Base de datos distribuida

Una Base de Datos Distribuida es, una base de datos construida sobre una red computacional y no por el contrario en una máquina aislada. La información que constituye la base de datos esta almacenada en diferentes sitios en la red, y las aplicaciones que se ejecutan accesan datos en distintos sitios.

Una Base de Datos Distribuida entonces es una colección de datos que pertenecen lógicamente a un sólo sistema, pero se encuentra físicamente esparcido en varios "sitios" de la red. Un sistema de base de datos distribuidas se compone de un conjunto de sitios, conectados entre sí mediante algún tipo de red de comunicaciones, en el cual :

- Cada sitio es un sistema de base de datos en sí mismo, pero,
- Los sitios han convenido en trabajar juntos (si es necesario) con el fin de que un usuario de cualquier sitio pueda obtener acceso a los datos de cualquier punto de la red tal como si todos los datos estuvieran almacenados en el sitio propio del usuario.

En consecuencia, la llamada "*base de datos distribuida*" es en realidad una especie de objeto virtual, cuyas partes componentes se almacenan físicamente en varias bases de datos "reales" distintas ubicadas en diferentes sitios. De hecho, es la unión lógica de esas bases de datos. En otras palabras, cada sitio tiene sus propias bases de datos "reales" locales, sus propios usuarios locales, sus propios DBMS y programas para la administración de transacciones (incluyendo programas de bloqueo, bitácoras, recuperación, etc), y su propio administrador local de comunicación de datos (administrador DC). En particular un usuario dado puede realizar operaciones sobre los datos en su propio sitio local exactamente como si ese sitio no participara en absoluto en el sistema distribuido (al menos, ése es uno de los objetivos). Así pues, el sistema de bases de datos distribuidas puede considerarse como una especie de sociedad entre los DBMS individuales locales de todos los sitios. Un nuevo componente de software en cada sitio (en el aspecto lógico, una extensión del DBMS local) realiza las funciones de sociedad necesarias; y es la combinación de este nuevo componente y el DBMS ya existente lo que constituye el llamado "sistema de administración de bases de datos distribuidas" (DDBMS, distributed database management system).

[Base de Datos Distribuidas]

Conceptos básicos.

El sistema de administración de Base de Datos Distribuida (DDBMS), esta formado por las transacciones y los administradores de base de datos distribuidos de todas las computadoras. Tal DDBMS en un esquema genérico implica un conjunto de programas que operan en diversas computadoras. Estos programas pueden ser subsistemas de un producto único DDBMS, concesionado por un sólo fabricante, o también pudiera resultar de una colección de programas de fuentes dispares : algunos considerados por fabricantes y algunos otros escritos en casa.

Un administrador de base de datos (DTM) es un programa que recibe solicitudes de procesamiento de los programas de consulta o de transacciones y a su vez las traduce en acciones para los administradores de la

base de datos . Una función importante del DTM es coordinar y controlar dichas acciones. · Cada sitio tiene sus propias bases de datos "reales" locales, sus propios usuarios locales, sus propios DBMS y programas para administración de transacciones y su propio administrador local de comunicación de datos. La diferencia principal entre los sistemas de bases de datos centralizados y los distribuidos es que en los primeros, los datos residen en una sola localidad, mientras que, en lo últimos, se encuentran en varias localidades. Cada localidad puede procesar transacciones locales , es decir, aquellas que sólo acceden a datos que residen en esa localidad. Además, una localidad puede participar en la ejecución de transacciones globales , es decir, aquellas que acceden a datos de varias localidades, ésta requiere comunicación entre las localidades. · Una transacción local es la que accede a cuentas en la localidad individual donde se inicio. En cambio, una transacción global accede a cuentas de una localidad distinta a la localidad donde se inicio o a cuentas de varias localidades diferentes.

Ejemplo 1.1

Considere un banco que tiene tres sucursales, en cada sucursal, un computador controla las terminales de la misma y el sistema de cuentas. Cada computador con su sistema de cuentas local en cada sucursal constituye un "sitio" de la BDD; las computadoras están conectadas por la red. Durante las operaciones normales, las aplicaciones en las terminales de la sucursal necesitan solo accesar la BD de la misma. Como solo accesan la misma red local, se les llaman aplicaciones locales .

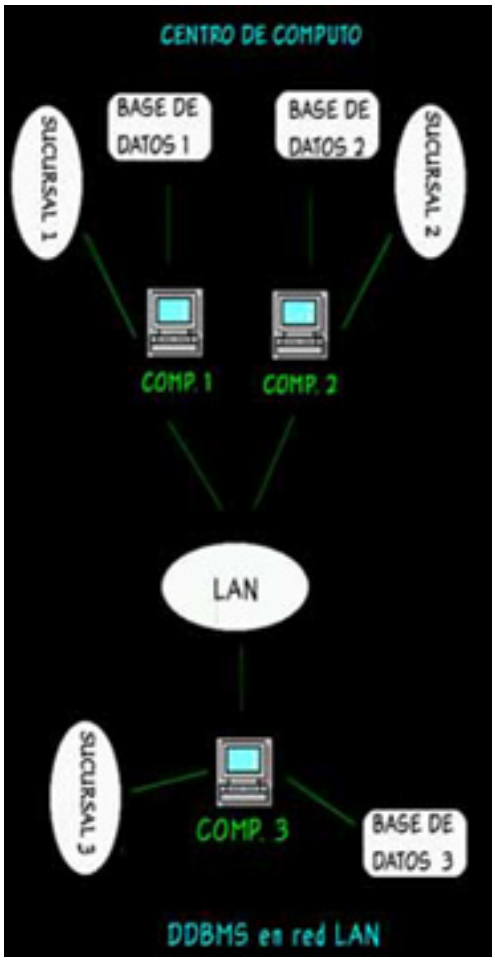
Desde el punto de vista tecnológico, aparentemente lo importante es la existencia de algunas transacciones que accesen información en más de una sucursal. Éstas transacciones son llamadas transacciones globales o transacciones distribuidas.
La existencia de transacciones globales será considerada como una característica que nos ayude a discriminar entre las BDD y un conjunto de base de datos locales.

Una típica transacción global sería una transferencia de fondos de una sucursal a otra. Esta aplicación requiere de actualizar datos en dos diferentes sucursales y asegurarse de la real actualización en ambos sitios o en ninguno. Asegurar el buen funcionamiento de aplicaciones globales es una tarea difícil. En el ejemplo 1.1 las computadoras estaban geográficamente en diferentes puntos; también, BDD pueden ser construidas en una red local.

[Base de Datos Distribuida geográficamente dispersada]

Ejemplo 1.2

Considere el mismo banco del ejemplo previo, con las mismas aplicaciones, pero con un sistema configurado como en la figura. Los mismos procesadores con sus bases de datos han sido movidos de sus sucursales a un edificio común y ahora están conectados entre si en un radio con un amplio ancho de banda. Las terminales de las sucursales están conectadas a sus respectivos computadores por líneas telefónicas. Cada procesador y su base de datos constituye un sitio de la red local.



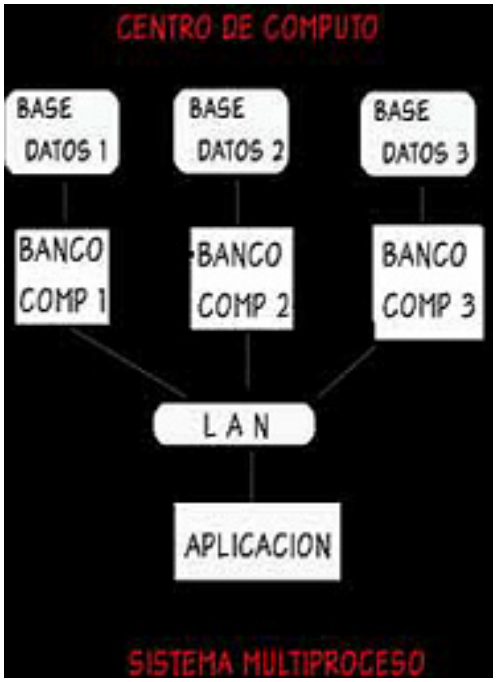
[Base de Datos Distribuidas en una LAN]

Vemos que la estructura física de las conexiones a cambiado con respecto al ejemplo 1, pero las características de la arquitectura son las mismas. En particular, los mismos computadores ejecutan las mismas aplicaciones, accesando las mismas bases de datos. La transacción local del ejemplo anterior aún es local, no por el hecho geográfico, si no por el hecho de que solo un computador por bases de datos está envuelto en el proceso. Si hay aplicaciones globales, es conveniente considerar este ejemplo como BDD, ya que muchas características que el ejemplo previo presentó son aún válidas. A pesar de todo, el hecho de que la BDD sea implementada en una red local o en una gráficamente distribuida, cambian muchas veces el tipo de solución que se busca para un problema.

Ejemplo 1.3

¿Qué no es una Base de Datos Distribuida?

Un caso de sistema NO considerado BDD : Considere el mismo banco del ejemplo anterior, pero con la configuración del sistema mostrado en la figura 1.3. La información en diferentes sucursales esta distribuida en tres computadores ("backend" computers), que realizan el control de funciones de la base de datos. Las aplicaciones son ejecutadas por diferentes computadores.



[BDD sistema multiproceso]

La razón para no considerar esta una base de datos distribuida: aún cuando la información se encuentra físicamente distribuida en diferentes procesadores, su distribución, no es relevante desde el punto de vista de la aplicación. Lo que perdemos aquí es la existencia de aplicaciones locales, en el sentido de que la integración del sistema ha alcanzado el punto donde ninguno de los computadores será capaz de ejecutar una transacción por si mismo.

¿Por qué son deseables las bases de datos distribuidas?

La respuesta básica a esta pregunta es que por lo regular las empresas ya están distribuidas, por lo menos desde el punto de vista lógico (en divisiones, departamentos, proyectos, etc) y muy probablemente en el sentido físico también (en plantas, talleres, laboratorios, y demás), de lo cual se desprende que en general la información ya está también distribuida, porque cada unidad de organización dentro de la empresa mantendrá por fuerza los datos pertinentes a su propio funcionamiento. Así pues, un sistema distribuido permite que la estructura de la base de datos refleje la estructura de la empresa : los datos locales se pueden mantener en forma local, donde por lógica deben estar, pero al mismo tiempo es posible obtener acceso a datos remotos en caso necesario.

Un principio fundamental de los sistemas de bases de datos distribuidos

El principio fundamental de las bases de datos distribuidas :

Desde el punto de vista del usuario, un sistema distribuido deberá ser idéntico 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 son (o deberían ser) internos o a nivel de realización, no externos o a nivel del usuario. Llamaremos al principio fundamental recién identificado la "regla cero" de los sistemas distribuidos. La regla cero conduce a varios objetivos o reglas secundarios − doce en realidad− siguientes :

- *Autonomía local.*
- *No dependencia de un sitio central.*
- *Operación continua.*
- *Independencia con respecto a la localización.*
- *Independencia con respecto a la fragmentación.*
- *Independencia de réplica.*
- *Procesamiento distribuido de consultas.*
- *Manejo distribuido de transacciones.*
- *Independencia con respecto al equipo.*
- *Independencia con respecto al sistema operativo.*
- *Independencia con respecto a la red.*
- *Independencia con respecto al DBMS.*

Estas doce reglas no son todas independientes entre sí, ni son por fuerza exhaustivas, ni tienen todas la misma importancia (diferentes usuarios darán diferentes grados de importancia a diferentes reglas en diferentes ambientes). Sin embargo, sí son útiles como fundamento para entender la tecnología distribuida y como marco de referencia para caracterizar la funcionalidad de sistemas distribuidos específicos.

Un último punto introductorio: es importante distinguir los sistemas distribuidos de bases de datos verdaderos, generalizados, de los sistemas que tan solo ofrecen algún tipo de acceso remoto a los datos (llamados a veces sistemas de procesamiento distribuido o sistemas de red). En un " sistema de acceso remoto a los datos ", el usuario podría ser capaz de trabajar con datos de un sitio remoto, o aun con datos de varios sitios remotos al mismo tiempo, pero " se notan las costuras" ; el usuario definitivamente está consciente (en mayor o menor grado) de que los datos son remotos, y debe comportarse de manera acorde. En cambio, en un sistema distribuido verdadero, las costuras son invisibles.

Las doce reglas.

- **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 cae el sitio Y, situación a todas luces indeseable). 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.

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 pro 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 : en primer lugar, ese sitio central podrí ser un cuello de botella; en segundo lugar, 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 añadirse un nuevo sitio o instalar una versión mejorada del DBMS en un sitio ya existente.

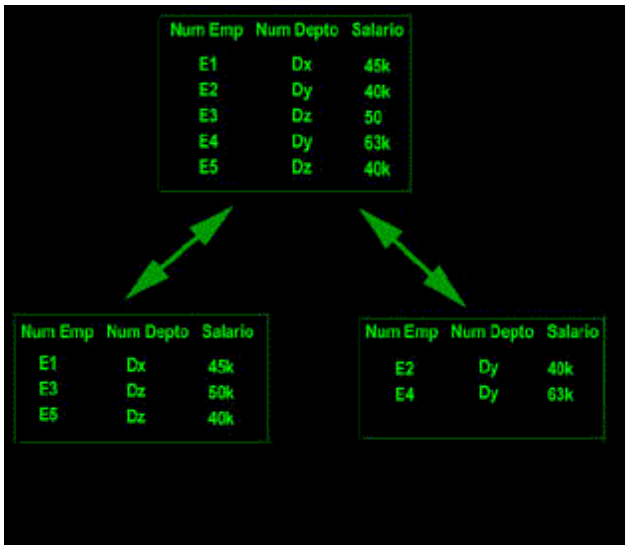
4. Independencia con respecto a la localización.

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 la 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 pues permite modificar la distribución de los datos dentro de la red en respuesta a cambios en los requerimientos de desempeño.

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 sean sólo locales y se reduzca al tráfico en la red. Por ejemplo, la relación empleados EMP podría fragmentarse de manera que los registros de los empleados de Nueva York se almacenen en el sitio de Nueva York, en tanto que los registros de los empleados de Londres se almacenan en el sitio de Londres.

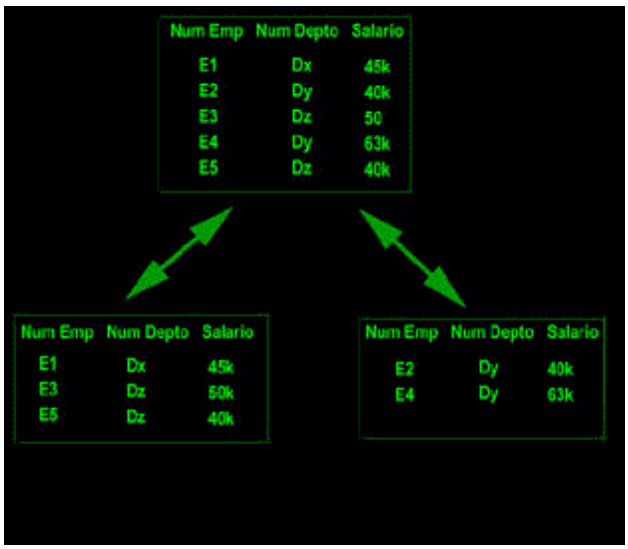
Existen en esencia dos clases de fragmentación, horizontal y vertical, correspondientes a las operaciones relacionales de restricción y proyección; respectivamente. En términos más generales, un fragmento puede ser cualquier subrelación arbitraria que pueda derivarse de la relación original mediante 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 fragmentación vertical, y la unión en casos de fragmentación horizontal).



Ahora llegamos a un punto principal : un sistema que maneja la fragmentación de los datos deberá ofrecer también una independencia con respecto a la fragmentación (llamada también transparencia de fragmentación). La independencia con respecto a la fragmentación (al igual que la independencia con respecto a la independencia con respecto a la localización) es deseable porque simplifica los programas de los usuarios y sus actividades en la terminal.

6. Independencia de réplica.

Un sistema maneja réplica de datos si una relación dada (ó en términos más generales, un fragmento dado en 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 en tanto esté disponible por lo menos una copia, al menos para propósitos de recuperación). La desventaja principal de las réplicas es desde luego que cuando se pone al día un cierto objeto copiado, deben ponerse al día 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 (conocida también como transparencia de réplica); es decir, los usuarios deberán poder comportarse como si sólo existiera una copia de los datos. La independencia de réplica es buena porque simplifica los programas de los usuarios y sus actividades en la 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 esos 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 partes rojas en Londres". Supongamos que el usuario está en la instalación de Nueva York y los datos están en el sitio de Londres. Supongamos también que son n</I< los registros proveedor que satisfacen solicitud. Si sistema es relacional, consulta implicará en esencia dos mensajes : uno transmitir la solicitud Nueva York a Londres, y otro para devolver el conjunto resultante de n registros de Londres a Nueva York. 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 Nueva York a Londres solicitando el siguiente registro, y n de Londres a Nueva York 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), quizá en varios órdenes de magnitud.

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 al red para satisfacer la solicitud, y es crucial encontrar una estrategia suficiente. Por ejemplo, una solicitud de unión de una relación Rx almacenada en el sitio X y una relación Ry almacenada en el sitio Y podría llevarse a cabo trasladando Rx a Y o trasladando Ry a X, o trasladando las dos a un tercer sitio Z .

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 varios sitios (en particular puede implicar a 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 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. La cuestión especifica del control de recuperación; : para asegurar, pues 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.

En cuanto al control de concurrencia, esta función en un ambiente distribuido estará basada con toda seguridad en el bloqueo, como sucede en los sistemas no distribuidos.

9. Independencia con respecto al equipo.

En realidad, no hay mucho que decir acerca de este tema, el título lo dice todo. Las instalaciones de cómputo en el mundo real por lo regular incluyen varias máquinas diferentes −máquinas IBM, DEC, HP, UNISYS, PC etc− 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 conviene 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 un corolario del anterior. Es obvia la conveniencia no sólo de poder ejecutar el mismo DBMS en diferentes equipos, sino también poder ejecutarlo en diferentes sistemas operativos y lograr que una versión MVS y una UNIX y una 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 comunicación distintas.

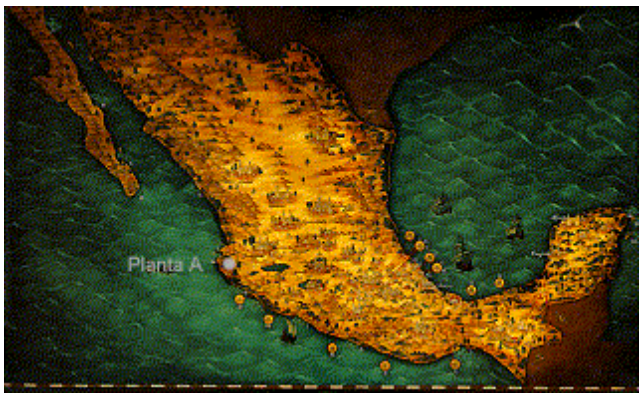
12. Independencia con respecto al DBMS

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.

Ventajas

Existen cuatro ventajas del procesamiento de bases de datos distribuidas. La primera , puede dar como resultado un mejor rendimiento que el que se obtiene por un procesamiento centralizado. Los datos pueden colocarse cerca del punto de su utilización, de forma que el tiempo de comunicación sea más mas corto. Varias computadoras operando en forma simultánea pueden entregar más volumen de procesamiento que una sola computadora.

Segundo , los datos duplicados aumentan su confiabilidad. Cuando falla una computadora, se pueden obtener los datos extraídos de otras computadoras. Los usuarios no dependen de la disponibilidad de una sola fuente para sus datos .



Una tercera ventaja , es que los sistemas distribuidos pueden variar su tamaño de un modo más sencillo. Se pueden agregar computadoras adicionales a la red conforme aumentan el número de usuarios y su carga de procesamiento. A menudo es más fácil y más barato agregar una nueva computadora más pequeña que actualizar una computadora única y centralizada. Después, si la carga de trabajo se reduce, el tamaño de la red también puede reducirse.

Por último , los sistemas distribuidos se puede adecuar de una manera más sencilla a las estructuras de la organización de los usuarios.

Nota:

Los siguientes puntos están basados en una referencia bibliográfica
distinta, y me pareció importante hablar sobre ello, espero y aclare el tema en cuestión.

Utilización compartida de los datos y distribución del control

Si varias localidades diferentes están conectadas entre si, entonces un usuario de una localidad puede acceder a datos disponibles en otra localidad. La ventaja principal de compartir datos por medio de la distribución es que cada localidad pueda controlar hasta cierto punto los datos almacenados localmente.

Fiabilidad y disponibilidad

Si se produce un fallo en una localidad en un sistema distribuido, es posible que las demás localidades puedan seguir trabajando. En particular si los datos se repiten en varias localidades, una transacción o aplicación que requiere un dato especifico puede encontrarlo en más de una localidad. Así el fallo, de una localidad no implica necesariamente la desactivación del sistema.

Agilización del procesamiento de consultas

Si una consulta comprende datos de varias localidades, puede ser posible dividir la consulta en varias subconsultas que se ejecuten en paralelo en distintas localidades. En los casos en que hay repetición de los datos, el sistema puede pasar la consulta a las localidades mas ligeras de carga.

Ejemplo de sistemas.

Para efectos de referencia posterior, mencionaremos brevemente algunas de las realizaciones de sistemas distribuidos más conocidas. En primer término, los prototipos. Entre los sistemas investigados, tres de los más conocidos son :

- SDD−1 creado en la división de investigación de Computer Corporation of America (CCA) a finales de la década de los 1970 y principios de la siguiente.
- R* (pronunciado "R estrella"), versión distribuida del prototipo System R elaborada en IBM Research a principios de la década de 1980 y ;
- INGRES distribuido, versión distribuida del prototipo INGRES, creada también a principios de la década de 1980 en la University of California en Berkeley.

Pasando a productos comerciales, algunos de los más conocidos son :

- INGRES/STAR de Relational Technology, Inc.
- SQL*STAR, de Oracle Corp. y
- DB2 versión 2 Edición 2, de IBM.

Desventajas

Las primeras dos desventajas de las bases de datos distribuidas son las mismas que las dos primeras ventajas.

Primero , el rendimiento puede ser peor para el procesamiento distribuido que para el procesamiento

centralizado. Depende de la naturaleza de la carga de trabajo, la red, el DDBMS y las estrategias utilizadas de concurrencia y de falla, así como las ventajas del acceso local a los datos y de los procesadores múltiples, ya que éstos pueden ser abrumados por las tareas de coordinación y de control requeridas. Tal situación es probable cuando la carga de trabajo necesita un gran número de actualizaciones concurrentes sobre datos duplicados, y que deben estar muy distribuidos.

Segundo , el procesamiento de base de datos distribuida puede resultar menos confiable que el procesamiento centralizado. De nuevo, depende de la confiabilidad de las computadoras de procesamiento, de la red, del DDBMS, de las transacciones y de las tasas de error en la carga de trabajo. Un sistema distribuido puede estar menos disponible que uno centralizado. Estas dos desventajas indican que un procesamiento distribuido no es ninguna panacea. A pesar de que tiene la promesa de un mejor rendimiento y de una mayor confiabilidad, tal promesa no está garantizada.

Una tercera desventaja es su mayor complejidad, a menudo se traduce en altos gastos de construcción y mantenimiento. Ya que existen más componentes de hardware, hay más cantidad de cosas por aprender y más interfaces susceptibles de fallar. El control de concurrencia y recuperación de fallas puede convertirse en algo complicado y difícil de implementar, puede empujar a una mayor carga sobre programadores y personal de operaciones y quizá se requiera de personal más experimentado y más costoso.

El procesamiento de bases de datos distribuido es difícil de controlar. Una computadora centralizada reside en un entorno controlado, con personal de operaciones que supervisa muy de cerca, y las actividades de procesamiento pueden ser vigiladas, aunque a veces con dificultad. En un sistema distribuido, las computadoras de proceso, residen muchas veces en las áreas de trabajo de los usuarios. En ocasiones el acceso físico no está controlado, y los procedimientos operativos son demasiado suaves y efectuados por personas que tienen escasa apreciación o comprensión sobre su importancia. En sistemas centralizados, en caso de un desastre o catástrofe, la recuperación puede ser más difícil de sincronizar.

Nota:

De las desventajas que se mencionan a continuación, están relacionadas con las doce reglas mencionadas anteriormente.

Procesamiento de Consultas

El problema más grande es que las redes de comunicación (las de larga distancia en especial) son lentas. El objetivo es reducir al mínimo el tráfico en la red y esto implica que el proceso mismo de optimización de consultas debe ser distribuido, además del proceso de ejecución de las consultas. Es decir un proceso representativo consistirá en un paso de optimización global, seguido de pasos de optimización local en cada unos de los sitios afectados.

Administración de Catálogo

En un sistema distribuido, el catálogo del sistema incluirá no solo la información usual acerca de las relaciones, índices, usuarios, sino también toda la información de control necesaria para que el sistema pueda ofrecer la independencia deseada con respecto a la localización, la fragmentación y la réplica.

- Centralizado (" no depender de un sitio central")
- Replicas completas (" falta de autonomía, toda la actualización debe ser propagada a cada sitio ")
- Dividido (muy costoso)
- Combinación de 1 y 3 (" no depender de un sitio central ").

Propagación de Actualizaciones

El problema básico con la réplica de datos, es la necesidad de propagar cualquier modificación de un objeto lógico dado a todas las copias almacenadas de ese objeto. Un problema que surge es que algún sitio donde se mantiene una copia del objeto puede NO estar disponible, y fracasaría; la modificación si cualquiera de las copias no esta disponible.

Para tratar este problema se habla de " una copia primaria " y funciona así :

· Una de las copias del objeto se designa como copia primaria, y las otras serán secundarias.

· Las copias primarias de los distintos objetos están en sitios diferentes.

· Las operaciones de actualización se consideran completas después de que se ha modificado la copia primaria.

. El sitio donde se encuentra esa copia se encarga entonces de propagar la actualización a las copias secundarias.

Recuperación

Basado en el protocolo de compromiso de dos fases. El compromiso de dos fases es obligatorio en cualquier ambiente en el cual una sola transacción puede interactuar con varios manejadores de recursos autónomos, pero tiene especial importancia en un sistema distribuido porque los manejadores de recursos en cuestión (o sea los DBMS locales) operan en sitios distintos y por tanto son muy autónomos. En particular, son vulnerables a fallas dependientes. Surgen los siguientes puntos :

- El objetivo de "no dependencia de un sitio central" dicta que la función de coordinador no debe asignarse a un sitio específico de la red, sino que deben realizarla diferentes sitios para diferentes transacciones. Por lo regular se encarga de ella el sitio en el cual se inicia la transacción en cuestión.
- El proceso de compromiso en dos fases requiere una comunicación entre el coordinador y todos los sitios participantes, lo cual implica más mensajes y mayor costo extra.
- Si el sitio Y actúa como participante en un proceso de compromiso en dos fases coordinado por el sitio X, el sitio Y deberá hacer lo ordenado pro el sitio X (compromiso o retroceso, según se aplique), lo cual implica otra pérdida de autonomía local.
- En condiciones ideales, nos gustaría que el proceso de compromiso en dos fases funcionara aun en caso de presentarse fallas de sitios o de la red en cualquier punto. Idealmente, el proceso debería ser capaz de soportar cualquier tipo concebible de falla. Por desgracia es fácil ver que este problema es en esencia imposible de resolver; es decir, no existe un protocolo finito que garantice el compromiso al unísono de una transacción exitosa por parte de todos los agentes, o el retroceso al unísono de una transacción no exitosa en caso de fallas arbitrarias.

Concurrencia

Este concepto tiene que ver con la definición de un agente. El manejo de transacciones tiene dos aspectos principales, el control de recuperación y el control de concurrencia. En un sistema distribuido, una sola transacción puede implicar la ejecución de código en varios sitios (puede implicar actualizaciones en varios sitios), entonces se dice que una transacción esta compuesta por varios agentes, donde un agente es el proceso ejecutado en nombre de una transacción dada en determinado sitio. Y el sistema necesita saber cuando dos agentes son parte de la misma transacción.

Un Panorama de las Bases de Datos Orientadas a Objetos

Como cualquier base de datos programable, una base de datos orientada a objetos (BDOO) da un ambiente para el desarrollo de aplicaciones con un depósito persistente listo para su explotación. Una BDOO almacena y manipula información que puede ser digitalizada (representada) como objetos, proporciona una estructura flexible con acceso ágil, rápido, con gran capacidad de modificación. Además combina las mejores cualidades de los archivos planos, las bases jerárquicas y relaciónales.

Actualmente, el creciente uso de las metodologías de programación orientadas a objetos está promoviendo la aparición de manejadores de BDOO en el mercado. Esto tiene sentido, puesto que la tecnología de objetos proviene del desarrollo de metodologías avanzadas de programación. Más aún, la comunidad internacional está convencida de que los manejadores de BDOO tienen la flexibilidad tanto en la definición del modelo de datos como en el desempeño tan anhelado por muchos desarrolladores de aplicaciones, lo que es imposible encontrar en los modelos jerárquicos de red o relaciónales.

Aspectos de Tecnología

Los objetos pueden estar compuestos por cualquier tipo de información que, eventual mente, puede almacenarse en forma digital; por ejemplo, imágenes barridas, voz, sonido, dibujos, planos arquitectónicos complejos, esquemas electrónicos y diagramas desarrollados por ingenieros, así como los tradicionales tipos de datos alfanuméricos. Comúnmente, las aplicaciones que producen este tipo de objetos complejos, al terminar, los guardan en archivos de datos en distintos formatos. Cuando el programa es reactivado, los objetos, se cargan nuevamente. En estos ambientes, los objetos son accesibles sólo a un usuario en cada momento, no existen mecanismos de seguridad, no hay manera de protegerse ante la eliminación accidental de un objeto complejo. Las BDOO superan todas estas dificultades porque permiten que múltiples usuarios compartan objetos complejos para manipularlos en ambiente seguro y estructurado.

Las bases de datos convencionales fueron diseñadas para manejar tipos de datos alfanuméricos, por ello, difícilmente pueden usar objetos y métodos. Algunos proveedores de bases de datos relacionales han respondido a las tendencias de la tecnología facilitando "front−ends" orientados a objetos, una capa filtrante que traduce entre objetos y la base de datos interna. Sin embargo, este enfoque es limitado porque los objetos deben ser interceptados, desmenuzados en una forma que se almacene en la base de datos relacional, lo que resulta en un proceso difícil. Los objetos deben ser repetidamente ensamblados (para trabajar con ellos) y desarticulados (para guardarlos).

Una base de datos de red o jerárquica puede almacenar objetos complejos, pero esta arquitectura no es flexible, lo cual motiva, el uso del modelo relacional. El problema principal con los modelos en red o jerárquicos es que la estructura es definida rígidamente, cuando la base de datos se crea. Estos sistemas casi no permiten flexibilidad para modificaciones, el sistema debe desactivarse cuando se requiere modificar estructuras de objetos o métodos.

Una base de datos relacional tiene una estructura más flexible, pero no puede manejar tipos de datos complejos. Para sobreponerse a estas limitaciones, algunos proveedores han desarrollado las bases de datos orientadas a objetos, las cuales son diseñadas para manipular los objetos con los conceptos de la programación orientada a objetos, proporcionando un concepto persistente en un ambiente multiusuario seguro.

Existen niveles en los cuales las bases de datos incorporan los conceptos alrededor de la metodología de objetos. La primera clase, puede denominarse BDOO pasivas o "estructuralmente orientadas a objetos", que permiten manejar objetos compuestos. Una base de datos pasiva puede almacenar objetos complejos pero no puede definir comportamientos. Este tipo de bases de datos se utiliza para almacenar objetos de otras aplicaciones. Una BDOO pasiva incluye conceptos como "jerarquía parte de", pero no incluye mecanismos para tipos definidos por el usuario o aspectos que definen comportamientos. Una BDOO es activa u "orientada

a objetos por comportamiento" si permite definir y ejecutar comportamiento de los objetos dentro de la base de datos, incorpora conceptos como la "herencia" y permite el manejo de tipos definidos por el usuario. Si se incorporan todos los aspectos se denomina "plenamente orientada a objetos". En bases de datos activas, es sencillo programar una señal de alerta en un objeto inventario cuando se llega a un nivel mínimo.

Ventajas en BDOOs

Entre las ventajas más ilustrativas de las BDOOs está su flexibilidad, soporte para el manejo de tipos de datos complejos. Por ejemplo, en una base de datos convencional, si una empresa adquiere varios clientes por referencia de clientes servicio, pero la base de datos existente, que mantiene la información de clientes y sus compras, no tiene un campo para registrar quién proporcionó la referencia, de qué manera fue dicho contacto, o si debe compensarse con una comisión, sería necesario reestructurar la base de datos para añadir este tipo de modificaciones. Por el contrario, en una BDOO, el usuario puede añadir una "subclase" de la clase de clientes para manejar las modificaciones que representan los clientes por referencia.

La subclase heredará todos los atributos, características de la definición original, además se especializará en especificar los nuevos campos que se requieren así como los métodos para manipular solamente estos campos. Naturalmente se generan los espacios para almacenar la información adicional de los nuevos campos. Esto presenta la ventaja adicional que una BDOO puede ajustarse a usar siempre el espacio de los campos que son necesarios, eliminando espacio desperdiciado en registros con campos que nunca usan.

La segunda ventaja de una BDOO, es que manipula datos complejos en forma rápida y ágilmente. La estructura de la base de datos está dada por referencias (o apuntadores lógicos) entre objetos. No se requieren búsquedas en tablas o uniones para crear relaciones. Esta capacidad resulta atractiva en aplicaciones de la ingeniería, donde las relaciones entre componentes dependen de factores diversos. Por ejemplo, considérese una aplicación en el diseño de vehículos automotores. El fabricante que quiere determinar una lista de partes necesarias para un auto, para un modelo particular requiere de diferentes decisiones subsecuentes para elaborarla, Si el modelo es automático o estándar, se necesita de un chasis particular como de la caja de velocidades correspondientes.

Escoger un tipo de motor obliga a decidir sobre otras partes requeridas, todo esto hasta el nivel de componentes y piezas individuales. Armar esta lista de componentes resulta más ágil en una BDOO que en una base de datos relacional. En un modelo relacional las tablas deben ser barridas, buscadas cada vez que se indica una condición, resultando, posiblemente, en miles de direccionamientos.

Posibles Problemas

Al considerar la adopción de la tecnología orientada a objetos, la inmadurez del mercado de BDOO constituye una posible fuente de problemas por lo que debe analizarse con detalle la presencia en el mercado del proveedor para adoptar su producto en una línea de producción sustantiva. Por eso, en este artículo proponemos que se explore esta tecnología en un proyecto piloto.

El segundo problema es la falta de estándares en la industria orientada a objetos. Sin embargo, el "Object Managenent Group" (OMG), es una organización Internacional de proveedores de sistemas de información y usuarios dedicada a promover estándares para el desarrollo de aplicaciones y sistemas orientados a objetos en ambientes de cómputo en red.

La implantación de una nueva tecnología requiere que los usuarios iniciales acepten cierto riesgo. Aquellos que esperan resultados a corto plazo y con un costo reducido quedarán desilusionados. Sin embargo, para aquellos usuarios que planean a un futuro intermedio con una visión tecnológica de avanzada, el uso de tecnología de avanzada, el uso de tecnología orientada a objetos, paulatinamente compensará todos los riesgos.

La tecnología de bases de datos orientadas a objetos está en su infancia, sin embargo, establece amplios signos de madurez. Ante la disyuntiva de tomar una decisión estratégica, recalcamos que las empresas e industrias que desean conformar un liderazgo tecnológico están en la posibilidad de explorar los productos comercialmente disponibles o los prototipos de los centros de investigación para iniciar la experimentación y desarrollo de proyectos pilotos.

La asociación de proyectos piloto con instituciones de investigación, permitirá establecer el vínculo entre tecnología, estrategia empresarial y mercado. Las empresas que inicien ahora la exploración de la BDOO podrán apropiarse de esta tecnología y consolidar una ventaja competitiva determinante cuando dominen e incorporen las BDOO en sus procesos productivos.

FOX PRO

Es un sistema que maneja bases de datos.

Fox Pro te permite:

1) Crear una tabla: Para introducir datos y crear tu propia bases de datos.

Las tablas se crean en un archivo al cual tú le das el nombre y una extensión .dbf.

Nota: casi todos los paquetes de bases de datos manejan a sus tablas con la extensión nombre.dbf. como Fox Pro. Access. File Maker, etc.

Al definir una tabla tu puedes elegir entre varios tipos de datos:

Character (carácter)

Numeric (numérico: incluye decim)

Date (fecha)

Memo (resumen de texto)

Gen (general: para imágenes)

Una tabla está formada por campos y registros (fields and records)

Un campo es en sí una columna que agrupa datos similares de varios registros, son los grandes grupos de la base de datos..

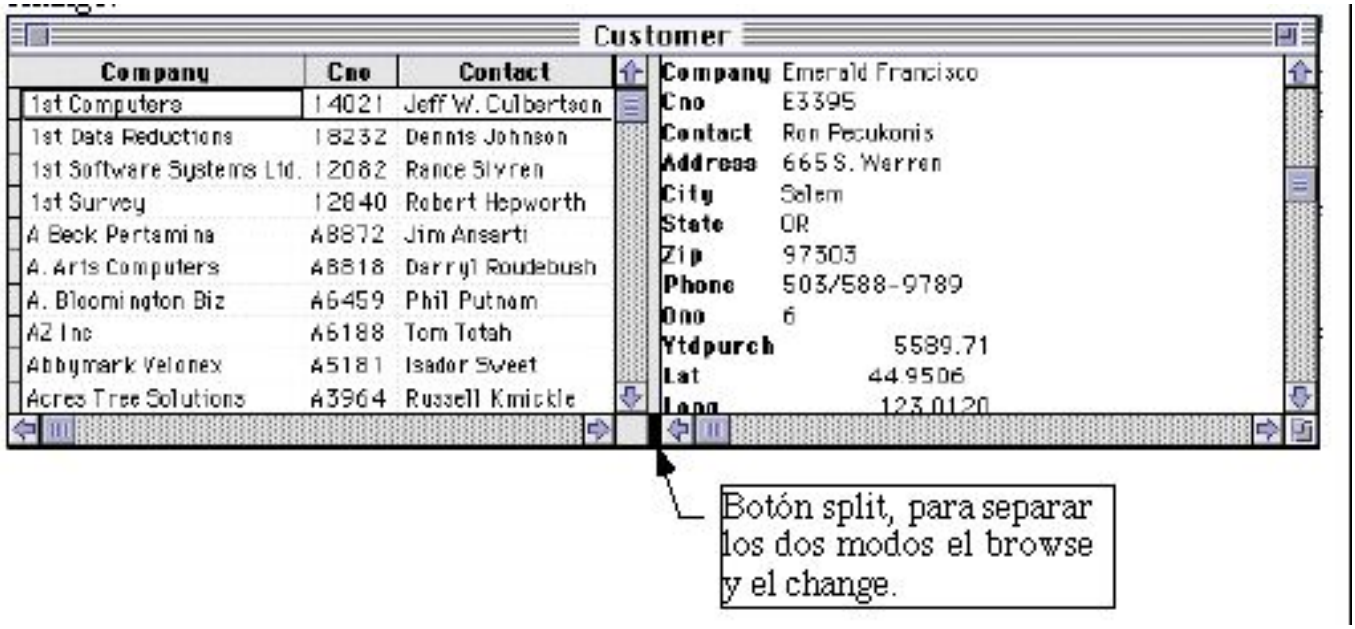
Un registro es una serie de información que detalla a una persona o compañía con sus datos de teléfono, dirección, sus ventas, zona de trabajo, etc. Es decir agrupa los campos para este registro.

Los campos Mem y Gen abren ventanas especiales para colocar la información y/o imágenes respectivamente, dando doble click en ellos. En el memo la puedes copiar y pegar de un procesador de texto, y la imagen desde cualquier procesador de imágenes o del propio Internet.

2) Cambiar una tabla: Uno puede cambiar la estructura de una tabla en sus campos o modificar un registro nuevo.

Esto se realiza en el menú Record opción append para adicionar un nuevo registro o en menú Record opción change para modificar un registro. En el menú database opción setup oprimes el botón modify para modificar la estructura de la tabla ya sea que agregues un campo nuevo o modifiques las propiedades de una que ya existe.

Una base de datos puede observarse en dos modos el browse y el change respectivamente:



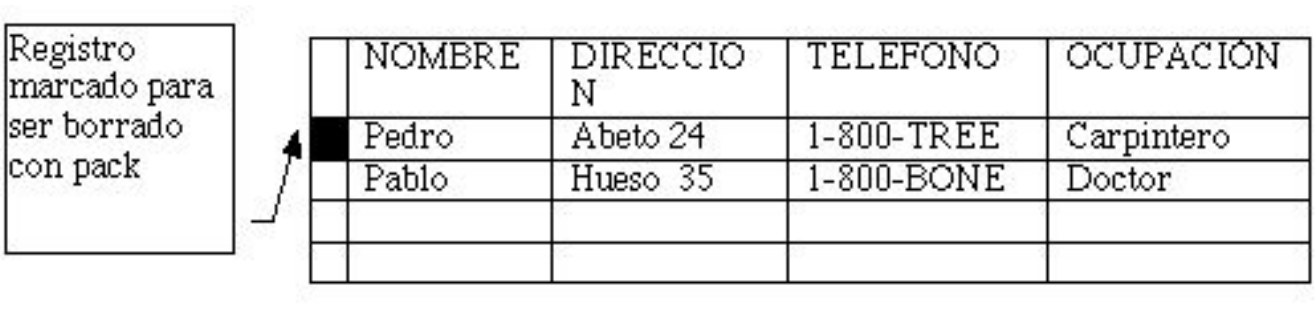
3) Adicionar Registros de una base existente a otra base en la que estas trabajando.

La condición aquí es que ambas bases tengan campos en común. La base de datos abierta es la que recibe los datos y la abierta al salvarla se queda con los datos.

Esto se hace en el menú Database opción Append From y seleccionas a la base de datos que vas a adicionar y se te pregunta mediante que campo las unirás (estos campos son el mismo en ambas, no importa que no tengan la misma información pero las características de ambos deberá ser igual.)

Nota: no importa que los datos provengan de otro paquete (es decir, puede ser de D−base u otra).

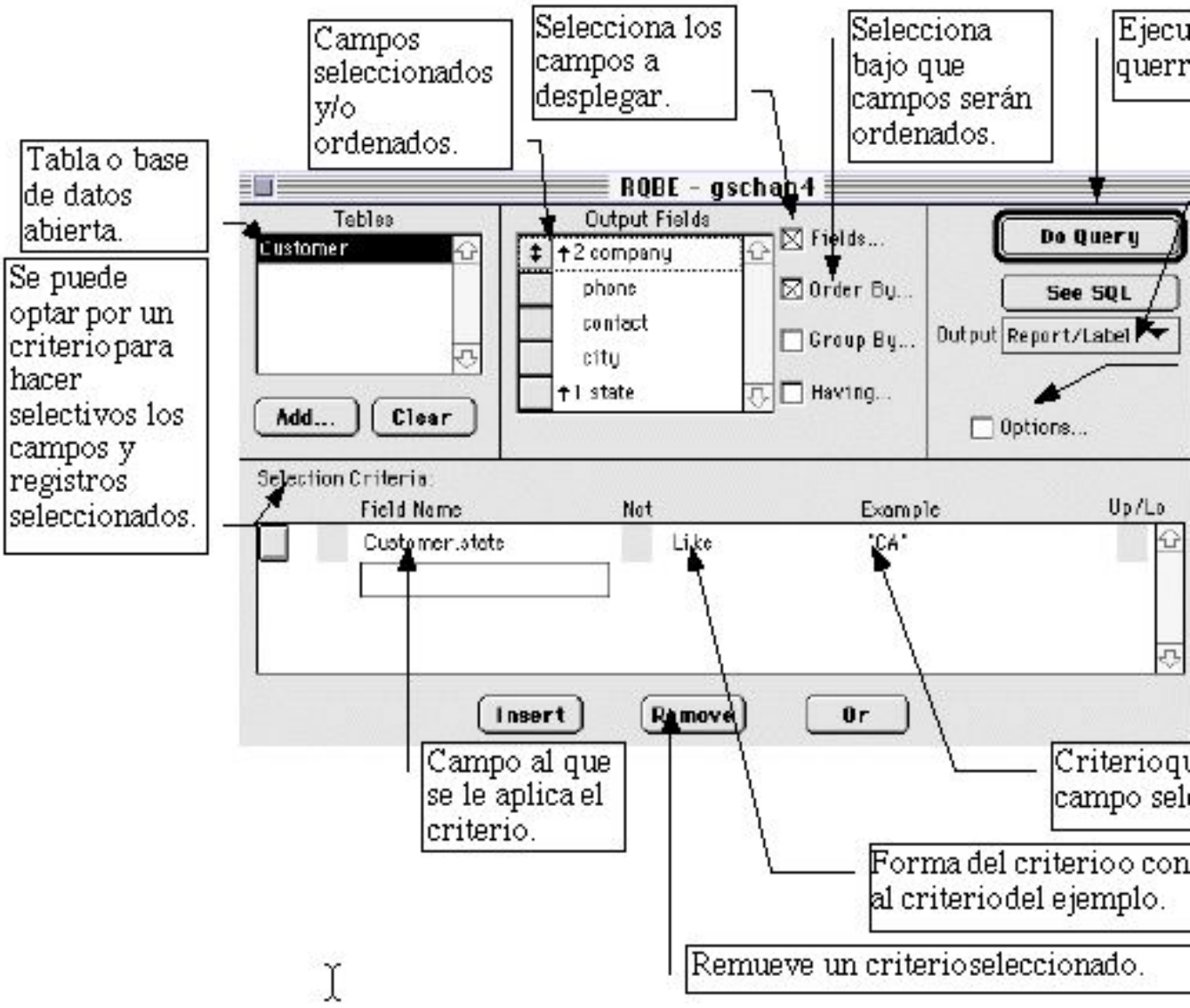
4) Borrar Registros: Hay dos formas de hacerlo; la primera es por medio de mouse, teniendo la ventana en modo Browse de los datos, se hace click con el ratón en un cuadro a la izquierda, de forma que quede negro.



O te para en el registro a borrar luego del menú Record opción Delete aparece una ventana y das clic en scope, se selecciona el número del registro o pisas registro y pones el número y se da ok. al botón Delete.

Cualquiera de las dos formas marca al registro y para borrarlo definitivamente por cualquiera de los dos caminos, en el menú Database opción Pack; y se pregunta si se desea el Pack (empacado) y le das yes.

5) Crear Query; búsquedas o desplegados de información específica para ser vistos en Pantalla (Browse) o en reportes (page preview/printer) (página previa/impresora).



Botones:

- Do Query: genera el Query y depende de lo elegido.
- Add: adiciona otra base de datos, pero la condición es que tenga un campo en común; y sirve para Browse o Reportes.
- Clear: quita base de datos seleccionados.
- Remove: remueve un criterio con todo.

6) Generar un reporte (Report: extension.frx).

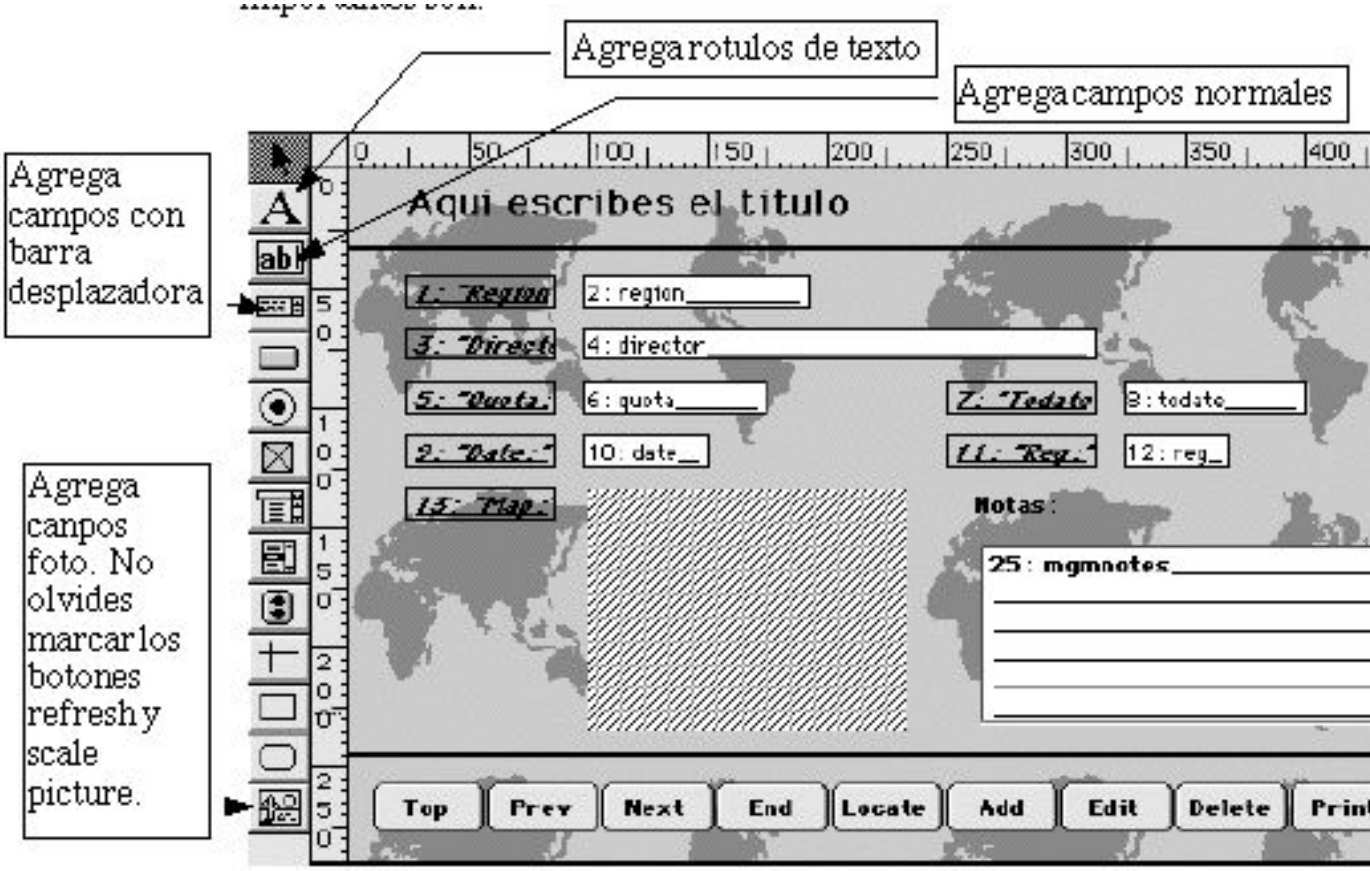
La ventana RQBE genera el reporte. Una vez generado se abre el reporte con menú Fide opción open en type se elige Report y se da open al archivo nombre.frx y se edita y si faltan campos se agregan se le pone color, o una papel tapiz y se salva. Una vez salvado el archivo .frx se corre del menú Run opción Report y se busca a la plantilla del reporte que acabas de salvar y se te preguntara en modo preview o printer y dependiendo de lo que deseas lo mandas.

La ventana Reporte trabaja conjuntamente con la RQBE (Query) y está formada por:

- Page Header: aquí todo lo que se escriba aparecerá en todas las hojas.
- Detail: es donde se colocan los campos y los títulos de los campos, por lo general los campos están encerrados en un cuadro.
- Page Footer: Todo lo que se coloque aquí aparece en todas las hojas.

7) Para la generación de Screen (nombre.scx plantilla y nombre.spr generada) se realiza mediante el wizard, donde se te va proporcionando las preguntas de como deseas el screen, al final (el quinto paso) se te pide el titulo dentro del screen y debes seleccionar el radio botón Modify.....

para que aparezca la plantilla. En ella deberás editar los cambios pertinentes y si faltara algún campo lo debes adicionar con los botones de las herramientas, los mas importantes son:



El papel tapiz se agrega del menú Screen −Layout botón color, botón wallpaper y seleccionas la imagen en formato pict.

En el menú Object encontrarás las opciones para color de pluma, opciones de justificar, tipos de letra y otras opciones para los textos.

Una vez que terminaste de editar, salva tus cambios, ve al menú Program −Generate boton Generate y generarás tu screen (nombre.spr) que posteriormente correras del menú Run− Screen.

8) Por último debes recordar las extensiones de cada archivo:

table	nombre.dbf
query	nombre.qpr
plantilla screen	nombre.scx
plantilla reporte	nombre.frx
screen generado	nombre.spr

EL MODELO RELACIONAL

Al utilizar una base de datos se está tratando de modelar los datos y sus conexiones en un problema del mundo real. Para definir el modelo relacional se inicia con una definición de lo que es un modelo de datos en general.

Un modelo de datos es un sistema formal y abstracto que permite describir los datos de acuerdo con reglas y

convenios predefinidos. Es formal pues los objetos del sistema se manipulan siguiendo reglas perfectamente definidas y utilizando exclusivamente los operadores definidos en el sistema, independientemente de lo que estos objetos y operadores puedan significar [Rivero].

Según el mismo autor, un modelo de datos tiene tres componentes que son:

Componentes:

- Estructuras de datos: es la colección de objetos abstractos formados por los datos.
- Operadores entre las estructuras: el conjunto de operadores con reglas bien definidas que permiten manipular a dichas estructuras.
- Definiciones de integridad: es una colección de conceptos y reglas.
- Que permite expresar que valores de datos pueden aparecer válidamente en el modelo.

En el manejo de bases de datos hay tres modelos de datos principales que son el jerárquico, permite modelar los datos en base a una jerarquización; el de red, donde los datos forman retículas; y el relacional basado en el concepto matemático de relación. Como modelo de datos el modelo relacional tiene las siguientes componentes:

- Estructuras de datos: son los conceptos de relación, entidades, atributos y dominios.
- Operadores: sus operadores incluyen los de actualización y la llamada álgebra relacional.
- Definiciones de integridad: está dada por el concepto de llave, posibilidades de valores nulos y dos reglas de integridad.

ESTRUCTURAS DE DATOS DEL MODELO RELACIONAL

En el modelo relacional las estructuras de datos son los conceptos de relación, dominio, atributo y entidad.

Relación: denota una colección ó conexión entre objetos que tienen los mismos tipos de características o atributos.

Entidad: es un elemento de datos con un conjunto finito de atributos. También se le llama eneada por consistir de n valores, uno para cada atributo.

Atributo o característica, cada atributo tiene un dominio asociado.

Dominio es el conjunto de valores que puede tomar un atributo.

Las relaciones se representan por tablas donde las columnas son los atributos o características. En los renglones se almacenan los elementos de datos con sus valores para cada atributo. En el modelo relacional no se consideran ordenados los renglones. Una representación de una relación es indicar su nombre y entre llaves el conjunto de atributos. A esta representación también se le llama esquema de la relación.

Ejemplo:

Se tiene una relación llamada Alumnos cuyos atributos son:

Alumnos ={No.Cta,nombre−al,promedio,carrera}

También se tienen las relaciones cuyos esquemas son:

Materias ={Clave−mat,nombre−mat,grupo,carrera}

Evaluaciones ={No.Cta,Clave−mat,calificación}

Cada atributo tiene un dominio asociado. Los dominios son los conjuntos de los valores posibles.

Ejemplo:

Los dominios de los atributos de la relación Alumnos son:

Dominios:

- DNo.Cta.={C1,C2,C3,C4,C5}
- Dnombre−al={Nombres de los alumnos}
- Dpromedio={0..10}
- Dcarrera={física,matemáticas,biología}

DEFINICIONES DE INTEGRIDAD EN EL MODELO RELACIONAL

Los conceptos de definiciones de integridad para el modelo relacional son llave primaria y foránea, los valores nulos y dos reglas de integridad que se enuncian a continuación.

Dentro de los atributos debe haber uno o varios que sirvan para distinguir cada entidad en la relación. Es a lo que se llama llave primaria.

Ejemplo:

En la relación Alumnos anterior la llave primaria es

No.Cta.; en la relación Materias la llave es Clave−mat

y en Evaluaciones es la unión de No.Cta y Clave−mat.

Llave Foránea: hace referencia a una llave primaria en

otra relación. Una relación puede tener una o varias

llaves foráneas.

Ejemplo:

Los atributos No.Cta y Clave−mat son llaves foráneas

en la relación Evaluaciones.

Una relación puede tener varias llaves. La que se elige para

identificar a una relación se le llama llave primaria. En el modelo

relacional es el concepto de llave la única forma de encontrar una

entidad. Un valor nulo denotado por ``?", proporciona la posibilidad de manejar situaciones como las siguientes:

1.− Se crea una eneada y no se conocen los valores de los atributos.

2.− Se agrega un atributo a una relación ya existente.

3.− Se usan para no introducir valores numéricos al hacer cálculos.

A fin de mantener la integridad a lo largo del tiempo en una base de datos relacional, se debe cumplir con algunas restricciones en cuanto a los valores de las llaves primarias.

Integridad de Relaciones : Ningún componente de la llave primaria puede tener valores nulos.

Integridad Referencial: Si se tiene una relación q con una llave primaria A de dominio D, y r otra relación con atributo A no llave. Entonces cualquier valor del campo de A en r debe ser:

i) nulo o,

ii) el valor de una llave primaria de la otra relación q donde se tiene la llave primaria sobre D.

OPERADORES DEL MODELO RELACIONAL

Los operadores del modelo relacional son de dos tipos: operadores de actualización de entidades y operadores del álgebra relacional.

Operadores de actualización

Para actualizar los valores de los atributos en las entidades se pueden efectuar las operaciones de agregar, borrar o modificar.

El manejo de llaves foráneas hace necesario establecer reglas que determinan como manejar las operaciones de actualización de relaciones para no introducir inconsistencias, a continuación se indican dichas reglas :

Reglas:

- Reglas para agregar. Al insertar una entidad en una relación, el valor de un atributo que es llave foránea puede ser nulo, o algún valor del dominio de la llave primaria.

- Reglas para borrar. Si se va a borrar una entidad en una relación r1 con cierta llave primaria y otra relación r2 tiene ese campo como llave foránea, hay 3 casos:

Casos de Borrado:

- Borrado restringido. No se puede borrar una entidad en r1 que tenga entidades en r2 con el mismo valor como llave foránea.

- Borrado encascada. Al borrar una entidad en r1 se borrarán todas las entidades en r2 con ese valor.

- Borrado con nulificación. Al borrar la entidad en r1,a todas las entidades con igual valor en r2 se les pone el valor nulo.

Regla para Modificar.

i) Modificación en cascada : Al modificar una llave primaria en r1 se le cambian los valores correspondientes

en r2.

ii)Modificación con nulificación : Al cambiar los valores de la llave primaria en r1 a los correspondientes en r2 se les pone el valor nulo.

Ejemplo :

Se tienen las relaciones:

Alumnos	No.Cta.	Nombre−al	Promedio	Carrera
	C1	Suárez	8	física
	C2	López	9	matemáticas
	C3	Aguirre	7	física
	C4	Cortés	10	biología
	C5	Aguilar	9	biología

Materias	Clave−Mat	Nombre−Mat	Grupo	Carrera
	M1	Análisis	101	matemáticas
	M2	Mecánica	102	física
	M3	Zoología	100	biología
	M4	Botánica	103	biología
	M5	Topología	104	matemáticas
	M6	Óptica	105	física
	M7	Cálculo	106	matemáticas

Evaluaciones	No.Cta.	Clave−Mat	Calif.
	C1	M2	10
	C1	M6	6
	C2	M1	9
	C2	M5	9
	C3	M2	5
	C3	M6	9
	C4	M3	10
	C5	M3	10
	C5	M6	8

PRINCIPIOS MATEMATICOS DE LAS BASES DE DATOS RELACIONALES

Se presenta la definición de un modelo de datos en general, así como sus componentes principales. Se enuncian los componentes del modelo de datos relacional y como se aplican en las bases de datos relaciónales.

INTRODUCCIÓN

Las bases de datos relacionales surgen a partir de un artículo de Codd en 1970. Desde entonces es posible encontrarlas en todo centro de cómputo. Muchas aplicaciones y usuarios de bases de datos tienen a su alcance software para el manejo de bases de datos relacionales de muchas marcas comerciales. Los hay para computadoras personales, medianas y grandes.

Pocas veces se reflexiona en cual es el motivo de su éxito y por qué se ha popularizado tanto. Una de las causas principales es el modelo de datos matemático que lo respalda.

Progress

Una base de datos relacional que aprovecha la potencia del concepto Cliente/Servidor

En los próximos apartados detallamos las características principales:

Cliente / Servidor	Multi plataforma
Multi base de datos	Seguridad
Copias increméntales	

Cliente / Servidor : Desde hace muchos años PROGRESS dispone de esta característica en su base de datos, por lo que hoy en día sus usuarios pueden beneficiarse de la enorme experiencia que tiene la compañía en esta filosofía de trabajo. La filosofía cliente/servidor, consiste en repartir la carga del trabajo entre el servidor y las estaciones de forma que los procesos queden mucho más compensados y el tiempo de respuesta cuando existen gran número de usuarios concurrentes, baja considerablemente. Con la aparición de las estaciones de trabajo PC, máquinas con una magnífica relación rendimiento/precio, este tipo de arquitectura adquiere cada vez mayor sentido, siendo ésta una filosofía que en el mundo de la informática se considera indudablemente de futuro.
Multiplataforma: PROGRESS permite trabajar en un gran número de plataformas tanto en el cliente como en el servidor. SIE ha desarrollado sus aplicaciones intentando aprovechar toda la potencia de WINDOWS como cliente y en la parte del servidor el usuario podrá elegir entre el amplio abanico de posibilidades que le brinda PROGRESS. Entre otras, PROGRESS trabaja sobre las siguientes plataformas y sistemas operativos: PLATAFORMAS SIST. OPERATIVOS Alpha Unix SCO VAX AIX HP 3000 Unixware HP 9000 Windows NT

AS 400 NOVELL PC   SUN   Esta característica unida a la política de reconocimiento de inversión de PROGRESS, garantiza a sus usuarios la inversión que hoy se realice, ya que si en un futuro, por crecimiento, por aumentar las exigencias informáticas o por cualquier otra causa, se desea cambiar la plataforma o el sistema operativo, el software y los datos de la compañía pueden pasar al nuevo entorno sin perder ninguna de sus prestaciones.
Multi base de datos: Cualquier producto desarrollado en PROGRESS puede almacenar la información además de la base de datos nativa que PROGRESS incorpora en otras como DB/2−400, ORACLE, etc... y vía ODBC a cualquier otra que admita este protocolo. Para ello sólo hay que recompilar la aplicación en el nuevo entorno.
Seguridad: PROGRESS gracias a un seguro sistema de transacciones, puede garantizar a sus usuarios la integridad de su información. Aquellas operaciones que afectan a varias entidades, y que pueden quedar interrumpidas por cualquier fallo del sistema, PROGRESS garantiza que o se realizan al completo o se restauran a la situación que se encontraban antes de empezar la transacción. Así pues no puede suceder como en otros sistemas que si a medio generar una factura se produce un corte en el suministro eléctrico, no se almacenen todas sus líneas, o sólo una parte de los apuntes o que todos los movimientos de stock no queden reflejados.
Copias increméntales: Ante cualquier problema que pudiera causar la pérdida de una base de datos, PROGRESS es capaz de reconstruir hasta el último movimiento completo realizado por los usuarios a partir de la última copia y reconstruyendo los movimientos gracias a un fichero en que va almacenando todas las operaciones de que se realizan.

Una Base de Datos es, esencialmente, un conjunto de datos ordenados en filas y columnas que se cargan y se ven en un programa de Planilla de Cálculos. Cada dato esta compuesto por un grupo de informaciones llamadas campos; entonces podemos decir que una base de datos es un banco de informaciones ordenadas en filas y columnas. Los campos varían de acuerdo al rubro y el universo seleccionados.

Rubro: El rubro es la persona jurídica (Ej. una empresa) o persona física (Ej.. un individuo) de la que necesitamos saber un numero de datos. El rubro es muy variable ya que lo especifica el cliente de acuerdo a sus requerimientos. Están segmentados en distintas áreas: Económicos o por ingresos, Profesionales, Empresas e Individuos en general, Titulares de tarjetas de crédito, etc.

Universo: Es el área geográfica en la que Ud. busca el rubro seleccionado. Por ejemplo, si necesita los datos de empresas productoras de cereal de la Republica Argentina, se estará refiriendo como universo al total país de la Argentina, o sea a todas las empresas productoras de cereal de dicho país. Pero también podrá centralizarse solo en Capital Federal, Gran Buenos Aires, Interior del País, o mas específicamente: Podrá referirse a determinadas localidades de la Capital Federal, del Gran Buenos Aires o de las Provincias del Interior del País. Para centralizar aun mas la búsqueda, Ud. tiene la posibilidad de seleccionar los diferentes barrios, zonas, áreas, radios o calles de Capital o Gran Buenos Aires.

Campos:

Tomando como ejemplo una empresa tipo P y ME, tenemos los siguientes campos:

- Razón Social.
- Dirección.
- Código Postal.
- Numero de Teléfono.
- Numero de Fax.
- Facturación.
- Cantidad de empleados.
- Contacto. (nombres y apellidos de individuos de mayor jerarquía con sus cargos principales pertenecientes a la empresa. − Ej.: Propietario, Gerente General, Gerente de Marketing, Gerente de Sistemas, etc.)

Si lo que se busca son individuos los campos serán los siguientes (siempre dependiendo del tipo de base)

- Nombre.
- Apellido.
- Dirección.
- Código Postal.
- Numero de Teléfono.

Potencial: Es la cantidad de datos que existen del rubro y universo seleccionados.

¿Base de datos Enlatada o a Medida?

Las Enlatadas, son bases cuyos datos pertenecen a los rubros mas utilizados por la mayoría de nuestros clientes. Pero esto no quita que Ud. no necesite ofrecer su producto o servicio a empresas y/o individuos fuera del contenido de las bases enlatadas. Todo lo contrario: Puede designar rubros y/o universos mas específicos, partiendo de un modelo de base enlatada. Pero además puede designar rubros, universos, campos y potenciales a la medida de sus requerimientos. Esto se denomina Base de Datos a Medida.

Ejemplo Práctico:

Rubro: Empresas de Informática y Resellers;

Universo: Total País y Exterior.

Campos: Razón Social, dirección, localidad, código postal, teléfono, fax y contacto.

Potencial: 3000 datos.

Esta base seguirá siendo Enlatada si Ud. nos pide variantes como:

Rubro: Resellers.

Universo: Total País.

O por ejemplo:

Rubro: Empresas de Informática.

Universo: Exterior.

En cambio, esta base será hecha a medida si Ud. nos pide:

Rubro: Empresas de Informática que vendan Computadoras.

El conjunto de todo forman una base de datos. Dentro de un volumen pueden existir varias bases de datos que se pueden interrelacionar.

CONSULTA Es una selección de información.

Una Base de Datos puede tener mínimo una tabla, puede tener un nombre la base y la tabla, las consultas y los informes. etc.

El nombre de la Base de Datos hace la diferencia dentro, los nombres de las consultas e informes. Internet es el mejor ejemplo de base de datos.

Una base de datos no es mas que un mantenedor de registros, es decir, un sistema cuyo propósito general es registrar y mantener información la que puede estar relacionada con cualquier cosa u objeto que sea significativo para la organización.

En un sistema de Base de Datos se distinguen 4 componentes:

- Datos
- Software
- Hardware
- Usuarios

1.− *Datos*: Los datos almacenados se dividen en una o más Bases de Datos. Por lo tanto, una Base de Datos es un recipiente de Datos almacenados y en general es Integrada como Compartida.

Integrada : Se entiende que la base de dato puede considerarse como una unificación de varios archivos de datos independientes de donde se elimina parcial o totalmente cualquier redundancia entre los mismos.

* se puede revisar que información existe.

*Redundancia = El mismo dato puede tener más de una base de datos.

* cuando hay archivos con diferentes llaves o estructuras de datos distintas va hacer difícil poder accesarlos porque corren bajo diferente ambiente.

Lo más recomendable es tener una base de datos única. Normalizando la aplicación en grandes archivos (normalización de datos)*(las dejamos depuradas).

* Eliminando la redundancia de datos en los archivos.

* optimización

* eliminar los datos que sean claves.

Compartida: Partes individuales de las Bases de Datos pueden compartirse entre varios usuarios distintos en el sentido de que cada uno de ellos puede tener acceso a la misma parte de la Base de datos y usarla con

propósitos distintos. Tal compartimiento es en verdad consecuencia del hecho de que la base de Datos es integrada. (privilegio se puede dar por programa a de administración de red).

2.− *Software*: Entre las bases de datos físicas en sí, es decir, el almacenamiento real de datos y los usuarios existe un nivel de software que a menudo recibe el nombre de DBMS (Software de interfaz entre las bases de datos y usuario). Su objetivo es manejar todas las solicitudes formuladas por los usuarios y/o programas de acuerdo a una función general del DBMS, otro objetivo es proteger a los usuarios contra los detalles a nivel de hardware, es decir, el DBMS ofrece una visión a los usuarios que esta por encima del hardware y apoya las operaciones de este.

Ejemplo: Obtener el registro del empleado 205

SELECT * NOMBRE

FROM Xi (TABLA)

WHERE COD_EMPL = 205

EL DBMS VA A LA TABLA DE LA BASE DE DATOS LO BUSCA Y LO MUESTRA.

3.− *HADWARE*: Compuesto por volumen de disco, tambores, etc., donde esta residente la Base de Datos. Junto con los dispositivos asociados como unidades de control, canales, etc., ya que se asume que la Base de Datos es demasiado grande para localizarla en memoria.

4.− *Usuarios*: Existen 3 tipos de usuarios.

- Usuario Final: Usa la Base de Datos realizando consultas, manutención etc., a través de un lenguaje de consulta, o a través de un programa mediante el lenguaje de consulta el usuario queda libre para poder hacer cualquier operación y mediante un programa el usuario queda restringido a lo que en este se estableció.

- Segundo Usuario: Programador de aplicaciones, desarrolla los sistemas necesarios para permitir la posibilidad de comunicación o extensión de información desde la Base de Datos.

- Administrador de la Base de Datos: Dentro de sus funciones podemos mencionar:

− Mantener en forma optima y eficiente la base de datos controlando procedimientos, instalaciones, procesos, etc. Realizando funciones de auditoria, manejando la seguridad de la Base de Datos. además de crear usuarios y accesos permitidos.

VENTAJAS:

- Se evita la inconsistencia de la información.
- Se evita la redundancia de la información, manteniendo integrada las Bases de Datos y bien definidas las tablas.
- Evita la duplicidad de datos.
- Los datos pueden ser compartidos por varias aplicaciones ovarios usuarios.
- Se mantienen niveles de seguridad en los datos.

DESVENTAJAS:

- Al perder o dañarse la base de datos se hace irrecuperable las tablas, informes, consultas, etc., que en

ella existen.
- Al tener una Base de Datos con demasiada información su acceso es lento ya que varios usuarios pueden estar haciendo uso de esta.

Datos persistentes: Son aquellos que mantienen la Base de Datos existiendo 2 tipos denominados Datos de Entrada y Datos de Salida .

Datos de Entrada: Se refiere específicamente a la información que entra al sistema por primera vez generalmente a través del teclado (o touch, screen), está información va ha alterar los datos persistentes por ende esta información no forma parte de la Base de Datos (Son parámetros para extraer información).

Datos de Salida: Es la información que se obtiene a través de la manipulación, puede ser conocida a través de consulta de usuario, programa o sistema.

ENTIDADES E INTERRELACIONES: Se define por entidad a cualquier objeto distinguible que pueda representarse en una Base de Datos.

RELACION: Correspondencia entre entidades en base a un atributo generalmente estas son bi−direccionales y se clasifican en 1a 1, 1 a N, N a 1, N a N.

ATRIBUTO: Es el calificador de la entidad.

registro = entidad

campo = atributo

** Las tablas tienen que tener los mismos atributos para poder relacionarse.

ENTIDADES Y CONJUNTO DE ENTIDADES

Una entidad es un objeto que existe y es distinguible de otros objetos.

ejemplo: Nombre ===> Tsantis ====> registro ====> entidad

Rut ===> 9.233.283−7 ====> campo ====> atributo

*Es una entidad ya que identifica únicamente a una persona en el universo.

*una persona es única en una base de datos.

*Un conjunto de entidades forman una tabla.

ENTIDADES QUE PUEDEN INTERRELACIONARSE

Un conjunto de entidades es un conjunto de entidades del mismo tipo. El conjunto de todas las personas que tienen una cuenta en un banco puede definirse como el conjunto de entidades clientes. También podríamos definir el conjunto de todas las cuentas de un banco como el conjunto de entidades de cuentas.

Los conjuntos de entidades no necesitan ser disjuntos Por ejemplo Es posible definir el conjunto de entidades de todos los clientes de banco, y el conjunto de entidades todos los empleados de un banco. Una persona puede ser una entidad empleado, una entidad cliente, ambas o ninguna de las dos. Ejemplo:

Dirección ====> Cliente

Posibles atributos del Conjunto de entidades Cta. Cliente son:

SALDO ====> CTA.CTE.

Nº ====> CTA.CTE.

MOVIMIENTO ====> CTA.CTE.

Para cada atributo hay un conjunto de valores permitidos llamados DOMINIO de ese ATRIBUTO.

Ejemplo El Dominio del ATRIBUTO Nombre >>Cliente podría ser el conjunto de todas las cadenas de texto de una determinada longitud, El Dominio del Atributo Saldo >>>Cta.Cte. Podría ser el conjunto de todos los enteros positivos. (el dominio se refleja en el atributo) Formalmente un atributo es una función que asigna un conjunto de entidades a un dominio. A sí cada entidad se describe por medio de un conjunto de pares.

DOMINIO: (Atributo, valor del dato) Un par para cada atributo del conjunto de entidades. Una entidad cliente determinada se describe por medio del conjunto. ejemplo: (calle, Alameda Nº 271) (Ciudad, Santiago) (Rut,101224−6) (Nombre, Carlos), lo cual significa que la Entidad describe a una persona llamada Carlos con rut 1012214−6, que vive en Santiago Alameda Nº 271.

INDEPENDENCIA DE LOS DATOS: Se habla Datos independientes cuando en una Base de Datos se puede alterar la estructura de almacenamiento o la forma de acceder los datos sin afectar gravemente la aplicación, se dice que la independencia de los datos es parte de los objetivos de la base de datos, y se puede definir como la inmunidad de las aplicaciones ante los cambios de estructura de datos (almacenamiento) y a la técnica de acceso, cabe señalar por ende que un sistema de Base de Datos no es recomendable tener aplicaciones dependiente de los datos.

Cada aplicación requiere de una lista diferente de los mismos datos. Es decir, ver como se puede utilizar la información.

ADMINISTRADOR DE BASE DE DATOS (DBA)

El DBA (Administrador de Base de Datos) debe tener libertad para modificar la estructura de almacenamiento o la técnica de acceso para adaptarse a cambios de requerimientos sin tener que modificar las aplicaciones ya existentes. Si las aplicaciones dependen de los datos tales cambios requerirán modificaciones en las aplicaciones ya existentes.

Ejemplos: De los tipos de modificaciones que podría realizar el DBA los cuales deberían ser inmune a las aplicaciones.

- Representación de los Datos Numéricos: Esta asociado a su forma de representación interna, es decir, como se van a almacenar. Ejemplo: Formato Binario, empaquetado, etc.
- Representación de datos de caracteres: esto podría ser almacenados en formato ASCII, ABCD, etc.
- Unidades para Datos Numéricos: Esta asociado específicamente al cambio de unidades numéricas ejemplo: Pulgadas, Metros, Hectáreas, Cuadras, etc).
- Codificación de los Datos: Esta relacionada a la estandarización de los datos propiamente tal, específicamente normalización. Ej. 1 amarrillo 2. soltero.

ARQUITECTURA DE UN SISTEMA DE BASE DE DATOS

La arquitectura de un sistema de base de datos se divide en 3 niveles comunes, nivel interno, conceptual y externo.

Nivel Interno: Es el más cercano al almacenamiento físico, es decir, es el que se ocupa de la forma como se almacenan físicamente los datos.

Nivel Externo: Es el mas cercano a los usuarios, es decir, es el que se ocupa de la forma como los usuarios reciben los datos.

Nivel Conceptual: Es el nivel de mediación entre los 2 anteriores:

externo (aplicaciones)

Conceptual (modelo,(entidad/relaciòn))

Interno (Hardware)

ARQUITECTURA DE UN SISTEMA DE BASES DE DATOS

Nivel Externo: Es el nivel del usuario individual, es decir, los usuarios pueden ser programadores en algunos casos usuarios finales, cada usuario dispone de un lenguaje y en el caso de un programador. Dispone de un lenguaje convencional. En el caso de un usuario final, será un lenguaje de consulta o un Lenguaje orientado hacia los usuarios. El punto importante de todos estos lenguajes es que debe incluir un sublenguaje de datos del cual estará inmerso o dentro de un lenguaje anfitrión, un lenguaje dado, cualquier va ha permitir el empleo de varios lenguajes anfitriones y varios sublenguajes para datos.

ejemplo:

lenguaje VB o C	>>>>>	lenguaje anfitrión
Access	>>>>>	Sublenguaje.
DBSE	>>>>>	lenguaje arquitectónico
ASSIST	>>>>>	SUB−LENGUAJE.
SQL	>>>>>	sub−lenguaje.

Nivel Conceptual: La vista conceptual es una presentación de toda la información contenida en la base de datos. Además puede ser muy diferente en la forma en que percibe los datos cualquier usuario final, es decir, debe ser un panorama de los datos. Tal como son y no como los percibe los usuarios. Debido a las limitaciones del lenguaje o bien al equipo que se esta utilizando.

El nivel conceptual se define mediante un esquema conceptual el cual incluye la definición de cada uno de los tipos de registros (entidades), además, el esquema conceptual no debe asociarse a representaciones de campos almacenados tales como punteros, índices, etc., si el esquema conceptual se desarrolla en forma independiente de los datos entonces el esquema externo definido en base al esquema conceptual será también independiente de los datos.

Nivel Interno: Representación de bajo nivel de toda la base de datos, se compone de varias ocurrencias, de varios tipos de registros, el nivel interno todavía esta aún paso del nivel físico ya que no se manejan los registros fijos. La vista interna se define a través de un esquema interno el cual no sólo define los diversos tipos de registros almacenados, si no, también especifica los índices asociados, representación de los campos almacenados, secuencia física de los registros, etc.

CORRESPONDENCIA ENTRE NIVELES

- LA CORRESPONDENCIA ENTRE NIVEL EXTERNO Y NIVEL CONCEPTUAL: Es la que existe entre una determinada Vista Externa y la Vista Conceptual. La diferencia que puede existir entre estos dos niveles son similares a las que pueden existir entre la vista conceptual y la vista interna. Ejemplo: Los campos pueden tener distintos tipos de datos, los nombre de los campos y registros pueden diferir entre sí, pueden combinarse varios campos conceptuales para formar un campo externo.
- CORRESPONDENCIA ENTRE EL NIVEL CONCEPTUAL Y EL NIVEL INTERNO: Es la que existe entre la vista conceptual y la vista interna especifica como se representan los registros y campos conceptuales, si se modifica la estructura de la Base de Datos, es decir, nivel interno, debe también modificarse la correspondencia para no variar el esquema conceptual.

Como se sabe el DBA es la persona que toma decisiones estratégicas y políticas con respecto a la información de la empresa, además de diseñar los aspectos técnicos necesarios para poner en práctica estas decisiones, es decir, será encargado del control general del problema a nivel técnico, además otras funciones podrían ser:

- Definir el sistema conceptual y
- Definir entidades y relaciones,
- Definir el esquema interno, además, de decir como se representa la información en la base de datos.
- Vinculación con los usuarios encargándose de la comunicación con estos, garantizando la disponibilidad de los datos y además describir los esquemas externos necesarios (aplicaciones).

SISTEMAS DE ADMINISTRACION DE BASES DE DATOS (DBMS)

Es un conjunto de programas que maneja todo el acceso a la Base de Datos , es decir :

1.− Un usuario solicita acceso empleando algún sub−lenguaje de datos.

2.− El DBMS interpreta la solicitud y la analiza.

3.− El DBMS inspecciona en orden el esquema externo de ese usuario, la correspondencia externa conceptual asociada, el esquema conceptual la correspondencia conceptual interna y la estructura de almacenamiento.

4.− El DBMS ejecuta las operaciones necesarias sobre la base de datos almacenados.

FUNCIONES ESPECIFICAS DEL DBMS

1.− Definición de Datos: Debe ser capaz de aceptar definiciones de datos (esquema externo, conceptual Interno y todas las correspondencias asociadas. en versión Fuente) y convertirla en una versión de objeto apropiada. Debe incluir componentes de procesadores de lenguajes para cada uno de los diversos lenguajes de definición de datos, además debe tener las definiciones en DBL (Lenguaje Definición de datos) para poder interpretar y resolver las solicitudes.

2.− Manipulación de Datos: Debe atender las solicitudes del usuario tales como eliminación, modificación, extracción etc., es decir, debe incluir un componente procesador de lenguajes de manipulación de datos (DML). En general las solicitudes en DML pueden ser planificadas y no planificadas.

EJEMPLOS

SOLICTUD PLANIFICADA: Es aquella que se estima con anterioridad antes de que sea ejecutada por primera vez y para estos casos el DBA debe garantizar un buen desempeño de estas solicitudes.

SOLICITUD NO PLANIFICADA: Es una consulta, cuya necesidad no estimo en el diseño físico de la BD puede no estar preparada ante este tipo de consultas y la mejor prueba para el DBMS es que pueda responder ante dicha solicitud con un buen desempeño.

3.− SEGURIDAD E INTEGRIDAD DE LOS DATOS: El DBMS debe supervisar las solicitudes de los usuarios y rechazar los intentos de violar las necesidades de seguridad e integridad de los datos definidos para el DBA.

4.− RECUPERACION Y CONCURRENCIA DE LOS DATOS: El DBMS debe cuidar el cumplimiento de ciertos controles de recuperación y concurrencia. El Administrador de transacciones actúa en caso de que el DBMS no funciones.

5.− DICCIONARIO DE DATOS: El DBMS debiera incluir una función de diccionarios de datos (Meta _ datos), se puede decir que es una base de Datos del sistema y no del usuario cuyo contenido puede considerarse como Datos acerca de los Datos que en el fondo son definiciones de objetos de datos y otros objetos del sistema.

6.− DESEMPEÑO. El DBMS debiera ejecutar todas las funciones anteriores en la forma mas eficiente.

7.− ADMINISTRADOR DE COMUNICACIONES DE DATOS: Las solicitudes de un usuario final son dirigidas a la Base de datos donde son transmitidas en forma de mensajes de comunicación o a la inversa las respuestas al usuario son tomados como mensajes del mismo tipo, todas las transmisiones se efectúan bajo el control de otros grupos de programas llamados administrador de comunicación de datos el cual no forma parte del DBMS, sino que es un programa autónomo pero debe trabajar en forma conjunta con el DBMS para poder satisfacer los requerimientos de los usuarios.

ENFOQUE RELACIONAL

El modelo relacional es una forma de ver los datos, es decir, una receta para representar los datos mediante tablas y para manipular su representación. El modelo relacional se ocupa de 3 puntos importantes de los datos:

1. su estructura.

2.− su integridad, y

3.− Manipulación

1.− ESTRUCTURA DEL ENFOQUE RELACIONAL: Los componentes de la estructura son básicamente Relación, Tupla, Cardinalidad, Atributo, Grado, Dominio y Clave Primaria.

Una relación corresponde a lo que se conoce como tabla o entidad; una tupla corresponde a una fila de esa tabla, un atributo corresponde a una columna de la misma tupla, el número de tuplas se denomina cardinalidad, el número de atributos se llama grado, la clave primaria es un identificador único para la tabla, es decir, una columna o combinación de una columna con la siguiente propiedad NUNCA EXISTEN DOS FILAS DE LA TABLA CON EL MISMO VALOR DE UNA COLUMNA O COMVINACION DE UNA COLUMNA. El Dominio es una combinación de valores de los cuales uno o mas atributos obtienen sus valores reales.

La estructura de un enfoque relacional es en sí donde las asociaciones entre tuplas se representan únicamente por valores de datos en las columnas sacadas de un dominio común.

TERMINO RELACIONAL	TERMINO INFORMAL

RELACION	TABLA
TUPLA	FILA O REGISTRO
CARDINALIDAD	Nº DE FILAS
ATRIBUTO	CAMPO O COLUMNA
GRADO	Nº DE COLUMNA
CLAVE PRIMARIA	IDENTIFICADOR UNICO
DOMINIO	VALORES LEGALES

DOMINIO: Corresponde un conjunto de valores que están entre algún tipo de rango, por ejemplo: El Dominio del código producto; es el conjunto de todos los productos posibles, el dominio del valor unitario es el conjunto de los reales mayores que 0, es decir, los dominios son fondos de valores de los cuales se extraen los valores reales, que aparecen en los atributos. Cada atributo debe estar definido sobre un dominio lo que significa que los valores de ese atributo deben proceder de ese dominio especifico.

RELACIONES: Están compuestas por dos partes una cabecera y un cuerpo, se define la relación sobre un conjunto de dominio, la cabecera esta formada por un conjunto fijo de atributos tal que cada atributo A(j) corresponde a cada uno de los Dominios de J, con J variando hasta N.

El cuerpo esta formado por un conjunto de tuplas el cual varía con el tiempo, cada tupla esta formada por un conjunto de pares (atributo, valor) Donde (i) varia de 1 hasta N y N es el número de tuplas del conjunto (A1, V1) (A2, V2), (A3,V3).....(An, Vn) N va hacer el grado de esa relación.

TIPOS DE RELACIONES:

1.− RELACIONES BASE O REALES: Corresponde al concepto de Tabla es decir una relación autónoma cuya importancia esta dada por el diseñador para un uso especifico dentro de una aplicación

2.− RELACIONES VIRTUALES: (Relaciones de Vistas) Una vista es una relación derivada con nombre representada dentro del sistema exclusivamente mediante su definición en término de otras relaciones, no posee datos almacenados propios, separados y distinguibles a diferencia de las relaciones Bases, en si una VISTA.

3.− RELACIONES INSTANTANEAS: (Snap Shop) Es también una relación derivada con nombre como una vista pero a diferencia de esta última las instantáneas son reales no virtuales, es decir, están representadas no solo por su definición, en término de otras relaciones con nombre, sino, también por sus propios datos almacenados

(Snap Shop = consulta rápida, corta)

REGLAS GENERALES DE INTEGRIDAD

La mayor parte de las bases de datos están sujetas a un gran número de reglas de integridad. Por ejemplo los códigos de ciertos objetos deben tener la forma XXX los valores posibles de un atributo podrían variar entre 1 y 9.999; las ciudades deberían provenir de cierta tabla previamente definida; El atributo stock físico debería ser mayor que 0, etc.

La primera regla general de integridad se relaciona con las claves primaria y la segunda, con las claves ajenas.

Claves Primarias: En términos informales la clave primaria de una relación, es solo un identificador único para esa relación, la componen dos o mas atributos de la misma relación o también puede ser un resumen de alguna porción de los atributos de la relación. Ca be la posibilidad de definir claves cuyo identificador no es

único para una tupla especifica o para un conjunto de tuplas, en este caso la clave pasa a ser llamada clave secundaria, presentando una probabilidad de el 60% de que sea duplicada, también en este tipo de clave se conoce como clave alternativa duplicada, para el caso de accesar una o un grupo de tuplas, siendo esta como una segunda alternativa. Para poder determinar cual es la clave especifica se debe primero examinar los atributos de la relación, es decir, claves candidatas y dentro de esta claves candidatas una de ella sería la clave primaria y el resto claves secundarias si es posible. Las condiciones que debiera tener la clave candidata es si el atributo K de la relación R es una clave candidata de R si y solo si satisface las siguientes dos propiedades independiente (unicidad y minimicidad).

1. Unicidad: En cualquier momento no existen dos tuplas en R con el mismo valor de K.

2. Minimicidad: Si K es compuesto no será posible eliminar ningún componente de K sin destruir la propiedad de Unicidad.

Toda relación tiene por lo menos una clave candidata, el razonamiento para elegir la clave primaria cuando existen varias claves candidatas, queda del alcance del modelo relacional en la práctica la elección es mas sencilla. En si la clave primaria es la que tiene mayor importancia, las claves candidatas y alternativas son solo conceptos que nacen durante el proceso de definición de la clave primaria.

Sub reglas de Integridad:

a) Ninguna clave primaria debe contener valores nulos.

b) Ninguna base de datos relacional registrará información de algo que no se puede identificar plenamente.

CLAVES AJENAS: Es un atributo de una relación R2 cuyos valores deben concordar con las llaves primarias de alguna relación R1, es decir, una valor de la clave ajena representa una referencia a la tupla donde se encuentra el valor correspondiente de la clave primaria(Tupla referenciada o tupla objetivo).

ALGEBRA RELACIONAL

Consiste en un conjunto de operadores de alto nivel que operan son relaciones, cada uno de estos operadores toma una o dos relaciones como entrada y produce una nueva relación como salida. Los operadores se dividen en dos grupos:

a) El primero involucra las operaciones tradicionales de conjunto tales como unión, intersección producto cartesiano y diferencia.

b) El segundo involucra las operaciones relacionales tales como restricción, proyección, división y reuniones.

DEFINICIONES:

UNION: Construye una relación formada por todas las tuplas que aparecen en cualquiera de las dos relaciones involucradas.

INTERSECCIÒN: Construye una relación formada por aquellas tuplas que aparezcan en las dos relaciones involucradas.

DIFERENCIA: construye una relación formada por todas las tuplas de la primera relación que no aparezcan en la segunda relación.

PRODUCTO: A partir de 2 relaciones especificadas construye una relación que contiene todas las tuplas

combinadas (Una de cada una de las relaciones).

RESTRICCION: Extrae las tuplas especificadas en una relación dada, es decir restringe la relación a tuplas que satisfagan una condición.

PROYECCIÒN: Extrae los atributos especificados de una relación dada, (salida o lista)

REUNION: A partir de 2 relaciones especificadas construye una relación que contiene todas las posibles combinaciones de tuplas, una de cada una de las relaciones tales que las dos tuplas participantes en una combinación dada satisfagan alguna condición especifica. Esta es muy parecida a la restricción pero depende de los elementos que intervienen.

NORMALIZACION DE DATOS

Frecuentemente se reorganiza la información a partir de la información que obtiene el analista o programador a través de un análisis de requerimientos. El principio Básico de la normalización es optimizar dicha información a partir de reagrupamientos sucesivos, eliminando completamente la redundancia e inconsistencia de la información; Además de simplificar la estructura de los datos, por lo tanto, el proceso de Normalización identifica los datos redundantes que pueden existir en una estructura lógica, determina claves únicas necesarias para el acceso de los elementos de datos y ayuda a establecer las relaciones necesarias entre los elementos de datos, generalmente se aplican 3 niveles de Normalización llamados FORMAS NORMALES (1FN, 2 FN, 3FN).

1.− 1FN: La Primera Forma Normal consiste en agrupar los datos relacionados entre si de una manera tal que ninguna estructura en lo posible tenga datos repetidos.

2.− 2 FN: La Segunda Forma Normal se debe reorganizar las relaciones de manera que ningún dato que no sea clave quede completamente dependiente.

3.− 3FN: La Tercera Forma Normal: no se puede realizar si todas las condiciones de la 2 FN no son satisfechas y consiste en eliminar aquellos datos que no sean claves y que puedan derivarse de una combinación de otros datos y tampoco son claves en ninguna otra relación.

ACCES

BASES DE DATOS, TABLAS, CONSULTAS, INFORMES, FORMULARIOS

NOMBRE DE CAMPO: Corresponde al nombre definido de los atributos que forman parte de la tabla en cuestión. Ej. Código curso, dirección teléfono etc., Las características soportan blancos pero no es recomendable.

TIPOS DE TABLAS: Access soporta 8 tipos de datos diferentes cada uno con un propósito especial.

Tipos de Datos: Uso Tamaño Texto Datos Alfanuméricos Hasta 255 bytes Memo Datos Alfanuméricos Frases y párrafos Hasta 644 bytes Numéricos Datos Numéricos 1,2,4, u 8 bytes Fecha/Hora Fechas − Horas 8 bytes MonedanDatos que representan cantidades monetarias almacenados con 4 lugares decimales de precisión 46 bytes Auto numero Valor único generado por Access para cada nuevo registro 4 bytes Si/No Datos Boleado 1 BIT Objeto Ole Imágenes, gráficos u otros objetos de otras aplicaciones Windows Hasta 1 gigabyte.

Para elegir datos de carácter normalmente, se selecciona el tipo Dato Texto la longitud máxima de un campo de texto se controla utilizando una propiedad de campo (tamaño de campo)

Para utilizar el Tipo de Datos Memo solamente lo haremos para texto que superen los 255 caracteres o que deban contener caracteres de formato tales como tabuladores, retorno de carro etc.

Cuando elijamos el tipo de datos Numéricos es necesario tener mucho cuidado con lo que se introducirá en la propiedad tamaño del campo ya que estas elecciones afectan tanto a la precisión como a la longitud. Ejemplo: un número entero no posee decimales.

El tipo de datos Fecha Hora es útil para datos de calendario o de reloj y tiene la ventaja de permitir operaciones aritméticas con los minutos, segundos, hora día mes o año. Por ejemplo podemos encontrar la diferencia entre dos valores de fecha hora .

El Tipo de Datos Moneda: para almacenar cantidades de dinero, también podemos utilizar este tipo de datos para cualquier campo numérico que posea un numero fijo de hasta 4 decimales. El tipo de Dato Moneda posee la precisión de los enteros pero con un número fijo de decimales.

El Tipo de Datos Autonumericos (contador) esta diseñado especialmente para la generación automática de valores de clave principal. Dependiendo de las propiedades del tamaño de campo y nuevos valores que hayan sido seleccionados para un campo auto numérico (contador) podrá hacer que Access cree un entero largo aleatorio o secuencial Recordemos que una tabla podrá contener un único campo con el tipo de datos autonumerico.

Para almacenar valores boléanos (verdadero o falso) usaremos el tipo de datos Si/No. Este tipo de Datos es particularmente útil para indicar si una cuenta está pagada o no. O para verificar si una determinada comprobación ha sido superada. Etc.

El tipo de datos Objeto Ole permite almacenar datos complejos como fotografías, gráficos, sonidos que pueden ser mantenidos mediante un enlace dinámico con otra aplicación basada en Microsoft Windows. Ejemplo Access puede almacenar y permitir la edición de un documento de Microsoft Windows una hoja electrónica de Excel, una diapositiva de Power Point un archivo de sonido .wav un archivo de video . avi o imágenes creadas utilizando la aplicación Draw o Paint Break.

Propiedades de los Campos: Podemos personalizar cada uno de los campos mediante el ajuste de determinadas propiedades. Estas propiedades varían según el tipo de datos que hayan sido seleccionados.

Tamaño del campo: Permite especificar la longitud de los tipos de datos, texto, y numéricos.

Tipo texto: Contiene de 0 a 255 caracteres de longitud, con una longitud predeterminada de 50 caracteres. Para el tipo de datos numéricos, los tamaños de campos son:

Byte: Contiene valores comprendidos entre 0 y 255.

Entero: Contiene valores comprendidos entre 32.768 hasta 32.767.

Entero Largo: Contiene valores comprendidos entre −2.147.483.648 y +2.147.483.647.

Simples: Número coma flotante que contiene valores que van desde −3,4X10 elevado 38 hasta +3,4 X10 elevado 38.

Doble: Número coma flotante que contiene valores que abarcan desde −1.797x 10 elevado a 308 hasta + 1.797 X10 elevado a 308.

Formato: Permite controlar la forma de visualización o de impresión de los datos, las opciones de formato

varían según el tipo de datos, para los tipos texto y memo, podemos especificar un formato personal. Para los tipos de datos Numéricos, Moneda y Autonumericos las opciones de formato estándar son:

Número General: Es el valor predeterminado sin puntos ni símbolos de moneda, los lugares decimales mostrados dependen de la posición de los datos Ej. 3456,78%.

Moneda: Antepones símbolos de moneda y 2 símbolos decimales.

Fijo: Al menos 1 dígito y dos lugares decimales.

Estándar: Formato de dos lugares decimales y millares por punto.

Porcentaje: Antepone el carácter porcentaje Ej. 75%.

Científico: Expresa el número en una notación científica Ej. 1,05 x 10 elevado 4.

Para el tipo de datos fecha y hora: Las opciones de formato son fecha general = mes/día/año 10/26/98 19:40:40 PM (EE.UU.) día/mes/año 26/10/98 19:50:40 PM (Gran Bretaña). Fecha Larga; Lunes, 26 de octubre de 1998. Fecha mediana 26 Oct. 98; Fecha Corta 26/10/98 hora larga 17:34:34 PM Hora mediana 05:34 PM Hora Corta 17:34.

Para el tipo de datos Si/No (Boléanos)

Si = Verdadero o activado

No = Falso o Desactivado.

Lugares decimales: Cuantos decimales queremos que contenga nuestro campo. Para los tipos de datos numéricos y moneda podemos especificar el número de decimales a visualizar. Por defecto asume 2 decimales. También podemos solicitar una presentación fija de lugares decimales con un rango de 0 a 15.

Máscara de Entrada: Para los tipos de datos texto numérico, moneda, fecha y hora podemos especificar la plantilla o máscara que el usuario verá cuando introduzca datos en el campo. Por ejemplo para un campo fecha (−−/−−/−−) también podemos presentar un formato para una combinación números y letras tal como (###)000−000.

Carácter de Máscara(fotocopias).

Título: Permite introducir un nombre de campo más descriptivo que Access visualizar en las etiquetas de los formularios y en los encabezados de informes.

Valor Predeterminado: Permite especificar un valor predeterminado para todos los tipos de datos excepto para autonumerico y objeto ole. Para los números el valor predeterminado es 0. Para los tipos de datos texto y memo Access proporciona una cadena vacía estándar.

Regla de Validación: Permite especificar una expresión que deberá cumplirse para que se puedan introducir o modificar los datos de ese campo ejemplo <100 especifica que un número debe ser menor que 100. También podemos comprobar que el valor del campo se encuentre dentro de una serie de valores. Por ejemplo podemos hacer que Access compruebe una lista de ciudades validas especificando Santiago o Iquique o Valdivia.

Texto de validación: Permite definir el texto que Access visualizará siempre que los datos introducidos no cumplan la regla de validación.

Requerido: Si no deseamos que este campo tenga un valor nulo debemos establecer su propiedad como si.

Permite Longitud cero (Texto): Para los campos memo y texto podemos establecer el campo igual a cadena de longitud 0 () sin espacio.

Indexado: Podemos acelerar el acceso a los tipos de datos texto, numérico, fecha hora, moneda y autonumerico. Solicitando la incorporación de un índice. Además podemos requerir que los valores del campo indexado siempre sean únicos en toda la tabla. Ejemplo código tipo teléfono

100−2 (7)

108−4 (8)

100−2 (9)

50−1(10)

60−3 (12)

50−1 (5)

SQL (LENGUAJE DE CONSULTA)

Introducciones que permiten trabajar sobre las bases de datos hay dos formas de trabajar las consultas 1 en forma de diseño, es decir.

CONSULTA: Este lenguaje consiste en sentencias muy próximas al ingles diseñadas para seleccionar registros de una o varias tablas de acuerdo con el criterio utilizado. Las sentencias de solicitud SQL pueden ser usadas en el momento del diseño en la propiedad recordsource de un control de datos esto permitiría crear mediante programación Dynaset, Shapstmos asociados a controles de datos los criterios SQL utilizan la palabra clave SELECT, seguida de una de estas palabras claves WHERE, FROM HAVING, GROUP−BY, ORDER BY.

CARACTERISTICAS DE SQL: El SQL es un lenguaje basado en el idioma ingles usa palabras tales como SELECT, FROM, WHERE, GROUP−BY, ORDER−BY, como parte del conjunto de comandos. El SQL no es un lenguaje de procedimientos es decir el usuario especifica que información necesita y no como obtenerla, en otras palabras el SQL no le pide al usuario el método de acceso. SQL procesa 1 o más tablas a la vez. SQL puede ser usado por un rango de usuarios que incluye al administrador de base de datos, programadores, personal administrativo y otros tipos de usuarios.

SELECT : La sentencia SELECT forma el núcleo de base de datos de SQL esta sentencia sirve para seleccionar o recuperar la filas y columnas deseadas de las tablas de nuestra base de datos. La sintaxis de la sentencia SELECT consta de 5 cláusula construida normalmente de la siguiente manera:

SELECT < LISTA DE CAMPOS>

FROM<LISTA DE TABLAS>

(WHERE < Especificación de Selección de Filas>)

(GROUP BY <Especificación de Agrupación>)

(HAVING<Especificación de selección de grupos>)

(ORDER BY <Especificación de Ordenación>).

WHERE: Para hacer una selección desde la misma tabla bajo ciertas condiciones o con alguna restricción, se debe usar la cláusula WHERE, que corresponde al operador del Álgebra Relacional llamado Restricción.

Los operadores de combinación que pueden ser usados por el comando WHERE son los lógicos y los propios de SQL:

OPERADORES LOGICOS:

= IGUAL

MAYOR

>= MAYOR IGUAL

< MENOR

<= MENOR IGUAL.

OPERADORES SQL:

Existen 4 operadores SQL con los cuales se puede operar cualquier tipo de datos de una Base de Datos.

BETWEEN ... AND .... : Permite encontrar un conjunto de valores a partir de dos valores dados incluyendo también estos mismos.

Ej. Sueldo Base y teléfono (70000 y 300000)

SELECT NOMBRES, APELLIDOS, S_BASE, TELEFONO

FROM EMPLEADOS

WHERE S_BASE BETWEEN 70000 AND 300000;

IN : El operador IN chequea los valores en una lista especifica.

WHERE NOMBRE ATRIBUTO IN (11111−11112−11113)

Ejemplo:

SELECT NOMBRE, APELLIDOS, S_BASE, TELEFONO, FICHA

FROM EMPLEADOS

WHERE FICHA IN (11111−1112−11113);

** CUANDO QUIERA VER CARACTERES PONER APOSTROFE.

LIKE: El operador LIKE se usa cuando no se sabe exactamente el valor a buscar, a se conoce solo una parte

de él usando este operador es posible seleccionar filas parecidas con un patrón de caracteres en el cual se reemplaza lo desconocido por un caracteres en el cual se reemplaza lo desconocido por un carácter *, para reemplazar un solo carácter ocupamos el comodín ¿

*reemplaza todo

¿ reemplaza un carácter

Mostrar todos los atributos de la tabla empleados cuyos apellidos comiencen con la letra S. los apellidos que terminen con S

IS NULL: El operador concretamente busca los campos cuyos valores son nulos.

Ejemplo:

Buscar todos los empleados que no tienen valoren campo carga

WHERE NOMBRE atributo IS NULL

SELECT *

FROM EMPLEADOS

1 WHERE CARGAS IS NULL;

2 WHERE CARGAS IS NULL OR NOMBRES IS NULL;

OPERADORES NEGADOS:

NOT BETWEEN.... AND...

CARGAS (2 AND 4)

EJEMPLO

SELECT *

FROM EMPLEADOS

WHERE S_BASE NOT BETWWEN 100000 AND 800000;

NOT IN: Va ha mostrar todos lo que este fuera de la cadena

NOT LIKE: El patrón del campo no debe ser igual al patrón dado.

IS NOT NULL:

OPERADORES ARITMETICOS: Otros componentes de la cláusula SELECT son las expresiones aritméticas, una expresiones aritméticas es una combinación de uno o más valores con operadores que le dan valor a un dato las expresiones aritméticas pueden contener: Nombres de las columnas, valores constantes numéricos y los operadores aritméticos suma, resta, multiplicación, división. Si una expresión aritmética tiene más de un operador la primera prioridad la tiene los operadores, multiplicación y división, si se quiere alterar

este orden se debe usar los paréntesis.

SELECT NOMBRE, APELLIDOS sueldo *1,5.

SELECT NOMBRES, APELLIDOS, S_BASE * 1,5

FRON EMPLEADOS;

OPERADORES RELACIONALES: Los operadores relacionales con una o más tablas restricción esta operación selecciona y despliega datos de una tabla es posible desplegar todas las filas o sólo aquellas filas que cumplan con una condición o varias condiciones esto también es conocido como sub−conjunto horizontal.

RESTRICION: Pregunta cuales son los proveedores Santiaguinos

Respuesta: Ciudad = Santiago

PROYECCIÓN: Pregunta cuales son las ciudad en donde hay Proveedores

Respuesta: SELECT CIUDAD

FROM PROVEEDORES;

PRODUCTO:

Tabla elemento Tabla Proveedores

CODIGO 2 1 ATRIBUTO

Descripción Color

Elemento Tarro pintura.

PRODUCTO es el resultado de cuando las filas de dos tablas son concatenadas, todas las filas de la primera tabla son concatenada con las filas de la segunda tabla, esto produce una nueva tabla.

Las tablas no necesitan tener la misma estructura (se van unir). Usualmente al hacer esta operación se produce un producto cartesiano sin una condición de igualdad.

** cuando se crea la consulta hay que agregar en el SELECT el nombre de la tabla más el atributo.

Ejemplo:

SELECT ELEMENTOS.DESCRIPCION, COLORES.COLOR

FROM COLORES, ELEMENTO

Restricción WHERE.COLORES.COLOR='AZUL';

EL JOIN: Es el resultado de cuando las filas que están en dos tablas son conectadas de acuerdo a una condición doble.

Como trabaja: crear dos tablas

1 Pacientes

Código número de 3

Nombre paciente

Código de cama

2 Ubicación

Código de cama.

Descripción

Seleccionar

Nombre de paciente y ubicación. Y queda?

Ejemplo:

Conectar dos tablas de acuerdo a un atributo en común:

SELECT PACIENTE.NOMBRE_PACIENTE, UBICACIÓN.Ubicación

FRON PACIENTE, Ubicación

WHERE UBICACIÓN Ubicación.CODCAMA=PACIENTE.CODCAMA:

Otra etapa importante es la unión.

UNION: Despliega todas las filas que están en una de las dos tablas. Para que dos tablas puedan unir las estructuras de los datos seleccionados dichas estructuras de los datos deben ser compatibles; es decir, si en ambas tablas hay tuplas iguales queda solo una de ellas por lo tanto la duplicidad de elementos no se nota:

ORDER BY DIVISION

Crear dos tablas = Proveedores

Nombre texto

Ciudad Texto

Código N° entero o 0000

SELECT NOMBRE, CIUDAD, CODIGO

FROM PROVEEDORES

UNION

SELECT NOMBRE, CIUDAD, CODIGO

FROM PROVEEDORES, PROVEEDORES 2;

INTERSECCIÓN: Muestra todos los elementos que se repiten en las dos tablas. Despliega todas las filas que están en las dos tablas a la vez, también debe cumplir las mismas condiciones que para la unión y queda:

SELECT PROVEEDORES

FROM PROVEEEDORES, PROVEEDORES

WHERE PROVEEDORES.CODIGO=PROVEEDORES2.CODIGO

DIFERENCIA: Es lo que está en la primera tabla que no se repite en la segunda.

SELECT PROVEEDORES

FROM PROVEEDORES

WHERE PROVEEDORES.CODIGO<>PROVEEDORES.CODIGO

O WHERE PROVEEDORES.CODIGO NOT=PROVEEDORES.CODIGO

FUNCIONES AGREGADAS

Las funciones agregadas son las que actúan sobre los grupos de datos, m{as sobre filas individuales. Los grupos de datos pueden ser una columna o un conjunto de columnas incluyendo toda la tabla.

La función agregada es aplicada en la línea del SELECT como si fuera un nombre de columna la sintaxis general es:

SELECT FUNCION (NOMBRE COLUMA O *)

FROM

FUNCION AVG= Calcula el promedio de un conjunto de valores.

FUNCION COUNT= Cuenta el número de ocurrencia de los miembros de un conjunto.

FUNCION MAX= Determina el valor máximo de un conjunto de valores.

FUNCION MIN= Determina el valor mínimo de un conjunto de valores.

FUNCION SUM= Suma un conjunto de valores.

FUNCIONES DE GRUPO DE FILAS

Las cláusulas DROPDRIVE sirve para ocupar los datos de acuerdo a un atributo en común AVG(S_BASE)

El SQL internamente hace la clasificación de acuerdo al atributo incorporando en el atributo DROPDRIVE

Una vez que los grupos están formados, la función asociada al comando SELECT es aplicada en forma individual a estos grupos, es decir, primero se ocupa DROPDRIVE.

Ejemplo:

Tabla empleado

SELECT CentroCosto, AVG(S_BASE)

FROM EMPLEADOS GROUP BY CentroCosto

HAVING= Al igual que la cláusula WHERE para especificar la búsqueda por condición para grupos de filas se usa la cláusula HAVING.

SELECT

FROM

(WHERE)

GROUP BY

HAVING= Actúa sobre un grupo seleccionado.

Buscar todos los promedios −<< 150000.

SELECT CentroCosto, AVG(S_BASE) AS PROMEDIO.

FROM EMPLEADOS

GROUP BY CentroCosto

HAVING AVG(S_BASE) < 150000;

ORDER BY= La función ORDER BY permite ordenar los datos de acuerdo al valor de un atributo asociado, este orden puede ser tanto ascendente como descendente.

ORDER BY NOMBRE CAMPO (ASC), CAMPO2 ASC

ORDER BY NOMBRE CAMPO (DESC), CAMPO2 DESC

Ejemplo:

SELECT CentroCosto, AVG(s_BASE)

FROM EMPLEADOS

GROUP BY CentroCosto

ORDER BY CentroCosto ASC; O

ORDER BY CentroCosto DESC;

SELECT CentroCosto, AVG(s_BASE), COUNT(FICHA)

FROM EMPLEADOS

WHERE TURNO <>NO

GROUP BY CentroCosto

ORDER BY CentroCosto DESC;

Consideraciones Generales

En un sistema de base de datos distribuida, los datos se almacenan en varios computadores. Los computadores de un sistema distribuido se comunican entre sí a través de diversos medios de comunicación, tales como cables de alta velocidad o líneas telefónicas. No comparten la memoria principal ni el reloj.

Los procesadores de un sistema distribuido pueden variar en cuanto su tamaño y función. Pueden incluir microcomputadores pequeños, estaciones de trabajo y sistemas de computadores grandes de aplicación general. Estos procesadores reciben diferentes nombres, tales como localidades, nodos o computadores.

Un sistema distribuido de bases de datos consiste en un conjunto de localidades, cada uno de las cuales puede participar en la ejecución de transacciones que accedan a datos de una o varias localidades. La diferencia principal entre los sistemas de base de datos centralizados y distribuidos es que, en los primeros, los datos residen en una sola localidad, mientras que, en los últimos, se encuentran en varias localidades.

Estructura de Base de Datos Distribuidas

Un sistema distribuido de base de datos consiste en un conjunto de localidades, cada una de las cuales mantiene un sistema de base de datos local. Cada localidad puede procesar transacciones locales, o bien transacciones globales entre varias localidades, requiriendo para ello comunicación entre ellas.

Las localidades pueden conectarse físicamente de diversas formas, las principales son:

Red totalmente conectada

Red prácticamente conectada

Red con estructura de árbol

Red de estrella

Red de anillo

Las diferencias principales entre estas configuraciones son:

Coste de instalación: El coste de conectar físicamente las localidades del sistema

Coste de comunicación: El coste en tiempo y dinero que implica enviar un mensaje desde la localidad A a la B.

Fiabilidad: La frecuencia con que falla una línea de comunicación o una localidad.

Disponibilidad: La posibilidad de acceder a información a pesar de fallos en algunas localidades o líneas de comunicación.

Las localidades pueden estar dispersas, ya sea por un área geográfica extensa (a lo largo de un país), llamadas redes de larga distancia; o en un área reducida (en un mismo edificio), llamadas redes de área local. Para las primeras se utilizan en la comunicación líneas telefónicas, conexiones de microondas y canales de satélites; mientras que para las segundas se utiliza cables coaxiales de banda base o banda ancha y fibra óptica.

Consideraciones al distribuir la base de datos

Existen varias razones para construir sistemas distribuidos de bases de datos que incluyen compartir la información, fiabilidad y disponibilidad y agilizar el procesamiento de las consultas. Pero también tiene sus desventajas, como desarrollos de software más costosos, mayor posibilidad de errores y costos extras de procesamiento.

Ventajas de la distribución de datos

La principal ventaja de los sistemas distribuidos es la capacidad de compartir y acceder a la información de una forma fiable y eficaz.

Utilización compartida de los datos y distribución del control

La ventaja principal de compartir los datos por medio de la distribución es que cada localidad pueda controlar hasta cierto punto los datos almacenados localmente. En un sistema centralizado, el administrador de base de datos de la localidad central controla la base de datos. En un sistema distribuido existe un administrador global de la base de datos que se encarga de todo el sistema. Parte de esta responsabilidad se delega al administrador de base de datos de cada localidad. Dependiendo del diseño del sistema distribuido, cada administrador local podrá tener un grado de autonomía diferente, que se conoce como autonomía local. La posibilidad de contar con autonomía local es en muchos casos una ventaja importante de las bases de datos distribuidas.

Fiabilidad y disponibilidad

Si se produce un fallo en una localidad de un sistema distribuido, es posible que las demás localidades puedan seguir trabajando. En particular, si los datos se repiten en varias localidades, una transacción que requiere un dato específico puede encontrarlo en más de una localidad. Así, el fallo de una localidad no implica necesariamente la desactivación del sistema.

El sistema debe detectar cuando falla una localidad y tomar las medidas necesarias para recuperarse del fallo. El sistema no debe seguir utilizando la localidad que falló. Por último, cuando se recupere o repare esta localidad, debe contarse con mecanismos para reintegrarla al sistema con el mínimo de complicaciones.

La disponibilidad es fundamental para los sistemas de bases de datos que se utilizan en aplicaciones de tiempo real. Por ejemplo, si una línea aérea no puede tener acceso a la información, es posible que pierda clientes a favor de la competencia.

Agilización del procesamiento de consultas

Si una consulta comprende datos de varias localidades, puede ser posible dividir la consulta en varias subconsultas que se ejecuten en paralelo en distintas localidades. Sin embargo, en un sistema distribuido no se comparte la memoria principal, así que no todas las estrategias de intersección se pueden aplicar en estos sistemas. En los casos en que hay repetición de los datos, el sistema puede pasar la consulta a las localidades más ligeras de carga.

Desventajas de la distribución de los datos

La desventaja principal de los sistemas distribuidos es la mayor complejidad que se requiere para garantizar una coordinación adecuada entre localidades.

El aumento de la complejidad se refleja en:

Coste del desarrollo de software: es más difícil estructura un sistema de bases de datos distribuidos y por tanto su coste es menor

Mayor posibilidad de errores: puesto que las localidades del sistema distribuido operan en paralelo, es más difícil garantizar que los algoritmos sean correctos.

Mayor tiempo extra de procesamiento: el intercambio de mensajes y los cálculos adicionales son una forma de tiempo extra que no existe en los sistemas centralizados.

Transparencia y Autonomía

En la sección anterior se vio que una relación r puede almacenarse de varias formas en un sistema de base de datos distribuida. Es esencial que el sistema reduzca al mínimo la necesidad de que el usuario se dé cuenta de cómo está almacenada una relación. Como veremos. un sistema puede ocultar los detalles de la distribución de la información en la red. Esto se denomina transparencia de la red. La transparencia de la red se relaciona, en algún modo, a la autonomía local. La transparencia de la red es el grado hasta el cual los usuarios del sistema pueden ignorar los detalles del diseño distribuido. La autonomía local es el grado hasta el cual el diseñador o administrador de una localidad pueden ser independientes del resto del sistema distribuido . Los temas de transparencia y autonomía serán considerados desde los siguientes puntos de vista:

Nombre de los datos.

Repetición de los datos.

Fragmentación de los datos.

Localización de los fragmentos y copias.

Asignación de nombres y autonomía local

Todo elemento de información de una base de datos debe tener un nombre único. Esta propiedad se asegura fácilmente en una base de datos que no esté distribuida. Sin embargo, en una base de dalos distribuida, las distintas localidades deben asegurarse no utilizar el mismo nombre para dos datos diferentes.

Una solución para este problema es requerir que se registren todos los nombres en un asignador central de nombres. Sin embargo, este enfoque tiene varias desventajas:

Es posible que el asignador de nombres se convierta en un cuello de botella.

Si el asignador de nombres se cae, es posible que ninguna de las localidades del sistema distribuido pueda seguir trabajando.

Se reduce la autonomía local, ya que la asignación de nombres se controla de forma centralizada.

Un enfoque diferente que origina una mayor autonomía local es exigir que cada localidad ponga como prefijo un identificador de localidad a cualquier nombre que genere. Esto garantiza que dos localidades nunca generarán el mismo nombre (ya que cada localidad tiene un identificador único). Además, no se requiere un

control central.

Esta solución al problema de asignación de nombres, logra autonomía local, pero no transparencia de la red, ya que se agregan identificadores de localidad a los nombres. Así, la relación depósito podría llamarse localidad17.depósito en vez de depósito simplemente.

Cada copia y fragmento de un elemento de información deben tener un nombre único. Es importante que el sistema pueda determinar qué copias son copias del mismo elemento de información y qué fragmentos son fragmentos del mismo elemento de información.

Transparencia de la repetición y la fragmentación

No es conveniente requerir que los usuarios hagan referencia a una copia específica de un elemento de información. El sistema debe ser el que determine a qué copia debe acceder cuando se le solicite su lectura, y debe modificar todas las copias cuando se produzca una petición de escritura.

Cuando se solicita un dato, no es necesario especificar la copia. El sistema utiliza una tabla−catálogo para determinar cuáles son todas las copias de ese dato.

De manera similar, no debe exigirse a los usuarios que sepan cómo está fragmentado un elemento de información. Es posible que los fragmentos verticales contengan id−tuplas, que representan direcciones de tuplas. Los fragmentos horizontales pueden haberse obtenido por predicados de selección complejos. Por tanto, un sistema de bases de datos distribuido debe permitir las consultas que se hagan en términos de elementos de información sin fragmentar. Esto no presenta problemas graves, ya que siempre es posible reconstruir el elemento de información original a partir de sus fragmentos. Sin embargo, este proceso puede ser ineficiente.

Transparencia de localización

Si el sistema es transparente en cuanto a repetición y fragmentación, se ocultará al usuario gran parte del esquema de la base de datos distribuida. Sin embargo, el componente de los nombres que identifican a la localidad obliga al usuario a darse cuenta del hecho de que cl sistema está distribuido.

La transparencia de localización se logra creando un conjunto de seudónimos o alias para cada usuario. Así, el usuario puede referirse a los datos usando nombres sencillos que el sistema traduce a nombres completos.

Con el uso de seudónimos, no será necesario que el usuario conozca la localización física de un dato. Además, el administrador de la base de datos puede cambiar un dato de una localidad a otra sin afectar a los usuarios.

Esquema completo de asignación de nombres

Ya vimos que un nombre proporcionado por el usuario debe pasar por varios pasos de traducción antes de que pueda servir como referencia a una copia específica de un fragmento determinado en una localidad específica.

Para ilustrar cómo funciona el esquema, consideramos un usuario que se encuentra en la sucursal 1 (L1). Este usuario emplea el seudónimo depósito−local para el fragmento local depósito−F1 de la relación deposito. Cuando este usuario hace referencia a depósito−local, el subsistema de procesamiento de consultas busca depósito−local en la tabla de seudónimos y la sustituye por Ll.depósito.F1. Es posible que L1.depósito.Fl esté repetido. Si es así, debe consultarse la tabla de copias para elegir una copia. Esta copia podría también estar fragmentada, lo que haría necesario consultar la tabla de fragmentación. En la mayor parte de los casos, sólo es preciso consultar una o dos tablas.

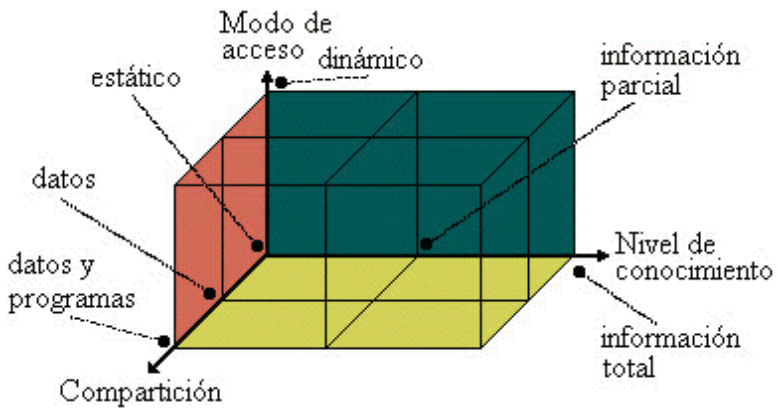
Transparencia y actualizaciones

De alguna forma es más difícil hacer transparente la base de datos para usuarios que la actualizan que para aquellos que sólo leen datos. El problema principal es asegurarse de que se actualizan todas las copias de un dato y también los fragmentos afectados.

En el caso más general, el problema de actualización de información repetida y fragmentada está relacionado con el problema de actualización de vistas.

Diseño de la distribución:

El diseño de un sistema de base de datos distribuido implica la toma de decisiones sobre la ubicación de los programas que accederán a la base de datos y sobre los propios datos que constituyen esta última, a lo largo de los diferentes puestos que configuren una red de ordenadores. La ubicación de los programas, a priori, no debería suponer un excesivo problema dado que se puede tener una copia de ellos en cada máquina de la red (de hecho, en este documento se asumirá que así es). Sin embargo, cuál es la mejor opción para colocar los datos: en una gran máquina que albergue a todos ellos, encargada de responder a todas las peticiones del resto de las estaciones − sistema de base de datos centralizado −, o podríamos pensar en repartir las relaciones, las tablas, por toda la red. En el supuesto que nos decantásemos por esta segunda opción, ¿qué criterios se deberían seguir para llevar a cabo tal distribución? ¿Realmente este enfoque ofrecerá un mayor rendimiento que el caso centralizado? ¿Podría optarse por alguna otra alternativa? En los párrafos sucesivos se tratará de responder a estas cuestiones.



Tradicionalmente se ha clasificado la organización de los sistemas de bases de datos distribuidos sobre tres dimensiones: el nivel de compartición, las características de acceso a los datos y el nivel de conocimiento de esas características de acceso (vea la figura 1). El nivel de compartición presenta tres alternativas: inexistencia, es decir, cada aplicación y sus datos se ejecutan en un ordenador con ausencia total de comunicación con otros programas u otros datos; se comparten sólo los datos y no los programas, en tal caso existe una réplica de las aplicaciones en cada máquina y los datos viajan por la red; y, se reparten datos y programas, dado un programa ubicado en un determinado sitio, éste puede solicitar un servicio a otro programa localizado en un segundo lugar, el cual podrá acceder a los datos situados en un tercer emplazamiento. Como se comentó líneas atrás, en este caso se optará por el punto intermedio de compartición.

Respecto a las características de acceso a los datos existen dos alternativas principalmente: el modo de acceso a los datos que solicitan los usuarios puede ser estático, es decir, no cambiará a lo largo del tiempo, o bien, dinámico. El lector podrá comprender fácilmente la dificultad de encontrar sistemas distribuidos reales que puedan clasificarse como estáticos. Sin embargo, lo realmente importante radica, estableciendo el dinamismo como base, cómo de dinámico es, cuántas variaciones sufre a lo largo del tiempo. Esta dimensión establece la

relación entre el diseño de bases de datos distribuidas y el procesamiento de consultas.

La tercera clasificación es el nivel de conocimiento de las características de acceso. Una posibilidad es, evidentemente, que los diseñadores carezcan de información alguna sobre cómo los usuarios acceden a la base de datos. Es una posibilidad teórica, pero sería muy laborioso abordar el diseño de la base de datos con tal ausencia de información. Lo más práctico sería conocer con detenimiento la forma de acceso de los usuarios o, en el caso de su imposibilidad, conformarnos con una información parcial de ésta.

El problema del diseño de bases de datos distribuidas podría enfocarse a través de esta trama de opciones. En todos los casos, excepto aquel en el que no existe compartición, aparecerán una serie de nuevos problemas que son irrelevantes en el caso centralizado.

A la hora de abordar el diseño de una base de datos distribuida podremos optar principalmente por dos tipos de estrategias: la estrategia ascendente y la estrategia descendente. Ambos tipos no son excluyentes, y no resultaría extraño a la hora de abordar un trabajo real de diseño de una base de datos que se pudiesen emplear en diferentes etapas del proyecto una u otra estrategia. La estrategia ascendente podría aplicarse en aquel caso donde haya que proceder a un diseño a partir de un número de pequeñas bases de datos existentes, con el fin de integrarlas en una sola. En este caso se partiría de los esquemas conceptuales locales y se trabajaría para llegar a conseguir el esquema conceptual global. Aunque este caso se pueda presentar con facilidad en la vida real, se prefiere pensar en el caso donde se parte de cero y se avanza en el desarrollo del trabajo siguiendo la estrategia descendente. La estrategia descendente (vea la figura 2) debería resultar familiar a la persona que posea conocimientos sobre el diseño de bases de datos, exceptuando la fase del diseño de la distribución. Pese a todo, se resumirán brevemente las etapas por las que se transcurre.

Todo comienza con un análisis de los requisitos que definirán el entorno del sistema en aras a obtener tanto los datos como las necesidades de procesamiento de todos los posibles usuarios del banco de datos. Igualmente, se deberán fijar los requisitos del sistema, los objetivos que debe cumplir respecto a unos grados de rendimiento, seguridad, disponibilidad y flexibilidad, sin olvidar el importante aspecto económico. Como puede observarse, los resultados de este último paso sirven de entrada para dos actividades que se realizan de forma paralela. El diseño de las vistas trata de definir las interfaces para el usuario final y, por otro lado, el diseño conceptual se encarga de examinar la empresa para determinar los tipos de entidades y establecer la relación entre ellas. Existe un vínculo entre el diseño de las vistas y el diseño conceptual. El diseño conceptual puede interpretarse como la integración de las vistas del usuario, este aspecto es de vital importancia ya que el modelo conceptual debería soportar no sólo las aplicaciones existentes, sino que debería estar preparado para futuras aplicaciones. En el diseño conceptual y de las vistas del usuario se especificarán las entidades de datos y se determinarán las aplicaciones que funcionarán sobre la base de datos, así mismo, se recopilarán datos estadísticos o estimaciones sobre la actividad de estas aplicaciones.

Dichas estimaciones deberían girar en torno a la frecuencia de acceso, por parte de una aplicación, a las distintas relaciones de las que hace uso, podría afinarse más anotando los atributos de la relación a la que accede. Desarrollado el trabajo hasta aquí, se puede abordar la confección del esquema conceptual global. Este esquema y la información relativa al acceso a los datos sirven de entrada al paso distintivo: el diseño de la distribución. El objetivo de esta etapa consiste en diseñar los esquemas conceptuales locales que se distribuirán a lo largo de todos los puestos del sistema distribuido. Sería posible tratar cada entidad como una unidad de distribución; en el caso del modelo relacional, cada entidad se corresponde con una relación.

Resulta bastante frecuente dividir cada relación en subrelaciones menores denominadas fragmentos que luego se ubican en uno u otro sitio. De ahí, que el proceso del diseño de la distribución conste de dos actividades fundamentales: la fragmentación y la asignación. El último paso del diseño de la distribución es el diseño físico, el cual proyecta los esquemas conceptuales locales sobre los dispositivos de almacenamiento físico disponibles en los distintos sitios. Las entradas para este paso son los esquemas conceptuales locales y la información de acceso a los fragmentos. Por último, se sabe que la actividad de desarrollo y diseño es un tipo

de proceso que necesita de una monitorización y un ajuste periódicos, para que si se llegan a producir desviaciones, se pueda retornar a alguna de las fases anteriores.

Diseño

Existen diversas formas de afrontar el problema del diseño de la distribución. Las más usuales se muestran en la figura 3. En el primer caso, caso A, los dos procesos fundamentales, la fragmentación y la asignación, se abordan de forma simultánea. Esta metodología se encuentra en desuso, sustituida por el enfoque en dos fases, caso B: la realización primeramente de la partición para luego asignar los fragmentos generados. El resto de los casos se comentan en la sección referente a los distintos tipos de la fragmentación.

Enfoques para realizar el diseño distributivo.

Antes de exponer las alternativas existentes de fragmentación, se desean presentar las ventajas e inconvenientes de esta técnica. Se ha comentado en la introducción la conveniencia de descomponer las relaciones de la base de datos en pequeños fragmentos, pero no se ha justificado el hecho ni se han aportado razones para efectuarlo. Por ello, desde este punto se va a intentar aportar las razones necesarias para llevar a cabo esa descomposición, esa fragmentación.

El principal problema de la fragmentación radica en encontrar la unidad apropiada de distribución. Una relación no es una buena unidad por muchas razones. Primero, las vistas de la aplicación normalmente son subconjuntos de relaciones. Además, la localidad de los accesos de las aplicaciones no está definida sobre relaciones enteras pero sí sobre subconjuntos de las mismas. Por ello, sería normal considerar como unidad de distribución a estos subconjuntos de relaciones.

Segundo, si las aplicaciones tienen vistas definidas sobre una determinada relación (considerándola ahora una unidad de distribución) que reside en varios sitios de la red, se puede optar por dos alternativas. Por un lado, la relación no estará replicada y se almacena en un único sitio, o existe réplica en todos o algunos de los sitios en los cuales reside la aplicación. Las consecuencias de esta estrategia son la generación de un volumen de accesos remotos innecesario. Además, se pueden realizar réplicas innecesarias que causen problemas en la ejecución de las actualizaciones y puede no ser deseable si el espacio de almacenamiento está limitado.

Tercero, la descomposición de una relación en fragmentos, tratados cada uno de ellos como una unidad de distribución, permite el proceso concurrente de las transacciones. También la relación de estas relaciones, normalmente, provocará la ejecución paralela de una consulta al dividirla en una serie de subconsultas que operará sobre los fragmentos.

Pero la fragmentación también acarrea inconvenientes. Si las aplicaciones tienen requisitos tales que prevengan la descomposición de la relación en fragmentos mutuamente exclusivos, estas aplicaciones cuyas vistas estén definidas sobre más de un fragmento pueden sufrir una degradación en el rendimiento. Por tanto, puede ser necesario recuperar los datos de dos fragmentos y llevar a cabo sobre ellos operación de unión y yunto , lo cual es costoso.

Un segundo problema se refiere al control semántico. Como resultado de la fragmentación los atributos implicados en una dependencia se descomponen en diferentes fragmentos los cuales pueden destinarse a sitios diferentes. En este caso, la sencilla tarea de verificar las dependencias puede resultar una tarea de búsqueda de los datos implicados en un gran número de sitios.

Tipos de fragmentación:

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).

Puede observarse como los casos C y D se basan en la mencionada generación de la rejilla, con la diferencia que en el primero de ellos se produce una fusión, una desfragmentación de las celdas, agrupándolas de la manera más adecuada para obtener mayor rendimiento, ya que los fragmentos generados son muy pequeños. En el segundo caso se asignan las celdas a los sitios y luego se realiza una rigurosa optimización de cada sitio. El caso E sería aquel en el que se utiliza la fragmentación VH o la fragmentación HV.

Enfoques para realizar el diseño distributivo.

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.

Distintos tipos de fragmentación.

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.v

Compleción. 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.

Reconstrucción. Si una relación R se descompone en una serie de fragmentos R1, R2, ..., Rn, puede definirse una operador relacional tal que el operador será diferente dependiendo de las diferentes formas de fragmentación. La reconstrucción de la relación a partir de sus fragmentos asegura la preservación de las restricciones definidas sobre los datos en forma de dependencias.

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.

Alternativas de asignación

Partiendo del supuesto que el banco de datos se haya fragmentado correctamente, habrá que decidir sobre la manera de asignar los fragmentos a los distintos sitios de la red. Cuando una serie de datos se asignan, éstos pueden replicarse para mantener una copia. Las razones para la réplica giran en torno a la seguridad y a la eficiencia de las consultas de lectura. Si existen muchas reproducciones de un elemento de datos, en caso de fallo en el sistema se podría acceder a esos datos ubicados en sitios distintos. Además, las consultas que acceden a los mismos datos pueden ejecutarse en paralelo, ya que habrá copias en diferentes sitios. Por otra parte, la ejecución de consultas de actualización, de escritura, implicaría la actualización de todas las copias que existan en la red, cuyo proceso puede resultar problemático y complicado. 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. Una base de datos fragmentada es aquella donde no existe réplica alguna. Los fragmentos se alojan en sitios donde únicamente existe una copia de cada uno de ellos a lo largo de toda la red. En caso de réplica, podemos considerar una base de datos totalmente replicada, donde existe una copia de todo el banco de datos en cada sitio, o considerar una base de datos parcialmente replicada donde existan copias de los fragmentos ubicados en diferentes sitios. El número de copias de un fragmento será una de las posibles entradas a los algoritmos de asignación, o una variable de decisión cuyo valor lo determine el algoritmo. La figura 5 compara las tres alternativas de réplica con respecto a distintas funciones de un sistema de base de datos distribuido.

	Réplica total	**Réplica parcial**	**Partición**
Procesamiento de consultas	fácil	dificultad	similar
Gestión del directorio	fácil o inexistente	dificultad	similar
Control de concurrencia	moderado	difícil	fácil
Seguridad	muy alta	alta	baja
\Realidad	posible aplicación	realista	posible aplicación

Información necesaria

Un aspecto importante en el diseño de la distribución es la cantidad de factores que contribuyen a un diseño óptimo. La organización lógica de la base de datos, la localización de las aplicaciones, las características de acceso de las aplicaciones a la base de datos y las características del sistema en cada sitio, tienen una decisiva influencia sobre la distribución. La información necesaria para el diseño de la distribución puede dividirse en cuatro categorías: la información del banco de datos, la información de la aplicación, la información sobre la red de ordenadores y la información sobre los ordenadores en sí. Las dos últimas son de carácter cuantitativo

y servirán, principalmente, para desarrollar el proceso de asignación. Se entrará en detalle sobre la información empleada cuando se aborden los distintos algoritmos de fragmentación y asignación.

Fragmentación Horizontal:

Como se ha explicada anteriormente, la fragmentación horizontal se realiza sobre las tuplas de la relación. Cada fragmento será un subconjunto de las tuplas de la relación. Existen dos variantes de la fragmentación horizontal: la primaria y la derivada. La fragmentación horizontal primaria de una relación se desarrolla empleando los predicados definidos en esa relación. Por el contrario, la fragmentación horizontal derivada consiste en dividir una relación partiendo de los predicados definidos sobre alguna otra.

Información necesaria para la fragmentación horizontal

Información sobre la base de datos.

Esta información implica al esquema conceptual global. Es importante señalar cómo las relaciones de la base de datos se conectan con otras. En una conexión de relaciones normalmente se denomina relación propietaria a aquella situada en la cola del enlace, mientras que se llama relación miembro a la ubicada en la cabecera del vínculo. Dicho de otra forma podemos pensar en relaciones de origen cuando nos refiramos a las propietarias y relaciones destino cuando lo hagamos con las miembro. Definiremos dos funciones: propietaria y miembro, las cuales proyectarán un conjunto de enlaces sobre un conjunto de relaciones. Además, dado un enlace, devolverán el miembro y el propietario de la relación, respectivamente. La información cuantitativa necesaria gira en torno a la cardinalidad de cada relación, notada como card(R).

Información sobre la aplicación.

Necesitaremos tanto información cualitativa como cuantitativa. La información cualitativa guiará la fragmentación, mientras que la cuantitativa se necesitará en los modelos de asignación. La principal información de carácter cualitativo son los predicados empleados en las consultas de usuario. Si no fuese posible investigar todas las aplicaciones para determinar estos predicados, al menos se deberían investigar las más importantes. Podemos pensar en la regla "80/20" para guiarnos en nuestro análisis, esta regla dice que el 20% de las consultas existentes acceden al 80% de los datos. Llegados a este punto, sería interesante determinar los predicados simples.

A parte de los predicados simples, las consultas emplean predicados más complejos resultado de combinaciones lógicas de los simples. Una combinación especialmente interesante es la conjunción de predicados simples, al predicado resultante se le denomina predicado mintérmino. Partiendo de que siempre es posible transformar una expresión lógica en su forma normal conjuntiva, usaremos los predicados mintérmino en los algoritmos para no causar ninguna pérdida de generalidad.

Sobre la información cuantitativa necesaria relativa a las aplicaciones, necesitaremos definir dos conjuntos de datos.

Selectividad mintérmino. Es el número de tuplas de una relación a las que accede una consulta de acuerdo a un predicado mintérmino dado. Por ejemplo, en el ejemplo anterior, la selectividad de m6 es 0 ya que no existe ninguna tupla que satisfaga las condiciones; en cambio, la selectividad de m1 es 2. Notaremos la selectividad de un mintérmino mi como sel(mi).

Frecuencia de acceso. Es la frecuencia con la que un usuario accede a los datos. Si Q = {q1, q2, ..., qq} es un conjunto de consultas de usuario, acc(qi) indica la frecuencia de acceso a la consulta qi en un periodo dado.

Fragmentación horizontal primaria

Antes de presentar un algoritmo formal que lleve a cabo la fragmentación horizontal, intentaremos explicar de manera intuitiva los procesos de fragmentación horizontal primaria y derivada. La fragmentación horizontal primaria se define como una operación de selección de las relaciones propietarias del esquema de la base de datos

Ahora definiremos la fragmentación horizontal más formalmente. Un fragmento horizontal Ri de una relación R contiene todas las tuplas de R que satisfacen un predicado mintérmino mi. Por tanto, dado un conjunto de predicados mintérmino M, existen tantos fragmentos horizontales de la relación R como predicados mintérmino. Este conjunto de fragmentos horizontales también se conocen como conjuntos de fragmentos mintérmino. En los párrafos siguientes se asumirá que la definición de fragmentos horizontales se basa en los predicados mintérmino. Además, el primer paso para el algoritmo de fragmentación consiste en establecer un conjunto de predicados con ciertas propiedades.

Un aspecto importante de los predicados simples es su compleción, así como su minimalidad. Un conjunto de predicados simples Pr se dice que es completo si y solo si existe una probabilidad idéntica de acceder por cada aplicación a cualquier par de tuplas pertenecientes a cualquier fragmento mintérmino que se define de acuerdo con Pr. Se puede apreciar como la definición de compleción de un conjunto de predicados simples difiere de la regla de compleción de la fragmentación.

El segundo paso en el proceso de fragmentación primaria consiste en derivar el conjunto de predicados mintérmino que pueden definirse sobre los predicados del conjunto Pr'. Estos predicados mintérmino establecen los fragmentos candidatos para el proceso de asignación. El establecimiento de los predicados mintérmino es trivial; la dificultad radica en el tamaño del conjunto de predicados mintérmino, que puede ser muy grande (de hecho, exponencial respecto al número de predicados simples). En el paso siguiente se presentarán formas de reducir el número de predicados mintérmino necesarios para la fragmentación.

El tercer paso aborda, como ya se ha citado, la eliminación de algunos fragmentos mintérmino que puedan ser redundantes. Esta eliminación se desarrolla identificando aquellos mintérminos que puedan resultar contradictorios sobre un conjunto de implicaciones.

Fragmentación horizontal derivada

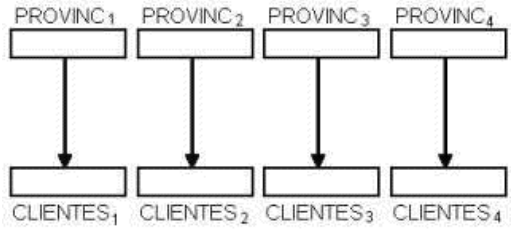
Una fragmentación horizontal derivada se define sobre una relación miembro de acuerdo a una operación de selección especificada sobre su propietaria. Se deben dejar claros dos puntos. Primero, el enlace entre las relaciones propietaria y miembro se define como un equi−yunto. Segundo, un equi−yunto puede desarrollarse a través de semiyuntos. Este segundo punto es especialmente importante para nuestros propósitos, ya que deseamos fraccionar una relación miembro según la fragmentación de su propietaria, además es necesario que el fragmento resultante se defina únicamente sobre los atributos de la relación miembro.

Las tres entradas necesarias para desarrollar la fragmentación horizontal derivada son las siguientes: el conjunto de particiones de la relación propietaria, la relación miembro y el conjunto se predicados resultados de aplicar el semi−yunto entre la propietaria y la miembro. El algoritmo de fragmentación resulta tan trivial que no se ve la necesidad de entrar en detalles.

Existe una posible complicación que necesita nuestro estudio. En un esquema de base de datos, resulta frecuente que existan más de dos enlaces sobre una relación R. En este caso, aparecen más de una posibilidad de fragmentación horizontal derivada. La decisión para elegir una u otra se basa en dos criterios: Uno, la fragmentación con mejores características de yunto. Dos, la fragmentación empleada en más aplicaciones.

Discutamos el segundo criterio primero. Resulta sencillo de establecer si tomamos en consideración la frecuencia con la que cada aplicación accede a los datos. Si es posible, deberíamos intentar facilitar el acceso a los usuarios que hagan mayor uso de los datos para, de esta manera, minimizar el impacto total del

rendimiento del sistema.



Grafo de yuntos entre fragmentos.

El primer criterio, sin embargo, no es tan sencillo. Considere, por ejemplo, la fragmentación expuesta en el ejemplo 8. El objetivo de esta fragmentación consiste en beneficiar a la consulta que haga uso de las dos relaciones al poder realizarse el yunto de CLIENTES y PROVINC sobre relaciones más pequeñas (es decir, fragmentos), y posibilitar la confección de yuntos de manera distribuida. El primer aspecto resulta obvio. Los fragmentos de CLIENTES son más pequeños que la propia relación CLIENTES. Por tanto, resultará más rápido llevar a cabo el yunto de un fragmento de PROVINC con otro de CLIENTES que trabajar con las propias relaciones. El segundo punto, sin embargo, es más importante ya que es la esencia de las bases de datos distribuidas. Si, además de estar ejecutando un número de consultas en diferentes sitios, podemos ejecutar una consulta en paralelo, se espera que el tiempo de respuesta del sistema aumente. En el caso de yuntos, esto es posible bajo ciertas circunstancias. Considere, por ejemplo, el grafo de yunto (los enlaces) entre los fragmentos de CLIENTES y la derivada PROVINC. Hay únicamente un enlace entrando o saliendo de un fragmento. De ahí, que se denomine a este grafo, grafo simple. La ventaja de este diseño donde la relación de yunto entre los fragmentos es simple, radica en la asignación a un sitio tanto de la propietaria como de la miembro y los yuntos entre pares diferentes de fragmentos pueden realizarse independientemente y en paralelo. Desgraciadamente, la obtención de grafos de yunto simples no siempre es posible. En tal caso, la mejor alternativa sería realizar un diseño que provoque un grafo de yuntos fragmentados. Un grafo fragmentado consiste en dos o más subgrafos que no están enlazados entre ellos. Por tanto, los fragmentos que se obtengan no se distribuirán para ejecuciones paralelas de un modo tan fácil como aquellos obtenidos a través de grafos simples, pero su asignación aún será posible.

Procederemos ahora a probar la corrección de los algoritmos presentados con respecto a los tres criterios enunciados páginas atrás.

Compleción. La compleción de una fragmentación horizontal primaria se basa en la selección de los predicados a usar. En la medida que los predicados seleccionados sean completos, se garantizará que el resultado de la fragmentación también lo será. Partiendo de la base que el algoritmo de fragmentación es un conjunto de predicados completos y mínimos Pr', se garantiza la compleción siempre que no aparezcan errores al realizar la definición de Pr'. La compleción de una fragmentación horizontal derivada es algo más difícil de definir. La dificultad viene dada por el hecho de que los predicados que intervienen en la fragmentación forman parte de dos relaciones. Definamos la regla de compleción formalmente. Sea R la relación miembro de un enlace cuya propietaria es la relación S, la cual está fragmentada como FS = {S1, S2, ..., Sw}. Además, sea A el atributo de yunto entre R y S. Entonces para cada tupla t de R, existirá una tupla t' de S tal que t[A] = t'[A].

Reconstrucción. La reconstrucción de una relación global a partir de sus fragmentos se desarrolla con el operador de unión tanto para la fragmentación horizontal primaria como para la derivada.

Disyunción. Resulta sencillo establecer la disyunción de la fragmentación tanto para la primaria como para la derivada. En el primer caso, la disyunción se garantiza en la medida en que los predicados mintérmino que determinan la fragmentación son mutuamente exclusivos. En la fragmentación derivada, sin embargo, implica un semi−yunto que añade complejidad al asunto. La disyunción puede garantizarse si el grafo de yunto es

simple. Si no es simple, será necesario investigar los valores de las tuplas. En general, no se desea juntar una tupla de una relación miembro con dos o más tuplas de una relación propietaria cuando estas tuplas se encuentran en fragmentos diferentes a los de la propietaria. Esto no es fácil de establecer, e ilustra por qué los esquemas de la fragmentación derivada que generan un grafo de yunto simple son siempre más atractivos.

Fragmentación Vertical:

Recuérdese que la fragmentación vertical de una relación R produce una serie de fragmentos R1, R2, ..., Rr, cada uno de los cuales contiene un subconjunto de los atributos de R así como la clave primaria de R. El objetivo de la fragmentación vertical consiste en dividir la relación en un conjunto de relaciones más pequeñas tal que algunas de las aplicaciones de usuario sólo hagan uso de un fragmento. Sobre este marco, una fragmentación óptima es aquella que produce un esquema de división que minimiza el tiempo de ejecución de las aplicaciones que emplean esos fragmentos.

La partición vertical resulta más complicada que la horizontal. Esto se debe al aumento del número total de alternativas que tenemos disponibles. Por ejemplo, en la partición horizontal, si el número total de predicados simples de Pr es n, existen 2n predicados mintérminos posibles que puedan definirse. Además, sabemos que algunos de estos predicados resultarán contradictorios con algunas de las aplicaciones existentes, por lo que podremos reducir el número inicial. En el caso vertical, si una relación tiene m atributos clave no primarios, el número de posibles fragmentos es igual a B(m), es decir el m−ésimo número de Bell [3]. Para valores grandes de m, B(m) mm; por ejemplo, para m = 10, B(m) 115.000, para m = 15, B(m) 109, para m = 30, B(m) = 1023.

Estos valores indican que la obtención de una solución óptima de la fragmentación vertical resultará una tarea inútil, sino nos apoyamos en el uso de heurísticos. Existen dos enfoques heurísticos para la fragmentación vertical de relaciones:

Agrupación. Comienza asignando cada atributo a un fragmento, y en cada paso, junta algunos de los fragmentos hasta que satisface un determinado criterio. La agrupación sugirió en principio para bases de datos centralizadas y se usó posteriormente para las bases de datos distribuidas.

Escisión. A partir de la relación se deciden que fragmentos resultan mejores, basándose en las características de acceso de las aplicaciones a los atributos. Esta técnica se presentó, también, para bases de datos centralizadas. Posteriormente, se extendió al entorno distribuido.

En este documento se tratará únicamente la técnica de escisión, ya que es más apropiada para la estrategia descendente y porque resulta más probable encontrar la solución para la relación entera que a partir de un conjunto de fragmentos con un único atributo. Además, la escisión genera fragmentos no solapados mientras que la agrupación normalmente produce fragmentos solapados. Dentro del contexto de los sistemas de bases de datos distribuidos, son preferibles los fragmentos no solapados por razones obvias. Evidentemente, los fragmentos no solapados se refieren únicamente a atributos clave no primarios.

Antes de comenzar, vamos a aclarar un problema: la réplica de las claves de la relación en los fragmentos. Esta es una característica de la fragmentación vertical que permite la reconstrucción de la relación global. Por tanto, la escisión considera únicamente aquellos atributos que no son parte de la clave primaria.

La réplica de los atributos clave supone una gran ventaja, a pesar de los problemas que pueda causar. La ventaja está relacionada con el esfuerzo para mantener la integridad semántica. Tenga en cuenta que cada dependencia (funcional, multivaluada ...) es, de hecho, una restricción que influye sobre el valor de los atributos de las respectivas relaciones en todo momento. También muchas de estas dependencias implican a los atributos clave de una relación. Si queremos diseñar una base de datos tal que los atributos clave sean parte de una fragmento que está ubicado en un sitio, y los atributos relacionados sean parte de otro fragmento asignado a un segundo sitio, cada petición de actualización provocará la verificación de integridad que

necesitará de una comunicación entre esos sitios. La réplica de los atributos clave de cada fragmento reduce esta problemática, pero no elimina toda su complejidad, ya que la comunicación puede ser necesaria para las restricciones de integridad que implican a las claves primarias, así como para el control de concurrencia.

Una posible alternativa a la réplica de los atributos clave es el empleo de identificadores de tuplas, que son valores únicos asignados por el sistema a las tuplas de una relación. Mientras el sistema mantenga los identificadores, los fragmentos permanecerán disjuntos.

Información necesaria para la fragmentación vertical

La principal información que necesitaremos se referirá a las aplicaciones. Por tanto, este punto tratará de especificar la información que de una aplicación que funciona sobre la base de datos podamos extraer. Teniendo en cuenta que la fragmentación vertical coloca en un fragmento aquellos atributos a los que se accede de manera simultánea, necesitaremos alguna medida que defina con más precisión el concepto de simultaneidad. Esta medida es la afinidad de los atributos, que indica la relación estrecha existente entre los atributos. Desgraciadamente, no es muy realista esperar que el diseñador o los usuarios puedan especificar estos valores. Por ello, presentaremos una forma por la cual obtengamos esos valores partiendo de datos más básicos.

El principal dato necesario relativo a las aplicaciones es la frecuencia de acceso. Sea Q = {q1, q2, ..., qq} el conjunto de consultas de usuario (aplicaciones) que funcionan sobre una relación R(A1, A2, ..., An).

Los vectores uso pueden definirse muy fácilmente para cada aplicación siempre que el diseñador conozca las aplicaciones existentes en el sistema. La regla 80/20 expuesta páginas atrás podría resultar útil para el desarrollo de esta tarea.

Los valores del uso de los atributos en general no son suficientes para desarrollar la base de la escisión y la fragmentación de los atributos, ya que estos valores no representan el peso de las frecuencias de la aplicación. La dimensión de esta frecuencia puede incluirse en la definición de la medida de los atributos afines afd(Ai, Aj), la cual mide el límite entre dos atributos de una relación de acuerdo a cómo las aplicaciones acceden a ellos.

Fragmentación mixta o híbrida:

En muchos casos la fragmentación vertical u horizontal del esquema de la base de datos no será suficiente para satisfacer los requisitos de las aplicaciones. Como ya se citó al comienzo de este documento podemos combinar ambas, utilizando por ello la denominada fragmentación mixta. Cuando al proceso de fragmentación vertical le sigue una horizontal, es decir, se fragmentan horizontalmente los fragmentos verticales resultantes, se habla de la fragmentación mixta HV. En el caso contrario, estaremos ante una fragmentación VH. Una característica común a ambas es la generación de árboles que representan la estructura de fragmentación.

Considere, por ejemplo, la relación PROVINC. Recordará que se le aplicó una fragmentación horizontal de acuerdo al valor del atributo CCODZONA resultando cuatro fragmentos horizontales. Podríamos pensar en aplicarle una nueva fragmentación de carácter vertical. Entonces resultarían cuatro fragmentos horizontales divididos, por ejemplo, en dos fragmentos verticales. En este caso el número total de fragmentos ascendería, lógicamente, a ocho.

Estructura arbórea de fragmentación mixta.

No se desea entrar en excesivos detalles sobre las reglas y condiciones para efectuar la fragmentación mixta. Entre otras razones porque, tanto a la fragmentación HV como la fragmentación VH, se le pueden aplicar los mismos criterios y reglas que a la fragmentación horizontal y vertical. Es decir, volviendo al ejemplo anterior,

al cual le practicamos la fragmentación HV, al realizar la fragmentación horizontal tal como se ha expuesto, lo que se obtienen no son más que subrelaciones, la unión de las cuales da lugar a la relación PROVINC. Por tanto, para fragmentar cada subrelación sería perfectamente viable aplicarle el método de fragmentación vertical que se ha desarrollado. Como, en este caso, se han querido generar dos fragmentos verticales por cada uno horizontal, simplemente deberíamos confeccionar la matriz de grupos afines (a través del algoritmo BEA) para cada fragmento horizontal y aplicarle, posteriormente, el algoritmo de fragmentación binaria PARTICIÓN.

También debe tenerse en cuenta el número de niveles arbóreos que se generen, es decir, nadie impide que tras realizar una fragmentación VH, podamos aplicar a los fragmentos resultantes una nueva fragmentación vertical, y a estos últimos una nueva fragmentación horizontal, etc. Dicho número puede ser grande, pero también será ciertamente finito. En el caso horizontal, el nivel máximo de profundidad se alcanzará cuando cada fragmento albergue una única tupla, mientras que en el caso vertical el final llegará cuando cada fragmento contenga un único atributo. Sin embargo, aunque no deba tomarse como dogma, el número de niveles no debería superar el par (VH y HV). El porqué de esta afirmación es bien sencillo, piense, por ejemplo, en el coste que supondría realizar la unión o el yunto de una relación con fragmentación nivel 7. Evidentemente, el coste sería muy elevado y ese aumento de rendimiento que se persigue al aplicar estas técnicas, quizás, no se produzca.

Antes de pasar a estudiar el problema de la asignación se desea comentar la técnica de fragmentación mixta basada en celdas [2]. Esta técnica se basa en la generación de celdas de rejilla. Qué es una celda de rejilla, podríamos definirla como un fragmento horizontal y vertical simultáneo. La técnica aplica un algoritmo de fragmentación vertical y otro horizontal de manera concurrente sobre la relación. Los algoritmos realizan una fragmentación máxima, es decir, se persigue que en cada celda únicamente haya un atributo y una tupla. Quizá el lector pueda encontrar el método contradictorio con lo citado anteriormente respecto a la eficiencia, dada la gran cantidad de fragmentos generados, el número es, efectivamente, el máximo. Sin embargo, este sólo es el primer paso del proceso. Una vez generadas las celdas se aplica un método para optimizar la rejilla mediante fusión o desfragmentación, de acuerdo, fundamentalmente, a las aplicaciones que actúen sobre esos fragmentos. El método, por tanto, persigue una fragmentación los más específica posible acorde con las aplicaciones y los sitios existentes en la red.

Información necesaria

En esta etapa de la asignación, necesitaremos datos cuantitativos sobre la base de datos, las aplicaciones que funcionan sobre ella, la red de comunicaciones, las características de proceso, y el límite de almacenamiento de cada sitio de la red. Procederemos a discutirlos en detalle.

Información de la base de datos. Para desarrollar la fragmentación horizontal, definimos la selectividad de los mintérminos. Ahora, necesitamos extender esta definición a los fragmentos y definir la selectividad de un fragmento Fj con respecto a una consulta qi. Es el número de tuplas de Fj a las que se necesita acceder para procesar qi. Este valor lo notaremos como seli(Fj). Otro elemento informativo de los fragmentos de la base de datos es su tamaño. El tamaño de un fragmento Fj viene dado por tamaño(Fj) = card(Fj)*long(Fj), donde long(Fj) es la longitud (en octetos) de una tupla del fragmento Fj.

Información de los sitios. Sobre cada ordenador necesitamos conocer sus capacidades de procesamiento y almacenamiento. Obviamente, estos valores pueden calcularse a través de funciones elaboradas o por simples estimaciones. La unidad de coste de almacenar datos en el sitio Sk será denotada como UCAk. Así mismo, especificaremos como medida de coste UPTk al coste de procesar una unidad de trabajo en el sitio Sk. La unidad de trabajo debería ser idéntica a aquella utilizada en las medidas RR y UR.

Información sobre la red. En nuestro modelo asumiremos la existencia de una red simple donde el coste de comunicaciones se define respecto a una trama de datos. Entonces gij nota el coste de comunicación por trama

entre los sitios Si y Sj. Para permitir el cálculo del número de mensajes, usaremos ftamaño como el tamaño (en octetos) de una trama. Es evidente que existen modelos de red mucho más elaborados que toman en cuenta las capacidades del canal, las distancias entre sitios, las características del protocolo, etc. Sin embargo, se cree que la derivación de estas ecuaciones se sale fuera de este documento.

Procesamiento distribuido de consultas

Existen varios medios para calcular la respuesta a una consulta. En el caso de sistemas centralizado, el criterio principal para determinar el costo de una estrategia especifica es el número de accesos al disco. En un sistema distribuido es preciso tener en cuenta otros factores, como son:

El costo de transmisión de datos en la red.

El beneficio potencial que supondría en la ejecución el que varias localidades procesaran en paralelo partes de la consulta.

El costo relativo de la transferencia de datos en la red y la transferencia de datos entre la memoria y el disco varia en forma considerable, dependiendo del tipo de red y de la velocidad de los discos. Por tanto, en un caso general, no podemos tener en cuenta solo los costos del disco o los de la red. es necesario llegar a un equilibrio adecuado entre los dos.

Repetición y fragmentación

Considere una consulta muy sencilla: encontrar todas las tuplas de la relación depósito. Aunque la consulta es muy simple, de hecho es trivial; su procesamiento no es trivial, ya que es posible que la relación depósito esté fragmentada, repetido o las dos cosas, como ya se vio. Si la relación deposito está repetida, es preciso decidir qué copia se va a utilizar. Si ninguna de las copias está fragmentada, se elige la copia que implique costos de transmisión más reducidos. Pero si una copia está fragmentada, la elección no se tan sencilla, ya que es preciso calcular varios productos o uniones para reconstruir la relación depósito. En tal caso, el número de estrategias para este ejemplo sencillo puede ser grande. De hecho, la elección de una estrategia puede ser una tarea tan compleja como hacer una consulta arbitraria.

Procesamiento de intersección simple

Considere la expresión en álgebra relacional:

cliente x deposito x sucursal

Suponemos que ninguna de las tres relaciones está repetida o fragmentada y que cliente está almacenada en la localidad Lc, deposito en la Ld y sucursal en la Lb. Sea Li la localidad donde se originó la consulta. El sistema debe producir el resultado en la localidad Li. Entre las posibles estrategias para procesar esta consulta se encuentran las siguientes:

Enviar copias de las tres relaciones a la localidad Li. Al emplear las técnicas de procesamiento de consulta, escoger una estrategia para procesar en forma local la consulta completa en Li.

Enviar una copia de la relación cliente a la localidad Li y calcular cliente x depósito de Ld. Enviar cliente x depósito de Ld a Lb, donde se calcula (cliente x depósito) x sucursal. El resultado de esta operación es enviado a Li.

Pueden elaborarse estrategias similares a la anterior al intercambiar los papeles de Lc, Ld y Lb.

No puede garantizarse que una estrategia sea la mejor en todos los casos. entre los factores que deben tomarse en cuenta están la cantidad de datos que debe transmitirse, el costo de transmitir un bloque de datos entre dos localidades determinadas y la velocidad de procesamiento relativa en cada localidad.

Estrategias de intersección utilizando el paralelismo

Considere un producto de cuatro relaciones:

r1 x r2 r3 r4

donde la relación r1 está almacenada en la localidad Li. Suponemos que el resultado ha de presentarse en la localidad Li. Existen, por supuesto muchas estrategias que se pueden considerar. Un método atractivo sería utilizar la estrategia de intersección encauzada. Por ejemplo, se puede enviar r1 a L2 y calcular r1 x r2 en L2. al mismo tiempo se puede enviar r3 a L4 y calcular r3 x r4 en L4.

La localidad L2 puede enviar tuplas de (r1 x r2) a Li conforme se vayan produciendo, en vez de esperar a que se calcule el producto completo. De forma similar, L4 pude enviar tuplas de (r3 x r4) a Li. Una vez que las tuplas de (r1 x r2) y (r3 x r4) lleguen a Li, esta localidad podrá empezar el cálculo de (r1 x r2) x (r3 x r4) en paralelo con el cálculo de (r1 x r2) en L2 y de (r3 x r4) en L4.

Estrategia de semintersección

Suponer que deseamos calcular la expresión r1 x r2, donde r1 y r2 están almacenados en las localidades L1 y L2 respectivamente. Sean R1 y R2 los esquemas de r1 y r2. Suponer que queremos obtener el resultado en L1. Si hay muchas tuplas de r2 que no interseccionan con ninguna de r1, entonces el envío de r2 a S1 requiere el envío de tuplas que no contribuyen al resultado. Es conveniente borrar tales tuplas antes de enviar los datos a L1, particularmente si los costos de la red son muy elevados.

Para hacerlo vemos la siguiente estrategia:

Calcular temp1 (r1 (r2 (r1) en L1.

enviar temp1 de L1 a L2.

Calcular temp2 r2 x temp1 en L2.

Enviar temp2 de L2 a L1.

Calcular r1 x temp2 en L1.

La estrategia anterior es ventajosa particularmente cuando en el producto participan relativamente pocas tuplas de r2. Es probable que suceda esta situación si r1 es el resultado de una expresión de álgebra relacional que contenga la selección.

Esta estrategia es conocida como una estrategia de semiproducto, después del operador de semiproducto, indicado por x, de álgebra relacional.

<u>Conclusiones y consideraciones:</u>

A lo largo de este documento se ha intentado dar una visión global y genérica de los problemas y características que contiene el diseño de una base de datos. Se ha hecho especial hincapié en las técnicas de fragmentación horizontal y vertical a través de métodos y algoritmos muy frecuentes en la literatura referida al

tema. Se espera que el lector no haya tenido demasiados problemas para su comprensión, las técnicas son sencillas y se ha procurado incluir distintos ejemplos para facilitar el entendimiento. Igualmente, la puesta en práctica de los algoritmos, es decir, su codificación, no es un proceso complicado si se poseen nociones en el desarrollo de algoritmos. Piense, por ejemplo, que los dos algoritmos de partición vertical presentados, no hacen más que manipular matrices.

También debería tenerse presente la existencia de enfoques de fragmentación distintos y, posiblemente, más complejos, pero se debe pensar que más eficientes. Sean, por ejemplo, las técnicas de fragmentación vertical basadas en grafos, como el algoritmo de Navathe y Ra que genera en un solo paso fragmentos verticales. Además, están apareciendo métodos de fragmentación mixta como el que se ha comentado. Si bien, estos métodos son enfoques formales más que prácticos, desarrollados por insignes investigadores en universidades, por tanto, lejos todavía de su desarrollo comercial.

Pese a la aparición de los métodos de bases de datos distribuidas hace ya años, parece que el salto de lo centralizado a lo distribuido a escala comercial está por venir. Todavía no se ha extendido suficientemente el esquema distribuido, pero se espera que próximamente se produzca el avance definitivo. Considere los dos componentes básicos de los sistemas de bases de datos distribuidos (la propia base de datos y la red de ordenadores) y piense en la situación actual de la informática. Si las bases de datos es una de las ramas más antiguas e importantes de la informática, muchas empresas compran ordenadores para dedicarlos exclusivamente a la gestión de sus datos (pienso que, prácticamente, en el 100% de las PYMES se produce este hecho) y, como parece ser que se ha asumido por parte de todo tipo de empresarios los beneficios que acarrea la conexión de los ordenadores, la instalación de una red, se puede concluir diciendo que el terreno ya está abonado para su extensión comercial. Sólo falta que determinadas multinacionales decidan apostar más fuerte por este enfoque a través de sus famosos sistemas gestores de bases de datos y que se produzca la consolidación de la resolución de los problemas que el enfoque distribuido acarrea.