



CENTRO DE ESTUDIOS FINANCIEROS

VIRIATO, 52	28010 MADRID	914 44 49 20
PONZANO, 15	28010 MADRID	914 44 49 20
G. DE GRÀCIA, 171	08012 BARCELONA	934 15 09 88
ALBORAYA, 23	46010 VALENCIA	963 61 41 99

www.cef.es

info@cef.es

Índice Tema 5

1. Antecedentes históricos.
2. Ventajas e inconvenientes de los sistemas de bases de datos.
3. Características y elementos constitutivos del SGBD.
 - 3.1. Definición.
 - 3.2. Clasificación de los sistemas de gestión de bases de datos.
 - 3.3. Características SGBD.
 - 3.4. Elementos de un SGBD.
4. Sistemas gestores de bases de datos relacionales.
 - 4.1. El modelo relacional.
 - 4.2. Las doce reglas de Codd.
 - 4.3. Lenguajes relacionales.
 - 4.3.1. Álgebra relacional.
 - 4.3.2. Cálculo relacional.
 - 4.3.3. Cálculo vs álgebra relacional.
5. El lenguaje SQL.
 - 5.1. Introducción.
 - 5.2. Comandos DDL.
 - 5.3. Comandos DML.
6. Estándares de conectividad: ODBC y JDBC.
 - 6.1. ODBC.
 - 6.2. OLE-DB.
 - 6.3. JDBC.
 - 6.4. Ventajas JDBD vs ODBC.





CENTRO DE ESTUDIOS FINANCIEROS

VIRIATO, 52	28010 MADRID	914 44 49 20
PONZANO, 15	28010 MADRID	914 44 49 20
G. DE GRÀCIA, 171	08012 BARCELONA	934 15 09 88
ALBORAYA, 23	46010 VALENCIA	963 61 41 99

www.cef.es

info@cef.es

TEMA 5

Sistemas de gestión de bases de datos relacionales. Antecedentes históricos. Características y elementos constitutivos. El lenguaje SQL. Estándares de conectividad: ODBC y JDBC.

1. ANTECEDENTES HISTÓRICOS.

En los comienzos de la historia de la informática los datos estaban integrados en los programas como constantes.

Con la aparición de los ficheros como colección de datos homogénea, es decir, integrados por elementos estructurados de la misma manera, en el que cada elemento contiene el mismo tipo de información y almacenada en soporte informático determinado, se diferencian de manera incipiente la estructura lógica que representa el punto de vista del usuario y la estructura física de los datos (en general, cinta magnética y por lo tanto, organización secuencial de los datos).

Posteriormente, la definición de ficheros independientes del resto del programa podrá llevarse a cabo por medio del lenguaje de programación. De esta manera, se facilita el acceso y actualización de los ficheros, y se evita la programación de muchas tareas repetitivas. Como consecuencia de la aparición de subsistemas de gestión de datos integrados en los sistemas operativos, las estructuras lógica y física se empiezan a diferenciar.

El nivel de diferenciación entre las estructuras lógica y física alcanzado hasta este momento no es suficiente para evitar la dependencia casi total de los datos respecto a los programas y viceversa, y de ambos respecto a la máquina.

Con el fin de atenuar estas dependencias (entre datos y aplicaciones) se pasa a una arquitectura que diferencia claramente la representación de los datos orientados hacia el problema (estructura lógica de los datos) de la representación orientada hacia la máquina (estructura física de los datos), siendo necesaria una transformación (mapping) de una en otra.

En la década de los 60 nacen los primeros SGBD (Sistemas Gestores de Bases de Datos). Los sistemas dejan de estar orientados al proceso o tratamiento y se orientan hacia las Bases de Datos. Los datos y sus interrelaciones se integran en las bases de datos y se aíslan de las aplicaciones. La estructura lógica se hace más flexible y sencilla y la estructura física se complica buscando mejorar el rendimiento. El tipo de Arquitectura de estos primeros SGBDs era a dos niveles:

1. Estructura Global (características lógicas y físicas): esquema.
2. Vistas Lógicas Externas de los usuarios: subesquemas.

Los sistemas jerárquico y de red constituyen esta primera generación de los SGBDs. Pero estos sistemas presentan algunos inconvenientes:

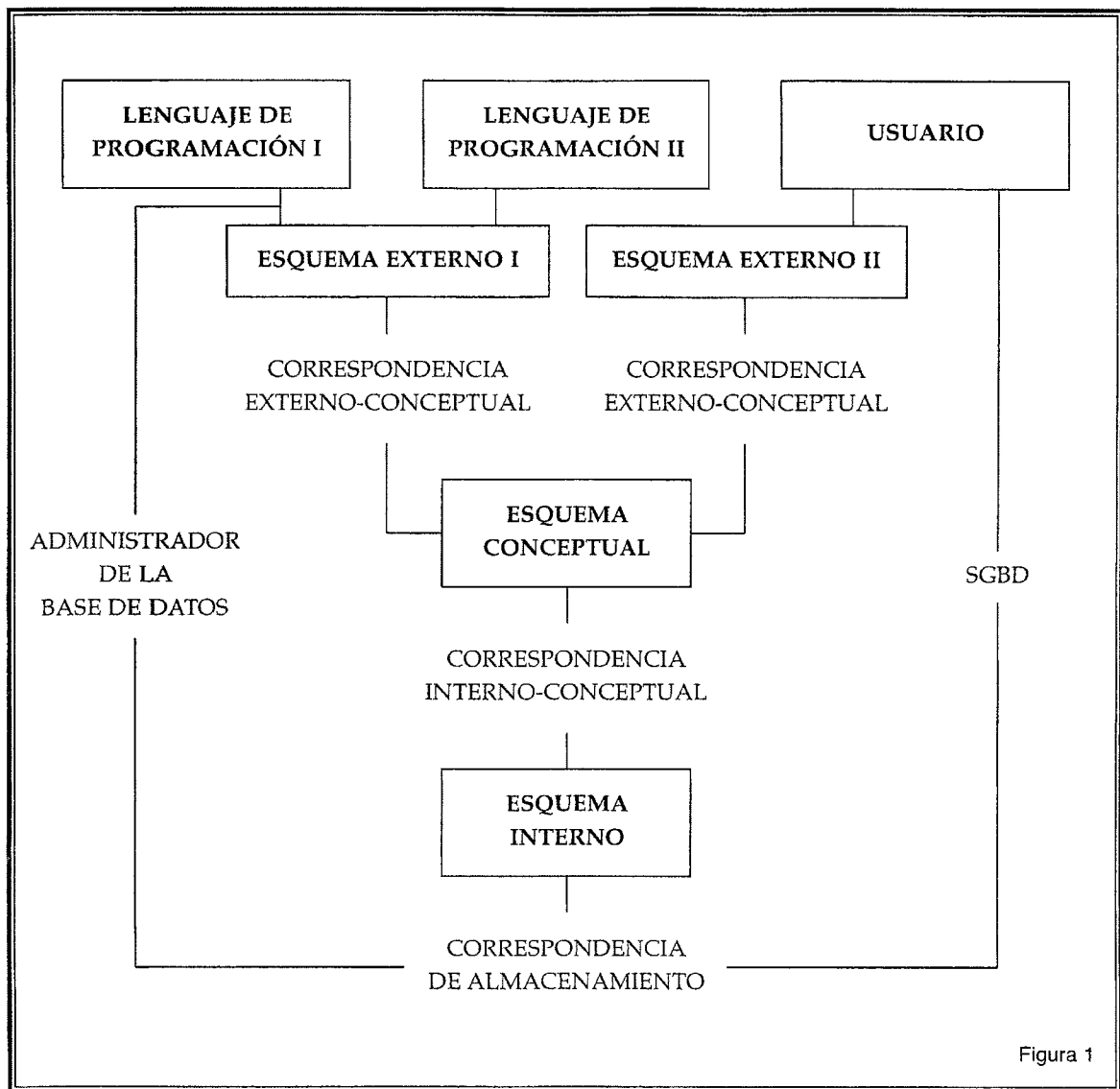
- Es necesario escribir complejos programas de aplicación para responder a cualquier tipo de consulta de datos, por simple que ésta sea.
- La independencia de datos es mínima.
- No tienen un fundamento teórico.

En 1970 Codd, de los laboratorios de investigación de IBM, escribió un artículo presentando el modelo relacional. En este artículo, presentaba también los inconvenientes de los sistemas previos, el jerárquico y el de red. Fue entonces cuando se comenzaron a desarrollar muchos sistemas relacionales, apareciendo los primeros a finales de los 70 y principios de los 80. Uno de los primeros es System R, de IBM, que se desarrolló para probar la funcionalidad del modelo relacional, proporcionando una implementación de sus estructuras de datos y sus operaciones. Esto condujo a dos grandes desarrollos:

1. El desarrollo de un lenguaje de consultas estructurado denominado SQL, que se ha convertido en el lenguaje estándar de los sistemas relacionales.
2. La producción de varios SGBD relacionales durante los años 80, como DB2 y SLQ/DS de IBM, y ORACLE de ORACLE Corporation.

En 1975 el Organismo de Estandarización de Los Estados Unidos (ANSI) publica un informe clave para la posterior evolución de los SGBDs. Insiste en la necesidad de conseguir total independencia entre datos y aplicaciones; propone una arquitectura a tres niveles y define el modelo conceptual para conseguir tales objetivos. Todo ello dio lugar a la denominada «Arquitectura ANSI / X3 / SPARC». En esta arquitectura se definen los siguientes niveles:

- Nivel Interno: representación más cercana al almacenamiento físico de los datos. Origina el denominado Esquema interno: ficheros, organización de ficheros, registros...
- Nivel conceptual: representación de los datos que intervienen en el problema. Origina el Esquema conceptual: datos, relaciones y restricciones.
- Nivel externo: visión que de la base de datos tiene el usuario. Origina el Esquema externo: datos de usuario + permisos de acceso.



2. VENTAJAS E INCONVENIENTES DE LOS SISTEMAS DE BASES DE DATOS.

Los sistemas de bases de datos presentan numerosas ventajas que se pueden dividir en dos grupos: las que se deben a la integración de datos y las que se deben a la interface común que proporciona el SGBD.

Ventajas por la integración de datos:

- **Control sobre la redundancia de datos.** Los sistemas de ficheros almacenan varias copias de los mismos datos en ficheros distintos. Esto hace que se desperdicie espacio de almacenamiento, además de provocar la falta de consistencia de datos. En los sistemas de bases de datos todos estos ficheros están integrados, por lo que no se almacenan varias copias de los mismos datos. Sin embargo, en una base de datos no se puede eliminar la redundancia completamente, ya que en ocasiones es necesaria para modelar las relaciones entre los datos, o bien es necesaria para mejorar las prestaciones.

- Consistencia de datos. Eliminando o controlando las redundancias de datos se reduce en gran medida el riesgo de que haya inconsistencias. Si un dato está almacenado una sola vez, cualquier actualización se debe realizar sólo una vez, y está disponible para todos los usuarios inmediatamente. Si un dato está duplicado y el sistema conoce esta redundancia, el propio sistema puede encargarse de garantizar que todas las copias se mantengan consistentes. Desgraciadamente, no todos los SGBD de hoy en día se encargan de mantener automáticamente la consistencia.
- Más información sobre la misma cantidad de datos. Al estar todos los datos integrados, se puede extraer información adicional sobre los mismos.
- Compartición de datos. En los sistemas de ficheros, los ficheros pertenecen a las personas o a los departamentos que los utilizan. Pero en los sistemas de bases de datos, la base de datos pertenece a la empresa y puede ser compartida por todos los usuarios que estén autorizados. Además, las nuevas aplicaciones que se vayan creando pueden utilizar los datos de la base de datos existente.
- Mantenimiento de estándares. Gracias a la integración es más fácil respetar los estándares necesarios, tanto los establecidos a nivel de la empresa como los nacionales e internacionales. Estos estándares pueden establecerse sobre el formato de los datos para facilitar su intercambio, pueden ser estándares de documentación, procedimientos de actualización y también reglas de acceso.

Ventajas por la existencia del SGBD:

- Mejora en la integridad de datos. La integridad de la base de datos se refiere a la validez y la consistencia de los datos almacenados. Normalmente, la integridad se expresa mediante restricciones o reglas que no se pueden violar. Estas restricciones se pueden aplicar tanto a los datos, como a sus relaciones, y es el SGBD quien se debe encargar de mantenerlas.
- Mejora en la seguridad. La seguridad de la base de datos es la protección de la base de datos frente a usuarios no autorizados. Sin unas buenas medidas de seguridad, la integración de datos en los sistemas de bases de datos hace que éstos sean más vulnerables que en los sistemas de ficheros. Sin embargo, los SGBD permiten mantener la seguridad mediante el establecimiento de claves para identificar al personal autorizado a utilizar la base de datos. Las autorizaciones se pueden realizar a nivel de operaciones, de modo que un usuario puede estar autorizado a consultar ciertos datos pero no a actualizarlos, por ejemplo.
- Mejora en la accesibilidad a los datos. Muchos SGBD proporcionan lenguajes de consultas o generadores de informes que permiten al usuario hacer cualquier tipo de consulta sobre los datos, sin que sea necesario que un programador escriba una aplicación que realice tal tarea.
- Mejora en la productividad. El SGBD proporciona muchas de las funciones estándar que el programador necesita escribir en un sistema de ficheros. A nivel básico, el SGBD proporciona todas las rutinas de manejo de ficheros típicas de los programas de aplicación. El hecho de disponer de estas funciones permite al programador centrarse mejor en la función específica requerida por los usuarios, sin tener que preocuparse de los detalles de implementación de bajo nivel. Muchos SGBD también proporcionan un entorno de cuarta generación consistente en un conjunto de herramientas que simplifican, en gran medida, el desarrollo de las aplicaciones que acceden a la base de datos. Gracias a estas herramientas, el programador puede ofrecer una mayor productividad en un tiempo menor.
- Mejora en el mantenimiento gracias a la independencia de datos. En los sistemas de ficheros, las descripciones de los datos se encuentran inmersas en los programas de aplicación que los manejan. Esto hace que los programas sean dependientes de los datos, de modo que un cambio en su estructura, o un cambio en el modo en que se almacenan en disco, requiere cambios importantes

en los programas cuyos datos se ven afectados. Sin embargo, los SGBD separan las descripciones de los datos de las aplicaciones. Esto es lo que se conoce como independencia de datos, gracias a la cual se simplifica el mantenimiento de las aplicaciones que acceden a la base de datos.

- **Aumento de la concurrencia.** En algunos sistemas de ficheros, si hay varios usuarios que pueden acceder simultáneamente a un mismo fichero, es posible que el acceso interfiera entre ellos de modo que se pierda información o, incluso, que se pierda la integridad. La mayoría de los SGBD gestionan el acceso concurrente a la base de datos y garantizan que no ocurran problemas de este tipo.
- **Mejora en los servicios de copias de seguridad y de recuperación ante fallos.** Muchos sistemas de ficheros dejan que sea el usuario quien proporcione las medidas necesarias para proteger los datos ante fallos en el sistema o en las aplicaciones. Los usuarios tienen que hacer copias de seguridad cada día, y si se produce algún fallo, utilizar estas copias para restaurarlos. En este caso, todo el trabajo realizado sobre los datos desde que se hizo la última copia de seguridad se pierde y se tiene que volver a realizar. Sin embargo, los SGBD actuales funcionan de modo que se minimiza la cantidad de trabajo perdido cuando se produce un fallo.

Inconvenientes de los sistemas de bases de datos:

- **Complejidad.** Los SGBD son conjuntos de programas muy complejos con una gran funcionalidad. Es preciso comprender muy bien esta funcionalidad para poder sacar un buen partido de ellos.
- **Tamaño.** Los SGBD son programas complejos y muy extensos que requieren una gran cantidad de espacio en disco y de memoria para trabajar de forma eficiente.
- **Coste económico del SGBD.** El coste de un SGBD varía dependiendo del entorno y de la funcionalidad que ofrece. Por ejemplo, un SGBD para un ordenador personal puede costar 500 euros, mientras que un SGBD para un sistema multiusuario que dé servicio a cientos de usuarios puede costar entre 10.000 y 100.000 euros. Además, hay que pagar una cuota anual de mantenimiento que suele ser un porcentaje del precio del SGBD.
- **Coste del equipamiento adicional.** Tanto el SGBD, como la propia base de datos, pueden hacer que sea necesario adquirir más espacio de almacenamiento. Además, para alcanzar las prestaciones deseadas, es posible que sea necesario adquirir una máquina más grande o una máquina que se dedique solamente al SGBD. Todo esto hará que la implantación de un sistema de bases de datos sea más cara.
- **Coste de la conversión.** En algunas ocasiones, el coste del SGBD y el coste del equipo informático que sea necesario adquirir para su buen funcionamiento, es insignificante comparado al coste de convertir la aplicación actual en un sistema de bases de datos. Este coste incluye el coste de enseñar a la plantilla a utilizar estos sistemas y, probablemente, el coste del personal especializado para ayudar a realizar la conversión y poner en marcha el sistema. Este coste es una de las razones principales por las que algunas empresas y organizaciones se resisten a cambiar su sistema actual de ficheros por un sistema de bases de datos.
- **Prestaciones.** Un sistema de ficheros está escrito para una aplicación específica, por lo que sus prestaciones suelen ser muy buenas. Sin embargo, los SGBD están escritos para ser más generales y ser útiles en muchas aplicaciones, lo que puede hacer que algunas de ellas no sean tan rápidas como antes.
- **Vulnerable a los fallos.** El hecho de que todo esté centralizado en el SGBD hace que el sistema sea más vulnerable ante los fallos que puedan producirse.

3. CARACTERÍSTICAS Y ELEMENTOS CONSTITUTIVOS DEL SGBD.

3.1. DEFINICIÓN.

Un sistema Gestor de Base de Datos (SGBD) es un conjunto coordinado de programas, procedimientos, lenguajes, etc., que suministra, tanto a los usuarios no informáticos como a los analistas, programadores o al administrador, los medios necesarios para describir, recuperar y manipular los datos almacenados en la base, manteniendo su integridad, confidencialidad y seguridad.

3.2. CLASIFICACIÓN DE LOS SISTEMAS DE GESTIÓN DE BASES DE DATOS.

El criterio principal que se utiliza para clasificar los SGBD es el modelo lógico en que se basan. Los modelos lógicos empleados con mayor frecuencia en los SGBD comerciales actuales son el relacional, el de red y el jerárquico. Algunos SGBD más modernos se basan en modelos orientados a objetos.

El modelo relacional se basa en el concepto matemático denominado «relación», que gráficamente se puede representar como una tabla. En el modelo relacional, los datos y las relaciones existentes entre los datos se representan mediante estas relaciones matemáticas, cada una con un nombre que es único y con un conjunto de columnas.

En el modelo relacional la base de datos es percibida por el usuario como un conjunto de tablas. Esta percepción es sólo a nivel lógico (en los niveles externo y conceptual de la arquitectura de tres niveles), ya que a nivel físico puede estar implementada mediante distintas estructuras de almacenamiento.

En el modelo de red los datos se representan como colecciones de registros y las relaciones entre los datos se representan mediante conjuntos, que son punteros en la implementación física. Los registros se organizan como un grafo: los registros son los nodos y los arcos son los conjuntos. El SGBD de red más popular es el sistema IDMS.

El modelo jerárquico es un tipo de modelo de red con algunas restricciones. De nuevo los datos se representan como colecciones de registros y las relaciones entre los datos se representan mediante conjuntos. Sin embargo, en el modelo jerárquico cada nodo puede tener un solo padre. Una base de datos jerárquica puede representarse mediante un árbol: los registros son los nodos, también denominados segmentos, y los arcos son los conjuntos. El SGBD jerárquico más importante es el sistema IMS.

La mayoría de los SGBD comerciales actuales están basados en el modelo relacional, mientras que los sistemas más antiguos estaban basados en el modelo de red o el modelo jerárquico. Estos dos últimos modelos requieren que el usuario tenga conocimiento de la estructura física de la base de datos a la que se accede, mientras que el modelo relacional proporciona una mayor independencia de datos. Se dice que el modelo relacional es declarativo (se especifica qué datos se han de obtener) y los modelos de red y jerárquico son navegacionales (se especifica cómo se deben obtener los datos).

El modelo orientado a objetos define una base de datos en términos de objetos, sus propiedades y sus operaciones. Los objetos con la misma estructura y comportamiento pertenecen a una clase, y las clases se organizan en jerarquías o grafos acíclicos. Las operaciones de cada clase se especifican en términos de procedimientos predefinidos denominados métodos. Algunos SGBD relacionales existentes en el mercado han estado extendiendo sus modelos para incorporar conceptos orientados a objetos. A estos SGBD se les conoce como sistemas objeto-relacionales.

Un segundo criterio para clasificar los SGBD es el número de usuarios a los que da servicio el sistema. Los sistemas monousuario sólo atienden a un usuario a la vez, y su principal uso se da en los ordenadores personales. Los sistemas multiusuario, entre los que se encuentran la mayor parte de los SGBD, atienden a varios usuarios al mismo tiempo.

Un tercer criterio es el número de sitios en los que está distribuida la base de datos. Casi todos los SGBD son centralizados: sus datos se almacenan en un solo computador. Los SGBD centralizados pueden atender a varios usuarios, pero el SGBD y la base de datos en sí residen por completo en una sola máquina. En los SGBD distribuidos la base de datos real y el propio software del SGBD pueden estar distribuidos en varios sitios conectados por una red. Los SGBD distribuidos homogéneos utilizan el mismo SGBD en múltiples sitios. Una tendencia reciente consiste en crear software para tener acceso a varias bases de datos autónomas preexistentes almacenadas en SGBD distribuidos heterogéneos. Esto da lugar a los SGBD federados o sistemas multibase de datos en los que los SGBD participantes tienen cierto grado de autonomía local. Muchos SGBD distribuidos emplean una arquitectura cliente-servidor.

3.3. CARACTERÍSTICAS SGBD.

- Un SGBD debe proporcionar un catálogo en el que se almacenen las descripciones de los datos y que sea accesible por los usuarios. Este catálogo es lo que se denomina diccionario de datos y contiene información que describe los datos de la base de datos (metadatos). Normalmente, un diccionario de datos almacena:
 - Nombre, tipo y tamaño de los datos.
 - Nombre de las relaciones entre los datos.
 - Restricciones de integridad sobre los datos.
 - Nombre de los usuarios autorizados a acceder a la base de datos.
 - Esquemas externos, conceptuales e internos, y correspondencia entre los esquemas.
 - Estadísticas de utilización, tales como la frecuencia de las transacciones y el número de accesos realizados a los objetos de la base de datos.

Algunos de los beneficios que reporta el diccionario de datos son los siguientes:

- La información sobre los datos se puede almacenar de un modo centralizado. Esto ayuda a mantener el control sobre los datos, como un recurso que son.
- El significado de los datos se puede definir, lo que ayudará a los usuarios a entender el propósito de los mismos.
- La comunicación se simplifica ya que se almacena el significado exacto. El diccionario de datos también puede identificar al usuario o usuarios que poseen los datos o que los acceden.
- Las redundancias y las inconsistencias se pueden identificar más fácilmente ya que los datos están centralizados.

- Se puede tener un historial de los cambios realizados sobre la base de datos.
 - El impacto que puede producir un cambio se puede determinar antes de que sea implementado, ya que el diccionario de datos mantiene información sobre cada tipo de dato, todas sus relaciones y todos sus usuarios.
 - Se puede hacer respetar la seguridad.
 - Se puede garantizar la integridad.
 - Se puede proporcionar información para auditorías.
- Un SGBD debe proporcionar a los usuarios la capacidad de almacenar grandes volúmenes de datos en la base de datos, acceder a ellos y actualizarlos. Esta es la función fundamental de un SGBD y por supuesto, el SGBD debe ocultar al usuario la estructura física interna (la organización de los ficheros y las estructuras de almacenamiento). El ocultamiento de lo físico únicamente lo consigue de forma efectiva el modelo relacional.
 - Un SGBD debe facilitar que la base de datos se actualice correctamente cuando varios usuarios la están actualizando concurrentemente. Uno de los principales objetivos de los SGBD es el permitir que varios usuarios tengan acceso concurrente a los datos que comparten. El acceso concurrente es relativamente fácil de gestionar si todos los usuarios se dedican a leer datos, ya que no pueden interferir unos con otros. Sin embargo, cuando dos o más usuarios están accediendo a la base de datos y al menos uno de ellos está actualizando datos, pueden interferir de modo que se produzcan inconsistencias en la base de datos. El SGBD se debe encargar de que estas interferencias no se produzcan en el acceso simultáneo.
 - Un SGBD debe proporcionar el que todas las actualizaciones correspondientes a una determinada transacción se realicen, o que no se realice ninguna. Una transacción es un conjunto de acciones que cambian el contenido de la base de datos. Una transacción en el sistema informático de la empresa inmobiliaria sería dar de alta a un empleado o eliminar un inmueble. Una transacción un poco más complicada sería eliminar un empleado y reasignar sus inmuebles a otro empleado. En este caso hay que realizar varios cambios sobre la base de datos. Si la transacción falla durante su realización, por ejemplo porque falla el hardware, la base de datos quedará en un estado inconsistente. Algunos de los cambios se habrán hecho y otros no, por lo tanto, los cambios realizados deberán ser deshechos para devolver la base de datos a un estado consistente.
 - Un SGBD debe posibilitar la recuperación de la base de datos en caso de que ocurra algún suceso que la dañe. Como se ha comentado antes, cuando el sistema falla en medio de una transacción, la base de datos se debe devolver a un estado consistente. Este fallo puede ser a causa de un fallo en algún dispositivo hardware o un error del software, que hagan que el SGBD aborte, o puede ser a causa de que el usuario detecte un error durante la transacción y la aborte antes de que finalice. En todos estos casos, el SGBD debe proporcionar un mecanismo capaz de recuperar la base de datos llevándola a un estado consistente.
 - Un SGBD debe proporcionar un mecanismo que garantice que sólo los usuarios autorizados pueden acceder a la base de datos. La protección debe ser contra accesos no autorizados, tanto intencionados como accidentales.
 - Un SGBD debe ser capaz de integrarse con algún software de comunicación. Muchos usuarios acceden a la base de datos desde terminales. En ocasiones estos terminales se encuentran co-

nectados directamente a la máquina sobre la que funciona el SGBD. En otras ocasiones los terminales están en lugares remotos, por lo que la comunicación con la máquina que alberga al SGBD se debe hacer a través de una red. En cualquiera de los dos casos, el SGBD recibe peticiones en forma de mensajes y responde de modo similar. Todas estas transmisiones de mensajes las maneja el gestor de comunicaciones de datos. Aunque este gestor no forma parte del SGBD, es necesario que el SGBD se pueda integrar con él para que el sistema sea comercialmente viable.

- Un SGBD debe proporcionar los medios necesarios para garantizar que tanto los datos de la base de datos, como los cambios que se realizan sobre estos datos, sigan ciertas reglas. La integridad de la base de datos requiere la validez y consistencia de los datos almacenados. Se puede considerar como otro modo de proteger la base de datos, pero además de tener que ver con la seguridad, tiene otras implicaciones. La integridad se ocupa de la calidad de los datos. Normalmente se expresa mediante restricciones, que son una serie de reglas que la base de datos no puede violar.
- Un SGBD debe permitir que se mantenga la independencia entre los programas y la estructura de la base de datos. La independencia de datos se alcanza mediante las vistas o subesquemas. La independencia de datos física es más fácil de alcanzar, de hecho hay varios tipos de cambios que se pueden realizar sobre la estructura física de la base de datos sin afectar a las vistas. Sin embargo, lograr una completa independencia de datos lógica es más difícil. Añadir una nueva entidad, un atributo o una relación puede ser sencillo, pero no es tan sencillo eliminarlos.
- Todo SGBD ha de incorporar un lenguaje de definición de datos (DDL) y otro para la manipulación de los datos (DML). Adicionalmente se aporta el lenguaje de control (DCL).
- Un SGBD debe proporcionar una serie de herramientas que permitan administrar la base de datos de modo efectivo. Algunas herramientas trabajan a nivel externo, por lo que habrán sido producidas por el administrador de la base de datos. Las herramientas que trabajan a nivel interno deben ser proporcionadas por el distribuidor del SGBD. Algunas de ellas son:
 - Herramientas para importar y exportar datos.
 - Herramientas para monitorizar el uso y el funcionamiento de la base de datos.
 - Programas de análisis estadístico para examinar las prestaciones o las estadísticas de utilización.
 - Herramientas para reorganización de índices.
 - Herramientas para aprovechar el espacio dejado en el almacenamiento físico por los registros borrados y que consoliden el espacio liberado para reutilizarlo cuando sea necesario.

3.4. ELEMENTOS DE UN SGBD:

Los SGBD son paquetes de software muy complejos y sofisticados que deben proporcionar los servicios comentados en la sección anterior. No se puede generalizar sobre los elementos que componen un SGBD ya que varían mucho unos de otros. Sin embargo, es muy útil conocer sus componentes y cómo se relacionan cuando se trata de comprender lo que es un sistema de bases de datos.

Un SGBD tiene varios módulos, cada uno de los cuales realiza una función específica. El sistema operativo proporciona servicios básicos al SGBD, que es construido sobre él.

- El procesador de consultas es el componente principal de un SGBD. Transforma las consultas en un conjunto de instrucciones de bajo nivel que se dirigen al gestor de la base de datos.
- El gestor de la base de datos es el interface con los programas de aplicación y las consultas de los usuarios. El gestor de la base de datos acepta consultas y examina los esquemas externo y conceptual para determinar qué registros se requieren para satisfacer la petición. Entonces el gestor de la base de datos realiza una llamada al gestor de ficheros para ejecutar la petición.
- El gestor de ficheros maneja los ficheros en disco en donde se almacena la base de datos. Este gestor establece y mantiene la lista de estructuras e índices definidos en el esquema interno. Si se utilizan ficheros dispersos, llama a la función de dispersión para generar la dirección de los registros. Pero el gestor de ficheros no realiza directamente la entrada y salida de datos. Lo que hace es pasar la petición a los métodos de acceso del sistema operativo que se encargan de leer o escribir los datos en el buffer del sistema.
- El preprocesador del LMD (Lenguaje de Manipulación de Datos) convierte las sentencias del LMD embebidas en los programas de aplicación, en llamadas a funciones estándar escritas en el lenguaje anfitrión. El preprocesador del LMD debe trabajar con el procesador de consultas para generar el código apropiado.
- El compilador del LDD (Lenguaje de Definición de Datos) convierte las sentencias del LDD en un conjunto de tablas que contienen metadatos. Estas tablas se almacenan en el diccionario de datos.
- El gestor del diccionario controla los accesos al diccionario de datos y se encarga de mantenerlo. La mayoría de los componentes del SGBD acceden al diccionario de datos.

Los principales componentes del gestor de la base de datos son los siguientes:

- Control de autorización. Este módulo comprueba que el usuario tiene los permisos necesarios para llevar a cabo la operación que solicita.
- Procesador de comandos. Una vez que el sistema ha comprobado los permisos del usuario, se pasa el control al procesador de comandos.
- Control de la integridad. Cuando una operación cambia los datos de la base de datos, este módulo debe comprobar que la operación a realizar satisface todas las restricciones de integridad necesarias.
- Optimizador de consultas. Este módulo determina la estrategia óptima para la ejecución de las consultas.
- Gestor de transacciones. Este módulo realiza el procesamiento de las transacciones.
- Planificador (scheduler). Este módulo es el responsable de asegurar que las operaciones que se realizan concurrentemente sobre la base de datos tengan lugar sin conflictos.

- Gestor de recuperación. Este módulo garantiza que la base de datos permanezca en un estado consistente en caso de que se produzca algún fallo.
- Gestor de buffers. Este módulo es el responsable de transferir los datos entre memoria principal y los dispositivos de almacenamiento secundario. A este módulo también se le denomina gestor de datos.

4. SISTEMAS GESTORES DE BASES DE DATOS RELACIONALES (SGBDR).

4.1. EL MODELO RELACIONAL.

El modelo relacional es un modelo con sólidos fundamentos matemáticos, basado en la teoría de conjuntos. Fue definido por E. F. Codd en 1970.

Las características fundamentales del modelo relacional son:

1. Las estructuras de datos son simples: Son relaciones que se presentan al usuario en forma de tablas bidimensionales, permitiendo un alto grado de independencia de la información con respecto al medio físico de almacenamiento de los datos.
2. Proporciona una base sólida para la consistencia de los datos a través de las reglas de integridad. Igualmente, el proceso de normalización, al eliminar ciertas anomalías que pueden presentarse en las relaciones, representa una valiosa ayuda para el diseño de la base de datos.
3. Permite la manipulación de las relaciones en forma orientada a conjuntos. Esto ha conducido al desarrollo de lenguajes muy potentes basados, bien en la teoría de conjuntos (álgebra relacional), bien en la lógica de predicados (cálculo relacional).

Conceptos fundamentales del Modelo Relacional:

El esquema de una Base de Datos Relacional se compone de uno o más esquemas de relación y de un conjunto de restricciones de integridad.

Un esquema de relación (o intensión de una relación) consiste en el nombre de relación, seguido de los nombres de los atributos: nombre relación (ATRIBUTO1, ATRIBUTO2,..... ATRIBUTO n).

Se define una relación: «Dados los dominios D_1, D_2, \dots, D_n (no necesariamente distintos), R es una relación entre estos n -conjuntos si es un conjunto de n -tuplas (d_1, d_2, \dots, d_n) tal que d_1 pertenece a D_1, \dots, d_n pertenece a D_n ».

Un dominio es un conjunto de valores.

Cada atributo, o propiedad con interés informativo de una relación, está asociado a un dominio del que toma sus posibles valores.

El número de atributos de una relación define su grado, mientras que el número de tuplas de la relación define su cardinalidad.

La extensión u ocurrencia de una relación es una tabla donde las filas corresponden a las tuplas y las columnas a los atributos.

De estas definiciones se deducen las características de una tabla de estructura relacional:

- Cada tabla debe contener un solo tipo de filas. El formato de cada fila queda definido por el esquema de la relación. Es decir todas las filas tienen las mismas columnas y formato.
- Cada fila tiene que ser única, no puede haber filas duplicadas.
- El orden de las filas dentro de una tabla es indiferente.
- Cada columna debe estar identificada por un nombre específico.
- El orden de las columnas dentro de una tabla es indiferente.
- Cada columna debe extraer sus valores de un dominio.
- Un mismo dominio podrá servir para definir los valores de varias columnas diferentes.
- El valor individual de la intersección de cualquier fila y columna será un único dato.

Restricciones del Modelo Relacional. Reglas de Integridad:

Al definir cada atributo sobre un dominio se impone una restricción sobre el conjunto de valores permitidos para cada atributo. A este tipo de restricciones se les denomina restricciones de dominios. Hay además dos reglas de integridad muy importantes que son restricciones que se deben cumplir en todas las bases de datos relacionales y en todos sus estados o instancias (las reglas se deben cumplir todo el tiempo). Estas reglas son la regla de integridad de entidades y la regla de integridad referencial. Antes de definir las, es preciso conocer el concepto de nulo.

- Nulo: cuando en una tupla un atributo es desconocido, se dice que es nulo. Un nulo no representa el valor cero ni la cadena vacía, éstos son valores que tienen significado. El nulo implica ausencia de información, bien porque al insertar la tupla se desconocía el valor del atributo, o bien porque para dicha tupla el atributo no tiene sentido.

Ya que los nulos no son valores, deben tratarse de modo diferente, lo que causa problemas de implementación. De hecho, no todos los SGBD relacionales soportan los nulos.

- Regla de integridad de la entidad: se aplica a las claves primarias de las relaciones base: ninguno de los atributos que componen la clave primaria puede ser nulo.

Por definición, una clave primaria es un identificador irreducible que se utiliza para identificar de modo único las tuplas. Que sea irreducible significa que ningún subconjunto de la clave primaria sirve para identificar las tuplas de modo único. Si se permite que parte de la clave primaria sea nula, se está diciendo que no todos sus atributos son necesarios para distinguir las tuplas, con lo que se contradice la irreducibilidad.

Esta regla sólo se aplica a las relaciones base y a las claves primarias, no a las claves alternativas.

- Regla de integridad referencial: la segunda regla de integridad se aplica a las claves ajenas: si en una relación hay alguna clave ajena, sus valores deben coincidir con valores de la clave primaria a la que hace referencia, o bien, deben ser completamente nulos.

Por lo tanto, para cada clave ajena de la base de datos habrá que contestar a tres preguntas:

- Regla de los nulos: ¿tiene sentido que la clave ajena acepte nulos?
- Regla de borrado: ¿qué ocurre si se intenta borrar la tupla referenciada por la clave ajena?
 - Restringir: no se permite borrar la tupla referenciada.
 - Propagar: se borra la tupla referenciada y se propaga el borrado a las tuplas que la referencian mediante la clave ajena.
 - Anular: se borra la tupla referenciada y las tuplas que la referenciaban ponen a nulo la clave ajena (sólo si acepta nulos).
- Regla de modificación: ¿qué ocurre si se intenta modificar el valor de la clave primaria de la tupla referenciada por la clave ajena?
 - Restringir: no se permite modificar el valor de la clave primaria de la tupla referenciada.
 - Propagar: se modifica el valor de la clave primaria de la tupla referenciada y se propaga la modificación a las tuplas que la referencian mediante la clave ajena.
 - Anular: se modifica la tupla referenciada y las tuplas que la referenciaban ponen a nulo la clave ajena (sólo si acepta nulos).

4.2. LAS DOCE REGLAS DE CODD.

Para que un sistema Gestor de Base de Datos pueda ser considerado como relacional ha de cumplir al menos 6 de las 12 reglas establecidas por el padre de la Teoría Relacional, F. Codd, y que van precedidas de la Regla 0, obligatoria obviamente para todo SGBDR.

0. Regla 0: gestión de una BDR. Un SGBDR debe ser capaz de manejar base de datos exclusivamente con capacidades relacionales.
1. La regla de información. Toda la información de una base de datos relacional está representada explícitamente a nivel lógico y exactamente de un modo, mediante valores en tablas.
2. Regla de acceso garantizado. Todos y cada uno de los datos (valor atómico) de una base de datos relacional se garantiza que sean lógicamente accesibles recurriendo a una combinación de nombre de tabla, valor de clave primaria y nombre de columna.
3. Tratamiento sistemático de valores nulos. Los valores nulos (distintos de la cadena de caracteres vacías o de una cadena de caracteres en blanco y distintos del cero o de cualquier otro número) se soportan en los DBMS completamente relacionales para representar la falta de información y la información inaplicable de un modo sistemático e independiente del tipo de datos.

4. Catálogo en línea dinámico basado en el modelo relacional. La descripción de la base de datos se representa a nivel lógico del mismo modo que los datos ordinarios, de modo que los usuarios autorizados puedan aplicar a su interrogación el mismo lenguaje relacional que aplican a los datos regulares.
5. Regla de sublenguaje completo de datos. Un sistema relacional puede soportar varios lenguajes y varios modos de uso terminal (por ejemplo, el modo de rellenar con blancos). Sin embargo, debe haber al menos un lenguaje cuyas sentencias sean expresables, mediante alguna sintaxis bien definida, como cadenas de caracteres, y que sea completa en cuanto al soporte de todos los puntos siguientes:
 - Definición de datos.
 - Definición de vista.
 - Manipulación de datos (interactiva y por programa).
 - Restricciones de integridad.
 - Autorización.
 - Fronteras de transacciones (comienzo, vuelta atrás).
6. Regla de actualización de vista. Todas las vistas que sean teóricas actualizables son también actualizables por el sistema.
7. Inserción, actualización y supresión de alto nivel. La capacidad de manejar una relación de base de datos o una relación derivada como un único operando se aplica no solamente a la recuperación de datos, sino también a la inserción, actualización y supresión de los datos.
8. Independencia física de los datos. Independencia del soporte y del método de acceso.
9. Independencia lógica de los datos. Los datos deben ser independientes de las aplicaciones que los utilicen.
10. Independencia de integridad. Las restricciones de integridad específicas para una base de datos relacional particular deben ser definibles en el sublenguaje de datos relacional y almacenables en el catálogo, no en los programas de aplicación.
11. Independencia de distribución. Un DBMS relacional tiene independencia de distribución.
12. Regla de no subversión (o no inversión). Si un sistema relacional tiene un lenguaje de bajo nivel (un solo registro cada vez), ese bajo nivel no puede ser utilizado para subvertir o suprimir las reglas de integridad y las restricciones expresadas en el lenguaje relacional de nivel superior (múltiples registros a la vez).

4.3. LENGUAJES RELACIONALES.

Son varios los lenguajes utilizados por los SGBD relacionales para manejar las relaciones. Algunos de ellos son procedurales, lo que quiere decir que el usuario dice al sistema exactamente cómo debe manipular los datos. Otros son no procedurales, que significa que el usuario dice qué datos necesita, en lugar de decir cómo deben obtenerse.

En este apartado se presentan el álgebra relacional y el cálculo relacional, definidos por Codd como la base de los lenguajes relacionales. Se puede decir que el álgebra es un lenguaje procedural (de alto nivel), mientras que el cálculo relacional es un lenguaje no procedural. Sin embargo, ambos lenguajes son equivalentes: para cada expresión del álgebra, se puede encontrar una expresión equivalente en el cálculo, y viceversa.

El álgebra relacional o el cálculo relacional se utilizan para medir la potencia de los lenguajes relacionales. Si un lenguaje permite obtener cualquier relación que se pueda derivar mediante el álgebra relacional, se dice que es relacionalmente completo. La mayoría de los lenguajes relacionales son relacionalmente completos, pero tienen más potencia que el álgebra o el cálculo porque se les han añadido operadores especiales.

4.3.1. Álgebra relacional.

El álgebra relacional es un lenguaje formal con una serie de operadores que trabajan sobre una o varias relaciones para obtener otra relación resultado, sin que cambien las relaciones originales. Tanto los operandos como los resultados son relaciones, por lo que la salida de una operación puede ser la entrada de otra operación. Esto permite anidar expresiones del álgebra, del mismo modo que se pueden anidar las expresiones aritméticas. A esta propiedad se le denomina clausura: las relaciones son cerradas bajo el álgebra, del mismo modo que los números son cerrados bajo las operaciones aritméticas.

De los ocho operadores, sólo hay cinco que son fundamentales: restricción, proyección, producto cartesiano, unión y diferencia, que permiten realizar la mayoría de las operaciones de obtención de datos. Los operadores no fundamentales son la concatenación (join), la intersección y la división, que se pueden expresar a partir de los cinco operadores fundamentales.

La restricción y la proyección son operaciones unarias porque operan sobre una sola relación. El resto de las operaciones son binarias porque trabajan sobre pares de relaciones. En las definiciones que se presentan a continuación, se supone que R y S son dos relaciones cuyos atributos son $R = (r_1, r_2, \dots, r_N)$ y $S = (s_1, s_2, \dots, s_M)$ respectivamente.

- Selección: opera sobre una sola relación R y da como resultado otra relación cuyas tuplas son las tuplas de R que satisfacen la condición especificada (C). Esta condición es una comparación en la que aparece al menos un atributo de R , o una combinación booleana de varias de estas comparaciones. Se representa como $\sigma_c(R)$.
- Proyección: opera sobre una sola relación R y da como resultado otra relación que contiene un subconjunto vertical de R , extrayendo los valores de los atributos especificados y eliminando duplicados. Se representa como $\pi_{1, 2, \dots, n}(R)$.
- Producto cartesiano: dadas dos relaciones R y S , se define el producto cartesiano como la relación resultante de combinar cada fila de R con todas las de S . El grado de la relación resultante será la suma del número de columnas de R y S . La cardinalidad será el producto del número de filas de R por el número de filas de S . Ya que es posible que haya atributos con el mismo nombre en las dos relaciones, el nombre de la relación se antepondrá al del atributo en este caso, para que los nombres de los atributos sigan siendo únicos en la relación resultado. Se representa como $R \times S$.
- Join: dadas dos relaciones R y S , se define la concatenación o join como la relación resultante del producto cartesiano entre ambas tablas una vez seleccionadas aquellas filas que tomen

igual valor en aquella o aquellas fila/s expresadas en la operación. Para que la operación se pueda producir es necesario que ambas relaciones presenten, al menos, un campo en común sobre el que establecer la condición de igualdad. Se representa como: $R \bowtie X \bowtie S$.

- Unión: dadas dos relaciones R y S, se define la unión entre ambas como la relación resultante de tomar las filas que están en una u otra relación y además las que están en ambas (sólo una vez). Para que se pueda producir la unión es necesario que las relaciones R y S presenten igual grado y compatibilidad de dominios para cada par de atributos tomados de uno en uno entre ambas relaciones, es decir, r_1 ha de ser de un dominio igual o compatible a s_1 , en general, r_N ha de ser de un dominio igual o compatible a s_N . Se representa por $R \cup S$.
- Intersección: dadas dos relaciones R y S, se define la intersección como la relación resultante de tomar las filas que están tanto en R como en S. Al igual que en la unión, las relaciones R y S han de presentar igual grado y compatibilidad de dominios para cada par de atributos tomados de uno en uno entre ambas relaciones. Se representa como $R \cap S$.
- Diferencia: dadas dos relaciones R y S, se define la diferencia entre ambas como la relación resultante de tomar las filas que están en R y no están en S. Se trata de una operación no conmutativa, a diferencia de las anteriores. Al igual que en la unión, las relaciones R y S han de presentar igual grado y compatibilidad de dominios para cada par de atributos tomados de uno en uno entre ambas relaciones. Se representa como $R - S$.
- Cociente: dadas dos relaciones R y S en las que existe un subconjunto de atributos (S') que están formando parte de S y R al mismo tiempo y otro conjunto de atributos (R') que únicamente forman parte de R, se define la división o cociente como la relación resultante de combinar cada fila de R' con todas las filas que forman parte de los atributos S' . Se representa por R/S .

4.3.2. Cálculo relacional.

El álgebra relacional y el cálculo relacional son formalismos diferentes que representan distintos estilos de expresión del manejo de datos en el ámbito del modelo relacional. El álgebra relacional proporciona una serie de operaciones que se pueden usar para decir al sistema cómo construir la relación deseada a partir de las relaciones de la base de datos. El cálculo relacional proporciona una notación para formular la definición de la relación deseada en términos de las relaciones de la base de datos.

El cálculo relacional toma su nombre del cálculo de predicados, que es una rama de la lógica. Hay dos tipos de cálculo relacional, el orientado a tuplas, propuesto por Codd, y el orientado a dominios, propuesto por otros autores.

En el cálculo de predicados (lógica de primer orden), un predicado es una función con argumentos que se puede evaluar a verdadero o falso. Cuando los argumentos se sustituyen por valores, la función lleva a una expresión denominada proposición, que puede ser verdadera o falsa.

Si un predicado tiene una variable, como en «x es un tema de esta oposición», esta variable debe tener un rango asociado. Cuando la variable se sustituye por alguno de los valores de su rango, la proposición puede ser cierta; para otros valores puede ser falsa. Por ejemplo, si el rango de x es el conjunto de todos los temas y reemplazamos x por «Sistemas Gestores de Bases de Datos», la proposición «Sistemas Gestores de Bases de Datos es un tema de esta oposición» es cierta. Pero si reemplazamos x por un tema de otra oposición la proposición es falsa.

Si F es un predicado, la siguiente expresión corresponde al conjunto de todos los valores de x para los que F es cierto:

$$x \text{ WHERE } F(x)$$

Los predicados se pueden conectar mediante AND, OR y NOT para formar predicados compuestos.

Cálculo orientado a tuplas:

En el cálculo relacional orientado a tuplas, lo que interesa es encontrar tuplas para las que se cumple cierto predicado. El cálculo orientado a tuplas se basa en el uso de variables tupla. Una variable tupla es una variable cuyo rango de valores son las tuplas de una relación.

Por ejemplo, para especificar el rango de la variable tupla TX sobre la relación TEMAS_GESTION_INFORMATICA se utiliza la siguiente expresión:

$$\text{RANGE OF TX IS TEMAS_GESTION_INFORMATICA}$$

Para expresar la consulta «obtener todas las tuplas TX para las que F(PX) es cierto», se escribe la siguiente expresión:

$$\text{TX WHERE } F(\text{TX})$$

donde F es lo que se denomina una fórmula bien formada. Por ejemplo, para expresar la consulta «obtener todos los datos de los temas que tienen más de 100 folios» se puede escribir:

$$\text{RANGE OF TX IS TEMAS_GESTION_INFORMATICA}$$
$$\text{TX WHERE TX.NFolios} > 100$$

TX.NFolios se refiere al valor del atributo número de folios para la tupla TX. Para que se muestren solamente algunos atributos, por ejemplo, Descripción y Grupo, en lugar de todos los atributos de la relación, se escribe:

$$\text{RANGE OF TX IS TEMAS_GESTION_INFORMATICA}$$
$$\text{TX.Descripcion, TX.Grupo WHERE TX.NFolios} > 100$$

Hay dos cuantificadores que se utilizan en las fórmulas bien formadas para decir a cuántas instancias se aplica el predicado. El cuantificador existencial («existe») se utiliza en las fórmulas bien formadas que deben ser ciertas para al menos una instancia.

$$\text{RANGE OF OX IS OPOSCIONES}$$
$$\exists \text{ OX (OX.onum} = \text{TX.onum AND OX.Descripcion} = \text{'Gestión Informática'})}$$

Esta fórmula bien formada dice que «existe una oposición que tiene el mismo número que el número de oposición de la tupla que ahora se encuentra en la variable de Temas, TX, y que está es Gestión Informática». El cuantificador universal \forall («para todo») se utiliza en las fórmulas bien formadas que deben ser ciertas para todas las instancias.

$$\forall TX (TX.Gruppo \neq \text{«Grupo V»})$$

Esta fórmula bien formada dice que «para todas las tuplas de Temas», el grupo no es el quinto. Utilizando las reglas de las operaciones lógicas, esta fórmula bien formada se puede escribir también del siguiente modo:

$$\text{NOT } \exists TX (TX.Gruppo = \text{«Grupo V»})$$

que dice que «no hay ningún tema del grupo V».

Las variables tupla que no están cuantificadas por \forall o \exists se denominan variables libres. Si están cuantificadas, se denominan variables ligadas. El cálculo, al igual que cualquier lenguaje, tiene una sintaxis que permite construir expresiones válidas. Para que una expresión no sea ambigua y tenga sentido, debe seguir esta sintaxis:

- Si P es una fórmula bien formada n-ária (un predicado con n argumentos) y t_1, t_2, \dots, t_n son constantes o variables, entonces es también una fórmula bien formada.
- Si t_1 y t_2 son constantes o variables del mismo dominio y θ es un operador de comparación ($<, <=, >=, =, \neq$), entonces $t_1 \theta t_2$ es una fórmula bien formada.
- Si P_1 y P_2 son fórmulas bien formadas, también lo son su conjunción $P_1 \text{ AND } P_2$, su disyunción $P_1 \text{ OR } P_2$ y la negación $\text{NOT } P_1$. Además, si P es una fórmula bien formada que tiene una variable libre X, entonces $\exists X(P)$ y $\forall X(P)$ y también son fórmulas bien formadas.

4.3.3. Cálculo vs álgebra relacional.

El álgebra relacional y el cálculo relacional tienen el mismo poder de expresión, es decir, todas las consultas que se pueden formular utilizando álgebra relacional pueden también formularse utilizando el cálculo relacional, y viceversa. Esto fue probado por E. F. Codd en 1972. Este profesor se basó en un algoritmo («algoritmo de reducción de Codd») mediante el cual una expresión arbitraria del cálculo relacional se puede reducir a la expresión semánticamente equivalente del álgebra relacional.

Se dice a veces que los lenguajes basados en el cálculo relacional son de «más alto nivel» o «más declarativos» que los basados en el álgebra relacional porque el álgebra especifica (parcialmente) el orden de las operaciones, mientras el cálculo lo traslada a un compilador o interprete que determina el orden de evaluación más eficiente.

5. EL LENGUAJE SQL.

5.1. INTRODUCCIÓN.

SQL se ha convertido en el lenguaje de consulta relacional más popular. El nombre «SQL» es una abreviatura de Structured Query Language (Lenguaje de consulta estructurado). En 1974 Donald Chamberlain y otros definieron el lenguaje SEQUEL (Structured English Query Language) en IBM Research. Este lenguaje fue implementado inicialmente en un prototipo de IBM llamado SEQUEL-XRM en 1974-75. En 1976-77 se definió una revisión de SEQUEL llamada SEQUEL/2 y el nombre se cambió a SQL.

IBM desarrolló un nuevo prototipo llamado System R en 1977. System R implementó un amplio subconjunto de SEQUEL/2 (now SQL) y un número de cambios que se le hicieron a (now SQL) durante el proyecto. System R se instaló en un número de puestos de usuario, tanto internos en IBM como en algunos clientes seleccionados. Gracias al éxito y aceptación de System R en los mismos, IBM inició el desarrollo de productos comerciales que implementaban el lenguaje SQL basado en la tecnología System R.

Durante los años siguientes, IBM y bastantes otros vendedores anunciaron productos SQL tales como SQL/DS (IBM), DB2 (IBM), ORACLE (Oracle Corp.), DG/SQL (Data General Corp.), y SYBASE (Sybase Inc.).

SQL es también un estándar oficial hoy. En 1982, la American National Standards Institute (ANSI) encargó a su Comité de Bases de Datos X3H2 el desarrollo de una propuesta de lenguaje relacional estándar. Esta propuesta fue ratificada en 1986 y consistía básicamente en el dialecto de IBM de SQL. En 1987, este estándar ANSI fue también aceptado por la Organización Internacional de Estandarización (ISO). Esta versión estándar original de SQL recibió informalmente el nombre de «SQL/86». En 1989, el estándar original fue extendido, y recibió el nuevo nombre, también informal, de «SQL/89». También en 1989 se desarrolló un estándar relacionado llamado Database Language Embedded SQL (ESQL).

Los comités ISO y ANSI han estado trabajando durante muchos años en la definición de una versión muy ampliada del estándar original, llamado informalmente SQL2 o SQL/92. Esta versión se convirtió en un estándar ratificado durante 1992: International Standard ISO/IEC 9075:1992, Database Language SQL. SQL/92 es la versión a la que normalmente la gente se refiere cuando habla de «SQL estándar».

Como en el caso de los más modernos lenguajes relacionales, SQL está basado en el cálculo relacional de tuplas. Como resultado, toda consulta formulada utilizando el cálculo relacional de tuplas (o su equivalente, el álgebra relacional) se puede formular también utilizando SQL. Hay, sin embargo, capacidades que van más allá del cálculo o del álgebra relacional. Aquí tenemos una lista de algunas características proporcionadas por SQL que no forman parte del álgebra y del cálculo relacionales:

- Comandos para inserción, borrado o modificación de datos.
- Capacidades aritméticas: en SQL es posible incluir operaciones aritméticas así como comparaciones, por ejemplo $A < B + 3$. Nótese que ni $+$ ni otros operadores aritméticos aparecían en el álgebra relacional ni en cálculo relacional.

- Asignación y comandos de impresión: es posible imprimir una relación construida por una consulta y asignar una relación calculada a un nombre de relación.
- Funciones agregadas: operaciones tales como promedio (average), suma (sum), máximo (max), etc., se pueden aplicar a las columnas de una relación para obtener una cantidad única.

A continuación se describen los comandos más importantes utilizados en el «SQL estándar», para lo cual utilizaremos ejemplos basados en las tablas siguientes y suponiendo un SGBR como PostgreSQL.

Base de Datos de Proveedores y Artículos:

SUPPLIER	SNO	SNAME	CITY	SELLS	SNO	PNO
	1	Smith	London		1	1
	2	Jones	Paris		1	2
	3	Adams	Vienna		2	4
	4	Blake	Rome		3	1
					3	3
					4	2
					4	3
					4	4
PART	PNO	PNAME	PRICE			
	1	Tornillos	10			
	2	Tuercas	8			
	3	Cerros	15			
	4	Levas	25			

Las tablas PART y SUPPLIER se pueden ver como entidades y SELLS se puede ver como una relación entre un artículo particular y un proveedor particular.

5.2. COMANDOS DDL (DATA DEFINITION LANGUAGE).

El lenguaje SQL incluye un conjunto de comandos para definición de datos.

CREATE TABLE.

El comando fundamental para definir datos es el que crea una nueva relación (una nueva tabla). La sintaxis del comando CREATE TABLE es:

```
CREATE TABLE table_name
    (name_of_attr_1 type_of_attr_1[constraints_in_line]
    [, name_of_attr_2 type_of_attr_2 [constraints_in_line]
    [, ...]]
    [CONSTRAINT constraint_name tipo_constraint [argumentos de constraint]
    [, ...]] );
```

Ejemplo. creación de una tabla:

Para crear las tablas definidas en la base de datos de Proveedores y Artículos se utilizaron las siguientes instrucciones de SQL:

```
CREATE TABLE SUPPLIER
(SNO INTEGER,
SNAME VARCHAR(20) UNIQUE,
CITY VARCHAR(20),
CONSTRAINT pk_supplier PRIMARY KEY (SNO));
```

```
CREATE TABLE PART
(PNO INTEGER PRIMARY KEY,
PNAME VARCHAR(20) NOT NULL,
PRICE DECIMAL(4, 2));
```

```
CREATE TABLE SELLS
(SNO INTEGER,
PNO INTEGER,
CONSTRAINT pk_sells PRIMARY KEY (SNO, PNO),
CONSTRAINT supplier_fk_sells FOREIGN KEY (SNO) REFERENCES supplier,
CONSTRAINT part_fk_sells FOREIGN KEY (PNO) REFERENCES part);
```

Tipos de Datos en SQL:

A continuación sigue una lista de algunos tipos de datos soportados por SQL:

- **INTEGER**: entero binario con signo de palabra completa (31 bits de precisión).
- **SMALLINT**: entero binario con signo de media palabra (15 bits de precisión).
- **DECIMAL (p[,q])**: número decimal con signo de p dígitos de precisión, asumiendo q a la derecha para el punto decimal. ($15 \geq p \geq q \geq 0$). Si q se omite, se asume que vale 0. Oracle denomina a este tipo Number.
- **FLOAT**: numérico con signo de doble palabra y coma flotante.
- **CHAR(n)**: cadena de caracteres de longitud fija, de longitud n.
- **VARCHAR(n)**: cadena de caracteres de longitud variable, de longitud máxima n. Oracle denomina a este tipo VARCHAR2 (n).

Create Index:

Se utilizan los índices para acelerar el acceso a una relación. Si una relación R tiene un índice en el atributo A podremos recuperar todas las tuplas t que tienen $t(A) = a$ en un tiempo aproximadamente proporcional al número de tales tuplas t más que en un tiempo proporcional al tamaño de R.

Para crear un índice en SQL se utiliza el comando CREATE INDEX. La sintaxis es:

```
CREATE INDEX index_name  
ON table_name ( name_of_attribute );
```

Ejemplo. Create Index:

Para crear un índice llamado I sobre el atributo SNAME de la relación SUPPLIER, utilizaremos la siguiente instrucción:

```
CREATE INDEX I  
ON SUPPLIER (SNAME);
```

El índice creado se mantiene automáticamente, es decir, cada vez que una nueva tupla se inserte en la relación SUPPLIER, se adaptará el índice I. Nótese que el único cambio que un usuario puede percibir cuando se crea un índice es un incremento en la velocidad.

Create View:

Se puede ver una vista como una tabla virtual, es decir, una tabla que no existe físicamente en la base de datos, pero aparece al usuario como si existiese. Por contra, cuando hablamos de una tabla base, hay realmente un equivalente almacenado para cada fila en la tabla en algún sitio del almacenamiento físico.

Las vistas no tienen datos almacenados propios, distinguibles y físicamente almacenados. En su lugar, el sistema almacena la definición de la vista (es decir, las reglas para acceder a las tablas base físicamente almacenadas para materializar la vista) en algún lugar de los catálogos del sistema.

En SQL se utiliza el comando CREATE VIEW para definir una vista. La sintaxis es:

```
CREATE VIEW view_name  
AS select_stmt
```

donde select_stmt es una instrucción select válida. Nótese que select_stmt no se ejecuta cuando se crea la vista. Simplemente se almacena en los catálogos del sistema y se ejecuta cada vez que se realiza una consulta contra la vista.

Sea la siguiente definición de una vista (utilizamos de nuevo las tablas de la base de datos de Proveedores y Artículos):


```

CREATE VIEW London_Suppliers
AS SELECT S.SNAME, P.PNAME
FROM SUPPLIER S, PART P, SELLS SE
WHERE S.SNO = SE.SNO AND
      P.PNO = SE.PNO AND
      S.CITY = 'London';

```

Ahora podemos utilizar esta relación virtual London_Suppliers como si se tratase de otra tabla base:

```

SELECT *
FROM London_Suppliers
WHERE P.PNAME = 'Tornillos';

```

Lo cual nos devolverá la siguiente tabla:

SNAME	PNAME
Smith	Tornillos

Para calcular este resultado, el sistema de base de datos ha realizado previamente un acceso oculto a las tablas de la base SUPPLIER, SELLS y PART. Hace esto ejecutando la consulta dada en la definición de la vista contra aquellas tablas base. Tras eso, las cualificaciones adicionales (dadas en la consulta contra la vista) se podrán aplicar para obtener la tabla resultante.

Drop Table, Drop Index, Drop View:

Se utiliza el comando DROP TABLE para eliminar una tabla (incluyendo todas las tuplas almacenadas en ella):

```
DROP TABLE table_name;
```

Para eliminar la tabla SUPPLIER, utilizaremos la instrucción:

```
DROP TABLE SUPPLIER;
```

Se utiliza el comando DROP INDEX para eliminar un índice:

```
DROP INDEX index_name;
```

Finalmente, eliminaremos una vista dada utilizando el comando DROP VIEW:

```
DROP VIEW view_name;
```

5.3. COMANDOS DML (DATA MANIPULATION LANGUAGE).

Insert Into.

Una vez que se crea una tabla, puede ser llenada con tuplas mediante el comando INSERT INTO. La sintaxis es:

```
INSERT INTO table_name (name_of_attr_1  
                        [, name_of_attr_2 [...]])  
VALUES (val_attr_1  
      [, val_attr_2 [, ...]]);
```

Para insertar la primera tupla en la relación SUPPLIER (de la base de datos de Proveedores y Artículos) utilizamos la siguiente instrucción:

```
INSERT INTO SUPPLIER (SNO, SNAME, CITY)  
VALUES (1, 'Smith', 'London');
```

Para insertar la primera tupla en la relación SELLS, utilizamos:

```
INSERT INTO SELLS (SNO, PNO)  
VALUES (1, 1);
```

Update:

Para cambiar uno o más valores de atributos de tuplas en una relación, se utiliza el comando UPDATE. La sintaxis es:

```
UPDATE table_name  
SET name_of_attr_1 = value_1  
  [, ... [, name_of_attr_k = value_k]]  
WHERE condition;
```

Para cambiar el valor del atributo PRICE en el artículo «Tornillos» de la relación PART, utilizamos:

```
UPDATE PART  
SET PRICE = 15  
WHERE PNAME = 'Tornillos';
```

El nuevo valor del atributo PRICE de la tupla cuyo nombre es 'Tornillos' es ahora 15.

Delete:

Para borrar una tupla de una tabla particular, utilizamos el comando DELETE FROM. La sintaxis es:

```
DELETE FROM table_name
WHERE condition;
```

Para borrar el proveedor llamado 'Smith' de la tabla SUPPLIER, utilizamos la siguiente instrucción:

```
DELETE FROM SUPPLIER
WHERE SNAME = 'Smith';
```

Select:

El comando más usado en SQL es la instrucción SELECT, que se utiliza para recuperar datos. La sintaxis es:

```
SELECT [ALL|DISTINCT]
      { * | expr_1 [AS c_alias_1] [, ...
        [, expr_k [AS c_alias_k]]}
FROM table_name_1 [t_alias_1]
     [, ... [, table_name_n [t_alias_n]]]
[WHERE condition]
[GROUP BY name_of_attr_i
     [, ... [, name_of_attr_j]] [HAVING condition]]
[{UNION [ALL] | INTERSECT | EXCEPT} SELECT ...]
[ORDER BY name_of_attr_i [ASC|DESC]
     [, ... [, name_of_attr_j [ASC|DESC]]]];
```

Ilustraremos ahora la compleja sintaxis de la instrucción SELECT con varios ejemplos.

Select sencillas:

Ejemplo. Consulta sencilla con cualificación:

Para recuperar todas las tuplas de la tabla PART donde el atributo PRICE es mayor que 10, formularemos la siguiente consulta:

```
SELECT * FROM PART
WHERE PRICE > 10;
```

y obtenemos la siguiente tabla:

PNO	PNAME	PRICE
3	Cerrojos	15
4	Levas	25

Utilizando «*» en la instrucción SELECT solicitaremos todos los atributos de la tabla. Si queremos recuperar sólo los atributos PNAME y PRICE de la tabla PART utilizaremos la instrucción:

```
SELECT PNAME, PRICE
FROM PART
WHERE PRICE > 10;
```

En este caso el resultado es:

PNAME	PRICE
Cerros	15
Levas	25

Nótese que la SELECT SQL corresponde a la «proyección» en álgebra relaciona.

Las cualificaciones o condiciones en la cláusula WHERE pueden también conectarse lógicamente utilizando las palabras claves OR, AND, y NOT:

```
SELECT PNAME, PRICE
FROM PART
WHERE PNAME = 'Cerros' AND
      (PRICE = 0 OR PRICE < 15);
```

dará como resultado:

PNAME	PRICE
Cerros	15

Las operaciones aritméticas se pueden utilizar en la lista de objetivos y en la cláusula WHERE. Por ejemplo, si queremos conocer cuanto cuestan si tomamos dos piezas de un artículo, podríamos utilizar la siguiente consulta:

```
SELECT PNAME, PRICE * 2 AS DOUBLE
FROM PART
WHERE PRICE * 2 < 50;
```

y obtenemos:

PNAME	DOUBLE
Tornillos	20
Tuercas	16
Cerros	30

Nótese que la palabra DOBLE tras la palabra clave AS es el nuevo título de la segunda columna. Esta técnica puede utilizarse para cada elemento de la lista objetivo para asignar un nuevo título a la columna resultante. Este nuevo título recibe el calificativo de «un alias». El alias no puede utilizarse en todo el resto de la consulta.

Joins (combinaciones):

El siguiente ejemplo muestra como las joins (combinaciones) se realizan en SQL.

Para combinar tres tablas SUPPLIER, PART y SELLS a través de sus atributos comunes, formularemos la siguiente instrucción:

```
SELECT S.SNAME, P.PNAME
FROM SUPPLIER S, PART P, SELLS SE
WHERE S.SNO = SE.SNO AND
      P.PNO = SE.PNO;
```

y obtendremos la siguiente tabla como resultado:

SNAME	PNAME
Smith	Tornillos
Smith	Tuercas
Jones	Levas
Adams	Tornillos
Adams	Cerrojos
Blake	Tuercas
Blake	Cerrojos
Blake	Levas

En la cláusula FROM hemos introducido un alias al nombre para cada relación porque hay atributos con nombre común (SNO y PNO) en las relaciones. Ahora podemos distinguir entre los atributos con nombre común simplificando la adicción de un prefijo al nombre del atributo con el nombre del alias seguido de un punto. La combinación se calcula de la misma forma. Primero el producto cartesiano: SUPPLIER _ PART _ SELLS. Ahora seleccionamos únicamente aquellas tuplas que satisfagan las condiciones dadas en la cláusula WHERE (es decir, los atributos con nombre común deben ser iguales). Finalmente eliminamos las columnas repetidas (S.SNAME, P.PNAME).

Operadores Agregados:

SQL proporciona operadores agregados (como son AVG, COUNT, SUM, MIN, MAX) que toman el nombre de un atributo como argumento. El valor del operador agregado se calcula sobre todos los valores de la columna especificada en la tabla completa. Si se especifican grupos en la consulta, el cálculo se hace sólo sobre los valores de cada grupo.

Ejemplo. Agregados.

Si queremos conocer el coste promedio de todos los artículos de la tabla PART, utilizaremos la siguiente consulta

```
SELECT AVG(PRICE) AS AVG_PRICE  
FROM PART;
```

El resultado es:

AVG_PRICE
14.5

Si queremos conocer cuantos artículos se recogen en la tabla PART, utilizaremos la instrucción:

```
SELECT COUNT(PNO)  
FROM PART;
```

y obtendremos:

COUNT
4

Agregación por Grupos:

SQL nos permite particionar las tuplas de una tabla en grupos. En estas condiciones, los operadores agregados descritos antes pueden aplicarse a los grupos (es decir, el valor del operador agregado no se calculan sobre todos los valores de la columna especificada, sino sobre todos los valores de un grupo. El operador agregado se calcula individualmente para cada grupo).

El particionamiento de las tuplas en grupos se hace utilizando las palabras clave GROUP BY seguidas de una lista de atributos que definen los grupos. Si tenemos GROUP BY A1, ..., Ak habremos particionado la relación en grupos, de tal modo que dos tuplas son del mismo grupo si y sólo si tienen el mismo valor en sus atributos A1, ..., Ak.

Ejemplo. Agregados

Si queremos conocer cuántos artículos han sido vendidos por cada proveedor formularemos la consulta:

```
SELECT S.SNO, S.SNAME, COUNT(SE.PNO)  
FROM SUPPLIER S, SELLS SE  
WHERE S.SNO = SE.SNO  
GROUP BY S.SNO, S.SNAME;
```

y obtendremos:

SNO	SNAME	COUNT
1	Smith	2
2	Jones	1
3	Adams	2
4	Blake	3

Demos ahora una mirada a lo que está ocurriendo aquí. Primero, la combinación (join) de las tablas SUPPLIER y SELLS:

S.SNO	S.SNAME	SE.PNO
1	Smith	1
1	Smith	2
2	Jones	4
3	Adams	1
3	Adams	3
4	Blake	2
4	Blake	3
4	Blake	4

Ahora particionamos las tuplas en grupos reuniendo todas las tuplas que tiene el mismo atributo en S.SNO y S.SNAME:

S.SNO	S.SNAME	SE.PNO
1	Smith	1
		2
2	Jones	4
3	Adams	1
		3
4	Blake	2
		3
		4

En nuestro ejemplo, obtenemos cuatro grupos y ahora podemos aplicar el operador agregado COUNT para cada grupo, obteniendo el resultado total de la consulta dada anteriormente.

Nótese que para el resultado de una consulta utilizando GROUP BY y operadores agregados para dar sentido a los atributos agrupados, debemos primero obtener la lista objetivo. Los demás

atributos que no aparecen en la cláusula GROUP BY se seleccionarán utilizando una función agregada. Por otro lado, no se pueden utilizar funciones agregadas en atributos que aparecen en la cláusula GROUP BY.

Having:

La cláusula HAVING trabaja de forma muy parecida a la cláusula WHERE, y se utiliza para considerar sólo aquellos grupos que satisfagan la cualificación dada en la misma. Las expresiones permitidas en la cláusula HAVING deben involucrar funciones agregadas. Cada expresión que utilice sólo atributos planos deberá recogerse en la cláusula WHERE. Por otro lado, toda expresión que involucre funciones agregadas debe aparecer en la cláusula HAVING.

Ejemplo. Having.

Si queremos solamente los proveedores que venden más de un artículo, utilizaremos la consulta:

```
SELECT S.SNO, S.SNAME, COUNT(SE.PNO)
FROM SUPPLIER S, SELLS SE
WHERE S.SNO = SE.SNO
GROUP BY S.SNO, S.SNAME
HAVING COUNT(SE.PNO) > 1;
```

y obtendremos:

SNO	SNAME	COUNT
1	Smith	2
3	Adams	2
4	Blake	3

Subconsultas:

En las cláusulas WHERE y HAVING se permite el uso de subconsultas (subselects) en cualquier lugar donde se espere un valor. En este caso, el valor debe derivar de la evaluación previa de la subconsulta. El uso de subconsultas amplía el poder expresivo de SQL.

Ejemplo. Subselect.

Si queremos conocer los artículos que tienen mayor precio que el artículo llamado 'Tornillos', utilizaremos la consulta:

```
SELECT *
FROM PART
WHERE PRICE > (SELECT PRICE FROM PART
WHERE PNAME='Tornillos');
```


El resultado será:

PNO	PNAME	PRICE
3	Cerros	15
4	Levas	25

Cuando revisamos la consulta anterior, podemos ver la palabra clave **SELECT** dos veces. La primera al principio de la consulta -a la que nos referiremos como la **SELECT** externa- y la segunda en la cláusula **WHERE**, donde empieza una consulta anidada -nos referiremos a ella como la **SELECT** interna. Para cada tupla de la **SELECT** externa, la **SELECT** interna deberá ser evaluada. Tras cada evaluación, conoceremos el precio de la tupla llamada 'Tornillos', y podremos chequear si el precio de la tupla actual es mayor.

Si queremos conocer todos los proveedores que no venden ningún artículo (por ejemplo, para poderlos eliminar de la base de datos), utilizaremos:

```
SELECT *  
FROM SUPPLIER S  
WHERE NOT EXISTS  
  (SELECT * FROM SELLS SE  
   WHERE SE.SNO = S.SNO);
```

En nuestro ejemplo, obtendremos un resultado vacío, porque cada proveedor vende al menos un artículo. Nótese que utilizamos **S.SNO** de la **SELECT** externa en la cláusula **WHERE** de la **SELECT** interna. Como hemos descrito antes, la subconsulta se evalúa para cada tupla de la consulta externa, es decir, el valor de **S.SNO** se toma siempre de la tupla actual de la **SELECT** externa.

Unión, intersección, excepción:

Estas operaciones calculan la unión, la intersección y la diferencia de la teoría de conjuntos de las tuplas derivadas de dos subconsultas.

Ejemplo. Union, intersect, except:

La siguiente consulta es un ejemplo de **UNION**:

```
SELECT S.SNO, S.SNAME, S.CITY  
FROM SUPPLIER S  
WHERE S.SNAME = 'Jones'  
UNION  
SELECT S.SNO, S.SNAME, S.CITY  
FROM SUPPLIER S  
WHERE S.SNAME = 'Adams';
```

Dará el resultado:

SNO	SNAME	CITY
2	Jones	Paris
3	Adams	Vienna

Aquí tenemos un ejemplo para INTERSECT:

```
SELECT S.SNO, S.SNAME, S.CITY
FROM SUPPLIER S
WHERE S.SNO > 1
INTERSECT
SELECT S.SNO, S.SNAME, S.CITY
FROM SUPPLIER S
WHERE S.SNO > 2;
```

que dará como resultado:

SNO	SNAME	CITY
2	Jones	Paris

La única tupla devuelta por ambas partes de la consulta es la única que tiene SNO=2\$.

Finalmente, un ejemplo de EXCEPT:

```
SELECT S.SNO, S.SNAME, S.CITY
FROM SUPPLIER S
WHERE S.SNO > 1
EXCEPT
SELECT S.SNO, S.SNAME, S.CITY
FROM SUPPLIER S
WHERE S.SNO > 3;
```

que dará como resultado:

SNO	SNAME	CITY
2	Jones	Paris
3	Adams	Vienna

6. ESTÁNDARES DE CONECTIVIDAD: ODBC Y JDBC.

6.1. ODBC (OPEN DATABASE CONNECTIVITY).

Proporciona un interfaz, un estándar abierto de API para el acceso a bases de datos heterogéneas vía SQL, lo cual facilita la interoperabilidad sin necesidad de conocer los interfaces propietarios de bases de datos.

Está basado en la especificación X/Open CAE Data Management y ISO/IEC 9075-3:1999 Call-Level Interface (SQL/CLI).

El uso de ODBC implica:

- Renuncia a características avanzadas y exclusivas de muchos motores
- Añadir una capa más de indirección.

Los componentes de la arquitectura ODBC de una aplicación son:

1. Aplicación.
2. Administrador de drivers ODBC.
3. Driver.
4. DSN (Data Source Name).

6.2. OLE-DB.

Se trata de un conjunto de interfases basadas en COM que proporcionan a las aplicaciones un acceso uniforme a datos almacenados en distintos orígenes de información.

OLE-DB Service Provider: los proveedores OLEDB proporcionan un mecanismo uniforme de acceso a los datos relacionales y a los datos no relacionales. Se construyen sobre la base del modelo COM (Component Object Model) mientras que los drivers ODBC están basados en una especificación de APIs de C.

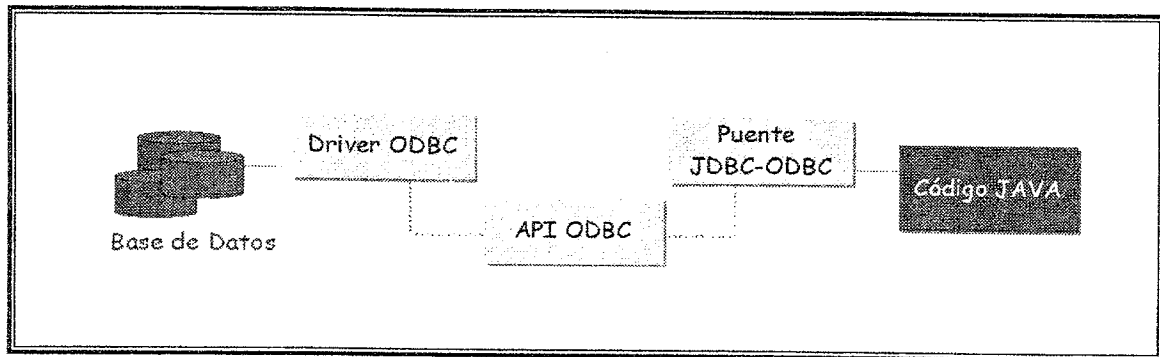
ADO = ActiveX Data Objects.

6.3. JDBC (JAVA DATABASE CONNECTIVITY).

JDBC es una API, formada por conjunto de clases e interfaces en el lenguaje de programación Java, para ejecutar sentencias SQL.

Tipos de drivers JDBC:

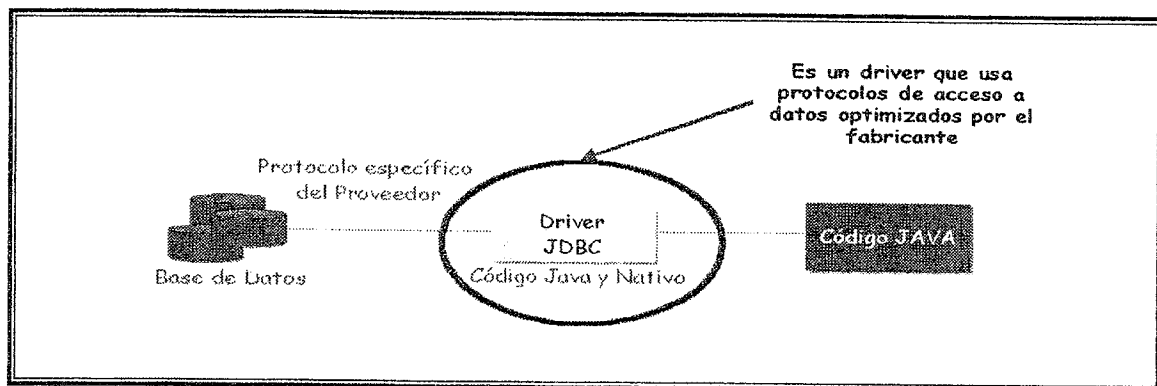
TIPO 1: PUENTE JDBC-ODBC.



Ventaja: se proporciona con la plataforma Java 2.

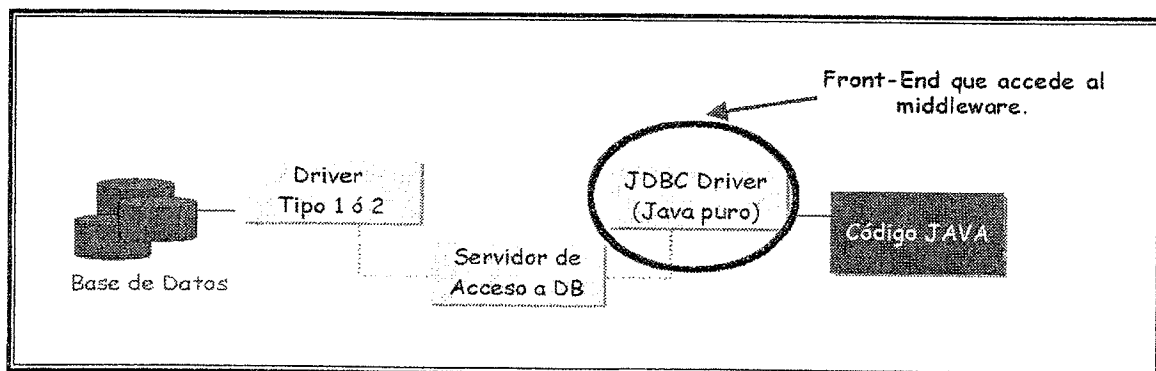
Desventajas: lentitud (dos drivers) y requiere de una instalación previa de ODBC bien instalada.

TIPO 2: DRIVER ESCRITO PARTE EN JAVA QUE HACE LLAMADAS A LA API NATIVA DEL SGBDR.



No necesita ODBC, e interactúa directamente con él la librería nativa (p. ej. DB-Library para SQL Server o CT-Lib para Sybase).

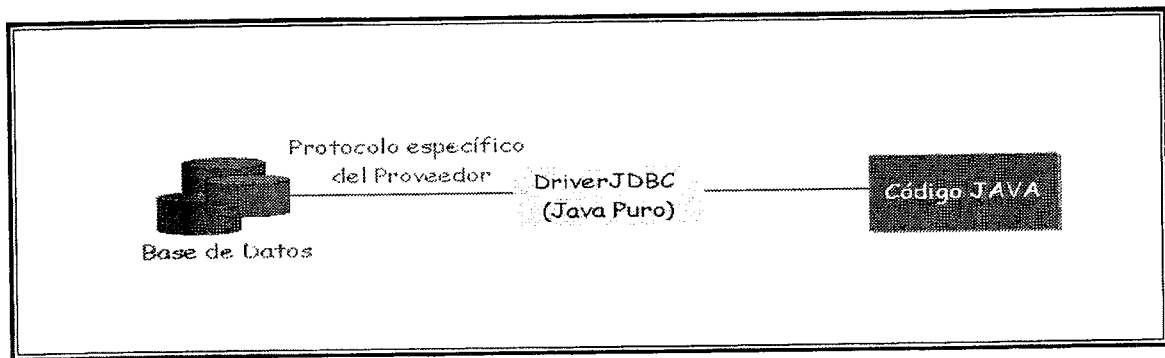
TIPO 3: FRONT-END QUE ACCEDE AL MIDDLEWARE.



Solución típica Cliente/Servidor basada íntegramente en Java.

No requiere instalación en el cliente.

TIPO 4: DRIVER NATIVO ESCRITO COMPLETAMENTE EN JAVA, ESPECÍFICO DE CADA SGBD.



Ventajas: solución 100% java e independiente de la máquina donde se va a ejecutar. Puede no requerir ninguna configuración adicional del lado del cliente. Mejor rendimiento.

Inconveniente: el cliente está ligado a un SGBD concreto, ya que los protocolos de red son distintos para cada uno: Oracle, SQL Server ...

6.4. VENTAJAS JDBC VS ODBC.

1. ODBC es una interfaz escrita en lenguaje C, que al no ser un lenguaje portable, haría que las aplicaciones Java perdieran esta cualidad.
2. Los drivers JDBC son autoinstalables, portables y seguros, mientras los ODBC hay que instalarlos manualmente en cada máquina.

BIBLIOGRAFÍA

- C. BATINI, S. CERI, S.B. NAVATHE (1994): *Diseño Conceptual de Bases de Datos. Un enfoque de entidades-interrelaciones*. Addison-Wesley / Díaz de Santos.
- T. CONNOLLY, C. BEGG, A. STRACHAN (1996): *Database Systems. A Practical Approach to Design, Implementation and Management*. Addison-Wesley. Segunda Edición en 1998.
- C.J. DATE (1993): *Introducción a los Sistemas de Bases de Datos*. Volumen I, Quinta Edición. Addison-Wesley Iberoamericana. Sexta Edición en 1995 (en inglés, por Addison-Wesley).
- R. ELMASRI, S.B. NAVATHE (1997): *Sistemas de Bases de Datos. Conceptos fundamentales*. Segunda Edición. Addison-Wesley Iberoamericana. Tercera Edición en 1999 (en inglés, por Addison-Wesley).

- M.J. FOLK, B. ZOELICK (1992): *File Structures*. Segunda Edición. Addison-Wesley.
- G.W. HANSEN, J.V. HANSEN (1997): *Diseño y Administración de Bases de Datos*. Segunda Edición. Prentice Hall.
- M.J. HERNÁNDEZ (1997): *Database Design for Mere Mortals*. Addison-Wesley Developers Press.
- POSTGRESQL. *Manual del Programador*. 1999.

