sistemas monolíticos, en lo que todo el paquete de software DBMS era un sistema integrado, hasta los modernos paquetes DBMS con un diseño modular y una arquitectura de sistema cliente/servidor.

En una **arquitectura DBMS cliente/servidor** básica, la funcionalidad del sistema se distribuye entre dos tipos de módulos. Un **módulo cliente** se diseña normalmente para que se pueda ejecutar en la estación de trabajo de un usuario o en un computador personal. Normalmente, las aplicaciones y las interfaces de usuario que acceden a las bases de datos se ejecutan en el módulo cliente. Por tanto, 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.

MODELOS DE DATOS, ESQUEMAS E INSTANCIAS

Una característica fundamental de la metodología de bases de datos es que ofrece algún nivel de abstracción de los datos.

Abstracción de datos: se refiere generalmente a la supresión de detalles de la organización y el almacenamiento de datos y a la relevancia de las características fundamentales para un conocimiento mejorado de los datos.

Una de las características principales de la metodología de bases de datos es soportar la abstracción de datos para que diferentes usuarios puedan percibir esos datos con el nivel de detalle que prefieren.

Un modelo de datos proporciona los medios necesarios para conseguir esa abstracción.

Modelo de datos: es una colección de conceptos que se pueden utilizar para describir la estructura de una base de datos.

Estructura de una base de datos: se refiere a los tipos de datos, relaciones y restricciones que deben mantenerse para los datos.

La mayoría de los modelos de datos incluye un conjunto de operaciones básicas para especificar las recuperaciones y actualizaciones en la base de datos.

Además de las operaciones básicas proporcionadas por el modelo de datos, es cada vez más común incluir conceptos en el modelo de datos para especificar el **aspecto dinámico** o **comportamiento** de una aplicación de base de datos (en modelo de base de datos relacional son los procedimientos almacenados). Esto permite al diseñador de la base de datos especificar un conjunto de operaciones válidas definidas por el usuario que son permitidas en los objetos de la base de datos.

CATEGORÍAS DE MODELOS DE DATOS

- **Modelos de datos de alto nivel (o conceptuales):** ofrecen conceptos muy cercanos a como muchos usuarios perciben los datos. Utilizan conceptos de entidades, atributos y relaciones. Una entidad representa un objeto o concepto del mundo real. Un atributo representa alguna propiedad de interés que describe a una entidad. Una relación entre dos o más entidades representa una asociación entre dos o más entidades.
- **Modelos de datos de bajo nivel (o físicos):** ofrecen conceptos que describen los detalles de cómo se almacenan los datos en el computador. Los conceptos ofrecidos por los modelos de datos de bajo nivel están pensados principalmente para los especialistas en computadoras, no para usuarios finales normales. Describen como se almacenan los datos en el computador en forma de archivos, representado la información como formatos de registro, ordenación de registro y rutas de acceso.
- **Modelos de datos representativos (o de implementación):** ofrecen conceptos que los usuarios finales pueden entender pero que no están demasiado alejados de cómo se organizan los datos dentro del computador. Ocultan algunos detalles relativos al almacenamiento de los datos, pero pueden implementarse directamente en un computador. Son los más utilizados en los DBMS comerciales y tradicionales. Representan los datos mediante estructuras de registro y por tanto se los conoce a veces como modelos de datos basados en registros.

ESQUEMAS, INSTANCIAS Y ESTADO DE LA BASE DE DATOS

En cualquier modelo de datos es importante distinguir entre la descripción de la base de datos y la misma base de datos.

Esquema de la base de datos: es la descripción de una base de datos (el DBMS lo almacena en el catálogo). Se especifica durante la fase de diseño y no se espera que cambie con frecuencia.

La mayoría de los modelos de datos tienen ciertas convenciones para la visualización de los esquemas a modo de diagramas.

Diagrama del esquema: es un esquema visualizado. El diagrama muestra la estructura de cada tipo de registro, pero no las instancias reales de los registros. Un diagrama del esquema solo muestra algunos aspectos de un esquema, como los nombres de los tipos de registros y los elementos de datos, y algunos tipos de restricciones. Otros aspectos no se especifican, no se representan muchos de los tipos de restricciones.

Estructura del esquema: es cada objeto del esquema.

Los datos reales de una base de datos pueden cambiar con mucha frecuencia.

Estado de la base de datos: son los datos de la base de datos en un momento concreto. También se denominan *snapshot* (captura), **ocurrencias** o **instancias** de la base de datos.

En un estado dado de la base de datos, cada estructura de esquema tiene su propio conjunto actual de instancias. Es posible construir muchos estados de la base de datos para que se correspondan con un esquema de bases de datos particular. Cada vez que se inserta o borra un registro, o cambia el valor de un elemento de datos de un registro, cambia el estado de la base de datos por otro.

Esta distinción entre esquema de la base de datos y estado de la base de datos es muy importante. Cuando se define una nueva base de datos, solo se especifica el esquema al DBMS. A estas alturas, el estado de la base de datos es el *estado vacío*, sin datos. El *estado inicial* se da cuando ésta se **rellena** o **carga** por primera vez con los datos iniciales. Desde ese momento, cada vez que sobre la base de datos se aplica una operación de actualización, obtenemos otro estado de la base de datos. En cualquier momento del tiempo la base de datos tiene un estado actual. El DBMS debe garantizar que cada estado de la base de datos sea un **estado válido**, es decir, un estado que satisfaga la estructura y las restricciones especificadas en el esquema. Por tanto, especificar un esquema correcto al DBMS es sumamente importante, por lo que el esquema debe diseñarse con sumo cuidado. El DBMS almacena las descripciones de las construcciones de esquema y las restricciones (también denominadas **metadatos**) en el catálogo del DBMS, para que el software DBMS pueda dirigirse al esquema siempre que lo necesite. En ocasiones, el esquema recibe el nombre de **intención**, y el estado de la base de datos **extensión** del esquema.

Aunque no se supone que el esquema cambie con frecuencia, no es raro que ocasionalmente haya que introducir algún cambio en él al cambiar los requisitos de la aplicación.

Evolución del esquema: cambios en el esquema de la base de datos al cambiar los requisitos de la aplicación.

ARQUITECTURA DE TRES ESQUEMAS E INDEPENDENCIA DE LOS DATOS

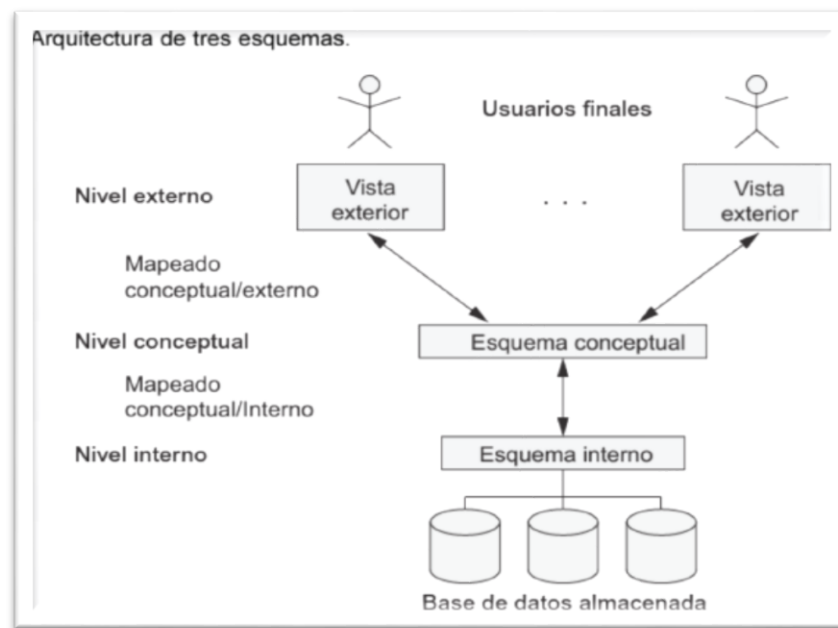
La arquitectura para los sistemas de bases de datos llamada arquitectura de tres esquemas, se propuso para ayudar a conseguir y visualizar tres características de la metodología de bases de datos: independencia programa-datos y programa-operación, soporte de varias vistas de usuario y uso de un catálogo para almacenar la descripción de la base de datos (esquema).

ARQUITECTURA DE TRES ESQUEMAS

El objetivo de la arquitectura de 3 esquemas es separar las aplicaciones de usuarios y las bases de datos físicas. En esta arquitectura se pueden definir esquemas en los siguientes tres niveles:

- **Nivel interno:** tiene un **esquema interno**, que describe la estructura de almacenamiento físico de la base de datos. Utiliza un modelo de datos físico y describe todos los detalles de almacenamiento de datos y las rutas de acceso a la base de datos.
- **Nivel conceptual:** tiene un **esquema conceptual**, que describe la estructura de toda la base de datos para una comunidad de usuarios. El esquema conceptual oculta los detalles de la estructuras de almacenamiento físico y se concentran en describir las entidades, los tipos de datos, las relaciones, las operaciones de los usuarios y las restricciones.

- **Nivel de vista o externo:** tiene una cierta cantidad de **esquemas externos** o **vistas de usuario**. 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.



La arquitectura de tres esquemas es una buena herramienta con la que el usuario puede visualizar los niveles del esquema de un sistema de base de datos. La mayoría de los DBMS no separan completa y explícitamente los tres niveles. Pero soportan esta arquitectura en cierta medida.

La arquitectura de tres niveles ANSI ocupa un lugar importante en el desarrollo de tecnologías de bases de datos porque separa el nivel externo de los usuarios, el nivel conceptual del sistema y el nivel de almacenamiento interno para diseñar una base de datos.

Los 3 esquemas son solo *descripciones* de datos, los datos que existen en realidad están en el nivel físico. Cada grupo de usuarios solo se refiere a su propio esquema externo. Por tanto, el DBMS debe transformar una solicitud especificada en el esquema externo en una solicitud contra el esquema conceptual y esta a su vez en una solicitud contra el esquema interno para el procesamiento sobre la base de datos almacenada. Estos procesos de conversión de solicitudes se denominan **mapeados**. Estos mapeados pueden consumir bastante tiempo, por lo que algunos DBMS no soportan las vistas externas.

INDEPENDENCIA DE LOS DATOS (LÓGICA Y FÍSICA)

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

- **Independencia lógica de datos:** es la capacidad de cambiar el esquema conceptual sin tener que cambiar los esquemas externos o los programas de aplicación. Eso puede ser para expandir la base de datos, cambiar restricciones, o para reducir la base de datos. En el último caso, no deben verse afectados los esquemas externos que solo se refieren a los datos restantes. Solo es necesario cambiar la definición de la vista y los mapeados en un DBMS que soporta independencia lógica de datos.
- **Independencia física de datos:** es la capacidad de cambiar el esquema interno sin que haya que cambiar el esquema conceptual. Por lo tanto, tampoco es necesario cambiar esquemas externos. Puede que haya que realizar cambios en el esquema interno porque algunos archivos físicos fueran reorganizados 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 base de datos y de los entornos de archivo en los que al usuario se le ocultan la ubicación exacta de los datos en el disco, los detalles de hardware de la codificación del almacenamiento, la colocación, la compresión, la división, la fusión de registros, etc. Las aplicaciones siguen obviando estos detalles. Por el contrario, 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).

Siempre que tengamos un DBMS de varios niveles, su catálogo debe ampliarse para incluir información de cómo mapear las consultas y los datos entre los diferentes niveles. El DBMS utiliza software adicional para acometer estos mapeados refiriéndose a la información de mapeado que hay en el catálogo. La independencia de datos ocurre porque cuando el esquema cambia a algún nivel, el esquema en el siguiente nivel más alto permanece inalterado; solo cambia el mapeado entre los dos niveles. Por tanto, las aplicaciones que hacen referencia al esquema del nivel más alto no tienen que cambiar.

La arquitectura de tres esquemas puede facilitar el conseguir la independencia de datos verdadera, tanto física como lógica. Sin embargo, los dos niveles de mapeados crean un sobrecoste durante la compilación o la ejecución de una consulta o un programa, induciendo a deficiencias en el DBMS. Debido a esto, pocos DBMS han implementado la arquitectura de tres esquemas completa.

LENGUAJES E INTERFACES DE BASES DE DATOS

EL DBMS debe proporcionar los lenguajes e interfaces apropiados para cada categoría de usuarios.

LENGUAJES DBMS

Una vez realizado el diseño de una base de datos y elegido un DBMS para implementarla, el primer paso es especificar los esquemas conceptual e interno y el mapeado entre los dos. En los DBMS donde no hay una separación estricta de niveles, el DBA y los diseñadores de la BD utilizan un lenguaje denominado **lenguaje de definición de datos (DDL, *data definition language*)**, para definir los dos esquemas. El DBMS deberá tener 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. Las sentencias DDL son aquellas utilizadas para la creación de una base de datos y todos sus componentes: tablas, índices, relaciones, disparadores (*triggers*), procedimientos almacenados, etc.

En DBMS donde hay una clara separación entre los niveles conceptual e interno, se utiliza DDL sólo para especificar el esquema conceptual, y para especificar el esquema interno se utiliza el **lenguaje de definición de almacenamiento (SDL, *storage definition language*)**. El mapeado entre los dos esquemas se puede especificar en cualquiera de los dos lenguajes. En la mayoría de los DBMS relacionales actuales, no hay un lenguaje específico que asuma el papel de SDL. En cambio, el esquema interno se especifica mediante una combinación de parámetros y especificaciones relacionadas con el almacenamiento: el personal del DBA normalmente controla la indexación y la asignación de datos al almacenamiento. Para conseguir una arquitectura de 3 niveles real se necesita un tercer lenguaje, el **lenguaje de definición de vistas (VDL, *view definition language*)**, que especifica las vistas del usuario y sus mapeados al esquema conceptual. Sin embargo, en la mayoría de los DBMS se utiliza el DDL para definir tanto el esquema conceptual como el externo.

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 un **lenguaje denominado lenguaje de manipulación de datos (DML, *data manipulation language*)**, para todas estas tareas.

En los DBMS actuales, los tipos de lenguajes anteriormente citados normalmente no están considerados como lenguajes distintos; más bien, se utiliza un lenguaje integrado comprensivo que incluye construcciones para la definición del esquema conceptual, la definición de vistas y la manipulación de datos. La definición del almacenamiento normalmente se guarda aparte, ya que se utiliza para definir las estructuras de almacenamiento físico a fin de refinar el rendimiento del sistema de bases de datos, que

normalmente lo lleva a cabo el personal del DBA. SQL es un claro ejemplo de una combinación de DDL, VDL y DML, así como sentencias para la especificación de restricciones, la evolución del esquema y otras características.

Hay dos tipos principales de DML:

- **DML de alto nivel o no procedimental** para especificar de forma concisa las operaciones complejas con las bases de datos. Muchos DBMS admiten sentencias DML de alto nivel mediante la introducción interactiva desde el monitor o terminal, o incrustadas en un lenguaje de programación de propósito general. Especifican y recuperan muchos registros con una sola sentencia DML (*DML set-at-a-time*). Ejemplo: SQL.
- **DML de bajo nivel o procedimental** debe incrustarse en un lenguaje de programación de propósito general. Normalmente, este tipo de DML recupera registros individuales u objetos de la base de datos, y los procesa por separado. Se conocen como “registros de una sola vez” (*DML record-at-a-time*).

Una consulta en DML de alto nivel a menudo especifica los datos que hay que recuperar, en lugar de como recuperarlos; en consecuencia, dichos lenguajes se conocen también como **declarativos**.

Siempre que hay comandos DML, de alto o bajo nivel, incrustados en un lenguaje de programación de propósito general, ese lenguaje se denomina **lenguaje host**, y el DML **sublenguaje de datos**. Por el contrario, un DML de alto nivel utilizado de forma interactiva independiente se conoce como **lenguaje de consulta**.

Los usuarios finales casuales normalmente utilizan un lenguaje de consulta de alto nivel para especificar sus consultas, mientras que los programadores utilizan DML en su forma incrustada. Los usuarios principiantes y paramétricos normalmente utilizan interfaces amigables para el usuario para interactuar con la base de datos.

INTERFACES DE LOS DBMS

El DBMS puede incluir las siguientes interfaces amigables para el usuario:

- **Interfaces basadas en menús para los clientes web o la exploración:** estas interfaces presentan al usuario listas de opciones (denominadas menús) que le guían para la formulación de una consulta. Los menús eliminan la necesidad de memorizar los comandos específicos y la sintaxis de un lenguaje de consulta.
- **Interfaces basadas en formularios:** muestran un formulario a cada usuario. Los usuarios pueden rellenar las entradas del formulario para insertar datos nuevos, o rellenar únicamente ciertas entradas, en cuyo caso el DBMS recuperará los datos coincidentes para el resto de entradas.
- **Interfaces gráficas de usuario:** una GUI normalmente muestra un esquema al usuario de forma esquemática. El usuario puede especificar entonces una consulta manipulando el diagrama. Utilizan tanto menús como formularios.
- **Interfaces de lenguaje natural:** estas interfaces aceptan consultas escritas en inglés u otro idioma e intentan entenderlas.
- **Entrada y salida de lenguaje hablado:** cada vez es más común el uso limitado del habla como consulta de entrada y como respuesta a una pregunta o como resultado de una consulta.
- **Interfaces para los usuarios paramétricos:** incluyen un pequeño conjunto de comandos abreviados, con el objetivo de minimizar el número de pulsaciones necesarias por cada consulta.
- **Interfaces para el DBA:** contienen comandos privilegiados que sólo puede utilizar el personal del DBA. Entre ellos hay comandos para crear cuentas, configurar los parámetros del sistema, conceder la autorización de una cuenta, cambiar un esquema y reorganizar las estructuras de almacenamiento de una base de datos.

ENTORNO DE UN SISTEMA DE BASES DE DATOS

MÓDULOS COMPONENTES DE UN DBMS

Las bases de datos y el catálogo del DBMS normalmente se almacenan en disco. El acceso al disco está principalmente controlado por el **sistema operativo**, que planifica la entrada/salida del disco. Un módulo **administrador de datos almacenados** de alto nivel del DBMS controla el acceso a la información del DBMS almacenada en el disco, sea parte de la base de datos o del catálogo.

El personal del DBA utiliza sentencias DDL y otros comandos privilegiados para definir y refinar la base de datos. El **compilador DDL** procesa las definiciones de esquema, especificadas en el DDL, y almacena las descripciones de los esquemas (metadatos) en el **catálogo** (nombre y tamaños de los archivos, información de mapeado entre esquemas, las restricciones, entre otra información) del DBMS. Los módulos software del DBMS buscan después en el catálogo la información que necesitan.

Los usuarios casuales de la base de datos interactúan utilizando alguna forma de interfaz, como la **interfaz de consulta interactiva**. Un **compilador de consultas** analiza estas consultas sintácticamente para garantizar la corrección de las operaciones de los modelos, los nombres de los elementos de datos, etc., y luego lo compila todo en un formato interno. El **optimizador de consultas** se ocupa de la reconfiguración y la posible reordenación de operaciones, eliminación de redundancias y uso de algoritmos e índices correctos durante la ejecución. Consulta el catálogo del sistema para información estadística y física acerca de los datos almacenados, y genera un código ejecutable que lleva a cabo las operaciones necesarias para la consulta y realiza las llamadas al **procesador runtime**.

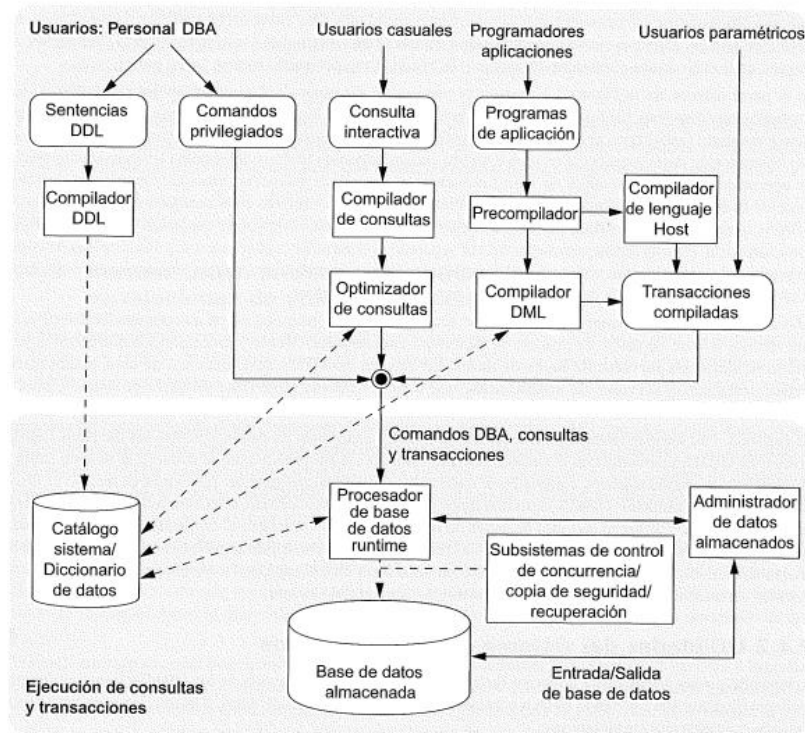
Los programadores de aplicaciones escriben **programas en lenguajes host** (como Java, C, etc.) que son enviados a un **precompilador**. Éste extrae los comandos DML de un programa de aplicación escrito en un lenguaje de programación *host*.

Estos comandos se envían al **compilador DML** para su compilación en código objeto y así poder acceder a la base de datos. El resto del programa se envía al **compilador de lenguaje host**. Los códigos objeto para los comandos DML y el resto del programa se enlazan, formando una **transacción enlatada** cuyo código ejecutable incluye llamadas al **procesador de base de datos runtime**.

El procesador de bases de datos *runtime* para ejecutar (1) los comandos privilegiados, (2) los proyectos de consulta ejecutables y (3) las transacciones enlatadas con parámetros *runtime*, trabaja con el **diccionario del sistema**. Funciona con el administrador de datos almacenados, que a su vez utiliza servicios básicos del sistema operativo para ejecutar operaciones de entrada/salida de bajo nivel entre el disco y la memoria principal. Se encarga de otros aspectos de la transferencia de datos como la administración de los búferes en la memoria principal. Algunos DBMS tienen su propio módulo de administración de búfer, mientras que otros dependen del SO para dicha administración.

El DBMS interactúa con el sistema operativo cuando se necesita acceso al disco (a la base de datos o al catálogo). Si varios usuarios comparten el computador, el SO planificará las peticiones de acceso al disco DBMS y el procesamiento DBMS junto con otros procesos.

Procesador de BD runtime: recibe comandos privilegiados del DBA, consultas y transacciones enlazadas con programas y se encarga de ejecutarlas accediendo al diccionario de datos y comunicándose con el S.O. para efectuar las operaciones de entrada/salida de información. También se encarga de la administración del buffer en la memoria de procesamiento, esta función puede o no compartirla con el S.O.



UTILIDADES DEL SISTEMA DE BASES DE DATOS

La mayoría de los DBMS tienen utilidades de bases de datos que ayudan al DBA a administrar el sistema de base de datos. Las funciones de las utilidades más comunes son:

- **Carga:** la carga de los archivos de datos existentes (como archivos de texto o archivos secuenciales) en la base de datos se realiza con una utilidad de carga.
- **Copia de seguridad:** una utilidad de copia de seguridad crea una copia de respaldo de la base de datos. Esa copia de seguridad se la puede utilizar para restaurar la base de datos en caso de un fallo desastroso.
- **Reorganización del almacenamiento de la base de datos:** se utiliza para reorganizar un conjunto de archivos de bases de datos en una organización de archivos diferente a fin de mejorar el rendimiento.
- **Monitorización del rendimiento:** monitoriza el uso de la base de datos y ofrece estadísticas al DBA. Este último utiliza las estadísticas para tomar decisiones, como si debe o no reorganizar los archivos, o si tiene que añadir o eliminar índices para mejorar el rendimiento.

Hay otras utilidades para ordenar archivos, manipular la comprensión de datos, monitorizar el acceso de los usuarios, interactuar con la red y llevar a cabo otras funciones.

ARQUITECTURAS CLIENTE/SERVIDOR CENTRALIZADAS PARA LOS DBMS

1. Arquitectura centralizada de los DBMS.
2. Arquitecturas cliente/servidor básicas.
3. Arquitecturas cliente/servidor de dos capas para los DBMS.
4. Arquitectura de tres capas y n capas para las aplicaciones web.

ARQUITECTURA CENTRALIZADA DE LOS DBMS

Las primeras arquitecturas utilizaban *mainframes* para proporcionar el procesamiento principal a todas las funciones del sistema, incluyendo las aplicaciones de usuario y los programas de interfaz de usuario, así como toda la funcionalidad del DBMS. Todo el procesamiento se realizaba remotamente en el sistema computador, y sólo se enviaba la información de visualización y los controles desde el computador a los terminales de visualización, que estaban conectados con el computador central a través de diferentes redes de comunicaciones. El DBMS era centralizado, en el que toda la funcionalidad

DBMS, ejecución de aplicaciones e interacción con el usuario se llevaba a cabo en una máquina. Esto se debía a la poca potencia de procesamiento de los computadores de los usuarios.

ARQUITECTURAS CLIENTE/SERVIDOR BÁSICAS

Se desarrolló para ocuparse de los entornos de computación en los que una gran cantidad de **PCs, estaciones de trabajo, servidores de archivos, impresoras, servidores de bases de datos, servidores web** y otros equipos están conectados a través de una red. La idea es definir **servidores especializados con funcionalidades específicas**. De este modo, muchas máquinas cliente pueden acceder a los recursos proporcionados por servidores especializados.

Las máquinas **cliente** proporcionan al usuario las interfaces apropiadas para utilizar estos servidores, así como potencia de procesamiento local para ejecutar aplicaciones locales.

En esta estructura, un cliente es normalmente la máquina de un usuario que proporciona capacidad de interfaz de usuario (las interfaces apropiadas para poder utilizar los servidores) y potencia de procesamiento local para ejecutar aplicaciones locales. Cuando un cliente requiere acceso a funcionalidad adicional que no existe en esa máquina, conecta con un servidor que ofrece la funcionalidad necesaria.

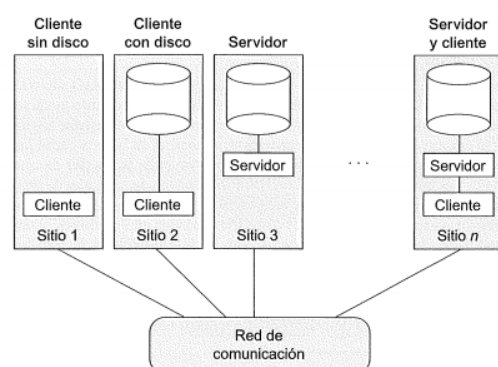
En el caso general, algunas máquinas solo instalan el software cliente, mientras otras solo instalan el software servidor, y otras pueden incluir los dos. No obstante, lo más normal es que el software cliente y el software servidor se ejecuten en máquinas separadas. Los dos tipos principales de arquitecturas DBMS básicas se crearon sobre esta estructura cliente/servidor fundamental: dos capas y tres capas.

ARQUITECTURAS CLIENTE/SERVIDOR DE DOS CAPAS PARA LOS DBMS

La arquitectura cliente/servidor se está incorporando progresivamente a los paquetes DBMS. En los sistemas de administración de bases de datos relacionales (RDBMS), muchos de los cuales empezaron como sistemas centralizados, los primeros componentes del sistema que se movieron al lado del cliente fueron la interfaz de usuario y las aplicaciones. Como SQL ofrecía un lenguaje estándar para los RDBMS, se creó un punto de división lógica entre cliente y servidor. Por tanto, la funcionalidad de consulta y transacción relacionada con el procesamiento SQL permanece en el lado del servidor, que se denomina **servidor de consultas** o **servidor de transacciones**. En un RDBMS también se lo llama **servidor SQL**.

En una arquitectura cliente/servidor, los programas de interfaz de usuario y los programas de aplicación se pueden ejecutar en el lado del cliente. Cuando se necesita acceso DBMS, el programa establece una conexión con el DBMS (que se encuentra en el lado del servidor); una vez establecida, el programa cliente puede comunicarse con el DBMS. El estándar **Conectividad abierta de bases de datos (ODBC, Open Database Connectivity)** proporciona una **interfaz de programación de aplicaciones (API, application programming interface)**, que permite a los programas del lado del cliente llamar al DBMS, siempre y cuando las máquinas cliente y servidor tengan instalado el software necesario. Los resultados de una consulta se envían de regreso al programa cliente que procesará o visualizará los resultados según las necesidades.

Las arquitecturas descritas aquí se llaman arquitecturas de dos capas porque los componentes software están distribuido en dos sistemas: cliente y servidor. Las ventajas de esta arquitectura son su simplicidad y su perfecta compatibilidad con los sistemas existentes.

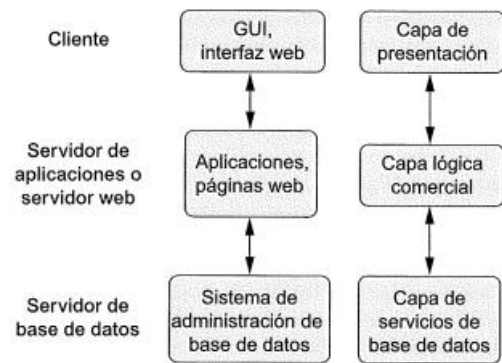


ARQUITECTURA DE TRES CAPAS Y N CAPAS PARA LAS APLICACIONES WEB

Muchas aplicaciones web utilizan una arquitectura denominada de tres capas, que añade una capa intermedia entre el cliente y el servidor de la base de datos. Esta capa intermedia se denomina a veces **servidor de aplicaciones** y, en ocasiones, **servidor web**, en función de la aplicación. Este servidor juega

un papel intermedio almacenando las reglas comerciales (procedimientos o restricciones) que se utilizan para acceder a los datos del servidor de base de datos. También puede mejorar la seguridad de la base de datos al comprobar las credenciales del cliente antes de enviar una solicitud al servidor de base de datos.

Los clientes contienen interfaces GUI y algunas reglas comerciales adicionales específicas de la aplicación. El servidor intermedio acepta solicitudes del cliente, las procesa y envía comandos de bases de datos al servidor de bases de datos, y después actúa como un conducto para pasar datos procesados (parcialmente) desde el servidor de base de datos a los clientes, donde son procesados de forma más avanzada para su presentación en formato GUI a los usuarios. De este modo, la interfaz de usuario, las reglas de aplicación y el acceso de datos actúan como las tres capas.



La capa de presentación muestra información al usuario y permite la entrada de datos. La capa lógica comercial manipula las reglas intermedias y las restricciones antes de que los datos sean pasados hacia arriba hasta el usuario, o hacia abajo, hasta el DBMS. La capa inferior incluye todos los servicios de administración de datos.

Es costumbre dividir las capas entre el usuario y los datos almacenados en componentes aún más sutiles, para de este modo llegar a arquitecturas de n capas, donde n puede ser cuatro o cinco. Normalmente, la capa lógica comercial está dividida en varias capas. Además de distribuir la programación y los datos por la red, las aplicaciones de n capas ofrecen la ventaja de que cualquiera de las capas se puede ejecutar en un procesador adecuado o plataforma de sistema operativo, además de poderse manipular independientemente.

CLASIFICACIÓN DE LOS DBMS

Se utilizan varios criterios para clasificar a los DBMS.

Según el **modelo de datos**:

- **Modelo de datos relacional:** es el modelo de datos principal que se utiliza en muchos DBMS comerciales actuales. Han ido evolucionando constantemente y, en particular, han ido incorporando muchos de los conceptos que se desarrollaron en las bases de datos de objetos. Esto ha producido una nueva clase de DBMS denominado DBMS objeto-relacional. Representa la base de datos como una colección de tablas, donde cada tabla se puede almacenar como un archivo separado.
- **Modelo de datos de objetos:** aplica conceptos del paradigma orientado a objetos, como clases, herencia, comportamiento (métodos). Su uso no se ha extendido.
- **Modelos de datos jerárquicos:** representa los datos como estructuras en forma de árboles jerárquicos. Cada jerarquía representa una cantidad de registros relacionados.
- **Modelo de datos de red:** representa los datos como tipos de registros, y también representa un tipo limitado de relación 1:N, denominado **tipo conjunto**, que relaciona una instancia de un registro con muchas instancias de registro mediante algún mecanismo de punteros en esos modelos.

Según el **número de usuarios** soportado por el sistema:

- **Sistemas monousuario:** solo soportan un solo usuario al mismo tiempo y se utilizan principalmente en los PC.
- **Sistemas multiusuario:** lo incluyen la mayoría de los DBMS, soportan varios usuarios simultáneamente.

Según el **número de sitios** sobre los que se ha distribuido la base de datos:

- **Centralizado:** si los datos están almacenados en un solo computador.
- **Distribuido:** denominado DDBMS, puede tener la base de datos y el software DBMS distribuidos por muchos sitios, conectados por una red de computadores.

ALMACENAMIENTO

Las bases de datos se almacenan físicamente como ficheros de registros, normalmente en discos magnéticos.

MEDIOS DE ALMACENAMIENTO

La colección de datos que constituye una base de datos computarizada debe almacenarse físicamente en un **medio de almacenamiento** de un computador. El software DBMS puede recuperar, actualizar y procesar después estos datos cuando lo necesite. Los medios de almacenamiento se categorizan jerárquicamente:

- **Almacenamiento principal o primario:** son los medios de almacenamiento en los que la CPU puede operar, como la memoria principal de la computadora y las memorias caché, más pequeñas pero más rápidas. Normalmente, proporciona un acceso rápido a los datos, pero tiene una capacidad de almacenamiento limitada. En el extremo más caro se encuentra la **memoria caché**, que es una RAM (memoria de acceso aleatorio) estática y es normalmente utilizada por el CPU para acelerar la ejecución de los programas. Le sigue la **memoria DRAM** (RAM dinámica), que proporciona el área de trabajo principal para que la CPU almacene datos y programas. Se denomina memoria principal siendo su ventaja el bajo costo, pero su desventaja la volatilidad y baja velocidad comparada a la caché.
- **Almacenamiento secundario y terciario:** esta categoría incluye los discos magnéticos, ópticos, y las cintas. Las unidades de disco duro se consideran como **secundarias (online)**, mientras que las unidades ópticas, medios removibles o extraíbles y cintas se consideran como **terciarias (offline)**. Estos dispositivos normalmente tienen gran capacidad, cuestan poco y proporcionan un acceso más lento a los datos que los dispositivos de almacenamiento principales. La CPU no puede procesar directamente los datos almacenados en un almacenamiento secundario o terciario, primero deben copiarse en el almacenamiento principal.

ALMACENAMIENTO DE BASES DE DATOS

Las bases de datos almacenan grandes cantidades de datos que pueden persistir durante largos periodos de tiempo. Durante ese periodo, se acceden y procesan los datos repetidamente. La mayoría de las bases de datos se almacenan de forma permanente en almacenamiento secundario de discos magnéticos por las siguientes razones:

- Generalmente, son demasiado grandes para entrar en la memoria principal.
- Las circunstancias que provocan la pérdida de datos almacenados son menos frecuentes con el almacenamiento secundario que con el principal. Por tanto, nos referimos al disco (y a otros dispositivos de almacenamiento secundario) como almacenamiento no volátil, mientras que la memoria principal se denomina, a menudo, almacenamiento volátil.
- El costo de almacenamiento por unidad de datos es un orden de magnitud inferior para el almacenamiento secundario en disco que para el almacenamiento principal.

Las cintas magnéticas se utilizan con frecuencia como medio de almacenamiento para hacer copias de seguridad de las bases de datos, porque el almacenamiento en cinta es mucho menos costoso que el de disco. Pero el acceso a los datos almacenados en una cinta es mucho más lento. Los datos almacenados en las cintas están **offline**; es decir, para que los datos estén disponibles, antes se necesita alguna intervención por parte de un operador (o un dispositivo de carga automática) para cargar una cinta. Por

el contrario, los discos son dispositivos **online** a los que se puede acceder directamente en cualquier momento.

Los diseñadores de bases de datos y el DBA deben conocer las ventajas y los inconvenientes de cada técnica de almacenamiento cuando se dispongan a diseñar, implementar y operar una base de datos en un DBMS concreto. El proceso de **diseño físico de la base de datos** implica elegir las técnicas de organización de datos particulares que mejor se adapten a los requisitos de la aplicación. Este proceso es importante ya que influye en performance, e implica colocar a los registros en los medios de almacenamiento.

Las aplicaciones típicas de bases de datos solo necesitan procesar una pequeña porción de la base de datos en cada momento. Siempre que se necesita una determinada porción de datos, debe buscarse en el disco, copiarse en la memoria principal para su procesamiento y, después, reescribirse en el disco si los datos han sido cambiados. Los datos en el disco se organizan como ficheros de registros. Un registro es una colección de valores de datos que pueden interpretarse como hechos relacionados con las entidades, sus atributos y sus relaciones. Los registros deben almacenarse en disco de un modo que haga posible su localización de la forma más eficaz cuando sean necesarios.

Fichero: secuencia de registros.

Registro: es una colección de valores.

ORGANIZACIÓN DE FICHEROS EN DISCO

Hay varias **organizaciones principales de ficheros**, que determinan la *colocación física* de los registros del fichero en el disco y, por tanto, cómo se puede acceder a ellos.

- **Fichero heap** (o **fichero desordenado**): coloca los registros en el disco sin un orden particular, añadiendo los registros nuevos al final del fichero.
- **Fichero ordenado** (o **fichero secuencial**): mantiene ordenado los registros por el valor de un campo particular, denominado *clave de ordenación*.
- **Fichero disperso**: utiliza una función de dispersión (*hash*) aplicada a un campo concreto para determinar la ubicación de un registro en el disco.

DISPOSITIVOS DE ALMACENAMIENTO SECUNDARIO

DESCRIPCIÓN DEL HARDWARE DE LOS DISPOSITIVOS DE DISCO

Los discos magnéticos se utilizan para almacenar grandes cantidades de datos. La unidad de datos más básica es el **bit** de información. Al magnetizar un área del disco, podemos hacer que represente un valor de bit de 0 (cero) o 1 (uno). Para codificar la información, los bits se agrupan en **bytes** (o caracteres). El tamaño de los bytes es normalmente de 4 a 8 bits, en función del computador y del dispositivo. La capacidad de un disco es el número de bytes que puede almacenar, que normalmente es muy grande.

Todos los discos están fabricados con un material magnético al que se le da forma de disco circular, que va protegido por una carcasa plástica o acrílica. Un disco es de una cara si sólo almacena información en una de sus superficies, y de doble cara si utiliza las dos superficies. Para aumentar la capacidad de almacenamiento, los discos se ensamblan como **paquetes de discos**, que pueden incluir muchos discos y, por tanto, muchas superficies. La información se almacena en la superficie del disco en círculos concéntricos de poca anchura, y cada uno de un diámetro distinto. Cada uno de esos círculos es una pista. En los paquetes de discos, las pistas de las distintas superficies que tienen el mismo diámetro reciben el nombre de **cilindro**. Este concepto es importante porque los datos almacenados en un mismo cilindro se pueden recuperar mucho más rápidamente que si estuvieran distribuidos por diferentes cilindros.

Como una pista normalmente contiene una gran cantidad de información, está dividida en bloques más pequeños o sectores. La división de una pista en sectores está codificada en la superficie del disco y no se puede cambiar.

La división de una pista en **bloques de disco** (o **páginas**) es establecida por el sistema operativo durante el formateo (o inicialización) del disco. El tamaño del bloque se fija durante la inicialización y no puede cambiarse dinámicamente. Los bloques están separados por **huecos entre bloques** de un tamaño fijo, que incluyen información de control codificada de forma especial y que se escribe durante la inicialización del disco. Esta información se utiliza para determinar el bloque de la pista que va a continuación de cada hueco.

Un disco es un dispositivo direccionable de *acceso aleatorio*. La transferencia de datos entre la memoria principal y el disco tiene lugar en unidades de bloque de disco. La **dirección de hardware** de un bloque (combinación de número de cilindro, número de pista y número de bloque) se suministra al hardware de E/S del disco. También se proporciona la dirección de un **búfer** (un área reservada contigua en el almacenamiento principal que almacena un bloque). Para un comando de lectura, el bloque del disco se copia en el búfer, mientras que para la escritura, el contenido del búfer se copia en el bloque de disco. En ocasiones, varios bloques contiguos, denominados **clúster**, se pueden transferir como unidad. En este caso, el tamaño de búfer se ajusta para que coincida con el número de bytes del clúster.

El mecanismo hardware actual que lee o escribe un bloque es la **cabeza de lectura/escritura** del disco, que es parte de un sistema denominado unidad de disco. En la unidad de disco se monta un disco o un paquete de discos, que giran gracias a un motor incluido en la unidad. Una cabeza de lectura/escritura incluye un componente electrónico conectado a un **brazo mecánico**. Los paquetes de discos con varias superficies están controlados por varias cabezas de lectura/escritura, una por cada superficie. Todos los brazos están conectados a un **activador**, conectado a otro motor eléctrico, que mueve las cabezas de lectura/escritura al unísono y las coloca con precisión sobre el cilindro o las pistas especificadas en una dirección de bloque.

El **controlador de disco**, normalmente incrustado en la unidad de disco, controla ésta y su interacción con el sistema. El controlador acepta comandos de E/S de alto nivel y toma la acción apropiada para posicionar el brazo y provoca la acción de lectura/escritura. Para transferir un bloque de disco, dada su dirección, el controlador de disco primero debe colocar mecánicamente la cabeza de lectura/escritura en la pista correcta. El tiempo requerido para ello es el **tiempo de búsqueda**. Siguiendo a esto, hay otro retardo, denominado **retardo rotacional** o **latencia**, que se produce mientras el principio del bloque deseado gira hasta su posición bajo la cabeza de lectura/escritura. Este retardo depende de la rpm del disco. Por último, aún se necesita algo más de tiempo para transferir los datos; es lo que se conoce como **tiempo de transferencia del bloque**. Por tanto, el tiempo total necesario para localizar y transferir un bloque arbitrario, dada su dirección, es la suma del tiempo de búsqueda, el retardo rotacional y el tiempo de transferencia de bloque. Los dos primeros, son normalmente mucho más grandes que el último. Para que la transferencia de varios bloques sea más eficaz, es común transferir varios bloques consecutivos de la misma pista o cilindro.

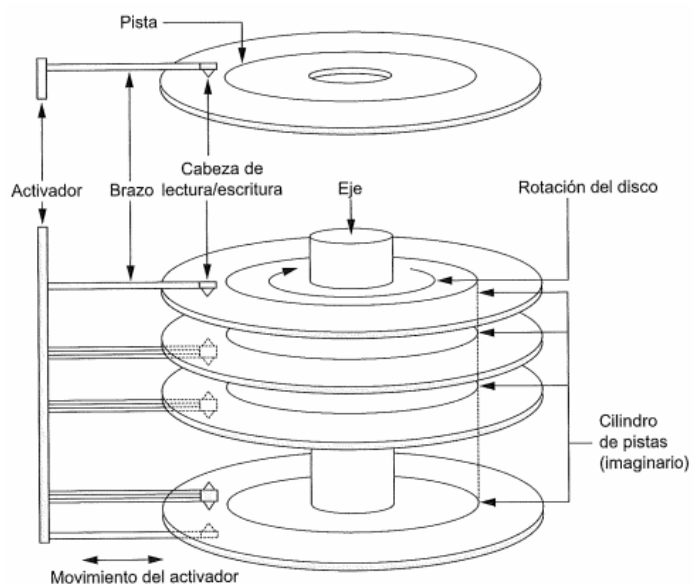
CONCEPTOS IMPORTANTES

Cilindro: pistas del mismo diámetro -influye en las búsquedas-

Bloque o página: tamaño establecido por el SO al inicializar -no dinámico- Es la unidad de transferencia entre disco y memoria.

Clúster: bloques contiguos en el disco.

Dirección del bloque: para hacer E/S N° cilindro + N° pista + N° bloque.



DISPOSITIVOS DE ALMACENAMIENTO EN CINTA MAGNÉTICA

Los datos en una cinta se almacenan en bobinas de cinta magnética de alta capacidad, parecidas a las cintas de audio o de video. Para leer datos de, o escribir datos en, una bobina de cinta se necesita una unidad de cinta.

Se utiliza una cabeza de lectura/escritura para leer o escribir datos en la cinta. Los registros de datos también se almacenan en la cinta en bloques (aunque los bloques pueden ser sustancialmente más grandes que los de los discos, y los huecos entre los bloques también son bastante grandes). Es costumbre agrupar muchos registros juntos en un bloque para una mejor utilización del espacio.

La principal característica de una cinta es su requisito de acceder a los bloques de datos en **orden secuencial**, es decir, que son dispositivos de acceso secuencial. Para obtener un bloque de en medio de una bobina de cinta, se monta la cinta y se escanea hasta que el bloque requerido se coloca bajo la cabeza de lectura/escritura. Por esta razón, el acceso a la cinta puede ser lento y las cintas no se utilizan para almacenar datos online, excepto para algunas aplicaciones especializadas. Sin embargo, las cintas sirven para una función muy importante, la copia de seguridad de la base de datos. Esto es para mantener copias de los ficheros en disco por si se pierden datos como consecuencia de un fallo del disco, que puede ocurrir si la cabeza de lectura/escritura toca la superficie del disco debido a un mal funcionamiento mecánico. Las cintas también se pueden utilizar para almacenar los ficheros de bases de datos excesivamente grandes. Los ficheros que rara vez se utilizan o están desactualizados, pero requieren que se guarde un registro histórico, pueden archivar en cinta. También se utilizan para almacenar imágenes y librerías del sistema.

ALMACENAMIENTO DE BLOQUES EN EL BÚFER

Cuando es necesario transferir varios bloques desde el disco a la memoria principal y se conocen todas las direcciones de bloque, se pueden reservar varios búferes en la memoria principal para acelerar la transferencia. Mientras se está leyendo o escribiendo en un búfer, la CPU puede procesar datos en otro búfer porque hay un procesador (controlador) de E/S de disco independiente que, una vez iniciado, puede proseguir con la transferencia de un bloque de datos entre la memoria y el disco independientemente de, y en paralelo, al procesamiento de la CPU.

Cuando una CPU controla varios procesos, la ejecución en paralelo no es posible. Sin embargo, los procesos todavía se pueden ejecutar al mismo tiempo de forma interpolada. El almacenamiento en búfer es más útil cuando los procesos se pueden ejecutar al mismo tiempo y en paralelo, porque un procesador de E/S de disco independiente esté disponible, o porque la CPU tenga varios procesadores.

Cuando el tiempo necesario para procesar un bloque de disco en la memoria es inferior al tiempo necesario para leer el siguiente bloque y llenar un búfer, la CPU puede iniciar el procesamiento de un bloque una vez completada su transferencia a la memoria principal, y, al mismo tiempo, el procesador de E/S de disco puede estar leyendo y transfiriendo el siguiente bloque a un búfer diferente. Esta técnica se denomina **doble búfer** y también puede utilizarse para leer un flujo continuo de bloques del disco a la memoria. El doble búfer permite la lectura o escritura continua de datos en bloques de disco consecutivos, lo que elimina el tiempo de búsqueda y el retardo rotacional de la transferencia de todos los bloques, excepto del primero. Además, los datos quedan listos para su procesamiento, lo que reduce el tiempo de espera en los programas.

UBICACIÓN DE LOS REGISTROS DE FICHERO EN EL DISCO

REGISTROS Y TIPOS DE REGISTROS

Los datos normalmente se almacenan en forma de **registros**. Un registro consta de una colección de **valores** o **elementos** de datos relacionados, donde cada valor está formado por uno o más bytes y corresponde a un campo concreto del registro. Los registros normalmente describen entidades y sus atributos.

Registro: es donde se almacenan los datos. Son una colección de valores de datos, donde cada valor corresponde a un campo del registro.

Tipo de registro: es una colección de nombres de campos y sus correspondientes tipos de datos.

Tipo de dato: cuando está asociado a un campo, especifica los tipos de valores que ese campo puede tomar.

El tipo de dato de un campo normalmente es uno de los tipos de datos estándar que se utilizan en programación: numéricos (entero, entero largo o coma flotante), cadenas de caracteres (de longitud fija o variable), booleanos (0 y 1; true y false), fechas y horas.

Puede darse la necesidad de almacenar elementos de datos consistentes en grandes objetos desestructurados, que representan imágenes, flujos de video o audio digitalizados, o texto libre. Son los denominados **BLOBS** (objetos binarios grandes, *binary large objects*). Un elemento de datos BLOB normalmente se almacena independientemente de su registro en un almacén de bloques de disco, y en el registro se incluye un puntero al BLOB.

FICHEROS, REGISTROS DE LONGITUD FIJA Y REGISTROS DE LONGITUD VARIABLE

Fichero: es una secuencia de registros. En muchos casos, todos los registros de un fichero son del mismo tipo de registro.

Registros de longitud fija: si cada registro del fichero tiene exactamente el mismo tamaño, se dice que el fichero está compuesto por registros de longitud fija.

Registros de longitud variable: si en el fichero hay registros que tienen tamaños diferentes, el fichero está compuesto por registros de longitud variable. Un fichero puede tener registros de longitud variable por varias razones:

- Los registros del fichero son del mismo tipo de registro, pero uno o más de los campos tienen un tamaño variable (**campos de longitud variable**).
- Los registros del fichero son del mismo tipo de registro, pero uno o más de los campos pueden tener varios valores para registros individuales; un campo así se denomina **campo repetitivo** y un grupo de valores para el campo normalmente se llama **grupo repetitivo**.
- Los registros del fichero son del mismo tipo, pero uno o más de los campos son **opcionales**.
- El fichero contiene registros de **diferentes tipos de registro** y, por tanto, de tamaño variable (**fichero mixto**).

BLOQUEO DE REGISTROS Y REGISTROS EXTENDIDOS A NO EXTENDIDOS

Los registros de un fichero deben asignarse a bloques de disco porque un bloque es la *unidad de transferencia de datos* entre el disco y la memoria. Cuando el tamaño del bloque es mayor que el tamaño del registro, un bloque puede contener varios registros, aunque algunos ficheros pueden tener registros que por su tamaño no entran en un bloque. Supongamos que el tamaño de bloque es de B bytes. En el caso de un fichero de registros con una longitud fija de R bytes, siendo $B \geq R$, tenemos que $bfr = [B/R]$ registros por bloque donde bfr se conoce como **factor de bloqueo** del fichero. En general, R no divide B con exactitud, por lo que en cada bloque tenemos algo de espacio inutilizado igual a: $B - (bfr * R)$ bytes

Para utilizar el espacio desaprovechado, se puede almacenar parte de un registro en un bloque y el resto en otro. Un **puntero** al final del primer bloque apunta al bloque que contiene el resto del registro, en caso de que no sean bloques consecutivos en el disco. Esta organización se denomina **extendida** porque los registros pueden abarcar más de un bloque. Siempre que los registros son más grandes que un bloque debemos usar esta organización. Si los registros no tienen permitido sobrepasar el tamaño del bloque, se denomina **no extendida**. Esta se utiliza con los registros de longitud fija cuando $B > R$, porque hace que cada registro empiece en una ubicación conocida del bloque, simplificándose el procesamiento del registro. En el caso de los registros de longitud variable, se puede utilizar cualquiera de las dos

organizaciones. Si el registro medio es grande, es recomendable utilizar la organización extendida para reducir la pérdida de espacio en cada bloque.

Factor de bloqueo: $bfr = \text{tamaño de bloques} / \text{longitud de registros}$. Representa la cantidad media de registros por bloques de ese fichero.

ASIGNACIÓN DE BLOQUES DE FICHERO EN UN DISCO

Asignación continua: los bloques del fichero se asignan a bloques de discos consecutivos. De este modo, la lectura del fichero entero es mucho más rápida mediante el buffer doble, pero se hace más complejo expandir el fichero.

Asignación enlazada: cada bloque del fichero contiene un puntero al siguiente bloque del fichero. Esto hace más sencilla la expansión del fichero, pero más lenta la lectura del fichero entero.

Clústeres: es una combinación de las dos, asigna clústeres de bloque de disco consecutivos, y los clústeres se enlazan. En ocasiones, los clústeres se les denominan **segmentos de ficheros** o **extensiones**.

Asignación indexada: uno o más bloques de índice contienen punteros a los bloques del fichero.

También se utilizan combinaciones de estas técnicas.

CABECERAS DE FICHERO

Una **cabecera de fichero** o **descriptor del fichero** contiene información sobre un fichero que los programas del sistema necesitan para acceder a los registros. La cabecera incluye información para determinar las direcciones de disco de los bloques del fichero y las descripciones de formato de registro.

OPERACIONES SOBRE FICHEROS

Las operaciones sobre ficheros se pueden agrupar en **operaciones de recuperación** y **operaciones de actualización**.

- **Operaciones de recuperación:** no cambian ningún dato del fichero, puesto que únicamente localizan ciertos registros para que los valores de sus campos se puedan examinar o procesar.
- **Operaciones de actualización:** modifican el fichero mediante la inserción o la eliminación de registros, o modificando los valores de los campos.

Las operaciones de localización y acceso a los registros de un fichero varían de un sistema a otro. Algunas operaciones representativas a todos ellos son:

- **Open (abrir):** prepara el fichero para la lectura o escritura. Asigna los búferes apropiados (normalmente, al menos dos) para albergar los bloques del fichero, y recupera su cabecera. Establece el puntero del fichero al principio del mismo.
- **Reset (reiniciar):** hace que el puntero de un fichero abierto apunte al principio del fichero.
- **Find (o Locate) (buscar):** busca el primer registro que satisface una condición de búsqueda. Transfiere el bloque que contiene ese registro a un búfer de la memoria principal (si todavía no está en el búfer). El puntero del fichero apunta al registro del búfer, que se convierte en el registro actual.
- **Read (o Get) (leer u obtener):** copia el registro actual desde el búfer a una variable de programa del programa de usuario.
- **FindNext (buscar siguiente):** busca el siguiente registro que satisface la condición de búsqueda. Transfiere el bloque en el que se encuentra el registro a un búfer de la memoria principal (si todavía no se encuentra aquí). El registro se almacena en el búfer y se convierte en el registro actual.
- **Delete (borrar):** borra el registro actual y actualiza el fichero en disco para reflejar el borrado.
- **Modify (modificar):** modifica los valores de algunos campos del registro actual y actualiza el fichero en disco para reflejar la modificación.

- **Insert (insertar):** inserta un registro nuevo en el fichero localizando el bloque donde se insertará el registro, transfiriendo ese bloque a un búfer de la memoria principal (si todavía no se encuentra allí), escribiendo el registro en el búfer y escribiendo el búfer en el disco para reflejar la inserción.
- **Close (cerrar):** completa el acceso al fichero liberando los búferes y ejecutando cualquier otra operación de limpieza necesaria.

Las operaciones anteriores (excepto abrir y cerrar) se conocen como **operaciones de un registro por vez**, porque cada operación se aplica a un solo registro. Es posible aglutinar las operaciones Find, FindNext y Read en una sola operación:

- **Scan:** si el fichero simplemente se ha abierto o reiniciado, Scan devuelve el primer registro; en caso contrario, devuelve al siguiente registro. Si con la operación especificamos una condición el registro devuelto es el primero o el siguiente que satisface esa condición.

En los sistemas de bases de datos se pueden aplicar operaciones de grupo al fichero como:

- **FindAll (buscar todo):** localiza todos los registros del fichero que satisfacen una condición de búsqueda.
- **Find (o Locate) n (buscar n):** busca el primer registro que satisface una condición de búsqueda y, después, continúa localizando los $n-1$ siguientes registros que satisfacen la misma condición. Transfiere los bloques que contienen los n registros a un búfer de memoria principal (si todavía no están allí).
- **FindOrdered (buscar ordenados):** recupera todos los registros del fichero en un orden específico.
- **Reorganize (reorganizar):** inicia el proceso de reorganización. Algunas organizaciones de ficheros requieren una reorganización periódica.

ORGANIZACIÓN DE FICHEROS VS. MÉTODOS DE ACCESO

Podemos ver que hay una clara diferencia entre organización del fichero y método de acceso.

Organización de fichero: se refiere a la organización de los datos de un fichero en registros, bloques y estructuras de acceso; esto incluye la forma en que los registros y los bloques se colocan e interconectan en el medio de almacenamiento.

Método de acceso: por el contrario, proporciona un grupo de operaciones (como las anteriores) que se pueden aplicar a un fichero. En general, es posible aplicar varios métodos de acceso a una organización de fichero. Algunos, no obstante, sólo se pueden aplicar a ficheros organizados de determinadas formas.

ORGANIZACIÓN DE FICHEROS

1. Ficheros de registros desordenados (ficheros heap).
2. Ficheros de registros ordenados (ficheros ordenados).
3. Técnicas de dispersión.

FICHERO DE REGISTROS DESORDENADOS (FICHEROS HEAP O PILA)

Es el tipo de organización más sencillo y básico. 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. Esta organización se conoce como **fichero heap o pila**.

La inserción de un registro nuevo es muy eficaz. El último bloque del 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, implica una **búsqueda lineal**, bloque a bloque por todo el fichero (un procedimiento muy costoso).

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, dejando espacio inutilizado. 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). Los programas de búsqueda solo consideran los registros válidos cuando realizan sus búsquedas. Estas dos técnicas de borrado requieren una **reorganización periódica del fichero** para reclamar el espacio inutilizado correspondiente a los registros borrados. Durante la reorganización, se accede consecutivamente a los bloques del fichero, y los registros se empaquetan eliminando los registros borrados. Tras la reorganización, los bloques se llenan de nuevo en toda su capacidad. Otra posibilidad es utilizar el espacio de los registros borrados al insertar registros nuevos, aunque esto requiere una contabilidad adicional para hacer un seguimiento de las ubicaciones vacías.

INSERCIÓN	Muy eficaz. Los registros se guardan en el mismo orden en que se insertan.
BÚSQUEDA	Muy costosa. Es una búsqueda lineal, bloque a bloque.
ELIMINACIÓN	Eliminación de registros del fichero (derroche de espacio) o por marcación de borrado. Necesitan reorganización periódica del fichero para reclamar el espacio inutilizado.

FICHEROS DE REGISTROS ORDENADOS (FICHEROS ORDENADOS)

Los registros de fichero se pueden ordenar físicamente en el disco en función de los valores de uno de sus campos, denominado **campo de ordenación**. Esto conduce a un **fichero ordenado o secuencial**. Si el campo de ordenación también es un **campo clave** (garantiza un valor único en cada registro) del fichero, entonces el campo se denomina **clave de ordenación** del fichero. Las ventajas de los registros ordenados son que, en primer lugar, 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. En segundo lugar, encontrar el siguiente registro al actual según el orden de la clave 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). En tercer lugar, 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, que constituye una mejora respecto a las búsquedas lineales, aunque no se utiliza habitualmente para los ficheros de disco.

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 costosas, ya que siempre deben permanecer los registros ordenados físicamente. 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.

Una manera de facilitar la inserción es reservar algo de espacio sin usar en cada bloque de cara a los registros nuevos. Sin embargo, una vez que se agota este espacio, resurge el problema original. Otra técnica es crear un fichero temporal desordenado llamado **fichero de desbordamiento (overflow)** o **transacciones**. Con esta técnica, el fichero ordenado actual se denomina **fichero principal o maestro**. Los registros nuevos se insertan al final del fichero de desbordamiento, en lugar de hacerlo en su posición correcta en el fichero principal. Periódicamente, el fichero de desbordamiento se ordena y mezcla con el maestro durante la reorganización del fichero. La inserción es muy eficaz, pero a costa de aumentar la complejidad del algoritmo de búsqueda. La búsqueda en el fichero de desbordamiento

debe hacerse con una búsqueda lineal si, después de la búsqueda binaria, no se encuentra el registro en el fichero principal. En las aplicaciones que no requieren la mayoría de la información actualizada pueden ignorarse los registros de desbordamiento durante una búsqueda.

La modificación del valor de un campo depende de dos factores: la condición de búsqueda para localizar el registro y el campo que se va a modificar. Si la condición de búsqueda involucra al campo clave de ordenación, podemos localizar el registro utilizando una búsqueda binaria; en caso contrario, debemos hacer una búsqueda lineal. Un campo no ordenado se puede modificar cambiando el registro y reescribiendo en la misma ubicación física del disco (si los registros son de longitud fija). La modificación del campo de ordenación significa que el registro puede cambiar su posición en el fichero. Esto requiere la eliminación del registro antiguo, seguida por la inserción del registro modificado.

LECTURA	Muy rápida si es por orden de la clave de ordenación.
INSERCIÓN	Muy costosa. Buscar el lugar a insertar y “mover” todos los otros registros. Se mejor con fichero de <i>overflow</i> , pero se hace complejo el algoritmo.
BÚSQUEDA	Muy rápida si la condición de búsqueda se aplica a la clave de ordenación (búsqueda binaria). Si no es un campo clave de ordenación se hace búsqueda lineal (lenta).
ELIMINACIÓN	Muy costosa (misma causa inserción). Aunque también se puede usar un marcador de borrado con reorganización periódica.
MODIFICACIÓN	Si se modifica la clave, mismo problema de inserción. Si se modifican otros campos es más rápida, se ubica el registro por la clave.

TÉCNICAS DE DISPERSIÓN

Proporciona un acceso muy rápido a los registros bajo ciertas condiciones de búsqueda. Esta organización se denomina normalmente **fichero disperso** o **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, este campo también es un campo clave del fichero, en cuyo caso se nomina **clave de dispersión** o **clave hash**. La idea 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 que está almacenado el registro.

1. Dispersión interna.
2. Dispersión externa.
3. Técnicas de dispersión que permiten la expansión dinámica del fichero.
 - 3.1. Dispersión extensible.
 - 3.2. Dispersión lineal.

DISPERSIÓN INTERNA

En los ficheros internos, la dispersión se implementa como una **tabla de dispersión** mediante el uso de un array de registros. Vamos a suponer que el índice del array va de 0 a $M-1$, por tanto tenemos M slots cuyas direcciones corresponden a los índices del array. Elegimos una función de dispersión que transforma el valor del campo de dispersión en un entero entre 0 y $M-1$. La función más común es: $h(K) = K \bmod M$, que devuelve el resto del valor entero de un campo de dispersión K después de dividirlo por M ; este valor se utiliza después para la dirección del registro.

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**. Métodos para resolver:

- **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:** 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. Se coloca el registro nuevo en una ubicación de desbordamiento

sin utilizar y se establece 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.

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.

DISPERSIÓN EXTERNA (PARA LOS FICHeros DE DISCO)

Es la dispersión para ficheros de disco. El espacio de direcciones de destino se compone de **cubos**, cada uno de los cuales almacena varios registros. Un cubo puede ser un bloque de disco o un grupo de bloques contiguos. La función de dispersión mapea una clave a un numero de cubo relativo, en lugar de asignar una dirección de bloque absoluta al cubo. Una tabla almacenada en la cabecera del fichero convierte el número de cubo en la correspondiente dirección de bloque del disco.

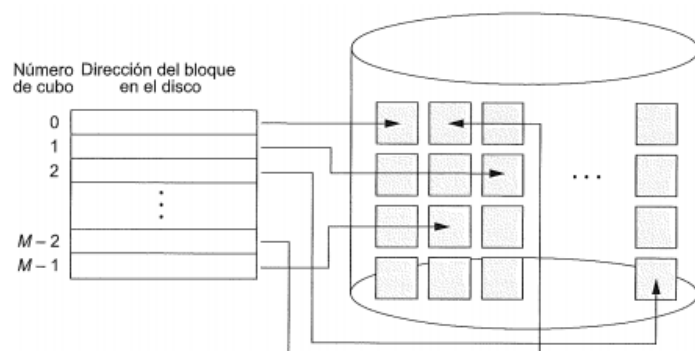
El problema de las colisiones es menos grave con los cubos, porque cuantos más registros encajen en un cubo más posible será que se dispersen al mismo cubo sin causar problemas. Si un cubo se llena y un registro nuevo se dispersa a ese cubo, se puede utilizar una variación del encadenamiento según la cual se mantiene un puntero en cada cubo apuntando a una lista enlazada de registros de desborde para el cubo. Los punteros de la lista enlazada deben ser punteros de registro, que incluyen tanto una dirección de bloque como una posición relativa de registro dentro de bloque.

La dispersión proporciona el acceso posible más rápido para recuperar un registro arbitrario dado el valor de su campo de dispersión. Aunque la mayoría de las funciones de dispersión no mantienen ordenados los registros por los valores del campo de dispersión.

El esquema de dispersión descrito se denomina **dispersión estática**, porque se asigna un número fijo de M cubos. Esto puede ser un gran inconveniente para los ficheros dinámicos. Suponga que asignamos M cubos para el espacio de direcciones y permitimos que m sea el número máximo de registros que podemos encajar en un cubo; por tanto, en el espacio asignado encajarán a lo sumo $(m \cdot M)$ registros. Si el número de registros fuera a ser sustancialmente inferior a $(M \cdot m)$ acabaríamos con mucho espacio desaprovechado. Por el contrario, si el número de registros sobrepasa sustancialmente la cifra de $(M \cdot m)$, tendremos numerosas colisiones y la recuperación será más lenta debido a las largas listas de registros desbordados. En cualquier caso, puede que tengamos que cambiar el número de M bloques asignado y utilizar entonces una nueva función de dispersión (basándonos en el nuevo valor de M) para redistribuir los registros. Estas reorganizaciones pueden consumir mucho tiempo si el fichero es grande. Las organizaciones de fichero dinámicas más nuevas basadas en la dispersión permiten que los cubos varíen dinámicamente solo con la reorganización localizada.

Cuando utilizamos la dispersión externa, la búsqueda de un registro dado el valor de algún otro campo distinto al campo de distribución es tan costosa como en el caso de un fichero desordenado. La eliminación de un registro se puede implementar eliminando el registro de su cubo. Si el cubo tiene una cadena de desbordamiento, podemos mover uno de los registros de desbordamiento del cubo para reemplazar el registro borrado. Si el

registro que vamos a borrar ya está desbordado, simplemente lo eliminamos de la lista enlazada. La eliminación de un registro desbordado implica hacer un seguimiento de las posiciones vacías del desbordamiento. Esto se consigue fácilmente manteniendo



una lista enlazada de ubicaciones de desbordamiento inutilizadas.

La modificación del valor de un campo del registro depende de dos factores: la condición de búsqueda para localizar el registro y el campo que se va a modificar. Si la condición de búsqueda es una comparación de igualdad en el campo de dispersión, podemos localizar el registro eficazmente utilizando la función de dispersión; en caso contrario, debemos realizar una búsqueda lineal. Un campo que no es de dispersión se puede modificar cambiando el registro y reescribiéndolo en el mismo cubo. La modificación del campo de dispersión significa que el registro se puede mover a otro cubo, lo que requiere la eliminación del registro antiguo seguida de la inserción del registro modificado.

BÚSQUEDA	Muy rápida si se aplica la condición de búsqueda al campo de dispersión. Muy lenta (como los ficheros desordenados) cuando se aplica condición de búsqueda a un campo distinto al de dispersión.
ELIMINACIÓN	Si hay desbordamiento, se “mueve” el registro desbordado a donde está el registro a eliminar (en el cubo) y se sobrescribe. Si se elimina un registro desbordado, se lo elimina de la lista enlazada.
MODIFICACIÓN	Depende de la condición de búsqueda y del campo a modificar.

TÉCNICAS DE DISPERSIÓN QUE PERMITEN LA EXPANSIÓN DINÁMICA DEL FICHERO

Intentan remediar el problema de la dispersión estática, en donde resulta difícil expandir o contraer dinámicamente el fichero. El primer esquema, la **dispersión extensible**, almacena una estructura de acceso además del fichero, por lo que se parece a la indexación. La segunda técnica, denominada **dispersión lineal**, no requiere estructuras de acceso adicionales.

- **Dispersión extensible:** consiste en agregar cubos a medida que se necesitan. La principal ventaja de la dispersión extensible es que el rendimiento del fichero no se degrada al crecer, al contrario de lo que ocurre con la dispersión externa estática. Además, en la dispersión extensible no se asigna espacio para un crecimiento futuro, sino que se asignan dinámicamente cubos adicionales a medida que se necesitan. Un inconveniente es que antes de acceder a los propios cubos, debe buscarse el directorio, lo que da lugar a dos accesos de bloque en lugar de uno, como ocurre en la dispersión estática. Esta penalización en el rendimiento se considera menor y, por tanto, se considera que este esquema es muy recomendable para los ficheros dinámicos.
- **Dispersión lineal:** al llenarse un cubo, el primer cubo (cubo 0) se divide en dos, y los registros de ese cubo se distribuyen entre los dos cubos según una función de dispersión.

ESTRUCTURAS DE INDEXACIÓN PARA LOS FICHEROS

Las estructuras de índice son estructuras de acceso auxiliares que se utilizan para acelerar la recuperación de registros en respuesta a ciertas condiciones de búsqueda. Proporcionan rutas de acceso, que ofrecen formas alternativas de acceder a los registros sin que se afecte la ubicación física de los registros en el disco. Permiten un acceso eficaz a los registros basándose en la indexación de los campos que se utilizan para construir el índice. Básicamente, es posible utilizar cualquier campo del fichero para crear un índice y se pueden construir varios índices con diferentes campos en el mismo fichero.

CLASIFICACIÓN DE ÍNDICES

- **Índices densos:** tiene una entrada de índice por cada valor de clave de búsqueda (y, por tanto, cada registro) del fichero de datos. *Una entrada por cada clave de indexación.*
- **Índices no densos o escasos:** solo tiene entradas para algunos de los valores de búsqueda. *Solo tiene entradas para algunos valores de la clave de indexación.*

TIPOS DE ÍNDICES ORDENADOS DE UN NIVEL

La idea tras la estructura de acceso de un índice ordenado es parecida a la que hay tras el índice de un libro.

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 de índice están ordenados de manera que se puede hacer una búsqueda binaria en el índice. El fichero de índice es mucho más pequeño que un 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, como se verá más adelante.

1. Índices principales.
2. Índices agrupados.
3. Índices secundarios.

ÍNDICES PRINCIPALES

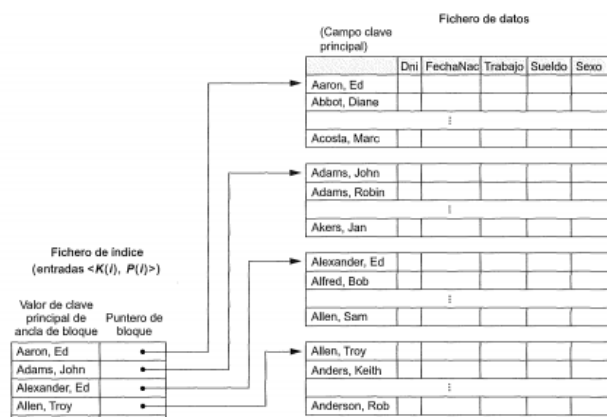
Un **índice principal** o **primario** se especifica en el campo clave de ordenación de un fichero ordenado de registros, es decir que, no contiene valores repetidos.

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 o simplemente **ancla de bloque**.

Un índice principal es un índice escaso, 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.

La inserción y el borrado de registros supone un gran problema, ya que para insertar un registro en su posición correcta dentro del fichero de datos, no sólo se deben mover los registros para hacer sitio al registro nuevo, sino que también hay que cambiar algunas entradas del índice, puesto que al mover los registros modificaremos los registros anclas de algunos bloques. El uso de un fichero de desbordamiento puede minimizar el problema. Otra posibilidad es utilizar una lista de enlaces de registros



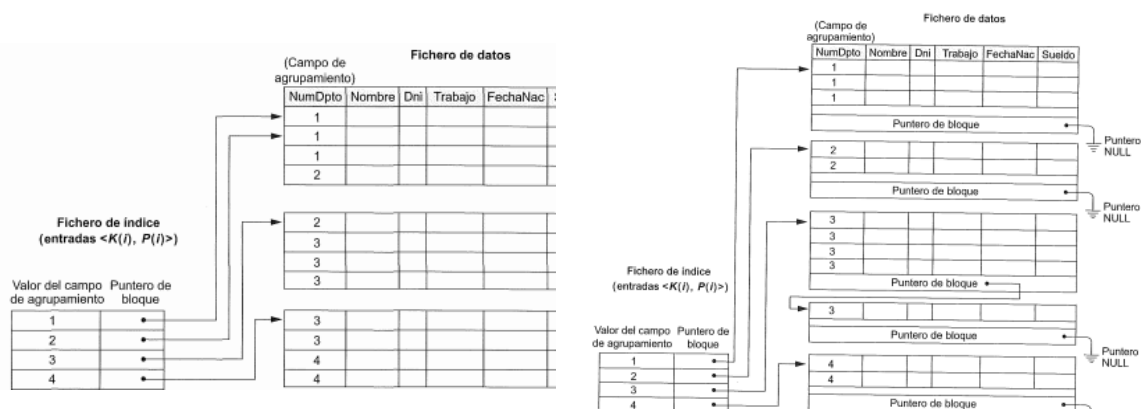
desbordados por cada bloque del fichero de datos. El borrado de un registro se realiza con marcadores de borrado.

ÍNDICES AGRUPADOS

Si el campo de ordenación física no es campo clave, es decir que varios ficheros tiene el mismo valor para el campo de ordenación, se utiliza otro tipo de índice denominado **índice agrupado**. El campo no clave se denomina **campo agrupado**. Un fichero puede tener como máximo un campo de ordenación física, por lo que puede tener un índice principal o un índice agrupado, pero no ambos. 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. Este es otro ejemplo de un índice no denso porque tiene una entrada por cada valor distinto del campo de indexación, que no es una clave por definición, y que por lo tanto, tiene valores duplicados en lugar de un valor único por cada registro del fichero.

Ocupa menos espacio que el fichero de datos y contiene registros de longitud fija.

La inserción y el borrado provocan problemas ya que los registros están ordenados físicamente. Para alivianar el problema 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.



ÍNDICES SECUNDARIOS

Un tercer índice es el **índice secundario**, que se puede especificar sobre cualquier campo no ordenado del fichero. Un fichero puede tener varios índices secundarios además de su método de acceso principal, es decir, puede haber muchos índices secundarios.

Proporciona un medio secundario de acceso a un fichero para el que ya existe algún acceso principal. El índice secundario puede ser un campo que es una clave candidata y que 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 dato 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*.

La estructura de acceso de un índice secundario en un *campo clave* tiene un *valor distinto* por cada registro, este recibe 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. Por tanto, dicho índice es denso.

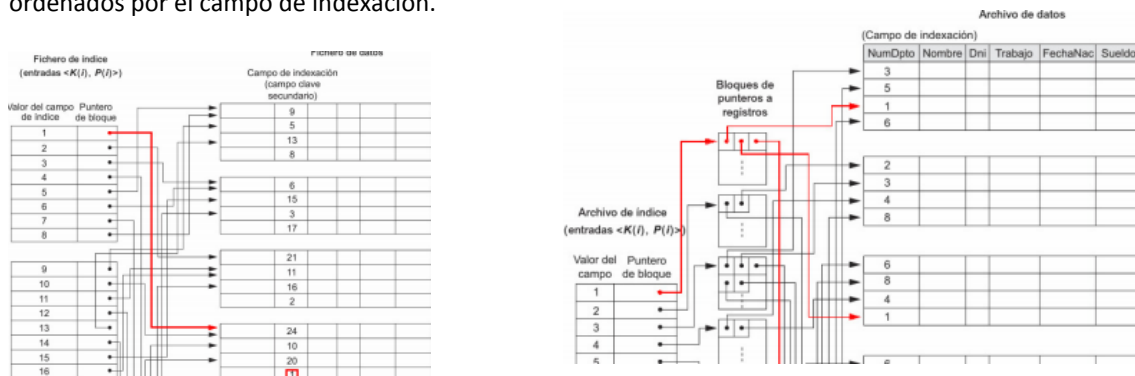
Como los registros del fichero de datos no están físicamente ordenados por los valores del campo clave secundario, no podemos utilizar las anclas de bloque. Por eso creamos una entrada de índice por cada registro del fichero de datos, en lugar de hacerlo por cada bloque, como en el caso de un índice principal.

Un índice secundario normalmente necesita más espacio de almacenamiento y un tiempo de búsqueda mayor que un índice principal, debido a su mayor cantidad de entradas. Sin embargo, la mejora en el tiempo de búsqueda de un registro arbitrario es mucho mayor para un índice secundario que para un índice principal, puesto que tendríamos que hacer una búsqueda lineal en el fichero de datos si no existiera el índice secundario.

Si el índice es *por campo no clave*, varios registros del fichero de datos pueden tener el mismo valor para el campo de indexación. Hay 3 opciones para implementar dicho índice:

1. Incluir varias entradas de índice con el mismo valor, una por cada registro, es decir, generar un índice denso con un registro por cada valor.
2. Tener registros de longitud variable para las entradas del índice, con un campo repetitivo para el puntero. Guardamos una lista de punteros en la entrada de índice.
3. Es la más utilizada y consiste en mantener las entradas del índice con una longitud fija y disponer de una sola entrada por cada valor del campo de índice, pero crear un nivel de indirección extra para manipular los múltiples punteros. Es un índice no denso, donde cada registro almacena punteros a bloque de punteros y estos apuntan a registros. Si algún valor se repite en demasiados registros, de modo que sus punteros de registro no pueden encajar en un solo bloque de disco, se utiliza un grupo o lista enlazada de bloques.

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.



ÍNDICES MULTINIVEL

Un índice de multinivel considera el fichero índice, al que ahora nos referimos como **primer nivel** o **base** de un índice multinivel, como un fichero ordenado con un valor distinto por cada $K(i)$, por lo tanto, podemos crear un índice principal para el primer nivel; este índice al primer nivel se lo 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. Las entradas son de longitud fija. El factor de bloqueo bfr_i 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 tiene el mismo tamaño, cada una con un valor del campo y una dirección de bloque.

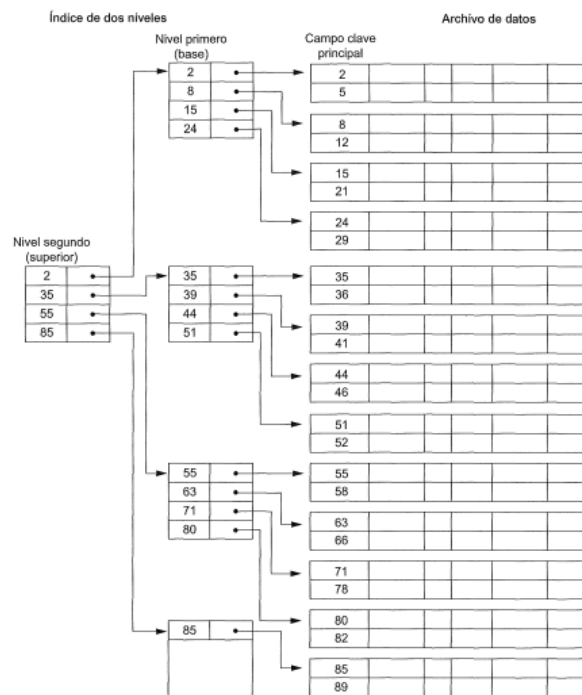
Podemos repetir este proceso para el segundo nivel. El **tercer nivel**, que es un índice principal para el segundo nivel, tiene una entrada por cada bloque de segundo nivel, tiene una entrada por cada bloque de segundo nivel. Se necesita un segundo nivel sólo si el primer nivel necesita más de un bloque de almacenamiento en disco y, de forma parecida, requerimos un tercer nivel sólo si el segundo nivel necesita más de un bloque. Podemos repetir el proceso anterior 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**.

El esquema multinivel puede utilizarse con 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.

Un índice multinivel reduce el número de accesos a bloques cuando se busca un registro, dado el valor de su campo de indexación. Todavía nos enfrentamos con los problemas de tratar con las inserciones y los borrados en el índice, porque todos los niveles de índice son ficheros ordenados físicamente.

- Se aplican después de generar índice (principal, agrupado, secundario).
- Ese índice de primer nivel es sobre el campo clave (valores no repetidos) y entradas de longitud fija.
- Genera segundo nivel principal: toma al primer nivel como fichero ordenado con valor distinto en clave.
- El segundo nivel posee una entrada por bloque.
- Así sucesivamente hasta que las entradas de un mismo nivel se almacenan en un bloque.

Figura 14.6. Índice principal de dos niveles que se parece a la organización ISAM.



ÍNDICES MULTINIVEL DINÁMICOS UTILIZANDO ÁRBOLES B Y B+

Los árboles B y B+ son casos especiales de estructuras de datos en forma de árbol bien conocidas. Un **árbol** está formado por **nodos**. Cada nodo del árbol, excepto el nodo especial denominado **raíz**, tiene un nodo padre y varios (ninguno o más) nodos **hijo**. El nodo raíz no tiene padre. Un nodo que no tiene hijos, se llama nodo **hoja**, un nodo que no es hoja es un nodo **interno**. El nivel de un nodo siempre es uno más que el nivel de su padre, siendo cero el nivel del nodo raíz. Un subárbol de un nodo consta de ese nodo y de todos sus nodos descendientes. Una forma de implementar un árbol es con tantos punteros en cada nodo como nodos hijos tiene ese nodo. En algunos casos, también se almacena un puntero padre en cada nodo. Además de los punteros, un nodo normalmente contiene algún tipo de información almacenada.

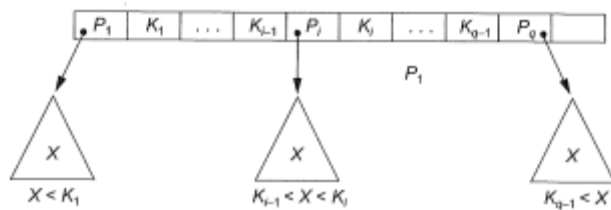
ÁRBOLES DE BÚSQUEDA

Un árbol de búsqueda es un tipo especial de árbol que sirve para guiar la búsqueda de un registro, dado el valor de uno de sus campos.

Un árbol de búsqueda de orden p es un árbol tal que cada nodo contiene, a lo sumo, $p-1$ valores de búsqueda y p punteros en el orden $\langle P_1, K_1, P_2, K_2, \dots, P_{q-1}, K_{q-1}, P_q \rangle$, donde $q \leq p$; cada P_i es un puntero a un nodo hijo (o puntero NULL o nulo); y cada K_i es un valor de búsqueda proveniente de algún conjunto ordenado de valores. Se supone que todos los valores de búsqueda son únicos. Un árbol de búsqueda debe cumplir en todo momento dos restricciones:

1. Dentro de cada nodo, $K_1 < K_2 < \dots < K_{q-1}$

- Para todos los valores de X del subárbol al cual apunta P_i , tenemos que $K_{i-1} < X < K_i$ para $1 < i < q$; $X < K_i$ para $i=1$; y $K_{i-1} < X$ para $i=q$

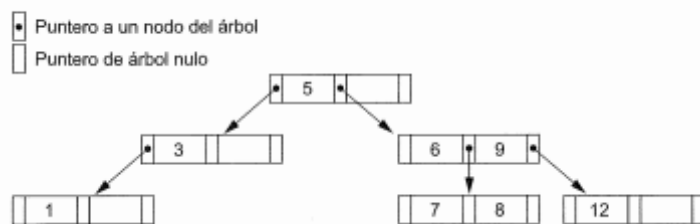


Siempre que busquemos un valor X , seguimos el puntero P_i apropiado, de acuerdo con las fórmulas de la condición 2. Algunos de los punteros P_i de un nodo pueden ser punteros nulos.

Podemos usar un árbol de búsqueda como mecanismo para buscar registros almacenados en un fichero de disco. Los valores del árbol pueden ser los valores de uno de los campos del registro, el llamado campo de búsqueda (que es el mismo que el campo de índice si un índice multinivel guía la búsqueda). Cada valor clave del árbol está asociado a un puntero de registro que tiene ese valor en el fichero de datos. El árbol de búsqueda en sí puede almacenarse en disco, asignando cada nodo del árbol a un bloque del disco. Cuando se inserta un registro nuevo, es preciso actualizar el árbol de búsqueda incluyendo en él una entrada con el valor del campo de búsqueda del nuevo registro y un puntero a éste.

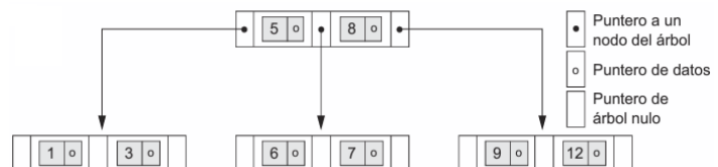
Para insertar valores de búsqueda en el árbol y eliminarlos, sin violar las dos restricciones anteriores, se utilizan algoritmos que, por regla general, no garantizan que el árbol de búsqueda este equilibrado (es decir que todas las hojas estén al mismo nivel). Es importante mantener equilibrados los árboles de búsqueda porque esto garantiza que no habrá nodos en niveles muy profundos que requieran muchos accesos a bloques durante una búsqueda. Los árboles equilibrados procuran una velocidad de búsqueda uniforme independientemente del valor de la clave de búsqueda. Además, las eliminaciones de registros pueden hacer que queden nodos casi vacíos, con lo que hay un desperdicio de espacio importante y un aumento en el número de niveles. El árbol B hace frente a estos dos problemas especificando restricciones adicionales en el árbol de búsqueda.

Árbol de búsqueda de orden $p = 3$.



ÁRBOL B

El árbol B tiene restricciones adicionales que garantizan que el árbol siempre estará equilibrado y que el espacio desperdiciado por la eliminación, si lo hay, nunca será excesivo. Los algoritmos para insertar y eliminar son más complejos para poder mantener estas restricciones. No obstante, la mayor parte de las inserciones y eliminaciones son procesos simples que se complican solo en circunstancias especiales (por ejemplo, siempre que intentamos una inserción en un nodo que ya está lleno o una eliminación de un nodo que está a menos de la mitad de su capacidad). De manera más formal, un árbol B de orden p , utilizado como estructura de acceso según un campo clave para buscar registros en un fichero de datos, se puede definir de este modo:



- Cada nodo interno del árbol B tiene la forma $\langle P_1, \langle K_1, Pr_1 \rangle, P_2, \langle K_2, Pr_2 \rangle, \dots, \langle K_{q-1}, Pr_{q-1} \rangle, P_q \rangle$ donde $q \leq p$. Cada P_i es un puntero de árbol (un puntero a otro nodo del árbol B). Cada Pr_i es un puntero de datos (un puntero al registro cuyo valor del campo clave de búsqueda es igual a K_i [o al bloque del fichero de datos que contiene ese registro])
- Dentro de cada nodo $K_1 < K_2 < \dots < K_{q-1}$
- Para todos los valores del campo clave de búsqueda X del subárbol al que apunta P_i (el subárbol i), tenemos: $K_{i-1} < X < K_i$ para $1 < i < q$; $X < K_i$ para $i=1$; y $K_{i-1} < X$ para $i=q$.

4. Cada nodo tiene a lo sumo p punteros de árbol
5. Cada nodo, excepto los nodos raíz y hoja, tiene por lo menos $p/2$ punteros de árbol. El nodo raíz tiene como mínimo 2 punteros de árbol, a menos que sea el único nodo del árbol.
6. Un nodo con q punteros de árbol, $q \leq p$, tiene $q-1$ valores del campo clave de búsqueda (y, por tanto, tiene $q-1$ punteros de datos).
7. Todos los nodos hoja están en el mismo nivel. Los nodos hoja tienen la misma estructura que los internos, excepto que todos sus punteros de árbol P_i son nulos. y sus punteros de árbol son nulos

Todos los valores de búsqueda K del árbol B son únicos si el árbol se utiliza como una estructura de acceso en un campo clave. Si utilizamos un árbol B en un campo que no es clave, debemos cambiar la definición de los punteros de fichero P_i para que apunten a un bloque (o grupo de bloques) que contienen los punteros a los registros del fichero.

Un árbol B empieza con un solo nodo raíz (que también es un nodo hoja) en el nivel 0(cero). Una vez que el nodo raíz está lleno con $p-1$ valores de clave de búsqueda e intentamos insertar otra entrada en el árbol, el nodo raíz se divide en dos nodos en el nivel 1. En el nodo raíz solo queda el valor central, y el resto de valores se dividen uniformemente entre los otros dos nodos. Cuando un nodo no raíz está lleno y se inserta en él una entrada nueva, dicho nodo se divide en otros dos al mismo nivel, y la entrada central se mueve al nodo padre junto con dos punteros a los nuevos nodos divididos. Si el nodo padre está lleno, también es dividido. La división puede propagarse hasta llegar al nodo raíz, creando un nuevo nivel si se divide la raíz.

Si la eliminación de un valor provoca que un nodo no llegue a la mitad de su capacidad, es combinado con sus nodos vecinos, algo que también puede propagarse hasta la raíz. Por tanto, la eliminación puede reducir el número de niveles de árbol. Con ayuda de análisis y simulaciones se ha demostrado que, después de numerosas inserciones y eliminaciones aleatorias en un árbol B , los nodos quedan ocupados aproximadamente al 69% de su capacidad cuando se estabiliza el número de valores en el árbol. Esto también es cierto con los árboles $B+$. Si ocurre esto, la división y combinación de nodos solo se dará raramente, por lo que la inserción y la eliminación son muy eficaces. Si el número de valores aumenta, el árbol se expandirá sin problemas (aunque podría darse la división de nodos, de modo que algunas inserciones tardarían algo más de tiempo).

Los árboles B se utilizan a veces como organizaciones de fichero principales. En este caso, dentro de los nodos del árbol B se guardan registros enteros, en lugar de entradas <clave de búsqueda, puntero de registro>. Esto funciona bien para ficheros con un número relativamente pequeño de registros y un tamaño pequeño de registro. En caso contrario, el acceso no es eficaz.

En resumen, los árboles B proporcionan una estructura de acceso multinivel que es una estructura de árbol equilibrado en la que cada nodo está, como mínimo, medio lleno. Cada nodo de un árbol B de orden p puede tener como máximo $p-1$ valores de búsqueda.

SEGURIDAD EN LAS BASES DE DATOS

TIPOS DE SEGURIDAD

La seguridad en las bases de datos es un tema muy amplio que comprende muchos conceptos, entre los cuales se incluyen los siguientes:

- **Aspectos legales y éticos** en relación con el derecho de acceso a determinada información. Ciertos tipos de información están considerados como privados y no pueden ser accedidos legalmente por personas no autorizadas. (Ejemplo: leyes que regulan la privacidad de la información).

- Temas de **políticas a nivel gubernamental, institucional o de empresas** en relación con los tipos de información que no debería estar disponible públicamente. (Ejemplo: créditos o informes médicos personales).
- Temas **relativos al sistema**, como los **niveles del sistema** en los que se debería reforzar las distintas funciones de seguridad. (Ejemplo: hardware, software, DBMS)
- Necesidad dentro de la organización de **identificar niveles de seguridad** y de clasificar según éstos a los datos y a los usuarios: por ejemplo, alto secreto, secreto, confidencial y no clasificado.

AMENAZAS A LAS BASES DE DATOS

Las amenazas a las bases de datos tienen como consecuencia la pérdida o la degradación de todos o de algunos de los siguientes objetivos de seguridad comúnmente aceptados: integridad, disponibilidad y confidencialidad.

- **Pérdida de integridad:** la integridad de la base de datos tiene relación con el requisito de cumplir de que la información se encuentre protegida frente a modificaciones inadecuadas. La modificación de datos incluye la creación, inserción, modificación y el borrado de los datos. Estos cambios pueden ocurrir de manera intencional o accidental. El uso continuado de un sistema “contaminado” o de datos corrompidos podría tener como consecuencia la toma de decisiones inexactas, fraudulentas o erróneas.
- **Pérdida de disponibilidad:** la disponibilidad de la base de datos tiene relación con que los objetos estén disponibles para un usuario humano o para un programa que tenga los derechos correspondientes.
- **Pérdida de confidencialidad:** la confidencialidad de la base de datos tiene relación con la protección de los datos frente al acceso no autorizado.

El DBMS debe proporcionar técnicas que permitan a determinados usuarios o grupos de usuarios el acceso a partes concretas de una base de datos sin tener acceso al resto.

Un DBMS incluye, por lo general, **un subsistema de autorizaciones y seguridad en base de datos** responsable de garantizar la seguridad de partes de una base de datos frente accesos no autorizados.

MECANISMOS DE SEGURIDAD EN BASES DE DATOS

- **Mecanismos de seguridad discrecionales:** se utilizan para conceder permisos a usuarios, incluyendo la capacidad de acceso a determinados archivos de datos, registros o campos en un modo en concreto (lectura, inserción, borrado o actualización). Son flexibles y vulnerables.
- **Mecanismos de seguridad obligatorios:** se utilizan para reforzar la seguridad a varios niveles mediante la clasificación de datos y de los usuarios en varias clases de seguridad (o niveles) para después implementar la política de seguridad adecuada a la organización. Son rígidos y de mayor protección.

MEDIDAS DE CONTROL

Para proteger las bases de datos contra las amenazas es habitual implementar cuatro tipos de medidas de control:

- **Control de accesos:** prevenir que personas no autorizadas tengan acceso al sistema, ya sea para obtener información o para realizar cambios malintencionados. El mecanismo de seguridad de un DBMS debe incluir medidas para restringir el acceso al sistema de bases de datos en su totalidad. El control de acceso se gestiona mediante la creación de cuentas de usuario y contraseñas para controlar el proceso de entrada al DBMS.
- **Control de inferencias:** las bases de datos estadísticas se utilizan para proporcionar información estadística o resúmenes de valores basados en diversos criterios. Los usuarios de bases de datos estadísticas, tienen privilegios para acceder a la base de datos para obtener

información estadística sobre un determinado grupo de población, pero no para acceder a la información confidencial detallada sobre individuos en particular. La seguridad en las bases de datos estadísticas debe garantizar que la información relativa a los individuos no pueda ser accesible. A veces es posible deducir ciertos datos acerca de los individuos a partir de las consultas relativas únicamente a valores resumen sobre grupos; esto tampoco se debe permitir.

- **Control de flujo:** previene que la información fluya de tal manera que llegue a usuarios no autorizados.
- **Cifrado de datos:** es una medida para proteger datos confidenciales (como los números de las tarjetas de crédito) que sean transmitidos a través de algún tipo de red de comunicación. Los datos se codifican utilizando algún algoritmo de codificación o cifrado. Un usuario no autorizado que acceda a datos codificados tendrá dificultades para descifrarlos, pero a los usuarios autorizados se les proporcionarán algoritmos de descodificación o descifrado (claves) para descifrar los datos.

LA SEGURIDAD EN BASES DE DATOS Y EL DBA

El DBA es el responsable principal de la gestión de un sistema de base de datos. Entre las responsabilidades del DBA se encuentran la concesión y retirada de privilegios a los usuarios que necesitan utilizar el sistema, y la clasificación de usuarios y datos en función de la política de la organización. El DBA dispone de una cuenta de DBA en el DBMS, llamada también a veces cuenta de sistema o de superusuario, que proporciona enormes posibilidades que no están disponibles para las cuentas y los usuarios normales de la base de datos. Entre los comandos privilegiados a nivel del DBA se encuentran los comandos para conceder o retirar privilegios a las cuentas o usuarios individuales o a los grupos de usuarios y para ejecutar los siguientes tipos de acciones:

- **Creación de cuentas:** mediante esta acción se crea una nueva cuenta y contraseña para posibilitar el acceso al DBMS a un usuario o grupo de usuarios.
- **Concesión de privilegios:** esta acción permite al DBA conceder determinados permisos a determinadas cuentas.
- **Retirada de privilegios:** esta acción permite al DBA retirar (cancelar) determinados permisos concedidos previamente a determinadas cuentas.
- **Asignación de nivel de seguridad:** consiste en asignar cuentas de usuario al nivel de clasificación de seguridad adecuado.

AUDITORÍAS DE BASES DE DATOS

Para mantener un registro de todas las actualizaciones realizadas en la base de datos y de los usuarios en particular que realizaron cada actualización, podemos modificar el registro de sucesos del sistema (*system log*).

Si se sospecha de alguna modificación malintencionada, se realizará una **auditoría de la base de datos**, que consiste en la revisión del archivo de registro para examinar todos los accesos y operaciones realizadas sobre la base de datos durante un periodo de tiempo, en busca de la cuenta que realizó tal operación.

Aun registro de una base de datos que se utilice principalmente para propósitos de seguridad se le llama, a veces, **registro de auditoría**.

CONTROL DE ACCESO DISCRECIONAL BASADO EN CONCESIÓN Y REVOCACIÓN DE PRIVILEGIOS

El método habitual para reforzar el control de acceso discrecional en un sistema de base de datos se basa en la concesión y revocación de privilegios.

TIPOS DE PRIVILEGIOS DISCRECIONALES

El DBMS debe proporcionar acceso selectivo a cada relación de la base de datos basándose en cuentas determinadas. También podemos controlar las operaciones; de este modo, la posesión de una cuenta no significa necesariamente que el propietario de la cuenta esté autorizado a toda la funcionalidad que permita el DBMS. Existen dos niveles de asignación de privilegios de uso del sistema de base de datos:

- **El nivel de cuenta:** en este nivel, el DBA especifica los privilegios en particular que posee cada cuenta, independientemente de las relaciones existentes en la base de datos.
- **El nivel de relación (o tabla):** en este nivel, el DBA puede controlar el privilegio de acceso a cada relación o vista individual en la base de datos.

CONTROL DE ACCESO OBLIGATORIO Y CONTROL DE ACCESO BASADO EN ROLES PARA LA SEGURIDAD MULTINIVEL

En muchas aplicaciones se necesita una política de seguridad adicional que clasifique los datos y los usuarios basándose en clases de seguridad. Este modelo se conoce como control de acceso obligatorio y se combina habitualmente con los mecanismos de control de acceso discrecional.

Las clases de seguridad típicas son (ordenadas desde el nivel más alto al más bajo):

- Alto secreto (TS) TS es el nivel más alto y U el más bajo. El modelo utilizado habitualmente
- Secreto (S) para la seguridad a varios niveles, conocido como modelo Bell-LaPadula,
- Confidencial (C) clasifica cada **sujeto** (usuario, cuenta, programa) y **objeto** (relación, tupla,
- No clasificado (U) columna, vista, operación) en una de las clases típicas, llamadas nivel de autorización.

BASES DE DATOS DISTRIBUIDAS

DDBS (Sistema de bases de datos distribuidas, *Distributed DataBase System*), los DDBMS (Sistema de administración de bases de datos distribuidas, *Distributed DataBase Management System*). La tecnología DDB emergió gracias a la unión de otras dos tecnologías: la de base de datos y la de comunicación de datos y redes.

CONCEPTOS DE BASES DE DATOS DISTRIBUIDAS

Las bases de datos distribuidas aportan las ventajas del procesamiento distribuido al dominio de la administración de una base de datos.

Sistema de computación distribuido: consiste en un número de elementos de procesamiento, no necesariamente homogéneos, interconectados mediante una red de computadoras y que cooperan para la realización de ciertas tareas asignadas. *Conjunto de sitios conectados mediante red de comunicación.*

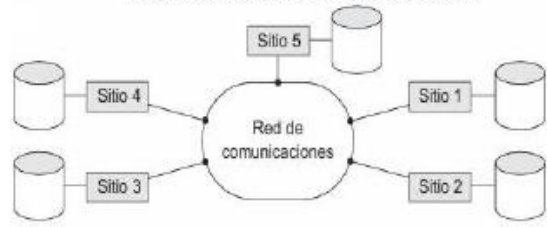
Estos sistemas dividen un gran e inmanejable problema en piezas más pequeñas para resolverlo de una manera coordinada. La viabilidad económica de este planteamiento procede de dos razones: una mayor potencia de computación, y cada elemento de procesamiento autónomo pueda ser administrado de manera independiente y desarrollar sus propias aplicaciones.

DDB (Base de datos distribuida): es una colección de múltiples bases de datos distribuidas interrelacionadas de forma lógica sobre una red de computadoras, y un DDBMS (sistema de administración de bases de datos distribuidas) es el software encargado de administrar la base de datos distribuida mientras hace la distribución transparente para el usuario.

Arquitectura en red con una base de datos centralizada en una de sus ubicaciones.



Arquitectura de base de datos distribuida auténtica.



VENTAJAS DE LAS BASES DE DATOS DISTRIBUIDAS

- **Administración de datos distribuidos con distintos niveles de transparencia:** un DBMS distribuido debe ser una distribución transparente en el sentido de ocultar los detalles de dónde está físicamente ubicado cada fichero (tabla, relación) dentro del sistema.
 - Transparencia de red o de distribución: hace referencia a la autonomía del usuario de los detalles operacionales de la red.
 - Transparencia de localización: el comando usado para realizar una tarea es independiente de la ubicación de los datos y del sistema que ejecutó dicho comando.
 - Transparencia de denominación: implica, que una vez especificado un nombre, puede accederse a los objetos nombrados sin ambigüedad y sin necesidad de ninguna especificación adicional.
 - Transparencias de replicación: permite que el usuario no se entere de la existencia de copias.
 - Transparencia de fragmentación: permite que el usuario no se entere de la existencia de fragmentos.
 - Fragmentación horizontal: distribuye una relación (tabla) en conjuntos de tuplas (filas).
 - Fragmentación vertical: distribuye una relación en subrelaciones (conjuntos de campos o columnas). La clave primaria permanece en todos los fragmentos.
 - Mixta: combinación de las dos anteriores.
 - Transparencia de diseño y de ejecución: hace referencia a la libertad de saber cómo está diseñada la base de datos distribuida y donde ejecuta una transacción.
- **Incremento de la fiabilidad y la disponibilidad:** La **fiabilidad** está definida ampliamente como la probabilidad de que un sistema esté funcionando en un momento del tiempo. La **disponibilidad** es la probabilidad de que el sistema esté continuamente disponible durante un intervalo de tiempo. Cuando los datos y el software DBMS están distribuidos a lo largo de distintas localizaciones, uno de ellos puede fallar, mientras el resto continúa operativo. Sólo los datos y el software almacenados en la localización que falla serán los que no estén disponibles. Esto mejora tanto la fiabilidad como la disponibilidad del sistema, y si se realiza replicación de datos, aún más.
- **Rendimiento mejorado:** un DDBMS fragmenta la base de datos manteniendo la información lo más cerca posible del punto donde es más necesaria. Esto contribuye a mejorar el rendimiento.
- **Expansión más sencilla:** la expansión del sistema en términos de incorporación de más datos, incremento del tamaño de las bases de datos o la adición de más procesadores es mucho más sencilla. La expansión puede ser en datos y procesadores.

FUNCIONES ADICIONALES DE LAS BASES DE DATOS DISTRIBUIDAS

La distribución conlleva a un incremento en la complejidad del diseño del sistema y de su implementación. El software DDBMS debe ser capaz de ofrecer las siguientes funciones además de todas las de un DBMS centralizado:

- **Seguimiento de los datos:** capacidad de controlar la distribución de los datos, la fragmentación, y la replicación expandiendo el catálogo DDBMS.
- **Procesamiento de consultas distribuidas:** posibilidad de acceder a sitios remotos y de transmitir consultas y datos a lo largo de todos esos sitios mediante una red de comunicación.
- **Administración de transacciones distribuidas:** facultad de diseñar estrategias de ejecución de consultas y transacciones que accedan a los datos desde más de una ubicación y de sincronizar el acceso a los datos distribuidos y de mantener la integridad de toda la base de datos.
- **Administración de datos replicados:** capacidad de decidir a qué copia de un dato acceder y de mantener la consistencia de las copias de un elemento de datos replicado.
- **Recuperación de una BD distribuida:** facultad de recuperarse de las caídas de una localización individual u otro tipo de fallos, como los fallos en los enlaces de comunicación.
- **Seguridad:** las transacciones distribuidas deben ejecutarse con una adecuada administración de la seguridad de los datos y contando con los privilegios de autorización/acceso de los usuarios.
- **Administración del directorio (catálogo) distribuido:** un directorio contiene información (metadatos) sobre los datos de la base de datos. Puede ser global a toda DDB, o local para cada sitio. La colocación y distribución del directorio son temas relacionados con el diseño y las políticas.

DIFERENCIAS A NIVEL DE HARDWARE ENTRE BD DISTRIBUIDAS Y BD CENTRALIZADAS

A nivel de **hardware**, los principales factores que distinguen un DDBMS de un sistema centralizado son:

- Existen múltiples computadores llamados **sitios** o **nodos**. Cada sitio es un sistema de BD en sí mismo (hardware, software, usuarios, datos).
- Estos sitios deben estar conectados por algún tipo de red de comunicación para transmitir los datos y los comandos entre ellos.

CONEXIÓN

Estos sitios pueden estar cercanos entre sí (digamos, dentro del mismo edificio o grupo de edificios adyacentes) y conectados mediante una **red de área local**, o estar geográficamente distribuidos a larga distancia y enlazados a través de una **red de área expandida** o *long-haul*. Las redes de área local suelen emplear cables mientras que las *long-haul* utilizan líneas telefónicas o satélites. Es posible también, una combinación de ambas.

Además las redes pueden tener diferentes **topologías** que definen las rutas de comunicación directa entre los sitios. El tipo y la topología pueden tener un efecto significativo sobre el rendimiento.

TÉCNICAS DE FRAGMENTACIÓN DE DATOS

- **Fragmentación Horizontal:** la fragmentación horizontal de una relación es un subconjunto de las tuplas de esa relación. Divide a la relación horizontalmente agrupando filas para crear subconjuntos de tuplas cada uno de ellos con un cierto significado lógico. Para reconstruir una relación fragmentada horizontalmente se necesita aplicar la operación UNION.
- **Fragmentación Vertical:** puede que cada sitio no necesite todos los atributos de una relación, lo que indicaría la necesidad de un tipo distinto de fragmentación. Divide a la relación por columnas. Un fragmento vertical de una relación solo mantiene ciertos atributos de la misma. Es necesario incluir la clave primaria, o alguna otra clave candidata, en cada fragmento vertical de modo que nos permita reconstruir la relación completa a partir de ellos. Para reconstruir una relación fragmentada verticalmente se necesita aplicar la operación OUTER UNION o FULL OUTER JOIN.
- **Fragmentación mixta (híbrida):** utiliza los dos tipos de fragmentación.

REPLICACIÓN Y UBICACIÓN DE LOS DATOS

La replicación es útil para mejorar la disponibilidad de los datos.

- **Replicación total:** es el caso más extremo de replicación. Consiste en la replicación de toda la base de datos en cada sitio del sistema distribuido. Esto puede mejorar considerablemente la disponibilidad, ya que el sistema puede seguir funcionando con tal de que uno de los sitios esté activo. También mejora el rendimiento en la recuperación de consultas globales porque los resultados de este tipo de consultas pueden obtenerse localmente desde cada uno de estos sitios. La principal desventaja de este entorno es que puede ralentizar drásticamente las operaciones de actualización, ya que es preciso ejecutar dicha operación en cada copia de la base de datos para mantener la consistencia de las mismas.
- **No replicación:** es el otro extremo de replicación. Aquí, cada fragmento está almacenado en un sitio (fragmentación). En este caso, todos los fragmentos deben ser disjuntos, excepto en la repetición de claves primarias a lo largo de los fragmentos verticales (o mixtos). Esto suele conocerse también como ubicación no redundante.
- **Replicación parcial:** es un punto intermedio. Algunos fragmentos de la base de datos pueden replicarse mientras que otros no. El número de copias de cada fragmento puede oscilar desde una hasta el número total de sitios del sistema distribuido.

Cada fragmento, o cada copia del mismo, deben estar asignado a un sitio concreto del sistema distribuido. Este proceso recibe el nombre de **distribución de datos** (o **ubicación de datos**). La elección del sitio y el grado de replicación dependen de los objetivos de rendimiento y la disponibilidad del sistema, así como de los tipos y frecuencias de las transacciones efectuadas en cada sitio.

TIPOS DE SISTEMAS DE BASES DE DATOS DISTRIBUIDAS

Lo que todos estos sistemas tienen en común es el hecho de que tanto los datos como el software están distribuidos a lo largo de distintas localizaciones que están conectadas por algún tipo de red de comunicaciones.

- **Según grado de homogeneidad del DDBMS:** si todos los servidores (o DBMS locales individuales) y todos los usuarios (clientes) usan idéntico software, se dice que el DDBMS es **homogéneo**; en cualquier otro caso, es **heterogéneo**.
- **Según el grado de autonomía local:** si no hay previsión de que el sitio funcione como un DBMS aislado, se dice que **no tiene autonomía local**. Por otro lado, si se permite que las transacciones tengan acceso directo a un servidor, el sistema tiene cierto grado de **autonomía local**.

CUESTIONES QUE CAMBIAN ENTRE BD CENTRALIZADA Y BD DISTRIBUIDA

- Costo de transferencia de datos.
- Acceso concurrente y consistencia de datos.
- Fallo de sitios.
- Fallo de enlaces.
- Confirmar actualización en datos de distintos sitios.
- Interbloqueo (*deadlock*): Ocurren con más frecuencia si están “totalmente conectados”.

MODELOS DE DATOS

Modelo de datos: es una colección de conceptos que se pueden utilizar para describir la estructura de una base de datos.

Colección de herramientas conceptuales para:

- Describir datos,
- Relaciones entre datos,
- Definir restricciones.

CLASIFICACIÓN

BASADOS EN OBJETOS

Los modelos lógicos basados en objetos se usan para describir datos en el nivel conceptual y el externo. Se caracterizan porque proporcionan capacidad de estructuración bastante flexible y permiten especificar restricciones de datos. Los modelos más conocidos son el modelo entidad-relación y el orientado a objetos.

- **Basados en objetos**
 - [Modelo Entidad-Relación \(ER\)](#)
 - [Modelo Orientado a Objetos \(OO\)](#)

BASADOS EN REGISTROS

Los modelos lógicos basados en registros se utilizan para describir los datos en los modelos conceptual y físico. A diferencia de los modelos lógicos basados en objetos, se usan para especificar la estructura lógica global de la BD y para proporcionar una descripción a nivel más alto de la implementación.

Los modelos basados en registros se llaman así porque la BD está estructurada en registros de formato fijo de varios tipos. Cada tipo de registro define un número fijo de campos, o atributos, y cada campo normalmente es de longitud fija. La estructura más rica de estas BBDD a menudo lleva a registros de longitud variable en el nivel físico.

- **Basados en registros**
 - [Modelo jerárquico](#)
 - [Modelo de red](#)
 - [Modelo relacional](#)

MODELO ENTIDAD-RELACIÓN

CONCEPTO

El **modelo entidad-relación (ER)** es un modelo de datos conceptual de alto nivel. Se utiliza con frecuencia para el diseño conceptual de las aplicaciones de bases de datos, y muchas herramientas de diseño emplean estos conceptos.

USO DE MODELOS DE DATOS CONCEPTUALES DE ALTO NIVEL PARA EL DISEÑO DE BD

El proceso de diseño de una base de datos consiste en lo siguiente.

1. **Recopilación de requisitos y análisis:** se busca comprender y documentar los requisitos del usuario en cuanto a datos. El resultado de este paso es un conjunto por escrito de requisitos del usuario.

2. **Especificación de requisitos funcionales:** consiste en las operaciones (o transacciones) definidas por el usuario que se aplicarán a la base de datos, incluyendo las recuperaciones y las actualizaciones.
3. **Diseño conceptual:** se crea un modelo conceptual para la base de datos, mediante un modelo de datos conceptual de alto nivel.
Esquema conceptual es una descripción concisa de los requisitos de datos por parte de los usuarios e incluye descripciones detalladas de los tipos de entidades, relaciones y restricciones. Estos conceptos no incluyen detalles de implementación, son más fáciles de entender y se pueden utilizar para comunicar con usuarios no técnicos. También sirve para garantizar que se han cumplido todos los requisitos y que éstos no entran en conflicto.
4. Se pueden utilizar las operaciones básicas del modelo de datos para especificar las operaciones de usuario de alto nivel identificadas en el análisis funcional. Esto también sirve para confirmar que el esquema conceptual satisface todos los requisitos funcionales identificados.
5. **Diseño lógico (o asignación de datos):** se implementa la base de datos mediante un DBMS comercial. La mayoría de los DBMS comerciales actuales utilizan un modelo de implementación, de modo que el esquema conceptual se transforma de modelo de datos de alto nivel en modelo de datos de implementación.
6. **Diseño físico:** se especifican las estructuras de almacenamiento interno, los índices, las rutas de acceso y la organización de los archivos para la base de datos. En paralelo, se diseñan e implementan los programas de aplicación como transacciones de base de datos correspondientes a las especificaciones de transacción de alto nivel.

CONCEPTOS DEL MODELO ER

El modelo ER describe los datos como entidades, relaciones y atributos.

ENTIDAD

Es el objeto básico representado por el modelo ER. Es una “cosa” del mundo real con una existencia independiente. Una entidad puede ser un objeto con una existencia física (persona, auto, casa, empleado, etc.) o puede ser un objeto con una existencia conceptual (empresa, trabajo, curso, etc.). Cada entidad tiene **atributos**.

ATRIBUTOS

Son propiedades particulares que describen a una entidad. Los valores de los atributos que describen cada entidad se convierten en la parte principal de los datos almacenados en la base de datos.

ATRIBUTOS COMPLEJOS VS ATRIBUTOS SIMPLES (ATÓMICOS)

- **Atributos compuestos:** se pueden dividir en subpartes más pequeñas, que representan atributos más básicos con significados independientes (por ejemplo, dirección se puede dividir en calle, número, ciudad, provincia). El valor de un atributo compuesto es la concatenación de los valores de sus atributos simples.
- **Atributos simples (o atómicos):** son los atributos que no se pueden dividir.

ATRIBUTOS MONOVALOR VS ATRIBUTOS MULTIVALOR

- **Atributos monovalor:** los atributos tienen un solo valor para una entidad en particular.
- **Atributos multivalor:** los atributos tienen un conjunto de valores para una entidad en particular.

ATRIBUTOS ALMACENADOS VS ATRIBUTOS DERIVADOS

Ejemplo: si se guarda la fecha de nacimiento de una persona, la fecha es un **dato almacenado**. A partir de la fecha de nacimiento almacenada, podemos calcular la edad de tal persona. La edad sería un **dato derivado**. Lo mismo ocurre cuando tenemos un detalle de factura, los datos almacenados serían el producto, la cantidad, y el precio. El subtotal (precio x cantidad) es un dato derivado.

VALORES NULL (NULOS)

Es cuando un atributo de una entidad no tiene un valor asignado (no confundir con vacío o 0 –cero-).

ATRIBUTOS COMPLEJOS

Son atributos compuestos y multivalor anidados arbitrariamente.

TIPOS DE ENTIDADES Y CONJUNTOS DE ENTIDADES

- **Tipo de entidad:** define una *colección* o *conjunto* de entidades que tienen los mismos atributos.
- **Conjunto de entidades:** es la colección de todas las entidades de un tipo de entidad en particular.

Un tipo de entidad describe el **esquema** o la **intención** de un *conjunto de entidades* que comparten la misma estructura.

ATRIBUTOS CLAVE DE UN TIPO DE ENTIDAD

- **Atributo clave:** es un atributo de un tipo de entidad cuyos valores son distintos para cada entidad individual del conjunto de entidades. Es utilizado para identificar cada identidad. En ocasiones, una clave está formada por varios atributos juntos, lo que da a entender que la *combinación* de valores de atributo debe ser distinta para cada entidad. Una clave compuesta debe ser mínima; es decir, en el atributo compuesto se deben incluir todos los atributos componente para tener una propiedad de unicidad.

CONJUNTOS DE VALORES (DOMINIOS) DE ATRIBUTOS

Cada atributo simple de un tipo de entidad está asociado con un conjunto de valores (o dominio de valores), que especifica el conjunto de valores que se pueden asignar a ese atributo por cada entidad individual. Normalmente se especifican mediante los **tipos de datos** básicos disponibles en la mayoría de los lenguajes de programación.

RELACIONES

Relación: es cuando un atributo de un tipo de entidad se refiere a otro tipo de entidad.+

TIPO DE RELACIÓN

Un **tipo de relación** R entre n tipos de entidades E_1, E_2, \dots, E_n define un conjunto de asociaciones (o un **conjunto de relaciones**) entre las entidades de esos tipos de entidades.

GRADO DE UN TIPO DE RELACIÓN

El **grado** de un tipo de relación es el número de tipos de entidades participantes. Un tipo de relación de grado dos se denomina **binario**, y uno de tres **ternario**. Las relaciones pueden ser generalmente de cualquier grado, pero las más comunes son las binarias.

NOMBRES DE ROL Y RELACIONES RECURSIVAS

Cada tipo de entidad que participa en un tipo de relación juega un papel o rol particular en la relación. El **nombre de rol** hace referencia al papel que una entidad participante del tipo de entidad juega en cada instancia de relación, y ayuda a explicar el significado de la relación. En algunos casos el mismo tipo de entidad participa más de una vez en un tipo de relación con diferentes roles. En esos casos, el nombre de rol es esencial para distinguir el significado de cada participación. Dichos tipos de relaciones se denominan **relaciones recursivas**.

RESTRICCIONES EN LOS TIPOS DE RELACIONES

Los tipos de relaciones normalmente tienen ciertas restricciones que limitan las posibles combinaciones entre las entidades que pueden participar en el conjunto de relaciones correspondientes. Estas restricciones están determinadas por la situación del minimundo representado por las relaciones. Podemos distinguir dos tipos principales de restricciones de relación: *razón de cardinalidad* y *participación*.

- **Razones de cardinalidad para las relaciones binarias:** la **razón de cardinalidad** de una relación binaria especifica el número *máximo* de instancias de relación en las que una entidad puede participar. Las posibles razones de cardinalidad para los tipos de relación binaria son 1:1, 1:N, N:1, y M:N.
- **Restricciones de participación y dependencias de existencia:** la **restricción de participación**, especifica si la existencia de una entidad depende de si está relacionada con otra entidad a través de un tipo de relación. Esta restricción especifica el número *mínimo* de instancias de relación en las que puede participar cada entidad, y en ocasiones recibe el nombre de **restricción de cardinalidad mínima**. Hay dos tipos de restricciones de participación: total y parcial.
 - Restricción de participación total: cada entidad del conjunto total de entidades debe estar relacionada con otra entidad de otro tipo de entidad (**dependencia de existencia**).
 - Restricción de participación parcial: algo o parte del conjunto de entidades está relacionado con alguna otra entidad de otro tipo de entidad, pero no necesariamente con todas.

ATRIBUTOS DE LOS TIPO DE RELACIÓN

Los tipos de relación también pueden tener atributos, parecidos a los de los tipos de entidad.

Los atributos de los tipos de relación 1:1 o 1:N se pueden trasladar a uno de los tipos de entidad participantes. En el caso de un tipo de relación 1:N, un atributo de relación sólo se puede migrar al tipo de entidad que se encuentra en el lado N de la relación.

Para los tipos de relación M:N, algunos atributos pueden determinarse mediante la combinación de entidades participantes en una instancia de relación, no mediante una sola relación. Dichos atributos deben especificarse como atributos de relación.

TIPOS DE ENTIDADES DÉBILES

- **Tipos de entidades débiles:** son los tipos de entidad que no tienen atributos clave propios.
- **Tipos de entidades fuertes:** son los tipos de entidad regulares que tienen un atributo clave (propietarios).

El tipo de relación que relaciona un tipo de entidad débil con su propietario se llama **relación identificativa** del tipo de entidad débil.

BASES DE DATOS ORIENTADAS A OBJETOS

Las bases de datos orientadas a objetos (BDOO) se propusieron para satisfacer las necesidades de las aplicaciones complejas. La metodología de orientación a objetos ofrece la flexibilidad de manipular algunos de los requisitos sin la limitación impuesta por los tipos de datos y los lenguajes de consulta disponibles en los sistemas de bases de datos tradicionales. Una característica fundamental de las BDOO es la potencia que otorgan al diseñador especificar tanto la *estructura* de los objetos complejos como las *operaciones* que pueden aplicarse a esos objetos.

Otra razón para la creación de BDOO es el incremento del uso de lenguajes de programación orientados a objetos en el desarrollo de aplicaciones de software.

Las BDOO están diseñadas para que se integren directamente y sin problemas con las aplicaciones que están desarrolladas en dichos lenguajes.

Aunque se han creado muchos prototipos experimentales y sistemas de BDOO comerciales, no se ha extendido su uso debido a la popularidad de los sistemas relacionales.

Las BDOO han adoptado muchos conceptos que se desarrollan principalmente para los lenguajes de programación orientados a objetos.

OBJETO

Un **objeto** tiene dos componentes: un estado (valor) y un comportamiento (operaciones). Los objetos en un lenguaje de programación orientado a objetos (OOPL, *oriented object programming language*) sólo existen durante la ejecución del programa, por ello se denominan *objetos transitorios*. Una BDOO puede extender la existencia de los objetos guardándolos permanentemente (en almacenamiento secundario), de modo que los objetos *persisten* más allá de la terminación del programa y pueden recuperarse y compartirse con posterioridad con otros programas.

IDENTIFICADOR DE OBJETO

Un objetivo de las BDOO es mantener una correspondencia directa entre los objetos del mundo real y de la base de datos, para que los objetos no pierdan su integridad e identidad, puedan identificarse fácilmente y pueda trabajarse con ellos. Por tanto, las BDOO proporcionan un **identificador de objeto** (OID, *object identifier*) generado por el sistema para cada objeto. El valor de este OID no es visible para los usuarios y es inmutable.

CARACTERÍSTICAS

Una base de datos orientada a objetos es una base de datos que incorpora todos los conceptos importantes del paradigma de objetos:

- **Encapsulamiento:** propiedad que permite ocultar la información de un objeto al resto de los objetos, impidiendo así accesos incorrectos o conflictos. Para acceder a la información de un objeto se debe hacer a través de métodos.
- **Herencia:** propiedad a través de la cual los objetos heredan comportamiento dentro de una jerarquía de clases.
- **Polimorfismo:** propiedad mediante la cual una operación 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.

MODELO JERÁRQUICO Y DE RED

MODELO JERÁRQUICO

Éstas son bases de datos que, como su nombre indica, almacenan su información 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 información 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. Cada jerarquía representa una cantidad de registros relacionados. Las relaciones son punteros o enlaces. Permite relaciones uno a uno y uno a muchos. Dentro de sus restricciones tenemos: todo hijo tiene un solo padre, un padre puede tener uno o más hijos, no hay hijos sin padre. En esta estructura hay repetición de datos. Las bases de datos jerárquicas son especialmente útiles en el caso de aplicaciones que manejan un gran volumen de información 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. Las órdenes implican acciones físicas:

- **Get/Where:** localiza y copia registro a memoria. Se recorren ramas.
- **Insert:** se localiza a su registro padre y luego inserta.
- **Replace:** modifica uno o más campos de un registro.
- **Delete:** elimina el registro actual de la base de datos.

MODELO 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), es decir que permite relaciones uno a muchos y muchos a muchos. 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 la información en una base de datos de red ha significado que sea un modelo utilizado en su mayoría por programadores más que por usuarios finales. Al ser un modelo basado en registros los datos son representados como colección de registros. Las relaciones son punteros o enlaces. Tienen una estructura compleja para problemas simples. Una restricción muy importante es la de no insertar un hijo si no hay padre. Las órdenes implican acciones físicas:

- **Find:** localiza un registro por condición y ubica el puntero.
- **Get:** copia registro señalado por el puntero actual al buffer.
- **Store:** crea un registro, con valores de datos ubicados en memoria.
- **Modify:** implica encontrarlo, subirlo a memoria y modificarlo.
- **Erase:** elimina registro, con previa búsqueda.
- **Connect:** conecta un registro con un conjunto de registros.
- **Disconnect:** desconecta un registro de un conjunto.
- **Reconnect:** mueve un registro de un conjunto a otro.

MODELO RELACIONAL

El modelo utiliza el concepto de una *relación matemática* (algo parecido a una tabla de valores) como su bloque de construcción básico, y tiene su base teórica en la *teoría de conjuntos* y la *lógica de predicado de primer orden*.

CONCEPTOS

El modelo relacional representa la base de datos como una colección de *relaciones*. Cada una de estas relaciones se parece a una tabla de valores.

Cuando una relación está pensada como una tabla de valores, cada fila representa una colección de valores seleccionados.

Una fila recibe el nombre de **tupla**, una cabecera de columna es un **atributo**, y el nombre de la tabla una **relación**.

Relación: es un conjunto de tuplas.

DOMINIO

Un **dominio** es un conjunto de valores atómicos que puede asumir un atributo.

Atómico: quiere decir que cada valor de un dominio es indivisible en lo que al modelo relacional se refiere.

Una forma habitual de especificar un dominio es indicar un tipo de dato desde el que se dibujan los valores del mismo. También resulta útil darle un nombre que ayude a la interpretación de sus valores (ejemplo: Nombres, NumeroTelefonoFijo, EdadEmpleado, etc.). Para cada uno de ellos se especifica también un tipo de dato o formato (ejemplo: NumeroTelefono con formato *ddddddddd* –números decimales-).

Un dominio cuenta, por lo tanto, con un nombre, un tipo de dato y un formato.

CARACTERÍSTICAS DE LAS RELACIONES

- **Ordenación de tuplas en una relación:** una relación está definida como un conjunto de tuplas (tener en cuenta la teoría de conjuntos). Los elementos de un conjunto no guardan orden entre ellos, por tanto, las tuplas en una relación tampoco lo tienen (las tuplas no están ordenadas). Entonces, una relación no es sensible al ordenamiento de tuplas.
- **Ordenación de los valores dentro de una tupla y definición alternativa de una relación:** a nivel lógico, el orden de los atributos y sus valores no es tan importante mientras se mantenga la correspondencia entre ellos.
- **Valores:** cada valor en una tupla es un valor **atómico**, es decir, no es divisible en componentes dentro del esqueleto del modelo relacional básico. Por tanto, no están permitidos los atributos compuestos y multivalor.
- **NULLs (nulos):** un valor NULL significa que el valor de un atributo es desconocido, valor existente pero no disponible o atributo no aplicable a la tupla.
- **Interpretación (significado) de una relación:** algunas relaciones pueden representar hechos sobre entidades, mientras que otras pueden hacerlo sobre relaciones, es decir que, el modelo relacional representa hechos sobre entidades y relaciones uniformemente como relaciones. Esto puede comprometer a veces la comprensibilidad porque se tiene que adivinar si una relación representa un tipo de entidad o relación. Se debe tener en cuenta que no deben haber tuplas duplicadas, es decir, se representan hechos diferentes del mundo real.

RESTRICCIONES DEL MODELO RELACIONAL (CONSTRAINTS)

Existen muchas restricciones, o *constraints*, en los valores de un estado de base de datos. Pueden dividirse en tres categorías:

- **Restricciones implícitas o inherentes basadas en el modelo:** las características de las relaciones vistas anteriormente son las restricciones inherentes, como por ejemplo, que no haya tuplas duplicadas, que los valores sean atómicos, entre otros.
- **Restricciones explícitas o basadas en el esquema:** pueden expresarse directamente en los esquemas del modelo de datos, por lo general especificándolas en el DDL. Entre ellas se incluyen las restricciones de dominio, las de clave, las restricciones en valores NULL, las de integridad de entidad y las de integridad referencial. *Posteriormente se explica cada una de ellas.*
- **Restricciones semánticas, basadas en aplicación o reglas de negocio:** no pueden expresarse directamente en los esquemas del modelo de datos, por consiguiente deben ser expresadas e implementadas por los programas. Uso de triggers y aserciones para especificarlas.

RESTRICCIONES BASADAS EN EL ESQUEMA

1. Restricciones de dominio.
 2. Restricciones de clave.
 3. Restricciones en valores NULL.
 4. Restricciones de integridad de entidad,
 5. Restricciones de integridad referencial y *foreign keys*.
- **Restricciones de dominio:** especifican que dentro de cada tupla, el valor de un atributo A debe ser un valor atómico del dominio $dom(A)$.
 - **Restricciones de clave:** especifica una restricción de exclusividad por la que dos tuplas distintas en cualquier estado r de R puede tener el mismo valor para SK. Cada relación tiene, al menos,

una superclave predeterminada: el conjunto de todos sus atributos. Sin embargo, una superclave puede contar con atributos redundantes, por lo que un concepto más importante es el de clave, que no tiene redundancia. Una clave K de un esquema de relación R es una superclave de R con la propiedad adicional que eliminando cualquier atributo A de K deja un conjunto de atributos K' que ya no es una superclave de R. Por lo tanto una clave satisface dos restricciones:

- 1) Dos tuplas diferentes en cualquier estado de la relación no pueden tener valores idénticos para todos los atributos de la clave. **Propiedad de unicidad.**
 - 2) Es una superclave mínima, de la cual no podemos eliminar ningún atributo y seguiremos teniendo almacenada la restricción de exclusividad de la condición 1. **Propiedad de minimalidad.**
- **Restricción de valores NULL:** especifica si se permite o no los valores NULL en un atributo.
 - **Restricción de integridad de entidad:** declaran que el valor de ninguna clave principal puede ser NULL. Esto se debe a que dicha clave se emplea para identificar tuplas individuales en una relación. Si se permitiera este valor, significaría que no se podrían identificar ciertas tuplas.
 - **Restricciones de integridad referencial:** estas restricciones están especificadas entre dos relaciones y se utilizan para mantener la consistencia entre las tuplas de dos relaciones. Las restricciones de integridad referencial dicen que una tupla de una relación que hace referencia a otra relación debe hacer referencia a una tupla existente de esa relación. Al atributo que hace referencia a otra relación se lo denomina **foreign key**, y debe tener el mismo dominio que el atributo de la relación referenciada. Una **foreign key** puede hacer referencia a su propia relación.

La regla de integridad referencial, en resumen, determina que una base de datos no debe contener valores de referencia sin concordancia.

OPERACIONES DEL MODELO RELACIONAL

Existen tres tipos de operaciones básicas: inserción, borrado y modificación.

INSERT

Se utiliza para insertar una nueva tupla o tuplas en una relación. Proporciona una lista de los valores de atributo para una nueva tupla que será insertada en una relación. Esta operación puede violar cualquiera de las restricciones vistas anteriormente: las de dominio, si el valor dado a un atributo no aparece en el dominio correspondiente; las de clave, si el valor de dicha clave en la nueva tupla ya existe en otra tupla en la relación; las de integridad de entidad; si la clave principal de la nueva tupla es NULL; y las de integridad referencial, si el valor de cualquier foreign key en la nueva tupla se refiere a una tupla que no exista en la relación referenciada.

DELETE

Se encarga de borrar una o varias tuplas en una relación. Solo puede violar la integridad referencial en caso de que la tupla a eliminar esté referenciada por las foreign keys de otras tuplas de la base de datos. Para especificar un borrado, hay que especificar una condición en los atributos de la relación que selecciona la tupla (o tuplas) a eliminar.

Son varios los caminos que pueden tomarse si un borrado provoca una violación. El primero consiste en rechazar el borrado. El segundo pasa por intentar propagar el borrado eliminando las tuplas que hacen referencia a la que estamos intentando borrar. Una tercera posibilidad es modificar los valores del atributo referenciado que provocan la violación; a cada uno de ellos podría asignársele NULL, o modificarlo de forma que haga referencia a otra tupla válida. Tenga en cuenta que si el atributo referenciado que provoca la violación es parte de la clave principal, no puede ser establecido a NULL; de lo contrario, podría violar la integridad de entidad. Son posibles combinaciones de estas tres posibilidades.

En general, cuando se especifica una restricción de integridad referencial en el DDL, el DBMS permitirá al usuario especificar las opciones que se aplicarán en el caso de producirse una violación de dichas restricciones.

UPDATE

Se utiliza para cambiar los valores de uno o más atributos de una tupla (o tuplas) de una relación. Para seleccionar la información a modificar es necesario indicar una condición en los atributos de la relación.

La actualización de un atributo que no forma parte ni de una clave principal ni de una foreign key no suele plantear problemas; el DBMS sólo tiene que verificar que el nuevo valor tiene el tipo de dato y dominio correctos. La modificación de una clave principal es una operación similar al borrado de una tupla y la inserción de otra en su lugar, ya que usamos esta clave principal para identificar dichas tuplas. Por tanto, los problemas de Insert y Delete pueden aparecer. Si una foreign key se modifica, la base de datos debe asegurarse de que el nuevo valor hace referencia a una tupla existente en la relación referenciada (o es NULL). Existen opciones similares para tratar con las violaciones de integridad referencial provocadas por Update a las comentadas por la operación Delete. De hecho, cuando se especifica una restricción de este tipo en el DDL, el DBMS permitirá que el usuario elija de forma separada las acciones a tomar en cada caso.

TRANSACCIÓN

Este proceso implica tanto la lectura desde una base de datos como efectuar inserciones, borrados y actualizaciones en los valores de la misma.

ÁLGEBRA RELACIONAL

Un modelo de datos debe incluir un conjunto de operaciones para manipular la base de datos junto con los conceptos necesarios para la definición de su estructura y restricciones. El conjunto de operaciones básicas del modelo relacional es el **álgebra relacional**, el cual permite al usuario especificar las peticiones fundamentales de recuperación. El resultado de una recuperación es una nueva relación, la cual puede estar constituida por una o más relaciones (propiedad de cerradura). Por consiguiente, las operaciones de álgebra producen nuevas relaciones que pueden ser manipuladas más adelante usando operaciones de la misma álgebra. Una secuencia de operaciones de álgebra relacional conforma una expresión de álgebra relacional, cuyo resultado será también una nueva relación que representa el resultado de una consulta a la base de datos (o una petición de recuperación).

El álgebra relacional es muy importante debido a que proporciona un fundamento formal para las operaciones del modelo relacional y se utiliza como base para la implementación y optimización de consultas en los RDBMS (Sistemas de administración de bases de datos relacionales).

Mientras que el álgebra define un conjunto de operaciones del modelo relacional, los **cálculos relacionales** ofrecen una notación declarativa de alto nivel para especificar las consultas relacionales. Una expresión de cálculo relacional crea una nueva relación, la cual está especificada en términos de variables que engloban filas de las relaciones almacenadas en la base de datos (en cálculos de tupla) o columnas de las relaciones almacenadas (para los cálculos de dominio). En una expresión de cálculo, no existe un orden de operaciones para recuperar los resultados de la consulta: la expresión sólo especifica la información que el resultado debería contener.

Las operaciones del álgebra relacional pueden dividirse en dos grupos. Uno de ellos incluye el conjunto de operaciones de la teoría matemática de conjuntos, los cuales son aplicables porque cada relación está definida de modo que sea un conjunto de tuplas en el modelo relacional formal. Estas operaciones, llamadas **tradicionales**, incluyen **UNIÓN (UNION)**, **INTERSECCIÓN (INTERSECTION)**, **DIFERENCIA DE CONJUNTOS (SET DIFFERENCE)** y **PRODUCTO CARTESIANO (CARTESIAN PRODUCT)**. El otro grupo está constituido por las operaciones desarrolladas específicamente para las bases de datos relacionales,

llamadas **operaciones relacionales** especiales, como la **SELECCIÓN (SELECT)**, la **PROYECCIÓN (PROJECT)**, la **CONCATENACIÓN o COMBINACIÓN (JOIN)** y otras.

Operación unaria: se aplica a una sola relación.

Operación binaria: se aplican a dos conjuntos de tuplas.

OPERACIONES RELACIONALES UNARIAS

1. SELECT (SELECCIÓN)
2. PROJECT (PROYECCIÓN)
3. RENAME (RENOMBRAR)

SELECT (SELECCIÓN)

SELECT se emplea para seleccionar un subconjunto de las tuplas de una relación que satisfacen una condición de selección. Es como un filtro que mantiene sólo las tuplas que satisfacen una determinada condición. Puede visualizarse también como una partición horizontal de la relación en dos conjuntos de tuplas: las que satisfacen la condición son seleccionadas y las que no, descartadas.

La SELECCIÓN está designada como:

$$\sigma_{\langle \text{condición de selección} \rangle}(R)$$

donde el símbolo σ (sigma) se utiliza para especificar el operador de SELECCIÓN, mientras que la condición de selección es una expresión lógica (o booleana) especificada sobre los atributos de la relación R. R, es generalmente, una expresión de álgebra relacional cuyo resultado es una relación. El resultado de SELECCIÓN tiene los mismos atributos que R.

La expresión lógica especificada en $\langle \text{condición de selección} \rangle$ está constituida por un número de **cláusulas** de la forma:

$\langle \text{nombre de atributo} \rangle \langle \text{operador de comparación} \rangle \langle \text{valor constante} \rangle$, o bien:

$\langle \text{nombre de atributo} \rangle \langle \text{operador de comparación} \rangle \langle \text{nombre de atributo} \rangle$

donde $\langle \text{nombre de atributo} \rangle$ es el nombre de un atributo de R, $\langle \text{operador de comparación} \rangle$ suele ser uno de los operadores $\{=, <, \leq, >, \geq, \neq\}$ y $\langle \text{valor constante} \rangle$ es un valor del dominio del atributo.

Las cláusulas pueden estar conectadas arbitrariamente por operadores lógicos **and, or y not** para formar una condición de selección general.

Los operadores de comparación del conjunto $\{=, <, \leq, >, \geq, \neq\}$ se aplican a los atributos cuyos dominios son valores ordenados, como los de tipo numérico o de fecha. Los de cadenas de caracteres también se consideran del mismo tipo debido a la secuencia en los códigos. Si el dominio de un atributo es un conjunto de valores desordenados, solo pueden usarse los operadores $\{=, \neq\}$.

SELECCIÓN es **unaria**, es decir, se aplica a una sola relación. Además, esta operación se aplica a cada tupla individualmente; por consiguiente, las condiciones de selección no pueden implicar a más de una tupla. El grado de la relación resultante de una operación SELECCIÓN (su número de atributos) es el mismo que el de R. El número de tuplas en la relación resultante es siempre menor que o igual que el número de tuplas en R. Esto es, $\sigma_C(R) \leq |R|$ para cualquier condición C. La fracción de tuplas seleccionadas por una condición de relación está referida como la selectividad de la condición.

La operación SELECCIÓN es conmutativa.

Ejemplos:

$$\sigma_{\text{Levajo} > 14000}(\text{Estudiantes})$$

$\sigma_{(Dno=4 \text{ AND } Sueldo>25000) \text{ OR } (Dno=5 \text{ AND } Sueldo>30000)}$ (EMPLEADO)

PROJECT (PROYECCIÓN)

Mientras que la SELECCIÓN elige algunas de las filas de la tabla y descarta otras, la PROYECCIÓN selecciona ciertas columnas de la tabla y descarta otras. Si sólo estamos interesados en algunos atributos de una relación, usamos la operación PROYECCIÓN para planear la relación sólo sobre esos atributos. El resultado de esta operación puede visualizarse como una partición vertical de la relación en otras dos: una contiene las columnas (atributos) necesarias y otra las descartadas. La forma general de la operación PROYECCIÓN es:

$$\pi_{\langle \text{lista de atributos} \rangle}(R)$$

donde π (π) es el símbolo usado para representar la operación PROYECCIÓN, mientras que $\langle \text{lista de atributos} \rangle$ contiene la lista de campos de la relación R que queremos. R es, en general, una expresión del álgebra relacional cuyo resultado es una relación. El resultado de la operación PROYECCIÓN sólo tiene los atributos especificados en $\langle \text{lista de atributos} \rangle$ en el mismo orden a como aparecen en la lista. Por tanto, su grado es igual al número de atributos contenidos en $\langle \text{lista de atributos} \rangle$.

Si la lista de atributos sólo incluye atributos no clave de R, es posible que se dupliquen tuplas. La operación PROYECCIÓN elimina cualquier tupla duplicada, por lo que el resultado de la misma es un conjunto de tuplas y, por consiguiente, una relación válida. Esto se conoce como eliminación de duplicados. La eliminación de duplicados lleva implícito un proceso de ordenación para detectar esos registros repetidos y, por tanto, añadir más capacidad de procesamiento. Si no se llevara a cabo este proceso, el resultado sería un multiconjunto o bolsa de tuplas en lugar de un conjunto.

El número de tuplas de una relación resultante de una operación PROYECCIÓN es siempre menor o igual que el de las contenidas en R. Si la lista de proyección es una superclave de R (esto es, incluye alguna clave de R) la relación resultante tiene el mismo número de tuplas que R.

La conmutatividad no se almacena en una PROYECCIÓN.

Ejemplo:

$$\pi_{\text{Apellido, Nombres}}(\text{NoDocentes})$$

SECUENCIAS DE OPERACIONES Y LA OPERACIÓN RENAME (RENOMBRAR)

En general, podemos querer aplicar varias operaciones de álgebra relacional una tras otra. De cualquier forma, podemos escribir las operaciones como una única expresión de álgebra relacional anidando dichas operaciones, o aplicar una sola expresión una única vez y crear relaciones intermedias. En el último caso, puede que queramos asignar nombre a dichas relaciones intermedias.

Podemos usar también esta técnica para renombrar los atributos en las relaciones intermedias y resultantes. Para renombrar los atributos de una relación, simplemente enumeramos los nuevos nombres de atributos dentro de los paréntesis.

Si no se realiza un renombrado, los nombres de atributo de la relación resultante de una SELECCIÓN son los mismos que los de la relación original y aparecen en el mismo orden. Para el caso de una operación PROYECCIÓN, la relación resultante tiene los mismos nombres de atributo que los indicados en la lista de proyección y están en el mismo orden en que aparecen en dicha lista.

Podemos definir una operación RENOMBRAR como un operador unario. Una operación RENOMBRAR aplicada a una relación R de grado n aparece denotada de cualquiera de estas tres formas:

- $\rho S(B_1, B_2, \dots, B_n)(R)$
- $\rho S(R)$
- $\rho(B_1, B_2, \dots, B_n)(R)$

donde el símbolo ρ (rho) se utiliza para especificar el operador RENOMBRAR, S es el nombre de la nueva relación y B1, B2, ..., Bn son los de los nuevos atributos. La primera expresión renombra tanto la relación como sus atributos, la segunda sólo lo hace con la relación y la tercera sólo con los atributos. Si los atributos de R son (A1, A2, ..., An) por este orden, entonces cada Ai es renombrado como Bi.

Ejemplo:

$TEMP \leftarrow \sigma_{Dno=5}(EMPLEADO)$

$R(NuevoNombre, NuevosApellido, NuevoSueldo) \leftarrow \pi_{Nombre, Apellido1, Sueldo}(TEMP)$

OPERACIONES DE ÁLGEBRA RELACIONAL DE LA TEORÍA DE CONJUNTOS

UNION (UNIÓN), INTERSECTION (INTERSECCIÓN), MINUS (MENOS)

Para combinar los elementos de dos conjuntos se utilizan varias operaciones de la teoría de conjuntos, como la UNIÓN, la INTERSECCIÓN y la DIFERENCIA DE CONJUNTOS (llamada también a veces MENOS, o MINUS). Todas ellas son operaciones binarias, es decir, se aplican a dos conjuntos (de tuplas). Cuando se refieren a las bases de datos relacionales, las relaciones sobre las que se aplican estas tres operaciones deben tener el mismo tipo de tuplas; esta condición recibe el nombre de compatibilidad respecto a la unión. Esto significa que ambas relaciones tienen el mismo número de atributos y que cada par correspondiente cuenta con el mismo dominio (cabeceras idénticas e igual dominio).

Dos relaciones $R(A1, A2, \dots, An)$ y $S(B1, B2, \dots, Bn)$ se dice que son de unión compatible si tienen el mismo grado n y si $\text{dom}(Ai) = \text{dom}(Bi)$ para $1 \leq i \leq n$. Podemos definir las tres operaciones UNIÓN, INTERSECCIÓN y DIFERENCIA DE CONJUNTO en dos relaciones de unión compatible R y S del siguiente modo:

- **UNIÓN:** el resultado de esta operación, especificada como $R \cup S$, es una relación que incluye todas las tuplas que están tanto en R como en S o en ambas, R y S. Las tuplas duplicadas se eliminan.
- **INTERSECCIÓN:** el resultado de esta operación, especificada como $R \cap S$, es una relación que incluye todas las tuplas que están en R y S.
- **DIFERENCIA DE CONJUNTO (o MENOS):** el resultado de esta operación, especificada como $R - S$, es una relación que incluye todas las tuplas que están en R pero no en S.

Tanto la UNIÓN como la INTERSECCIÓN son operaciones conmutativas: $R \cup S = S \cup R$ y $R \cap S = S \cap R$.

La UNIÓN y la INTERSECCIÓN pueden tratarse como operaciones n-ary aplicables a cualquier número de relaciones porque son estructuras asociativas: $R \cup (S \cup T) = (R \cup S) \cup T$ y $(R \cap S) \cap T = R \cap (S \cap T)$.

La operación MENOS no es conmutativa; por tanto, en general: $R - S \neq S - R$.

La INTERSECCIÓN puede expresarse en término de UNIÓN y DIFERENCIA DE CONJUNTOS del siguiente modo: $R \cap S = R \cup S - (R - S) - (S - R)$.

Ejemplos:

$Estudiantes \cup Docentes$ $Docentes \cap NoDocentes$
 $Docentes - NoDocentes$

EL PRODUCTO CARTESIANO (PRODUCTO CRUZADO)

EL PRODUCTO CARTESIANO (CARTESIAN PRODUCT) es conocido también como PRODUCTO CRUZADO (CROSS PRODUCT) o CONCATENACIÓN CRUZADA (CROSS JOIN) y se identifica por "X". Se trata también de una operación de conjuntos binarios, aunque no es necesario que las relaciones en las que se aplica sean una unión compatible. En su forma binaria produce un nuevo elemento combinando cada miembro (tupla) de una relación (conjunto) con los de la otra. En general, el resultado de $R(A1, A2, \dots,$

$A_n) \times S(B_1, B_2, \dots, B_n)$ es una relación Q con un grado de $n + m$ atributos $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_n)$, en este orden. La relación resultante Q tiene una tupla por cada combinación de éstas (una para R y otra para S). Por tanto, si R tiene nR tuplas (indicado como $|R|=nR$), y S cuenta con nS tuplas, $R \times S$ tendrá $nR \times nS$ tuplas.

La operación PRODUCTO CARTESIANO n -ary es una extensión del concepto indicado más arriba que produce nuevas tuplas concatenando todas las posibles combinaciones de tuplas desde n relaciones subyacentes. La operación aplicada es, por sí misma, absurda. Es útil cuando va seguida por una selección que correlacione los valores de los atributos procedentes de las relaciones componentes.

El PRODUCTO CARTESIANO crea tuplas con los atributos combinados de ambas relaciones. Sólo podemos hacer una SELECCIÓN de tuplas de las dos relaciones especificando una condición de selección apropiada. Ya que esta secuencia de PRODUCTO CARTESIANO seguido de una SELECCIÓN se emplea con mucha frecuencia para identificar y elegir tuplas de dos relaciones, existe una operación especial llamada CONCATENACIÓN que permite especificar esta secuencia como operación única.

Ejemplo:

Materias \times Provincias

OPERACIONES RELACIONALES BINARIAS

1. JOIN (CONCATENACIÓN)
2. DIVISION (DIVISIÓN)

CONCATENACIÓN (JOIN)

CONCATENACIÓN, especificada mediante \bowtie , se emplea para combinar tuplas relacionadas de dos relaciones en una sola. Esta operación es muy importante para cualquier base de datos relacional que cuente con más de una relación, ya que nos permite procesar relaciones entre relaciones.

La CONCATENACIÓN puede ser enunciada en términos de un PRODUCTO CARTESIANO seguido de una SELECCIÓN. Sin embargo, CONCATENACIÓN es muy importante porque se utiliza con mucha frecuencia cuando se definen consultas a la base de datos.

La forma general de una CONCATENACIÓN en dos relaciones $R(A_1, A_2, \dots, A_n)$ y $S(B_1, B_2, \dots, B_m)$ es:


$$R \bowtie \langle \text{condición de conexión} \rangle S$$

El resultado de la CONCATENACIÓN es una relación Q de $n+m$ atributos $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$ por este orden; Q tiene una tupla por cada combinación de éstas (una para R y otra para S) siempre que dicha combinación satisfaga la condición de conexión. Ésta es la principal diferencia existente entre el PRODUCTO CARTESIANO y la CONCATENACIÓN. En la CONCATENACIÓN sólo aparecen en el resultado las combinaciones de tuplas que satisfacen la condición de conexión, mientras que en el PRODUCTO CARTESIANO se incluyen todas las combinaciones de tuplas. La condición de conexión está especificada sobre los atributos de las dos relaciones R y S y es evaluada para cada combinación de tuplas, incluyéndose en la relación Q resultante en forma de única tupla combinada sólo aquellas cuya condición de conexión se evalúe como VERDADERO. Una condición general de conexión tiene la forma:

$$\langle \text{condición} \rangle \text{ AND } \langle \text{condición} \rangle \text{ AND } \dots \text{ AND } \langle \text{condición} \rangle$$

donde cada condición es de la forma $A_i \theta B_j$, A_i es un atributo de R , B_j lo es de S , A_i y B_j tienen el mismo dominio y θ (theta) es uno de los operadores de comparación $\{=, <, \leq, >, \geq, \neq\}$. Una CONCATENACIÓN con una condición de conexión de este tipo recibe el nombre de ASOCIACIÓN (THETA JOIN). Las tuplas cuyos atributos de conexión son NULL, o aquellas cuya condición de conexión es FALSA, no aparecen en el resultado. En este caso, CONCATENACIÓN no preserva necesariamente toda la información de las relaciones participantes.

Ejemplo:

Localidades 

Provincias

$Id\ Pr\ ovincia = Id\ Pr\ ovincia$

VARIACIÓN: CONCATENACIÓN: EQUIJOIN

El uso más habitual de CONCATENACIÓN supone el uso de condiciones de conexión sólo con comparaciones de igualdad, en cuyo caso recibe el nombre de EQUIJOIN. En el resultado de EQUIJOIN siempre tenemos uno o más pares de atributos que cuentan con valores idénticos en cada tupla. Ya que uno de estos valores idénticos es innecesarios, se creó una nueva operación llamada CONCATENACIÓN NATURAL.

VARIACIÓN: CONCATENACIÓN NATURAL (NATURAL JOIN)

Se creó esta nueva operación llamada CONCATENACIÓN NATURAL (identificada por *) para deshacerse del segundo atributo superfluo en una condición EQUIJOIN. La definición estándar de esta operación precisa que los dos atributos de conexión tengan el mismo nombre en ambas relaciones. Si éste no es el caso, se aplica en primer lugar una operación de renombrado.

Si los atributos en los que se aplicará la concatenación natural ya tienen el mismo nombre en ambas relaciones, el renombrado es innecesario. El resultado es un único atributo llamado atributo de conexión.

La CONCATENACIÓN se emplea para combinar datos procedentes de múltiples relaciones, de forma que la información pueda presentarse en una única tabla. Estas operaciones se conocen también como concatenaciones internas (*inner joins*) para distinguirlas de una variación llamada concatenaciones externas (*outer joins*). Una concatenación interna es un tipo de operación de correspondencia y asociación definida formalmente como una combinación de un PRODUCTO CARTEASIANO y una SELECCIÓN. Una concatenación externa es otra versión más permisiva de la otra. La CONCATENACIÓN NATURAL o la EQUIJOIN pueden establecerse también entre múltiples tablas, lo que lleva a una concatenación de n-vías.

DIVISION (DIVISIÓN)

La DIVISIÓN, especificada mediante \div , es útil para cierto tipo de consultas que a veces se realizan en aplicaciones de bases de datos.

En general, la DIVISIÓN se aplica a dos relaciones $R(Z) \div S(X)$, donde $X \subseteq Z$. Permite $Y=Z-X$ (y por tanto, $Z=X \cup Y$); es decir, consiente que Y sea el conjunto de atributos de R que no lo son de S . El resultado de una DIVISIÓN es una relación $T(Y)$ que incluye una tupla t si las tuplas t_R aparecen en R con $t_R[Y]=t$, y con $t_R[X]=t_S$ en S . Esto significa que, para que una tupla t aparezca en el resultado T de la DIVISIÓN, los valores de aquella deben aparecer en R en combinación con cada tupla en S . En la formulación de la operación DIVISIÓN, las tuplas de la relación denominador restringen la relación numerador seleccionando aquellas tuplas del resultado que sean iguales a todos los valores presentes en el denominador. No es necesario saber qué valores son los que están presentes.

Atributos del divisor incluidos en la cabecera del dividendo.

La DIVISIÓN puede expresarse como una secuencia de operaciones Π , X , y – del siguiente modo:

$$\begin{aligned} T_1 &\leftarrow \Pi_Y(R) \\ T_2 &\leftarrow \Pi_Y((S \times T_1) - R) \\ T &\leftarrow T_1 - T_2 \end{aligned}$$

Ejemplo:

La operación DIVISIÓN. (a) Dividiendo DNI_PNOS entre PEREZ_PNOS. (b) $T \leftarrow R \div S$.

(a)		(b)	
DNI_PNOS		PEREZ_PNOS	S
DniEmpleado	NumProy	NumProy	A
123456789	1	1	a1
123456789	2	2	a2
666884444	3		a3
453453453	1		
453453453	2		
333445555	2		
333445555	3		
333445555	10		
333445555	20		
999887777	30		
999887777	10		
987987987	10		
987987987	30		
987654321	30		
987654321	20		
888665555	20		

PEREZ_PNOS		R		S	
DNI		A	B	A	B
123456789		a1	b1	a1	
453453453		a2	b1	a2	
		a3	b1	a3	
		a4	b1		
		a1	b2		
		a3	b2T		
		a2	b3		
		a3	b3		
		a4	b3		
		a1	b4		

OPERACIONES RELACIONALES ADICIONALES

PROYECCIÓN GENERALIZADA

Es una operación que amplía las posibilidades de la proyección original permitiendo la inclusión de funciones de atributos en la lista de proyección. La forma generalizada puede expresarse del siguiente modo:

$$\pi F_1, F_2, \dots, F_n (R)$$

donde F_1, F_2, \dots, F_n son funciones sobre los atributos de la relación R y puede involucrar constantes. Esta operación está ideada como una ayuda a la hora de desarrollar informes en los que los valores calculados deben generarse en columnas.

Ejemplo:

INFORME. $\leftarrow \rho_{(Dni, SalarioNeto, Gratificaciones, Impuestos)}$
 $(\pi_{Dni, Sueldo - Deducción, 2000 * Antigüedad, 0.25 * Sueldo} (EMPLEADO)).$

FUNCIONES DE AGREGACIÓN Y AGRUPAMIENTO

Las funciones matemáticas de **agregación** se utilizan en consultas estadísticas sencillas que resumen información procedente de las tuplas de la base de datos. Las más comunes son SUMA (SUM), MEDIA (AVERAGE), MÁXIMO (MAXIMUM) Y MÍNIMO (MINIMUM). La función CONTAR (COUNT) se emplea para contar tuplas o valores.

Otro tipo común de petición supone realizar la **agrupación de las tuplas de una relación por el valor de uno de sus atributos** y la aplicación posterior de una función de agregación independiente a cada grupo.

Se puede definir la operación FUNCIÓN AGREGADA (AGGREGATE FUNCTION) usando el símbolo F (f gótica), para especificar este tipo de peticiones:

$$\langle \text{atributos de agrupamiento} \rangle F \langle \text{lista de funciones} \rangle (R)$$

donde $\langle \text{atributos de agrupamiento} \rangle$ es una lista de los atributos de la relación especificados en R , y $\langle \text{lista de funciones} \rangle$ es una lista de parejas ($\langle \text{función} \rangle \langle \text{atributo} \rangle$). En cada una de estas parejas, $\langle \text{función} \rangle$ puede ser SUMA, MEDIA, MÁXIMO, MÍNIMO o CONTAR, mientras que $\langle \text{atributo} \rangle$ es un atributo de la relación

especificada por R. La relación resultante cuenta con los atributos de agrupamiento además de otro por cada elemento de la lista de funciones.

Por lo general, se especifica una lista de nombres de atributos (entre los paréntesis de la operación resultante de R, con el fin de colocar un nombre a los resultados de las funciones. En caso de no aplicarse este cambio de nombre, los atributos correspondientes a la lista de funciones serán el resultado de la concatenación del nombre de la función con el del atributo en la forma <función>_<atributo>.

Si no se especifican atributos de agrupamiento, las funciones se aplican a todas las tuplas de la relación, por lo que obtendremos como resultado una única tupla.

Las duplicaciones no se eliminan cuando se aplica una función de agregación; de esta forma, la interpretación normal de funciones como SUMA y MEDIA es calculada. El resultado de aplicar una función de agregación es una relación, y no un número escalar, aun cuando sólo tenga un valor. Esto hace del álgebra relacional un sistema cerrado.

Ejemplos:

$$\tau_{COUNT(Legajo), AVG(Nota)}(Exámenes) \quad Legajo \tau_{AVG(Nota)}(Exámenes)$$

CONCATENACIÓN EXTERNA (OUTER JOIN)

Las operaciones CONCATENACIÓN descritas anteriormente emparejan tuplas que satisfacen la condición de conexión. Las tuplas con valores NULL en los atributos de conexión se eliminan. Esto equivale a una pérdida de información en el caso de que CONCATENACIÓN se utilice para generar un informe basado en todos los datos de las relaciones.

Existe un conjunto de operaciones, llamadas **concatenaciones externas**, que pueden usarse cuando queremos mantener en el resultado todas las tuplas de R, o de S, o de ambas, independientemente de si tienen correspondencias o no en la otra relación. Esto permite ejecutar consultas en las que tuplas procedentes de dos tablas se combinan emparejando las filas correspondientes, pero sin perder aquéllas que no tienen ese “compañero”. Las operaciones de concatenación que sólo mantenían las tuplas coincidentes, reciben el nombre de **concatenaciones internas**.

La operación **CONCATENACIÓN EXTERNA IZQUIERDA (LEFT OUTER JOIN)** mantiene cada tupla de la primera relación, o relación izquierda, R en $R \bowtie S$; si no se encuentra ninguna tupla en S, sus atributos en la concatenación resultante se rellenan con valores NULL.

Una operación parecida, la **CONCATENACIÓN EXTERNA DERECHA (RIGHT OUTER JOIN)**, especificada por $R \bowtie S$, es una operación similar que mantiene cada tupla de la segunda relación o relación derecha, S en el resultado de $R \bowtie S$. Una tercera operación, **CONCATENACIÓN EXTERNA COMPLETA (FULL OUTER JOIN)**, expresada por $R \bowtie S$, mantiene todas las tuplas de ambas relaciones (las de la derecha y las de la izquierda) cuando no existen tuplas coincidentes, rellenándolas con valores NULL cuando sea necesario. A veces, se hace necesario generar informes de datos procedentes de varias tablas independientemente de que existan o no valores coincidentes.

RELACIONES TEMPORALES

Permiten subdividir operaciones complejas y anidadas. Ejemplos:

$$\begin{aligned} temp1 &\leftarrow (\tau_{Legajo \tau_{AVG(Nota)}}(Exámenes)) \\ temp2 &\leftarrow (temp1 \bowtie_{Legajo=Legajo}(Estudiantes)) \\ temp3 &\leftarrow (\pi_{Legajo, Apellido, Nombres, AVG_Nota}(temp2)) \end{aligned}$$

CÁLCULOS RELACIONALES DE TUPLA

En el **cálculo relacional** escribimos una expresión **declarativa** para especificar una consulta de recuperación; por tanto, no hay una descripción del modo de evaluar una consulta. Una expresión de cálculo especifica qué se quiere recuperar en lugar de como hacerlo, por lo que se le considera un **lenguaje no procedural**. Esto le hace diferente del álgebra relacional, en donde debemos escribir una secuencia de operaciones para especificar la consulta de recuperación, razón por la que se le puede considerar como una forma **procedural** de declarar la misma. Es posible anidar operaciones de álgebra para formar una única expresión; sin embargo, siempre es necesario indicar explícitamente un cierto orden en una expresión de álgebra relacional. Este orden influye también en la estrategia de evaluación de la consulta. Una expresión de cálculo puede escribirse de diferentes formas, aunque esto no influye en el modo en que dicha consulta será evaluada.

Cualquier recuperación que pueda especificarse mediante el álgebra relacional básica puede hacerse del mismo modo a través de cálculos relacionales, y viceversa; en otras palabras, la **potencia expresiva** de ambos lenguajes es idéntica. Esto conduce a la definición del concepto de un lenguaje relacionalmente completo. Se considera que un lenguaje de consulta relacional L es relacionalmente completo si podemos expresar en él cualquier consulta que pueda realizarse mediante un cálculo relacional.

NORMALIZACIÓN

Cada esquema de relación consta de un número de atributos, mientras que un esquema de base de datos relacional está compuesto por un número de esquemas de relación.

Hay dos niveles a los que podemos explicar la bondad de los esquemas de relación. El primero es el **nivel lógico (o conceptual)**: cómo los usuarios interpretan los esquemas de relación y el significado de sus atributos. Disponer de un buen esquema de relación a este nivel permite a los usuarios comprender con claridad el significado de los datos en las relaciones y, por consiguiente, formular consultas correctamente. El segundo nivel es el de **nivel de implementación (o almacenamiento)**: de qué modo se almacenan y actualizan las tuplas en una relación base.

CONCEPTO

El procedimiento de **normalización** consiste en la aplicación de una serie de comprobaciones de las relaciones para cumplir con unos requisitos cada vez más restrictivos y descomponer las relaciones cuando sea necesario. *Hacer comprobaciones para que se cumplen las formas normales.*

Las bases de datos relacionales se normalizan para:

- Evitar la redundancia de los datos.
- Disminuir problemas de actualización de los datos en las tablas.
- Proteger la integridad de los datos.

DIRECTRICES DE DISEÑO INFORMALES PARA LOS ESQUEMAS DE RELACIÓN

Hay cuatro medidas informales de calidad para el diseño de un esquema de relación:

- La semántica de los atributos.
- La reducción de información redundante en las tuplas.
- La reducción de valores NULL en las tuplas.
- Prohibición de la posibilidad de generar tuplas falsas.

DIRECTRIZ 1

Diseñar un esquema de relación para que sea fácil explicar su significado. No combinar atributos de varios tipos de entidad y de relación en una única relación.

DIRECTRIZ 2

Diseñar los esquemas de relación base de forma que no se presenten anomalías de inserción, borrado o actualización en las relaciones.

DIRECTRIZ 3

Hasta donde sea posible, evitar usar en una relación base atributos cuyos sean NULL frecuentemente.

DIRECTRIZ 4

Diseñar los esquemas de relación de forma que puedan concatenarse con condiciones de igualdad en los atributos que son parejas de clave principal y *foreign key* de forma que se garantice que no se van a generar tuplas falsas. Evitar relaciones que contienen atributos coincidentes que no son combinaciones de *foreign key* y clave principal porque la concatenación de estos atributos puede producir tuplas falsas.

PROBLEMAS QUE CUBREN LAS DIRECTRICES

- Anomalías que causan trabajo redundante durante la inserción y modificación de una relación, y que puedan causar pérdidas accidentales de información durante el borrado de la misma.
- Desaprovechamiento del espacio de almacenamiento debido a valores NULL y la dificultad de llevar a cabo operaciones de selección, agregación y concatenación debido a estos valores.
- Generación de datos incorrectos y falsos durante las concatenaciones en relaciones base incorrectamente relacionadas.

DEPENDENCIAS FUNCIONALES

Dependencia funcional: es una restricción que se establece entre dos conjuntos de atributos de la base de datos.

Supongamos que nuestro esquema de base de datos relacional tiene n atributos A_1, A_2, \dots, A_n ; pensemos que la base de datos completa está descrita por un único esquema de relación universal $R = \{A_1, A_2, \dots, A_n\}$.

Una dependencia funcional, denotada por $X \rightarrow Y$, entre dos conjuntos de atributos X e Y que son subconjuntos de R , especifica una restricción en las posibles tuplas que pueden formar un estado de relación r de R . La restricción dice que dos tuplas t_1 y t_2 en r que cumplen que $t_1[X] = t_2[X]$, deben cumplir también que $t_1[Y] = t_2[Y]$.

Esto significa que los valores del componente Y de una tupla de r dependen de, o están determinados por, los valores del componente X ; alternatively, los valores del componente X de una tupla únicamente (o funcionalmente) determinan los valores del componente Y . Decimos también que existe una dependencia funcional de X hacia Y , o que Y es funcionalmente dependiente de X . La abreviatura de dependencia funcional es DF, o FD. El conjunto de atributos X recibe el nombre de lado izquierdo de la DF, mientras que Y es el lado derecho.

Por tanto, X determina funcionalmente Y si para toda instancia de r del esquema de relación R , no es posible que r tenga dos tuplas que coincidan en los atributos de X y no lo hagan en los atributos de Y .

- Si una restricción de R indica que no puede haber más de una tupla con un valor X concreto en cualquier instancia de relación $r(R)$, es decir, que X es una clave candidata de R , se cumple que $X \rightarrow Y$ para cualquier subconjunto de atributos de Y de R [ya que la restricción de clave implica que dos tuplas en cualquier estado legal $r(R)$ no tendrán el mismo valor de X].
- Si $X \rightarrow Y$ en R , esto no supone que $Y \rightarrow X$ en R .

Una dependencia funcional es una propiedad de la **semántica** o **significado de los atributos**. Los diseñadores de la base de datos utilizarán su comprensión de la semántica de los atributos de R (estos es, como se relacionan unos con otros) para especificar las dependencias funcionales que deben mantenerse en todos los estados de relación (extensiones) r de R . Siempre que la semántica de dos

conjuntos de atributos de R indique que debe mantenerse una dependencia funcional, la especificamos como una restricción. Las extensiones de relación $r(R)$ que satisfacen la restricción de dependencia funcional reciben el nombre de **estados de relación legales** (o **extensiones legales**) de R. Por tanto, el uso fundamental de las dependencias funcionales es describir más en profundidad un esquema de relación R especificando restricciones de sus atributos que siempre deben cumplirse. También es posible que ciertas dependencias funcionales puedan dejar de existir en el mundo real si cambia la relación.

Una dependencia funcional es una propiedad del esquema de relación R, y no un estado de relación legal particular r de R. Por consiguiente, una DF no puede ser inferida automáticamente a partir de una extensión de relación r , sino que alguien que conozca la semántica de los atributos de R debe definirla explícitamente.

Decimos que F es el conjunto de dependencias funcionales especificadas en un esquema de relación R. Habitualmente, el diseñador del esquema especifica las dependencias funcionales que son semánticamente obvias; sin embargo, es habitual que otras muchas dependencias funcionales se encuentren en todas las instancias de relación legales entre los conjuntos de atributos que pueden derivarse y satisfacen las dependencias de F. Esas otras dependencias pueden inferirse o deducirse de las DF de F. En la vida real, es imposible especificar todas las dependencias funcionales posibles para una situación concreta. Por tanto, formalmente es útil definir un concepto llamado **clausura** que incluye todas las posibles dependencias que pueden inferirse de un conjunto F dado.

Formalmente, el conjunto de todas las dependencias que incluyen F, junto con las dependencias que pueden inferirse de F, reciben el nombre de clausuras de F; y está designada por F^+ .

Dependencia funcional: $X \rightarrow Y$ “Y depende funcionalmente de X”, “Y está determinado por X” o “X determina funcionalmente a Y”.

- Por cada X hay un solo Y.
- No se da lo inverso.

Ejemplo:

DNI \rightarrow NombreCliente

FORMAS NORMALES

NORMALIZACIÓN DE RELACIONES

El proceso de normalización, tal y como fue propuesto en un principio por Codd, hace pasar un esquema de relación por una serie de comprobaciones para certificar que satisface una determinada **forma normal**.

El proceso, que sigue un método descendente evaluando cada relación contra el criterio de las formas normales y descomponiendo las relaciones según sea necesario, puede considerarse como un diseño relacional por análisis. Inicialmente, Codd propuso tres formas normales: la primera, la segunda y la tercera. Una definición más estricta de la 3FN, llamada BCNF (Forma normal Boyce-Codd) fue propuesta posteriormente por Boyce y Codd. Todas estas formas normales están basadas en una única herramienta analítica: las dependencias funcionales entre los atributos de una relación. Más adelante se propusieron una cuarta (4FN) y una quinta (5FN) formas normales basadas en los conceptos de dependencias multivalor y dependencias de concatenación, respectivamente.

La **normalización de datos** puede considerarse como un proceso de análisis de un esquema de relación, basado en sus DF y sus claves principales, para obtener las propiedades deseables de (1) minimizar la redundancia y (2) minimizar las anomalías de inserción, borrado y actualización. Los esquemas de relación no satisfactorios que no cumplen ciertas condiciones (las pruebas de forma normal) se descomponen en esquemas de relación más pequeños que cumplen esas pruebas y que, por consiguiente, cuentan con las propiedades deseables.

La **forma normal** de una relación hace referencia a la forma normal más alta que cumple, e indica por tanto el grado al que ha sido normalizada.

Las formas normales, consideradas aisladas de otros factores, no garantizan un buen diseño de base de datos. Generalmente, no basta con comprobar por separado que cada esquema de relación de la base de datos está en BCNF o en 3FN. En lugar de ello, el proceso de normalización por descomposición debe confirmar también la existencia de las propiedades adicionales que los esquemas relacionales, en conjunto, deben poseer.

Dos de estas propiedades son las siguientes:

- La propiedad de **reunión sin pérdida** o **reunión no aditiva**, que garantiza que no se presentará el problema de las tuplas falsas respecto a los esquemas de relación creados después de la descomposición.
- La **propiedad de conservación de las dependencias**, que asegura que todas las dependencias funcionales están representadas en alguna relación individual resultante tras la descomposición.

La propiedad de reunión no aditiva es extremadamente crítica y debe cumplirse a cualquier precio, mientras que la de conservación de las dependencias, aunque deseable, puede sacrificarse a veces.

El diseño de bases de datos, tal y como se realiza en la actualidad en la industria, presta especial atención a la normalización hasta la 3FN, la BCNF o la 4FN. Sin embargo, los diseñadores de bases de datos, no necesitan normalizar hasta la forma normal más alta posible. Por razones de rendimiento, las relaciones podrían dejarse en un estado de normalización menor, como el 2FN.

El proceso para almacenar la concatenación de relaciones de forma normal superiores como relación base (que se encuentran en una forma normal inferior) recibe el nombre de **desnormalización**.

DEFINICIONES DE CLAVES Y ATRIBUTOS QUE PARTICIPAN EN LAS CLAVES

Una superclave de un esquema de relación $R = \{A_1, A_2, \dots, A_n\}$ es un conjunto de atributos $S \subseteq R$ con la propiedad de que no habrá un par de tuplas t_1 y t_2 en ningún estado de relación permitido r de R tal que $t_1[S] = t_2[S]$. Una clave K es una superclave con la propiedad adicional de que la eliminación de cualquier atributo de K provocará que K deje de ser una superclave.

La diferencia entre una clave y una superclave es que la primera tiene que ser mínima, es decir, si tenemos una clave $K = \{A_1, A_2, \dots, A_k\}$ de R , entonces $K - \{A_i\}$ no es una clave de R para ningún A_i , $1 \leq i \leq k$.

Si un esquema de relación tiene más de una clave, cada una de ellas se denomina **clave candidata**. Una de ellas se elige arbitrariamente como **clave principal**, mientras que el resto son claves secundarias. Todo esquema de relación debe contar con una clave principal.

Un atributo del esquema de relación R recibe el nombre de atributo primo de R si es miembro de alguna de las claves candidatas de R . Un atributo es no primo si no es miembro de ninguna clave candidata.

PROPIEDADES DE LAS CLAVES CANDIDATAS

- **Unicidad:** en un momento dado, no existen dos tuplas en la relación con el mismo valor de clave candidata.
- **Minimalidad:** si la clave candidata es compuesta, no es posible destruir ningún elemento de esta clave, sin destruir su unicidad.

PRIMERA FORMA NORMAL (1FN)

La primera forma normal (**1FN**) está considerada como una parte de la definición formal de una relación en el modelo relacional básico. Fue definida para prohibir los atributos multivalor, los atributos compuestos y sus combinaciones. Afirma que el dominio de un atributo sólo debe incluir valores atómicos (simples, indivisibles) y que el valor de cualquier atributo en una tupla debe ser un valor simple

del dominio de ese atributo. Por tanto, 1FN prohíbe tener un conjunto de valores, una tupla de valores o una combinación de ambos como valor de un atributo para una tupla individual. En otras palabras, prohíbe las relaciones dentro de las relaciones o las relaciones como valores de atributo dentro de las tuplas. Los únicos valores de atributo permitidos por la 1FN son los **atómicos** (o indivisibles).

SEGUNDA FORMA NORMAL (2FN)

Está basada en el concepto de **dependencia funcional completa** (o **total**). Una dependencia funcional $X \rightarrow Y$ es completa si la eliminación de cualquier atributo A de X implica que la dependencia funcional deje de ser válida, es decir, para cualquier atributo $A \in X$, $(X - \{A\})$ no determina funcionalmente a Y. Una dependencia funcional $X \rightarrow Y$ es parcial si al eliminarse algún atributo A de X la dependencia sigue siendo válida, es decir para algún $A \in X$, $(X - \{A\}) \rightarrow Y$.

*Un esquema de relación R está en **2FN** si todo atributo no primo A en R es completa y funcionalmente dependiente de la clave principal de R.*

La comprobación para 2FN implica la verificación de las dependencias funcionales cuyos atributos del lado izquierdo forman parte de la clave principal. Si ésta contiene un único atributo, no es necesario aplicar la verificación.

La definición vista anteriormente no tiene en cuenta otras claves candidatas además de la clave principal. Si consideramos las dependencias funcionales respecto a todas las claves candidatas de una relación podemos establecer una definición general de la 2FN de la siguiente manera: Un esquema de relación R está en 2FN si cada atributo no primo A en R no es parcialmente dependiente de ninguna clave de R.

Definición: Debe estar en 1FN, y todo atributo que no forma parte de la clave depende funcionalmente en forma completa de la clave.

Remedio: nueva relación con el/los atributos X que definían a Y como clave primaria, y los atributos Y como atributos no parte de la clave.

TERCERA FORMA NORMAL (3FN)

Se basa en el concepto de **dependencia transitiva**. Una dependencia funcional $X \rightarrow Y$ en un esquema de relación R es una dependencia transitiva si existe un conjunto de atributos Z que ni es clave candidata ni un subconjunto de ninguna clave de R, y se cumple tanto $X \rightarrow Z$ como $Z \rightarrow Y$.

*Un esquema de relación R está en **3FN** si satisface 2FN y ningún atributo no primo de R es transitivamente dependiente en la clave principal.*

Intuitivamente podemos ver que cualquier dependencia funcional en la que el lado izquierdo es parte de la clave principal (subconjunto propio), o cualquier dependencia funcional en la que el lado izquierdo es un atributo no clave, implica una DF problemática. La normalización 2FN y 3FN elimina estas DFs descomponiendo la relación original en nuevas relaciones. En términos del proceso de normalización, no es necesario quitar las dependencias parciales antes que las dependencias transitivas, pero históricamente, la 3FN se ha definido con la presunción de que una relación se ha verificado antes para 2FN que para 3FN.

Si consideramos las dependencias transitivas respecto a todas las claves candidatas de una relación podemos establecer una definición general de la 3FN de la siguiente manera: Un esquema de relación R está en 3FN sí, siempre que una dependencia funcional no trivial $X \rightarrow A$ se cumple en R, ya sea (a) X una superclave de R, o (b) A un atributo primo de R.

Un esquema de relación R viola la definición general de 3FN si una dependencia funcional $X \rightarrow A$ se cumple en R y viola las dos condiciones (a) y (b) de la 3FN. La violación del apartado (b) significa que A es un atributo no primo, mientras que la vulneración del (a) implica que X no es una superclave de ninguna clave de R; por consiguiente, X podría ser no primo o ser un subconjunto propio de una clave de R. Si X

es no primo, lo que tenemos es una dependencia transitiva que viola 3FN, mientras que si X es un subconjunto propio de una clave de R, lo que aparece es una dependencia parcial que viola 3FN (y también 2FN). Por tanto, podemos declarar una definición alternativa general de 3FN de la siguiente manera: Un esquema de relación R está en 3FN si cada atributo no primo de R cumple las siguientes condiciones:

- Es completa y funcionalmente dependiente de cada clave de R.
- No depende transitivamente de cada clave de R.

Definición: Definición: Debe estar en 2FN, y ningún atributo que no forme parte de la clave debe depender funcionalmente de otro atributo que no forma parte de la clave.

Remedio: nueva relación con el/los atributos X que definían a Y como clave primaria, y los atributos Y como atributos no parte de la clave.

SQL

CONCEPTO

SQL es una herramienta para organizar, gestionar y recuperar datos almacenados en una base de datos informática. Es un lenguaje informático que se utiliza para interactuar con una base de datos. Funciona con un tipo específico de base de datos, llamado base de datos relacional.

SQL se utiliza para controlar todas las funciones que suministra un DBMS a sus usuarios:

1. **Definición de datos**, permite que un usuario defina la estructura y la organización de los datos almacenados, así como las relaciones existentes entre ellos.
2. **Recuperación de datos**, permite a un usuario o a un programa recuperar y utilizar los datos almacenados en una base de datos.
3. **Manipulación de datos**, permite a un usuario o a un programa actualizar la base de datos añadiendo datos nuevos, borrando los viejos y modificando los almacenados.
4. **Control de acceso**, puede ser utilizado para restringir la capacidad de un usuario para recuperar, añadir y modificar datos, protegiendo los datos almacenados contra accesos no autorizados.
5. **Compartición de información**, coordina la compartición de datos entre usuarios recurrentes.
6. **Integridad de datos**, define restricciones de integridad en la base de datos, protegiéndola de alteraciones debidas a actualizaciones inconsistentes o fallos del sistema.

SQL no es realmente un lenguaje completo. SQL no contiene sentencias `IF` para probar condiciones, ni sentencias `GOTO` de salto, ni siquiera sentencias `DO` o `FOR` para realizar bucles. En vez de eso, SQL es un *sublenguaje* de bases de datos que consiste en alrededor de treinta sentencias especializadas en tareas de gestión de datos. Estas sentencias SQL están *embebidas* en otro lenguaje.

Las sentencias SQL parecen frases en inglés, que tienen “palabras de relleno” que no añaden significado a la sentencias sino que hacen que se lea de forma más natural.

SQL ha emergido como el lenguaje estándar de utilización de bases de datos relacionales.

EL PAPEL DE SQL

SQL es una parte integral de un DBMS, un lenguaje y una herramienta para comunicarse con él.

SQL juega muchos papeles diferentes:

1. **Es un lenguaje interactivo de consultas**, los usuarios escriben órdenes SQL en un programa SQL interactivo para recuperar datos y presentarlos en pantalla (como MS SQL Server Management Studio).
2. **Es un lenguaje de programación de bases de datos**, los programadores introducen ordenes SQL en sus programas para acceder a los datos de las base de datos.
3. **Es su lenguaje de administración de base de datos**, el administrador responsable de la gestión de la base de datos utiliza SQL para definir la estructura de la base de datos y el control de acceso a datos almacenados.
4. **Es un lenguaje cliente/servidor**, los programas de las computadoras personales utilizan SQL para comunicarse con los servidores de base de datos.
5. **Es un lenguaje de bases de datos distribuidas**, los sistemas de gestión de bases de datos distribuidas utilizan SQL para ayudar a distribuir los datos a través de muchos sistemas informáticos conectados.

6. **Es un lenguaje pasarela de base de datos**, en una red informática de donde se mezclan diferentes productos DBMS, SQL se usa a menudo como una pasarela que permite que un determinado producto DBMS se comunique con otros diferentes.

VENTAJAS

1. **Independencia de los proveedores:** SQL se ofrece en todos los principales proveedores de DBMS.
2. **Portabilidad entre sistemas informáticos.**
3. **Estándares SQL:** el ANSI y el ISO, publicaron el estándar en 1986. El ANSI y el ISO publicaron un estándar oficial de SQL en 1986 que se extendió rápidamente en 1992.
4. **Aprobación de IBM:** fue originado por investigadores de IBM, y desde entonces se ha convertido en un producto estratégico de IBM basado en su DB2.
5. **ODBC y Microsoft:** el estándar patrocinado por Microsoft es el acceso a las base de datos abiertas, una facilidad basada en SQL.
6. **Fundamento relacional:** es un lenguaje de bases de datos relacionales.
7. **Estructura de alto nivel parecida al inglés.**
8. **Consultas interactivas *ad hoc*:** proporciona a los usuarios acceso *ad hoc* a los datos almacenados. Los datos son más accesibles y pueden ser utilizados para tomar mejores decisiones y más informadas. Los datos se obtienen rápido (no hace falta hacer un programa para hacer consultas).
9. **Acceso a la base de datos mediante programas.**
10. **Definición dinámica de datos:** usando SQL, una base de datos puede ser modificada y/o ampliada dinámicamente incluyendo su arquitectura, incluso mientras los usuarios están conectados a la base de datos.
11. **Arquitectura cliente/servidor:** sirve de vehículo para implementar aplicaciones cliente/servidor.
12. **Múltiple vistas de los datos:** se pueden crear vistas diferentes del contenido de la base de datos.
13. **Lenguaje completo de bases de datos:** sirve para crear bases de datos, gestionar la seguridad, actualizar contenido, recuperar datos y compartirlos.

HISTORIA DE SQL

Desde finales de los años ochenta un tipo específico de DBMS, llamado sistema de gestión de bases de datos relacionales (RDBMS), ha ganado una gran popularidad y se ha convertido en el estándar de bases de datos. Las bases de datos relacionales organizan los datos de forma tabular sencilla y proporcionan muchas ventajas sobre los anteriores tipos de bases de datos. SQL es específicamente un lenguaje relacional de bases de datos utilizado para trabajar con bases de datos relacionales.

El concepto de base de datos relacional fue desarrollado por el Dr. E. F. "Ted" Codd, un investigador de IBM. En junio de 1970, el Dr. Codd publicó un artículo titulado "Un modelo relacional de datos para grandes bancos de datos compartidos" que perfilaba una teoría matemática de cómo almacenar y manipular datos utilizando una estructura tabular.

LOS PRIMEROS AÑOS

El artículo desencadenó una intensa actividad de investigaciones en bases de datos relacionales, incluyendo un importante proyecto de investigación dentro de IBM. El objetivo del proyecto, llamado System/R, fue demostrar la operatividad del concepto relacional y proporcionar alguna experiencia en la implementación efectiva de un DBMS relacional.

En 1974 y 1975 la primera fase del proyecto System/R produjo un mínimo prototipo de un DBMS relacional. El proyecto incluía trabajos sobre lenguajes de consulta de bases de datos. Uno de estos lenguajes fue denominado SEQUEL. En 1976 y 1977 el prototipo de investigación System/R fue reescrito desde el principio. La nueva implementación soportaba consultas multitable y permitía que varios usuarios compartieran el acceso a datos.

En 1978 y 1979, la implementación de System/R fue distribuida entre diversas instalaciones de clientes de IBM para ser evaluada, y renombraron el lenguaje de consultas a SQL (*Structured Query Language*, lenguaje estructurado de consultas).

En 1979 el proyecto de investigación System/R llegó a su fin, e IBM concluyó que las bases de datos relacionales no solamente eran factibles, sino que podrían ser la base de un producto comercial útil.

PRIMEROS PRODUCTOS RELACIONALES

Durante los años setenta, el proyecto System/R y su lenguaje SQL recibieron buenos comentarios e IBM se entusiasmó con la tecnología de bases de datos relacionales.

La publicidad referente al System/R atrajo la atención de un grupo de ingenieros en Menlo Park, California, que presagiaban un mercado comercial para las BD relacionales. En 1977 formaron una compañía llamada Relational Software, Inc., para construir un DBMS basado en SQL. El producto, llamado Oracle, apareció en 1979, y se convirtió en el primer DBMS relacional disponible comercialmente, adelantándose a IBM.

PRODUCTOS IBM

En 1983, IBM introdujo también Database 2 (DB2), otro DBMS relacional para sus mainframes. DB2 se ejecutaba con el sistema operativo MVS de IBM. La primera versión de DB2 comenzó a entregarse en 1985, y los directivos de IBM lo aclamaron como una pieza estratégica en la tecnología software de IBM. Desde entonces, DB2 se convirtió en el principal producto DBMS relacional de IBM, y SQL se convirtió en estándar.

ACEPTACIÓN COMERCIAL

Los productos relacionales tenían varias desventajas cuando se comparaban con las arquitecturas tradicionales de bases de datos. El rendimiento de las bases de datos relacionales era inferior al de las bases de datos tradicionales.

Sin embargo, los productos relacionales tenían ciertamente una ventaja importante. Sus lenguajes de consulta relacional (SQL, QUEL y otros) permitían a los usuarios realizar consultas *ad hoc* a la base de datos, y obtener respuestas inmediatamente, sin escribir programas. Lentamente, las BD relacionales comenzaron a introducirse lentamente en las aplicaciones de centros de información como herramientas de soporte a decisiones.

Durante la última mitad de los ochenta, SQL y las bases de datos relacionales fueron rápidamente aceptadas como la tecnología de bases de datos del futuro. El rendimiento de los productos de bases de datos relacionales mejoró enormemente.

La publicación del estándar ANSI/ISO para SQL en 1986 dio estatus “oficial” a SQL como estándar. SQL también emergió como una estándar para sistemas informáticos basados en UNIX, cuya popularidad se aceleró en los ochenta.

SQL se convirtió en una parte esencial de la arquitectura/cliente servidor que conectaba los PC con servidores en red en sistemas de procesamiento de información de bajo coste.

SQL se había convertido, tanto en el estándar de hecho, como oficial, de bases de datos relacionales.

ESTÁNDARES SQL

Después de varias revisiones, el estándar fue oficialmente adoptado como estándar ANSI X3.135 en 1986, y como estándar ISO en 1987. Este estándar, revisado y ampliado ligeramente en 1989, es llamado generalmente “SQL-89” o estándar “SQL 1”.

EL ESTÁNDAR SQL2

Para afrontar los agujeros del estándar original, el comité ANSI continuó su trabajo, haciendo circular borradores de un nuevo estándar SQL2 más riguroso. El borrador de SQL2 especificaba características considerablemente más avanzadas de las encontradas en los productos SQL actuales.

El estándar SQL2 siguió su camino por el proceso de aprobación ANSI, y finalmente fue aprobado en 1992.

A pesar de la existencia de un estándar SQL2, actualmente no hay ningún producto comercial disponible que implemente todas sus características, ni siquiera hay dos productos SQL comerciales que ofrezcan el mismo dialecto SQL. Además, según los vendedores de bases de datos introducen nuevas capacidades, se amplían sus dialectos SQL, alejándolos entre ellos todavía más. Sin embargo, el núcleo central del lenguaje SQL está bastante estandarizado.

EL MITO DE LA PORTABILIDAD

La existencia de un estándar SQL publicado ha generado unas cuantas aseveraciones exageradas acerca de la portabilidad de SQL y sus aplicaciones. De hecho, las insuficiencias del estándar SQL-89 y las diferencias actuales entre los dialectos SQL son lo suficientemente significativas para que una aplicación debe ser siempre modificada cuando se traslada de una base de datos SQL a otra. Estas diferencias, muchas de las cuales son eliminadas por el estándar SQL2 incluyen.

1. **Códigos de error:** el estándar SQL-89 (o SQL1) no especifica los códigos de error a devolver cuando SQL detecta un error, y todas las implementaciones comerciales utilizan su propio conjunto de códigos de error. SQL2 especifica códigos de error estándar.
2. **Tipos de datos:** el estándar SQL1 define un conjunto mínimo de tipos de datos, pero omite alguno de los tipos más populares y útiles, tales como las cadenas de caracteres de longitud variable, las fechas y horas y los datos monetarios. SQL2 afronta esto.
3. **Tablas del sistema:** el estándar SQL1 no dice nada acerca de las tablas del sistema que proporcionan información referente de la estructura de la propia base de datos. Cada vendedor tiene su propia estructura para estas tablas. Las tablas se estandarizaron en SQL2.
4. **SQL interactivo:** el estándar especifica el SQL de programa utilizado por un programa de aplicación, y no el SQL interactivo.
5. **Interfaz de programa:** el estándar SQL1 especifica una técnica abstracta para utilizar SQL desde dentro de un programa de aplicaciones. Ningún producto SQL comercial utiliza esta técnica. SQL2 especifica una interfaz SQL embebida para lenguajes de programación populares, pero no interfaces a nivel de llamada.
6. **SQL dinámico:** el estándar SQL1 no incluye las características requeridas para desarrollar frontales de bases de datos de propósito general, tales como herramientas de consulta y generadores de informes. SQL2 lo incluye.
7. **Diferencias semánticas:** es posible ejecutar la misma consulta con dos implementaciones SQL diferentes, y producir resultados diferentes.
8. **Secuencias de cotejo:** el estándar SQL1 no define la secuencia de cotejo (ordenación) de los caracteres almacenados en la base de datos. SQL2 incluye una especificación detallada.
9. **Estructura de bases de datos:** los detalles del nombrado de la base de datos y de cómo se establece la conexión inicial varían enormemente y no son portables. SQL2 crea más uniformidad, pero no puede enmascarar completamente estos detalles.

ARQUITECTURAS

1. **Arquitectura centralizada:** se basa en un computador central, en el cual está alojado la base de datos y el programa de aplicación. Tanto el DBMS como los datos físicos residen un mismo lugar, junto con el programa de aplicaciones que acepta entradas desde el terminal de usuario. El DBMS accede a la base de datos para extraer los registros almacenados recuperando los datos necesarios, y el programa de aplicación muestra el resultado en pantalla. El sistema es multiusuario por lo que cuando acceden varios usuarios, se degrada el rendimiento.

2. **Arquitectura servidora de archivos:** una aplicación que se ejecuta en una computadora personal puede acceder de forma transparente a los datos localizados en un servidor de archivos, que almacena los archivos compartidos. Esta arquitectura proporciona un rendimiento excelente para consultas típicas, ya que cada usuario dispone de la potencia completa de una computadora personal ejecutando su propia copia del DBMS. Produce un intenso tráfico de red y un bajo rendimiento para consultas de este tipo.
3. **Arquitectura cliente/servidor:** las computadoras personales están conectadas en una red junto a un servidor de bases de datos que almacena las bases de datos compartidas. Las funciones del DBMS están divididas en dos partes. El *front-end* de bases de datos, tales como herramientas de consulta interactiva, generadores de informe y programas de aplicación, se ejecutan en la computadora personal. El *back-end* de la base de datos que almacena y gestiona los datos se ejecuta en el servidor.
 El servidor de bases de datos recibe una petición SQL, la procesa y la ejecuta, y cuando obtiene el resultado, envía una respuesta, a través de la red, y la aplicación del *front-end* la muestra en la pantalla.
 Reduce el tráfico de red y divide la carga de trabajo de la base de datos. Las funciones íntimamente relacionadas con el usuario, como la entrada y la visualización de los datos, se ejecutan en el PC.

SENTENCIAS

El lenguaje SQL consta de unas treinta sentencias. Cada sentencia demanda una acción específica por parte del DBMS.

Todas las sentencias SQL comienzan con un **verbo**. Luego, continua con una o más **cláusulas**. Todas las cláusulas comienzan también con una palabra clave. Algunas cláusulas son opcionales, otras son necesarias.

El estándar SQL1 ANSI/ISO especifica las palabras clave SQL que se utilizan como verbos y en cláusulas de sentencias. Según el estándar, estas palabras clave no pueden ser utilizadas para designar objetos de la base de datos (tablas, columnas, etc.). Algunas palabras clave son:

Manipulación de datos	Definición de datos	Control de acceso	Control de transacc.
SELECT	CREATE TABLE	GRANT	COMMIT
INSERT	DROP TABLE*	REVOKE	ROLLBACK
DELETE	ALTER TABLE*		
UPDATE	CREATE VIEW*		

* No están en el estándar pero están incluidas en la mayoría de los productos SQL.

Entre otras. El estándar SQL2 expande la lista a cerca de 300 palabras clave.

NOMBRES

Los objetos de una base de datos basada en SQL se identifican asignándoles nombres únicos. Los nombres se utilizan en las sentencias SQL para identificar el objeto de la base de datos sobre la que la sentencia debe actuar.

El estándar ANSI/ISO especifica que los nombres SQL deben contener de 1 a 18 caracteres, comenzar con una letra, y que no puedan contener espacios o caracteres de puntuación especiales. En la práctica, los nombres contemplados por los productos DBMS basados en SQL varían significativamente.

Por portabilidad es mejor mantener los nombres relativamente breves y evitar el uso de caracteres especiales.

NOMBRES DE TABLA

Hace referencia al nombre del objeto tabla. Existen los nombres de tabla cualificados, que especifica el nombre del propietario de la tabla junto con el nombre de la tabla, separados por un punto (.).

NOMBRES DE COLUMNA

Cuando se especifica un nombre de columna en una sentencia SQL, SQL puede determinar normalmente a qué columna se refiere a partir del contexto. Sin embargo, si la sentencia afecta a dos columnas con el mismo nombre correspondientes a dos tablas diferentes, debe utilizarse un nombre de columna cualificado para identificar sin ambigüedad la columna designada. Un nombre de columna cualificado especifica tanto el nombre de la tabla que contiene la columna como el nombre de la columna, separados por un punto (.).

TIPOS DE DATOS

El estándar SQL ANSI/ISO especifica varios tipos de datos que pueden ser almacenados en una base de datos basada en SQL y manipulado por el lenguaje SQL.

TIPOS DE DATOS DE SQL1

1. **Cadena de caracteres de longitud fija:** almacenan típicamente nombres de personas y empresas, direcciones, descripciones, etc.
2. **Enteros:** almacenan típicamente cuentas, cantidades, edades, etc. Se utilizan con frecuencia para mantener números identificadores, tales como número de cliente, de empleado y de pedido.
3. **Números decimales:** almacenan números que tienen una parte fraccionaria y deben ser calculados exactamente; ejemplo: porcentajes y tasas. Se utilizan frecuentemente para almacenar importes monetarios.
4. **Números en coma flotante:** se utilizan para almacenar números científicos que pueden ser calculados aproximadamente, tales como pesos y distancias.

TIPOS DE DATOS EXTENDIDOS (SQL2)

1. **Cadenas de caracteres de longitud variable:** permiten que una columna almacene cadenas de caracteres que varían de longitud de una fila a otra, hasta una longitud máxima.
2. **Importes monetarios:** al tener un tipo monetario distinto permite al DBMS dar formato adecuadamente a los importes monetarios cuando son visualizados.
3. **Fechas y horas:** permiten almacenar valores de fechas, horas y fechas/horas, aunque los detalles varían ampliamente de un producto a otro.
4. **Datos booleanos:** algunos productos SQL permiten los valores lógicos (`TRUE` o `FALSE`) como un tipo explícito.
5. **Texto extenso:** permiten columnas que almacenan cadenas de texto extensas (típicamente hasta 32 000 o 65 000 caracteres, incluso más).
6. **Flujos de bytes no estructurados:** permiten almacenar y recuperar secuencias de bytes de longitud variable sin estructurar. Se utilizan para almacenar imágenes, videos, código ejecutable y otros tipos de datos sin estructurar.
7. **Caracteres asiáticos.**

CONSTANTES

En algunas sentencias SQL, un valor de datos numérico, de caracteres, o de fecha debe ser expresado en forma textual.

El estándar SQL ANSI/ISO especifica el formato de las constantes numéricas y de caracteres, o literales, que representan valores de datos específicos.

CONSTANTES NUMÉRICAS

Las constantes enteras y decimales se escriben como números decimales ordinarios dentro de las sentencias SQL, con un signo más o menos opcional adelante.

21 -375 2000.00 +497500.8778

Las constantes en coma flotante se especifican utilizando la notación *E* hallada comúnmente en lenguajes de programación tales como C y FORTRAN.

1.5E3 -314159E1 2.5E-7 0.7839E21

CONSTANTES DE CADENA

El estándar ANSI/ISO especifica que las constantes SQL de caracteres han de ir encerradas entre comillas simples ('...').

'Jones, John J.' 'Bruce Wayne' 'Batman'

Si una comilla simple debe incluirse en el texto constante, ésta se escribe como dos caracteres comilla simple consecutivos:

'I can''t'

CONSTANTES DE FECHA Y HORA

En productos SQL que incluyen datos fecha/hora, los valores constantes para las fechas, horas e intervalos de tiempo se especifican como constantes de cadenas de caracteres. El formato de estas constantes varía de un DBMS a otro (o sea dd/MM/yyyy, yyyy-MM-dd, etc.)

CONSTANTES SIMBÓLICAS

El lenguaje SQL incluye constantes simbólicas especiales que devuelven valores de datos mantenidos por el propia DBMS, como:

CURRENT DATE USER SYSDATE ROWID

EXPRESIONES

Las expresiones se utilizan en el lenguaje SQL para calcular valores que se recuperar de una base de datos y para calcular valores utilizados en la búsqueda en la base de datos. Ejemplo:

CANTIDAD * PRECIO_UNITARIO OBJETIVO + 500

El estándar SQL ANSI/ISO especifica cuatro operaciones aritméticas que pueden usarse en expresiones: suma (X+Y), resta (X-Y), multiplicación (X*Y) y división (X/Y).

FUNCIONES INTERNAS

Aunque el estándar SQL1 no las especifica, la mayoría de las implementaciones SQL incluyen una serie de *funciones internas útiles*. Estas utilidades proporcionan con frecuencia facilidades de conversión de tipo de datos.

MONTH(FECHA_PEDIDO) DATEADD(dd, 5, FECHA_ALTA) GETDATE()

En general, una función interna puede ser especificada en una expresión SQL en cualquier lugar en que una constante del mismo tipo de datos pueda ser especificada.

FALTA DE DATOS (VALORES NULL)

SQL incorpora explícitamente los datos que faltan, son desconocidos o no son aplicables, a través del concepto de *valor nulo*. Un valor nulo es un *indicador* que dice a SQL (y al usuario) que el dato falta o que no es aplicable. Por conveniencia, un dato que falta normalmente se dice que tiene el valor NULL, pero el valor NULL, no es un valor de dato real. Es una señal o recordatorio de que el valor falta o es desconocido.

CONSULTAS SIMPLES

LA SENTENCIA SELECT

La sentencia `SELECT` recupera datos de una base de datos y los devuelve en forma de resultados de la consulta.

```
SELECT [lista_de_columnas]
FROM [tabla]
WHERE [condición_de_búsqueda]
GROUP BY [columnas_de_agrupación]
HAVING [condición_de_búsqueda]
ORDER BY [orden_de_columnas]
```

- La cláusula `SELECT` lista los datos a recuperar por la sentencia `SELECT`. Los ítems pueden ser columnas de la base de datos o expresiones SQL para calcular por SQL cuando se efectúe la consulta.
- La cláusula `FROM` lista las tablas que contienen los datos a recuperar por la consulta.
- La cláusula `WHERE` dice a SQL que incluye sólo ciertas filas de datos en los resultados de la consulta. Se utiliza una condición de búsqueda para especificar las filas deseadas.
- La cláusula `GROUP BY` especifica una consulta resumen. En vez de producir una fila de resultados por cada fila de datos de la base de datos, una consulta resumen agrupa todas las filas similares y luego produce una fila resumen de resultados para cada grupo.
- La cláusula `HAVING` dice a SQL que incluya sólo ciertos grupos producidos por la cláusula `GROUP BY`. Utiliza una condición de búsqueda para especificar los grupos deseados (igual que `WHERE` pero en vez de filas, grupos).
- La cláusula `ORDER BY` ordena los resultados de la consulta en base a los datos de una o más columnas.

LA CLÁUSULA SELECT

La cláusula `SELECT` que empieza cada sentencia `SELECT` especifica los ítems de datos a recuperar por la consulta. Cada ítem de selección de la lista genera una única columna de resultados de consulta, en orden de izquierda a derecha. Un ítem de selección puede ser:

- Un **nombre de columna**, identificando una columna de la tabla designada en la cláusula `FROM`.
- Una **constante**, especificando que el mismo valor constante va a aparecer en todas las filas de los resultados de la consulta.
- Una **expresión SQL**, indicando que SQL debe calcular el valor a colocar en los resultados, según el estilo especificado por la expresión.

LA CLÁUSULA FROM

La cláusula `FROM` consta de la palabra clave `FROM`, seguida de una lista de especificaciones de tablas separadas por comas. Cada especificación de tabla identifica una tabla que contiene datos a recuperar por la consulta.

RESULTADOS DE CONSULTAS

El resultado de una consulta SQL es siempre una tabla de datos, semejante a las tablas de la base de datos.

El hecho de que una consulta SQL produzca una tabla de datos es muy importante. Significa que los resultados de la consulta pueden volverse a almacenar en la base de datos como una tabla. Finalmente, significa que los resultados de la consulta pueden ser objetivo ellos mismo de consultas adicionales.

CONSULTAS SENCILLAS

Las consultas SQL más sencillas solicitan columnas de datos de una única tabla en la base de datos.

Ejemplo:

```
SELECT Ciudad, Region, Ventas
FROM Oficinas
```

Conceptualmente, SQL procesa la consulta recorriendo la tabla nominada en la cláusula FROM, una fila cada vez. Por cada fila, SQL toma los valores de las columnas solicitadas en la lista de selección y produce una única fila de resultados. Los resultados contienen por tanto una fila de datos por cada fila de la tabla.

COLUMNAS CALCULADAS

Una consulta SQL puede incluir *columnas calculadas* cuyos valores se calculan a partir de los valores de los datos almacenados. Para solicitar una columna calculada, se especifica una expresión SQL en la lista de selección. Las expresiones SQL pueden contener sumas, restas, multiplicaciones y divisiones.

Naturalmente las columnas referenciadas en una expresión aritmética deben tener un tipo numérico.

Ejemplo:

```
SELECT Ciudad, Region, (Ventas - Objetivo)
FROM Oficinas
```

SELECCIÓN DE TODAS LAS COLUMNAS (SELECT *)

A veces es conveniente visualizar el contenido de todas las columnas de una tabla. Por conveniencia, SQL permite utilizar un asterisco (*) en lugar de la lista de selección como abreviatura de “todas las columnas”.

Ejemplo:

```
SELECT * FROM Oficinas
```

El estándar SQL ANSI/ISO especifica que una sentencia SELECT puede tener o bien una selección de todas las columnas o bien una lista de selección, pero no ambas. Sin embargo, muchas implementaciones SQL tratan al asterisco (*) como cualquier otro elemento de la lista de selección.

La selección de todas las columnas es muy adecuada cuando se está utilizando el SQL interactivo de forma casual. Debería evitarse en SQL programado que cambios en la estructura de la base de datos pueden hacer que un programa falle.

FILAS DUPLICADAS (DISTINCT)

Si una consulta incluye la clave primaria de una tabla en su lista de selección, entonces cada fila de resultados será única. Si no se incluye la clave primaria en los resultados, pueden producirse filas duplicadas.

Se pueden eliminar las filas duplicadas de los resultados de la consulta insertando la palabra clave **DISTINCT** en la sentencia **SELECT** justo antes de la lista de selección.

SQL efectúa esta consulta generando primero un conjunto completo de resultados y eliminando luego las filas que son duplicados exactos de alguna otra para formar los resultados finales.

SELECCIÓN DE FILA (CLÁUSULA WHERE)

La cláusula **WHERE** se emplea para especificar las filas que se desean recuperar.

Ejemplo:

```
SELECT Ciudad, Ventas, Objetivos
FROM Oficinas
WHERE Ventas > Objetivo
```


La cláusula `WHERE` consta de la palabra clave `WHERE` seguida de una condición de búsqueda que especifica las filas a recuperar. Por cada fila, la condición de búsqueda puede producir uno de los tres resultados:

- Si la condición de búsqueda es `TRUE`, la fila se incluye en los resultados de la consulta.
- Si la condición de búsqueda es `FALSE`, la fila se excluye de los resultados de la consulta.
- Si la condición de búsqueda tiene un valor `NULL`, la fila se excluye de los resultados de la consulta.

Básicamente, la condición de búsqueda actúa como un filtro para las filas de la tabla. Las filas que satisfacen la condición de búsqueda atraviesan el filtro y forman parte de los resultados de la consulta. Las filas que no satisfacen la condición de búsqueda son atrapadas por el filtro y quedan excluidas de los resultados de la consulta.

CONDICIONES DE BÚSQUEDA

1. **Test de comparación:** compara el valor de una expresión con el valor de otra.
2. **Test de rango:** examina si el valor de una expresión cae dentro de un rango especificado de valores.
3. **Test de pertenencia a conjunto:** comprueba si el valor de una expresión se corresponde con uno de un conjunto de valores.
4. **Test de correspondencia con patrón:** comprueba si el valor de una columna que contiene datos de cadena de caracteres se corresponde a un patrón especificado.
5. **Test de valor nulo:** comprueba si una columna tiene un valor `NULL`.

TEST DE COMPARACIÓN (`=`, `<`, `>`, `<=`, `>=`)

En un test de comparación, SQL calcula y compara los valores de dos expresiones SQL por cada fila de datos. Las expresiones pueden ser tan simples como un nombre de columna o una constante, o pueden ser expresiones aritméticas más complejas.

Ejemplo:

```
SELECT Nombre
FROM RepVentas
WHERE Contrato < '01-ENE-99'
```

TEST DE RANGO (`BETWEEN`)

El test de rango comprueba si un valor de dato se encuentra entre dos valores especificados. Implica el uso de tres expresiones SQL. La primera expresión define el valor a comprobar, las expresiones segunda y tercera definen los extremos superior e inferior del rango a comprobar. Los tipos de datos de las expresiones deben ser comprobables.

Ejemplos:

```
SELECT NumPedido, Importe
FROM Pedidos
WHERE Importe BETWEEN 2000.00 AND 2999.99

SELECT NumPedido, Importe, Cliente
FROM Pedidos
WHERE Fecha BETWEEN '01-01-2001' AND '31-08-2001'

SELECT Nombre, Ventas, Cuota
FROM RepVentas
WHERE Ventas NOT BETWEEN (0.8 * Cuota) AND (1.2 * Cuota)
```

También se puede comprobar si el valor del dato `NO` se encuentra dentro del rango. Eso se hace con `NOT BETWEEN`.

TEST DE PERTENENCIA A CONJUNTO (IN)

Examina si un valor de dato coincide con uno de una lista de valores objetivo.

Ejemplo:

```
SELECT Nombre, Cuota, Ventas
FROM RepVenas
WHERE OficinaRep IN (11, 13, 22)
```

Se puede comprobar si el valor del dato no corresponde a ninguno de los valores objetivos usando la forma `NOT IN` del test de pertenencia a conjunto.

TEST DE CORRESPONDENCIA CON PATRÓN (LIKE)

Se puede utilizar un test de correspondencia con patrón para recuperar las filas en donde el contenido de una consulta de texto se corresponde con un cierto texto particular.

El patrón es una cadena que puede incluir uno o más caracteres *comodines*. Estos caracteres se interpretan de una manera especial.

CARACTERES COMODINES

- **Porcentaje (%):** se corresponde con cualquier secuencia de cero o más caracteres.
- **Guion bajo (_):** se corresponde con cualquier carácter simple.

Los caracteres comodines pueden aparecer en cualquier lugar de la cadena patrón, y puede haber varios caracteres comodines dentro de una misma cadena.

Ejemplo:

```
SELECT Empresa, LimiteCredito
FROM Clientes
WHERE Empresa LIKE 'Arc%' AND Empresa NOT LIKE '_far%'
```

Se pueden localizar cadenas que no se ajusten a un patrón utilizando el formato `NOT LIKE` del test de correspondencia de patrones. El test `LIKE` debe aplicarse a una columna con un tipo de dato cadena.

TEST DE VALOR NULO (IS NULL)

A veces es útil comprobar explícitamente los valores `NULL` en una condición de búsqueda y gestionarlos directamente.

Ejemplo:

```
SELECT Nombre
FROM RepVentas
WHERE OficinaRep IS NULL
```

La forma negada del test de valor nulo (`IS NOT NULL`) encuentra las filas que no contienen un valor `NULL`.

A diferencia de las condiciones de búsqueda anteriores, el test de valor nulo no puede producir resultados `NULL`. Será siempre `TRUE` o `FALSE`.

CONDICIONES DE BÚSQUEDAS COMPUESTAS (AND, OR Y NOT)

Las condiciones de búsqueda simples, descritas anteriormente, devuelven un valor `TRUE`, `FALSE` o `NULL` cuando se aplican a una fila de datos.

Utilizando las reglas de la lógica, se pueden combinar estas condiciones de búsqueda SQL simples para formar otras más complejas.

- **OR:** se utiliza para combinar dos condiciones de búsqueda cuando una o la otra (o ambas) deben ser ciertas.
- **AND:** se utiliza para combinar dos condiciones de búsqueda cuando ambas deban ser ciertas.
- **NOT:** se utiliza para seleccionar filas en donde la condición de búsqueda es falsa.

Utilizando estas palabras clave y los paréntesis para agrupar los criterios de búsqueda, se pueden construir criterios de búsqueda muy complejos.

El estándar SQL2 añade otra condición lógica de búsqueda, el test **IS** a la lógica proporcionada por **AND**, **OR** y **NOT**. **IS** comprueba si el valor lógico de una expresión o comparación es **TRUE**, **FALSE** o **UNKNOWN** (**NULL**). Aunque conviene evitarse este tipo de test y solo usar **AND**, **OR** y **NOT** para no tener problemas de portabilidad.

Ejemplo:

```
SELECT ... FROM ...
WHERE OficinaRep = 2 AND ((Ventas - Cuota) > 10000.00) IS FALSE
```

ORDENACIÓN DE LOS RESULTADOS DE UNA CONSULTA (CLÁUSULA ORDER BY)

Las filas de los resultados de una consulta no están dispuestas en ningún orden particular. Se puede pedir a SQL que ordene los resultados de una consulta incluyendo la cláusula **ORDER BY** en la sentencia **SELECT**. La cláusula **ORDER BY** consta de las palabras clave **ORDER BY**, seguidas de una lista de especificaciones de ordenación separadas por comas.

Por defecto, SQL ordena los datos en secuencia ascendente. Para solicitar ordenación en secuencia descendente, se incluye la palabra clave **DESC** en la especificación de ordenación.

También se puede utilizar la palabra clave **ASC** para especificar el orden ascendente, pero se suele omitir ya que es la secuencia de ordenación por defecto.

Si la columna de resultados de la consulta utilizada para ordenación es una columna calculada, no tiene nombre de columna que se pueda emplear en una especificación de ordenación. En este caso, debe especificarse un número de columna en lugar de un nombre. También se puede utilizar un alias.

Ejemplo:

```
SELECT Apellido, Nombre, Ventas
FROM Empleados
ORDER BY Apellido, Nombre, Ventas DESC

SELECT Ciudad, Region, (Ventas - Objetivo)
FROM Oficinas
ORDER BY 3 DESC

SELECT Ciudad, Region, (Ventas - Objetvio) AS Monto
FROM Oficinas
ORDER BY Monto DESC
```

REGLAS PARA PROCESAMIENTO DE CONSULTAS DE TABLA ÚNICA

Los resultados producidos por una sentencia **SELECT**, se especifica aplicando cada una de sus cláusulas, una por una. La cláusula **FROM** se aplica en primer lugar, luego se aplica la cláusula **WHERE**, posteriormente se aplica la cláusula **SELECT**, finalmente se aplica la cláusula **ORDER BY** para ordenar los resultados.

PASOS

1. Comenzar con la tabla designada en `FROM`.
2. Si hay cláusula `WHERE`, aplicar condición de búsqueda a cada fila de la tabla, reteniendo aquellas filas para las cuales la condición de búsqueda es `TRUE`.
3. Se arma la lista de selección.
4. Si se especifica `SELECT DISTINCT`, se eliminan las filas duplicadas.
5. Si hay una cláusula `ORDER BY`, se ordenan los resultados de la consulta según la especificación.

COMBINACIÓN DE LOS RESULTADOS DE UNA CONSULTA (`UNION`)

La operación `UNION` produce una única tabla de resultados que combina las filas de la primera consulta con las filas de los resultados de la segunda consulta. En ambas consultas no se puede usar `ORDER BY`, sino que puede ser utilizado después de la segunda sentencia `SELECT`.

Restricciones:

1. Ambas tablas deben contener el mismo número de columnas.
2. El tipo de dato de cada columna en la primera tabla debe ser el mismo que el tipo de datos de la columna correspondiente en la segunda tabla (mismo dominio).
3. Ninguna de las dos tablas puede estar ordenadas con la cláusula `ORDER BY`. Sin embargo, los resultados combinados pueden ser ordenados.
4. El estándar solamente permite nombres de columna o una especificación de todas las columnas en la lista de selección, y prohíbe las expresiones en la lista de selección.

Por omisión la unión elimina las filas duplicadas como parte de su procesamiento. Si se desea duplicar las filas, se debe incorporar la palabra `ALL`. Si se sabe que la combinación de las consultas no produce filas duplicadas, debe agregarse el `ALL`, ya que aumenta el tiempo de la consulta al no ordenarle al motor que revise la duplicación de las filas.

Ejemplo:

```
SELECT IdFab, IdProducto
FROM Productos
WHERE Precio > 2000.00
UNION
SELECT DISTINCT Fab, Producto
FROM Pedidos
WHERE Importe > 30000.00
```

CONSULTAS MULTITABLA (COMPOSICIONES)

Consiste en la recuperación de datos procedentes de dos o más tablas de la base de datos, a través de la sentencia `SELECT`.

Producto cartesiano: es la combinación de todas las tuplas (filas) de una tabla con todas las tuplas de la otra.

SQL permite recuperar datos que responden a estas peticiones mediante consultas multitabla que componen (*join*) datos procedentes de dos o más tablas.

COMPOSICIONES SIMPLES (EQUICOMPOSICIONES)

El proceso de formar parejas de filas haciendo coincidir los contenidos de las columnas relacionadas se denomina componer (*joining*) de las tablas, la tabla resultante se denomina composición entre las dos tablas.

Ejemplo:

```
SELECT NumPedido, Importe, Empresa, LimiteCredito
FROM Pedidos, Clientes
WHERE Clie = NumClie
```

CONSULTAS PADRE/HIJO

Las consultas más comunes implican a dos tablas que tienen una relación natural padre/hijo.

Las claves ajenas y las claves primarias crean relaciones padre/hijo en una base de datos. La tabla que contiene la clave ajena es el hijo en la relación, la tabla con la clave primaria es el padre. Para realizar la consulta padre/hijo, debe especificarse la comparación de las claves en la condición de búsqueda.

En general, las composiciones sobre las columnas de emparejamiento arbitrarias generan relaciones de muchos a muchos.

CONSIDERACIONES SQL PARA CONSULTAS MULTITABLA

- Los nombres de columna cualificados son necesarios a veces en consultas multitabla para eliminar referencias de columnas ambiguas. Cuando el nombre de una columna aparece en más de una tabla.
- Las selecciones de todas las columnas (`SELECT *`) tienen un significado especial para las consultas multitabla. Ejemplo:

```
SELECT Ventas.* FROM Ventas
WHERE Monto > 200
```

```
SELECT v.* FROM Ventas v
WHERE Monto > 200
```

- Las autocomposiciones pueden ser utilizadas para crear una consulta multitabla que relaciones una tabla consigo misma.
- Los alias de tablas pueden ser utilizados en la cláusula `FROM` para simplificar nombre de columna cualificados y permitir referencias de columna no ambiguas en autocomposiciones. Se refiere a hacer, por ejemplo:

```
SELECT e.NumEmpleado, e.Nombre, j.Nombre AS Jefe
FROM Empleados e, Empleados j
WHERE e.Jefe = j.NumEmpleado
```

La cláusula `FROM` tiene dos funciones importantes:

1. Identifica todas las tablas que contribuyen con datos a los resultados de la consulta. Las columnas referenciadas en la sentencia `SELECT` deben provenir de una de las tablas designadas en la cláusula `FROM`.
2. Determinar la marca que se utiliza para identificar la tabla en la referencia de columna cualificada dentro de la sentencia `SELECT`. Si se especifica un alias, esta pasa a ser la marca de tabla; en caso contrario se utiliza como marca el nombre de la tabla, tal como aparece en la cláusula `FROM`.

Consideraciones:

- Cualificar atributos utilizados y que figuran en ambas tablas (por lo menos).
- Emparejar clave primaria con clave ajena en el `WHERE`.
- Posibilidad de simplificación, utilizando alias de tablas.
- Efecto del `*`, sólo y cualificado.
- No aparecen las filas sin emparejar.
- El efecto de `NULL` en clave foránea.

REGLAS DE PROCESAMIENTO DE CONSULTAS MULTITABLA

PASOS

1. Formar el producto de las tablas indicadas en la cláusula `FROM`.
2. Si hay una consulta `WHERE`, aplicar su condición de búsqueda a cada fila de la tabla producto, reteniendo aquellas filas para las cuales la condición de búsqueda es verdadera.
3. Extraer las columnas seleccionadas en la cláusula `SELECT`.
4. Si se especifica `SELECT DISTINCT`, eliminar las filas duplicadas de los resultados que se hubieran producido.
5. Si hay una cláusula `ORDER BY`, ordenar los resultados de las consultas.

COMPOSICIONES EXTERNAS

La composición SQL estándar tiene el potencial de perder información si las tablas que se componen contienen filas sin emparejar. Para evitar esto se utiliza la **composición externa**.

La composición externa es una extensión de la composición estándar (*composición interna*). El estándar SQL1 especifica la composición interna, pero no la externa.

Ejemplo:

```
SELECT *
FROM Chicas, Chicos
WHERE Chicas.Ciudad *=* Chicos.Ciudad
```

1. Comenzar con la composición interna de las dos tablas.
2. Por cada fila de la primera tabla que no haya correspondido a ninguna fila de la segunda tabla, añadir una fila a los resultados, utilizando los valores de las columnas de la primera tabla, y suponiendo un valor `NULL` para todas las columnas de la segunda tabla.
3. Por cada fila de la segunda tabla que no haya correspondido a ninguna fila de la primera tabla, añadir una fila a los resultados, utilizando los valores de las columnas de la segunda tabla, y suponiendo un valor `NULL` para todas las columnas de la primera tabla.
4. La tabla resultante es la composición externa de las dos tablas.

COMPOSICIONES EXTERNAS IZQUIERDA Y DERECHA

- **Composición externa izquierda:** aquí se omite el paso 3. Incluye copias `NULL`, ampliadas de las filas no emparejadas de la primera tabla (izquierda), pero no incluye las filas no emparejadas de la segunda tabla (derecha). Se denota con: `*=`.
- **Composición externa derecha:** lo mismo que la composición externa izquierda, pero al revés. Se denota con: `=*`.

COMPOSICIONES Y EL ESTÁNDAR SQL2

SQL2 especificó un nuevo método para permitir las composiciones externas. La especificación SQL2 permite las composiciones externas por medio de la cláusula `FROM`, con una elaborada sintaxis que permite que el usuario especifique exactamente cómo se han de componer las tablas fuentes de una consulta.

Con estas características, el soporte del estándar SQL2 a la composición externa tiene dos ventajas. La primera es que el estándar SQL2 puede expresar las composiciones más complejas. La segunda es que los productos de bases de datos existentes pueden dar soporte a las extensiones de SQL2 a SQL1 y al mismo tiempo mantener el soporte a sus propias composiciones externas sin conflicto.

COMPOSICIONES INTERNAS EN SQL2

Se hace uso de las palabras clave `INNER JOIN`.

Ejemplo en SQL1:

```
SELECT *
FROM Chicas, Chicos
WHERE Chicas.Ciudad = Chicos.Ciudad
```

Mismo ejemplo en SQL2:

```
SELECT *
FROM Chicas
INNER JOIN Chicos ON Chicas.Ciudad = Chicos.Ciudad
```

COMPOSICIONES EXTERNAS EN SQL2

- `FULL OUTER JOIN`: composición externa completa.
- `LEFT OUTER JOIN`: composición externa izquierda.
- `RIGHT OUTER JOIN`: composición externa derecha.
- Otras.

CONSULTAS RESUMEN (O SUMARIAS)

Permiten resumir datos de una base de datos mediante funciones que aceptan una columna como argumento y producen un dato resumen.

FUNCIONES DE COLUMNA

SQL permite resumir datos de la base de datos mediante un conjunto de funciones de columna. Una función de columna SQL acepta una columna entera de datos como argumento y produce un único dato que resume la columna.

SQL ofrece seis funciones de columna diferentes. Son:

1. `SUM()`: calcula el total de una suma.
2. `AVG()`: calcula el valor promedio de una columna.
3. `MIN()`: encuentra el valor mínimo (más chico) de una columna.
4. `MAX()`: encuentra el valor máximo (más grande) de una columna.
5. `COUNT()`: cuenta el número de valores de una columna.
6. `COUNT(*)`: cuenta las filas de una consulta.

El argumento de una función columna puede ser un solo nombre de columna, o puede ser una expresión SQL.

CÁLCULO DEL TOTAL DE UNA COLUMNA (SUM)

La función columna `SUM()` calcula la suma de una columna de valores de datos. Los datos de la columna deben tener un tipo numérico. El resultado de la función `SUM()` tiene el mismo tipo de dato básico que los datos de la columna, pero el resultado puede tener una precisión superior.

Ejemplo:

```
SELECT SUM(Importe)
FROM Pedidos
WHERE Nombre = 'Bill Adams'
```

CÁLCULO DEL PROMEDIO DE UNA COLUMNA (AVG)

La función de columna `AVG()` calcula el promedio de una columna de valores de datos. Los datos de la columna deben tener un tipo numérico. El resultado será un número decimal o un número de coma flotante, dependiendo del producto DBMS concreto que se esté utilizando.

Ejemplo:

```
SELECT AVG(Precio)
FROM Productos
WHERE IdFab = 'ACI'
```

DETERMINACIÓN DE VALORES EXTREMOS (MIN Y MAX)

Las funciones de columna `MIN()` y `MAX()` determinan los valores menor y mayor de una columna, respectivamente. Los datos de la columna pueden contener información numérica, de cadena o de fecha/hora. El resultado de la función `MIN()` y `MAX()` tiene exactamente el mismo tipo de dato que los datos de la columna.

Ejemplo:

```
SELECT MIN(Cuota), MAX(Cuota)
FROM RepVentas
```

Cuando se aplican a datos numéricos, SQL compara los números en orden algebraico. Las fechas se comparan secuencialmente (las fechas más antiguas son más pequeñas que las fechas más recientes). Las duraciones se comparan en base a su longitud (las duraciones más breves son más pequeñas que las duraciones más amplias). Cuando se utilizan en cadenas, la comparación de las cadenas depende del conjunto de caracteres que se esté utilizando (por ejemplo, ASCII).

CUENTA DE VALORES DE DATOS (COUNT)

La función de columna `COUNT()` cuenta el número de valores de datos que hay en una columna. Los datos de la columna pueden ser de cualquier tipo. La función `COUNT()` siempre devuelve un entero, independientemente del tipo de datos de la columna.

Ejemplo:

```
SELECT COUNT(NumCliente)
FROM Clientes
```

SQL permite una función de columna especial `COUNT(*)` que cuenta filas en lugar de valores de datos.

Ejemplo:

```
SELECT COUNT(*)
FROM Pedidos
WHERE Importe > 2500.00
```

REGLAS DE PROCESAMIENTO DE CONSULTAS RESUMEN

PASOS

1. Se forma el producto de las tablas indicadas en la cláusula `FROM` (si hay una sola tabla, se designa esa tabla).
2. Si hay una cláusula `WHERE`, se aplica la condición de búsqueda a cada fila de la tabla producto, reteniendo las que cumplen con la condición de búsqueda.
3. Se extraen las columnas de la lista de selección. Para una columna simple, se utiliza el valor de la columna. Si es una función de columna, se utiliza como argumento el conjunto entero de filas.
4. Si se especifica `SELECT DISTINCT`, se eliminan las filas duplicadas.
5. Si hay `ORDER BY`, se ordenan los resultados.

VALORES NULL Y FUNCIONES DE COLUMNA

El estándar ANSI/ISO especifica reglas precisas para gestionar valores NULL en funciones de columna:

- Si alguno de los valores de datos de una columna es NULL, se ignora para el propósito de calcular el valor de la función de columna.
- Si todos los datos de una columna son NULL, las funciones de columna SUM(), AVG(), MIN() y MAX() devuelven un valor NULL; la función COUNT() devuelve un valor cero.
- Si no hay datos en la columna (columna vacía), las funciones de columna SUM(), AVG(), MIN() Y MAX() devuelven un valor cero.
- La función COUNT(*) cuenta filas, y no depende de la presencia o ausencia de valores NULL, en la columna. Si no hay filas devuelve un valor cero.

ELIMINACIÓN DE FILAS DUPLICADAS (DISTINCT)

Se le puede pedir a SQL que elimine los valores duplicados de una columna antes de aplicarle la función de columna. Para eliminar valores duplicados, la palabra clave DISTINCT se incluye delante del argumento de la función de columna, inmediatamente después del paréntesis abierto.

Ejemplo:

```
SELECT COUNT(DISTINCT Titulo)
FROM RepVentas
```

El argumento debe ser una columna, no puede ser una expresión. No es aplicable a las funciones MIN() y MAX(), aunque algunas implementaciones lo permiten.

La palabra clave DISTINCT sólo puede aparecer una vez en una consulta. Si aparece en el argumento de una función de columna, no puede aparecer en ninguna otra. Si se especifica delante de la lista de selección, no puede aparecer en ninguna función de columna. La única excepción es que DISTINCT puede ser especificada una segunda vez dentro de una subconsulta.

CONSULTAS AGRUPADAS (GROUP BY)

Con frecuencia es conveniente resumir los resultados de la consulta a un nivel “subtotal”. La cláusula GROUP BY de la sentencia SELECT proporciona esta capacidad.

Se denomina consulta agrupada ya que agrupa los datos de las tablas fuentes y produce una única fila resumen por cada grupo de filas. Las columnas indicadas en la cláusula se denominan *columnas de agrupación* de la consulta, ya que son las que determinan como se dividen las filas en grupos.

Hay una estrecha relación entre las funciones de columna SQL y la cláusula GROUP BY. Cuando la cláusula GROUP BY está presente, informa a SQL que debe dividir los resultados detallados en grupos y aplicar la función de columna separadamente a cada grupo, produciendo un único resultado por cada grupo.

SQL también puede agrupar resultados de consulta basándose en contenidos de dos o más columnas.

Ejemplo:

```
SELECT COUNT(DISTINCT NumCli)
FROM Clientes
GROUP BY RepClie
```

```
SELECT Re, Clie, SUM(Importe)
FROM Pedidos
GROUP BY Rep, Clie
```

RESTRICCIONES DE CONSULTAS AGRUPADAS

Las columnas de agrupación deben ser columnas efectivas de las tablas designadas en la cláusula `FROM` de la consulta. No se pueden agrupar las filas basándose en el valor de una expresión calculada.

Todos los elementos de la lista de selección deben tener un único valor por cada grupo de filas. Básicamente, esto significa que un elemento de selección en una consulta agrupada puede ser:

- Una constante.
- Una función de columna.
- Una columna de agrupación.
- Una expresión que afecte a combinaciones anteriores.

En la práctica, una consulta agrupada incluirá siempre una columna de agrupación y una función de columna en su lista de selección. Si no aparece la función de columna, la consulta puede expresarse más sencillamente utilizando `SELECT DISTINCT`, sin `GROUP BY`.

Otra limitación de las consultas agrupadas es que SQL ignora información referente a claves primarias y claves foráneas cuando analiza la validez de una consulta agrupada.

Todas las columnas indicadas en la lista de selección, deben ir en la lista de columnas de agrupación.

VALORES NULL EN COLUMNAS DE AGRUPACIÓN

Se forzará a SQL a colocar cada fila con una columna de agrupación `NULL` en un grupo aparte.

Si dos filas tienen `NULL` en las mismas columnas de agrupación y valores idénticos en las columnas de agrupación no `NULL`, se agrupan dentro del mismo grupo de filas.

REGLAS DE PROCESAMIENTO DE CONSULTAS AGRUPADAS

PASOS

1. Se forma el producto de las tablas indicadas en la cláusula `FROM` (si hay una sola tabla, se designa esa tabla).
2. Si hay una cláusula `WHERE`, se aplica la condición de búsqueda a cada fila de la tabla producto, reteniendo las que cumplen con la condición de búsqueda.
3. Si hay una cláusula `GROUP BY`, disponer las filas restantes de la tabla producto en grupos de filas.
4. Para cada fila (o para cada grupo de filas), se extraen las columnas de la lista de selección. Para una columna simple, se utiliza el valor de la columna. Si es una función de columna, se utiliza como argumento el conjunto entero de filas.
5. Si se especifica `SELECT DISTINCT`, se eliminan las filas duplicadas.
6. Si hay `ORDER BY`, se ordenan los resultados.

CONDICIONES DE BÚSQUEDA DE GRUPOS (CLÁUSULA HAVING)

Al igual que la cláusula `WHERE` puede ser utilizada para filtrar o seleccionar filas, la cláusula `HAVING` hace lo mismo, filtra o selecciona grupos de filas. O sea, que especifica una condición de búsqueda para grupos.

Ejemplo:

```
SELECT Rep, AVG(Importe)
FROM Pedidos
GROUP BY Rep
HAVING SUM(Importe) > 30000.00
```

RESTRICCIONES DE CONDICIONES DE BÚSQUEDA DE GRUPOS

La cláusula `HAVING` se utiliza para incluir o excluir grupos de filas de los resultados de la consulta, por lo que la condición de búsqueda que especifica debe ser aplicable al grupo en su totalidad en lugar de filas individuales.

En la práctica, la condición de búsqueda de cláusula `HAVING` incluirá siempre al menos una función de columna. Si no lo hiciera, la condición de búsqueda podría expresarse con la cláusula `WHERE` y aplicarse a filas individuales.

VALORES NULL Y CONDICIONES DE BÚSQUEDA DE GRUPOS

Al igual que la condición de búsqueda de la cláusula `WHERE`, la cláusula `HAVING` puede producir uno de los resultados siguientes:

- Si la condición de búsqueda es `TRUE`, se retiene el grupo de filas y contribuye con una fila resumen a los resultados de la consulta.
- Si la condición de búsqueda es `FALSE`, el grupo de filas se descarta y no contribuye con una fila resumen a los resultados de la consulta.
- Si la condición de búsqueda es `NULL`, se trata como `FALSE`.

HAVING SIN GROUP BY

La cláusula `HAVING` se utiliza casi siempre junto con la cláusula `GROUP BY`, pero la sintaxis de la sentencia `SELECT` no lo precisa. Si una cláusula `HAVING` aparece sin una cláusula `GROUP BY`, SQL considera al conjunto entero de resultados detallados como un único grupo. El uso de una cláusula `HAVING` sin `GROUP BY` casi nunca se ve en la práctica.

REGLAS DE PROCESAMIENTO DE CONSULTAS AGRUPADAS CON FILTRO

PASOS

1. Se forma el producto de las tablas indicadas en la cláusula `FROM` (si hay una sola tabla, se designa esa tabla).
2. Si hay una cláusula `WHERE`, se aplica la condición de búsqueda a cada fila de la tabla producto, reteniendo las que cumplen con la condición de búsqueda.
3. Si hay una cláusula `GROUP BY`, disponer las filas restantes de la tabla producto en grupos de filas.
4. Si hay una cláusula `HAVING`, se aplica su condición de búsqueda a cada grupo de filas, reteniendo aquellos grupos que cumplen la condición de búsqueda.
5. Para cada fila (o para cada grupo de filas), se extraen las columnas de la lista de selección. Para una columna simple, se utiliza el valor de la columna. Si es una función de columna, se utiliza como argumento el conjunto entero de filas.
6. Si se especifica `SELECT DISTINCT`, se eliminan las filas duplicadas.
7. Si hay `ORDER BY`, se ordenan los resultados.

SUBCONSULTAS

Una *subconsulta* es una consulta que aparece dentro de otra consulta, ya sea dentro de la cláusula `WHERE` o dentro de la cláusula `HAVING`. Las subconsultas proporcionan un modo eficaz y natural de gestionar peticiones de consultas que se expresan en términos de los resultados de otras consultas.

La subconsulta está siempre encerrada entre paréntesis, pero por otra parte tiene el formato familiar de una sentencia `SELECT`, con una cláusula `FROM` y cláusulas opcionales `WHERE`, `GROUP BY` y `HAVING`.

Hay unas cuantas diferencias entre una subconsulta y una sentencia `SELECT` real:

- Una subconsulta debe producir una única columna de datos como resultado.
- La cláusula `ORDER BY` no puede ser especificada en una subconsulta.

- Una subconsulta no puede ser la `UNION` de varias sentencias `SELECT`.
- Los nombres de columna que aparecen en una subconsulta pueden referirse a columna de tablas de la consulta principal. Esto se denomina **referencia externa**.

SUBCONSULTAS EN LA CLÁUSULA `WHERE`

Las subconsultas suelen ser utilizadas principalmente en la cláusula `WHERE` de una sentencia SQL. Aquí la subconsulta funciona como parte del proceso de selección de filas.

Ejemplo:

```
SELECT Ciudad FROM Oficinas
WHERE Objetivo > (SELECT SUM(Cuota) FROM RepVentas
                  WHERE OficinaRep = Oficina)
```

REFERENCIAS EXTERNAS

Dentro del cuerpo de una subconsulta, con frecuencia es necesaria referirse al valor de una columna en la fila “actual” de la consulta principal.

El valor de la columna en una referencia externa se toma de la fila que actualmente está siendo examinada por la consulta principal.

CONDICIONES DE BÚSQUEDA EN SUBCONSULTAS

1. **Test de comparación de consulta:** compara el valor de una expresión con un valor único producido por una subconsulta.
2. **Test de pertenencia a conjunto:** comprueba si el valor de una expresión coincide con uno del conjunto de valores producidos por una subconsulta.
3. **Test de existencia:** examina si una subconsulta produce alguna fila de resultados.
4. **Test de comparación cuantificada:** compara el valor de una expresión con cada uno del conjunto de valores producido por una subconsulta.

TEST DE COMPARACIÓN DE CONSULTA (`=`, `<>`, `<`, `<=`, `>`, `>=`)

Es una forma modificada del test de comparación simple. Compara el valor de una expresión con el valor producido por una subconsulta, y devuelve un resultado `TRUE` si la comparación es cierta.

El test de comparación de subconsulta ofrece los mismos seis operadores de comparación disponibles con el test de comparación simple. La subconsulta especificada en este test debe producir una *única fila* de resultados. Si la subconsulta produce múltiples filas, la comparación no tiene sentido y SQL informa de una condición de error. Si la subconsulta no produce filas o produce un valor `NULL`, el test de comparación devuelve `NULL`.

Ejemplo:

```
SELECT Empresa FROM Clientes
WHERE RepClie = (SELECT NumEmpl FROM RepVentas
                 WHERE Nombre = 'Bill Adams')
```

TEST DE PERTENENCIA A CONJUNTO (`IN`)

El test de pertenencia a conjunto subconsulta (`IN`) es una forma modificada del test de pertenencia a conjunto simple. Compara un único valor de datos con una columna de valores producida por una subconsulta y devuelve un resultado `TRUE` si el valor coincide con uno de los valores de la columna. Se puede invertir la lógica utilizando `NOT IN`.

Ejemplo:

```
SELECT Nombre FROM RepVentas
WHERE OficinaRep IN (SELECT Oficina FROM Oficinas
                     WHERE Ventas > Objetivo)
```

TEST DE EXISTENCIA (EXISTS)

El test de existencia (EXISTS) comprueba si una subconsulta produce alguna fila de resultados. No hay test de comparación simple que se asemeje al test de existencia: solamente se utiliza en subconsultas.

El test EXISTS es devuelto TRUE si existen resultados en la subconsulta, o FALSE si no existen resultados. Se puede invertir la lógica utilizando NOT EXISTS.

Ejemplo:

```
SELECT DISTINCT Descripcion FROM Productos
WHERE EXISTS (SELECT NumPedido FROM Pedidos
              WHERE Producto = IdProducto AND Fab = IdFab AND Importe > 2500)
```

Este test tiene sentido con el uso de referencias externas, sino la subconsulta arrojaría siempre el mismo resultado para todas las filas.

TEST CUANTIFICADOS (ANY Y ALL)

SQL proporciona dos test cuantificados, ANY y ALL. Ambos tests comparan un valor de dato con la columna de valores producidos por una subconsulta.

EL TEST ANY

Se utiliza conjuntamente con uno de los seis operadores de comparación SQL (=, <>, <, <=, >, >=) para comparar un único valor de test con una columna de valores producidos por una subconsulta. Para efectuar este test con *cada* valor de datos de la columna, uno cada vez.

Si **alguna** de las comparaciones individuales produce un resultado TRUE, el test ANY devuelve un resultado TRUE.

Ejemplo:

```
SELECT Nombre
FROM RepVentas
WHERE (.1 * Cuota) < ANY (SELECT Importe FROM Pedidos
                        WHERE Rep = NumEmpl)
```

EL TEST ALL

Es igual que el test ANY. La diferencia es que, si **todas** las comparaciones individuales producen un resultado TRUE, el test ALL devuelve un resultado TRUE.

Ejemplo:

```
SELECT Ciudad, Objetivo
FROM Oficinas
WHERE (0.5 * Objetivo) < ALL (SELECT Ventas FROM RepVentas
                             WHERE OficinaRep = Oficina)
```

SUBCONSULTAS EN LA CLÁUSULA HAVING

También pueden utilizarse subconsultas en la cláusula HAVING de una consulta. Cuando aparece una subconsulta en el HAVING, funciona como parte de selección de grupo de filas efectuada por la cláusula HAVING.

REGLAS PARA PROCESAMIENTO DE SUBCONSULTAS

PASOS

1. Se forma el producto de las tablas indicadas en la cláusula `FROM` (si hay una sola tabla, se designa esa tabla).
2. Si hay una cláusula `WHERE`, se aplica la condición de búsqueda a cada fila de la tabla producto, reteniendo las que cumplen con la condición de búsqueda. Si `WHERE` contiene una subconsulta, la subconsulta se efectúa para cada fila conforme es examinada.
3. Si hay una cláusula `GROUP BY`, disponer las filas restantes de la tabla producto en grupos de filas.
4. Si hay una cláusula `HAVING`, se aplica su condición de búsqueda a cada grupo de filas, reteniendo aquellos grupos que cumplen la condición de búsqueda. Si `HAVING` contiene una subconsulta, la subconsulta se efectúa para cada grupo de filas conforme es examinada.
5. Para cada fila (o para cada grupo de filas), se extraen las columnas de la lista de selección. Para una columna simple, se utiliza el valor de la columna. Si es una función de columna, se utiliza como argumento el conjunto entero de filas.
6. Si se especifica `SELECT DISTINCT`, se eliminan las filas duplicadas.
7. Si hay `ORDER BY`, se ordenan los resultados.

SUBCONSULTAS ANIDADAS

Del mismo modo, se puede escribir subconsultas dentro de las subconsultas, el estándar SQL no especifica un máximo de niveles de anidación, pero en la práctica una consulta consume mucho más tiempo cuando se incrementa el número de niveles.

SUBCONSULTAS CORRELACIONADAS

Una subconsulta que contiene una referencia externa se denomina subconsulta correlacionada, ya que sus resultados están correlacionados con cada fila individual de la consulta principal, llamándose a la referencia, referencia correlacionada.

ACTUALIZACIÓN DE DATOS

SQL es un lenguaje completo de manipulación de datos que se utiliza no solamente para consultas, sino también para modificar y actualizar los datos de la base de datos.

INTRODUCCIÓN DE DATOS EN LA BASE DE DATOS

Una nueva fila de datos se añade a una base de datos relacional cuando una nueva entidad representada por una fila “aparece en el mundo exterior”.

La unidad de datos más pequeña que puede añadirse a una base de datos relacional es una fila. Un DBMS basado en SQL proporciona 3 maneras de añadir nuevas filas de datos a una base de datos:

1. Sentencia `INSERT` de una fila, añade una única nueva fila de datos a una tabla.
2. Sentencia `INSERT` *multifila*, extrae filas de datos de otra parte de la base de datos y las añade a una tabla. Desde otra tabla de la base de datos.
3. Utilidad de *carga masiva*, extrae datos a una tabla desde un archivo externo a la base de datos. Se utiliza para cargar datos inicialmente.

LA SENTENCIA `INSERT` DE UNA FILA

Añade una fila a una tabla. La cláusula `INTO` especifica la tabla que recibe la nueva fila y la cláusula `VALUES` especifica los valores de los datos que contendrá la nueva fila.

Las filas de una tabla no están ordenadas, por lo que no existe noción de insertar la fila “al comienzo” o “al final” o “entre dos filas” de la tabla.

Ejemplo:

```
INSERT INTO Personas (Nombre, Apellido) VALUES ('Bill', 'Adams')
```

- **Inserción de valores NULL:** cuando SQL inserta una nueva fila de datos en una tabla, automáticamente asigna un valor NULL a cualquier columna cuyo nombre falte en la lista de columnas de la sentencia INSERT.
- **Inserción de todas las columnas:** SQL permite omitir la lista de columnas de la sentencia INSERT. Cuando se omite la lista de columnas, SQL genera automáticamente una lista formada por todas las columnas de la tabla, en secuencia de izquierda a derecha. Esta es la misma secuencia de columnas generadas por SQL cuando se utiliza una consulta SELECT *. Cuando se omite la lista de columnas, la palabra clave NULL, debe ser utilizada en la lista de valores para asignar explícitamente valores NULL a las columnas.
- **La sentencias INSERT multifila:** la segunda forma de la sentencia INSERT añade múltiples filas de datos a su tabla destino. En esta forma de la sentencia, los valores de datos para las nuevas filas no son especificados explícitamente dentro del texto de la sentencia. En su lugar, la fuente de las nuevas filas es una consulta de bases de datos especificada en la sentencia.

Restricciones sobre la consulta dentro de la sentencia INSERT:

- La consulta no puede tener un ORDER BY.
- El resultado de la consulta debe contener el mismo número de columnas que hay en la lista de columnas del INSERT.
- La consulta no puede ser la UNION de varias sentencias SELECT.
- La tabla destino de la sentencia INSERT no puede aparecer en el FROM de la consulta o de ninguna subconsulta que ésta contenga.

Ejemplo:

```
INSERT INTO Personas (Nombre, Apellido)
SELECT NombreCli, ApellidoCli FROM Clientes
WHERE Categoria > 5
```

SUPRESIÓN DE DATOS DE LA BASE DE DATOS

Una fila de datos se suprime de una base de datos cuando la entidad representada por la fila “desaparece del mundo exterior”. La fila se suprime para mantener la base de datos como un modelo preciso del mundo real.

LA SENTENCIA DELETE

Esta sentencia elimina filas seleccionadas de datos de una única tabla. La cláusula FROM especifica la tabla destino que contiene las filas. La cláusula WHERE especifica que filas de la tabla van a ser suprimidas. La condición de búsqueda que pueden especificarse en la cláusula WHERE son las mismas que hay disponibles en la cláusula WHERE de la sentencia SELECT.

Ejemplo:

```
DELETE FROM Producto
WHERE Precio < 20.5
```

SUPRESIÓN DE TODAS LAS FILAS

La cláusula WHERE en la sentencia DELETE es opcional, si esta se omite, DELETE elimina todas las filas de la tabla.

DELETE CON SUBCONSULTA

Las subconsultas pueden jugar un papel importante en la sentencia `DELETE`, ya que permiten suprimir filas en base a información contenida en otras tablas. Las referencias externas aparecerán con frecuencia en las subconsultas de una sentencia `DELETE`, ya que implementarán la composición entre la/s tabla/s de la subconsulta y la tabla destino de la sentencia `DELETE`. La única restricción al uso de subconsultas en una sentencia `DELETE` es que la tabla destino no puede aparecer en la cláusula `FROM` de la subconsulta.

MODIFICACIÓN DE DATOS EN LA BASE DE DATOS

Los valores almacenados en una base de datos se modifican cuando se producen cambios correspondientes en el mundo exterior.

LA SENTENCIA UPDATE

Esta sentencia modifica los valores de una o más columnas en las filas seleccionadas de una tabla única. La tabla destino a actualizar se indica en la sentencia y es necesario disponer de permiso para actualizar la tabla así como cada una de las columnas individuales que serán modificadas. La cláusula `WHERE` selecciona las filas de la tabla a modificar. La cláusula `SET` especifica que columnas se van a actualizar y calcula los nuevos valores.

Ejemplo:

```
UPDATE Personas
SET Nombre = 'Bruce', Apellido = 'Wayne'
WHERE IdPersona = 34;
```

```
UPDATE Productos
SET Precio = Precio - Precio * 0.25
WHERE YEAR(FechaAlta) = 2010
```

Las condiciones de búsqueda que pueden aparecer en la cláusula `WHERE` de una sentencia `UPDATE` son exactamente las mismas que las disponibles en las sentencias `SELECT` y `DELETE`.

Al igual que la sentencia `DELETE`, la sentencia `UPDATE` puede actualizar varias filas de una vez con la condición de búsqueda adecuada.

La cláusula `SET` es una lista de asignaciones separadas por comas. Cada asignación identifica una columna destino. Cada columna destino debería aparecer solamente una vez en la lista.

La expresión en cada asignación puede ser cualquier expresión SQL válida que genere un valor del tipo de dato apropiado para la columna destino. La expresión se debe poder calcular con los valores de la fila que actualmente está en actualización en la tabla destino. No pueden incluirse funciones de columna ni subconsultas.

ACTUALIZACIÓN DE TODAS LAS FILAS

La cláusula `WHERE` es opcional. Si se omite la cláusula, efectúa una actualización masiva en toda la tabla.

UPDATE CON SUBCONSULTA

Al igual que `DELETE`, las subconsultas pueden jugar un papel importante en la sentencia `UPDATE`. Ya que permite seleccionar las filas a actualizar en base a información contenida en otras tablas.

Al igual que en `DELETE`, pueden anidarse a cualquier nivel y pueden contener referencias externas a la tabla destino de la sentencia.

INTEGRIDAD DE DATOS

El término *integridad de datos* se refiere a la corrección y completitud de los datos en una base de datos. La integridad de los datos almacenados puede perderse de muchas formas diferentes. Por ejemplo:

- Pueden añadirse datos no válidos a la base de datos.
- Pueden modificarse datos existentes tomando un valor incorrecto.
- Los cambios en la base de datos pueden perderse debido a un error del sistema o fallo en el suministro de energía.
- Los cambios pueden ser aplicados parcialmente.

QUE ES LA INTEGRIDAD DE DATOS

Para preservar la consistencia y la corrección de los datos almacenados, un DBMS relacional impone típicamente una o más *restricciones de integridad de datos*. Estas restricciones restringen los valores que pueden ser insertados en la base de datos o creados mediante una actualización de la base de datos. Varios tipos diferentes de restricciones de integridad:

1. **Datos requeridos:** algunas columnas en una base de datos deben contener un valor de dato válido en cada fila; no se permite la ausencia de valor o que contengan valores `NULL`.
2. **Chequeo de validez:** cada columna de una base de datos tiene un *domino*, un conjunto de valores que son legales para esa columna.
3. **Integridad de entidad:** la clave primaria de una tabla debe contener un valor único en cada fila diferente de los valores de las filas restantes.
4. **Integridad referencial:** una clave foránea en una BD relacional enlaza cada fila de la tabla hija que contiene la clave ajena con la fila de la tabla padre que contiene el valor de clave primaria correspondiente.
5. **Reglas comerciales:** las actualizaciones de una base de datos pueden estar restringidas por reglas comerciales que gobiernan las transacciones en el mundo real que están representadas por las actualizaciones.
6. **Consistencia:** muchas transacciones del mundo real producen múltiples actualizaciones a una base de datos. Hay que mantener la BD en un estado correcto y consistente.

DATOS REQUERIDOS

La restricción de integridad de datos más simple requiere que una columna tenga un valor no `NULL`. Esta restricción `NOT NULL` se especifica como parte de la sentencia `CREATE TABLE`.

La sentencia `INSERT`, debe especificar un valor de datos no nulo para la columna. Un intento de hacerlo da lugar a un error. La sentencia `UPDATE`, al actualizar un dato, debe ser por uno no nulo, o dará un error.

COMPROBACIÓN DE VALIDEZ

Cada columna al crearse la tabla, tiene asignado un tipo de dato, el DBMS asegura que únicamente datos del tipo especificado sean introducidos en la columna. El procedimiento de validación comprueba los datos e indica mediante un valor de retorno si los datos son aceptables.

INTEGRIDAD DE ENTIDAD

La clave primaria de una tabla debe tener un valor único para cada fila de la tabla o si no la base de datos perderá su integridad como modelo del mundo exterior. Por ello, la razón de la exigencia de que las claves primarias tengan valores únicos se denomina *restricción de integridad entidad*. El DBMS comprueba automáticamente la unicidad del valor de la clave primaria con cada sentencia `INSERT` y `UPDATE`. Un intento de insertar una fila con una clave duplicada, falla.

- **Otras restricciones de unidad:** a veces es apropiado exigir que una columna que no es clave primaria de una tabla contenga un valor único en cada fila. El DBMS fuerza una restricción de unicidad del mismo modo que la fuerza la restricción de clave primaria. El estándar utiliza la sentencia `CREATE TABLE` para especificar restricciones de unicidad en columnas o combinaciones de columnas.
- **Unicidad y Valores NULL:** los valores `NULL` presenta un problema cuando aparecen en la clave primaria de una tabla o en una columna que esta especificada en una restricción de unicidad. Por lo tanto SQL requiere que toda columna que forma parte de una clave primaria y toda columna designada en una restricción de unicidad deben ser declaradas `NOT NULL`.

INTEGRIDAD REFERENCIAL

Asegura la integridad de las relaciones padre/hijo creadas mediante claves ajenas y claves primarias.

PROBLEMAS DE INTEGRIDAD REFERENCIAL

1. **La inserción de una nueva fila hijo:** cuando se inserta una nueva fila en la tabla hijo, su valor clave ajena debe coincidir con uno de los valores de clave primaria en la tabla padre.
2. **La actualización de la clave ajena en una fila hijo:** si la clave ajena se modifica mediante una secuencia de `UPDATES`, el nuevo valor debe coincidir con un valor de clave primaria en la tabla padre.
3. **La supresión de una fila padre:** si la tabla padre que tiene uno o más hijos se suprime, las filas hijo quedaran sin padre. Los valores de clave ajena en estas filas ya no se corresponderán con ningún valor de clave primaria en la tabla padre. Para resolver esto es más complejo, dependiendo de la situación se podrá:
 - a. Impedir la supresión hasta reasignar los hijos.
 - b. Suprimir las filas hijos automáticamente.
 - c. Seteando un valor `NULL` a todos los hijos.
 - d. Seteando un valor por defecto.
4. **La actualización de la clave primaria:** esta es una forma diferente del problema anterior. Si la clave primaria de una fila en la tabla padre se modifica, todos los hijos actuales de esa fila quedan huérfanos, puesto que sus claves ajenas ya no corresponden con ningún valor de la clave primaria. Se controla también por prohibición, antes de que la PK pueda ser modificada, el DBMS efectúa comprobaciones para asegurarse de que no haya filas hijo que tengan valores de clave ajena correspondientes. Dependiendo de la situación:
 - a. Impedir la supresión hasta reasignar los hijos.
 - b. Suprimir las filas hijos automáticamente.
 - c. Seteando un valor `NULL` a todos los hijos.
 - d. Seteando un valor por defecto.

REGLAS DE SUPRESIÓN

Por cada relación padre hijo, se puede especificar una regla de supresión asociada y una regla de actualización asociada. La regla de supresión le dice al DBMS que debe hacer cuando un usuario trate de suprimir una fila de la tabla padre.

- La Regla **RESTRICT** impide suprimir una fila de la tabla padre si la fila tiene algún hijo.
- La regla **CASCADE** le dice al DBMS que cuando una fila padre se suprima, todas sus filas hijo también deberían ser suprimidas automáticamente de la tabla hijo.
- La regla de supresión **SET NULL**, le dice al DBMS que cuando una fila padre sea suprimida, los valores de la clave ajena en todas las filas hijo deben automáticamente pasarse a `NULL`.
- La regla de supresión **SET DEFAULT** le dice al DBMS que cuando una fila padre se suprimida, los valores de clave ajena en todas las filas hijo deben automáticamente pasarse a un valor por defecto para esa columna particular.

REGLAS DE ACTUALIZACIÓN

La regla de actualización le dice al DBMS que hacer cuando un usuario intenta actualizar el valor de una de las columnas de clave primaria en la tabla padre.

- La Regla **RESTRICT** impide actualizar la clave primaria de una fila de la tabla padre si la fila tiene algún hijo.
- La regla **CASCADE** le dice al DBMS que cuando el valor de una clave primaria de una fila padre se actualiza, se modifican las claves ajenas en todas sus filas hijo también deberían ser actualizadas automáticamente de la tabla hijo.
- La regla de actualización **SET NULL**, le dice al DBMS que cuando el valor de la clave primaria de una fila padre sea modificada, los valores de la clave ajena en todas las filas hijo deben automáticamente pasarse a **NULL**.
- La regla de actualización **SET DEFAULT** le dice al DBMS que cuando un valor de la clave primaria de una fila padre se modificada, los valores de clave ajena en todas las filas hijo deben automáticamente pasarse a un valor por defecto para esa columna particular.

SUPRESIONES Y ACTUALIZACIONES EN CASCADA

Las reglas de supresión **CASCADE** deben ser especificadas con cuidado, ya que pueden provocar la supresión automática global de datos si fueran utilizadas incorrectamente.

La reglas de supresión **SET NULL** es una regla de dos niveles; su impacto se detiene en la tabla hijo.

CICLOS REFERENCIALES

Cuando una tabla apunta a otra, y ésta otra apunta a la primera (como un bucle). Esto provoca problemas en las restricciones de integridad referencial.

Problema en la inserción si las FK son **NOT NULL**. Se quiere insertar algo en las dos tablas y no se puede. La inserción primera hace referencia a una fila que no existe en la otra tabla (que es la de la segunda inserción que no llega a ejecutarse). Para impedir este “interbloqueo” al menos una de las claves foráneas en un ciclo referencial debe permitir valores **NULL**.

Los ciclos referenciales también restringen las reglas de supresión y actualización que pueden especificarse para las relaciones que forman el ciclo.

Este ciclo puede darse con más tablas, una apunta a una segunda, la segunda a una tercera, y la tercera a la primera. Al menos una relación del ciclo debe tener una regla de supresión **RESTRICT** o **SET NULL** para romper el ciclo de supresiones en cascada.

REGLAS COMERCIALES

Muchas de las cuestiones de integridad de datos en el mundo real tienen que ver con las reglas y procedimientos de una organización. Estas reglas caen fuera del ámbito de SQL. Forzar reglas comerciales es problema de los programas de aplicación que acceden a la BD.

Dejar las reglas comerciales a los programas de aplicación tiene desventajas:

- **Duplicación de esfuerzo:** se deben incluir las reglas comerciales en todas las operaciones con la BD.
- **Falta de consistencia:** muchos programas pueden forzar de distintas formas las reglas comerciales en una BD.
- **Problemas de mantenimiento:** si las reglas comerciales cambian, hay que identificar todos los lugares donde se fuerzan las reglas comerciales para actualizarlas.
- **Complejidad:** hay muchas reglas que recordar.

TRIGGERS (DISPARADORES)

Para cualquier evento que provoca un cambio en un contenido de una tabla, un usuario puede especificar una acción asociada que el DBMS debería efectuar. Los tres eventos que pueden disparar un disparador son intentos de INSERT, UPDATE o DELETE.

La **ventaja** es que las reglas comerciales pueden almacenarse en la base de datos. Las **desventajas** son:

- **Complejidad en la base de datos:** preparar la BD pasa a ser una tarea compleja.
- **Reglas ocultas:** programas que parecen efectuar sencillas actualizaciones de la BD, pueden de hecho, generar una cantidad enorme de actividad en la BD. El programador ya no controla totalmente lo que sucede en la BD.

STORED PROCEDURES (PROCEDIMIENTOS ALMACENADOS)

Son una colección de sentencias SQL que se almacenan en la base de datos. Aceptan parámetros de entrada y pueden devolver varios valores.

PROCESAMIENTO DE TRANSACCIONES

CONCEPTO

Transacción: secuencia de sentencias SQL que deben ejecutarse como unidad para asegurar la consistencia. Ejemplo: añadir un pedido puede incluir la actualización de stock, insertar pedido, insertar detalles dl pedido, y otros cálculos más.

Una **transacción** es una secuencia de una o más sentencias SQL que juntas forman una unidad de trabajo. Cada sentencia de una transacción efectúa una parte de una tarea, pero todas ellas son necesarias para completar la tarea. La agrupación de las sentencias en una sola transacción indica al DBMS que la secuencia de sentencias entera debe ser ejecutada atómicamente, todas las sentencias deben completarse para que la base de datos esté en un estado consistente.

*Las sentencias de una transacción se ejecutaran como una **unidad atómica** de trabajo en la base de datos. **O todas las sentencias son ejecutadas con éxito, o ninguna de las sentencias es ejecutada.***

El DBMS es responsable de mantener este compromiso incluso si el programa de aplicación aborta o se produce un fallo de hardware a mitad de la transacción. En cada caso, el DBMS debe asegurarse que cuando se complete una recuperación del fallo, la base de datos nunca refleje una “transacción parcial”.

PROPIEDADES DE LAS TRANSACCIONES

- **Atomicidad:** una transacción debe ser una unidad atómica de trabajo. Se realizan todas las tareas o no se realiza ninguna.
- **Coherencia:** cuando finaliza, una transacción debe dejar todos los datos en un estado coherente. En una base de datos relacional, se deben aplicar todas las reglas a las modificaciones de la transacción para mantener la integridad de todos los datos. Todas las estructuras internas de datos, como índices de árbol B o listas doblemente vinculadas, deben estar correctas al final de la transacción.
- **Aislamiento:** las modificaciones realizadas por transacciones simultáneas se deben aislar de las modificaciones llevadas a cabo por otras transacciones simultáneas. Una transacción reconoce los datos en el estado en que estaban antes de que otra transacción simultánea los modificara o después de que la segunda transacción haya concluido, pero no reconoce un estado intermedio. Esto se conoce como seriabilidad, ya que deriva en la capacidad de volver a cargar los datos iniciales y reproducir una serie de transacciones para finalizar con los datos en el mismo estado en que estaban después de realizar las transacciones originales.
- **Durabilidad:** una vez concluida una transacción, sus efectos son permanentes en el sistema. Las modificaciones persisten aún en el caso de producirse un error del sistema.

COMMIT Y ROLLBACK

SQL soporta las transacciones de base de datos mediante dos sentencias de procesamiento de transacciones SQL.

- **COMMIT:** señala el final correcto de una transacción. Informa al DBMS que la transacción está ahora completa, todas las sentencias que forman la transacción han sido ejecutadas y la base de datos es autoconsistente.
- **ROLLBACK:** señala el final sin éxito de una transacción. Informa al DBMS que el usuario no desea completar la transacción, en vez de ello, el DBMS debe deshacer los cambios efectuados a la base de datos durante la transacción. El DBMS restaura la base de datos a su estado antes de que la transacción comenzara.

EL MODELO DE TRANSACCIÓN ANSI/ISO

El estándar SQL ANSI/ISO define un modelo de transacción SQL y los papeles de las sentencias `COMMIT` y `ROLLBACK`. El estándar especifica que una transacción SQL comienza automáticamente con la primera sentencia SQL ejecutada, por usuario o un por un programa. La transacción continúa con las sentencias SQL subsiguientes hasta que finaliza de uno de cuatro modos posibles:

1. Sentencia `COMMIT`, finaliza la transacción con éxito. Una nueva transacción comienza inmediatamente después de la sentencia `COMMIT`.
2. Sentencia `ROLLBACK`, aborta la transacción deshaciendo las modificaciones que haya efectuado a la base de datos. Una nueva transacción comienza inmediatamente después de una sentencia `ROLLBACK`.
3. Terminación de un programa con éxito (para SQL programado) también finaliza la transacción con éxito, igual como si hubiera ejecutado una sentencia `COMMIT`. Puesto que el programa está finalizado, no hay ninguna nueva transacción que comenzar
4. Terminación anormal del programa (para SQL programado), también aborta la transacción, del mismo modo que si se hubiera ejecutado una sentencia `ROLLBACK`. Puesto que el programa está finalizado, no hay ninguna nueva transacción que comenzar.

OTROS MODELOS DE TRANSACCIONES

- La sentencia `BEGIN TRANSACTION` señala el comienzo de una transacción.
- La sentencia `COMMIT TRANSACTION` señala el final con éxito de una transacción.
- La sentencia `SAVE TRANSACTION` establece un *punto de guarda* a mitad de una transacción.
- La sentencia `ROLLBACK TRANSACTION` tiene dos papeles:
 - Si se designa un punto de guarda en la sentencia `ROLLBACK`, deshace los cambios de la BD efectuados desde el punto de guarda.
 - Si no hay punto de guarda designado, la sentencia `ROLLBACK` deshace todos los cambios efectuados desde la sentencia `BEGIN TRANSACTION`.

Esto lo usa Sybase y SQL Server.

LOG DE TRANSACCIONES

Cuando un usuario ejecuta una sentencia SQL que modifica la base de datos, el DBMS escribe automáticamente una anotación en el registro de transacción mostrando dos copias de cada fila afectada por la sentencia. Una copia muestra la fila antes del cambio y la otra copia muestra la fila después del cambio. Sólo después de que el DBMS realmente escribe el registro modifica la fila en disco. Si el usuario ejecuta posteriormente una sentencia `COMMIT`, el fin de transacción de anota en el registro de transacción. Si el usuario ejecuta un `ROLLBACK`, el DBMS examina el registro para encontrar las imágenes “de antes” de las filas que han sido modificadas desde que comenzó la transacción. Utilizando estas imágenes, el DBMS restaura las filas a su estado anterior, deshaciendo efectivamente todas las modificaciones a la BD efectuadas durante la transacción.

TRANSACCIONES Y PROCESAMIENTO MULTIUSUARIO

(pág. 277) Cuando dos o más usuarios acceden concurrentemente a una base de datos, el procesamiento de transacciones toma una nueva dimensión. Ahora el DBMS no solamente debe recuperarse adecuadamente de los fallos o errores del sistema, también debe asegurarse que las acciones de los usuarios no interfieran unas con otras. Idealmente, cada usuario debería ser capaz de acceder a la base de datos como si tuviera acceso exclusivo a ella, sin preocuparse de las acciones del resto de los usuarios. El modelo de transacción SQL permite a un DBMS basada en SQL aislar a los usuarios unos de otros de este modo.

El modelo de transacción SQL permite a un DBMS basado en SQL aislar a los usuarios unos de otros de este modo.

- **Problema de la actualización perdida:** se da cuando dos programas leen los mismos datos y utilizan los datos como base para un cálculo y luego tratan de actualizar los datos.

Ejemplo: se reciben dos pedidos (pedido1 y pedido2) de un producto “al mismo tiempo”, los dos programas leen un determinado stock del producto (stock inicial). El pedido1 se confirma y actualiza el stock (stock inicial – cantidad pedido1), pero luego se confirma el pedido2, pero toma como stock actual del producto el stock inicial (stock inicial – cantidad pedido2) y actualiza el stock. Esto hace que se pierda el dato de la primera transacción (pedido1) y deja inconsistente la base de datos. Dice que hay un determinado stock, cuando la existencia real es menor.

- **Problema de los datos no confirmados:** se da cuando ocurre una actualización y la transacción es abortada.

Ejemplo: un cliente viene a comprar un producto1 con un stock inicial y realiza un pedido por una determinada cantidad realizando stock inicial – cantidad pedido1. Luego viene otro cliente y ve que no hay stock, o no hay la cantidad que necesita/quiere del producto1 entonces rechaza el pedido. Pero después, el cliente del pedido1 se arrepiente y decide cambiar y llevar otro producto volviendo al stock inicial mediante un `ROLLBACK`. El problema está en que el segundo cliente no realizó el pedido porque el stock que figuraba en la base de datos no era el real, ya que el pedido1 no estaba confirmado. Se podrían haber tomado decisiones erróneas como reponer stock o que el cliente2 hubiera comprado el stock indicado (que no es real), y luego en el pedido del cliente1 se hiciera un `ROLLBACK` volviendo al stock inicial sin considerar los cambios del pedido del cliente2.

- **Problema de los datos inconsistentes:** se da con la consulta de datos con actualización confirmada posterior con otro usuario.

Ejemplo: viene un cliente1 a pedir un producto1 con un stock inicial. Después viene un cliente2 a pedir el producto1 y ve el stock inicial. El cliente2 pide ver otro producto (producto2). Mientras tanto el cliente1 decide comprar los productos, resultando (stock inicial – cantidad pedido cliente1). Ahora el cliente2, después de considerar el producto2, se decide a comprar el producto1. Al hacer una nueva consulta sobre el producto1 muestra que hay menos stock que cuando consulto la primera vez (debido al pedido del cliente1). Aquí, la base de datos ha sido un modelo exacto de la situación del mundo real. Esto es malo en el sentido de que para los usuarios, la BD no permaneció consistente durante su transición. Los valores cambiaron. Es un problema si nunca se hubiera vuelto a hacer una consulta sobre el producto1 durante el pedido del cliente2, o si se están realizando cálculos de estadísticas. Los datos no reflejan una visión estable y consistente de la base de datos.

- **Problema de la inserción fantasma:** se da con una consulta de BD y posterior inserción que afecta sobre la vista anterior.

Ejemplo: suponiendo que el gerente de ventas está recorriendo un informe de los pedidos del clienteX y calcula el total. Mientras tanto, viene el clienteX y realiza un pedido de \$5.000, se

inserta en la base de datos y se confirma la transacción. Unos segundos más tarde, el gerente de ventas vuelve a revisar los pedidos, hay un pedido nuevo y el total es \$5.000 mayor que el de la primera consulta. Aquí el problema son los datos inconsistentes. La BD permanece exacta al mundo real y su integridad está intacta, pero la misma consulta realizada dos veces durante la misma transacción produce dos resultados distintos. Las consecuencias de este problema son los cálculos inconsistentes e incorrectos.

TRANSACCIONES CONCURRENTES

Durante una transacción, el usuario verá una vista completamente consistente de la base de datos. El usuario nunca contemplará modificaciones no confirmadas de otros usuarios, e incluso los cambios confirmados efectuados por otros no afectarán a los datos examinados por el usuario en mitad de una transacción.

Si dos transacciones, A y B, se están ejecutando concurrentemente, el DBMS asegura que los resultados serán los mismos en cualquier caso tanto si:

- la transacción A se ejecutara primero, seguida de la Transacción B, como si
- la transacción B se ejecutara primero, seguida de la transacción A.

Este concepto es conocido como **serialidad** de las transacciones. Efectivamente, significa que cada usuario de base de datos puede acceder a la base de datos como si no hubiera otros usuarios accediendo concurrentemente en ella.

Las transacciones deberían ser siempre las más breves posibles. “Use `COMMIT` pronto y `COMMIT` a menudo” es un buen consejo cuando se está utilizando SQL programado.

Todos los productos SQL comerciales utilizan la técnica basada en **cerramiento (locking)**. El cerramiento lo maneja automáticamente el DBMS y es invisible al usuario de SQL.

CERRAMIENTO (LOCKING)

La técnica del cerramiento proporciona a una transacción acceso temporal exclusivo a una parte de una base de datos, impidiendo que otras transacciones modifiquen los datos encerrados. El cerramiento resuelve por tanto los tres problemas de transacción concurrente. Impide que las actualizaciones perdidas, los datos no confirmados, y los datos inconsistentes puedan corromper la base de datos. Sin embargo, el cerramiento introduce un nuevo problema, pudiendo hacer que una transacción espere mucho tiempo mientras las partes de la base de datos que desea acceder están bloqueados.

El DBMS bloquea la parte de la BD accedida por una transacción y hace esperar a las demás (si acceden a los mismos recursos de la BD que la primera transacción) hasta que la primera incluya un `COMMIT` o un `ROLLBACK`.

NIVELES DE CERRAMIENTO

1. **Nivel de las BD:** en su forma menos elaborada el DBMS puede bloquear la base entera por cada transacción. Fácil de implementar, pero solo permite un solo proceso a la vez.
2. **Nivel de tablas:** el DBMS bloquea aquellas tablas accedidas por una transacción.
3. **Nivel de página:** el DBMS bloquea los grupos individuales de datos procedentes del disco, conforme son accedidos por una transacción.
4. **Nivel de fila:** permite que dos transacciones diferentes y concurrentes accedan a dos filas diferentes en una misma tabla, incluso si están en el mismo bloque del disco. Es un problema para tablas pequeñas.
5. **Nivel de datos:** el cerramiento a nivel de datos individuales es una teoría, donde proporcionaría más paralelismo que los cerramientos a nivel de fila. Distintas transacciones pueden acceder a la misma fila siempre y cuando accedan a distintas columnas. Sería muy pesado. Ningún DBMS comercial lo utiliza.

TIPOS DE CERRAMIENTO

- **Cierre compartido:** se utiliza en el DBMS cuando una transacción desea leer datos de la base de datos (consultas).
- **Cierre exclusivo:** se utiliza en el DBMS cuando una transacción desea actualizar datos en la base de datos (actualizaciones).

INTERBLOQUEOS (DEADLOCKS)

Ejemplo: el programa A actualiza la tabla Pedidos, bloqueando parte de ella. Un programa B, actualiza la tabla Productos, bloqueando parte de ella. Ahora el programa A intenta actualizar la tabla Productos y el programa B trata de actualizar la tabla Pedidos. Cada programa espera eternamente la liberación del recurso.

Para tratar los interbloqueos, un DBMS normalmente incluye una lógica que periódicamente comprueba los cierres mantenidos por varias transacciones. Cuando detecte un interbloqueo, el DBMS elige arbitrariamente una de las transacciones como perdedora del interbloqueo y le da marcha atrás. Esto libre los cierres mantenidos por la transacción perdedora, permitiendo a la otra transacción proseguir.

La probabilidad de interbloqueo puede reducirse enormemente con un planeamiento cuidadoso de las actualizaciones de la base de datos.

PARÁMETROS DE CERRAMIENTO

Son fijados por el DBA para mejorar las prestaciones de estos sistemas fijando manualmente parámetros de cerramiento.

1. **Tamaño de cierre:** algunos productos DBMS ofrecen la opción de cierres a nivel de tabla, página, fila, y otros. Dependiendo la aplicación específica puede ser adecuado un tamaño de cierre diferente.
2. **Número de cierre:** un DBMS permite para cada transacción tenga un cierto número finito de cierres. El administrador de la base de datos puede frecuentemente fijar ese límite, elevándolo para permitir transacciones más complejas, o rebajándolo para estimular escalas de cierres previas.
3. **Estaca de cierre:** con frecuencia un DBMS escalará automáticamente los cierres, reemplazando muchos cierres pequeños con un único cierre mayor.
4. **Plazo de cierre:** aun cuando la transacción no se interbloquee con otra, puede tener que esperar mucho tiempo a que otras transacciones liberen esos cierres. Se establece un tiempo límite (*time out*), que lanza un error si lo alcanza.

ESTRUCTURA DE LA BASE DE DATOS

Los cambios de la estructura de la base de datos se realizan con un conjunto diferente de sentencias SQL, denominadas conjuntamente **DDL**, lenguaje de definición de datos.

Las sentencias **DML** pueden modificar los datos almacenados, pero no cambian su estructura.

Un lenguaje DDL puede:

- Definir y crear nuevas tablas.
- Suprimir una tabla que ya no es necesaria.
- Cambiar la definición de una tabla existente.
- Definir una tabla virtual de datos.
- Establecer controles de seguridad para una base de datos.
- Construir un índice para hacer más rápido el acceso a la tabla.
- Controlar el almacenamiento físico de los datos por parte del DBMS.

El núcleo de DDL está basado en tres verbos SQL:

- **CREATE** define y crea un objeto en la base de datos.
- **DROP** elimina un objeto de la base de datos.
- **ALTER** modifica la definición de un objeto de la base de datos.

CREACIÓN DE UNA BASE DE DATOS

El estándar SQL1 especifica el lenguaje SQL utilizado para describir la estructura de una BD, pero no especifica cómo se crean, y cada producto DBMS adopta un planeamiento diferente.

Ejemplo (SQL Server): `CREATE DATABASE NombreBD`

DEFINICIONES DE TABLAS

La creación de tablas se hace mediante la sentencia `CREATE TABLE`.

DEFINICIONES DE COLUMNAS

- **Nombre de columna:** nombre de la columna en la sentencia SQL. Debe ser un nombre único para esta tabla (puede estar repetido en otras tablas).
- **Tipo de dato:** el tipo de dato que la columna almacena.
- **Dato requerido:** indicar si acepta nulos o no. Mediante `NOT NULL`.
- **Valor por omisión:** es opcional. Es un valor que se utiliza cuando hay un `INSERT` y no se especifica el valor de la columna.

Ejemplo:

```
CREATE TABLE Personas (  
    Nombre VARCHAR(20) NOT NULL,  
    Apellido VARCHAR(20) NOT NULL,  
    FechaNacimiento DATE  
)
```

ELIMINACIÓN DE TABLAS

Mediante la sentencia `DROP TABLE`. Ejemplo: `DROP TABLE Personas`

MODIFICACIÓN DE TABLA

Mediante la sentencia `ALTER TABLE`.

- `ADD` definición de la columna
- `ALTER` nombre de la columna `SET DEFAULT` valor
- `ALTER` nombre de la columna `DROP DEFAULT`
- `DROP` nombre de la columna `CASCADE / RESTRICT`
- `ADD` definición de la clave primaria
- `ADD` definición de la clave ajena
- `ADD` restricción de unicidad
- `ADD` restricción comprobación
- `DROP CONSTRAINT` nombre de la restricción `CASCADE / RESTRICT`

Ejemplo:

```
ALTER TABLE Personas  
    ADD Direccion VARCHAR(30) NOT NULL
```

DEFINICIONES DE RESTRICCIÓN (*CONSTRAINTS*)

1. **Restricción de unicidad:** obliga que los datos de una columna o combinación de columnas sean únicos en cada una de las filas de la tabla.
2. **Restricción de clave primaria:** al igual que el de unicidad, pero también designa una columna o combinación de columnas como la clave de la fila, en el entorno de una relación padre/hija con otra tabla.
3. **Restricción de clave foránea:** establece que el valor de una columna o combinación de columnas coincida con el valor de una clave primaria en alguna tabla padre.

Cada una de estas restricciones se aplica a una tabla sencilla, que se especifica en la sentencia `CREATE TABLE` y que puede ser modificada con la sentencia `ALTER TABLE`.

RESTRICCIONES DE COMPROBACIÓN

Es una restricción de la base de datos que limita el contenido de una tabla particular. El estándar SQL2, se especifica como una condición de búsqueda y aparece como parte de la definición de la tabla. Palabra clave `CHECK`.

ASERCIONES

Una aserción es una restricción de la base de datos que restringe los contenidos de la base de datos en su conjunto. Se especifica como una condición de búsqueda. Pero a diferencia de la restricción de comprobación, la condición de búsqueda en una aserción puede restringir el contenido de múltiples tablas y los datos de las relaciones entre ellas. Una aserción se especifica como parte de la definición de la base de datos completa, utilizando la sentencia `CREATE ASSERTION` de SQL2.

DEFINICIONES DE DOMINIO

Un dominio de una columna es un conjunto nominado de valores de datos que realmente funciona como un tipo de dato adicional, para utilizarlo en la definición de la base de datos. Se crea con la sentencia `CREATE DOMAIN`.

ÍNDICES

Un índice es una estructura que proporciona un acceso rápido a las filas de una tabla en base a los valores de una o más columnas. El DBMS utiliza el índice al igual que el de un libro. La indexación es apropiada cuando las consultas de una tabla son más frecuentes que las inserciones y las actualizaciones. El DBMS siempre establece un índice para la clave primaria de una tabla.

Los índices nos ayudan a obtener datos de las tablas en forma más rápida. Sin un índice, el sistema de base de datos lee a través de toda la tabla (este proceso se denomina “escaneo de tabla”) para localizar la información deseada. Con el índice correcto en su lugar, el sistema de base de datos puede entonces primero dirigirse al índice para encontrar de dónde obtener los datos, y luego dirigirse a dichas ubicaciones para obtener los datos necesarios.

Un índice puede cubrir una o más columnas. La sintaxis general para la creación de un índice es:

Ejemplo:

```
CRATE UNIQUE INDEX PED_PROD_IDX [nombre del índice]
ON Pedidos (Fab, Producto) [tabla (columna1, columna2,...)]
```

VISTAS

Una vista es una tabla virtual en la base de datos cuyos contenidos están definidos por una consulta. Una vista no existe en una base de datos como un conjunto almacenado de valores, las filas y columnas de datos visibles a través de la vista son los resultados producidos por la consulta que define la vista.

Cuando el DBMS encuentra una referencia a una vista en una sentencia SQL determina la definición de la vista almacenada en la base de datos. Luego el DBMS traduce la petición que referencia a la vista a una petición equivalente con respecto a las tablas fuente de la vista y lleva a cabo la petición equivalente. De este modo el DBMS mantiene la ilusión de la vista mientras mantiene la integridad de las tablas fuentes.

VENTAJAS

1. **Seguridad:** cada usuario puede obtener permisos para acceder a la base de datos únicamente a través de un pequeño conjunto de vistas que contienen los datos específicos que el usuario está autorizado a ver, restringiendo así el acceso del usuario a los datos almacenados.
2. **Simplicidad de consulta:** una vista puede extraer datos de varias tablas diferentes y presentarlos como una única tabla, haciendo que consultas multitabla se formulen como consultas de una sola tabla con respecto a la vista.
3. **Simplicidad estructurada:** las vistas pueden dar a un usuario una visión de la estructura de la base de datos presentando esta como un conjunto de tablas virtuales que tienen sentido para ese usuario.
4. **Aislamiento frente al cambio.** Una vista puede presentar una imagen consistente inalterada de la estructura de la base de datos, incluso si las tablas fuente subyacentes se dividen, reestructuran o cambian de nombre.
5. **Integridad de datos.** Si se accede a los datos y se introducen a través de una vista, el DBMS puede comprobar automáticamente los datos para asegurarse que satisfacen restricciones de integridad especificadas.

DESVENTAJAS

1. **Rendimiento:** las vistas crean la apariencia de una tabla, pero el DBMS debe traducir las consultas con respecto a la vista en consultas con respecto a las tablas fuentes subyacentes. Si la vista se define mediante una consulta multitabla compleja, entonces incluso una consulta sencilla con respecto a la vista se convierte en una composición complicada y puede tardar mucho tiempo en completarse.
2. **Restricciones de actualización:** cuando un usuario trata de actualizar filas de una vista, el DBMS debe traducir la petición a una actualización sobre las filas de las tablas fuentes subyacentes. Esto es posible para vistas sencillas, pero vistas más complejas no pueden ser actualizadas, son de solo lectura.

SENTENCIAS

- **CREATE VIEW:** crea una vista.
- **DROP VIEW:** elimina una vista, teniendo en cuenta lo que ocurre cuando un usuario intenta eliminar una vista y la definición de otra vista depende de ella, agregando la cláusula **CASCADE** o **RESTRICT**.

SEGURIDAD SQL

Ver [Aspectos de seguridad](#).

CONCESIÓN DE PRIVILEGIOS (GRANT)

La sentencia GRANT se utiliza para conceder privilegios de seguridad sobre objetos de la BD específicos.

Ejemplo:

```
GRANT SELECT, INSERT, UPDATE
ON Pedidos
TO JaviPerez (Permite consultas, inserciones y actualizaciones al usuario JaviPerez en la tabla Pedidos)
```

```
GRANT SELECT (NumEmpl, Nombre, OficinaRep)
ON RepVentas
TO JaviPerez, MarcoGomez (Similar a la anterior, pero restringe columnas, y otro usuario)
```

PASO DE PRIVILEGIOS (GRANT OPTION)

Permite transferir privilegios a otros usuarios sobre un objeto de la base de datos.

Ejemplo:

```
GRANT SELECT
ON Pedidos
TO MarcoGomez
WITH GRANT OPTION
```

REVOCACIÓN DE PRIVILEGIOS (REVOKE)

La sentencia REVOKE puede retirar todos o parte de los privilegios a un usuario.

Ejemplo:

```
REVOKE SELECT
ON RepVentas
FROM JaviPerez
```

EL CATÁLOGO DEL SISTEMA

CONCEPTO

Es una colección de tablas especiales en una base de datos que son propiedad, están creadas y son mantenidas por el propio DBMS. Estas tablas del sistema contienen datos que describen la estructura de la base de datos. Las tablas del catálogo de sistema se crean automáticamente al crear la base de datos. Generalmente se recogen todas juntas bajo un id usuario de sistema especial con un nombre como SYSTEM, o SYSIBM, etc.

El DBMS se refiere constantemente a los datos del catálogo de sistema cuando procesa las sentencias SQL.

En un SELECT:

- Verifica que las tablas designadas existen.
- Asegura que el usuario tiene permisos.
- Comprueba si existen las columnas.
- Resuelve los nombres de las columnas.
- Determina el tipo de dato de cada columna.

HERRAMIENTAS DE CONSULTA

Unos de los beneficios más importantes del catálogo de sistema es que hace posible herramientas de consulta de fácil actualización. Permite un acceso simple y transparente a la base de datos sin tener que aprender el lenguaje SQL.

ESTÁNDARES

El estándar SQL2 proporciona una especificación de un conjunto de vistas que dan un acceso estandarizado a la información que normalmente se encuentra en el catálogo del sistema. Un DBMS que cumpla el estándar SQL debe soportar estas vistas, que en su conjunto se denominan INFORMATION_SCHEMA.

CONTENIDO DEL CATÁLOGO

- **Tablas:** el catálogo describe cada tabla de la base de datos, con nombre, propietario, número de columnas, tamaño, etc.
- **Columnas:** describe cada columna de la base de datos, con nombre, tabla a la que pertenece, tipo de dato, tamaño, si se permiten nulos, etc.
- **Usuarios:** describe cada usuario autorizado en la base de datos, con contraseña cifrada y otros datos.
- **Vistas:** describe cada vista definida en la base de datos, con nombre, propietarios, la consulta de definición, etc.
- **Privilegios:** describe cada grupo de privilegios concedidos, incluyendo los objetos, los usuarios, y los donantes.

EL ESQUEMA DE INFORMACIÓN DE SQL2

El estándar SQL2 no especifica directamente un catálogo de sistema que deba ser soportado por las implementaciones de los DBMS. Se definió un catálogo de sistema idealizado que los fabricantes podrían diseñar partiendo de cero. Estas tablas de un catálogo idealizado se denominan esquema de definición en el estándar. El estándar no exige que un DBMS realmente soporte las tablas del catálogo de sistema o cualquier otro catálogo de sistema. Sin embargo, define una serie de vistas sobre las tablas del catálogo que identifican los objetos de la base de datos que están accesibles al usuario actual. Cualquier DBMS que cumpla con el estándar debe soportar estas vistas.