UNIVERSIDAD DE CASTILLA LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA
BASES DE DATOS

ARQUITECTURAS DE SISTEMAS DE BASES DE DATOS

Óscar González Martín (Gestión)
Profesor: Francisco Ruiz González
1999/2000

INDICE

CAPÍTULO 4: ARQUITECTURA DE BASES DE DATOS DISTRIBUIDAS.

INTRODUCCIÓN

1. INTRODUCCIÓN.

En el trabajo que presentamos a continuación vamos a hablar sobre las diferentes arquitecturas de sistemas de bases de datos. En él abordaremos las siguientes arquitecturas:
Sistemas de bases de datos cliente-servidor.
Sistemas de bases de datos distribuidos.
También veremos cómo trata Oracle 8 la arquitectura cliente-servidor.

Vamos a comenzar haciendo un breve repaso a la evolución de la tecnología y los sistemas de bases de datos. Al comienzo del proceso de datos, durante los cincuenta y el comienzo de los sesenta, la regla era el tratamiento de archivos secuenciales. Todos los datos se almacenaban en archivos secuenciales, que exigían el tratamiento de archivos completos por los programas de aplicación. Durante los sesenta, debido a que el almacenamiento en disco utilizando el acceso directo llegó a estar ampliamente disponible, el procesamiento de archivos de acceso aleatorio llegó a ser factible y popular. Este método permitió el acceso directo a datos específicos en un archivo.

En la medida en que los sistemas computacionales de procesamiento de datos se hicieron más importantes, las empresas comenzaron a reconocer que la información era un recurso corporativo de valor considerable. Estas percibieron más y más que los datos necesarios para contestar numerosas preguntas estaban disponibles en sus archivos de procesamiento de datos. Como consecuencia, comenzaron a presionar a los sistemas de información para la gestión en cuanto a la utilización de la potencia del computador para producir información a partir de los datos corporativos. Esto inició la demanda de los sistemas de bases de datos, los que garantizarían más efectivamente el acceso a los datos y su manipulación.

A mediados de los sesenta se introdujeron los primeros sistemas de bases de datos, cuyo fundamento era una estructura jerárquica de los datos. Estos sistemas permitieron la recuperación de múltiples registros asociados con un registro único de otro archivo. Inmediatamente después, se desarrollaron los sistemas de base de datos en redes que soportaron interrelaciones entre registros de archivos diferentes mucho más complejas. Ambos modelos de base de datos, el jerárquico y el de red, requirieron el uso de punteros físicos predefinidos para enlazar los registros relacionados.

En 1970, Codd revolucionó el pensamiento en la industria de las bases de datos. El enfoque de Codd proponía el acceso y la manipulación de los datos únicamente desde el punto de vista de sus características lógicas. Durante los años setenta y ochenta se desarrollaron numerosos sistemas de bases de datos relacionales y, en la actualidad, éstos dominan el mercado comercial.

En años recientes han proliferado los computadores personales en los puestos de trabajo, por lo que se han desarrollado las redes de computadores, permitiendo a los usuarios de estos computadores compartir recursos. Un computador, que funciona como servidor de una red, garantiza el acceso a la base de datos desde las estaciones de trabajo en estos puestos, permitiendo una división poderosa y eficiente de la tarea: El servidor recupera los datos, los que la máquina cliente solicitante procesa y presenta en pantalla para su manipulación por parte del usuario final. Las redes de computadores en ambiente cliente/servidor han desarrollado un grado alto de sofisticación y se encuentran cada vez con más frecuencia en las empresas comerciales.

Desde el punto de vista conceptual, un sistema de base de datos en una organización grande está formado por el hardware, el software, los datos y las personas. La configuración del hardware comprende uno o más computadores, unidades de disco, terminales, impresoras, unidades de cinta magnética, conexiones de red y otros dispositivos físicos. El software incluye un sistema de gestión de bases de datos

(SGBD) y los programas de aplicación utilizan el SGBD para tener acceso y manipular la base de datos. Los datos, que representan los hechos importantes para la organización, radican físicamente en el disco, pero se estructuran lógicamente de forma que se logre un acceso fácil y eficiente.

2. SISTEMAS DE BASES DE DATOS.

Un sistema de gestión de bases de datos (SGBD o DBMS 'Database Management System") consiste en una colección de datos interrelacionados y un conjunto de programas que permiten a los usuarios acceder y modificar dichos datos. La colección de datos se denomina base de datos El primer objetivo de un SGBD es proporcionar un entorno que sea tanto práctico como eficiente de usar en la recuperación y el almacenamiento de la información de la base de datos. Otro de los objetivos principales de un SGBD es proporcionar al usuario una visión abstracta de la información, es decir, el sistema oculta detalles como los relativos a la forma de almacenar y mantener los datos, de tal forma que para que el sistema sea útil la información ha de recuperarse de forma eficiente.

La búsqueda de la eficiencia conduce al diseño de estructuras complejas para usuarios sin conocimientos de computación, para lo cual esta complejidad ha de estar oculta. Para poder lograr lo anterior es necesario definir los distintos niveles de abstracción de una base de datos, lo que constituirá el marco necesario para identificar las diferentes funciones que han de cumplir estos sistemas.

2.1 El estándar ANSI/SPARC.

El objetivo principal de la arquitectura ANSI/SPARC es definir un SGBD con el máximo grado de independencia, separando las aplicaciones de usuario y la base de datos física. Para ello se utilizan tres niveles de abstracción conocidos como interno, conceptual y externo.

1. El **nivel interno** es el más cercano a la máquina. Es una representación a bajo nivel de la BD en la que se define la forma en la que los datos se almacenan físicamente en la máquina. Se definen características como los dispositivos en donde se almacenan los datos, el espacio que se reserva, las estrategias de acceso, la creación de ficheros de índices, etc. Es dependiente de la máquina en que se vaya a instalar la BD, del sistema operativo que exista, etc.
2. El **nivel conceptual** tiene un esquema conceptual, que describe la estructura de los datos que van a ser almacenados en la base de datos. El esquema conceptual esconde los detalles del almacenamiento físico y se concentra en describir entidades, tipos de datos, relaciones, operaciones de usuario y restricciones .
3. El **nivel externo** o **nivel de vista** incluye varios esquemas externos o vistas de usuario. Cada esquema externo describe la parte de la base de datos en la que está interesado un grupo de usuarios en particular y esconde el resto de la base de datos para esos usuarios. La información se manipula sin saber cómo está almacenada internamente (nivel interno) ni su organización (nivel conceptual).

Existirán muchas vistas externas distintas, cada una formada por una representación más o menos abstracta (registros y campos lógicos) de alguna parte de la base de datos total, y existirá sólo una vista conceptual formada por una representación igualmente abstracta de la base de datos en su totalidad (hay que recordar que a la mayoría de los usuarios no les interesará toda la base de datos, sino sólo una porción limitada de ella). De manera similar, habrá sólo una vista interna, la cual representará a toda la base de datos tal como está almacenada físicamente.

El Nivel Externo

El nivel externo es el más cercano a los usuarios, es decir, es el que se ocupa de la forma en la que los usuarios perciben los datos. El nivel externo es del usuario individual. Estos usuarios pueden ser o bien

programadores de aplicaciones o usuarios finales con conocimientos muy variables de informática. El administrador de la base de datos es un caso especial (también debe interesarse por los demás niveles de la arquitectura).

Cada usuario dispone de un lenguaje:

En el caso del programador de aplicaciones, dicho lenguaje será o bien un lenguaje de programación convencional, o bien un lenguaje de cuarta generación (4GL) específico para el sistema en cuestión.

Para el usuario final será o bien un lenguaje de consulta, o algún lenguaje de aplicación especial, quizá manejado mediante formas o menús, adaptado a los requerimientos de ese usuario y apoyado por algún programa de aplicación en línea (cuya función es servir a un usuario final que tiene acceso a la base de datos desde una terminal en línea).

El aspecto importante de todos estos lenguajes es que deben incluir un sublenguaje de datos, es decir, un subconjunto del lenguaje total que se ocupe de manera específica de los objetos y operaciones de la base de datos. Se dice que el sublenguaje de datos (DSL 'data sublanguage') está embebido (o inmerso) dentro del lenguaje anfitrión correspondiente. Este último se encarga de varios aspectos no relacionados con la base de datos, como por ejemplo variables locales (temporales), operaciones de cálculo, lógica condicional, etc. Un sistema dado puede permitir el empleo de varios lenguajes anfitriones y varios sublenguajes de datos. Un sublenguaje de datos en particular cuyo uso es posible en casi todos los sistemas relacionales actuales es el lenguaje SQL.

En principio, cualquier sublenguaje de datos es en realidad una combinación de por lo menos dos lenguajes subordinados: un lenguaje de definición de datos (DDL 'data definition language'), con el cual es posible definir o declarar los objetos de la base de datos, y un lenguaje de manipulación de datos (DML, 'data manipulation language') con el que es posible manipular o procesar dichos objetos.

Como ya se ha dicho, al usuario individual (en general), sólo le interesará una porción de la base de datos total; por añadidura, la forma como ese usuario percibe dicha porción casi siempre será un tanto abstracta comparada con el almacenamiento físico de los datos. El término ANSI/SPARC para la vista individual de un usuario es vista externa. Así, una vista externa es el contenido de la base de datos tal como lo percibe algún usuario determinado (es decir, para ese usuario la vista externa es la base de datos). Por ejemplo, un usuario del departamento de personal podría contemplar la base de datos como un conjunto de ocurrencias de registros de departamento unido a un conjunto de ocurrencias de registros de proveedor y de parte vistas por los usuarios del departamento de compras).

Toda vista externa se define mediante un esquema externo, que consiste básicamente en definiciones de cada uno de los diversos tipos de registros externos en esa vista externa. El esquema externo se escribe con la porción DDL del sublenguaje de datos del usuario (por ello se le denomina a ese DDL en ocasiones como DDL externo). Por ejemplo, el tipo de registro externo de empleado puede definirse como un campo de número de empleado de seis caracteres unido a un campo de salario de cinco dígitos, etc. Además, debe haber una definición de la correspondencia entre el esquema externo y el esquema conceptual subyacente.

El Nivel Conceptual

El nivel conceptual es un nivel de mediación entre el nivel interno y externo. La vista conceptual es una representación de toda la información contenida en la base de datos, también (como en el caso de una vista externa) en una forma un tanto abstracta si se compara con el almacenamiento físico de los datos. Además, puede ser muy diferente de la forma como percibe los datos cualquier usuario individual. A grandes rasgos, la vista conceptual debe ser un panorama de los datos "tal como son", y no como por fuerza los perciben los usuarios debido a las limitaciones del lenguaje o el equipo específicos utilizados, por ejemplo.

La vista conceptual se compone de varias ocurrencias de varios tipos de registro conceptual. Por ejemplo, puede estar formada por un conjunto de ocurrencias de registros de departamento unido un conjunto de ocurrencias de registro de empleado y a un conjunto de ocurrencias de registros de proveedor y a un conjunto de ocurrencia de registros de parte... Un registro conceptual no es por necesidad idéntico a un registro externo, por un lado, ni a un registro almacenado, por el otro.

La vista conceptual se define mediante un esquema conceptual, el cual incluye definiciones de cada uno de los tipos de registro conceptual. El esquema conceptual se escribe utilizando otro lenguaje de definición de datos, el DDL conceptual. Si ha de lograrse la independencia de los datos, esas definiciones en DDL conceptual no deberán implicar consideraciones de estructura de almacenamiento o de técnica de acceso. Si el esquema conceptual se hace en verdad independiente de los datos de esta manera, entonces los esquemas externos, definidos en términos del esquema conceptual, serán por fuerza también independientes de los datos.

Así pues, la vista conceptual es una vista del contenido total de la base de datos, y el esquema conceptual es una definición de esa vista. No obstante, sería engañoso sugerir que el esquema conceptual es sólo un conjunto de definiciones similar a las sencillas definiciones de registros encontradas por ejemplo en un programa en Cobol. Es de esperar que las definiciones en el esquema conceptual incluyan muchas características más, como son las verificaciones de seguridad y de integridad. Algunos expertos podrían llegar a sugerir que el objetivo primordial del esque conceptual es describir la empresa en su totalidad (no sólo los datos en sí, sino también la forma como se utilizan: cómo fluyen de un punto a otro dentro de la empresa, qué se hace con ellos en cada punto, qué controles de auditoría o de otro tipo deben aplicarse en cada punto, etc. Debe hacerse hincapié en que en ningún sistema actual es posible mantener realmente un nivel conceptual que se aproxime siquiera a ese grado de complejidad; en casi todos los sistemas existentes el esquema conceptual no es mucho más que una simple unión de todos los esquemas externos individuales, con la posible adición de algunas verificaciones sencillas de integridad y seguridad. Con todo, parece evidente que los sistemas del futuro llegarán a mantener niveles conceptuales mucho más complejos.

El Nivel Interno

El tercer nivel de la arquitectura es el nivel interno. La vista interna es una representación de bajo nivel de toda la base de datos; se compone de varias ocurrencias de varios tipos de registro interno. Este último término es el que utiliza ANSI/SPARC para referirse a la construcción que hemos estado llamando registro almacenado. La vista interna, por tanto, todavía está a un paso del nivel físico, ya que no manejo registros físicos (llamados también páginas o bloques), ni otras consideraciones específicas de los dispositivos como son los tamaños de cilindros o de pistas.

La vista interna se define mediante el esquema interno, el cual no sólo define los diversos tipos de registros almacenados sino también especifica que índices hay, cómo se representan los campos almacenados, en qué secuencia física se encuentran los registros almacenados, etc. El esquema interno se escribe con otro lenguaje más de definición de datos, el DDL interno.

En algunas situaciones excepcionales podría permitirse a los programas de aplicación operar directamente en el nivel interno en vez de hacerlo en el nivel externo. Esta práctica no es recomendable ya que representa un riesgo para la seguridad (ya que pasan por alto las verificaciones de seguridad) y para la integridad (hace lo mismo), y el programa será en extremo dependiente de los datos; sin embargo, en ciertos casos puede ser la única forma de obtener la función o desempeño deseados, del mismo modo como el usuario de un lenguaje de programación de alto nivel puede verse obligado en ocasiones a descender al lenguaje ensamblador para satisfacer ciertos objetivos.

Correspondencias, asociaciones o ligaduras (mappings)

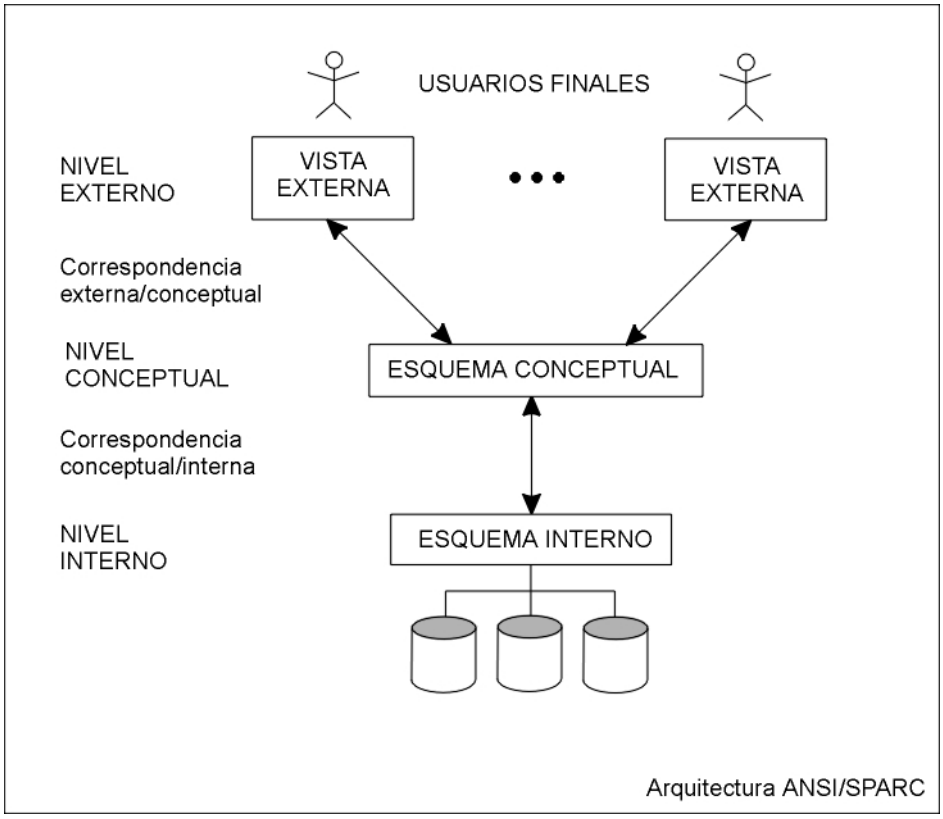
Para describir un mismo grupo de datos, un sistema puede gestionar varios niveles de esquemas, para lo cual el DBMS debe poder garantizar la transferencia de los datos desde el formato correspondiente

de un nivel al formato correspondiente a otro nivel; este proceso se denomina transformación de datos o mapping.

Hay dos niveles de correspondencia en la arquitectura ANSI/SPARC: uno entre los niveles externo y conceptual del sistema, y otro entre los niveles conceptual e interno. La correspondencia conceptual/interna es la que existe entre las vista conceptual y la base de datos almacenada; especifica cómo se representan los registros y campos conceptuales en el nivel interno. Si se modifica la estructura de la base de datos almacenada (es decir, si se altera la definición de la estructura de almacenamiento), la correspondencia conceptual/interna deberá modificarse también de acuerdo con ello, para que no varíe el esquema conceptual (el administrador de la base de datos se debe encargar de controlar tales modificaciones). Dicho de otra manera, los efectos de las alteraciones deberán aislarse por debajo del nivel conceptual, a fin de conservar la independencia de los datos.

La correspondencia externa/conceptual es la que existe entre una determinada vista externa y la vista conceptual. Las diferencias que pueden existir entre estos dos niveles son similares a las que pueden existir entre la vista conceptual y la base de datos almacenada. Por ejemplo, los campos pueden tener distintos tipos de datos, los nombre de los campos y los registros pueden diferir, pueden combinarse varios campos conceptuales para formar un solo campo externo (virtual), etc. Puede existir cualquier cantidad de vistas externas; cualquier número de usuarios puede compartir una determinada vista externa.

Algunos sistemas permiten expresar la definición de una vista externa en términos de otras (de hecho, a través de una correspondencia externa/externa), en vez de requerir siempre una definición explícita de la correspondencia respecto al nivel conceptual, cosa que resulta útil si existe una relación muy grande entre varias vistas externas. Los sistemas relacionales en particular casi siempre permiten hacer esto.



Arquitectura ANSI/SPARC

2.2 El sistema de gestión de bases de datos.

Un Sistema de Gestión de Bases de Datos (SGBD) es el conjunto de programas que permiten definir, manipular y utilizar la información que contienen las bases de datos, realizar todas las tareas de administración necesarias para mantenerlas operativas, mantener su integridad, confidencialidad y seguridad. Una BD nunca se accede o manipula directamente sino a través del SGBD. Se puede considerar al SGBD como el interfaz entre el usuario y la BD.

El funcionamiento del SGBD está muy interrelacionado con el del Sistema Operativo, especialmente con el sistema de comunicaciones. El SGBD utilizará las facilidades del sistema de comunicaciones para recibir las peticiones del usuario (que puede estar utilizando un terminal físicamente remoto) y para devolverle los resultados. Conceptualmente, lo que ocurre es lo siguiente:

1. Un usuario hace una petición de acceso, usando algún lenguaje en particular (normalmente SQL).
2. El SGBD intercepta esa petición y la analiza.
3. El SGBD inspecciona el esquema externo de ese usuario, la correspondencia externa/conceptual, el esquema conceptual, la correspondencia conceptual/interna, y la definición de la estructura de almacenamiento.
4. El SGBD ejecuta las operaciones necesarias en la base de datos almacenada.

El SGBD tiene las siguientes funciones:

Definición de datos

El SGBD debe ser capaz de aceptar definiciones de datos (esquemas externos, el esquema conceptual, el esquema interno, y todas las correspondencias asociadas) en versión fuente y convertirlas en la versión objeto apropiada. Dicho de otro modo, el SGBD debe incluir componentes procesadores de lenguajes para cada uno de los diversos lenguajes de definición de datos (DDL). El SGBD también debe entender las definiciones en DDL, en el sentido en que, por ejemplo, entiende que los registros externos 'Empleado' contienen un campo 'Salario'; y debe poder utilizar estos conocimientos para interpretar y responder las solicitudes de los usuarios (por ejemplo una consulta de todos los empleados cuyo salario sea inferior a 100.000 pts).

Manipulación de datos

El SGBD debe ser capaz de atender las solicitudes del usuario para extraer, y quizá actualizar, datos que ya existen en la base de datos, o para agregar en ella datos nuevos. Dicho de otro modo, el SGBD debe incluir un componente procesador de lenguaje de manipulación de datos (DML).

Seguridad e integridad de los datos

El SGBD debe supervisar las solicitudes de los usuarios y rechazar los intentos de violar las medidas de seguridad e integridad definidas por el administrador de la base de datos.

Recuperación y concurrencia de los datos

El SGBD (o en su defecto algún componente de software relacionado con él, al que normalmente se le denomina administrados de transacciones) debe cuidar del cumplimiento de ciertos controles de recuperación y concurrencia.

Diccionario de datos.

El SGBD debe incluir una función de diccionario de datos. Puede decirse que el diccionario de datos es una base de datos (del sistema, no del usuario). El contenido del diccionario puede considerarse como "datos acerca de los datos" (metadatos), es decir, definiciones de otros objetos en el sistema, y no sólo datos en bruto. En particular, en el diccionario de datos se almacenarán físicamente todos los diversos esquemas y correspondencias (externos, conceptuales,etc.) tanto en sus versiones fuente como en las versiones objeto. Un diccionario completo incluirá también referencias cruzadas para indicar, por ejemplo, qué bases de datos utilizan los programas, que informes requieren los usuarios, qué terminales están conectadas al sistema... Es más, el diccionario podría (y quizá debería) estar integrado a la base de datos a la cual define, e incluir por tanto su propia definición. Deberá ser posible consultar el diccionario igual que cualquier otra base de datos de modo que se puede saber, por ejemplo, qué programas o usuarios podrían verse afectados por alguna modificación propuesta para el sistema.

Como conclusión, podemos decir que el SGBD constituye la interfaz entre el usuario y el sistema de bases de datos. La interfaz del usuario puede definirse como una frontera del sistema, más allá de la cual todo resulta invisible para el usuario. Por definición, entonces, la interfaz del usuario está en el nivel externo.

2.2.1.- Características de extensibilidad de los SGBD

Los SGBD deben reunir una serie de características que contemplen las nuevas funcionalidades que deben proporcionar en estos momentos. Dichas características son:

Soporte ODBC

ODBC (siglas que significan Open DataBase Conectivity, Conectividad Abierta de Bases de Datos) se define como un método común de acceso a bases de datos, diseñado por Microsoft para simplificar la comunicación en Bases de Datos Cliente/Servidor. ODBC consiste en un conjunto de llamadas de bajo nivel que permite a las aplicaciones en el cliente intercambiar instrucciones con las aplicaciones del servidor y compartir datos, sin necesidad de conocer nada unas respecto a las otras. Las aplicaciones emplean módulos, llamados controladores de bases de datos, que unen la aplicación con el SGBD concreto elegido. Se emplea el SQL como lenguaje de acceso a los datos. El SGBD debe proporcionar los controladores adecuados para poder ser empleados por los distintos lenguajes de programación que soporten ODBC.

Orientación a objetos

Los SGDB relacionales tradicionales sólo pueden almacenar y tratar con números y cadenas de caracteres. Las mejoras en el terreno de la multimedia obligan a que las aplicaciones desarrolladas actualmente precisen cada vez más almacenar, junto con la información numérica y de caracteres, tipos de datos más complejos que permitan gestionar objetos de sonido, imágenes, vídeos, etc. Algunos SGBD relacionales avanzaron en este sentido dando cabida en sus BD a tipos de datos binarios (donde se puede guardar código binario, que es el que forma los objetos de sonido, imágenes, programas ejecutables, etc.), pero esto no es suficiente. La aparición de SGBD relacionales Orientados a Objetos (SGBDROO) proporcionan toda la potencia y robustez de los SGBD relacionales, y al mismo tiempo, permiten gestionar objetos de un modo nativo, así como los campos numéricos y de caracteres que se han visto recogidos tradicionalmente. Los SGBDROO cuentan con todas las posibilidades de un motor de consultas SQL clásico, pero el lenguaje puede manipular tipos definidos por el usuario, de la misma manera que gestiona los tipos predefinidos de los sistemas más antiguos. Por lo tanto, se trata de un SGBD relacional, pero extendido y ampliado de manera que soporte la gestión de objetos. Por otra parte, la tendencia a la

generación de aplicaciones distribuidas, donde los usuarios, datos y componentes de la aplicación están físicamente separados, facilita e impulsa el uso de SGDROO, pues los objetos y las aplicaciones distribuidas están "hechos el uno para el otro".

Conectividad en Internet

Los distintos SGDB existentes incorporan en sus últimas versiones software de tipo middleware (capa de software que se sitúa sobre el SGBD) para añadir conectividad a la base de datos a través de Internet. Microsoft ha desarrollado los ADO (Access Database Object, Objetos de Acceso a Bases de Datos) que, incorporados en scripts dentro de páginas Web en HTML, proporcionan conexión con Bases de Datos, tanto locales como remotas, empleando ODBC. Los middleware desarrollados en los distintos SGBDs suelen emplear ODBC (o JDBC, conectividad abierta de bases de datos preparada para el lenguaje Java) para conectar con la BD, junto con diversos conjuntos de herramientas para facilitar al usuario la implementación de la comunicación con la BD a través de Internet.

Soporte de estándares objetuales

Hay varios estándares de objetos diseñados para proporcionar una guía en el diseño y desarrollo de aplicaciones distribuidas que trabajen con BD relaciones con orientación a objetos. Los SGBDs actuales hacen uso de software del tipo middleware que asumen las tareas de servicio de transacciones de objeto siguiendo alguno de los estándares de objetos existentes. Los principales estándares de objeto son:

- CORBA (Common Object Broker Architecture, o Arquitectura común de gestores de solicitudes de objetos), del Object Management Group (OMG).

- DCOM (Distributed Component Model) de Microsoft.

- Java Remote Method Invocation de Sun.

Los actuales SGBD proporcionan soporte, como mínimo, a CORBA y DCOM.

Data Mining, Data Warehousing, OLAP

Los SGBD deben incorporar una serie de herramientas que permitan, de forma cómoda, sencilla e intuitiva, la extracción y disección-minería de datos (Data Mining), y soporte para OLAP (OnLine Analytical Processing, Procesamiento Análitico de Datos En Vivo), que se trata de una categoría de las nuevas tecnologías del software que permite obtener y extraer información mediante un complejo análisis y procesamiento del contenido de una Base de Datos, todo ello en tiempo real.

También deben proporcionar una estabilidad y robustez cada vez mejores, que permitan mejorar los almacenes de datos (Data Warehousing), mercados de datos (Data Marts) y Webs de datos, procesos de transacciones y otras aplicaciones de misión crítica.

3.- ARQUITECTURAS DE SISTEMAS DE BASES DE DATOS.

Anteriormente se ha analizado con cierto detalle la arquitectura ANSI/SPARC para sistemas de bases de datos. En esta sección vamos a examinar los sistemas de bases de datos desde un punto de vista diferente.

La arquitectura de un sistema de base de datos está influenciada en gran medida por el sistema informático subyacente en el que se ejecuta el sistema de base de datos. En la arquitectura de un sistema de base de datos se reflejan aspectos como la conexión en red, el paralelismo y la distribución:

- La arquitectura centralizada es la más clásica. En ella, el SGBD está implantado en una sola plataforma u ordenador desde donde se gestiona directamente, de modo centralizado, la totalidad de los recursos. Es la arquitectura de los centros de proceso de datos tradicionales. Se basa en tecnologías sencillas, muy experimentadas y de gran robustez.

- La conexión en red de varias computadoras permite que algunas tareas se ejecuten en un sistema servidor y que otras se ejecuten en los sistemas clientes. Esta división de trabajo ha conducido al desarrollo de sistemas de bases de datos cliente-servidor.

- La distribución de datos a través de las distintas sedes o departamentos de una organización permite que estos datos residan donde han sido generados o donde son más necesarios, pero continuar siendo accesibles desde otros lugares o departamentos diferentes. El hecho de guardar varias copias de la base de datos en diferentes sitios permite que puedan continuar las operaciones sobre la base de datos aunque algún sitio se vea afectado por algún desastre natural, como una inundación, un incendio o un terremoto. Se han desarrollado los sistemas de bases de datos distribuidos para manejar datos distribuidos geográfica o administrativamente a lo largo de múltiples sistemas de bases de datos.

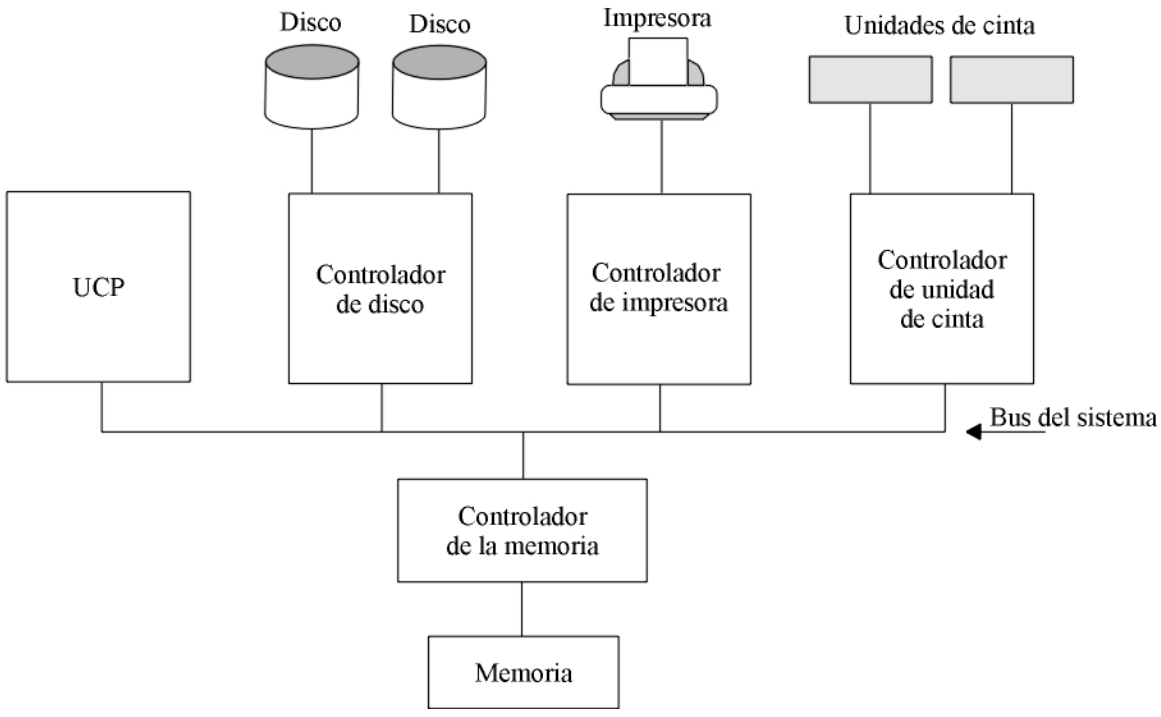
- El procesamiento paralelo dentro de una computadora permite acelerar las actividades del sistema de base de datos, proporcionando a las transacciones unas respuestas más rápidas, así como la capacidad de ejecutar más transacciones por segundo. Las consultas pueden procesarse de manera que se explote el paralelismo ofrecido por el sistema informático subyacente. La necesidad del procesamiento paralelo de consultas ha conducido al desarrollo de los sistemas de bases de datos paralelos.

No debe confundirse el SGBD con la arquitectura que se elige para implantarlo. Algunos SGBD sólo se pueden implantar en una de las arquitecturas y otros en todas ellas.

3.1. Arquitectura centralizada

Los sistemas de bases de datos centralizados son aquellos que se ejecutan en un único sistema informático sin interaccionar con ninguna otra computadora. Tales sistemas comprenden el rango desde los sistemas de bases de datos monousuario ejecutándose en computadoras personales hasta los sistemas de bases de datos de alto rendimiento ejecutándose en grandes sistemas.

Un sistema informático centralizado



Una computadora moderna de propósito general consiste en una o unas pocas CPU's y un número determinado de controladores para los dispositivos que se encuentren conectados a través de un bus común, el cual proporciona acceso a la memoria compartida. Las CPU's poseen memorias caché locales donde se almacenan copias de ciertas partes de la memoria para acelerar el acceso a los datos. Cada controlador de dispositivo se encarga de un tipo específico de dispositivos (por ejemplo, una unidad de disco, una tarjeta de sonido o un monitor). La CPU y los controladores de dispositivo pueden ejecutarse concurrentemente, compitiendo así por el acceso a la memoria. La memoria caché reduce la disputa por el acceso a la memoria, ya que la CPU necesita acceder a la memoria compartida un número de veces menor.

Se distinguen dos formas de utilizar las computadoras: como *sistemas monousuario* o como *sistemas multiusuario*. En la primera categoría están las computadoras personales y las estaciones de trabajo. Un sistema monousuario típico es una unidad de sobremesa utilizada por una única persona que dispone de una sola CPU, de uno o dos discos fijos y que trabaja con un sistema operativo que sólo permite un único usuario. Por el contrario, un sistema multiusuario típico tiene más discos y más memoria, puede disponer de varias CPU y trabaja con un sistema operativo multiusuario. Se encarga de dar servicio a un gran número de usuarios que están conectados al sistema a través de terminales. Estos sistemas se denominan con frecuencia sistemas servidores.

Normalmente, los sistemas de bases de datos diseñados para funcionar sobre sistemas monousuario, como las computadoras personales, no suelen proporcionar muchas de las facilidades que ofrecen los sistemas multiusuario. En particular, no tienen control de concurrencia, que no es necesario cuando solamente un usuario puede generar modificaciones. Las facilidades de recuperación en estos sistemas, o no existen o son primitivas; por ejemplo, realizar una copia de seguridad de la base de datos antes de cualquier modificación. La mayoría de estos sistemas no admiten SQL y proporcionan un lenguaje de consulta muy simple, que en algunos casos es una variante de QBE (Query By Example).

Aunque hoy en día las computadoras de propósito general tienen varios procesadores, utilizan paralelismo de grano grueso, disponiendo de unos pocos procesadores (normalmente dos o cuatro) que comparten la misma memoria principal. Las bases de datos que se ejecutan en tales máquinas habitualmente no intentan dividir una consulta simple entre los distintos procesadores, sino que ejecutan cada consulta en un único procesador, posibilitando la concurrencia de varias consultas. Así, estos sistemas soportan una mayor productividad, es decir, permiten ejecutar un mayor número de transacciones por segundo, a pesar de que cada transacción individualmente no se ejecuta más rápido.

Las bases de datos diseñadas para las máquinas monoprocesador ya disponen de multitarea, permitiendo que varios procesos se ejecuten a la vez en el mismo procesador, usando tiempo compartido, mientras que de cara al usuario parece que los procesos se están ejecutando en paralelo. De esta manera, desde un punto de vista lógico, las máquinas paralelas de grano grueso parecen ser idénticas a las máquinas monoprocesador, y pueden adaptarse fácilmente los sistemas de bases de datos diseñados para máquinas de tiempo compartido para que puedan ejecutarse sobre máquinas paralelas de grano grueso.

Por el contrario, las máquinas paralelas de grano fino tienen un gran número de procesadores y los sistemas de bases de datos que se ejecutan sobre ellas intentan paralelizar las tareas simples (consultas, por ejemplo) que solicitan los usuarios.

ARQUITECTURA DE BASES DE DATOS CLIENTE/SERVIDOR

1. INTRODUCCIÓN

Con el aumento de la velocidad y potencia de las computadoras personales y el decremento en su precio, los sistemas se han ido distanciando de la arquitectura centralizada. Los terminales conectados a un sistema central han sido suplantados por computadoras personales. De igual forma, la interfaz de usuario, que solía estar gestionada directamente por el sistema central, está pasando a ser gestionada cada vez más por las computadoras personales. Como consecuencia, los sistemas centralizados actúan hoy como sistemas servidores que satisfacen las peticiones generadas por los sistemas clientes.

La computación cliente/servidor es la extensión lógica de la programación modular. El supuesto principal de la programación modular es la división de un programa grande en pequeños programas (llamados módulos), siendo más fáciles el desarrollo y la mantenibilidad (divide y vencerás).

Cualquier LAN (red de área local) puede ser considerada como un sistema cliente/servidor, desde el momento en que el cliente solicita servicios como datos, ficheros o imprimir desde el servidor. Cuando un usuario se conecta a Internet, interactua con otros computadores utilizando el modelo cliente/servidor. Los recursos de Internet son proporcionados a través de computadores host, conocidos como servidores. El servidor es el computador que contiene información (bases de datos, ficheros de texto...). El usuario, o cliente, accede a esos recursos vía programas cliente (aplicaciones) que usan TCP/IP para entregar la información a su computadora.

Según esta definición, los sistemas cliente/servidor no están limitados a aplicaciones de bases de datos. Cualquier aplicación que tenga una interfaz de usuario (front-end, sección frontal o parte cliente) que se ejecute localmente en el cliente y un proceso que se ejecute en el servidor (back-end, sección posterior, o sistema subyacente) está en forma de computación cliente/servidor.

Conceptualmente, las plataformas cliente/servidor son parte del concepto de sistemas abiertos, en el cual todo tipo de computadores, sistemas operativos, protocolos de redes y otros, software y hardware, pueden interconectarse y trabajar coordinadamente para lograr los objetivos del usuario. Sin embargo, en la práctica, los problemas de alcanzar tal variedad de sistemas operativos, protocolos de redes, sistemas de base de datos y otros, que trabajen conjuntamente pueden ser muchos. El objetivo de los sistemas abiertos consiste en lograr la interoperabilidad, que es el estado de dos o más sistemas heterogéneos comunicándose y contribuyendo cada uno a alguna parte del trabajo que corresponde a una tarea común.

En cierto sentido, el enfoque cliente/servidor es la culminación de una percepción temprana de la potencia del cálculo distribuida conjuntamente con el control y el acceso a los datos inherentes a un computador centralizado.

2. CARACTERÍSTICAS DE UN SISTEMA CLIENTE/SERVIDOR.

Un sistema cliente/servidor es aquel en el que uno o más clientes y uno o más servidores, conjuntamente con un sistema operativo subyacente y un sistema de comunicación entre procesos, forma un sistema compuesto que permite cómputo distribuido, análisis, y presentación de los datos. Si existen múltiples servidores de procesamiento de base de datos, cada uno de ellos deberá procesar una base de datos

distinta, para que el sistema sea considerado un sistema cliente/servidor. Cuando dos servidores procesan la misma base de datos, el sistema ya no se llama un sistema cliente/servidor, sino que se trata de un sistema de base de datos distribuido.

Los clientes, a través de la red, pueden realizar consultas al servidor. El servidor tiene el control sobre los datos; sin embargo los clientes pueden tener datos privados que residen en sus computadoras. Las principales características de la arquitectura cliente/servidor son:

- El servidor presenta a todos sus clientes una interfaz única y bien definida.
- El cliente no necesita conocer la lógica del servidor, sólo su interfaz externa.
- El cliente no depende de la ubicación física del servidor, ni del tipo de equipo físico en el que se encuentra, ni de su sistema operativo.
- Los cambios en el servidor implican pocos o ningún cambio en el cliente.

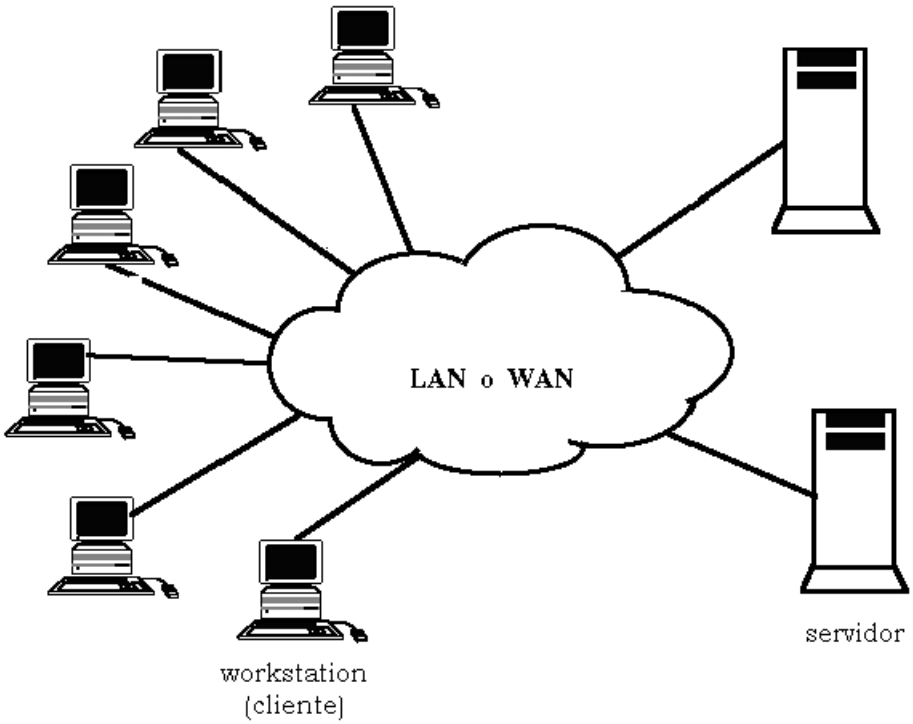
Como ejemplos de clientes pueden citarse interfaces de usuario para enviar comandos a un servidor, APIs (Aplication Program Interface) para el desarrollo de aplicaciones distribuidas, herramientas en el cliente para acceder a servidores remotos (por ejemplo, servidores de SQL) o aplicaciones que solicitan acceso a servidores para algunos servicios.

Como ejemplos de servidores pueden citarse servidores de ventanas como X-windows, servidores de archivos como NFS, servidores para el manejo de bases de datos (como los servidores de SQL), servidores de diseño y manufactura asistidos por computador, etc.

3. PARTES DE UN SISTEMA CLIENTE/SERVIDOR

Los principales componentes de un sistema cliente/servidor son:
- El núcleo (back-end o sección posterior). Es el SGBD propiamente (servidor).
- El interfaz (front-end o sección frontal). Aplicaciones que funcionan sobre el SGBD (cliente).

.



Ambiente Cliente/Servidor genérico

La diferencia entre la computación cliente/servidor y la computación centralizada multiusuario es que el cliente no es un terminal "tonto". El computador cliente tiene su propio sistema operativo y puede manejar entradas (teclado, ratón, etc...) y salidas (pantalla, impresora local, sonido, etc...) sin el servidor. El papel del servidor es esperar pasivamente la petición de servicio del cliente. Esta distribución del proceso permite al cliente ofrecer un ambiente de trabajo más amigable que un terminal "tonto" (interfaz de usuario gráfica, aplicaciones locales, ratón, etc...) y permite al servidor ser menos complejo y caro que los sistemas mainframe. El conjunto de la computación cliente/servidor conduce a un ambiente flexible y dinámico.

La parte cliente de la aplicación maneja la entrada de datos, acepta consultas de los usuarios y muestra los resultados. La parte cliente no procesa las consultas. En su lugar, envía la consulta del usuario al computador servidor, donde la parte servidor de la aplicación procesa la consulta. El servidor devuelve los resultados al cliente, que es quien se las muestra al usuario.

3.1. La sección frontal.

Las secciones frontales son las diversas aplicaciones ejecutadas dentro del SGBD, tanto las escritas por los usuarios como las "integradas" que son las proporcionadas por el proveedor del SGBD o bien por otros proveedores de programas (aunque para la sección posterior no existe diferencia entre las aplicaciones escritas por los usuarios y las integradas, ya que todas utilizan la misma interfaz con la sección posterior).

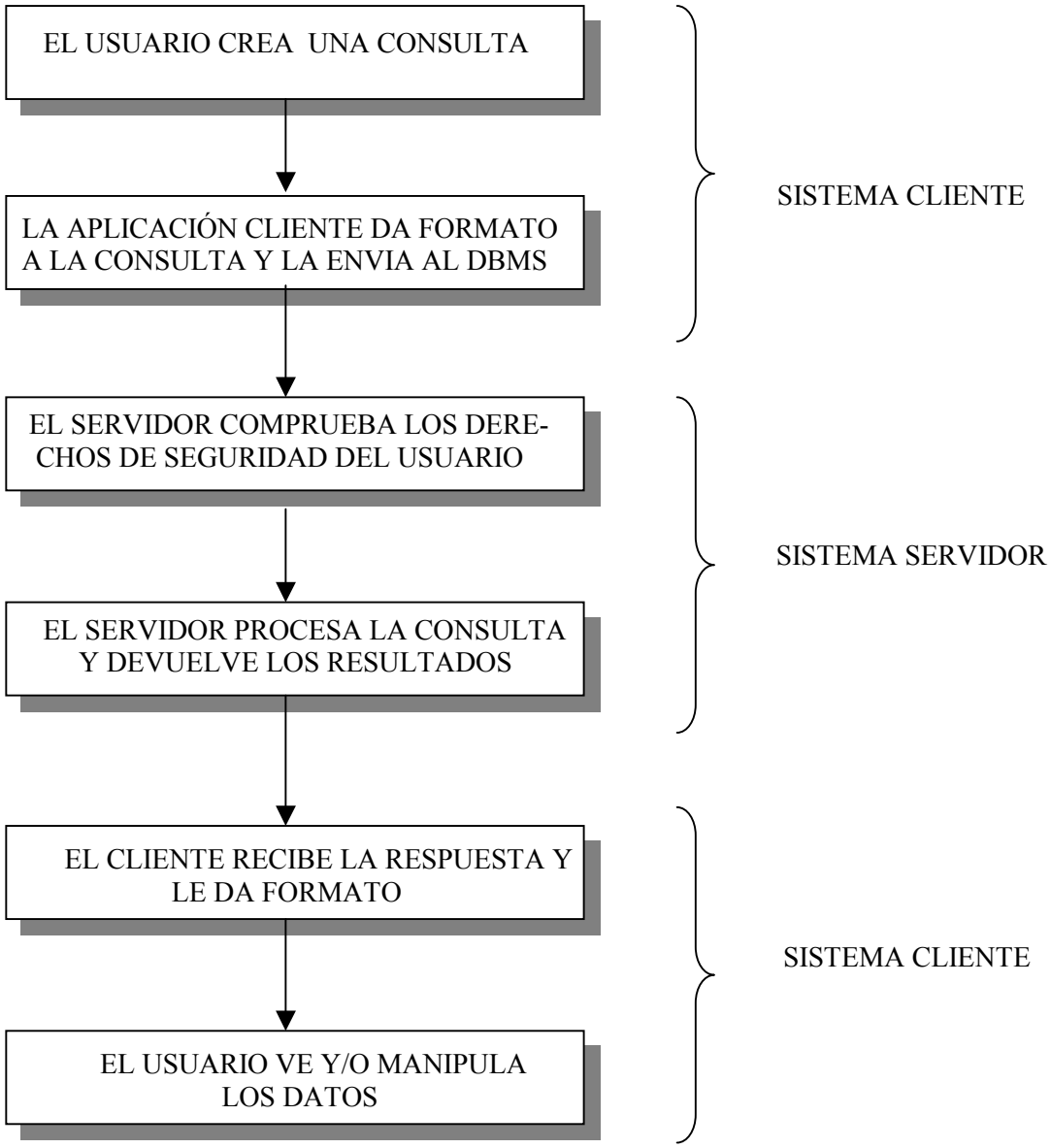
3.1.1. Funciones del cliente

Administrar la interfaz gráfica de usuario.
Aceptar datos del usuario.
Procesar la lógica de la aplicación.

Generar las solicitudes para la base de datos.
Transmitir las solicitudes de la base de datos al servidor.
Recibir los resultados del servidor.
Dar formato a los resultados.

3.1.2 Cómo trabaja la sección frontal

La secuencia de eventos que tiene lugar cuando el usuario accede al servidor de la base de datos se puede generalizar en 6 pasos básicos ilustrados en la figura. Para mayor simplicidad, el término consulta representa cualquier acción que el usuario pueda hacer en la base de datos, como actualizar, insertar, borrar o pedir datos de la base de datos.



Primero, el usuario crea la consulta. Puede ser una consulta creada en el instante o puede ser una consulta preprogramada o almacenada anteriormente. Después la aplicación cliente convierte la consulta al SQL usado por el servidor de la base de datos y la envía a través de la red al servidor.

El servidor verifica que el usuario tiene los derechos de seguridad apropiados a la consulta de datos requerida. Si es así, verifica la consulta y envía los datos apropiados de vuelta al cliente. La aplicación cliente recibe la respuesta y le da formato para presentarlo al usuario.

Finalmente, el usuario ve la respuesta en la pantalla y puede manipular los datos, o modificar la consulta y empezar el proceso de nuevo.

3.1.3. Tipos de aplicaciones cliente.

Las aplicaciones pueden dividirse en varias categorías más o menos bien definidas:

En primer lugar, aplicaciones escritas por los usuarios. Casi siempre se trata de programas comunes de aplicación, escritos (normalmente) en un lenguaje de programación convencional (por ejemplo Cobol), o bien en algún lenguaje propio (como Focus), aunque en ambos casos el lenguaje debe acoplarse de alguna manera con un sublenguaje de datos apropiado.

En segundo lugar, aplicaciones suministradas por los proveedores (herramientas). El objetivo general de estas herramientas es ayudar en el proceso de creación y ejecución de otras aplicaciones, o sea, aplicaciones hechas a la medida para alguna tarea específica (aunque la aplicación creada quizá no parezca una aplicación en el sentido convencional; de hecho, la verdadera razón para utilizar herramientas es que los usuarios, sobre todo los finales, puedan crear aplicaciones sin tener que escribir programas convencionales). Por ejemplo, una de las herramientas suministradas por los proveedores sería un procesador de lenguaje de consulta, cuyo propósito es desde luego permitir a los usuarios finales hacer consultas al sistema. En esencia, cada una de esas consultas no es más que una pequeña aplicación hecha a la medida para realizar alguna función de aplicación específica.

A su vez, las herramientas suministradas por los proveedores se dividen en varias clases distintas:

- Procesadores de lenguajes de consulta
- Generadores de informes
- Subsistemas de gráficas para negocios
- Hojas electrónicas de cálculo
- Procesadores de lenguajes naturales
- Paquetes estadísticos
- Herramientas para administrar copias
- Generadores de aplicaciones (incluyendo procesador de lenguajes de cuarta generación o 4GL)
- Otras herramientas para desarrollar aplicaciones, incluyendo productos para ingeniería de software asistida por computador (CASE).

3.2 La sección posterior.

La sección posterior es el SGBD en sí. Permite llevar a cabo todas las funciones básicas de un SGBD: definición de datos, manipulación de datos, seguridad, integridad, etc... En particular, permite establecer todos los aspectos de los niveles externo, conceptual e interno (arquitectura ANSI/SPARC)

3.2.1 Funciones del servidor

Aceptar las solicitudes de la base de datos de los clientes.

Procesar dichas solicitudes.
Dar formato a los resultados y transmitirlos al cliente.
Llevar a cabo la verificación de integridad.
Mantener los datos generales de la base de datos.
Proporcionar control de acceso concurrente.
Llevar a cabo la recuperación.
Optimizar el procesamiento de consultas y actualización.

3.2.2 Tipos de servidores

Podemos dividir los servidores en dos clases: **iterativos y concurrentes.**

Un servidor iterativo realiza los siguientes pasos:
1.- Espera que llegue una consulta de un cliente.
2.- Procesa la consulta.
3.- Envía la respuesta al cliente que envió la consulta.
4.- Vuelve al estado inicial.

El problema del servidor iterativo es el paso 2. Durante el tiempo en el que el servidor está procesando la consulta, ningún otro cliente es servido.

Un servidor concurrente realiza los siguientes pasos:
1.- Espera que llegue la consulta de un cliente.
2.- Cuando le llega una nueva consulta, comienza un nuevo proceso para manejar esta consulta (cómo se realiza este paso depende del sistema operativo). El nuevo servidor maneja la totalidad de la consulta. Cuando se ha procesado completamente, este nuevo proceso termina.
3.- Se vuelve al primer paso.

La ventaja del servidor concurrente es que el servidor ejecuta un nuevo proceso para manejar cada consulta. Cada cliente tiene su "propio" servidor. Asumiendo que el sistema operativo permite la multiprogramación, clientes múltiples y servicio concurrente.

También podemos dividir los servidores en **servidores de transacciones** y **servidores de datos**.
Los sistemas servidores de transacciones, también llamados sistemas servidores de consultas, proporcionan una interfaz a través de la cual los clientes pueden enviar peticiones para realizar una acción que el servidor ejecutará y cuyos resultados se devolverán al cliente. Los usuarios pueden especificar sus peticiones con SQL o mediante la interfaz de una aplicación utilizando un mecanismo de llamadas a procedimientos remotos (RPC: 'Remote Procedure Call').

Los sistemas servidores de datos permiten que los clientes puedan interaccionar con los servidores realizando peticiones de lectura o modificación de datos en unidades tales como archivos o páginas. Por ejemplo, los servidores de archivos proporcionan una interfaz de sistema de archivos a través de la cual los clientes pueden crear, modificar, leer y borrar archivos. Los servidores de datos de los sistemas de bases de datos ofrecen muchas más funcionalidades; soportan unidades de datos de menor tamaño que los archivos, como páginas, tuplas u objetos. Proporcionan facilidades de indexación de los datos, así como facilidades de transacción, de modo que los datos nunca se quedan en un estado inconsistente si falla una máquina cliente o un proceso.

3.2.2.1 Servidores de transacciones

En los sistemas centralizados, un solo sistema ejecuta tanto la parte visible al usuario como el sistema subyacente. Sin embargo, la arquitectura del servidor de transacciones responde a la división funcional entre la parte visible al usuario y el sistema subyacente. Las computadoras personales gestionan la funcionalidad de la parte visible al usuario debido a las grandes necesidades de procesamiento que requiere el código de la interfaz gráfica, ya que la creciente potencia de las computadoras personales permite responder a esas necesidades. Los sistemas servidores tienen almacenado un gran volumen de datos y soportan la funcionalidad del sistema subyacente, mientras que las computadoras personales actúan como clientes suyos. Los clientes envían transacciones a los sistemas servidores, que se encargan de ejecutar aquellas transacciones y enviar los resultados de vuelta a los clientes, que se encargan a su vez de mostrar los datos en pantalla.

Se han desarrollado distintas normas como ODBC ('Open Database Connectivity', Conectividad abierta de bases de datos), para la interacción entre clientes y servidores. ODBC es una interfaz de aplicación que permite que los clientes generen instrucciones SQL para enviarlas al servidor en donde se ejecutan. Cualquier cliente que utilice la interfaz ODBC puede conectarse a cualquier servidor que proporcione dicha interfaz. Los sistemas de bases de datos de generaciones anteriores, al no haber tales normas, necesitaban que la parte visible al usuario y el sistema subyacente fueran proporcionados por el mismo fabricante.

Con el crecimiento de las normas de interfaz, las herramientas de la parte visible al usuario y los servidores del sistema subyacente son administrados cada vez por más marcas. Gupta SQL y Power-Builder son ejemplos de sistemas visibles al usuario independientes de los servidores del sistema subyacente. Es más, ciertas aplicaciones, como algunas hojas de cálculo o algunos paquetes de análisis estadístico, utilizan directamente la interfaz cliente-servidor para acceder a los datos desde un servidor del sistema subyacente. Como resultado, proporcionan un sistema visible al usuario especializado en tareas concretas.

Algunos sistemas de procesamiento de transacciones utilizan, además de ODBC, otras interfaces cliente-servidor. Éstas se definen por medio de una interfaz de programación de aplicaciones y son utilizadas por los clientes para realizar llamadas a procedimientos remotos de transacciones sobre el servidor. Para el programador, estas llamadas son como las llamadas normales a procedimientos, pero en realidad todas las llamadas a procedimientos remotos desde un cliente se engloban en una única transacción en el servidor. De esta forma, si la transacción aborta, el servidor puede deshacer los efectos de las llamadas a procedimientos remotos individuales.

El hecho de que adquirir y mantener una máquina pequeña sea ahora más barato, ha desembocado en una tendencia hacia el fraccionamiento de costes cada vez más extendida en la industria. Muchas compañías están reemplazando sus grandes sistemas por redes de estaciones de trabajo o computadoras personales conectadas a máquinas servidoras del sistema subyacente. Entre las ventajas, destacan una mejor funcionalidad respecto del coste, una mayor flexibilidad para localizar recursos y facilidades de expansión, mejores interfaces de usuario y un mantenimiento más sencillo.

3.2.2.2 Servidores de datos

Los sistemas servidores de datos se utilizan en redes de área local en las que se alcanza una alta velocidad de conexión entre los clientes y el servidor, las máquinas clientes son comparables al servidor en cuanto a poder de procesamiento y se ejecutan tareas de cómputo intensivo. En este entorno, tiene sentido enviar los datos a las máquinas clientes, realizar allí todo el procesamiento (que puede durar un tiempo) y después enviar los datos de vuelta al servidor. Nótese que esta arquitectura necesita que los clientes posean todas las funcionalidades del sistema subyacente. Las arquitecturas de los servidores de datos se han hecho particularmente populares en los sistemas de bases de datos orientadas a objetos.

En esta arquitectura surgen algunos aspectos interesantes, ya que el coste en tiempo de comunicación entre el cliente y el servidor es alto comparado al de acceso a una memoria local (milisegundos frente a menos de 100 nanosegundos)

Envío de páginas o envío de elementos

La unidad de comunicación de datos puede ser de grano grueso (como una página), o de grano fino (como una tupla). Si la unidad de comunicación de datos es una única tupla, la sobrecarga por la transferencia de mensajes es alta comparada con el número de datos transmitidos. En vez de hacer esto, cuando se necesita una tupla, cobra sentido la idea de enviar junto a aquella otras tuplas que probablemente vayan a ser utilizadas en un futuro próximo. Se denomina preextracción a la acción de buscar y enviar tuplas antes de que sea estrictamente necesario. Si varias tuplas residen en una página, el envío de páginas puede considerarse como una forma de preextracción, ya que cuando un procese desee acceder a una única tupla de la página, se enviarán todos las tuplas de esa página.

Bloqueo

La concesión del bloqueo de las tuplas de datos que el servidor envía a los clientes la realiza habitualmente el propio servidor. Un inconveniente del envío de páginas es que los clientes pueden recibir bloqueos de grano grueso (el bloqueo de una página bloquea implícitamente a todas las tuplas que residan en ella). El cliente adquiere implícitamente bloqueos sobre todas las tuplas preextraídas, incluso aunque no esté accediendo a alguno de ellos. De esta forma, puede detenerse innecesariamente el procesamiento de otros clientes que necesitan bloquear esos elementos. Se han propuesto algunas técnicas para la liberación de bloqueos, en las que el servidor puede pedir a los clientes que le devuelvan el control sobre los bloqueos de las tuplas preextraídas. Si el cliente no necesita la tupla preextraída, puede devolver los bloqueos sobre esa tupla al servidor para que éstos puedan ser asignados a otros clientes.

Caché de datos

Los datos que se envían al cliente a favor de una transacción se pueden alojar en una caché del cliente, incluso una vez completada la transacción, si dispone de suficiente espacio de almacenamiento libre. Las transacciones sucesivas en el mismo cliente pueden hacer uso de los datos en caché. Sin embargo, se presenta el problema de la coherencia de la caché: si una transacción encuentra los datos en la caché, debe asegurarse de que esos datos están al día, ya que después de haber sido almacenados en la caché pueden haber sido modificados por otro cliente. Así, debe establecerse una comunicación con el servidor para comprobar la validez de los datos y poder adquirir un bloqueo sobre ellos.

Caché de bloqueos

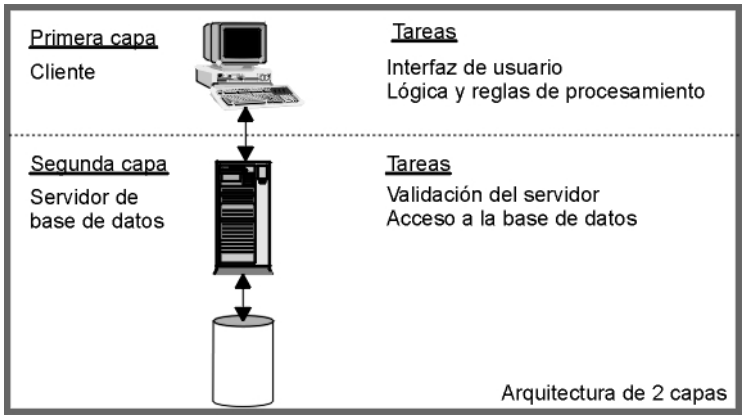
Los bloqueos también pueden ser almacenados en la memoria caché del cliente si la utilización de los datos está prácticamente dividida entre los clientes, de manera que un cliente rara vez necesita datos que están siendo utilizados por otros clientes. Supóngase que se encuentran en la memoria caché tanto el elemento de datos que se busca como el bloqueo requerido para acceder al mismo. Entonces, el cliente puede acceder al elemento de datos sin necesidad de comunicar nada al servidor. No obstante, el servidor debe seguir el rastro de los bloqueos en caché; si un cliente solicita un bloqueo al servidor, éste debe comunicar a todos los bloqueos sobre el elemento de datos que se encuentran en las memorias caché de otros clientes. La tarea se vuelve más complicada cuando se tienen en cuenta los posibles fallos de la máquina. Esta técnica se diferencia de la liberación de bloqueos en que la caché de bloqueo se realiza a través de transacciones; de otra forma, las dos técnicas serían similares.

4. TIPOS DE ARQUITECTURAS CLIENTE/SERVIDOR.

4.1. Arquitectura de 2 capas

La arquitectura cliente/servidor tradicional es una solución de 2 capas. La arquitectura de 2 capas consta de tres componentes distribuidos en dos capas: cliente (solicitante de servicios) y servidor (proveedor de servicios). Los tres componentes son:

- Interfaz de usuario.
- Gestión del procesamiento.
- Gestión de la base de datos.



Hay 2 tipos de arquitecturas cliente servidor de dos capas:

- Clientes obesos (thick clients): La mayor parte de la lógica de la aplicación (gestión del procesamiento) reside junto a la lógica de la presentación (interfaz de usuario) en el cliente, con la porción de acceso a datos en el servidor.

- Clientes delgados (thin clients): solo la lógica de la presentación reside en el cliente, con el acceso a datos y la mayoría de la lógica de la aplicación en el servidor.

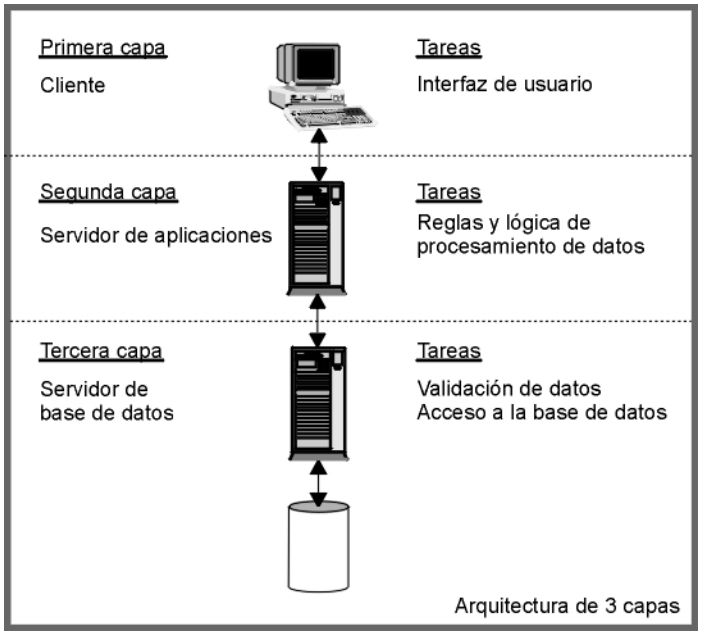
Es posible que un servidor funcione como cliente de otro servidor. Esto es conocido como diseño de dos capas encadenado.

Limitaciones

- El número usuarios máximo es de 100. Más allá de este número de usuarios se excede la capacidad de procesamiento.
- No hay independencia entre la interfaz de usuario y los tratamientos, lo que hace delicada la evolución de las aplicaciones.
- Dificultad de relocalizar las capas de tratamiento consumidoras de cálculo.
- Reutilización delicada del programa desarrollado bajo esta arquitectura.

4.2. Arquitectura de 3 capas

La arquitectura de 3 capas surgió para superar las limitaciones de la arquitectura de 2 capas. La tercera capa (servidor intermedio) está entre el interfaz de usuario (cliente) y el gestor de datos (servidor). La capa intermedia proporciona gestión del procesamiento y en ella se ejecutan las reglas y lógica de procesamiento. Permite cientos de usuarios (en comparación con sólo 100 usuarios de la arquitectura de 2 capas). La arquitectura de 3 capas es usada cuando se necesita un diseño cliente/servidor que proporcione, en comparación con la arquitectura de 2 capas, incrementar el rendimiento, flexibilidad, mantenibilidad, reusabilidad y escalabilidad mientras se esconde la complejidad del procesamiento distribuido al usuario.



Limitaciones

Construir una arquitectura de 3 capas es una tarea complicada. Las herramientas de programación que soportan el diseño de arquitecturas de 3 capas no proporcionan todos los servicios deseados que se necesitan para soportar un ambiente de computación distribuida.

Un problema potencial en el diseño de arquitecturas de 3 capas es que la separación de la interfaz gráfica de usuario, la lógica de gestión de procesamiento y la lógica de datos no es siempre obvia. Algunas lógicas de procesamiento de transacciones pueden aparecer en las 3 capas. La ubicación de una función particular en una capa u otra debería basarse en criterios como los siguientes:

- Facilidad de desarrollo y comprobación
- Facilidad de administración.
- Escalabilidad de los servidores.
- Funcionamiento (incluyendo procesamiento y carga de la red)

El middleware

Como hemos visto, las capas están localizadas en máquinas diferentes que están conectadas a través de la red. El middleware es el software que proporciona un conjunto de servicios que permite el acceso transparente a los recursos en una red. El middleware es un módulo intermedio que actúa como conductor

entre dos módulos de software. Para compartir datos, los dos módulos de software no necesitan saber cómo comunicarse entre ellos, sino cómo comunicarse con el módulo de middleware.

Es el encargado del acceso a los datos: acepta las consultas y datos recuperados directamente de la aplicación y los transmite por la red. También es responsable de enviar de vuelta a la aplicación, los datos de interés y de la generación de códigos de error.

5.- VENTAJAS E INCONVENIENTES.

5.1 Ventajas

Las principales ventajas del modelo Cliente/Servidor son las siguientes:

- Interoperabilidad: los componentes clave (cliente, servidor y red) trabajan juntos.

- Flexibilidad: la nueva tecnología puede incorporarse al sistema.

- Escalabilidad: cualquiera de los elementos del sistema puede reemplazarse cuando es necesario, sin impactar sobre otros elementos. Si la base de datos crece, las computadoras cliente no tienen que equiparse con memoria o discos adicionales. Esos cambios afectan solo a la computadora en la que se ejecuta la base de datos.

- Usabilidad: mayor facilidad de uso para el usuario.

- Integridad de los datos: entidades, dominios, e integridad referencial son mantenidas en el servidor de la base de datos.

- Accesibilidad: los datos pueden ser accedidos desde múltiples clientes.

- Rendimiento: se puede optimizar el rendimiento por hardware y procesos.

- Seguridad: la seguridad de los datos está centralizada en el servidor.

5.2 Inconvenientes

- Hay una alta complejidad tecnológica al tener que integrar una gran variedad de productos. El mantenimiento de los sistemas es más difícil pues implica la interacción de diferentes partes de hardware y de software, distribuidas por distintos proveedores, lo cual dificulta el diagnóstico de fallos.

- Requiere un fuerte rediseño de todos los elementos involucrados en los sistemas de información (modelos de datos, procesos, interfaces, comunicaciones, almacenamiento de datos, etc.). Además, en la actualidad existen pocas herramientas que ayuden a determinar la mejor forma de dividir las aplicaciones entre la parte cliente y la parte servidor.

- Es más difícil asegurar un elevado grado de seguridad en una red de clientes y servidores que en un sistema con un único ordenador centralizado (cuanto más distribuida es la red, mayor es su vulnerabilidad).

- A veces, los problemas de congestión de la red pueden reducir el rendimiento del sistema por debajo de lo que se obtendría con una única máquina (arquitectura centralizada). También la interfaz gráfica de usuario puede a veces ralentizar el funcionamiento de la aplicación.

- Existen multitud de costes ocultos (formación en nuevas tecnologías, cambios organizativos, etc.) que encarecen su implantación.

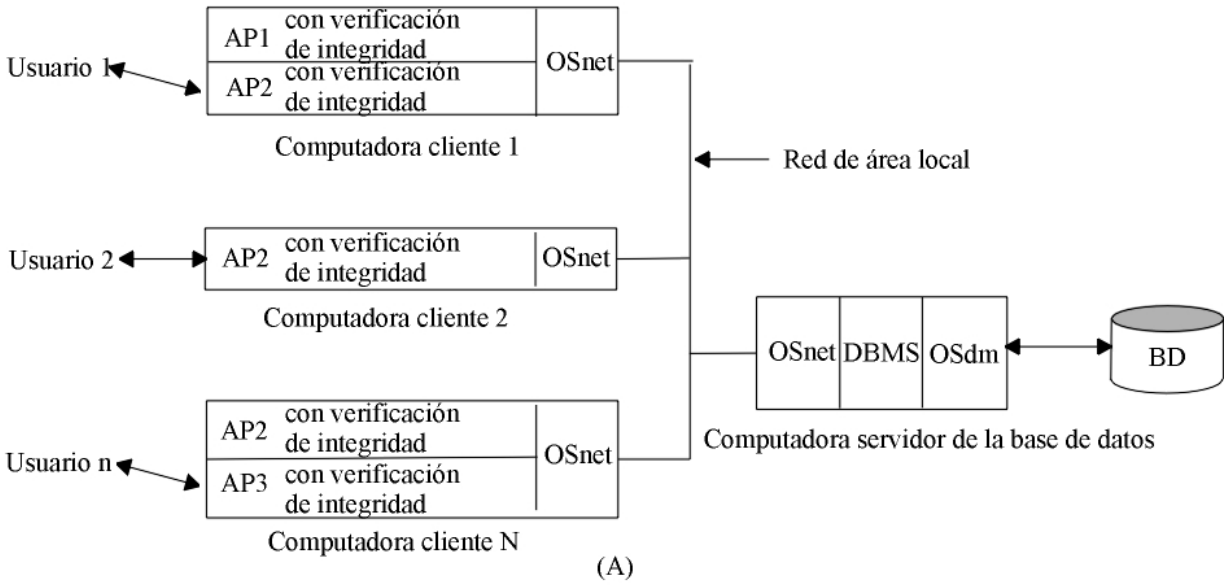
- La arquitectura cliente/servidor es una arquitectura que está en evolución, y como tal no existe estandarización.

6. INTEGRIDAD DE LA BASE DE DATOS.

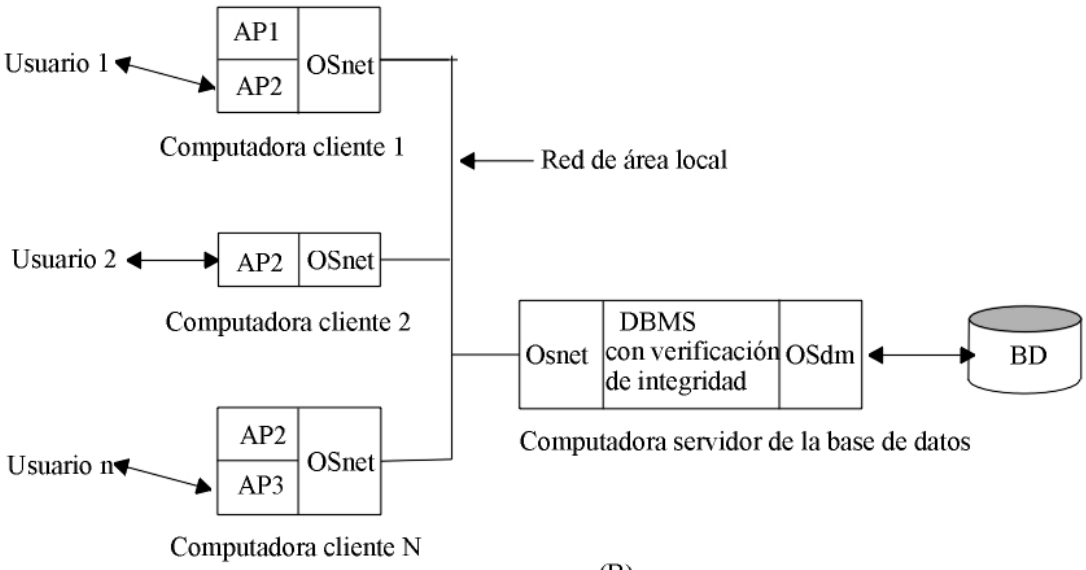
En la arquitectura cliente/servidor, todo el procesamiento de la base de datos queda consolidado en una sola computadora, y esta consolidación permite un alto grado de integridad de los datos. Al procesar las solicitudes a la base de datos en el servidor, si las restricciones han quedado definidas en el servidor, se pueden aplicar consistentemente.

Para comprender esto, veamos la siguiente figura:

Verificación de integridad cliente servidor: (a) Verificación de la integridad de la aplicación y (b) Verificación de integridad centralizada



(A)



(B)

En la parte (a), la verificación de integridad no es llevada a cabo por el servidor. En vez de ello, los programas de aplicación de las computadoras cliente requieren que durante su procesamiento se verifiquen la integridad. En la parte (b), el SGBD verifica la integridad en el servidor.

Si se comparan ambas figuras, se verá por qué es preferible la verificación en el servidor. Si los que verifican la integridad son los clientes, la lógica de revisión deberá estar incluida en cada uno de los programas de aplicación, lo que no sólo es un desperdicio y resulta ineficiente, sino que también puede ser susceptible de errores. Los programadores de la aplicación pueden entender las restricciones de forma distinta y al programarlas pueden cometer equivocaciones. Siempre que se desarrolla una aplicación, todas las verificaciones de las restricciones deberán duplicarse.

No todas las modificaciones de datos se efectúan a través de los programas de aplicación. Los usuarios pueden efectuar modificaciones a través de un lenguaje de consulta/actualización, y los datos pueden importarse de forma masiva. En demasiadas aplicaciones la integridad no es verificada al importar datos de estas fuentes.

Si el servidor se ocupa de la verificación de la integridad, las restricciones sólo necesitan definirse, verificarse y validarse una sola vez. Las modificaciones en los datos de todas las fuentes serán verificadas para asegurar la integridad. No tendrá ninguna importancia si una modificación ha sido sometida por un programa de aplicación, o por un proceso de consulta/actualización, o ha sido importada. Independientemente de la fuente, el SGBD se asegurará de que la integridad no ha sido violada.

7. GATILLOS

Un gatillo es un procedimiento de aplicación invocado de forma automática por el SGBD cuando ocurra algún evento. Por ejemplo, en una aplicación de inventario se puede escribir un gatillo para generar un pedido siempre que la cantidad de algún elemento se reduzca por debajo de algún valor de umbral.

Para poner en práctica un gatillo, el desarrollador escribe un código de gatillo e informa al SGBD de su existencia y de las condiciones bajo las cuales se deberá invocar dicho gatillo. Cuando se cumplen tales condiciones, el SGBD llamará al gatillo. En un sistema cliente/servidor, los gatillos residen en y son invocados por el servidor.

Son útiles, pero los gatillos pueden resultar un problema. En presencia de ellos, un usuario puede provocar una actividad en la base de datos que no espera o que ni siquiera conoce.

En un entorno multiusuario, los procedimientos de tipo gatillo necesitan bloqueos. La transacción que genera el evento que dispara el gatillo pudiera ya haber obtenido bloqueos que entran en conflicto con dicho gatillo. En tal situación, una transacción puede crear un interbloqueo consigo misma.

Los gatillos se pueden disparar en cascada e incluso formar un ciclo cerrado. Se ponen en cascada cuando un gatillo genera una condición que hace que se invoque otro gatillo que a su vez crea una condición que genera la invocación a un tercer gatillo y así sucesivamente. Los gatillos forman ciclos cuando en cascada regresan a ellos mismos. En caso de que ocurra lo anterior, el SGBD deberá tener algún medio de evitar un ciclo infinito.

8.- CONTROL DE PROCESAMIENTO CONCURRENTE.

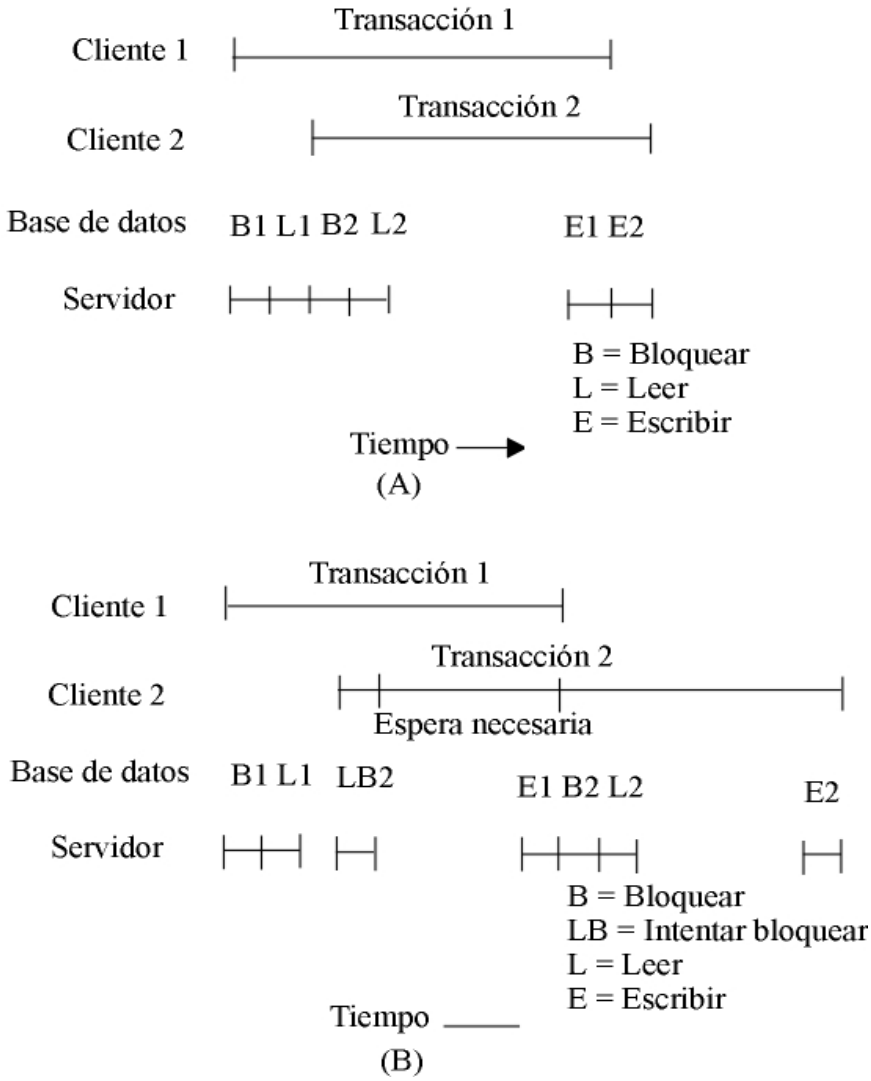
Uno de los retos del desarrollo de las aplicaciones cliente/servidor es obtener el mayor paralelismo posible de las computadoras cliente, mientras se protege contra problemas como actualizaciones perdidas y lecturas inconsistentes.

Ya que varios usuarios podrían procesar los mismos datos de la base de datos, es necesario proteger contra problemas de actualización perdida y lecturas inconsistentes. Existen dos formas de hacer esto:

Control mediante el bloqueo pesimista

La primera estrategia, a veces conocida como bloqueo pesimista (los bloqueos se colocan en anticipación de algún conflicto). Para poner en práctica esta estrategia, el SGBD coloca bloqueos implícitos en cada comando SGBD ejecutado después de START TRANSACTION. Tales bloqueos se mantienen entonces hasta que se emita un comando COMMIT o ROLLBACK.

Bloqueos en un sistema cliente servidor



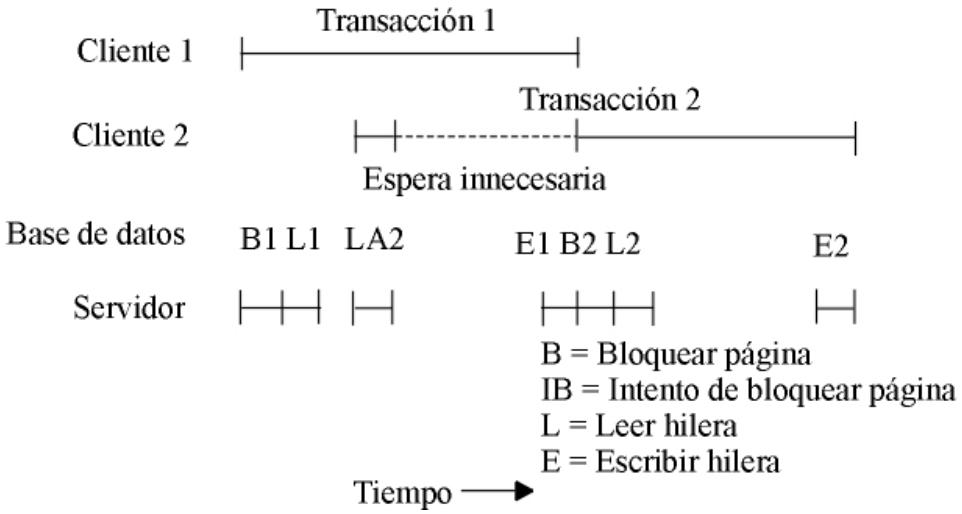
En la figura se muestran los tiempos requeridos para procesar una transacción. En la parte (a), las transacciones procesan diferentes datos, por lo que no existe conflicto de datos. El Cliente 2 espera mientras el servidor procesa la solicitud de datos del Cliente 1, pero una vez hecho esto, ambos clientes se procesan en paralelo hasta el final de la transacción.

En la parte (b) se muestran dos transacciones intentando procesar los mismos datos. En este caso, Cliente 2 espera datos hasta que Cliente 1 haya terminado de procesar su transacción. Esta espera puede resultar un problema. El Cliente 1 puede tener bloqueados los datos durante mucho tiempo (por ejemplo si tiene que introducir muchos datos en esa acción). Estos problemas empeoran si el SGBD no soporta

bloqueos a nivel de tupla. Algunos SGBD bloquean una página (una sección de tuplas), en vez de una sola tupla, y algunos incluso bloquean toda la tabla. Si el bloqueo se lleva a cabo con estos grandes niveles de granularidad, los bloqueos implícitos pueden dar como resultado retrasos muy notables e innecesarios.

La situación mostrada en la siguiente figura supone un bloqueo a nivel de página. Dos transacciones están procesando dos tuplas distintas que por casualidad están en la misma página. En este caso, el Cliente 2 debe de esperar a que termine Cliente 1, aun cuando las dos transacciones están procesándose en diferentes tuplas. El retraso puede ser sustancial y, en este caso, también es innecesario.

Procesamiento a nivel de página de solicitudes de datos no conflictivas sobre la misma página



Control mediante bloqueos optimistas

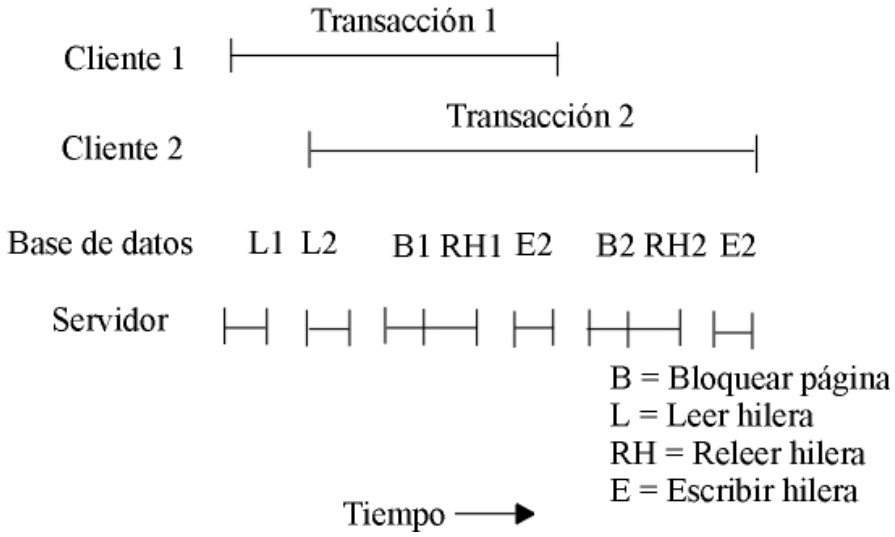
Una segunda estrategia, a veces conocida como bloqueo optimista, ya que no se prevé ningún conflicto, pero en caso de que hubiera alguno uno de los usuarios deberá volver a hacer su trabajo. Consiste en retrasar el bloqueo hasta el último momento posible, esperando que no exista conflicto. En este estilo, se procesa tanto de la transacción como sea posible antes de colocar cualquier bloqueo. A continuación se obtienen los bloqueos, conservándose de modo muy breve. Para utilizar esta estrategia, el SGBD no debe aplicar ningún bloqueo implícito, el programa de aplicación deberá colocar dichos bloqueos.

Con bloqueos retardados, los bloqueos se conservan durante periodos muy cortos, por lo cual la espera requerida se reduce. Si dos transacciones están procesando los mismos datos en paralelo, aquel que termina primera será el que realizará las modificaciones a la base de datos. La transacción más rápida no tendrá que esperar a que termine la más lenta.

La estrategia de bloqueos retardados es muy útil si el SGBD coloca los bloqueos a un nivel más alto que el de tupla. Dos usuarios procesando hileras distintas en la misma página o en la misma tabla, por ejemplo, pueden procesar en paralelo. En vista de que los bloqueos se conservan durante un periodo corto de tiempo, los usuarios minimizan sus interferencias. En la siguiente figura se muestra la misma situación que en la figura anterior. A diferencia del bloqueo implícito, prácticamente no existe ningún retardo.

El problema con los bloqueos retardados es que las transacciones pudieran necesitar procesarse dos y en ocasiones varias veces. Además la lógica es más complicada, y por lo tanto este método coloca una carga más pesada sobre el programador de la aplicación o la porción del SGBD, o sobre ambos.

Procesamiento de bloqueo retrasado de solicitudes de datos que no tienen conflictos en una página única



9.- RECUPERACIÓN.

La recuperación en sistemas cliente/servidor se puede llevar a cabo en la misma forma que en un sistema centralizado. La computadora servidor conserva un registro de las modificaciones en la base de datos, y la base de datos se almacena en algún dispositivo de almacenamiento. Cuando ocurre algún fallo, el SGBD del servidor puede deshacer la operación.

ORACLE8 Y LA ARQUITECTURA CLIENTE/SERVIDOR

1. INTRODUCCIÓN

Oracle Corporation ha sido líder en la introducción de tecnologías avanzadas de bases de datos cliente/servidor; ha dirigido el desarrollo de sus productos específicamente a soportar el diseño, la implementación y la gestión de los sistemas de bases de datos cliente/servidor.

Oracle8 es la primera versión de la base de datos de Oracle que incorpora la tecnología orientada a objetos. Se trata de una base de datos objeto-relacional, dado que esta implementación no es una base de datos orientada a objetos pura, ni tampoco es una base de datos relacional, representa un híbrido de ambas.

Oracle8 soporta la arquitectura cliente/servidor. Como ya hemos visto anteriormente, la premisa de la informática cliente/servidor es distribuir la ejecución de una tarea entre varios procesadores de una red. Cada procesador está dedicado a un conjunto de tareas secundarias enfocadas y específicas que realiza mejor. El resultado es una eficiencia incrementada y una efectividad del sistema como conjunto. La división de la ejecución de las tareas entre los procesadores es realizada a través de un protocolo de solicitudes de servicios. Un procesador, el cliente, solicita un servicio de otro procesador, el servidor. La implementación más extendida del procesamiento cliente/servidor implica la separación de la parte de la interfaz del usuario de una aplicación de la parte del acceso a los datos.

En el cliente, o sección de entrada (front-end), de la configuración típica cliente/servidor se encuentra una estación de trabajo operando con una plataforma GUI (Graphical User Interface, Interfaz gráfica de usuario), usualmente Windows, Macintosh o Motif. En el back-end (lado del servidor) de la configuración se encuentra un servidor de bases de datos, que a menudo está gestionado por un sistema operativo UNIX, NetWare, Windows NT o VMS.

La arquitectura cliente/servidor toma también la forma de una configuración servidor-a-servidor. En esta disposición, un servidor juega el papel de un cliente y solicita servicios de base de datos desde el otro servidor. Varios servidores de bases de datos pueden parecer como sólo una base de datos lógica y pueden proporcionar acceso transparente a los datos distribuidos por la red.

Diseñar una aplicación cliente/servidor eficiente es algo como un acto equilibrado; la meta es distribuir uniformemente la ejecución de tareas entre los procesadores mientras que se hace un uso óptimo de los recursos disponibles. Dada la complejidad creciente y la potencia de procesamiento requerida para gestionar una GUI y la demanda creciente para la capacidad de procesamiento en los servidores y redes de bases de datos, conseguir la distribución adecuada de las tareas es todo un reto. Los sistemas cliente/servidor son inherentemente más difíciles de desarrollar y de gestionar que los sistemas de aplicaciones tradicionales basados en ordenadores anfitriones debido a los siguientes desafíos que hay que superar:

- Los componentes de un sistema cliente/servidor están distribuidos a través de tipos de procesadores muy variados. Muchos más componentes de software gestionan funciones de clientes, de redes y de servidor, así como una serie de capas de infraestructura, todas las cuales deben estar en su lugar y configuradas para ser compatibles entre sí.
- La complejidad de las aplicaciones GUI es mayor que la de sus predecesores basados en caracteres. Las GUI son capaces de presentar mucha más información al usuario y proporcionan muchas rutas adicionales de navegación para los elementos de la interfaz.
- Es más difícil la localización de problemas de rendimiento y de errores al mayor número de componentes y de capas en el sistema.

2. FUNCIONES DE ORACLE8

Oracle8 posee una amplia gama de funcionalidades; las más importantes se comentan a continuación:

Mecanismos de seguridad

Los sofisticados mecanismos de seguridad de Oracle controlan el acceso a los datos sensibles utilizando un conjunto de privilegios. En función del nombre con el que se conectan a la base de datos, a los usuarios se les conceden derechos para consultar, modificar y crear datos. Los clientes usan estos mecanismos para asegurarse de que ciertos usuarios pueden consultar los datos de carácter sensible, mientras que a otros se les niega dicha posibilidad.

Realización de copias de seguridad y recuperación

Oracle proporciona sofisticados procedimientos de realización de copias de seguridad y recuperación de los datos. Las copias de seguridad permiten crear una copia secundaria de los datos de Oracle; los procedimientos de recuperación restauran los datos a partir de una copia de seguridad. La estrategia de copias de seguridad y recuperación de Oracle permite minimizar la pérdida de datos y el tiempo de para cuando se produce un problema. Oracle Server también proporciona esquemas de copia de seguridad y recuperación que permiten un acceso ininterrumpido a los datos 7 días a la semana, 24 horas al día y 365 días al año.

Control del espacio

Oracle ofrece una gestión flexible del espacio. Se puede asignar un cierto espacio de disco para el almacenamiento de los datos, y controlar las subsiguientes asignaciones instruyendo a Oracle sobre cuánto espacio reservar para los requerimientos futuros. También tiene una serie de características que fueron diseñadas teniendo en cuenta las necesidades de las bases de datos de muy gran tamaño.

Conectividad de carácter abierto

Oracle proporciona conectividad hacia y desde paquetes software de otros fabricantes. Utilizando las extensiones a la base de datos Oracle, se puede trabajar con información almacenada con otros sistemas de bases de datos, como Sybase o Acces. También se pueden almacenar los datos en la base de datos de Oracle y acceder a ellos desde otros paquetes software, como Visual Basic, Powerbuilder, o SQL *Windows.

Herramientas de desarrollo

El servidor Oracle, al que normalmente se denomina motor de la base de datos, funciona con un amplio conjunto de herramientas de desarrollo, herramientas de consulta para usuario final, aplicaciones comerciales y herramientas de gestión de la información de ámbito corporativo.

Mecanismos de integridad

El servidor Oracle también se encarga de la integridad de los datos. Si se produce cualquier tipo de fallo mientras un usuario está cambiando los datos en una base de datos, ésta tiene la capacidad de deshacer o cancelar cualquier transacción sospechosa. Con Oracle Server, nunca albergamos dudas en lo referente al estado de una determinada transacción. El servidor incluye también un bloqueo completo por filas de todos los datos almacenados.

Por ejemplo, si estuviéramos trabajando en una aplicación de compra de acciones de bolsa, construida sobre Oracle, y dos usuarios desearan comprar el lote número 5, formado por 100 acciones de la empresa Database Technologies, la base de datos impediría que dicha acción tuviera lugar. Puesto que sólo hay un único lote, la base de datos sólo permitiría a uno de los usuarios acceder al procedimiento de compra, y el otro usuario se vería obligado a esperar. Cuando el segundo usuario recibiera finalmente la autorización para continuar, se encontraría con que el lote ha pasado al estado de vendido. Oracle Server gestiona de modo transparente estas situaciones, manteniendo la integridad de los datos.

Componente procedimental

A partir de Oracle7, esta opción pasó a formar parte del núcleo del servidor. El fundamento de esta opción es el lenguaje de programación de Oracle, PL/SQL. Con esta opción se pueden implementar las siguientes funcionalidades:

- Procedimientos almacenados. Se pueden almacenar programas (o segmentos de código) en la base de datos Oracle, para realizar funciones de importancia para nuestro sistemas. Por ejemplo, en una aplicación de facturación para televisión por cable, se puede utilizar un procedimiento almacenado para generar cartas de aviso dirigidas a clientes morosos. La ejecución del procedimiento se desencadena al generar el resumen mensual de movimientos del cliente, cada vez que se detectan cargos no pagados durante un plazo superior a sesenta días.

- Disparadores de base de datos (triggers). Son segmentos de código almacenados en la base de datos, y que se disparan como respuesta a sucesos que tienen lugar en las aplicaciones. En una aplicación de gestión de recursos humanos, por ejemplo, cuando se contrata a un nuevo empleado, la adición de la nueva información a la base de datos de personal podría utilizar un disparador para generar mensajes que fueran enviados a otras partes de la empresa. Estos mensajes, desencadenados por la adición del nuevo empleado, podrían por ejemplo alertar a los encargados del sistema de comunicaciones de que se ha incorporado nuevo personal.

- Paquetes (packages). Los procedimientos se suelen agrupar, almacenándose el código como una única unidad de programación en la base de datos. Por ejemplo, un almacén central de una cadena de librerías podría definir un paquete que se encargue de dirigir ciertos pedidos especiales al distribuidor adecuado. Dentro de ese paquete, habría procedimientos para iniciar la transferencia de mercancías, procesar las notificaciones de órdenes pendientes, procesar los pedidos repetidos, etc.

Componente de procesamiento distribuido

En muchos sistemas, hay partes de los datos corporativas que residen en diferentes computadoras, situadas en ciudades distintas. Por ejemplo, las cuentas por cobrar pueden estar almacenadas en Madrid, los datos de compras en Barcelona, investigación y desarrollo en Lisboa y la oficina principal en Ciudad Real. Cada localización guarda una parte de los datos corporativos y, sin embargo, los usuarios necesitan acceder a dicha información como si todos los datos se encontraran almacenados en una computadora central. La capacidad distribuida del servidor Oracle permite que este escenario se convierta en realidad. Existe lo que se denomina transparencia de ubicación, de forma que un usuario de Barcelona trabajando con información de Lisboa no es consciente de la localización física de los datos. Por ejemplo, la ubicación física de los datos de compras es desconocida para todos los usuarios.

Componente de consulta en paralelo

La opción de consulta en paralelo permite sacar partido del procesamiento de consultas en computadoras dotadas de más de una unidad central de proceso (UCP). En las máquinas con una sola UCP, así como en las máquinas con múltiples UCP sin la opción de consulta en paralelo, sólo un proceso puede acceder a la base de datos y mostrar los datos que responden al criterio de selección definido.

Cuando se utiliza la opción de procesamiento paralelo en una máquina múltiples UCP, Oracle ejecuta varios procesos de consulta, los cuales se reparten el procesamiento de la consulta y trabajan de manera simultánea. Una vez que los resultados están preparados, se integran y son presentados al usuario.

Con esta capacidad de procesar consultas en paralelo, una consulta que podía tardar antes una hora puede ser procesada en cuestión de minutos, sacando partido de toda la potencia de UCP disponible.

3. ESTRUCTURA FÍSICA DE UNA BASE DE DATOS

Una base de datos es una colección de datos relacionados que son utilizados y recuperados conjuntamente para uno o más sistemas de aplicaciones. La situación física y la implementación de la base de datos es transparente para los programas de aplicaciones y, en realidad, se puede mover y reestructurar la base de datos física sin afectar a los programas.

Físicamente, en su forma más simple, una base de datos Oracle no es más que un conjunto de archivos que se encuentra en alguna parte del disco. La situación física de estos archivos es irrelevante para la función de la base de datos (aunque es importante para su rendimiento). Los archivos son binarios y sólo se puede acceder a ellos utilizando el software del núcleo de Oracle. La consulta de los datos de los archivos de la base de datos se realiza generalmente con una de las herramientas de Oracle (tales como SQL *Plus) usando el SQL (Structured Query Language, Lenguaje estructurado de consultas).

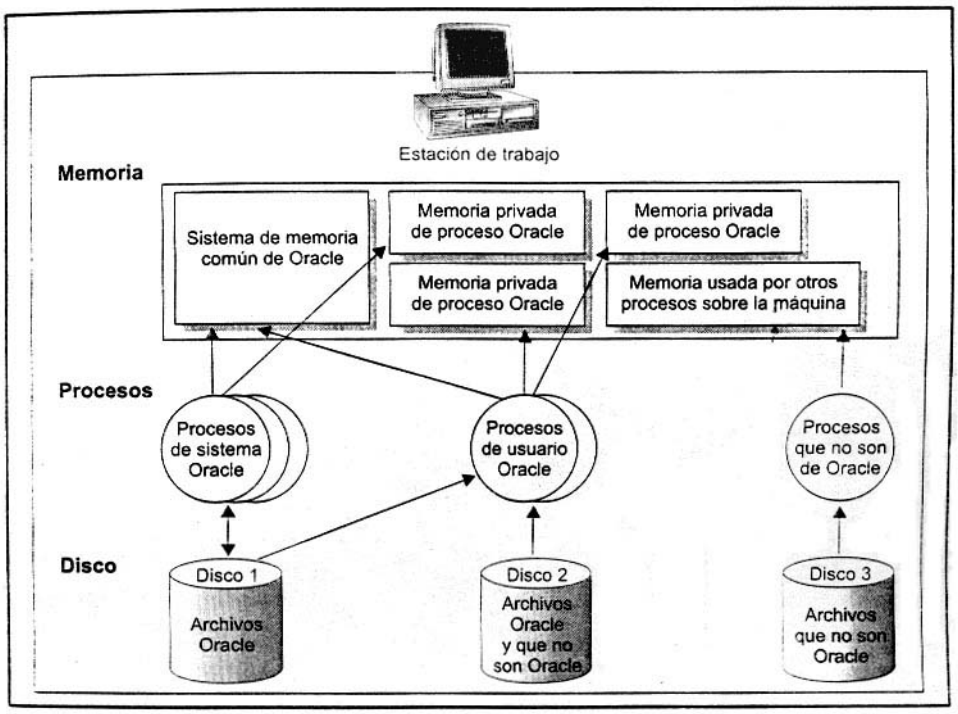
Lógicamente, la base de datos está dividida en un conjunto de cuentas de usuario de Oracle (esquemas), cada una de las cuales está identificada por un nombre de usuario (username) y una contraseña (password) única, exclusiva para esa base de datos. Las tablas y otros objetos pertenecen a uno de esos usuarios de Oracle y el acceso a los datos disponible solamente abriendo una sesión en la base de datos usando un nombre de usuario de Oracle y una contraseña. Sin un nombre de usuario y una contraseña válidos para la base de datos, el acceso a la base de datos es denegado a cualquiera. El nombre de usuario y la contraseña de Oracle es diferente del nombre del usuario y la contraseña del sistema operativo. Por ejemplo, una base de datos que reside en una máquina UNIX requiere que iniciemos la sesión en la máquina UNIX usando nuestro nombre de usuario y la contraseña del sistema operativo y que después iniciemos la sesión en Oracle antes de que podamos usar los objetos de bases de datos. La iniciación de sesión UNIX no se requerirá para una configuración cliente/servidor. Se requiere este proceso de inicio de sesión (login), o de conectarse, con una base de datos tanto si se usa una herramienta Oracle como si no es Oracle.

El mismo nombre de tabla puede coexistir en dos cuentas de usuario Oracle independientes. Aunque las tablas pueden tener el mismo nombre, son tablas diferentes. Algunas veces la misma base de datos (el mismo conjunto de archivos de base de datos) es utilizado para contener versiones diferentes de tablas (en cuentas Oracle independientes) para los desarrolladores, para la comprobación del sistema, o para comprobación de los usuarios, o el mismo nombre de tabla es utilizado en diferentes sistemas de aplicaciones.

Con frecuencia se confunde una cuenta de usuario Oracle como una base de datos, pero esto no es estrictamente correcto. Se pueden usar dos cuentas de usuario Oracle para contener datos para dos sistemas de aplicaciones completamente diferentes; se tendrían dos bases de datos lógicas implementadas en la misma base de datos física utilizando dos cuentas de usuario Oracle.

Además de los archivos físicos, los procesos de Oracle y las estructuras de la memoria deben estar también presentes antes de que se pueda usar la base de datos.

La siguiente figura muestra la arquitectura básica Oracle:



La arquitectura básica de Oracle

3.1. Archivos Oracle

Hay tres conjuntos de archivos en el disco que componen una base de datos, que son los siguientes:
- Archivos de base de datos.
- Archivos de control.
- Registros de rehacer.

Los más importantes son los archivos de base de datos, en los cuales reside la base de datos. Los archivos de control y los registros de rehacer soportan la funcionalidad de la arquitectura en sí.

Los tres conjuntos de archivos deben estar presentes, abiertos y disponibles para que Oracle pueda utilizar cualquier dato de la base de datos. Sin esos archivos no es posible acceder a la base de datos y el administrador de la base de datos es posible que tuviera que recuperar algo o toda la base de datos usando una copia de seguridad (si la hay). Todos estos archivos son binarios.

Archivos de bases de datos

Los archivos de bases de datos contienen los datos reales y son generalmente los de mayor tamaño (desde unos pocos Megabytes a muchos Gigabytes). Los demás archivos (los archivos de control y los registros de rehacer) soportan el resto de la arquitectura. Dependiendo de sus tamaños, las tablas (y otros objetos) de todas las cuentas de los usuarios pueden obviamente encontrarse en un archivo de base de datos; pero esta no es una situación ideal debido a que no hace muy flexible la estructura de la base de datos para controlar el acceso para el almacenamiento por diferentes usuarios de Oracle, poniendo la base de datos en diferentes unidades de disco o haciendo copias de seguridad y restableciendo solamente parte de la base de datos.

Debe disponerse por lo menos de un archivo de base de datos (adecuado para una base de datos pequeña o de comprobación), pero usualmente hay más de uno. En términos de acceso y de utilización de los datos de las tablas y de otros objetos, el número (o situación) de los archivos es irrelevante.

Por defecto, los archivos de bases de datos se mantendrán en el tamaño especificado cuando fueron creados. Sin embargo, utilizando las opciones AUTOEXTEND y RESIZE del comando ALTER DATABASE SQL, pueden ser aumentados o reducidos.

Archivos de control

Cualquier base de datos debe tener por lo menos un archivo de control, aunque generalmente se tiene más de uno para protegerse de las pérdidas de información. El archivo de control lleva un registro del nombre de la base de datos, de la fecha y de la hora en que ha sido creada, de la situación de la base de datos y de los registros de rehacer, y de la información de sincronización para asegurar que los tres conjuntos de archivos se encuentran siempre de acuerdo. Cada vez que se agrega una nueva base de datos o un archivo de registros de rehacer a la base de datos, se registra la información en los archivos de control.

Registros de rehacer (registros de transacción)

Cualquier base de datos debe tener por lo menos dos archivos de rehacer. Estos son los diarios de la base de datos; los registros de rehacer (redo logs) registran todos los cambios de los objetos de usuario o de los objetos de sistema. Oracle mantiene estos cambios en memoria por razones de rendimiento. Una entrada/salida a disco es 1000 veces más lenta que una acción en memoria. Si se produce cualquier tipo de fallo, como una pérdida de uno o más archivos de base de datos, se pueden usar los cambios registrados en los registros de rehacer para devolver la base de datos a un estado consistente sin perder ninguna transacción entregada. En caso de fallo sin pérdida de datos, como cuando falla la máquina Oracle puede aplicar la información de los registros de rehacer automáticamente sin intervención del administrador de la base de datos.

- Registros de rehacer en línea. Los registros de rehacer en línea son dos o más archivos de registros de rehacer que están siempre en uso mientras la instancia Oracle está activada y ejecutándose. Los cambios que se realicen son registrados para cada registro de rehacer por turnos. Cuando uno está lleno, se escribe en el otro; cuándo éste se ha llenado, se sobreescribe el primero y el ciclo continúa.

- Registros de rehacer fuera de línea. Los registros de rehacer fuera de línea son copias exactas de los registros de rehacer en línea que han sido llenados; es opcional si se pide que Oracle los cree.

4. PROCESOS DE SERVIDOR Y PROCESOS DE USUARIO

Cuando un programa se inicializa en la base de datos, se comunica con Oracle a través de un proceso.

4.1. Procesos de usuario (cliente)

Los procesos de usuario trabajan por el propio usuario, solicitando información a los procesos del servidor. Ejemplos de procesos de usuario son: Oracle Forms, Oracle Reports y SQL *Plus. Estas son herramientas comunes que cualquier usuario de la base de datos utiliza para comunicarse con ella.

4.2. Procesos de servidor

Los procesos de servidor reciben las solicitudes de los procesos de usuario y se comunican con la base de datos. A través de esta comunicación, los procesos de usuario trabajan con los datos de la base de datos.

Una buena forma de pensar en el proceso cliente/servidor es imaginarse un restaurante. El cliente (usuario), comunica al camarero lo que desea tomar. Éste comunica a la cocina la solicitud. El trabajo del personal de cocina es preparar la comida, decir al camarero cuándo está lista y hacer inventario. El camarero

entonces le lleva la comida. En este símil, el camarero representa el proceso de cliente y el personal de cocina representa el proceso de servidor.

4.3. Procesos auxiliares de la base de datos

Como acabamos de decir, los procesos de servidor toman las solicitudes de los procesos de usuario (cliente), y se comunican con la base de datos en nombre de os procesos de usuario. Vamos a ver un conjunto especial de procesos de servidor que ayudan a operar a la base de datos:

- Escritor de base de datos (DBWR). El escritor de base de datos es un proceso obligatorio, que escribe los bloques de datos modificados en los archivos de la base de datos. Es uno de los dos únicos procesos que permiten escribir en los archivos de datos que constituyen la base de datos Oracle. Con ciertos sistemas operativos, pro razones de rendimiento, Oracle permite tener múltiples escritores de base de datos.

- Checkpoint (punto de inspección, CKPT). El proceso checkpoint (punto de inspección) es opcional. Cuando los usuarios están trabajando con una base de datos Oracle, realizan solicitudes para acceder a los datos. Dichos datos se leen de los archivos de la base de datos y se colocan en un área de memoria, donde los usuarios pueden consultarlos. Eventualmente, algún usuario realiza un cambio en los datos, que debe registrarse de nuevo en los archivos de datos originales. Anteriormente hemos hablado de los registros de rehacer y de cómo registrar todas las transacciones. Cuando se produce la conmutación de los registros de rehacer se produce un checkpoint o punto de inspección. En ese momento, Oracle consulta la memoria y escribe la información de los bloques de datos modificados en el disco. También notifica al archivo de control que se ha producido una conmutación de los registros de rehacer.

  Normalmente, estas tareas las realiza el escritor de registros (LGWR) que veremos a continuación. Por razones de rendimiento, el DBA puede efectuar cambios en la base de datos para activar los procesos checkpoint. La única tarea de este proceso es realizar el checkpoint en lugar del escritor de registro.

- Escritor de registro (LGWR). El escritor de registro es un proceso obligatorio que escribe los datos de rehacer en los registros de rehacer. Recuerde que los registros de rehacer son una copia de cada transacción que se produce en la base de datos. Gracias a ello, Oracle puede recuperarse de distintos tipos de fallo. Además, puesto que una copia de cada transacción se escribe en el registro de rehacer, Oracle no gasta constantemente sus recursos escribiendo las modificaciones de los datos en los archivos de datos de forma inmediata. Esto mejora el rendimiento. El escritor de registro es el único proceso que escribe en los registros de rehacer, y también es el único proceso en una base de datos Oracle que lee los registros de rehacer.

- Monitor de sistema (SMON). El monitor de sistema es un proceso obligatorio que lleva a cabo cualquier recuperación necesaria durante la inicialización.

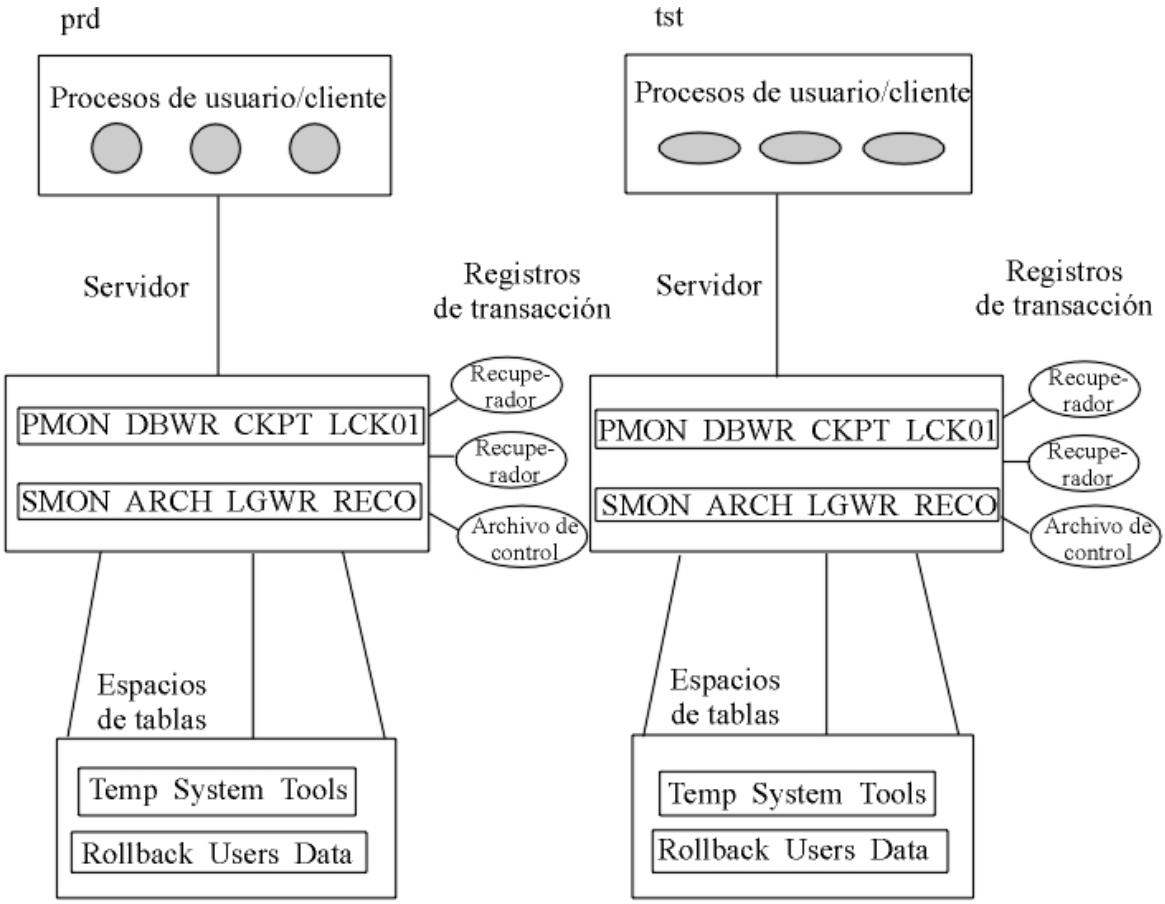
- Monitor de proceso (PMON). El monitor de proceso es un proceso obligatorio que lleva a cabo la tarea de recuperación cuando un usuario de la base de datos falla. Asume la identidad del usuario que falla, libera todos los recursos de la base de datos que el usuario tenía y deshace la transacción abortada.

- Archivador (ARCH). El archivador es un proceso opcional. Los registros de rehacer (transacción) se escriben en forma secuencial. Cuando un registro se llena, se conmuta al siguiente registro de rehacer disponible. Cuando una base de datos está funcionando en modo ArchiveLog, la base de datos hace una copia del archivo de rehacer para que, cuando la base de datos conmute de nuevo a ese registro de rehacer, se disponga de una copia del contenido de este archivo para propósitos de recuperación. Este es el trabajo del proceso archivador. Hace una copia del archivo de forma similar a una fotocopiadora.

- Bloqueo (LCKn). El bloqueo es un proceso opcional. Cuando la base de datos Oracle está funcionando en modo servidor paralelo, se crean múltiples procesos lck. En este modo, los bloqueos ayudan a que las bases de datos se comuniquen.

- Recuperador (RECO). Este proceso opcional sólo se emplea cuando la base de datos se ejecuta con la opción distribuida de Oracle. Una transacción distribuida es aquella en la que dos o más localizaciones de los datos deben mantenerse en sincronismo. Por ejemplo, tiene que tener una copia de los datos en Madrid y otra en Barcelona. Supongamos que, mientras se actualizan los datos, la comunicación telefónica con Barcelona se interrumpe debido a una tormenta y una avalancha de barro arrasa la línea telefónica. El trabajo del proceso recuperador es solucionar las transacciones que pueden haberse completado en Madrid pero no en Barcelona. Estas transacciones se denominan transacciones dudosas hasta que el proceso recuperador las resuelve.

- Expedidor (Dnnn). Los expedidores son procesos en segundo plano opcionales, presentes sólo cuando se usa una configuración de servidor multihebra. Para cada protocolo de comunicación en uso (por ejemplo TCP/IP, SNA) se crea al menos un proceso expedidor (D000,..., Dnnn). Cada proceso expedidor es responsable del encaminamiento de las solicitudes desde los procesos de usuario conectados hasta los procesos de servidor compartidos disponibles, y del retorno de las respuestas al proceso de usuario apropiado.



Dos instancias de Oracle

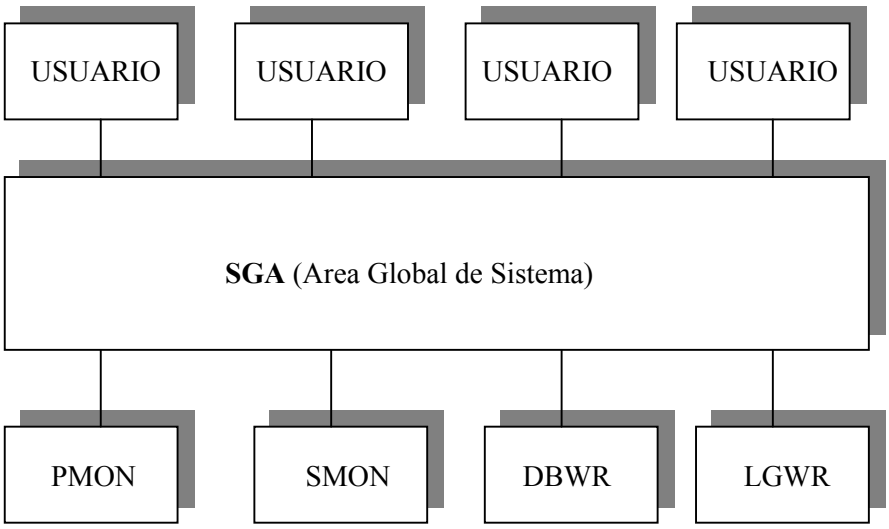
5. LA MEMORIA

Hasta aquí hemos hablado de archivos de datos y programas. También hemos hablado de los de servidor y los procesos de cliente. Ahora vamos ver cómo se comunican los procesos de servidor y de cliente entre sí y consigo mismos, mediante las estructuras de memoria. Como su nombre indica, se trata de un área de memoria especial donde los procesos pueden comunicarse entre sí o consigo mismos.

Oracle utiliza dos tipos de estructura de memoria; el área global de sistema o SGA, y el área global de programa o PGA.

5.1. Área global de sistema (SGA)

SGA es el lugar de la memoria donde la base de datos Oracle almacena la información sobre sí misma. Lo hace así dado que la memoria es la forma más rápida y más eficiente de permitir que los procesos se comuniquen. Esta estructura de memoria es accesible a todos los procesos de usuario y de servidor. La siguiente figura muestra como el SGA se encuentra en el centro de todas las comunicaciones.



Dado que el SGA es el mecanismo por el que distintos procesos de cliente y de servidor se comunican, es importan que se entiendan sus diferentes componentes. El SGA del servidor Oracle se divide en los siguientes componentes fundamentales:

Memoria caché del búfer de datos

La memoria caché del búfer de datos es donde Oracle almacena los bloques de datos más recientemente utilizados. En otras palabras, es la memoria caché de datos. Cuando se introduce información en la base de datos, se almacena en bloques de datos. La memoria caché del búfer de datos es un área de memoria en la que Oracle coloca dichos bloques de datos, para que un proceso de usuario pueda acceder a ellos. Antes de que ningún proceso de usuario pueda acceder a parte de los datos, éstos deben residir en la memoria caché del búfer de datos. El tamaño de la memoria caché el búfer de datos tiene un límite físico, por lo que cuando Oracle la ha llenado, deja los bloques de mayor uso en la memoria caché y elimina los bloques de menor uso. Esto se hace mediante el algoritmo LRU (menos recientemente usado).

Un punto importante que debemos clarificar es que si un proceso de cliente necesita una información que no está e la memoria caché, la base de datos accede a la unidad de disco físico, lee los bloques de datos necesarios y los coloca en la memoria caché del búfer de datos. Esto hace que todos los procesos de servidor y de cliente puedan aprovecharse de la lectura de disco físico.

Memoria caché de diccionario (memoria caché de fila)

Una memoria caché de diccionario está formada por filas extraídas del diccionario de datos. El diccionario de datos contiene toda la información que Oracle necesita para gestionarse, tal como qué usuarios tienen acceso a la base de datos Oracle, qué objetos de la base de datos poseen y dónde se localizan dichos objetos.

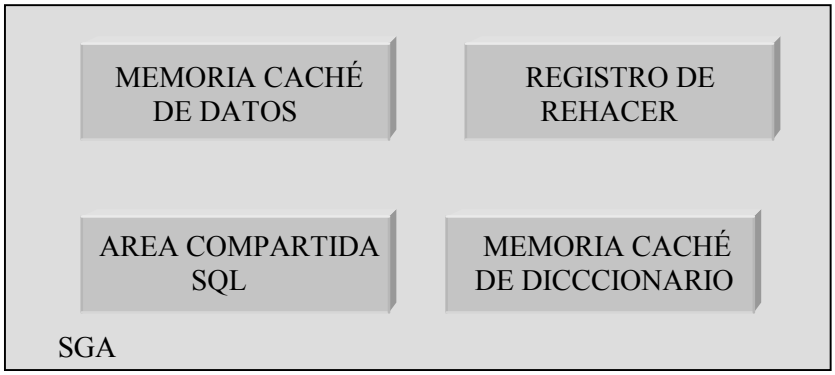
Búfer del registro de rehacer

Cualquier transacción que pueda registrarse en el registro de rehacer (los registros de rehacer en línea se necesitan para propósitos de recuperación) debe residir primero en el búfer del registro de rehacer. Se trata de un área de memoria reservada para este propósito. La base de datos, periódicamente, vacía este búfer en los registros de rehacer en línea.

Área compartida SQL

Hay que pensar en el área compartida SQL como en la memoria caché de programa. Es el área donde se almacenan todos los programas. Los programas de una base de datos Oracle se basan en el lenguaje estándar SQL. Esta memoria caché contiene todas las órdenes SQL analizadas que están listas para ejecutarse.

Resumiendo, el SGA es el gran comunicador. Es el lugar de memoria donde se coloca la información a la que pueden acceder los procesos de servidor y de cliente.



5.2. Área global de programa (PGA)

El PGA es un área de memoria utilizada por un único proceso Oracle. El área global de programa no se comparte; contiene datos e información de control para un único proceso. Contiene información tal como matrices internas y variables de sesión del proceso. Las distintas partes del proceso pueden comunicarse entre sí, pero no con el mundo exterior.

6. PROTECCIÓN DE LOS DATOS

6.1. Transacciones, confirmación y anulación

Los cambios realizados en la base de datos no son guardados hasta que el usuario decide explícitamente que las instrucciones de insertar, actualizar y eliminar deben hacerse permanentes. Hasta ese momento, los cambios se encuentran en un estado pendiente y cualquier fallo, como un fallo de la máquina, invertirá los cambios.

Una transacción es una unidad indivisible de trabajo que comprende una o más instrucciones SQL; comienza cuando el usuario se conecta por primera vez con la base de datos y termina cuando se emita una instrucción de confirmación, COMMIT, o de anulación, ROLLBACK. Sobre una instrucción COMMIT o ROLLBACK, comienza automáticamente la siguiente transacción. Todas las instrucciones dentro de una transacción son, o todas guardadas (confirmadas) o se da marcha atrás en todas (anulación).

Al confirmar una transacción se hacen permanentes los cambios de la transacción completa en la base de datos y, una vez confirmados, no pueden invertirse los cambios. La anulación invierte todas las inserciones, actualizaciones y eliminaciones de la transacción; una vez más, una vez realizada la anulación, esos cambios no pueden ser entregados después. Internamente, el proceso de la confirmación significa escribir los cambios registrados en la caché de búfer del registro de rehacer del SGA en los archivos del registro de rehacer en línea del disco. Si este disco de E/S tiene éxito, la aplicación recibe un mensaje indicando que se ha realizado con éxito la confirmación. El proceso DBWR en segundo plano puede escribir los bloques de datos Oracle reales en la caché de búfer de base de datos del SGA en un momento posterior. En el caso de que falle el sistema, Oracle puede volver a aplicar automáticamente los cambios desde los archivos de los registros de rehacer incluso si los bloques de datos Oracle no se han vuelto a escribir en los archivos de base de datos antes del fallo.

Oracle aplica también la idea de anulación a nivel de instrucción. Si una instrucción sencilla falla durante la transacción, fallará la transacción completa. En otras palabras, una instrucción INSERT para 1000 filas insertará o bien 1000 filas o ninguna en absoluto; funcionará la transacción completa o no sucede nada. Si una instrucción falla dentro de una transacción, el resto de las instrucciones de la transacción estará todavía en un estado pendiente y deberá ser confirmado o deberá producirse la anulación.

Si un proceso de usuario termina anormalmente (se cancela el proceso, por ejemplo), el proceso PMON en segundo plano produce una anulación de los cambios. Todos los bloqueos mantenidos por la transacción son liberados automáticamente cuando una transacción realiza una confirmación o anulación o cuando el proceso PMON en segundo plano hace anular la transacción. Además, otros recursos del sistema (como los segmentos de anulación) son liberados para ser utilizados por otras transacciones.

Los puntos de guardar permiten al usuario establecer marcadores dentro de una transacción de forma que tenga la opción de anular sólo parte del trabajo dentro de la transacción. Se pueden usar puntos de guardar en las transacciones largas y complejas para proporcionar la opción de dar marcha atrás para determinadas instrucciones. Sin embargo, esto hace que haya una carga extraordinaria en el sistema para que realice el trabajo de una instrucción e invierta después los cambios; usualmente los cambios en la lógica pueden producir una mejor solución. Cuando Oracle realiza una anulación en un punto de guardar, el resto de las instrucciones de la transacción permanecen en un estado pendiente y deben ser confirmadas o se deben anular. Oracle libera los bloqueos realizados por las instrucciones en las que se ha realizado la anulación.

6.2. Integridad de los datos

La integridad de los datos hace cumplir las reglas de validación de los datos, tal como comprobar que un valor de porcentaje se encuentra entre 0 y 100, para garantizar que datos no válidos no acceden a las tablas. Históricamente, estas reglas se hacían cumplir por los mismos programas de la aplicación (y las mismas reglas se comprobaban repetidamente en programas diferentes). Sin embargo, Oracle permite que el

usuario defina y almacene esas reglas con relación a los objetos de la base de datos con los cuales se relacionan de forma que sólo es necesario codificarlas una vez para que se hagan cumplir siempre que se realice cualquier tipo de cambio en la tabla, con independencia de la herramienta que emita la instrucción de insertar, actualizar o eliminar. Esta comprobación toma la forma de limitaciones de la integridad y de los disparadores de la base de datos.

Integridad de los datos

Las restricciones de la integridad hacen cumplir las reglas de las actividades en el nivel de la base de datos definiendo un conjunto de controles para las tablas el sistema de usuario. Estos controles se hacen cumplir automáticamente siempre que se emite una instrucción de insertar, actualizar o eliminar sobre la tabla. Si se viola cualquiera de las restricciones, se produce la anulación de la instrucción. Las demás instrucciones dentro de la transacción permanecen en un estado pendiente y pueden ser confirmadas o se pueden anular de acuerdo con la lógica de la aplicación.

Debido a que las restricciones de la integridad son controladas en el nivel de la base de datos, son realizadas con independencia de donde se haya originado la instrucción de insertar, actualizar o eliminar, ya sea una herramienta Oracle o no. Definir controles que utilicen estas restricciones es también más rápido que realizar los mismos controles usando SQL. Además, la información proporcionada al declarar restricciones es utilizada por el optimizador de Oracle para tomar decisiones mejores sobre la forma de ejecutar una instrucción sobre la tabla. El producto Oracle Forms puede utilizar también restricciones para generar código automáticamente en los programas front-end para proporcionar al usuario un aviso temprano ante cualquier error.

Los tipos de restricciones de integridad que se pueden establecer en una tabla son NOT NULL (no permitir valores nulos), PRIMARY KEY (clave primaria), UNIQUE (valor único), FOREIGN KEY (clave externa), CHECK(comprobar que se cumplen una serie de condiciones) y los índices.

Disparadores de base de datos

Un disparador de base de datos es un bloque PL/SQL que se puede definir para que ejecute automáticamente instrucciones de insertar, actualizar y eliminar sobre una tabla. Se puede definir el disparador para que se ejecute una vez para la instrucción completa o una vez para cada fila que es insertada, actualizada o bien eliminada. Para cada tabla, hay doce eventos para los que se puede definir disparadores de base de datos. Para cada uno de los doce eventos, se pueden definir muchos disparadores de base de datos para el mismo evento.

Un disparador de base de datos puede llamar a procedimientos e base de datos que son también escritos en PL/SQL. A diferencia de los disparadores de base de datos, los procedimientos de la base de datos son almacenados en una forma compilada. Por esta razón, deben ponerse los segmentos de código más largos dentro de un procedimiento y, después, llamar al procedimiento desde el disparador de la base de datos.

Además de implementar reglas complejas de actividad, de comprobación y de opciones por defecto, se pueden usar los disparadores de base de datos para insertar, actualizar y eliminar otras tablas. Un ejemplo de su uso es proporcionar una facilidad de auditoría siempre que se cambia una fila de una tabla. Sin los disparadores de base de datos, esta función sería implementada en los programas de sección de entrada (front-end) que realizan el cambio en la base de datos; sin embargo, alguien que pase por alto el código de los programas de sección de entrada (utilizando SQL *Plus, por ejemplo) no pasaría por las comprobaciones y el procesamiento definidos.

Los disparadores de base de datos difieren de las restricciones en que permiten al usuario intercalar instrucciones SQL dentro de ellos, mientras que las restricciones no pueden.

7. HERRAMIENTAS DE DESARROLLO CLIENTE/SERVIDOR

Oracle incluye una variedad de GUI del lado del cliente que completan su arquitectura integrada cliente/servidor. Estas series de productos incluyen herramientas CASE (Computer assisted software engineering, Ingeniería de software asistida por computador), entornos de desarrollo orientados a objetos y componentes del tiempo de ejecución capaces de operar con el Oracle8 Server y otras bases de datos SQL. A continuación se relacionan algunos productos que sirven de ayuda en el desarrollo cliente/servidor:

- Designer/2000, que sirve para el desarrollo de aplicaciones cliente/servidor sofisticadas y cuyo entorno CASE proporciona un "repositorio" (repository) amplio y un conjunto de herramientas potentes que permite analizar sistemáticamente, diseñar modelos y generar componentes cliente y servidor de una aplicación.

- Developer/2000, que comprende Oracle Forms, Oracle Reports, Oracle Graphic y Oracle Book dentro de un entorno de desarrollo integrado único. Se puede construir una aplicación utilizando solamente estas herramientas o usándolas con Designer/2000 para producir impresos e informes generados.

- Power Objects. Además de las herramientas anteriores, la serie de productos GUI de Oracle incluye otra orientada a objetos, entorno de aplicación GUI dieñado para competir como los similares de PowerBuilder y Visual Basic. Power Objects proporciona un entorno de desarrollo de aplicaciones rápido con muchas características de arrastrar y soltar y gestión de transacciones de bases de datos automáticas.

ARQUITECTURA DE BASES DE DATOS DISTRIBUIDAS

1. INTRODUCCIÓN

Una de las principales tendencias de la informática a lo largo de los últimos años ha sido el pasar de grandes computadoras centralizadas (que daban como resultado gigantescas bases de datos monolíticas en los setenta y principios de los ochenta) a redes distribuidas de sistemas informáticos (con una mayor descentralización y autonomía del procesamiento). Con la llegada de las minicomputadoras, las tareas de procesamiento de datos, tales como el control de inventarios y el procesamiento de pedidos pasaron de mainframes corporativas a sistemas departamentales más pequeños. El explosivo aumento de popularidad de las computadoras personales en los años ochenta llevó la potencia de las computadoras directamente a las mesas de los despachos de millones de personas.

Conforme las computadoras y las redes de computadoras se extendían a través de las organizaciones, los datos ya no residían en un único sistema bajo el control de un único SGBD. En vez de ello, los datos se fueron extendiendo a través de muchos sistemas diferentes, cada uno con su propio gestor de base de datos. Con frecuencia los diferentes sistemas informáticos y los diferentes sistemas de gestión de base de datos procedían de diferentes fabricantes.

Estas tendencias han hecho que la industria informática y la comunidad de gestión de datos enfocara su atención en los problemas de gestión de bases de datos distribuidas.

Un sistema de computación distribuida consiste en un conjunto de computadores (que no necesariamente tienen que ser homogéneos), que están interconectados entre sí formando una red, y que cooperan para realizar una determinada tarea. Un sistema de computación distribuida parte un problema grande en pequeñas piezas, y soluciona cada una de ellas eficientemente de una manera coordinada.

Podemos definir una base de datos distribuida (BDD o DDB 'Distributed Database System) como aquella cuyos datos están repartidos entre más de una máquina, y un sistema de gestión de bases de datos distribuidas (SGBDD o DDBMS 'Distributed Database Management System') como el software que gestiona una base de datos distribuida haciendo que la distribución de los datos sea **transparente** al usuario, es decir, los usuarios actúan como si todo estuviese junto en una sola base de datos.

En la arquitectura distribuida el SGBD y la BD no están asociados a un determinado ordenador, sino a una red cuyos nodos se reparten las funciones. Una base de datos distribuida es vista por las aplicaciones igual que si fuera centralizada. Es el SGBDD el que se encarga de preservar la integridad y coherencia de la BD. Sin embargo existe otra definición mucho menos estricta de base de datos distribuida utilizada por muchos fabricantes de SGBD, según la cual una base de datos es distribuida sí permite lecturas y modificaciones remotas, independientemente de que éstas sean transparentes o no para las aplicaciones. Esta definición no es adecuada cuando se desea seleccionar una BD realmente distribuida.

Se suele distinguir entre sistemas homogéneos y heterogéneos. Un sistema es homogéneo si el SGBD usado en todas las máquinas es el mismo. Si existe más de un SGBD distinto el sistema se denomina heterogéneo.

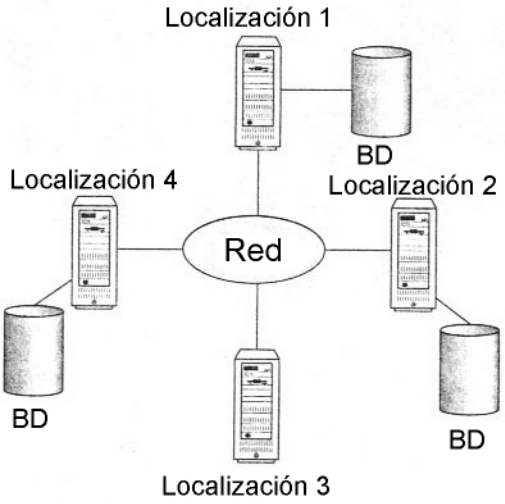
La distribución física, espacial o geográfica de la información puede aconsejar la utilización de esta arquitectura. Cada vez existen más productos disponibles en el mercado aunque no existen estándares.

Un SGBD que soporte una arquitectura de base de datos distribuida es mucho más complejo que uno para base de datos centralizada y el número de SGBDD disponibles en el mercado es mucho menor. Existen algunos SGBDs que ofrecen la posibilidad de implementar una BD distribuida sólo para sistemas homogéneos. Es una tecnología que no está tan probada como la centralizada.

Los usuarios acceden a la base de datos distribuida a través de aplicaciones. Estas aplicaciones se pueden clasificar en aquellas que no requieren datos de otros computadores (aplicaciones locales) y aquellos que requieren datos de otros computadores (aplicaciones globales). Un SGBDD tiene las siguientes características:

- Una colección de datos compartidos y relacionados lógicamente.
- Los datos están divididos en fragmentos.
- Los fragmentos se pueden duplicar.
- Los fragmentos se colocan en varios emplazamientos (computadores).
- Dichos emplazamientos están conectados por una red.
- Los datos de cada emplazamiento están bajo el control de un SGBD.
- El SGBD en cada emplazamiento puede manejar aplicaciones locales autónomamente.
- Cada SGBD participa en al menos una aplicación global.

No es necesario que todos los emplazamientos en el sistema tengan su propia base de datos local, como se muestra en la siguiente topología de un SGBDD:

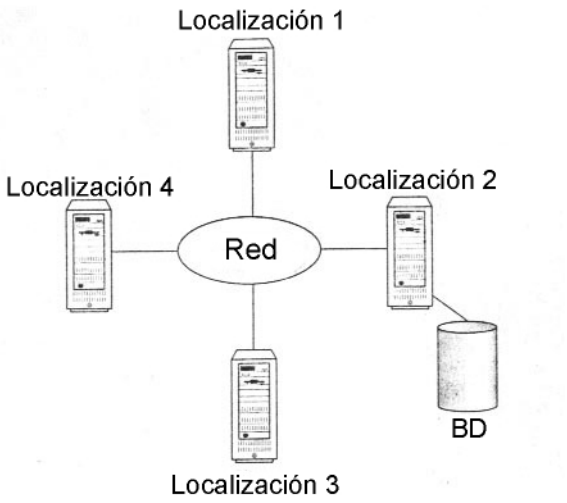


1.1. Procesamiento distribuido.

Es importante hacer una distinción entre SGBD distribuido y procesamiento distribuido.

Procesamiento distribuido: Una base de datos centralizada que puede ser accedida a través de una red.

El punto clave de la definición de una base de datos distribuida es que el sistema está formado por datos que están físicamente distribuidos a través de una red. Si los datos están centralizados, incluso aunque los usuarios puedan acceder a los datos a través de la red, no se puede considerar como un sistema distribuido. La topología de un sistema de procesamiento distribuido se muestra en la siguiente figura:



1.2. SGBD paralelo.

Debemos distinguir también entre SGBD distribuido y SGBD paralelo.

SGBD paralelo: Un SGBD que se ejecuta sobre múltiples procesadores y discos que han sido diseñados para ejecutar operaciones en paralelo, cuando sea posible, con el propósito de mejorar el rendimiento.

Los sistemas paralelos mejoran la velocidad de procesamiento y de E/S mediante la utilización de UCP y discos en paralelo. La fuerza que ha impulsado a los sistemas paralelos de bases de datos ha sido la demanda de aplicaciones que han de manejar bases de datos extremadamente grandes (del orden de terabytes, esto es, 10^{12} bytes) o que tienen que procesar un número enorme de transacciones por segundo (del orden de miles de transacciones por segundo).

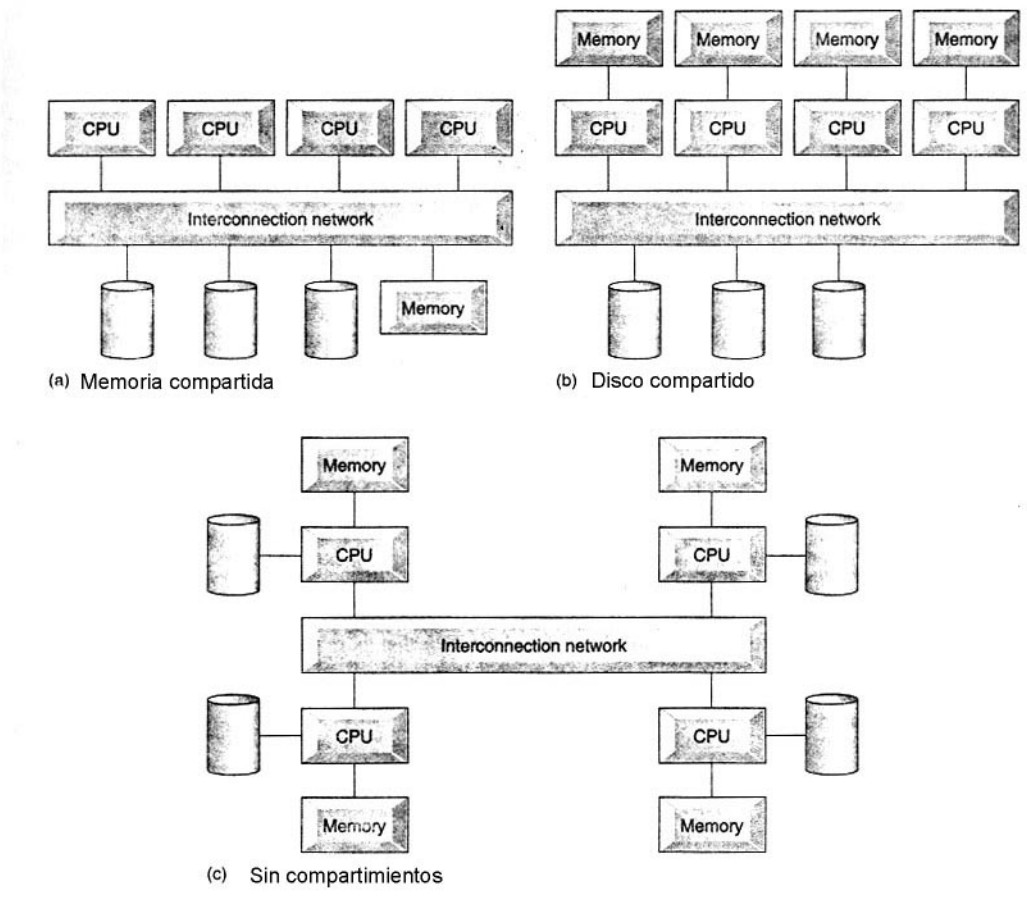
La ganancia de velocidad y la ampliabilidad son dos aspectos importantes en el estudio del paralelismo. La ganancia de velocidad se refiere a la ejecución en menos tiempo de una tarea dada mediante el incremento del grado del paralelismo. La ampliabilidad se refiere al manejo de transacciones más largas mediante el incremento del grado de paralelismo. La ampliabilidad es normalmente el factor más importante para medir la eficiencia de un sistema paralelo de bases de datos. El objetivo del paralelismo en los sistemas de bases de datos suele ser asegurar que la ejecución del sistema continuará realizándose a una velocidad aceptable, incluso en el caso de que aumente el tamaño de la base de datos o el número de transacciones. El incremento de la capacidad del sistema mediante el incremento del paralelismo proporciona a una empresa un modo de crecimiento más suave que el de reemplazar un sistema centralizado por una máquina más rápida.

Existen algunos factores que trabajan en contra de la eficiencia del paralelismo y pueden atenuar tanto la ganancia de velocidad como la ampliabilidad:

- Costes de inicio. El inicio de un único proceso lleva asociado un coste de inicio. En una operación paralela compuesta por miles de procesos, el tiempo de inicio puede llegar a ser mucho mayor que el tiempo real de procesamiento, lo que influye negativamente en la ganancia de velocidad.
- Interferencia. Como los procesos que se ejecutan en un sistema paralelo acceden con frecuencia a recursos compartidos, pueden sufrir un cierto retardo como consecuencia de la interferencia de cada nuevo proceso en la competencia con los procesos existentes por el acceso a los recursos más comunes, como el bus del sistema, los discos compartidos o incluso los bloqueos. Este fenómeno afecta tanto a la ganancia de velocidad como a la ampliabilidad.
- Sesgo. Al dividir cada tarea en un cierto número de pasos paralelos se reduce el tamaño del paso medio. Es más, el tiempo de servicio de la tarea completa vendrá determinado por el tiempo de servicio del paso más lento. Normalmente es difícil dividir una tarea en partes exactamente iguales, entonces se dice que la forma de distribución de los tamaños es sesgada. Por ejemplo, si se divide una tarea de tamaño 100 en 10 partes y la división está sesgada, puede haber algunas tareas de tamaño menor que 10 y otras de tamaño superior a 10; si el tamaño de una tarea fuera 20, la ganancia de velocidad que se obtendría al ejecutar las tareas en paralelo sólo valdría 5 en vez de lo que cabría esperarse, 10.

Existen varios modelos de arquitecturas para las máquinas paralelas:

Memoria compartida. Todos los procesadores comparten una memoria común.
Disco compartido. Todos los procesadores comparten un disco común.
Sin compartimiento. Los procesadores no comparten ni memoria ni disco.
Jerárquico. Es un híbrido de las anteriores.



Memoria compartida

En una arquitectura de memoria compartida, los procesadores y los discos tienen acceso a una memoria común, normalmente a través de un bus o de una red de interconexión. El beneficio de la memoria compartida es la extremada eficiencia en cuanto a la comunicación entre procesadores (cualquier procesador puede acceder a los datos de la memoria compartida sin necesidad de la intervención del software). Un procesador puede enviar mensajes a otros procesadores utilizando escrituras en la memoria, de modo que la velocidad de envío es mucho mayor (normalmente es inferior a un microsegundo) que la que se alcanza con un mecanismo de comunicación. El inconveniente de las máquinas con memoria compartida es que la arquitectura no puede ir más allá de 32 o 64 procesadores, porque el bus o la red de interconexión se convertiría en un cuello de botella (ya que está compartido por todos los procesadores). Llega un momento en el que no sirve de nada añadir más procesadores ya que éstos emplean la mayoría de su tiempo esperando su turno para utilizar el bus y así poder acceder a la memoria. Las arquitecturas de memoria compartida suelen dotar a cada procesador de una memoria caché muy grande para evitar las referencias a la memoria compartida siempre que sea posible. No obstante, en la caché no podrán estar todos los datos y no podrá evitarse el acceso a la memoria compartida. Por estas razones, las máquinas con memoria compartida no pueden extenderse llegado un punto; las máquinas actuales con memoria compartida no pueden soportar más de 64 procesadores.

Disco compartido

En el modelo de disco compartido, todos los procesadores pueden acceder directamente a todos los discos a través de una red de interconexión, pero los procesadores tienen memorias privadas. Las arquitecturas de disco compartido ofrecen dos ventajas respecto de las de memoria compartida. Primero, el bus de la memoria deja de ser un cuello de botella, ya que cada procesador dispone de memoria propia. Segundo, esta arquitectura ofrece una forma barata para proporcionar una cierta tolerancia ante fallos: si falla un procesador (o su memoria), los de más procesadores pueden hacerse cargo de sus tareas, ya que la base de datos reside en los discos, a los cuales tienen acceso todos los procesadores. La arquitectura de disco compartido tiene aceptación en bastantes aplicaciones; los sistemas construidos siguiendo esta arquitectura suelen denominarse agrupaciones (clusters).

El problema principal de los sistemas de discos compartidos es, de nuevo, la ampliabilidad. La interconexión con el subsistema de discos es ahora el nuevo cuello de botella. Esto es especialmente grave en situaciones en las que la base de datos realiza un gran número de acceso a los discos. Los sistemas de discos compartidos pueden soportar un mayor número de procesadores, en comparación con los sistemas de memoria compartida, pero la comunicación entre los procesadores es más lenta (hasta unos pocos milisegundos, si se carece de un hardware de propósito especial para comunicaciones), ya que se realiza a través de una red de interconexión.

Sin compartimiento

En un sistema sin compartimiento, cada nodo de la máquina consta de un procesador, memoria y uno o más discos. Los procesadores de un nodo pueden comunicarse con un procesador de otro nodo utilizando una red de interconexión de alta velocidad. Un nodo funciona como una red de interconexión de alta velocidad. Un nodo funciona como el servidor de los datos almacenados en los discos que posee. El modelo sin compartimiento salva el inconveniente de requerir que todas las operaciones de E/S vayan a través de una única red de interconexión, ya que las referencias a los discos locales son servidas por los discos locales de cada procesador; solamente van por la red las peticiones, los accesos a discos remotos y las relaciones de resultados. Es más, habitualmente, las redes de interconexión para los sistemas sin compartimiento se diseñan para ser ampliables, por lo que su capacidad de transmisión crece a medida que se añaden nuevos nodos. Como consecuencia, las arquitecturas sin compartimiento son más ampliables y

pueden soportar con facilidad un gran número de procesadores. El principal inconveniente de los sistemas sin compartimiento es el coste de comunicación y de acceso a discos remotos, coste que es mayor que el que se produce en las arquitecturas de memoria o disco compartido, ya que el envío de datos provoca la intervención del software en ambos extremos.

Jerárquica

La arquitectura jerárquica combina las características de las arquitecturas de memoria compartida, de disco compartido y sin compartimiento. A alto nivel, el sistema está formado por nodos que están conectados mediante una red de interconexión y que no comparten ni memoria ni discos. Así, el nivel más alto es una arquitectura sin compartimiento. Cada nodo del sistema podría ser en realidad un sistema de memoria compartida con algunos procesadores. Alternativamente, cada nodo podría ser un sistema de disco compartido y cada uno de estos sistemas de disco compartido podría ser a su vez un sistema de memoria compartida. De esta manera, un sistema podría construirse como una jerarquía con una arquitectura de memoria compartida con pocos procesadores en la base, en lo más alto una arquitectura sin compartimiento y quizá una arquitectura de disco compartido en el medio.

Las principales diferencias entre las bases de datos paralelas sin compartimientos y las bases de datos distribuidas son las siguientes: las bases de datos distribuidas normalmente se encuentran en varios lugares geográficos distintos, se administran de forma separada y poseen una interconexión más lenta. Otra gran diferencia es que en un sistema distribuido se dan dos tipos de transacciones, las locales y las globales. Una transacción local es aquella que accede a los datos del único emplazamiento en el cual se inició la transacción. Por otra parte, una transacción global es aquella que o bien accede a los datos situados en un emplazamiento diferente de aquel en el que se inició la transacción, o bien accede a datos de varios emplazamientos distintos.

2. FUNCIONES Y ARQUITECTURA DE UN SGBDD

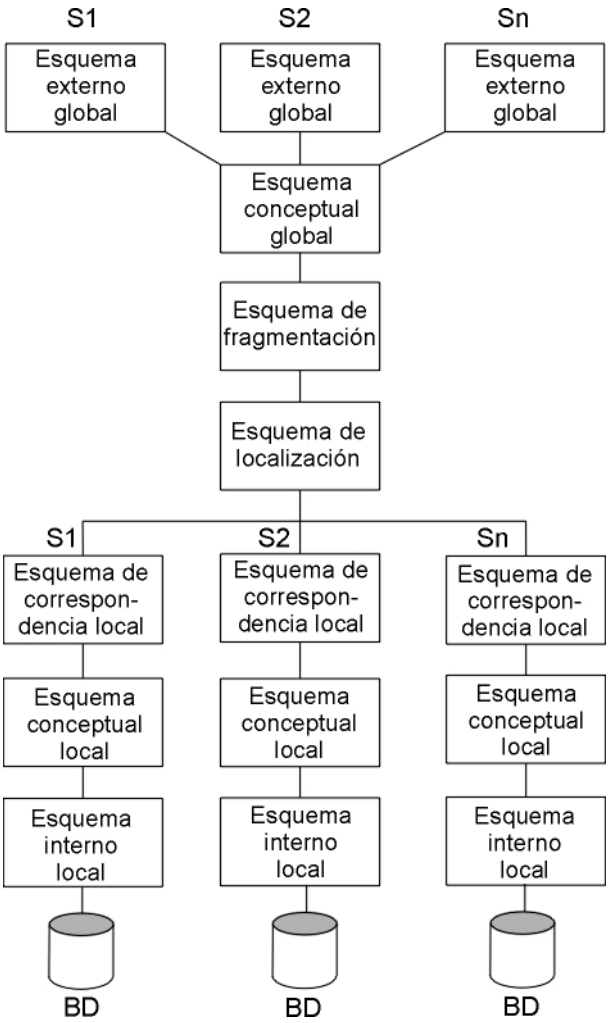
2.1. Funciones de un SGBDD.

Además de las funciones de un SGBD centralizado, un SGBDD debe tener las siguientes funciones:

- Servicios de comunicación extendidos para proporcionar acceso a localizaciones remotas y permitir la transferencia de consultar y datos entre las localizaciones usando una red.
- Catálogo del sistema extendido para almacenar detalles sobre la distribución de los datos.
- Procesamiento distribuido de consultas, incluyendo optimización de consultas y acceso a datos remotos.
- Control de concurrencia extendido para mantener la consistencia de los datos replicados.
- Servicios de recuperación extendidos que tengan en cuenta fallos en las localizaciones individuales y fallos de conexiones de comunicación.

2.2. Arquitectura de referencia para un SGBDD.

En la siguiente figura muestra una arquitectura de referencia para un SGBDD conforme a la arquitectura ANSI-SPARC:



Esquema conceptual global: El esquema conceptual global es la descripción lógica de la base de datos completa, como si no estuviera distribuida. Este nivel corresponde al nivel conceptual de la arquitectura ANSI-SPARC y contiene definiciones de entidades, relaciones, constantes e información sobre seguridad e integridad. Proporciona independencia de datos físicas desde el entorno distribuido. Los esquemas externos globales proporcionan independencia de datos lógica.

Esquemas de fragmentación y localización: El esquema de fragmentación es una descripción de cómo los datos están particionados lógicamente. El esquema de localización es una descripción de dónde están localizados los datos. El esquema de localización tiene en cuenta cualquier replicación.

Esquemas locales: Cada SGBD local tiene su propio conjunto de esquemas. Los esquemas conceptual e interno locales corresponden a los equivalentes de la arquitectura ANSI-SPARC.

3. LAS DOCE REGLAS

El principio fundamental de las bases de datos distribuidas es el siguiente:

Desde el punto de vista del usuario, un sistema distribuido deberá ser idéntico a un sistema no distribuido.

En otras palabras, los usuarios de un sistema distribuido deberán comportarse exactamente como si el sistema no estuviera distribuido. Todos los problemas de los sistemas distribuidos deberían ser internos (a nivel de realización) y no externos (a nivel de usuario). El término "usuario" significa un usuario (usuario final o programador de aplicaciones) que lleva a cabo operaciones de manipulación de datos. Dicho de otro

modo, en términos de SQL, la lógica de las operaciones SELECT (seleccionar), INSERT (insertar), UPDATE (actualizar) y DELETE (eliminar) no deberá sufrir cambios. Las operaciones de definición de datos, en cambio, requerirán cierta ampliación en un sistema distribuido, por ejemplo para que cuando se cree una relación en un sitio X, el creador pueda especificar que debe almacenarse en el sitio Y.

Esta regla fundamental da lugar a las siguientes doce reglas:

1.- Autonomía local

Los sitios de un sistema distribuido deben ser autónomos. La autonomía local significa que todas las operaciones en un sitio dado se controlan en ese sitio; ningún sitio X deberá depender de algún otro sitio Y para su buen funcionamiento (pues de otra manera el sitio X podría ser incapaz de trabajar, aunque no tenga en sí problema alguno, si se cae el sitio Y). La autonomía local implica también un propietario y una administración locales de los datos, con responsabilidad local: todos los datos pertenecen en realidad a una base de datos local, aunque sean accesibles desde algún sitio remoto. Por tanto, las cuestiones de seguridad, integridad y representación en almacenamiento de los datos locales permanecen bajo el control de la instalación local.

El objetivo de la autonomía local es imposible de lograr por completo, ya que existen situaciones en las cuales un sitio dado X debe ceder un cierto grado de control a otro sitio Y. Así pues, sería más correcto expresar el objetivo de la autonomía de esta manera: los sitios deber ser autónomos hasta donde sea posible.

2.- No dependencia de un sitio central

La autonomía local implica que todos los sitios deben tratarse igual; no debe haber dependencia de un sitio central "maestro" para obtener un servicio central, como por ejemplo un procesamiento centralizado de las consultas o una administración centralizada de las transacciones, de modo que todo el sistema dependa de ese sitio central. Este segundo objetivo es por tanto un corolario del primero (si se logra el primero, se logrará por fuerza el segundo). Pero la no dependencia de un sitio central es deseable por sí misma, aun si no se logra la autonomía local completa. Por ello vale la pena expresarlo como un objetivo separado.

La dependencia de un sitio central sería indeseable al menos por las siguientes razones:

- Ese sitio central podría ser un cuello de botella.
- El sistema sería vulnerable; si el sitio central sufriera un desperfecto, todo el sistema dejaría de funcionar.

3.- Operación continua.

En un sistema distribuido, lo mismo que en uno no distribuido, idealmente nunca debería haber necesidad de apagar a propósito el sistema. Es decir, el sistema nunca debería necesitar apagarse para que se pueda realizar alguna función, como añadir un nuevo sitio o instalar una nueva versión del DBMS en un sitio ya existente.

4.- Independencia con respecto a la localización

La idea básica de la independencia con respecto a la localización (también conocida como transparencia de localización) es simple: no debe ser necesario que los usuarios sepan dónde están almacenados físicamente los datos, sino que más bien deben poder comportarse (al menos desde un punto de vista lógico) como si todos los datos estuvieran almacenados en su propio sitio local. La independencia con respecto a la localización es deseable porque simplifica los programas de los usuarios y sus actividades en el terminal. En particular, hace posible la migración de datos de un sitio a otro sin anular la validez de ninguno de esos programas o actividades. Esta posibilidad de migración es deseable porque permite modificar la distribución de los datos dentro de la red en respuesta a cambios en los requerimientos.

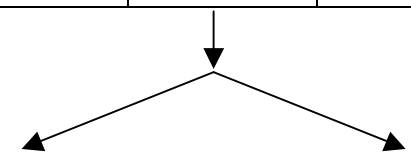
En realidad, esta independencia (y las que veremos) puede considerarse como una extensión de la independencia de los datos.

5.- Independencia con respecto a la fragmentación.

Un sistema maneja fragmentación de los datos si es posible dividir una relación en partes o "fragmentos" para propósitos de almacenamiento físico. La fragmentación es deseable por razones de desempeño: los datos pueden almacenarse en la localidad donde se utilizan con mayor frecuencia, de manera que la mayor parte de las operaciones sea sólo locales y se reduzca el tráfico en la red. Por ejemplo, una relación Empleados podría fragmentase de manera que los registros de los empleados de Ciudad Real se almacenen en el emplazamiento de Ciudad Real, mientras que los empleados de Valdepeñas se almacenan en el emplazamiento de Valdepeñas:

Empleados

NUM_EMP	NUM_DEP	SALARIO
E1	DX	150.000
E2	DY	125.000
E3	DZ	100.000
E4	DY	125.000
E5	DZ	150.000



Fragmento de Ciudad Real

NUM_EMP	NUM_DEP	SALARIO
E1	DX	150.000
E3	DZ	100.000
E5	DZ	150.000

Fragmento de Valdepeñas

NUM_EMP	NUM_DEP	SALARIO
E2	DY	125.000
E4	DY	125.000

Existen en esencia dos clases de fragmentación (aunque veremos que hay más), horizontal y vertical, correspondientes a las operaciones relacionales de restricción y proyección respectivamente. En términos más generales, un fragmento puede ser cualquier subrelación arbitraria que pueda derivarse de la relación original mediante operaciones de restricción y proyección (excepto que, en el caso de la proyección, es obvio que las proyecciones deben conservar la clave primaria de la relación original). La reconstrucción de la relación original a partir de los fragmentos se hace mediante operaciones de reunión y unión apropiadas (reunión en el caso de los fragmentos verticales, unión en el de los horizontales). La facilidad de la fragmentación y de la reconstrucción es una de las muchas razones por las cuales los sistemas distribuidos son relacionales; el modelo relacional ofrece las operaciones precisas requeridas para estas tareas.

Ahora llegamos al punto principal: un sistema que maneja la fragmentación de los datos deberá ofrecer también una independencia con respecto a la fragmentación (transparencia de fragmentación); es decir, los usuarios deberán poder comportarse (al menos desde el punto de vista lógico) como si los datos no estuvieran fragmentados en realidad. La independencia con respecto a la fragmentación (al igual que la independencia con respecto a la localización (es deseable porque simplifica los programas de los usuarios y sus actividades en el terminal. En particular, hace posible refragmentar los datos (y redistribuir los fragmentos) en cualquier momento en respuesta a cambios en los requerimientos de desempeño sin anular la validez de esos programas o actividades.

La independencia con respecto a la fragmentación implica que se presentará a los usuarios una vista de los datos en la cual los fragmentos están combinados lógicamente mediante las reuniones y uniones

apropiadas. Corresponde al optimizador del sistema determinar a cuáles fragmentos físicos es necesario tener acceso para satisfacer cualquier solicitud dada de un usuario.

6.- Independencia de réplica

Un sistema maneja réplica de datos si una relación dada (o, en términos más generales, un fragmento dado de una relación) se puede representar en el nivel físico mediante varias copias almacenadas o réplicas, en muchos sitios distintos.

La réplica es deseable al menos por dos razones: en primer lugar, puede producir un mejor desempeño (las aplicaciones pueden operar sobre copias locales en vez de tener que comunicarse con sitios remotos); en segundo lugar, también puede significar una mejor disponibilidad (un objeto estará disponible para su procesamiento mientras esté disponible por lo menos una copia, al menos para propósitos de recuperación). La desventaja principal de las réplicas es que cuando se actualiza un cierto objeto copiado, deben actualizarse todas las réplicas de ese objeto: el problema de la propagación de actualizaciones.

La réplica, como la fragmentación, debe ser transparente para el usuario. En otras palabras, un sistema que maneja la réplica de los datos deberá ofrecer también una independencia de réplica (transparencia de réplica); es decir, los usuarios deberán poder comportarse (al menos desde un punto de vista lógico) como si sólo existiera una copia de los datos. La independencia de réplica (como la independencia con respecto a la localización y a la fragmentación) es deseable porque simplifica los programas de los usuarios y sus actividades en el terminal. En particular, permite la creación y eliminación dinámicas de las réplicas en cualquier momento en respuesta a cambios en los requerimientos, sin anular la validez de los programas o actividades de los usuarios.

7.- Procesamiento distribuido de consultas.

En este aspecto debemos mencionar dos puntos amplios:

Primero consideremos la consulta “obtener los proveedores de tornillos en Valdepeñas”. Supongamos que el usuario está en la instalación de Ciudad Real y los datos están en el emplazamiento de Valdepeñas. Supongamos también que son n los registros de proveedor que satisfacen la solicitud. Si el sistema es relacional, la consulta implicará en esencia dos mensajes: uno para transmitir la solicitud de Ciudad Real a Valdepeñas, y otro para devolver el conjunto resultante de n registros de Valdepeñas a Ciudad Real. Si, por otro lado, el sistema no es relacional, sino de un registro a la vez, la consulta implicará en esencia 2n mensajes: n de Ciudad Real a Valdepeñas solicitando el siguiente registro, y n de Valdepeñas a Ciudad Real para devolver ese siguiente registro. Así, el ejemplo ilustra el punto de que un sistema relacional tendrá con toda probabilidad un mejor desempeño que uno no relacional (para cualquier consulta que solicite varios registros).

En segundo lugar, la optimización es todavía más importante en un sistema distribuido que en uno centralizado. Lo esencial es que, en una consulta como la anterior, donde están implicados varios sitios, habrá muchas maneras de trasladar los datos en la red para satisfacer la solicitud, y es crucial encontrar una estrategia eficiente, ya que de esta estrategia depende el tiempo de respuesta. Esta es otra razón más por la cual los sistemas distribuidos son siempre relacionales (pues las solicitudes relacionales son optimizables, mientras que las de un registro a la vez no lo son).

8.- Manejo distribuido de transacciones.

El manejo de transacciones tiene dos aspectos principales: el control de recuperación y el control de concurrencia, cada uno de los cuales requiere un tratamiento más amplio en el ambiente distribuido. Para explicar ese tratamiento más amplio es preciso introducir primero un término nuevo, “agente”. En un sistema distribuido, una sola transacción puede implicar la ejecución de código en vario sitios (en particular, puede implicar actualizaciones en varios sitios). Por tanto, se dice que cada transacción está compuesta de

varios agentes, donde un agente es el proceso ejecutado en nombre de una transacción dada en un determinado sitio. Y el sistema necesita saber cuándo dos agentes son parte de la misma transacción; por ejemplo, es obvio que no puede permitirse un bloqueo mutuo entre dos agentes que sean parte de la misma transacción.

Control del recuperación: para asegurar que una transacción dada sea atómica (todo o nada) en el ambiente distribuido, el sistema debe asegurarse de que todos los agentes correspondientes a esa transacción se comprometan al unísono o bien que retrocedan al unísono. Este efecto puede lograrse mediante el protocolo de compromiso en dos fases.

Control de concurrencia: esta función en un ambiente distribuido estará basada con toda seguridad en el bloque, como sucede en los sistemas no distribuidos.

9.- Independencia con respecto al equipo.

Las instalaciones de cómputo en el mundo real, por lo regular incluyen varias máquinas diferentes y existe una verdadera necesidad de poder integrar los datos en todos esos sistemas y presentar al usuario una sola imagen del sistema. Por tanto, es conveniente poder ejecutar el mismo DBMS en diferentes equipos, y además lograr que esos diferentes equipos participen como socios iguales en un sistema distribuido.

10.- Independencia con respecto al sistema operativo.

Este objetivo es en parte un corolario del anterior. Resulta obvia la conveniencia no sólo de poder ejecutar el mismo DBMS en diferentes equipos, sino también de poder ejecutarlo en diferentes sistemas operativos (incluso en diferentes sistemas operativos dentro del mismo equipo) y lograr que (por ejemplo) una versión MVS y una versión UNIX y una versión PC/DOS participen todas en el mismo sistema distribuido.

11.- Independencia con respecto a la red.

Si el sistema ha de poder manejar múltiples sitios diferentes, con equipo distinto y diferentes sistemas operativos, resulta obvia la conveniencia de poder manejar también varias redes de comunicaciones distintas.

12.- Independencia con respecto al DBMS.

Bajo este título consideramos las implicaciones de relajar la suposición de homogeneidad estricta. Puede alegarse que esa suposición es quizá demasiado rígida. En realidad, no se requiere sino que los DBMS en los diferentes sitios manejen todos la misma interfaz; no necesitan ser por fuerza copias del mismo sistema. Por ejemplo, si tanto Ingres como Oracle manejaran la norma oficial de SQL, podría ser posible lograr una comunicación entre los dos en el contexto de un sistema distribuido. Dicho de otro modo, el sistema distribuido podría ser heterogéneo, al menos hasta cierto grado.

Definitivamente sería deseable poder manejar la heterogeneidad. Una vez más, en la realidad las instalaciones de cómputo no sólo suelen emplear varias máquinas diferentes y varios sistemas operativos distintos, sino que también ejecutan diferentes DBMS; y sería agradable que todos esos DBMS distintos pudieran participar de alguna manera en un sistema distribuido. En otras palabras, el sistema distribuido ideal deberá ofrecer independencia respecto al DBMS.

4. VENTAJAS E INCONVENIENTES

La distribución de los datos y aplicaciones tiene ventajas potenciales con respecto a los sistemas de bases de datos centralizados. Desgraciadamente, hay también desventajas. En este punto vamos a ver las ventajas e inconvenientes de los SGBDD.

Existen varias razones que justifican la construcción de sistemas distribuidos de bases de datos:

- **Compartimiento de datos**: La mayor ventaja de los sistemas distribuidos de bases de datos es que proporcionan un entorno en el que los usuarios de un emplazamiento pueden ser capaces de acceder a los datos que residen en otros emplazamientos. Por ejemplo, un usuario del emplazamiento de Ciudad Real puede acceder a los datos del emplazamiento de Valdepeñas. Sin esta capacidad, un usuario que deseara realizar una transferencia entre dos emplazamientos distintos tendría que recurrir a algún mecanismo externo para poner en contacto los sistemas existentes.

- **Autonomía**: La ventaja principal del compartimiento de datos por medio de la distribución de los mismos es que cada emplazamiento puede conservar un cierto grado de control sobre los datos que tiene almacenados localmente. En un sistema centralizado hay un administrador local de la base de datos que es responsable del sistema completo. Cada administrador local de las bases de datos recibe una parte de las responsabilidades del administrador central. Cada administrador puede tener un grado de autonomía local diferente, dependiendo del diseño del sistema distribuido de bases de datos. La posibilidad de autonomía local es, con frecuencia, una ventaja fundamental de las bases de datos distribuidas.

- Disponibilidad: Si en un sistema distribuido falla un emplazamiento, los emplazamientos restantes pueden continuar funcionando. En particular si se duplican los elementos de datos en varios emplazamientos, una transacción que necesite un determinado elemento de datos puede encontrarlo en cualquiera de dichos emplazamientos. De esta manera, el fallo de un emplazamiento no implica necesariamente el cierre del sistema.

El sistema debe ser capaz de detectar un fallo en uno de los emplazamientos, de modo que pueda decidir si es necesario realizar alguna acción de recuperación. El sistema debe dejar de utilizar los servicios del emplazamiento que ha fallado. Por último, deben existir mecanismos para reintegrar fácilmente en el sistema al emplazamiento que falló cuando se haya recuperado o se haya reparado.

Aunque la recuperación de fallos es más compleja en los sistemas distribuidos que en los centralizados, la capacidad de la mayoría del sistema para continuar funcionando a pesar del fallo de un emplazamiento revierte en una disponibilidad creciente. La disponibilidad es un factor crucial en los sistemas de bases de datos que se utilizan para aplicaciones de tiempo real. La pérdida temporal del acceso a los datos de, por ejemplo, una compañía aérea puede conducir a la pérdida de compradores potenciales de billetes en beneficio de la competencia.

- Crecimiento modular. En un entorno distribuido, es mucho más fácil manejar la expansión. Se pueden añadir nuevas localizaciones a la red sin afectar las operaciones de las otras localizaciones. Esta flexibilidad permite a una que una organización se pueda expandir relativamente fácilmente. Es posible incrementar el tamaño de la base de datos añadiendo potencia de procesamiento y almacenamiento a la red. En un SGBD centralizado, el crecimiento puede conllevar cambios al hardware (la adquisición de un sistema más potente) y software (la adquisición de un SGBD más potente o más configurable).

El principal inconveniente de los sistemas distribuidos de bases de datos es la complejidad añadida que es necesaria para garantizar la coordinación apropiada entre los emplazamientos. Esta creciente complejidad tiene varias facetas:

- **Coste de desarrollo del software**. La implementación de un sistema distribuido de bases de datos es más difícil y por tanto más costoso.

- **Rendimiento**. En una base de datos centralizada, el camino desde el SGBD a los datos tiene una velocidad de acceso de unos pocos milisegundos y una velocidad de transferencia de varios millones de caracteres por segundo. Incluso en una red de área local rápida, las velocidades de acceso se alargan de décimas de segundo y las velocidades de transferencia caen hasta 100.000 caracteres por segundo o menos. En un enlace por módem a 9.600 baudios, el acceso a los datos puede tardar segundos o minutos, y 500

caracteres por segundo puede ser la máxima productividad. Esta gran diferencia en velocidades puede hacer bajar dramáticamente el rendimiento del acceso a datos remotos.

- **Integridad**. Si las transacciones distribuidas van a seguir siendo proposiciones "todo o nada", éstas requieren la cooperación activa de dos o más copias independientes del software SGBD que se está ejecutando en sistemas informáticos diferentes. Deben utilizarse protocolos especiales de transacción "confirmación en dos fases".

- **Optimización**. Cuando se accede a los datos a través de una red, no son aplicables las reglas normales de optimización de SQL. Por ejemplo, puede ser más eficaz rastrear secuencialmente una tabla local completa que realizar una búsqueda por índice en una tabla remota. El software de optimización debe conocer la(s) red(es) y su(s) velocidad(es). En general, la optimización pasa a ser más crítica y más difícil.

- **Compatibilidad de datos**. Diferentes sistemas informáticos soportan diferentes tipos de datos, e incluso cuando dos sistemas ofrecen los mismos tipos de datos utilizan con frecuencia formatos diferentes. Por ejemplo, un VAX y un Macintosh almacenan los enteros de 16 bits de forma distinta. Los mainframes IBM almacenan códigos de caracteres EBCDIC mientras que las minicomputadoras y los PC utilizan ASCII. Un SGBD distribuido debe enmascarar esas diferencias.

- **Catálogos del sistema**. Cuando un SGBD realiza sus tareas, accede frecuentemente a sus catálogos de sistema. ¿Dónde debería mantenerse el catálogo en una base de datos distribuida?. Si está centralizado en un sistema, el acceso remoto a él será lento, afectando al SGBD. Si está distribuido a través de muchos sistemas diferentes, los cambios deben propagarse y sincronizarse por toda la red.

- **Entorno con diferentes vendedores**. Es altamente improbable que todos los datos de una organización puedan ser gestionados por un único producto SGBD, por lo que el acceso a base de datos distribuida atravesará fronteras de productos SGBD. Esto requiere la cooperación activa entre productos SGBD de vendedores altamente competitivos (una perspectiva improbable).

- **Interbloqueos distribuidos**. Cuando las transacciones de dos sistemas diferentes tratan de acceder a datos bloqueados en el otro sistema, puede ocurrir un interbloqueo en la base de datos distribuida, incluso aunque el interbloqueo no sea visible a ninguno de los dos sistemas por separado. El SGBD debe detectar interbloqueos globales en una base de datos distribuida.

- **Recuperación**. Si uno de los sistemas donde se ejecuta un SGBD distribuido falla, el operador de ese sistema debe ser capaz de ejecutar sus procedimientos de recuperación con independencia del resto de los sistemas en la red, y el estado recuperado de la base de datos debe ser consistente con el de los otros sistemas.

5. SGBD HOMOGÉNEOS Y HETEROGÉNEOS

Un SGBDD se puede clasificar como homogéneo o heterogéneo. En un sistema homogéneo, todas las localizaciones usan el mismo SGBD. En un sistema heterogéneo, las localizaciones pueden ejecutar diferentes SGBD, que no tienen por qué basarse en el mismo modelo de datos subyacente, y el sistema puede estar compuesto de SGBDs relacionales, en red, jerárquicos y orientados a objetos.

Los sistemas homogéneos son más fáciles de diseñar y gestionar. Este enfoque proporciona crecimiento incremental, siendo más fácil añadir nuevas localizaciones al SGBDD, y permitiendo mejorar el funcionamiento explotando la capacidad de procesamiento paralelo en varias localizaciones.

Los sistemas heterogéneos normalmente aparecen cuando localizaciones individuales han implementado sus propias bases de datos y la integración de éstas es muy costosa. En un sistema heterogéneo, se requieren traducciones para permitir la comunicación entre SGBDs diferentes. Para proporcionar transparencia, los usuarios deben poder hacer solicitudes en el lenguaje de su SGBD local. El

sistema entonces tiene la tarea de localizar los datos y realizar todas las traducciones necesarias. Se realizan solicitudes de datos de otros sistemas que tienen:

- Diferente hardware
- Diferente SGBD
- Diferente hardware y SGBD

Si el hardware es diferente pero el SGBD es el mismo, la traducción es directa, implicando el cambio de códigos y longitudes de palabras. Si el SGBD es diferente, la traducción es complicada, implicando la traducción de las estructuras de datos de un modelo de datos al equivalente en otro modelo. Por ejemplo, las relaciones en el modelo relacional se transforman en registros (records) y conjuntos (sets) en el modelo de red. Es también necesario traducir el lenguaje de consultas usado (por ejemplo, la sentencia Select en SQL se traduce en la sentencia Find y Get en el modelo en red). Si el hardware y software son diferentes, entonces se requieren los dos tipos de traducciones. Esto hace que el procesamiento sea extremadamente complejo.

Una complejidad añadida es la necesidad de un esquema conceptual común, que se forma mediante la integración de los esquemas conceptuales locales. Esta integración puede ser muy complicada debido la heterogeneidad semántica. Por ejemplo, atributos con el mismo nombre en dos esquemas pueden representar cosas diferentes. Igualmente, atributos con diferentes nombres pueden modelar la misma cosa.

La solución típica usada por algunos sistemas relaciones que forman parte de un sistema heterogéneo de bases de datos distribuidas es usar *gateways* (combinación de hardware y software que comunica dos tipos diferentes de redes), que convierten el lenguaje y modelo de cada diferente SGBD al lenguaje y modelo del sistema relacional. Sin embargo, esta aproximación tiene algunas limitaciones serias. Primero, no soporta gestión de transacciones, incluso para un par de sistemas. En otras palabras, la gateway entre dos sistemas es meramente un traductor de consultas. Por ejemplo, un sistema no puede coordinar el control de concurrencia y recuperación de transacciones que implique actualizaciones en ambas bases de datos. Segundo, esta aproximación concierne solo con el problema de traducir una consulta expresada en un lenguaje a la consulta equivalente en otro lenguaje. Así pues , no está dirigida al problema de homogeneizar las diferencias estructurales y de representación entre diferentes esquemas.

Sistemas con múltiples bases de datos

Sistema con múltiples bases de datos: Es un sistema de base de datos distribuido en el que cada localización mantiene una completa autonomía.

En los últimos años se han desarrollado nuevas aplicaciones de bases de datos que necesitan datos de gran variedad de bases de datos ya existentes ubicadas en una colección heterogénea de entornos de hardware y de software. El tratamiento de la información ubicada en bases de datos heterogéneas exige otra capa más de software por encima de los sistemas de bases de datos existentes. Esta capa de software se denomina sistema con múltiples bases de datos. Puede que los sistemas locales de bases de datos utilicen modelos lógicos y lenguajes de definición y de tratamiento de datos diferentes, y es posible que se diferencien en los mecanismos de control de la concurrencia y de administración de transacciones. Los sistemas con múltiples bases de datos crean la ilusión de la integración lógica de bases de datos sin exigir su integración física.

La integración completa de los sistemas existentes en una base de datos distribuida homogénea resulta con frecuencia difícil o imposible:

- Dificultades técnicas. La inversión en programas de aplicaciones basados en los sistemas de bases de datos existentes puede ser enorme y el coste de transformar estas aplicaciones puede resultar prohibitivo.
- Dificultades de las organizaciones. Puede que aunque la integración sea técnicamente posible, no lo sea políticamente, debido a los sistemas de bases de datos existentes pertenecientes a diferentes empresas u organizaciones.

En el segundo caso, es importante que los sistemas con múltiples bases de datos permitan a los sistemas locales de bases de datos conservar un elevado grado de autonomía respecto a la base de datos local y a las transacciones que se ejecuten con esos datos. Por este motivo, los sistemas con múltiples bases de datos ofrecen ventajas significativas que superan la sobrecarga que ocasionan.

6. DISEÑO DE BASES DE DATOS DISTRIBUIDAS

Consideremos una relación *r* que debe guardarse en la base de datos. Hay varios enfoques del almacenamiento de esa relación en la base de datos distribuida:

- Réplica. El sistema conserva varias réplicas (copias) idénticas de la relación. Cada réplica se guarda en un emplazamiento diferente, lo que da lugar a la réplica de los datos. La alternativa a la réplica es guardar sólo una copia de la relación r.

- Fragmentación. La relación se divide en varios fragmentos. Cada fragmento se guarda en un emplazamiento diferente.

- Réplica y fragmentación. La relación se divide en varios fragmentos. El sistema conserva réplicas de cada fragmento.

6.1 Réplica de los datos.

Si se replica la relación *r*, se guardará una copia de la misma en dos o mas emplazamientos. En el caso más extremo, se tendrá una réplica completa, en la que se guarda una copia en cada emplazamiento del sistema.

La réplica presenta un cierto número de ventajas e inconvenientes:

- Disponibilidad. Si falla uno de los emplazamientos que contienen la relación r, ésta aún se podrá encontrar en otro emplazamiento. Por tanto, el sistema puede seguir procesando las consultas que impliquen a r, a pesar del fallo en un emplazamiento.

- Aumento del paralelismo. En el caso de que la mayor parte de los accesos a la relación r sea de lectura de la misma, varios emplazamientos pueden procesar en paralelo las consultas que impliquen a r. Cuantas más réplicas de r haya, mayor será la posibilidad de que el dato buscado se halle en el emplazamiento en que se esté ejecutando la transacción. Por tanto, la réplica de los datos minimiza el tráfico de datos entre los emplazamientos.

- Aumento de la sobrecarga en las actualizaciones. El sistema debe asegurarse de que todas las réplicas de la relación r sea consistentes; en caso contrario, puede dar lugar a resultados erróneos. Por tanto, siempre que se actualice r, la actualización debe extenderse a todos los emplazamientos que contengan réplicas. La consecuencia es un aumento en la sobrecarga. Por ejemplo, en un sistema bancario en el que la información sobre las cuentas se replican en varios emplazamientos, hay que asegurarse de que el saldo de una cuenta concreta coincida en todos los emplazamientos.

En general, la réplica mejora el rendimiento de las operaciones de lectura y aumenta la disponibilidad de los datos para las transacciones de sólo lectura. Sin embargo, las transacciones de actualización suponen una sobrecarga mayor. Por tanto, un buen parámetro para afrontar el grado de réplica

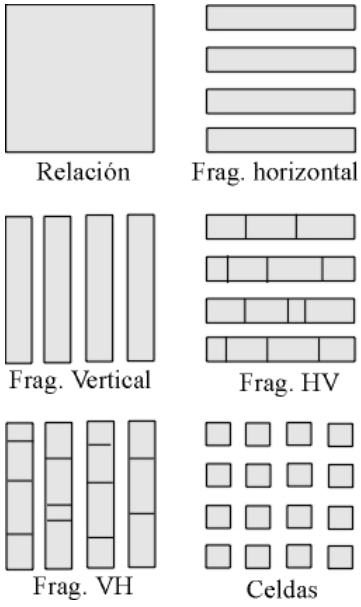
consistiría en sopesar la cantidad de consultas de lectura que se efectuarán, así como el número de consultas de escritura que se llevarán a cabo. En una red donde las consultas que se procesen sean mayoritariamente de lectura, se podría alcanzar un alto grado de réplica, no así en el caso contrario. El control de las actualizaciones concurrentes de los datos replicados por varias transacciones es más complicado que cuando se utiliza el enfoque centralizado del control de concurrencia. Se puede simplificar la administración de las réplicas de la relación r escogiendo una de ellas como copia principal de r. Por ejemplo, en un sistema bancario se puede asociar una cuanta con el emplazamiento en que se ha abierto. De manera parecida, en un sistema de reservas de unas líneas aéreas se puede asociar un vuelo con el punto en el que se origina.

6.2 Fragmentación de los datos.

Si la relación r está fragmentada, se dividirá en cierto número de fragmentos *r1,r2,...,rn*. Estos fragmentos contendrán suficiente información como para permitir la reconstrucción de la relación original *r*.

Dado que una relación se corresponde esencialmente con una tabla y la cuestión consiste en dividirla en fragmentos menores, inmediatamente surgen dos alternativas lógicas para llevar a cabo el proceso: la división horizontal y la división vertical.

La división o fragmentación horizontal trabaja sobre las tuplas, dividiendo la relación en subrelaciones que contienen un subconjunto de las tuplas que alberga la primera. La fragmentación vertical, en cambio, se basa en los atributos de la relación para efectuar la división. Estos dos tipos de partición podrían considerarse los fundamentales y básicos. Sin embargo, existen otras alternativas. Fundamentalmente, se habla de fragmentación mixta o híbrida cuando el proceso de partición hace uso de los dos tipos anteriores. La fragmentación mixta puede llevarse a cabo de tres formas diferentes: desarrollando primero la fragmentación vertical y, posteriormente, aplicando la partición horizontal sobre los fragmentos verticales (denominada partición VH), o aplicando primero una división horizontal para luego, sobre los fragmentos generados, desarrollar una fragmentación vertical (llamada partición HV), o bien, de forma directa considerando la semántica de las transacciones. Otro enfoque distinto y relativamente nuevo, consiste en aplicar sobre una relación, de forma simultánea y no secuencial, la fragmentación horizontal y la fragmentación vertical; en este caso, se generara una rejilla y los fragmentos formaran las celdas de esa rejilla, cada celda será exactamente un fragmento vertical y un fragmento horizontal (nótese que en este caso el grado de fragmentación alcanzado es máximo, y no por ello la descomposición resultará más eficiente).



Grado de fragmentación. Cuando se va a fragmentar una base de datos deberíamos sopesar qué grado de fragmentación va a alcanzar, ya que éste será un factor que influirá notablemente en el desarrollo de la ejecución de las consultas. El grado de fragmentación puede variar desde una ausencia de la división, considerando a las relaciones unidades de fragmentación; o bien, fragmentar a un grado en el cada tupla o atributo forme un fragmento. Ante estos dos casos extremos, evidentemente se ha de buscar un compromiso

intermedio, el cual debería establecerse sobre las características de las aplicaciones que hacen uso de la base de datos. Dichas características se podrán formalizar en una serie de parámetros. De acuerdo con sus valores, se podrá establecer el grado de fragmentación del banco de datos.

Reglas de corrección de la fragmentación.

A continuación se enuncian las tres reglas que se han de cumplir durante el proceso de fragmentación, las cuales asegurarán la ausencia de cambios semánticos en la base de datos durante el proceso.

1. Plenitud. Si una relación R se descompone en una serie de fragmentos R1, R2, ..., Rn, cada elemento de datos que pueda encontrarse en R deberá poder encontrarse en uno o varios fragmentos Ri. Esta propiedad extremadamente importante asegura que los datos de la relación global se proyectan sobre los fragmentos sin pérdida alguna. Tenga en cuenta que en el caso horizontal el elemento de datos, normalmente, es una tupla, mientras que en el caso vertical es un atributo.

2. Reconstrucción. Debe ser posible definir una operación relacional que permita reconstruir la relación R a partir de los fragmentos. Esta regla asegura que se preserven las dependencias funcionales.

3. Disyunción. Si una relación R se descompone horizontalmente en una serie de fragmentos R1, R2, ..., Rn, y un elemento de datos *di* se encuentra en algún fragmento Rj, entonces no se encuentra en otro fragmento Rk (k<> j). Esta regla asegura que los fragmentos horizontales sean disjuntos. Si una relación R se descompone verticalmente, sus atributos primarios clave normalmente se repiten en todos sus fragmentos.

La fragmentación vertical es una excepción a esta regla, donde las claves primarias deben repetirse para permitir la reconstrucción. Esta regla asegura la mínima redundancia de los datos.

Fragmentación horizontal

Tomemos como ejemplo la siguiente relación *cuenta*:

Nombre-sucursal	Número-cuenta	Saldo
Guadarrama	C-305	100.000
Guadarrama	C-226	64.200
Cercedilla	C-177	41.000
Cercedilla	C-402	2.000.000
Guadarrama	C-155	12.400
Cercedilla	C-408	224.600
Cercedilla	C-639	150.000

La relación *r* se divide en cierto número de subconjuntos r1, r2, rn. Cada tupla de la relación r debe pertenecer al menos a uno de los fragmentos, de modo que se pueda reconstruir la relación original si fuera necesario.

Un fragmento puede definirse como una selección de la relación global r. Es decir, se utiliza un predicado Pi para construir un fragmento ri de la manera siguiente:

$$ri = \sigma_{Pi}(r)$$

Se puede obtener la reconstrucción de la relación r tomando la unión de todos los fragmentos, es decir:

$$r = r1 \cup r2 \cup .... \cup rn$$

La relación *cuenta* se puede dividir en n fragmentos diferentes, cada uno de los cuales consiste en tuplas de cuentas que pertenecen a una sucursal concreta. Si el sistema bancario sólo tiene dos sucursales (Guadarrama y Cercedilla), entonces sólo habrá dos fragmentos diferentes:

$$\text{cuenta1} = \sigma_{\text{nombre-sucursal="Guadarrama"}} (\text{cuenta})$$
$$\text{cuenta2} = \sigma_{\text{nombre-sucursal="Cercedilla"}} (\text{cuenta})$$

Los dos fragmentos se muestran a continuación:

Cuenta1

Nombre-sucursal	Número-cuenta	Saldo
Guadarrama	C-305	100.000
Guadarrama	C-226	64.200
Guadarrama	C-155	12.400

Cuenta2

Nombre-sucursal	Número-cuenta	Saldo
Cercedilla	C-177	41.000
Cercedilla	C-402	2.000.000
Cercedilla	C-408	224.600
Cercedilla	C-639	150.000

En este ejemplo los fragmentos son disjuntos. Cambiando los predicados de selección utilizados para generar los fragmentos, se puede hacer que una tupla de r determinada aparezca en más de una de la ri. Esta forma de réplica de los datos se discute con más profundidad más adelante.

Fragmentación vertical

La fragmentación vertical de r(R) implica la definición de varios subconjuntos de atributos R1, R2, ..., Rn del esquema R tales que

$$R = R1 \cup R2 \cup \ldots \cup Rn$$

Cada fragmento ri de r queda definido por

$$ri = \Pi_{Ri} (r)$$

La fragmentación debe hacerse de modo que se pueda reconstruir la relación r a partir de los fragmentos tomando la reunión natural:

$$r = r1 \bowtie r2 \bowtie r3 \bowtie \ldots \bowtie rn$$

Un modo de asegurar que la relación r pueda reconstruirse es incluir los atributos de la clave primaria de R en cada uno de los Ri. De manera más general, puede utilizarse cualquier superclave. A menudo resulta conveniente añadir un atributo especial, denominado id-tupla, al esquema R. El valor de id-

tupla de una tupla es único, y se utiliza para distinguir esa tupla de las demás. El atributo id-tupla es, por tanto, un candidato a clave del esquema ampliado, y se incluye en cada uno de los Ri. La dirección física o lógica de una tupla puede utilizarse como id-tupla, dado que cada tupla tiene una dirección única.

Para ilustrar la fragmentación vertical se considerará relación depósito:

Nombre-sucursal	Número-cuenta	Nombre-cliente	Saldo	Id-tupla
Guadarrama	C-305	García	100.000	1
Guadarrama	C-226	Cordero	64.200	2
Cercedilla	C-177	Cordero	41.000	3
Cercedilla	C-402	Obeso	2.000.000	4
Guadarrama	C-155	Obeso	12.400	5
Cercedilla	C-408	Obeso	224.600	6
Cercedilla	C-639	Badorrey	150.000	7

A continuación se muestra una fragmentación vertical para la relación depósito:

Esquema-depósito-1 = (nombre-sucursal, nombre-cliente, id-tupla)
Esquema-depósito-2 = (número-cuenta, saldo, id-tupla)

depósito1

Nombre-sucursal	Nombre-cliente	Id-tupla
Guadarrama	García	1
Guadarrama	Cordero	2
Cercedilla	Cordero	3
Cercedilla	Obeso	4
Guadarrama	Obeso	5
Cercedilla	Obeso	6
Cercedilla	Badorrey	7

depósito2

Número-cuenta	Saldo	Id-tupla
C-305	100.000	1
C-226	64.200	2
C-177	41.000	3
C-402	2.000.000	4
C-155	12.400	5
C-408	224.600	6
C-639	150.000	7

Estas dos relaciones resultan de procesar:

$$\text{depósito1} = \sigma_{\text{Esquema-depósito-1}}(\text{depósito})$$
$$\text{depósito2} = \sigma_{\text{Esquema-depósito-2}}(\text{depósito})$$

Para reconstruir la relación depósito original a partir de los fragmentos hay que procesar:

$$\sigma_{\text{Esquema-depósito}}(\text{depósito1} \bowtie \text{depósito2})$$

La expresión depósito1 $\rtimes$ depósito2 es una forma especial de reunión natural. El atributo de reunión es id-tupla. Aunque el atributo id-tupla facilite la aplicación de la división vertical, no debe ser visible para los usuarios, dado que es un mecanismo interno de la aplicación y viola la independencia de los datos, que es una de las virtudes principales del modelo relacional.

Fragmentación mixta

La relación *r* se divide en una serie de relaciones fragmentarias r1, r2, ..., rn. Cada fragmento se obtiene como resultado de la aplicación del esquema de fragmentación horizontal o vertical a la relación r, o a un fragmento de r obtenido con anterioridad.

Supongamos la relación depósito vista anteriormente. Esta relación se divide en un principio en los fragmentos depósito1 y depósito2, tal y como se definieron anteriormente. Ahora se puede volver a dividir el fragmento depósito1 utilizando el esquema de fragmentación horizontal en los dos fragmentos siguientes:

$$\text{depósito1a} = \sigma_{\text{nombre-sucursal=”Guadarrama”}}(\text{depósito1})$$
$$\text{depósito1b} = \sigma_{\text{nombre-sucursal=”Cercedilla”}}(\text{depósito1})$$

Por tanto, la relación r queda dividida en tres fragmentos: depósito1a, depósito1b y depósito2. Cada uno de los fragmentos puede residir en un emplazamiento diferente.

6.3 Réplica y fragmentación de datos.

Las técnicas descritas anteriormente para la réplica y fragmentación de datos pueden aplicarse de manera sucesiva a la misma relación. Es decir, se puede replicar un fragmento, las réplicas de los fragmentos se pueden volver a fragmentar, etc. Por ejemplo, considérese un sistema distribuido que consista en los emplazamientos E1, E2, ..., E10. Se puede framentar depósito en depósito1a, depósito1b y depósito2, y, por ejemplo, guardar una copia de depósito1a en los emplazamientos E1, E3 y E7; una copia de depósito1b en los emplazamientos E7 y E10 y una copia de depósito2 en los emplazamientos E2, E8 y E9.

BIBLIOGRAFIA

Thomas Connolly ; Database Systems: A practical approach to design, implementation and management ; Second Edition; Addison-Wesley. Capítulo 19.

Abraham Silberschartz; Fundamentos de bases de datos; tercera edicion; Mc Graw Hill. Capítulos 16, 18.

David M. Kroenke; Procesamiento de bases de datos; 5ª edición; Prentice Hall; Capítulo 16.

Gary N. Hansen, James V. Hansen; Diseño y administración de bases de datos; 2ª edición; Prentice Hall; Capítulos 1, 9.

Elmasri/Navathe; Sistemas de bases de datos; 2ª edición; Addison Wesley; Capítulos 2, 23.

C.J. Date; Introducción a los sistemas de bases de datos; 5ª edición; Addison Wesley; Capítulos 2, 23.

La biblia de Oracle 8; Anaya Multimedia.

Michael Abbey, Michael J. Corey; Oracle 8: guía de aprendizaje; Osborne/McGrawHill

Páginas Web:

Members.es.tripod.de/jodr35/index.htm

www.oracle.com

www.map.es/csi/silice