BASES DE DATOS DISTRIBUIDAS BY BELZEBU

En esta pequeña obra encontraras toda la teoría sobre Bases de datos distribuidas, encontraras al final de cada capitulo algunos ejercicios resueltos. Además un apéndice que muestra paso a paso como se implementa una base distribuida en Oracle 8i.

1

Conceptos básicos sobre Sistemas Distribuidos

Al tratar sobre bases de datos distribuidas conviene conceptuar algunos elementos sobre sistemas distribuidos. Al hablar de distribución, no sólo se puede aplicar a los datos sino a otros aspectos como los procesos, los programas o los sistemas operativos.

Un sistema distribuido es un conjunto de sitios comunicados entre sí por una red para cooperar en la ejecución de aplicaciones. Un sitio es un equipo con capacidad de procesamiento y almacenamiento de datos.

1.1 Características de los sistemas distribuidos

Los sistemas distribuidos tienen, en general, las siguientes propiedades:

1. Múltiples recursos lógicos, es decir, los datos se almacenan en múltiples sitios, lo cual permite mayor flexibilidad y mayor disponibilidad.
2. Arquitectura física modular, lo que quiere decir que los equipos se encuentran dispersos pero permitiendo extensibilidad de una manera modular y fácil reconfiguración.
3. Necesidad de interacción entre sitios, lo cual le da al sistema unicidad.
4. Interacción por intercambio de mensajes en modo cooperativo pero conservando la autonomía de cada sitio. Esto es típico en todos los sistemas montados sobre redes y que permiten alguna interacción.
5. Control global distribuido y unificado a la vez: Múltiples programas en diferentes equipos, pero con unas reglas comunes, generando como consecuencia una mayor consistencia de todo el sistema.

Un sistema distribuido puede estar configurado en alguna de las siguientes maneras:

- ✓ Centralizada
- ✓ Bicentral
- ✓ Multicentro
- ✓ Horizontal
- ✓ Multinivel

1.2 Impacto en las organizaciones

Los sistemas distribuidos han influenciado fuertemente las organizaciones en aspectos como los siguientes:

1. Los departamentos de usuarios adquieren mayor poder computacional.
2. Los departamentos de usuarios se vuelven más responsables de sus datos y de sus procesos.
3. La descentralización de los procesos funcionales de información adquieren una mayor dimensión.
4. Crece la centralización de los procesos estratégicos de información.
5. El procesamiento de la organización se extiende geográficamente.
6. Los gerentes enfrentan problemas políticos pero también pueden detectar nuevas oportunidades.

1.3 Beneficios de los sistemas distribuidos

Resulta notorio hablar de sistemas distribuidos en las organizaciones modernas debido a las ventajas que esto representa. Entre ellas se pueden enumerar:

1. La descentralización evita la sobrecarga en un sitio y elimina la dependencia del mismo.
2. Mejor tiempo de respuesta, gracias al procesamiento local en cada uno de los nodos sobre los que se distribuye el sistema. Esto incrementa el rendimiento a través de un alto grado de paralelismo.
3. Sobrecarga de comunicación reducida, con la maximización de la localidad de las aplicaciones.
4. Reducción de costos, por la minimización del uso de canal y por el hecho de que es más rentable adquirir múltiples equipos que un gran computador central (por lo menos esto es válido en el caso de los mainframes).
5. La extensibilidad: En el hardware, al adicionar una estación a la red fácilmente. En la base de datos, al permitir interconectar las bases de datos pre-existentes, conformando una base de datos distribuida. La heterogeneidad es representativa aquí ya que se rompen dependencias con un proveedor, permitiendo que el sistema se extienda con productos diferentes.

6. Disponibilidad: Esta se logra precisamente por la redundancia del hardware, del software y de los datos y también por la dispersión de recursos.

Por otra parte, como elemento motivador importante, la evolución de las necesidades del usuario presiona el acercamiento de los datos al usuario. A su vez, la descentralización de los datos y el procesamiento se va reflejando en la estructura organizacional.

1.4 Problemas de los sistemas distribuidos

No todo resulta beneficioso con los sistemas distribuidos. También aparecen problemas como los siguientes:

1. El control del acceso a datos dispersos se vuelve más inmanejable.
2. La coordinación de procesos concurrentes y remotos resulta más compleja.
3. El montaje de sistemas distribuidos en ambientes heterogéneos no está completamente resuelto. A nivel de protocolos está resuelto pero los niveles aplicativos siguen presentando problemas.
4. Asegurar el desarrollo conjunto de software es un inconveniente que tiene que ver con la disciplina, la heterogeneidad de los grupos de trabajo y la misma idiosincrasia y costumbres de las personas.
5. Ningún sitio puede observar el estado global de todo el sistema, por lo cual se corre con el riesgo de tomar decisiones inconsistentes en el procesamiento.
6. Cuando hay una inadecuada migración hacia los sistemas distribuidos se pueden generar problemas adicionales como los siguientes:
    a. Pérdida del control gerencial
    b. Pérdida del control en el manejo de los sistemas de información
    c. Suboptimización
    d. Incompetencia distribuida, ya que cada usuario desarrolla y lo que se refleja a nivel global es la falta de integración y aún de profesionalismo
    e. Fallas en el aprovechamiento del DBMS, ya que los usuarios implementan archivos locales en otras herramientas

f. Duplicación de esfuerzo y de personal ya que varios ingenieros en diferentes sitios pueden desarrollar lo mismo
g. Problemas de mantenimiento

1.5 Introducción a bases de datos distribuidas

Hay dos carácterísticas importantes que deben tenerse en cuenta en una base de datos distribuida:

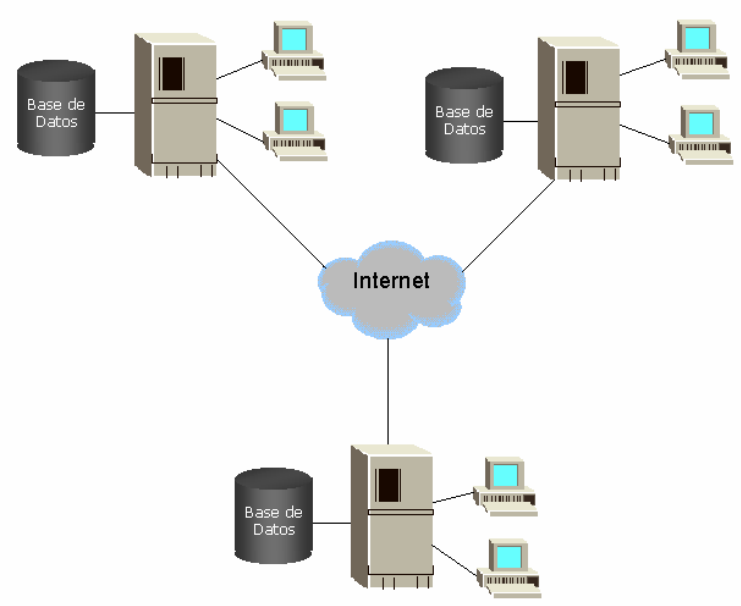
1. Distribución: Los datos no están residentes en el mismo sitio (procesador). Esto permite distinguir una base de datos distribuida de una base de datos centralizada.

2. Correlación lógica: Los datos tienen algunas propiedades que los relacionan, de tal manera que se puede distinguir una base de datos distribuida de un conjunto de bases de datos locales o archivos residentes en diferentes sitios de una red de computadores.

Las definiciones anteriores no son suficientes para distinguir siempre si una base de datos es distribuida o no lo es.

Ejemplo 1

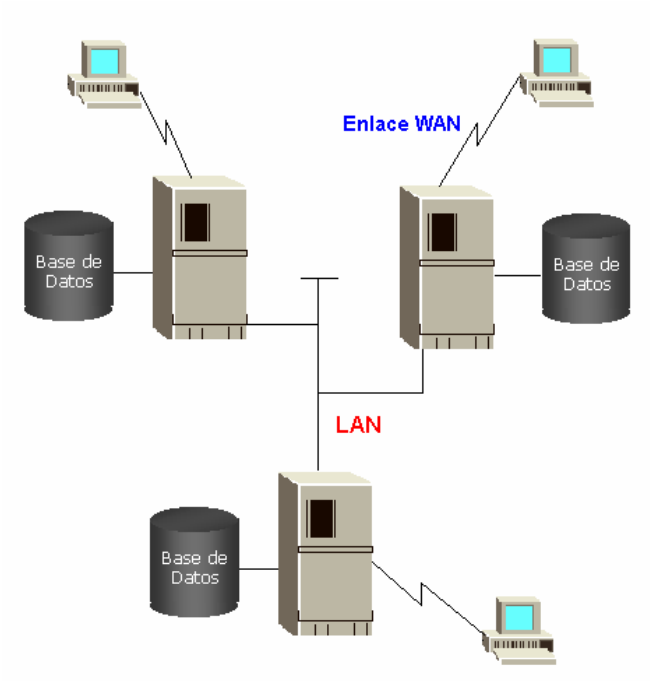
Considere un banco con tres áreas en diferentes sitios. En cada área se encuentra un servidor con sus estaciones, las terminales de cajeros de cada y la base de datos del área. Cada computador con su base de datos local en un área constituye un sitio. Los servidores están conectados a través de una red WAN (Internet).



Surge una idea: Que las aplicaciones que accesen datos en más de un área pueden hacerlo con Bases de Datos Globales o Distribuidas. Por ejemplo, una transferencia de fondos

Ejemplo 2

Suponga el mismo el mismo banco y las mismas aplicaciones. Los computadores han sido alejados a un edificio común y conectados en LAN. Las terminales de cajero de las áreas están conectadas a sus respectivos computadores por líneas telefónicas conmutadas pero las características de la arquitectura permanecen iguales.

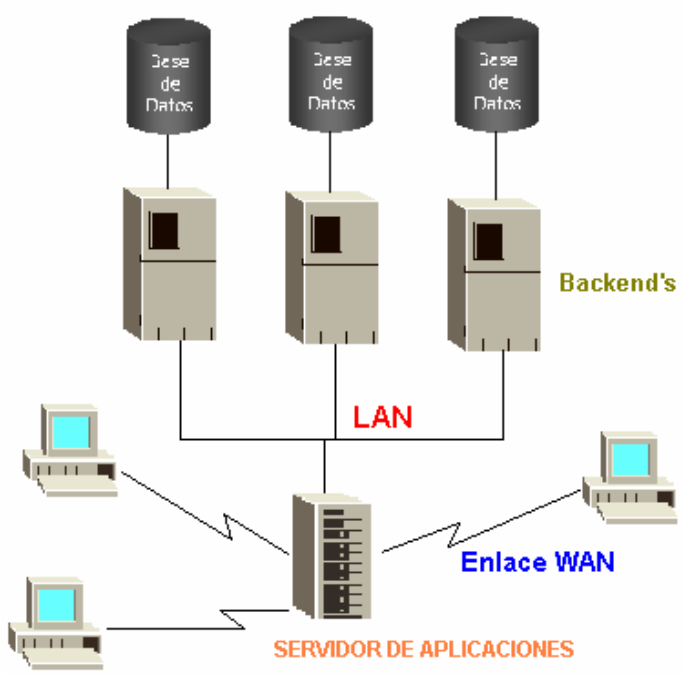


Esta también puede ser una base de datos distribuida, sólo que puede ser más rápida y confiable si los procesamientos son distribuidos y hay una alta interacción entre los sitios.

Ejemplo 3

Suponga el mismo banco. Los datos de las diferentes áreas están distribuidas en tres servidores. Los programas de aplicación son ejecutados por otro computador.

El caso señalado permite establecer que ninguno de los computadores es capaz de ejecutas una aplicación por sí solo, es decir, no hay aplicaciones locales, lo que implica que no es una base de datos distribuida.



1.6 Definición de Base de Datos Distribuida

Una base de datos distribuida es una colección de datos distribuidos en diferentes computadores de una red, donde cada sitio de la red tiene capacidad de procesamiento autónomo y puede ejecutar aplicaciones locales.

Además cada sitio participa en la ejecución de al menos una aplicación global que requiera acceder datos en diferentes sitios, usando un subsistema de comunicación.

Lo relevante es subrayar *la cooperación entre sitios autónomos*.

Una base de datos distribuida puede ser un ambiente en el cual los datos en dos o más bases de datos son accesados como si estuvieran en una base de datos simple. Este acceso puede ser de consulta o puede permitir actualización a una o varias bases de datos.

1.7 La Centralización y la Descentralización

Los costos en los sistemas centralizados

- ✓ Los sistemas centralizados, por razones de economía de escala, son propios de aplicaciones de mucha memoria.
- ✓ Los sistemas centralizados son propios de bases de datos que requieren mínima redundancia de datos.
- ✓ Los sistemas centralizados se requieren cuando hay que invertir menos en recursos humanos.
- ✓ La planeación centralizada minimiza los costos.
- ✓ Al hacer un mejor uso de los recursos, por el hecho de estar en un sitio, los costos tienden a ser menores que en los sistemas distribuidos.

Los costos en los sistemas distribuidos

- ✓ Los sistemas distribuidos tienen un menor costo en las comunicaciones.
- ✓ Hay un mejor uso de los recursos tecnológicos, por el aprovechamiento de los equipos existentes en los sitios.

Los aspectos políticos en los sistemas centralizados

- ✓ Los departamentos de sistemas retienen el control de toda la organización.
- ✓ La centralización de datos da más poder a la gerencia.
- ✓ La planeación central refleja las economías de la organización.

Los aspectos políticos en los sistemas distribuidos

- ✓ Los departamentos de sistemas requieren mayor autonomía local.
- ✓ En los departamentos de usuarios se fomenta la responsabilidad local.
- ✓ Los usuarios aprenden a apreciar los costos de procesamiento.
- ✓ La descentralización evita poner el poder en pocas manos.
- ✓ Las estructuras son más flexibles.

Los aspectos técnicos en los sistemas centralizados

- ✓ Los sistemas centralizados requieren un mantenimiento central de datos.
- ✓ Requieren un procesamiento total de datos.
- ✓ No requieren replicación alguna.
- ✓ Generalmente necesitan un control profesional de seguridad.

Los aspectos técnicos en los sistemas distribuidos

- ✓ Tienen mejores interfaces de usuario, como por ejemplo las de los sistemas cliente-servidor.
- ✓ Presentan cierto grado de disponibilidad aún en caso de falla.
- ✓ Tienen un mejor tiempo de respuesta.
- ✓ Las aplicaciones son más adaptadas a las necesidades locales.
- ✓ Los volúmenes de procesamiento deben ser relativamente grandes a nivel local.
- ✓ Los usuarios comprenden mejor sus problemas.
- ✓ Un grupo de soporte local reacciona más rápido a las demandas locales.
- ✓ Aprovechan la proliferación de equipos.

La administración de las bases de datos en los sistemas centralizados

- ✓ Existe un control total por parte del administrador de la base de datos global. Por razones de seguridad, hay gran énfasis en el control y este debe centralizarse.

La administración en los sistemas distribuidos

- ✓ Hay un alto grado de autonomía por parte de los administradores locales. En bases de datos distribuidas es posible identificar una estructura de control jerárquica basada en el administrador de la bases de datos global y en los administradores de las bases de datos locales. Estos pueden llegar a tener mucha autonomía (a tal punto que el GDBA pueda desaparecer) o todo lo contrario (un control casi completamente centralizado).

La organización de los datos en los sistemas centralizados

- ✓ La independencia de los datos: La organización real de los datos es transparente al programador de aplicaciones, es decir, los programas son escritos o construidos teniendo en cuenta el esquema conceptual de los datos. En este caso, los programas no son afectados por los cambios en la organización física de la base de datos.

La organización y la ubicación física de los datos en los sistemas distribuidos

- ✓ La independencia de los datos revista igual importancia para las bases de datos distribuidas.

- ✓ La transparencia de distribución es un aspecto adicional en las bases de datos distribuidas. Los programas deben poderse construir como si la base de datos no fuera distribuida. En este caso, los programas no son afectados por el movimiento de los datos de un sitio a otro (aunque el rendimiento sí cambie).

La redundancia de datos en bases de datos centralizadas

- ✓ Se reduce lo más que se pueda debido a la posibilidad de inconsistencias y al desperdicio de espacio.

La redundancia de datos en bases de datos distribuidas

- ✓ Es una característica deseable ya que lo que se pretende es que haya localidad de aplicaciones (la cual se incrementa si los datos se replican en los sitios donde se necesitan) y disponibilidad de la base de datos (aún en caídas de sitios). La disponibilidad de la base de datos depende de la relación consultas/actualizaciones.

Estructuras de acceso en las bases de datos centralizadas

- ✓ Los índices secundarios y las cadenas interarchivo (clusters) constituyen el aspecto más importante de los sistemas manejadores de bases de datos tradicionales, ya que su objetivo es lograr un acceso eficiente.

Estructuras de acceso en las bases de datos distribuidas

- ✓ Estructuras como los índices secundarios y los clusters interarchivo (o intertabla) no son las apropiadas para lograr un acceso eficiente, ya que es difícil construirlas y mantenerlas. Pues no son una respuesta tecnológica. No son la solución.
- ✓ Los accesos se logran a través de un plan de acceso distribuido, de tal manera que los programas tengan la mayor localidad posible y los resultados intermedios o las tablas sean transmitidos entre sitios.

Integridad en las bases de datos centralizadas

- ✓ Deben proveer el concepto de transacciones como unidades atómicas de ejecución como, por ejemplo, en el caso de una transferencia de fondos. Aunque las amenazas a la atomicidad existen como:
    - o Las fallas, en la mitad de una transacción

- La concurrencia, cuando una transacción ve el estado inconsistente (intermedio) de otra.

Integridad en las bases de datos distribuidas

- ✓ La aplicación del concepto de transacción se mantiene pero involucra un trabajo más complejo. Si hay dos sitios involucrados en una transacción y uno de ellos falla, hay dos posibilidades:
    - Que la transacción sea abortada
    - Que haya un sistema inteligente que termine la transacción correctamente aunque los dos sitios no estén operando simultáneamente.

Recuperación en bases de datos centralizadas

- ✓ Los sistemas encargados deben preservar la atomicidad de las transacciones en presencia de fallas.
- ✓ La recuperación debe hacerse hacia atrás o hacia delante, dependiendo de si la transacción se ha comprometido o no.

Recuperación en bases de datos distribuidas

- ✓ Deben preservar la atomicidad de las transacciones aun en casos de fallas en algunos sitios involucrados.
- ✓ La recuperación debe hacerse basado en protocolos complejos de compromiso en dos o en tres fases.

Control de concurrencia en bases de datos centralizadas

- ✓ Asegurar la atomicidad de las transacciones en la presencia de ejecución concurrente de transacciones.
- ✓ Trabajar adecuadamente los tipos de bloqueos.

Control de concurrencia en bases de datos distribuidas

- ✓ Asegurar la atomicidad de las transacciones distribuidas a través de la red.
- ✓ Trabajar adecuadamente los bloqueos en cada uno de los sitios.

Privacidad y seguridad en bases de datos centralizadas

- ✓ El DBA tiene un control centralizado a través del acceso autorizado a los datos. Pero es más vulnerable a las violaciones

de seguridad y privacidad que en archivos separados, desde el punto de vista de la globalidad de la información.

Privacidad y seguridad en bases de datos distribuidas

- ✓ Los administradores locales tienen el mismo problema de vulnerabilidad y privacidad de los administradores globales.
- ✓ Por la autonomía de cada sitio, los propietarios de los datos tienen sus propias protecciones, a veces buenas, a veces no tanto.
- ✓ Las bases de datos distribuidas presentan los problemas intrínsicos a los sistemas distribuidos ya que las redes representan un punto débil en protección.

2

Las Bases de Datos y las Redes de Computadores

El acceso y uso eficiente de la información son vitales para las empresas modernas. Características como la movilidad, los trabajos en grupo, la ubicación en zonas geográficas dispersas y la necesidad de obtener información en forma rápida y oportuna, hacen que la comunicación de datos, voz, texto, imagen y video sea un elemento clave para el éxito.

2.1 El Modelo relacional

Es un sistema que se apoya en el uso de expresiones asociativas poderosas orientadas a conjuntos (no a un registro, como en el caso de la mayoría de los modelos procedimentales).

El modelo relacional hace uso del álgebra relacional, la cual utiliza estrategias de acceso a la base de datos mientras que las herramientas como SQL son programas de aplicación directamente.

El modelo de datos relacional organiza y representa los datos en forma de tablas o relaciones. El concepto de Relación proviene de las matemáticas y representa una simple tabla de dos dimensiones, consistente en filas y columnas. Ejemplos de relaciones se encuentran en el desarrollo de bases de datos para las organizaciones.

La estructura de datos relacional maneja los conceptos de relación, atributo, tupla, grado, cardinalidad, dominio y esquema de una relación.

Una relación corresponde a los datos almacenados en una tabla, en lo que a bases de datos relacionales se refiere.

Un atributo, informalmente hablando, corresponde a lo que es una columna o un campo, tal como lo usa el programador de aplicaciones. De la misma manera, una tupla es un registro o una fila.

El grado es la cantidad de atributos definidos en una relación y la cardinalidad está definida por el número de tuplas.

El dominio es el tipo de datos o fondo de valores posibles para determinado atributo.

El esquema de una relación es la definición de la relación basada en el nombre de la relación y los nombres de cada uno de los atributos que la conforman. Por ejemplo, la relación EMP(CodEmp,Nombre,Salario) es el esquema de la relación Empleado.

Por otra parte, las reglas de integridad relacional, definen los mecanismos de llave primaria y llave foránea para permitir la conexión o asociación entre las relaciones (o tablas).

Una llave primaria constituye un identificador único para una relación, es decir, los valores de la llave primaria permiten identificar cada una de las filas de la relación. Una llave foránea en una relación sirve para referenciar datos en otra relación, es decir, los valores de la llave foránea deben coincidir con los valores de la llave primaria de la otra relación.

El álgebra relacional muestra un patrón de referencia que permite medir la capacidad expresiva de un lenguaje relacional a través de operaciones asociativas poderosas orientadas a conjuntos, no a un registro específico como lo hacen los lenguajes procedimentales. Estas operaciones pueden ser unarias o binarias.

Supongamos las siguientes tres relaciones:

R1

A	B	C
a	x	a
b	x	b
a	x	d
b	y	f

R2

A	B	C
a	x	a
a	z	f

R3

B	C	D
x	a	x
z	b	x
z	c	y
x	d	w
y	a	z

SELECCIÓN

Produce una relación con el mismo esquema de la relación operando (R1) y un subconjunto de tuplas del mismo que satisfacen un predicado (A=a).

$\mathbf{SL}_{A=a}R1$

A	B	C
a	x	a
a	x	d

PROYECCIÓN

Genera un conjunto de tuplas derivadas de la relación operando (R1) al proyectar un subconjunto de atributos de esta (A,B). Si hay tuplas repetidas, estas se eliminan.

PJ$_{A,B}$R1

A	B
a	x
b	x
b	y

UNION

Produce una relación con el mismo esquema de cada uno de los operandos y un conjunto de tuplas resultante de unir las de las relaciones operando (R1 y R2).

R1 **UN** R2

A	B	C
a	x	a
b	x	b
a	x	d
b	y	f
a	z	f

DIFERENCIA
Está formada por las tuplas de la primera relación (R1) que no se encuentran en la segunda (R2).

R1 **DF** R2

A	B	C
b	x	b
a	x	d
b	y	f

PRODUCTO CARTESIANO
Produce una relación con todas los atributos de los dos relaciones, donde cada tupla de la primera (R1) se combina con todas las tuplas de la segunda (R2).

R1 **CP** R2

R1.A	R1.B	R1.C	R2.A	R2.B	R2.C
a	x	a	a	x	a
b	x	b	a	x	a
a	x	d	a	x	a

b	y	f	a	x	a
a	x	a	a	z	f
b	x	b	a	z	f
a	x	d	a	z	f
b	y	f	a	z	f

JOIN

El join de dos relaciones (R1 y R3) se basa en una fórmula (R1.C=R3.C) que especifica el predicado del join. Normalmente está dada por conjunciones de comparaciones entre atributos tomados de los dos operandos. Un join se deriva del producto cartesiano y de la selección.

R1 **JN**$_{R1.C=R3.C}$ R3

R1.A	R1.B	R1.C	R3.B	R3.C	R3.D
a	x	**a**	x	**a**	x
a	x	**a**	y	**a**	z
b	x	**b**	z	**b**	x
a	x	**d**	x	**d**	w

JOIN NATURAL

También denominado equijoin, el join natural genera una relación en la que de cada par de atributos idénticos se descarta uno. Las tuplas resultantes son las que satisfacen la igualdad en dichos atributos.

R1 **NJN** R3

A	B	C	D
a	x	a	x
a	x	d	w

SEMIJOIN

Es el resultante de aplicar proyección sobre los atributos del primer operando (R1) después de haber hecho join a los dos operandos (R1 y R2). Debe tener en cuenta un predicado, igual que con el join.

R1 **SJ**$_{R1.C=R3.C}$ R3

A	B	C
a	x	a
b	x	b
a	x	d

SEMIJOIN NATURAL

Es el resultante de aplicar proyección sobre los atributos del primer operando (R1) después de haber hecho join natural a los dos operandos (R1 y R2).

R1 **NSJ** R3

A	B	C
a	x	a
a	x	d

AGRUPAMIENTO

Operación definida por un conjunto de atributos (A) que determinan el agrupamiento y unas funciones agregadas (COUNT(B)) a ser evaluadas en cada grupo de la relación (R1).

$\mathbf{GB}_{A,COUNT(B)}$R1

A	COUNT(B)
a	2
b	2

Las bases de datos están estrechamente relacionadas con las aplicaciones, los programas, las transacciones y las consultas. Las aplicaciones son secuencias de operaciones requeridas por un usuario final mientras los programas están más relacionados con cálculos o cómputos más específicos. Las transacciones son unidades lógicas de trabajo que se constituyen en objeto de las aplicaciones mientras que las consultas o queries son simples expresiones de datos que permiten obtener algún tipo de información.

2.2 Las redes de computadores

Las redes de computadores son sistemas en los cuales varias máquinas autónomas, separadas pero interconectados trabajan cooperativamente en la ejecución de alguna tarea.

Una red puede soportar un proceso corriendo en un sitio de tal manera que pueda enviar un mensaje a otro proceso corriendo en cualquier otro sitio de la misma red.

Discutiremos algunos aspectos fundamentales de redes a continuación.

Los parámetros básicos que se deben tener al transmitir un mensaje a través de una red son:

1. Retardo del mensaje en llegar al destino.
2. Costo de transmisión del mensaje: Costo fijo + Costo adicional.
3. Confiabilidad de la red: Probabilidad de que el mensaje llegue correctamente.

Las redes según su cobertura se clasifican en **Redes LAN** y **Redes WAN**. Las primeras son las redes de área local, las cuales se circunscriben a una empresa o a un campus, dentro de una extensión de más o menos un kilómetro de radio, aunque esto puede variar. Las redes WAN son redes de área amplia y se dispersan normalmente sobre un área geográfica, en un área metropolitana, en un departamento, en un país o inclusive en todo el mundo.

Un proceso **broadcast** es aquel que, estando en un sitio, envía el mismo mensaje a todos los otros sitios. Por eso en una red broadcast, al utilizar un medio común, los mensajes son recibidos por todos los sitios. Las redes en bus o satelitales son broadcast.

EL MODELO OSI

Los Requerimientos de Interconexión de múltiples ambientes de muchos proveedores llevaron a promover la demanda de estándares de comunicación que no estuvieran ligados a ningún propietario ni a ninguna arquitectura: Un Modelo para una arquitectura abierta: El modelo OSI. En él se manejan los conceptos de protocolo e interfaz.

Un protocolo es un conjunto de normas entre dos equipos al mismo nivel. Una interfaz es la comunicación entre dos niveles del mismo equipo.

Este modelo está formado por siete niveles, desde los más dependientes de los dispositivos hasta los más independientes y más inteligentes. Ellos son: Físico, Enlace, Red, Transporte, Sesión, Presentación y Aplicación.

El nivel **Físico** permite la transmisión de bits sobre un canal de comunicación. Sus componentes son tangibles, tales como los cables, los conectores y las tarjetas.

Permite la transmisión de señales eléctricas (a través de cobre) o de luz (a través de fibra óptica) representando bits. Entre los dispositivos de este nivel se encuentran el hub, el repetidor y el módem. En él se manejan las topologías físicas como el anillo, el bus y la estrella.

Entre los principales productos del nivel físico están las interfaces seriales RS232, RS449 y X.21 y las interfaces I.430, I.431 de ISDN.

El nivel de **Enlace** determina como un nodo accesa al medio físico, el momento y la manera en que se deposita la información en el canal. Permite la comunicación eficiente entre dos máquinas adyacentes.

Hace el manejo de tramas o marcos, en cuya transmisión los datos son enviados secuencialmente y a la vez hay recepción de marcos de confirmación.

Entre los dispositivos de este nivel se encuentran los bridges, los switches y las tarjetas de red.

Entre los productos soportados se encuentran Ethernet, Token-Ring, FDI, ATM y Redes Wavelan IEEE 802.11

El nivel de **Red** Maneja los algoritmos de enrutamiento, es decir, decide la interfaz que debe utilizar un paquete que llega y que debe ser retransmitido. Involucra bloques de control que permiten hacer chequeo de errores.

Permite control de congestión: La congestión es un problema de escasez de buffers. No hay donde colocar los paquetes.

Hace control de flujo : El generador de flujo no debe transmitir más paquetes de los que el receptor puede recibir.

Utiliza algoritmos de enrutamiento, los cuales se basan en el uso de tablas y maneja direcciones a nivel lógico para encontrar la estación destino. El dispositivo fundamental utilizado es el router.

Protocolos soportados: IP, ARP e ICMP de TCP/IP, IPX de IPX/SPX, I.450 e I.451 de ISDN, X.25 y X.75

El nivel de **Transporte** se encarga de establecer una comunicación confiable entre hosts. Tiene sentido entre puntos de la red que desean comunicarse, sin necesidad de que sean adyacentes.

En una red local no hay nodos intermedios, es decir, el transporte se vuelve inmediato y su función casi trivial.

El control de errores permite un transporte de datos fiable y eficiente. Libera a las capas superiores del problema del transporte de información entre ellas.

Tiene funciones como las de establecimiento de conexión, direccionamiento, secuenciación, recuperación de errores y fallas, multiplexado, control de flujo, gestión de memorias intermedias y sincronización.

Entre los protocolos de nivel de transporte se encuentran: TCP y UDP de TCP/IP y SPX de IPX/SPX.

El nivel de **Sesión** establece una comunicación confiable entre procesos, es decir, conecta un proceso local con un proceso en otra máquina.

Soporta puertos virtuales, o sea, canales múltiples en un solo enlace físico.

En el software de nivel de sesión encontramos RPC, Sockets, Shell de Netware y NetBIOS. Un ejemplo específico es el de TNS de Oracle.

El nivel de **Presentación** se encarga de las funciones de criptografía, compactación, cifrado, compresión, y conversión entre un código y otro. Resuelve inconsistencias entre sistemas operacionales.

Provee servicios utilizados para la interpretación de la sintaxis de los datos transmitidos: Transformación de Sintaxis.

Suministra un lenguaje estándar de representación para que los datos se presenten en forma uniforme. Por ejemplo: Entre una máquina AS400 (EBCDIC) y una UNIX (ASCII) o entre un computador con procesador MOTOROLA y otro con INTEL que representan un entero en forma diferente en memoria.

Maneja protocolos de terminales virtuales: Convierten de características de una terminal específica a un modelo virtual o genérico usado por los programas de aplicación.

XDR es un protocolo de nivel de sesión que permite la descripción y codificación de datos en un formato independiente del tipo de máquina.

En el nivel de **Aplicación** se muestra una forma transparente de accesar los servicios de comunicación de los niveles inferiores.

Aquí aparecen las herramientas del usuario, las aplicaciones desarrolladas por programadores para darle transparencia al usuario, aunque sean funcionalmente definidas por el usuario.

Se encuentran los protocolos de terminal virtual, de transferencia de archivos, de correo electrónico, de administración de red y las aplicaciones distribuidas.

Entre los servicios de nivel de aplicación de TCP/IP se encuentran: telnet, ftp, SMTP, DNS y SNMP. Por otro lado se encuentra NFS de Sun, NetBEUI, las herramientas cliente-servidor, los sistemas operacionales de red, e-mail, http y los navegadores web.

2.3 Los sistemas de manejo de bases de datos distribuidas

Conociendo las funcionalidades de un sistema de manejo de bases de datos y de un sistema de red, se concluye que un sistema manejador de bases de datos distribuidas debe permitir la creación y mantenimiento de bases de datos que se distribuyen en una red. Los componentes típicos para lograr esta tarea son:

1. Manejador de la base de datos (DB).
2. Componente de comunicación de datos (DC).
3. Diccionario de datos (DD): Información sobre la distribución de los datos en la red.
4. Manejador de la distribución de datos (DDB).

Los servicios soportados son:

1. Acceso a una base de datos remota por un programa de aplicación.
2. Algún grado de transparencia a la distribución.

3. Administración y control.
4. Control de concurrencia y recuperación.
5. Seguridad.

Los tipos de acceso a una base de datos distribuida pueden ser:

1. Acceso remoto por medio de primitivas del DBMS.
2. Acceso remoto a través de programas auxiliares.

Existen los DBMSs homogéneos, los cuales son sistemas que tienen igual DBMS en cada sitio aunque los computadores y los sistemas operacionales sean diferentes.

Por otro lado, los DBMSs heterogéneos son aquellos en los que se tienen por lo menos dos DBMSs diferentes.

EJERCICIOS PROPUESTOS

En las siguientes preguntas señale la correcta:

1. En un ambiente cliente/servidor siempre debe haber:
   a) Procesos distribuidos
   b) Bases de datos locales que se comunican
   c) Bases de datos distribuidas
   d) Ninguna anterior
2. Para poner a funcionar una base de datos distribuida es necesario
   a) Un sistema operativo de red
   b) Un sistema operativo distribuido
   c) Un sistema operativo que pueda operar sobre una red
   d) Un sistema operativo monousuario
3. Entre los beneficios de los sistemas distribuidos se encuentran:
   a) Mayor tiempo de respuesta
   b) Optimización en el uso de la red
   c) Disponibilidad de datos
   d) Redundancia de datos
4. Una base de datos distribuida es aquella en que
   a) Los datos están distribuidos en diferentes sitios de una red
   b) Los datos no están en un solo sitio y además existe alguna relación entre los datos de los sitios
   c) Existen aplicaciones que requieren acceder datos distribuidos en diferentes sitios de una red
   d) La b) y la c) son correctas

5. Si necesito comunicar una herramienta C++Builder con una BD Oracle a qué niveles OSI pertenecen los siguientes componentes:
    a) Comunicación de un proceso desde la máquina C++Builder con otro proceso servidor en la máquina Oracle: _____
    b) Protocolo de entrega confiable de datos, incluyendo implícita-mente todo el enrutamiento: _____
    c) Tarjeta de red y Hub: _____
    d) Conectividad de bases de datos heterogéneas: _____
6. Si la base de datos distribuida se vuelve lenta debido a que se comparte el ancho de bando entre todos las máquinas, el hub se debe cambiar por:
    a) Un boundary router
    b) Un bridge con agentes de administración
    c) Un switch
    d) Un terminal server
7. En una arquitectura 3-Tier debe haber:
    a) Un servidor de bases de datos, un servidor de correo y un Web browser
    b) Un motor de bases de datos, un servidor de bases de datos y una estación
    c) Un servidor de bases de datos, un servidor de aplicaciones y una estación
    d) Un servidor de bases de datos, un servidor Web y una aplicación
8. Las arquitecturas 2-Tier también son llamadas:
    a) Cliente/Servidor
    b) Arquitecturas multiusuario
    c) Arquitecturas de tiempo compartido
    d) Herramientas visuales
9. Se requiere instalar una estación en un sitio que se encuentra a 150 metros del centro de cableado más cercano. Ella requiere acceder la base de datos que se encuentra funcionando en la red. ¿Qué se requiere?
    a) Un hub adicional
    b) Instalar un centro de cableado en un sitio intermedio entre la estación y el otro centro de cableado
    c) Instalar un repetidor
    d) Desplazar el centro de cableado unos 50 metros
10. Entre los beneficios directos de los sistemas distribuidos se encuentran:
    a) Mayor tiempo de respuesta
    b) Optimización en el uso de la red
    c) Disponibilidad de datos

d) Redundancia de datos

11. Una base de datos distribuida es aquella en que
    a) Existe un control total de cuenta del DBA Global
    b) La redundancia es mínima o no existe
    c) El usuario percibe los datos como si estuvieran en un solo sitio
    d) Los clusters inter-tabla e intra-tabla se construyen dependiendo de cómo están los datos en los sitios.
12. Desde el punto de vista organizacional, cuál es el mayor impacto político de la descentralización:
    a) La gerencia central pierde poder
    b) Mayor aprovechamiento de los recursos existentes
    c) Permite tener unas estructuras más flexibles
    d) Fomenta la responsabilidad de los usuarios por sus datos

3

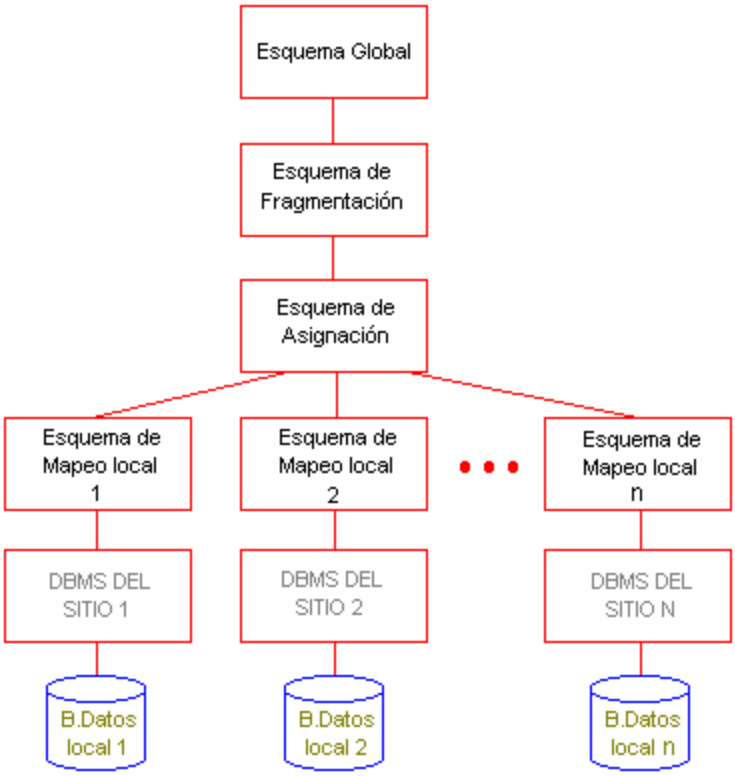
Niveles de Transparencia de Distribución

Los niveles de transparencia corresponden a las diferentes perspectivas en las que un programador de aplicaciones ve la base de datos distribuida, dependiendo de que tanto la transparencia de distribución es provista por el Sistema Manejador de la Base de Datos Distribuida.

La transparencia de distribución da independencia del programa de aplicación a la distribución de los datos, siempre que ella sea completa.

3.1 Arquitectura de referencia para bases de datos distribuidas

Para comprender la organización de una base de datos distribuidas hay cuatro niveles conceptualmente relevantes que son el nivel o esquema global, el esquema de fragmentación, el esquema de asignación y el esquema de mapeo local.



El **Esquema Global** define todos los datos contenidos en la base de datos distribuida como si ella no fuera distribuida.

Al usar el modelo relacional, el esquema global consiste en la definición de un conjunto de relaciones globales. Cada relación global puede ser partida en fragmentos (posiblemente disyuntos).

El **Esquema de Fragmentación** define la correspondencia entre relaciones globales y fragmentos.

Establece un mapeo 1:N donde a cada relación global le corresponden varios fragmentos. Cada fragmento lo denotamos como R_i para referirnos al iésimo fragmento de la relación global R.

Los fragmentos son porciones lógicas de relaciones globales que están localizadas físicamente en uno o varios sitios de la red.

El **Esquema de Asignación** define en qué sitios se localizan los fragmentos.

El tipo de mapeo definido en este esquema determina si la base de datos distribuida es redundante (en cuyo caso es 1:N) o no redundante (en cuyo caso es 1:1).

Los fragmentos de R en el sitio j constituyen la imagen física de R en el sitio j. A esta se le denota como R^j.

La copia de un fragmento se denota por R_i^j, lo que quiere decir, la copia del fragmento i en el sitio j.

Dos imágenes físicas pueden ser idénticas, o sea, una imagen física es copia de la otra: $R^k=R^j$

El **Esquema de Mapeo local** define el mapeo de las imágenes físicas con los objetos manipulados por los sistemas manejadores de bases de datos locales. Este nivel es dependiente del tipo de DBMSs local.

En un ambiente heterogéneo hay diferentes tipos de mapeos locales para los sitios.

OBJETIVOS IMPORTANTES QUE MOTIVAN ESTA ARQUITECTURA

Para Ceri y Pelagatti existen tres objetivos importantes que motivan la arquitectura presentada:

1. Separación del concepto de fragmentación de datos del concepto de asignación de datos a los sitios. Es muy conveniente en el diseño de una base de datos.
2. Control explícito de redundancia, lo cual es útil en actividades de administración y nos lleva a soportar la transparencia de replicación, en la cual el usuario o el programador no ven la replicación de fragmentos. La transparencia de replicación es un concepto derivado de la transparencia de ubicación.
3. Independencia de los DBMSs locales (o transparencia de mapeo local). Esto permite estudiar problemas de manejo de bases de datos distribuidas sin tener en cuenta los modelos de datos locales.

CONDICIONES QUE DEBEN CUMPLIR LOS FRAGMENTOS

Un fragmento puede ser definido por una expresión en un lenguaje relacional como el álgebra relacional y debe cumplir con las siguientes reglas:

1. Completitud: Todos los datos deben estar mapeados en los fragmentos.
2. Reconstrucción: Se debe poder reconstruir la relación global a partir de los fragmentos.
3. Disyunción: Los fragmentos deben ser disyuntos, aunque fundamentalmente esto se debe cumplir con los tipos de fragmentación horizontal y horizontal derivada, los cuales se verán en las siguientes secciones.

3.2 Tipos de fragmentación

FRAGMENTACIÓN HORIZONTAL PRIMARIA

Es el particionamiento de las tuplas de una relación global R, en subconjuntos.

Supongamos que existe una relación de clientes definida con el siguiente esquema:

CLIENTE (CNUM, NOMBRE, CIUDAD)

Definamos los siguientes fragmentos horizontales:

CLIEN1 = $\mathbf{SL}_{\text{CIUDAD='Armenia'}}$CLIENTE
CLIEN2 = $\mathbf{SL}_{\text{CIUDAD='Pereira'}}$CLIENTE

Esta fragmentación satisface la condición de completitud, siempre y cuando Armenia y Pereira sean las únicas ciudades de los clientes almacenados en la base de datos.

Si CLIENTE = CLIEN1 **UN** CLIEN2, entonces se satisface la condición de reconstrucción.

Si los predicados de selección son mutuamente excluyentes, la condición de disyunción se satisface. Y esto es apenas lógico ya que un cliente, de acuerdo con la definición pertenece a una y sólo una ciudad.

Una **cualificación** es un predicado usado en la selección para definir un fragmento, de tal manera que podemos definir los cualificadores q1 y q2 así:

q1: ciudad = 'Armenia'
q2: ciudad = 'Pereira'

Estos cualificadores satisfacen las condiciones 1, 2 y 3 para los fragmentos que definen.

FRAGMENTACIÓN HORIZONTAL DERIVADA

Es la fragmentación de una relación R1 que se deriva al aplicar semijoin a una relación R2 que ha sido previamente fragmentada horizontalmente. Consideremos la siguiente relación de partes despachadas:

PART_DESP(CNUM, PNUM, DNUM, CANT), donde CNUM es el número del cliente que recibe determinada parte, PNUM es el número de la parte y DNUM es el número de la dependencia que despacha la parte. Fragmentemos esta relación de la siguiente manera:

PD1 = PART_DESP **SJ** $_{CNUM=CNUM}$ CLIEN1
PD2 = PART_DESP **SJ** $_{CNUM=CNUM}$ CLIEN2

PD1 determina aquellas tuplas de partes despachadas a los clientes de Armenia y PD2 queda formada por las partes despachadas a los clientes de Pereira.

La cualificación es la siguiente:

q1: PART_DESP.CNUM = CLIENTE.CNUM AND

CLIENTE.CIUDAD = 'Armenia'
q2: PART_DESP.CNUM = CLIENTE.CNUM AND
CLIENTE.CIUDAD = 'Pereira'

FRAGMENTACIÓN VERTICAL

Es la subdivisión de los atributos de una relación R en grupos. Los fragmentos se obtienen proyectando sobre la relación global cada grupo. Cada grupo puede obedecer a propiedades geográficas comunes u otro criterio para su selección.

En una relación de vendedores podría definirse el siguiente esquema:

VEND (VNUM, NOMBRE, SAL, COMM, JEFE, DNUM), donde VNUM es el número del vendedor, SAL es el salario y COMM la comisión. DNUM es el número de dependencia donde trabaja el vendedor.

Definamos la siguiente fragmentación:

VEND1 = **PJ** $_{\text{VNUM,NOMBRE,JEFE,DNUM}}$ VEND
VEND2 = **PJ** $_{\text{VNUM,NOMBRE,SAL,COMM}}$ VEND

Visto así, VEND1 representa los datos básicos del vendedor mientras que VEND2 es la información de nómina.

La condición de completitud se cumple ya que cada atributo es mapeado en al menos un atributo de los fragmentos.

La condición de reconstrucción se cumple ya que es posible reconstruir la relación original: VEND = VEND1 **JN** $_{\text{VNUM=VNUM}}$ VEND2. Las columnas VNUM y NOMBRE se repiten, por tanto, se deben eliminar mediante una proyección.

La condición de disyunción no es obligada en el caso de la fragmentación vertical ya que las llaves primarias, por ejemplo, tienen que estar presentes en los fragmentos.

FRAGMENTACIÓN MIXTA

Es la aplicación de operaciones de fragmentación vertical y horizontal en forma recursiva.

Supóngase la relación VEND definida arriba. Sean los siguientes fragmentos:

VEND1 = **SL** $_{DNUM \leq 5}$ **PJ** $_{VNUM,NOMBRE,JEFE,DNUM}$ VEND
VEND2 = **SL** $_{5 < DNUM \leq 10}$ **PJ** $_{VNUM,NOMBRE,JEFE,DNUM}$ VEND
VEND3 = **SL** $_{DNUM > 10}$ **PJ** $_{VNUM,NOMBRE,JEFE,DNUM}$ VEND
VEND4 = **PJ** $_{VNUM,NOMBRE,SAL,COMM}$ VEND

VEND1	VEND4
VEND2	
VEND3	

Como puede verse, la fragmentación en una primera fase se ha hecho verticalmente mediante la proyección y seguidamente a uno de los fragmentos generados se le ha aplicado fragmentación horizontal mediante la selección.

La relación original se puede reconstruir mediante la expresión:

VEND=**UN**(VEND1,VEND2,VEND3) **JN**$_{VNUM=VNUM}$ **PJ** $_{VNUM, SAL, COMM}$VEND4

EJEMPLO DE UNA BASE DE DATOS DISTRIBUIDA

Esquema Global:

CLIENTE (CNUM, NOMBRE, CIUDAD)
PARTE (PNUM, NOMBRE)
PART_DESP (CNUM, PNUM, DNUM, CANT)
VEND (VNUM, NOMBRE, SAL, COMM, JEFE, DNUM)
DEPEND (DNUM, NOMDEP, ZONA)

Esquema de Fragmentación:

CLIEN1 = **SL**$_{CIUDAD='Armenia'}$CLIENTE
CLIEN2 = **SL**$_{CIUDAD='Pereira'}$CLIENTE
PD1 = PART_DESP **SJ** $_{CNUM=CNUM}$ CLIEN1
PD2 = PART_DESP **SJ** $_{CNUM=CNUM}$ CLIEN2
VEND1 = **SL** $_{DNUM \leq 5}$ **PJ** $_{VNUM,NOMBRE,JEFE,DNUM}$ VEND
VEND2 = **SL** $_{5 < DNUM \leq 10}$ **PJ** $_{VNUM,NOMBRE,JEFE,DNUM}$ VEND
VEND3 = **SL** $_{DNUM > 10}$ **PJ** $_{VNUM,NOMBRE,JEFE,DNUM}$ VEND
VEND4 = **PJ** $_{VNUM,NOMBRE,SAL,COMM}$ VEND
DEP1 = **SL** $_{DNUM \leq 5}$ DEPEND
DEP2 = **SL** $_{5 < DNUM \leq 10}$ DEPEND
DEP3 = **SL** $_{DNUM > 10}$ DEPEND

3.3 Transparencia de distribución para aplicaciones de consulta

Las aplicaciones pueden ser escritas en los diferentes niveles por el DDBMS, dependiendo la transparencia que se tenga. Veamos un

ejemplo sencillo: Aceptar un número de cliente desde una terminal, hallar el nombre correspondiente y mostrarlo en la terminal.

NIVEL 1: TRANSPARENCIA DE FRAGMENTACIÓN

El DDBMS interpreta las primitivas (instrucciones SQL) accesando las bases de datos, estén en el fragmento que estén y en el sitio que estén, en una forma completamente determinada por el sistema.

La aplicación se refiere a las relaciones globales, ignorando el hecho de que la base de datos esté distribuida. Ejemplo:

```
Read($cnum);
SELECT nombre INTO $nombre
FROM cliente
WHERE cnum=$cnum;
Write($nombre);
```

La aplicación es inmune a los cambios realizados a los esquemas que estén debajo del esquema global.

NIVEL 2: TRANSPARENCIA DE UBICACIÓN

El DDBMS interpreta las primitivas accesando las bases de datos, estén en el sitio que estén, pero especificando los fragmentos.

Las solicitudes se pueden ejecutar en secuencia o en paralelo para explotar el paralelismo de un sistema distribuido.

```
Read($cnum);                          /*versión en secuencia */
SELECT nombre INTO $nombre
FROM clien1
WHERE cnum=$cnum;
If not #found then begin
        SELECT nombre INTO $nombre
        FROM clien2
        WHERE cnum=$cnum;
End;
Write($nombre);

Read($cnum);                          /*versión en paralelo */
Read($ciudad);
Case $ciudad of
   'Armenia':   SELECT nombre INTO $nombre
                FROM clien1
                WHERE cnum=$cnum;
   'Pereira':   SELECT nombre INTO $nombre
```

```
                    FROM clien2
                    WHERE cnum=$cnum;
End;
Write($nombre);
```

La aplicación se refiere a las relaciones del esquema de fragmentación, ignorando los sitios en los que se encuentren las copias de cada fragmento.

La aplicación es independiente de los cambios en el esquema de asignación, pero no a los cambios en el esquema de fragmentación.

Conociendo la estructura de fragmentación, puede ser más eficiente escribir aplicaciones (versión en paralelo) sin embargo, es más dependiente del esquema de fragmentación. Si la ciudad llegare a cambiar, tocaría cambiar el programa.

NIVEL 3: TRANSPARENCIA DE MAPEO LOCAL

El DDBMS interpreta las primitivas, accesando las bases de datos, usando nombres de objetos independientes a los sistemas locales, pero especificando los sitios (y los fragmentos) donde residen.

Supongamos que el fragmento CLIEN1 se encuentra en el sitio 1 y que CLIEN2 está replicado en los sitios 2 y 3. Para consultar únicamente y en el caso de CLIEN2 es suficiente consultarlo en uno de los dos sitios:

```
Read($cnum);
SELECT nombre INTO $nombre
FROM clien1 AT SITE 1
WHERE cnum=$cnum;
If not #found then begin
        SELECT nombre INTO $nombre
        FROM clien2 AT SITE 3          /* se puede usar SITE 2 */
        WHERE cnum=$cnum;
End;
Write($nombre);
```

La aplicación se refiere a las relaciones del esquema de asignación. En este caso, las primitivas de acceso son direccionadas por el DDBMS a los sitios específicos. Ellas ignoran los nombres de los fragmentos específicos a cada sitio.

El aspecto más importante es el mapeo de las primitivas usadas por la aplicación en primitivas usadas por el DBMS local.

La transformación, por parte del DDBMS, de las primitivas de acceso entre un DBMS y otro, es una característica importante en un DDBMS heterogéneo.

NIVEL 4: SIN TRANSPARENCIA

Este nivel corresponde a programas auxiliares desarrollados en lenguajes anfitriones como en C++ (no directamente en SQL). Estos lenguajes corresponden a los DBMSs de cada sitio.

Los programas auxiliares deben manejar las condiciones y los códigos de retorno explícitamente.

El DDBMS aceptaría una primitiva del tipo *execute* que especifica cual programa se debe ejecutar en qué sitio, y los parámetros necesarios.

3.4 Transparencia de distribución en consultas con joins

En este ejercicio, el objetivo es consultar el nombre de la dependencia donde trabaja un vendedor dado. Se debe ingresar el número de vendedor.

NIVEL 1: TRANSPARENCIA DE FRAGMENTACIÓN

El comportamiento es igual que con cualquier otra consulta. Se diferencia del anterior en que la consulta global contiene un join.

```
Read($vnum);
SELECT nomdep INTO $nomdep
FROM vend, depend
WHERE vend.dnum = depend.dnum
  AND   vend.vnum = $vnum;
Write($nomdep);
```

NIVEL 2: TRANSPARENCIA DE UBICACIÓN

El programador define la estrategia de acceso, no el sistema, como en el caso del nivel global.

```
Read($vnum);
SELECT nomdep INTO $nomdep
FROM vend1, dep1
```

```
WHERE vend1.dnum = dep1.dnum
  AND   vend1.vnum = $vnum;
Write($nomdep);
If not #found then begin
   SELECT nomdep INTO $nomdep
   FROM vend2, dep2
   WHERE vend2.dnum = dep2.dnum
     AND vend2.vnum = $vnum;
   If not #found then begin
          SELECT nomdep INTO $nomdep
          FROM vend3, dep3
          WHERE vend3.dnum = dep3.dnum
            AND vend3.vnum = $vnum;
End;
```

NIVEL 3: TRANSPARENCIA DE MAPEO LOCAL

Consideremos que en el esquema de asignación, el fragmento DEP1 se encuentra en el sitio 1, DEP2 en el sitio 2, DEP3 en el sitio 3, VEND1 se encuentra en el sitio 4, VEND2 en el 5 y VEND3 en el 6.

```
Read($vnum);
SELECT dnum INTO $dnum
FROM vend1 AT SITE 4
WHERE vnum = $vnum;
If #found then begin
   SELECT nomdep INTO $nomdep
   FROM dep1 AT SITE 1
   WHERE dnum=$dnum;
End;
Else begin
   SELECT dnum INTO $dnum
   FROM vend2 AT SITE 5
   WHERE vnum = $vnum;
   If #found then begin
      SELECT nomdep INTO $nomdep
      FROM dep2 AT SITE 2
      WHERE dnum=$dnum;
   End;
   Else begin
          SELECT dnum INTO $dnum
          FROM vend3 AT SITE 6
          WHERE vnum = $vnum;
          SELECT nomdep INTO $nomdep
          FROM dep3 AT SITE 3
          WHERE dnum=$dnum;
   End;
End;
Write($nomdep);
```

3.5 Transparencia de distribución en aplicaciones de actualización

Las consultas sobre los datos distribuidos siempre se pueden hacer a partir de una copia de los datos. La actualización, sin embargo, se debe lograr sobre todas las copias de un dato. Es precisamente aquí donde tiene sentido la transparencia de ubicación y de replicación. Pues cuando se trabaja en el nivel de la ubicación, la actualización de una tupla puede generar migración de la tupla a otro fragmento.

Por ahora, nos dedicaremos a las actualizaciones sin garantizar la atomicidad de las transacciones.

Veamos nuevamente cómo es la fragmentación de una relación Vendedor, suponiendo que el esquema es el siguiente:

VEND (VNUM, NOMBRE, SAL, COM, JEFE, DNUM)

Asumamos que se ha fragmentado horizontalmente así:

$V1 = \mathbf{SL}_{DNUM \leq 5}$ (EMP)
$V2 = \mathbf{SL}_{DNUM > 5}$ (EMP)

Adicionalmente, V1 se ha fragmentado verticalmente en VEND1 (datos financieros) y VEND2 (datos de dependencia). Por otra parte, V2 se ha fragmentado verticalmente en VEND3 (datos básicos) y VEND4 (datos financieros y de jefe). El resultado es el siguiente:

$VEND1 = \mathbf{PJ}_{VNUM,NOMBRE,SAL,COM}\ \mathbf{SL}_{DNUM \leq 5}$ (EMP)
$VEND2 = \mathbf{PJ}_{VNUM,JEFE,DNUM}\ \mathbf{SL}_{DNUM \leq 5}$ (EMP)
$VEND3 = \mathbf{PJ}_{VNUM,NOMBRE,DNUM}\ \mathbf{SL}_{DNUM > 5}$ (EMP)
$VEND4 = \mathbf{PJ}_{VNUM,SAL,COM,JEFE}\ \mathbf{SL}_{DNUM > 5}$ (EMP)

NIVEL 1: TRANSPARENCIA DE FRAGMENTACIÓN

En este nivel, las actualizaciones se ejecutan como si la base de datos no fuera distribuida.

Es responsabilidad del DDBMS ejecutar todas las operaciones implicadas por los esquemas de fragmentación y de asignación.

Supongamos que en la fábrica de partes de automotores, el vendedor identificado con el número 79 es trasladado de la

dependencia de ventas identificada con el número 3 a la dependencia número 8. En el nivel global, lo único que tenemos que hacer es actualizar el número de la dependencia del empleado. No se necesita conocer ni la fragmentación ni los sitios de los fragmentos de la relación Vendedor:

```
UPDATE vend
SET dnum=8
WHERE vnum=79;
```

NIVEL 2: TRANSPARENCIA DE UBICACIÓN

Las consultas pueden explorar todas las posibilidades o bien tomar ventaja del contenido de la información del esquema de fragmentación para ser realizadas.

Las actualizaciones se ejecutan teniendo en cuenta la manera en que está definida la fragmentación. Hay que tener en cuenta que una actualización puede involucrar inserciones en un fragmento y borrados en otro.

El programa no requiere conocer los sitios a los que han sido asignados los fragmentos:

```
SELECT nombre, sal, com INTO $nombre, $sal, $com
FROM vend1
WHERE vnum=79;

SELECT jefe INTO $jefe
FROM vend2
WHERE vnum=79;

INSERT INTO vend3 (vnum, nombre, dnum)
VALUES (79, $nombre, 8);

INSERT INTO vend4 (vnum, sal, com, jefe)
VALUES (79, $sal, $com, $jefe);

DELETE FROM vend1 WHERE vnum=79;
DELETE FROM vend2 WHERE vnum=79;
```

NIVEL 3: TRANSPARENCIA DE MAPEO LOCAL

Las actualizaciones se deben entender con la ubicación de los fragmentos. Además deben tener en cuenta el grado de replicación.

Supongamos que en la fragmentación realizada, la asignación de sitios es la siguiente:

VEND1, en los sitios 1 y 2
VEND2, en los sitios 3 y 4
VEND3, en los sitios 5 y 6
VEND4, en los sitios 7 y 8

El programa sería:

```
SELECT nombre, sal, com INTO $nombre, $sal, $com
FROM vend1 AT SITE 1
WHERE vnum=79;

SELECT jefe INTO $jefe
FROM vend2 AT SITE 3
WHERE vnum=79;

INSERT INTO vend3 (vnum, nombre, dnum)
AT SITE 5 VALUES (79, $nombre, 8);

INSERT INTO vend3 (vnum, nombre, dnum)
AT SITE 6 VALUES (79, $nombre, 8);

INSERT INTO vend4 (vnum, sal, com, jefe)
AT SITE 7 VALUES (79, $sal, $com, $jefe);

INSERT INTO vend4 (vnum, sal, com, jefe)
AT SITE 8 VALUES (79, $sal, $com, $jefe);

DELETE FROM vend1
AT SITE 1 WHERE vnum=79;

DELETE FROM vend1
AT SITE 2 WHERE vnum=79;

DELETE FROM vend2
AT SITE 3 WHERE vnum=79;

DELETE FROM vend2
AT SITE 4 WHERE vnum=79;
```

NIVEL 4: SIN TRANSPARENCIA

Es el nivel más complejo ya que se entiende con detalles del lenguaje. En los ambientes heterogéneos, por ejemplo, no existe transparencia de mapeo local, luego hay que acudir a las posibilidades que brinden los lenguajes anfitriones particulares para lograr las operaciones en un ambiente distribuido.

CONSULTA A VALORES ÚNICOS Y VALORES MÚLTIPLES

Hay consultas que retornan un valor. En este caso se requiere el uso de una variable.

Los consultas o queries que retornan una relación requieren el manejo de un archivo o relación, trátese de un lenguaje de programación o de SQL. Analicemos la siguiente instrucción:

```
SELECT empnum, nombre INTO $emp_rel ($empnum, $nombre)
FROM emp;
```

$emp_rel es un parámetro o variable de tipo *archivo* para un lenguaje de programación o de tipo *relación* para SQL. Lo importante es que se utilice cuando la instrucción devuelve múltiples valores.

Ejemplo: Obtener las partes despachadas a un conjunto de clientes.

Podríamos pensar en tres posibilidades:

1. Accesar la base de datos cada vez que se ingrese un número de cliente. Por cada cliente obtener las partes despachadas.
2. Ingresar todos los clientes grabándolos en un archivo o relación. Luego accesar la base de datos por medio de un join entre la relación temporal y la de partes despachadas.
3. Obtener todos los registros (CNUM, PNUM) de las partes despachadas, generando una relación temporal. Luego, cada que se ingrese un número de cliente, accesar la relación temporal y obtener las partes despachadas a ese cliente.

RESTRICCIONES DE INTEGRIDAD EN BDDs

Las restricciones de integridad manejan dos aspectos:

1. Los valores permitidos en la base de datos
2. La integridad referencial, la cual es útil en bases de datos distribuidas para garantizar la exactitud de la fragmentación horizontal derivada.

EJERCICIOS RESUELTOS

Considere los siguientes esquemas:

Global:
EMPLEADO(NUMEMP, NOMBRE, DEPTO, SALARIO)
PROGRAMADOR(NUMEMP, LENGUAJE)

Fragmentación:
EMPLEADO1 = $\mathbf{SL}_{DEPTO \leq 20}$EMPLEADO
EMPLEADO2 = $\mathbf{SL}_{DEPTO > 20}$EMPLEADO
PROGRAMADOR1 = PROGRAMADOR $\mathbf{SJ}_{NUMEMP=NUMEMP}$EMPLEADO1
PROGRAMADOR2 = PROGRAMADOR $\mathbf{SJ}_{NUMEMP=NUMEMP}$EMPLEADO2

Asignación:
EMPLEADO1 EN SITIOS 1,2
EMPLEADO2 EN SITIOS 3,4
PROGRAMADOR1 EN SITIOS 5,6
PROGRAMADOR2 EN SITIOS 7,8

1. Codifique las instrucciones necesarias para que diga a qué departamento pertenece y que lenguaje de programación maneja el programador 125. Luego los escriba (write, print o output) en la estación de trabajo.

Con transparencia de fragmentación

```
SELECT lenguaje, depto INTO $leng, $dept
FROM empleado, programador
WHERE empleado.numemp = programador.numemp
AND programador.numemp = 125;
Write($dept, $leng);
```

Con transparencia de ubicación

```
SELECT depto INTO $dept
FROM empleado1
WHERE numemp = 125;
If #found then begin
        SELECT lenguaje INTO $leng FROM programador1
        WHERE numemp = 125;
End;
Else begin
        SELECT depto INTO $dept FROM empleado2
        WHERE numemp = 125;
        SELECT lenguaje INTO $leng   FROM programador2
        WHERE numemp = 125;
End;
Write($dept, $leng);
```

Con transparencia de mapeo local
(Se asume que la replicación es consistente)

```
SELECT depto INTO $dept FROM empleado1 AT SITE 1
WHERE numemp = 125;
If #found then begin
        SELECT lenguaje INTO $leng FROM programador1 AT SITE 5
        WHERE numemp = 125;
End;
Else begin
        SELECT depto INTO $dept FROM empleado2 AT SITE 3
        WHERE numemp = 125;
        SELECT lenguaje INTO $leng FROM programador2 AT SITE 7
        WHERE numemp = 125;
End;
Write($dept, $leng);
```

2. Modifique el departamento del empleado 147, del depto 08 al depto 25

Con transparencia de fragmentación

```
UPDATE empleado
SET depto = 25
WHERE numemp = 147;
```

Con transparencia de ubicación

```
SELECT nombre, salario, INTO $nom, $sal FROM empleado1
  WHERE numemp = 147;
INSERT INTO empleado2 VALUES (147, $nom, 25, $sal);
DELETE FROM empleado1
  WHERE numemp = 147;
SELECT lenguaje INTO $leng FROM programador1
  WHERE numemp = 147;
If #found then begin
        INSERT INTO programador2 VALUES (147, $leng);
        DELETE FROM programador1
          WHERE numemp = 147;
End;
```

Con transparencia de mapeo local

```
SELECT nombre, salario, INTO $nom, $sal FROM empleado1 AT SITE 1
  WHERE numemp = 147;
INSERT INTO empleado2 AT SITE 3 VALUES (147, $nom, 25, $sal);
```

```
INSERT INTO empleado2 AT SITE 4 VALUES (147, $nom, 25, $sal);
DELETE FROM empleado1 AT SITE 1
  WHERE numemp = 147;
DELETE FROM empleado1 AT SITE 2
  WHERE numemp = 147;
SELECT lenguaje INTO $leng FROM programador1 AT SITE 5
  WHERE numemp = 147;
If #found then begin
        INSERT INTO programador2 AT SITE 7 VALUES (147, $leng);
        INSERT INTO programador2 AT SITE 8 VALUES (147, $leng);
        DELETE FROM programador1 AT SITE 5
          WHERE numemp = 147;
        DELETE FROM programador1 AT SITE 6
          WHERE numemp = 147;
End;
```

EJERCICIOS PROPUESTOS

Considere los siguientes esquemas:

Esquema Global:

PROFESOR(ID, NOMBRE, PROGRAMA, PROFESION)
POSGRADUADO(ID, TITULO)

Fragmentación:

PROFESOR1 = $\mathbf{SL}_{PROGRAMA='biologia'}$PROFESOR
PROFESOR2 = $\mathbf{SL}_{PROGRAMA='matematicas'}$PROFESOR
PROFESOR3 = $\mathbf{SL}_{PROGRAMA='fisica'}$PROFESOR
POSGRADUADO1 = POSGRADUADO $\mathbf{SJ}_{ID=ID}$PROFESOR1
POSGRADUADO2 = POSGRADUADO $\mathbf{SJ}_{ID=ID}$PROFESOR2
POSGRADUADO3 = POSGRADUADO $\mathbf{SJ}_{ID=ID}$PROFESOR3

1. Elabore el programa que permita averiguar qué profesión (título de pregrado) posee y que título de posgrado tiene el posgraduado identificado con ID=21 (suponga que ya está validado). Luego escriba en la estación de trabajo dichos títulos.

    a. Con transparencia de fragmentación
    b. Con transparencia de ubicación

2. Actualice el programa del profesor identificado con el número 12, del programa **física** al programa **biología**.

a. Con transparencia de fragmentación
b. Con transparencia de ubicación

4

Diseño de Bases de Datos Distribuidas

El diseño de un sistema distribuido implica tomar decisiones relacionadas con la ubicación de los datos y los programas en los diferentes sitios de la red, incluso tomar decisiones sobre el diseño mismo de la red. De acuerdo con Tamer Öszu y Patrick Valduriez, en el caso de los DBMSs distribuidos, la distribución de aplicaciones involucra dos cosas: la distribución del software de la base de datos y la distribución de las aplicaciones que se ejecutan en él. La distribución del software no es un problema significativo ya que una copia de él puede existir en cada sitio de almacenamiento de datos y las aplicaciones surgen como necesidades de los usuarios en cada uno de los sitios.

En este capítulo se asume que la red ya se encuentra diseñada y nos concentraremos en el diseño de una base de datos distribuida, desde el punto de vista de la distribución de los datos.

Los problemas de diseño en bases de datos distribuidas son más cruciales que en bases de datos centralizadas. Las fases involucradas son las siguientes:

1. Diseño del esquema global de la base de datos, el cual describe la base de datos de una manera integrada, como si fuera centralizada. Es el esquema conceptual de la BDD.
2. Diseño de fragmentación, el cual permite determinar los tipos de fragmentación y la forma en que se fragmenta la base de datos.
3. Asignación de fragmentos, el cual establece los sitios donde residirán los fragmentos.
4. Diseño físico de las bases de datos locales, incluyendo las áreas de almacenamiento y los métodos de acceso.

La segunda y tercera fases caracterizan completamente el diseño de distribución. No es posible determinar estas dos actividades de manera óptica resolviéndolas independientemente, ya que están fuertemente interrelacionadas.

El conocimiento de los **requerimientos de aplicación** influencia el diseño para que este sea soportado eficientemente.

Los requerimientos de aplicación consisten básicamente de:

- ✓ El sitio origen de la aplicación
- ✓ La frecuencia de activación de la aplicación (en cada sitio)
- ✓ La cantidad de accesos, los tipos de acceso y la distribución estadística del acceso a los datos requeridos.

OBJETIVOS DEL DISEÑO DE DISTRIBUCIÓN DE DATOS

1. Localidad de procesamiento: Una vez los sitios de origen de las aplicaciones son conocidos, la localidad depende solo de la distribución de los datos.
2. Disponibilidad y confiabilidad de los datos distribuidos: Múltiples copias dan una alta disponibilidad y por ende la base de datos se vuelve más confiable.
3. Distribución de carga: Maximizando el grado de paralelismo de ejecución de aplicaciones.
4. Bajar costos de almacenamiento y disponibilidad: Pues estos no son relevantes comparados con los costos de CPU, de I/O y de transmisión.

ENFOQUES TOP-DOWN Y BOTTOM-UP EN EL DISEÑO DE DISTRIBUCIÓN

El enfoque top-down se aplica desde el esquema global hasta el diseño físico de los datos asignados a cada sitio. Se utiliza en sistemas desarrollados a partir de cero.

El enfoque bottom-up permite integrar bases de datos existentes, intercalando los datos comunes y resolviendo conflictos entre diferentes representaciones. En esencia requiere:

- ✓ Selección de un modelo de base de datos común para describir el esquema global.
- ✓ Traducción de cada esquema local en el modelo común seleccionado.
- ✓ Integración de los esquemas locales en un esquema global común.

4.1 Diseño de fragmentación de bases de datos

El objetivo de esta fase es diseñar fragmentos excluyentes que son *Unidades lógicas de asignación*.

Los fragmentos, como se mencionó en la sección de tipos de fragmentación, son grupos de tuplas o atributos con las mismas propiedades.

Es importante anotar que el sitio origen de cada aplicación (sitio donde se ejecuta) es relevante para determinar la propiedad de localidad.

DISEÑO DE FRAGMENTACIÓN HORIZONTAL PRIMARIA

Se requiere determinar un conjunto de predicados de selección disyuntos y completos.

Se requiere que los elementos de cada fragmento sean referenciados homogéneamente por todas las aplicaciones.

DEFINICIONES

Sea R una relación global.

1. Un **predicado simple** es un predicado expresado en la forma *Atributo REL Valor*, donde *REL* es un operador relacional (>, <, ≥, ≤, ≠).

2. Un **predicado mintérmino** $\mathbf{U}$ para un conjunto $\mathbf{R}$ de predicados simples es una conjunción de todos los predicados simples de $\mathbf{R}$, tomados en su forma natural o en su forma negada, Así:

   $$\mathbf{U} = \bigwedge_{p_i \in \mathbf{R}} p_i^*, \text{ donde } p_i^* = p_i \text{ ó } p_i^* = \text{NOT } p_i \text{ y } \mathbf{U} \neq \text{FALSO}$$

   Para un conjunto $\mathbf{R}$ de predicados simples se tienen varios predicados mintérmino. Ejemplo:

   Sea R = EMP(NumEmp, Nombre, Depto, Sal, Retención, Cargo)
   Sean D1 y D2 los dos únicos departamentos, de tal manera que al mencionar los departamentos diferentes a D1 hay una referencia implícita a D2.

   Sea $\mathbf{R}$ = {Depto='D1', Cargo='Programador'}

p_1: Depto='D1'	NOT p_1: Depto≠'D1'
p_2: Cargo='Programador'	NOT p_2: Cargo≠'Programador'

   Los predicados mintérmino para $\mathbf{R}$ son:

   $\mathbf{U_1}$: Depto='D1' AND Cargo='Programador'
   $\mathbf{U_2}$: Depto='D1' AND Cargo≠'Programador'
   $\mathbf{U_3}$: Depto≠'D1' AND Cargo='Programador'
   $\mathbf{U_4}$: Depto≠'D1' AND Cargo≠'Programador',

siempre y cuando Y_1, Y_2, Y_3 y Y_4 sean diferentes de FALSO, es decir, produzcan algún resultado.

3. Un **fragmento** es el conjunto de tuplas que satisfacen un mintérmino. Ejemplo: Sean la relación R, el conjunto de predicados simples $\mathbf{R}$ y los predicados mintérmino $\mathbf{U_i}$ como se definieron arriba. Estos cuatro predicados constituyen cuatro fragmentos de R.

4. Un **predicado relevante** es aquel predicado simple p_i de $\mathbf{R}$, tal que si existen al menos dos predicados mintérmino de $\mathbf{R}$, cuya expresión sólo difiere en el predicado p_i (es decir, natural en uno y negado en el otro, de resto hay igualdad) de modo que los fragmentos correspondientes son referenciados de manera diferente por al menos una aplicación.

   Supongamos las siguientes aplicaciones:

   A_{D1}, funciona en el departamento D1 y referencia los empleados de D1 con mayor probabilidad que los de D2.

   A_{D2}, funciona en el departamento D2 y accesa con más preferencia a los de D2.

   A_{PG}, requiere sólo los datos de los empleados que son programadores y se origina en cualquier sitio de la base de datos distribuida.

   Según lo anterior, Depto='D1' y Cargo='Programador' son relevantes. Cojamos dos predicados mintérmino:

   $\mathbf{U_1}$: Depto='D1' AND Cargo='Programador'
   $\mathbf{U_2}$: Depto='D1' AND Cargo≠'Programador'

   Su expresión sólo difiere en el predicado Cargo='Programador' (natural en $\mathbf{U_1}$ y negado en $\mathbf{U_2}$). Estos dos fragmentos son referenciados de manera diferente por la aplicación A_{PG}.

   En cambio si se tuviera:

   $\mathbf{U_{11}}$: Depto='D1' AND Cargo='Programador' AND Sal > 50
   $\mathbf{U_{12}}$: Depto='D1' AND Cargo='Programador' AND Sal ≤ 50

   Su expresión difiere en el predicado Sal > 50 pero los dos fragmentos son referenciados en la misma forma por todas las aplicaciones.

5. Un conjunto **R** de predicados es **completo** si y sólo si cualquier par de tuplas pertenecientes al mismo fragmento son referenciadas con la misma probabilidad por cualquier aplicación.

   El numeral anterior muestra, por algún supuesto, la división del fragmento $\mathbf{U_1}$ en $\mathbf{U_{11}}$ y $\mathbf{U_{12}}$. Vemos que al interior de cada fragmento de estos, todos los datos son referenciados con igual probabilidad, es decir, no hay preferencia por ningún dato.

   Según esto, **R** = {Depto='D1', Cargo='Programador', Sal > 50} generaría ocho fragmentos con la característica de ser completo.

6. Un conjunto **R** es **minimal** si todos sus predicados son relevantes (existe la relevancia para cada uno de sus predicados).

Si se quisiera fragmentar CLIENTE (CNUM, NOMBRE, CIUDAD) y del análisis de las aplicaciones resultaran los predicados siguientes:

p_1: Ciudad='Armenia'
p_2: Ciudad='Pereira'

se obtendrían los siguientes candidatos a predicados mintérmino:

$\mathbf{U_1}$: Ciudad='Armenia' AND Ciudad='Pereira'
$\mathbf{U_2}$: Ciudad='Armenia' AND Ciudad≠'Pereira'
$\mathbf{U_3}$: Ciudad≠'Armenia' AND Ciudad='Pereira'
$\mathbf{U_4}$: Ciudad≠'Armenia' AND Ciudad≠'Pereira'

Analizando las expresiones se concluye que $\mathbf{U_1}$ es contradictorio ya que un cliente no puede ser de dos ciudades a la vez. $\mathbf{U_4}$ también es contradictorio ya que un cliente tiene que ser por lo menos de alguna ciudad y estamos suponiendo un universo de solo dos ciudades.

En la segunda y en la tercera expresión debemos aplicar inferencias para eliminar redundancias para finalmente concluir que los predicados mintérmino deben ser:

$\mathbf{U_2}$: Ciudad='Armenia' (Ciudad='Armenia' ⇒ Ciudad≠'Pereira')
$\mathbf{U_3}$: Ciudad='Pereira' (Ciudad='Pereira' ⇒ Ciudad≠'Armenia')

DISEÑO DE FRAGMENTACIÓN HORIZONTAL DERIVADA

La fragmentación de una relación R no está basada en propiedades de sus propios atributos, pero es derivada de la fragmentación horizontal de otra relación.

La fragmentación derivada es utilizada para facilitar el join entre fragmentos. Aparece aquí el concepto de **join distribuido**, el cual es aplicado entre relaciones fragmentadas horizontalmente.

Un join distribuido es representado por un **grafo de join**. Un grafo de join G del join distribuido R **JN** S es un grafo <N,E>, donde N representa el conjunto de fragmentos de R y S; E es el conjunto de enlaces no dirigidos entre los nodos y representa los joins no vacíos entre fragmentos.

Tipos de grafos de join, para un join distribuido entre las relaciones R y S.

Grafo Total → Es aquel que tiene todos los enlaces posibles entre los fragmentos de R y los de S.

Grafo Reducido → Es aquel en el que existen algunos enlaces entre los fragmentos de R y S.

Grafo Particionado → Aquel en el que existen dos o más subgrafos sin enlaces entre ellos.

Grafo Simple → Es aquel grafo particionado en el que cada subgrafo tiene un enlace.

Todo grafo simple es un grafo particionado y, a su vez, todo grafo particionado es un grafo reducido.

DISEÑO DE FRAGMENTACIÓN VERTICAL

El diseño de fragmentación vertical de una relación R requiere del agrupamiento de los atributos en subconjuntos que sean referenciados **en la misma forma** por las aplicaciones.

Existe el **particionamiento vertical**, el cual establece que los subconjuntos deben ser disyuntos, mientras que el **agrupamiento vertical** permite que los conjuntos se puedan sobreponer. Este último introduce la replicación de valores que se sobreponen, la cual es una ventaja en aplicaciones *read-only*.

DISEÑO DE FRAGMENTACIÓN MIXTA

Las formas más sencillas de aplicar la fragmentación mixta son:

1. Aplicar fragmentación horizontal a fragmentos verticales. Si tenemos dos fragmentos verticales R_1 y R_2, uno de ellos (por ejemplo R_1) puede fragmentarse horizontalmente para obtener R_{11}, R_{12} y R_{13}.

<table>
<tr><td>R_{11}</td><td rowspan="3">R_2</td></tr>
<tr><td>R_{12}</td></tr>
<tr><td>R_{13}</td></tr>
</table>

2. Aplicar fragmentación vertical a fragmentos horizontales. Por ejemplo, si hay dos fragmentos horizontales S_1 y S_2, este último se le puede aplicar fragmentación vertical resultando en S_{21}, S_{22} y S_{23}.

<table>
<tr><td colspan="3">S_1</td></tr>
<tr><td>S_{21}</td><td>S_{22}</td><td>S_{23}</td></tr>
</table>

Aunque estas operaciones pueden ser repetidas de manera recursiva, llegar a tener más de dos niveles de fragmentación no es de interés en la práctica.

4.2 Asignación de fragmentos a los sitios

Para realizar esta actividad es fundamental decidir si la distribución va a ser redundante o no-redundante.

La replicación introduce complejidad adicional en el diseño ya que el grado de replicación de cada fragmento se convierte en una variable más del problema y, también, el proceso de seleccionar entre varios sitios alternativos no es una tarea fácil.

Para medir los costos y los beneficios en la asignación de fragmentos podemos utilizar las siguientes convenciones:

i	índice del fragmento (iésimo fragmento)
j	índice del sitio (jésimo sitio)
k	índice de la aplicación (késima aplicación)
f_{kj}	frecuencia de la aplicación k (A_k) en el sitio j.
r_{ki}	frecuencia de consultas de A_k al fragmento i.
u_{ki}	frecuencia de actualizaciones de A_k al fragmento i.
n_{ki}	$= r_{ki} + u_{ki}$
B_{ij}	beneficio de tener el fragmento i en el sitio j

C constante de costo (relación entre costo de una actualización y una consulta, $C \geq 1$)

Para determinar la asignación de fragmentos existen los siguientes tres métodos:

El sitio mejor (best-fit). Es el método más simple que mide el beneficio asociado con cada asignación posible del fragmento y, finalmente, selecciona el mejor. Es una asignación sin replicación. El beneficio de tener el fragmento i en el sitio j está dado por:

$$B_{ij} = \sum_{k} f_{kj} n_{ki}$$

Para cada fragmento i, se calculan los B_{ij} correspondientes a todos los sitios j. Finalmente se selecciona el sitio j* tal que B_{ij^*} sea máximo.

Todos los sitios benéficos (all benefical sites). Determina el conjunto de todos los sitios donde el beneficio de asignar una copia del fragmento es mayor que el costo.

$$B_{ij} = \sum_{k} f_{kj} r_{ki} - C \sum_{k} \sum_{j' \neq j} f_{kj'} u_{ki}$$

El fragmento i es ubicado en todos los sitios j* tales que B_{j^*} sea positivo.

Replicación adicional. Determina primero el sitio mejor, utilizando el primer método, y progresivamente introduce réplicas empezando en el más benéfico de los sitios restantes. El proceso termina cuando la replicación adicional que se haga no sea benéfica. En éste, la confiabilidad y la disponibilidad del sistema se incrementan si hay dos o tres copias del fragmento, pero con copias adicionales no da un incremento significativo, o el beneficio comienza a decrecer.

$$B_{ij} = \sum_{k} f_{kj} r_{ki} - C \sum_{k} \sum_{j' \neq j} f_{kj'} u_{ki} + \beta(d_i)$$

En este método, B_{ij} es el beneficio de ubicar una **nueva** copia de F_i, así:

El mejor B_{ij} es el beneficio de ubicar R_i en el mejor sitio, en cuyo caso el grado de redundancia de Ri es 1 (es decir, d_i=1) y $\beta(d_i)$=0.

El segundo B_{ij} es el beneficio de ubicar R_i en el segundo mejor sitio, el grado de redundancia de R_i es 2 (es decir, d=2) y $\beta(d_i)$=otro valor que puede ser positivo o negativo.

En el momento en que B_{ij} se haga negativo, se descarta ese sitio j.

Los B_{ij} se calculan parcialmente (como si fueran all benefical sites, es decir, sin sumar los β's) y luego al mejor B_{ij} se le suma β(1), al segundo mejor, se le suma β(2) y así sucesivamente. Ejemplo: sean los fragmentos R_1, R_2 y R_3, podríamos tener los siguientes βs:

Para R_1, β(1)= 0, β(2)= -2, β(3)= -3
Para R_2, β(1)= 0, β(2)= -2, β(3)= -3
Para R_3, β(1)= 0, β(2)= -4, β(3)= -6

EJERCICIOS RESUELTOS

Sean los fragmentos R_1, R_2 y R_3. Sean las aplicaciones A1, A2 y A3, las cuales residen en los sitios S1, S2 y S3 respectivamente. La aplicación **A1** se ejecuta 4 veces al día, la **A2** 1 vez y **A3** 2 veces; además una actualización cuesta 2 veces una consulta. Dónde deben quedar los fragmentos, usando replicación y adicional sabiendo lo siguiente?

A1 lee 5 tuplas de $\mathbf{R_1}$, lee 3 tuplas de $\mathbf{R_2}$, lee 6 de $\mathbf{R_3}$ y actualiza 2 tuplas de $\mathbf{R_3}$
A2 lee 17 tuplas de $\mathbf{R_1}$, actualiza 1 tuplas de $\mathbf{R_2}$ y actualiza 1 tupla de $\mathbf{R_3}$
A3 lee 4 tuplas de $\mathbf{R_1}$, lee 2 tuplas de $\mathbf{R_2}$ y lee 12 tuplas de $\mathbf{R_3}$ y actualiza 4 de $\mathbf{R_1}$

Los valores de β son los especificados arriba:

Para R_1, β(1)=0, β(2)=-2, β(3)=-3
Para R_2, β(1)=0, β(2)=-2, β(3)=-3
Para R_3, β(1)=0, β(2)=-4, β(3)=-6

a) B_{11} = **4***5 – 2(**2***4) + β(2) = 4-2 = 2
B_{12} = **1***17 – 2(**2***4) + β(3)= 1-3 = -2
B_{13} = **2***4 – 2(0) + β(1) = 8+0 = 8
b) B_{21} = **4***3 – 2(**1***1) + β(1) = 10+0 = 10
B_{22} = **1***0 – 2(0) + β(3) = 0-3 = -3
B_{23} = **2***2 – 2(**1***1) + β(2) = 2-2 = 0
c) B_{31} = **4***6 – 2(**1***1) + β(1) = 22+0 = 22
B_{32} = **1***0 – 2(**4***2) + β(3) = -16-6 = -22
B_{33} = **2***12 – 2(**4***2+**1***1) + β(2) = 6-4 = 2

d) R_1 en sitios = S1, S3
e) R_2 en sitios = S1
f) R_3 en sitios = S1, S3

EJERCICIOS PROPUESTOS

1. Sea el siguiente esquema global:

DEPTO(ID, NOMBRE, REGION)
MPIO(CODIGO, NOMBRE, CATEG, ID)

Defina la fragmentación de la relación MPIO, basándose en las siguientes aplicaciones, especificando los predicados mintérmino. Hay dos grandes regiones, la norte y la sur.

a) Dos aplicaciones de manejo de proyectos regionales (**AN** y **AS**) donde cada una maneja información de su propia región con una probabilidad del 75%.
b) Una aplicación de desarrollo industrial (**AI**) que maneja municipios de categoría 5 (El rango de categorías es de 1 a 5), además la región sur no tiene municipios de categoría 5.

2. Mediante la conjunción de predicados y utilizando implicaciones defina la fragmentación de la relación EMP para las siguientes aplicaciones de una organización:

   a. Una aplicación de nómina de empleados de planta.
   b. Una aplicación de nómina de contratistas. Estos normalmente ganan menos de 1'400.000
   c. Una aplicación de mantenimiento de equipos la cual referencia los empleados del departamento de mantenimiento. Todos los integrantes de este departamento son de planta.
   d. Una aplicación de bienestar para los hijos de los empleados.

La relación EMP consta de los siguientes atributos:
CodEmp, Nombre, Tipo, Depto, Salario, Hijos.

El Tipo tiene dos valores posibles: 'P'= Planta y 'C'=Contratista
El Depto es una cadena de caracteres
Hijos indica la cantidad de hijos del funcionario.

5

Descomposición de Consultas Globales en Fragmentadas

La principal función de un procesador de consultas relacionales es transformar una consulta global en un esquema de más bajo nivel equivalente que permita accesar cada uno de los fragmentos involucrados. La consulta de bajo nivel realmente implementa la estrategia de ejecución de la consulta global. La transformación debe lograr no solo exactitud sino también eficiencia. Para que sea correcta, la consulta fragmentada debe tener la misma semántica de la consulta original, es decir, ambas consultas deben producir el mismo resultado. Pero producir una estrategia de ejecución eficiente va más allá. Pues cada estrategia de ejecución equivalente puede llevar a suposiciones muy diferentes sobre los recursos computarizados. La principal dificultad consiste en seleccionar la estrategia de ejecución que minimice el consumo de recursos.

Este capítulo se entiende con la descomposición de una consulta global en consultas sobre fragmentos en la que, básicamente, se busca que la transformación sea correcta, aunque, alguna optimización se tendrá en cuenta.

5.1 Transformaciones de equivalencia para consultas

Una consulta puede lograrse mediante el uso de diferentes lenguajes, pero para nuestro propósito, se utilizarán el Álgebra Relacional y el SQL.

Hay dos aspectos que deben tenerse en cuenta: la **semántica**, la cual se entiende con el significado del programa, es decir, lo que debe hacer, y la **secuencia de operaciones**, la que se entiende con la forma de lograrlo. Lo que quiere decir que dos expresiones con la misma semántica pueden tener dos secuencias de operaciones diferentes.

Por facilidad en el entendimiento y en la manipulación, se utilizarán **árboles de operadores**, los cuales muestran la operación principal en la raíz y en las ramas los fragmentos.

La siguiente es una lista de transformaciones de equivalencia para el álgebra relacional, donde el símbolo "→" se utilizará para decir "*se puede transformar en*". Además, cuando haya alguna restricción para que la transformación sea válida, ésta se especificará al frente, mientras que cuando no haya restricción, no se especificará nada. Supongamos que R, S y T son relaciones.

1. $\mathbf{SL}_{F1}(\mathbf{SL}_{F2}(R)) \rightarrow \mathbf{SL}_{F2}(\mathbf{SL}_{F1}(R))$
2. $\mathbf{SL}_{F1}(\mathbf{PJ}_{A2}(R)) \rightarrow \mathbf{PJ}_{A2}(\mathbf{SL}_{F1}(R))$
3. $\mathbf{PJ}_{A1}(\mathbf{SL}_{F2}(R)) \rightarrow \mathbf{SL}_{F2}(\mathbf{PJ}_{A1}(R))$ $\quad$ $Attr(F2) \subseteq A1$
4. $\mathbf{PJ}_{A1}(\mathbf{PJ}_{A2}(R)) \rightarrow \mathbf{PJ}_{A2}(\mathbf{PJ}_{A1}(R))$ $\quad$ $A1 \equiv A2$
5. $R\ \mathbf{UN}\ S \rightarrow S\ \mathbf{UN}\ R$
6. $R\ \mathbf{CP}\ S \rightarrow S\ \mathbf{CP}\ R$
7. $R\ \mathbf{JN}_F\ S \rightarrow S\ \mathbf{JN}_F\ R$

   DF y JN no son conmutativas
8. $(R\ \mathbf{UN}\ S)\ \mathbf{UN}\ T \rightarrow R\ \mathbf{UN}\ (S\ \mathbf{UN}\ T)$
9. $(R\ \mathbf{CP}\ S)\ \mathbf{CP}\ T \rightarrow R\ \mathbf{CP}\ (S\ \mathbf{CP}\ T)$
10. $(R\ \mathbf{JN}_{F1}\ S)\ \mathbf{JN}_{F2}\ T \rightarrow R\ \mathbf{JN}_{F1}\ (S\ \mathbf{JN}_{F2}\ T)$ $\quad$ $Attr(F2) \subseteq Attr(S) \cup Attr(T)$

    DF y SJ no son asociativas
11. $\mathbf{PJ}_A(R) \rightarrow \mathbf{PJ}_{A1}\mathbf{PJ}_{A2}(R)$ $\quad$ $A \equiv A1,\ A \subseteq A2$
12. $\mathbf{SL}_F(R) \rightarrow \mathbf{SL}_{F1}\mathbf{SL}_{F2}(R)$ $\quad$ $F = F1$ AND $F2$
13. $\mathbf{SL}_F(R\ \mathbf{UN}\ S) \rightarrow \mathbf{SL}_{FR}(R)\ \mathbf{UN}\ \mathbf{SL}_{FS}(S)$ $\quad$ $FR=F,\ FS=F$
14. $\mathbf{SL}_F(R\ \mathbf{DF}\ S) \rightarrow \mathbf{SL}_{FR}(R)\ \mathbf{DF}\ \mathbf{SL}_{FS}(S)$ $\quad$ $FR=F$
15. $\mathbf{SL}_F(R\ \mathbf{CP}\ S) \rightarrow \mathbf{SL}_{FR}(R)\ \mathbf{CP}\ \mathbf{SL}_{FS}(S)$

    $\exists$ FR, FS: (F=FR AND FS), $Attr(FR) \subseteq R$, $Attr(FS) \subseteq S$
16. $\mathbf{SL}_F(R\ \mathbf{JN}_{FJ}\ S) \rightarrow \mathbf{SL}_{FR}(R)\ \mathbf{JN}_{FJ}\ \mathbf{SL}_{FS}(S)$

    $\exists$ FR, FS: (F=FR AND FS), $Attr(FR) \subseteq R$, $Attr(FS) \subseteq S$
17. $\mathbf{SL}_F(R\ \mathbf{SJ}_{FJ}\ S) \rightarrow \mathbf{SL}_{FR}(R)\ \mathbf{SJ}_{FJ}\ \mathbf{SL}_{FS}(S)$ $\quad$ $FR=F,\ FS=True$
18. $\mathbf{PJ}_A(R\ \mathbf{UN}\ S) \rightarrow \mathbf{PJ}_{AR}(R)\ \mathbf{UN}\ \mathbf{PJ}_{AS}(S)$ $\quad$ $AR=A,\ AS=A$
19. $\mathbf{PJ}_A(R\ \mathbf{CP}\ S) \rightarrow \mathbf{PJ}_{AR}(R)\ \mathbf{CP}\ \mathbf{PJ}_{AS}(S)$

    $AR=A-Attr(S),\ AS=A-Attr(R)$
20. $\mathbf{PJ}_A(R\ \mathbf{JN}_{FJ}\ S) \rightarrow \mathbf{PJ}_{AR}(R)\ \mathbf{JN}_{FJ}\ \mathbf{PJ}_{AS}(S)$

    $AR=A-Attr(S),\ AS=A-Attr(R),\ Attr(FJ) \subseteq A$
21. $\mathbf{PJ}_A(R\ \mathbf{SJ}_{FJ}\ S) \rightarrow \mathbf{PJ}_{AR}(R)\ \mathbf{SJ}_{FJ}\ \mathbf{PJ}_{AS}(S)$

    $AR=A,\ AS=Attr(S) \cap Attr(FJ)$

    PJ no es distributiva con respecto a DF
22. $\mathbf{SL}_{FR}(R)\ \mathbf{UN}\ \mathbf{SL}_{FS}(S) \rightarrow \mathbf{SL}_F(R\ \mathbf{UN}\ S)$ $\quad$ $F=FR,\ F=FS$
23. $\mathbf{SL}_{FR}(R)\ \mathbf{DF}\ \mathbf{SL}_{FS}(S) \rightarrow \mathbf{SL}_F(R\ \mathbf{DF}\ S)$ $\quad$ $F=FR,\ F=FS$
24. $\mathbf{SL}_{FR}(R)\ \mathbf{CP}\ \mathbf{SL}_{FS}(S) \rightarrow \mathbf{SL}_F(R\ \mathbf{CP}\ S)$ $\quad$ $F=FR$ AND FS
25. $\mathbf{SL}_{FR}(R)\ \mathbf{JN}_{FJ}\ \mathbf{SL}_{FS}(S) \rightarrow \mathbf{SL}_F(R\ \mathbf{JN}_{FJ}\ S)$ $\quad$ $F=FR$ AND FS
26. $\mathbf{SL}_{FR}(R)\ \mathbf{SJ}_{FJ}\ \mathbf{SL}_{FS}(S) \rightarrow \mathbf{SL}_F(R\ \mathbf{SJ}_{FJ}\ S)$ $\quad$ $F=FR,\ FS=True$
27. $\mathbf{PJ}_{AR}(R)\ \mathbf{UN}\ \mathbf{PJ}_{AS}(S) \rightarrow \mathbf{PJ}_A(R\ \mathbf{UN}\ S)$

    $A=AR,\ A=AS,\ Attrib(R)=Attrib(S)$
28. $\mathbf{PJ}_{AR}(R)\ \mathbf{CP}\ \mathbf{PJ}_{AS}(S) \rightarrow \mathbf{PJ}_A(R\ \mathbf{CP}\ S)$ $\quad$ $A=AR \cup AS$
29. $\mathbf{PJ}_{AR}(R)\ \mathbf{JN}_{FJ}\ \mathbf{PJ}_{AS}(S) \rightarrow \mathbf{PJ}_A(R\ \mathbf{JN}_{FJ}\ S)$ $\quad$ $A=AR \cup AS$
30. $\mathbf{PJ}_{AR}(R)\ \mathbf{SJ}_{FJ}\ \mathbf{PJ}_{AS}(S) \rightarrow \mathbf{PJ}_A(R\ \mathbf{SJ}_{FJ}\ S)$ $\quad$ $A=AR$

    PJ no es factorizable sobre DF
31. $(R'\ \mathbf{UN}\ R'')\mathbf{JN}_{FJ}(S'\ \mathbf{UN}\ S'') \leftrightarrow ((R'\mathbf{JN}_{FJ}S')\ \mathbf{UN}\ (R'\mathbf{JN}_{FJ}S'')\ \mathbf{UN}\ (R''\mathbf{JN}_{FJ}S')\ \mathbf{UN}\ (R''\mathbf{JN}_{FJ}S''))$

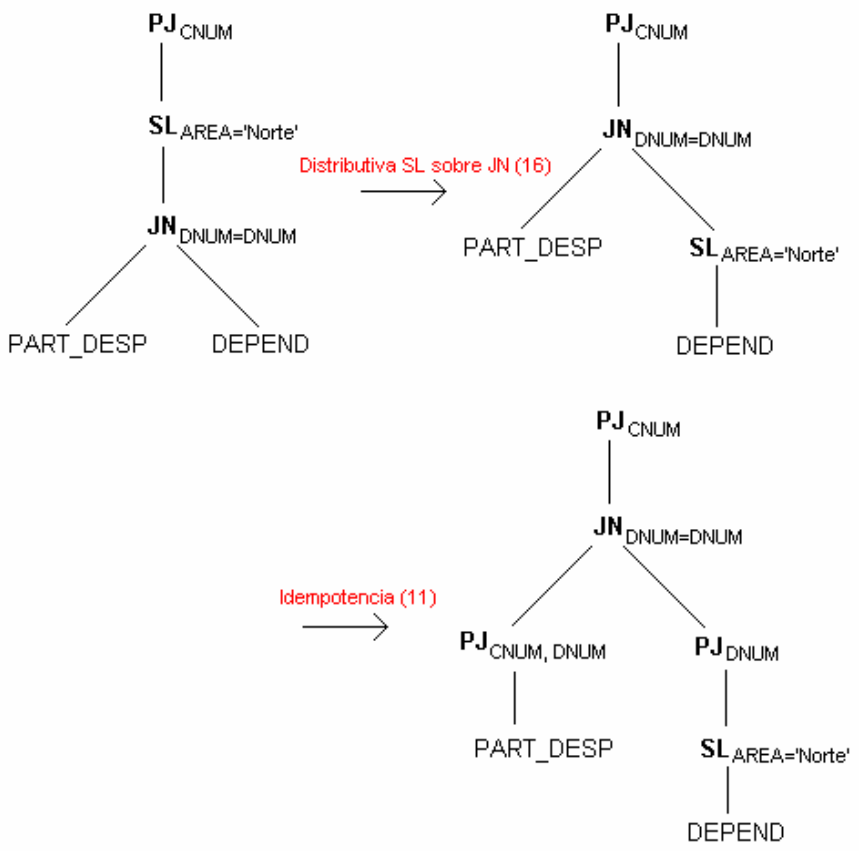
Los criterios de transformación en bases de datos centralizadas también son válidos en bases de datos distribuidas. Entre ellos tenemos los siguientes:

Criterio 1: Usar idempotencia de selección y proyección para generar selecciones y proyecciones apropiadas para cada relación operando.

Criterio 2: Empujar hacia abajo en el árbol de operadores, las selecciones y las proyecciones tanto como sea posible.

Ejemplo 1: Obtener los números de cliente que han recibido partes de las dependencias de la zona norte.

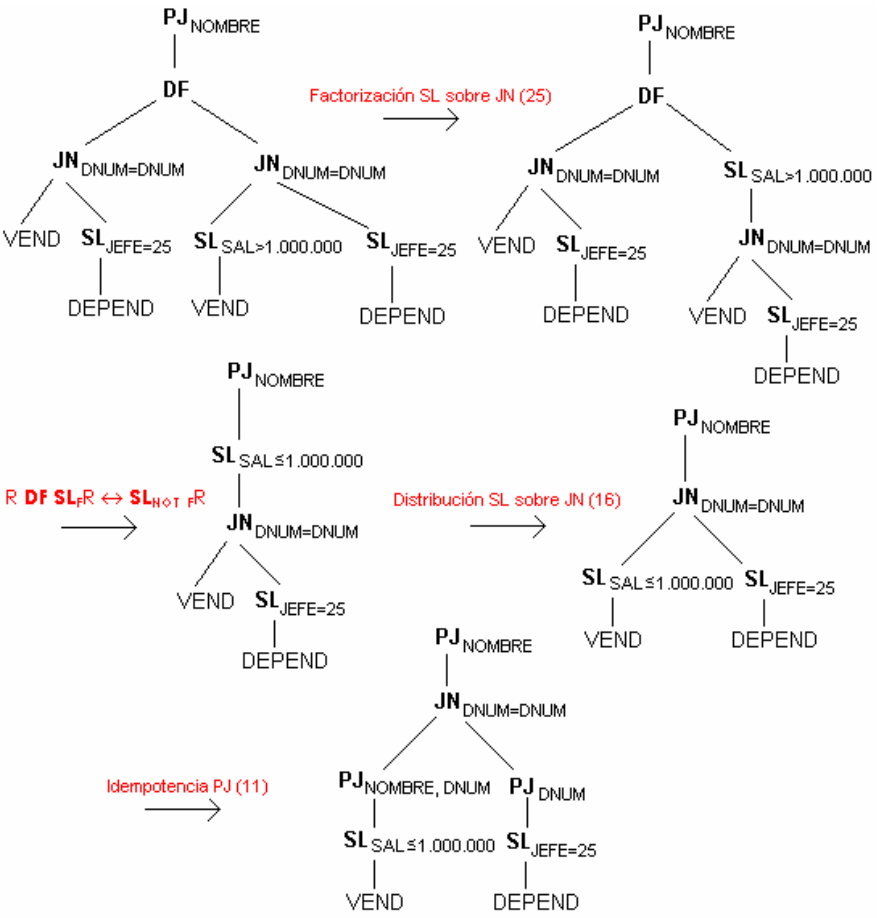
$\mathbf{PJ}_{CNUM}\,\mathbf{SL}_{ZONA='Norte'}(PART_DESP\,\mathbf{JN}_{DNUM=DNUM}DEPEND)$



Grafos de operadores y determinación de sub-expresiones comunes

Una forma de evitar que ciertas expresiones se evalúen más de una vez es eliminándolas donde sean redundantes. **Ejemplo 2**: Obtengamos los nombres de los vendedores cuyo jefe sea el número 25 pero que no ganen más de un millón de pesos.

$\mathbf{PJ}_{NOMBRE}$((VEND $\mathbf{JN}_{DNUM=DNUM}$ $\mathbf{SL}_{JEFE=25}$DEPEND) **DF**
($\mathbf{SL}_{SAL>1.000.000}$VEND $\mathbf{JN}_{DNUM=DNUM}$ $\mathbf{SL}_{JEFE=25}$DEPEND))



Existen una serie de propiedades que pueden utilizarse (como en el ejercicio anterior) una vez hayamos encontrado las sub-expresiones comunes. Ellas son:

1. R **NJN** R ↔ R

2. R **UN** R $\leftrightarrow$ R
3. R **DF** R $\leftrightarrow \varnothing$
4. R **NJN** $\mathbf{SL}_F$R $\leftrightarrow$ $\mathbf{SL}_F$R
5. R **UN** $\mathbf{SL}_F$R $\leftrightarrow$ R
6. R **DF** $\mathbf{SL}_F$R $\leftrightarrow$ $\mathbf{SL}_{\text{NOT } F}$R
7. ($\mathbf{SL}_{F1}$R) **NJN** ($\mathbf{SL}_{F2}$R)$\leftrightarrow$ $\mathbf{SL}_{F1 \text{ AND } F2}$R
8. ($\mathbf{SL}_{F1}$R) **UN** ($\mathbf{SL}_{F2}$R)$\leftrightarrow$ $\mathbf{SL}_{F1 \text{ OR } F2}$R
9. ($\mathbf{SL}_{F1}$R) **DF** ($\mathbf{SL}_{F2}$R)$\leftrightarrow$ $\mathbf{SL}_{F1 \text{ AND NOT } F2}$R

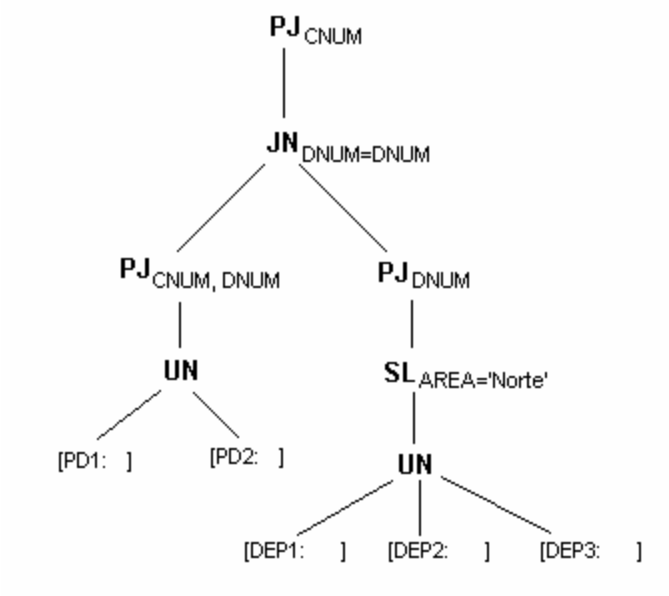
5.2 Transformación de consultas globales en fragmentadas

Una expresión algebraica sobre una consulta global se puede transformar en una expresión algebraica sobre una consulta fragmentada.

Una **expresión canónica de una consulta fragmentada** es aquella expresión algebraica sobre el esquema global a la que se le sustituyen los nombres de las relaciones globales por las expresiones que las reconstruyen.

El **árbol de operadores de una consulta** es el árbol de operadores de la expresión canónica de la consulta.

Ejemplo 3: Representar mediante un árbol de operadores de una consulta fragmentada la consulta del ejemplo 1: obtener los números de cliente que han recibido partes de las dependencias de la zona norte.



En la ilustración presentada, en cada hoja del árbol se representa un fragmento y entre corchetes debe especificarse la definición del fragmento (aunque no se hizo por razones de espacio). A esto se le llama la relación cualificada y se denota por el par [R:q_R], donde R es el cuerpo y q_R es la cualificación.

En el ejemplo anterior se tienen las siguientes relaciones cualificadas:

[PD1: CNUM=CLIENTE.CNUM AND CLIENTE.CIUDAD='Armenia']
[PD2: CNUM=CLIENTE.CNUM AND CLIENTE.CIUDAD='Pereira']
[DEP1: DNUM $\leq$ 5]
[DEP2: 5<DNUM $\leq$ 10]
[DEP3: DNUM > 10]

El **algebra de relaciones cualificadas** es una extensión del álgebra relacional que usa relaciones cualificadas como operandos. Las relaciones cualificadas resultan ser fragmentos horizontales.

Entre las reglas sobre relaciones cualificadas se tienen:

1. **SL**$_F$ [R:q_R] → [**SL**$_F$R: q_R]
2. **PJ**$_A$ [R:q_R] → [**PJ**$_A$R: q_R]
3. [R:q_R] **CP** [S:q_S] → [R **CP** S: q_R AND q_S]
4. [R:q_R] **DF** [S:q_S] → [R **DF** S: q_R]
5. [R:q_R] **UN** [S:q_S] → [R **UN** S: q_R OR q_S]

6. $[R:q_R]$ $\mathbf{JN}_F$ $[S:q_S]$ → $[R\ \mathbf{JN}_F\ S:\ q_R \text{ AND } q_S \text{ AND } F]$
7. $[R:q_R]$ $\mathbf{SJ}_F$ $[S:q_S]$ → $[R\ \mathbf{SJ}_F\ S:\ q_R \text{ AND } q_S \text{ AND } F]$

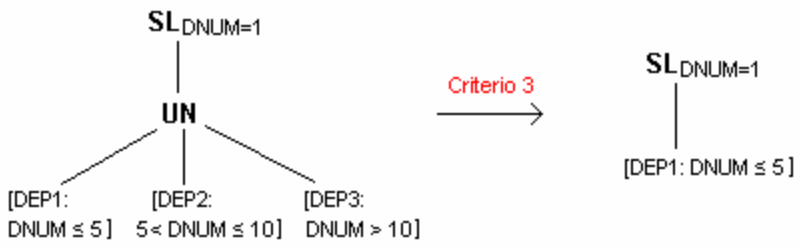
Nuevos criterios sobre transformación teniendo en cuenta el álgebra de relaciones cualificadas:

Criterio 3: Empujar las selecciones hacia las hojas del árbol, y luego aplicar el álgebra de relaciones cualificadas. Cuando el resultado de una cualificación sea contradictorio, debe reemplazarse por la relación vacía.

Criterio 4: Usar el álgebra de relaciones cualificadas para evaluar la cualificación de operandos de joins. Si la cualificación del resultado del join es contradictoria, sustituir el subárbol que contiene el join y sus operandos con la relación vacía.

Simplificación de Relaciones Fragmentadas Horizontalmente

Para obtener la información de la dependencia número uno, la expresión es $\mathbf{SL}_{DNUM=1}$DEPT, lo representamos en su forma canónica y mediante el uso del criterio 3 se simplifica.

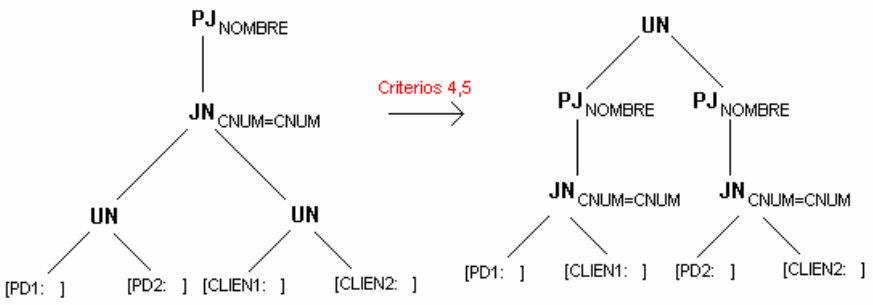


Simplificación de Joins entre Relaciones Fragmentadas Horizontalmente

Criterio 5: Para distribuir joins que aparecen en el query global, las uniones (que representan colecciones de fragmentos), deben ser llevadas hacia arriba, más allá de los joins que se pretenden distribuir.

Ejemplo 4: Obtener los nombres de clientes a los que se han despachado partes. Para esto, la expresión algebraica es la siguiente:

$\mathbf{PJ}_{NOMBRE}$ (PART_DESP **NJN** CLIENTE)



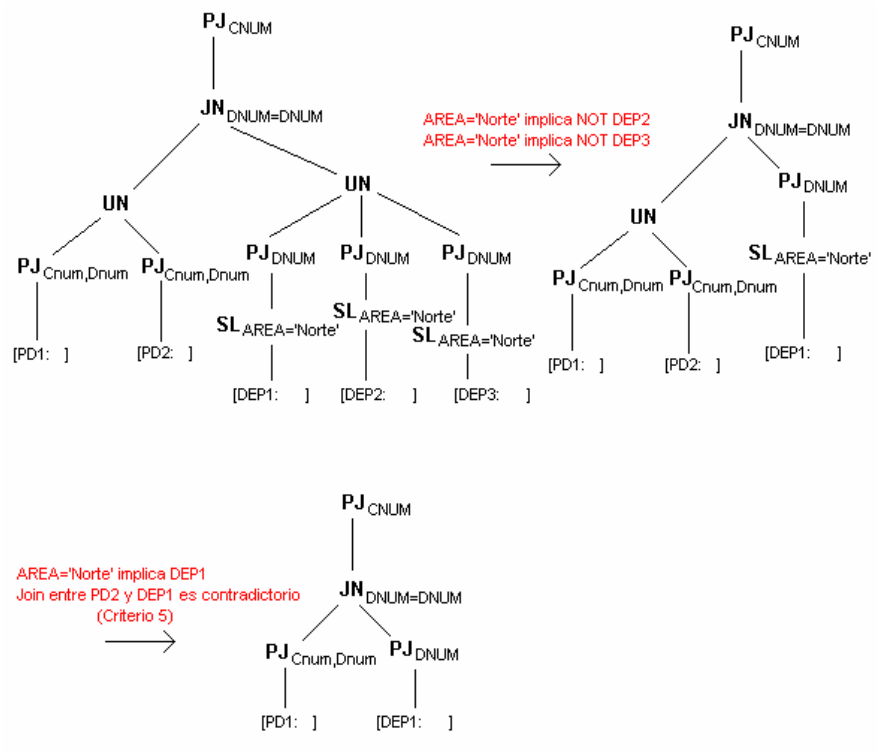
De la forma canónica de la consulta hemos pasado al join distribuido de la consulta.

Uso de inferencia para simplificaciones adicionales

Supongamos que la zona norte incluye las dependencias 1 a 5 y que estas dependencias despachan a clientes de Pereira.

El objetivo, en este caso, es inferir contradicciones que permitan eliminar sub-expresiones.

Ejemplo 5: Listar los números de cliente a los que se han suministrado partes provenientes de la zona norte.

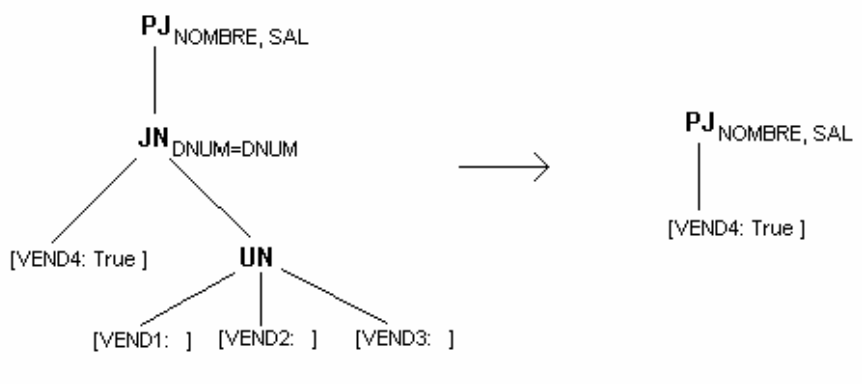


Simplificación de Relaciones Fragmentadas Verticalmente

El fondo de esta simplificación consiste en determinar un subconjunto de fragmentos que sea suficiente para responder a una consulta, eliminando los otros fragmentos de la expresión de la consulta, como también los joins (no necesarios) usados para reconstruir la relación global. Pues si los fragmentos requeridos se reducen a sólo uno, no hay necesidad de ejecutar operaciones tipo join.

Ejemplo 6: obtengamos los nombre y los salarios de los vendedores y luego simplifiquemos:

$\mathbf{PJ}_{\text{NOMBRE, SAL}}$VEND



Pues los atributos requeridos, nombre y salario, están en el fragmento VEND4, por consiguiente los demás se eliminan.

ALGORITMOS BASADOS EN SEMI-JOIN

La operación semijoin puede ser usada para decrementar el tiempo total de consultas con joins. La principal desventaja de las operaciones con joins sobre relaciones en diferentes sitios es que las relaciones operando deben ser transferidas completamente entre los sitios.

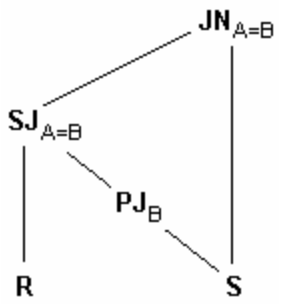
El join sobre dos relaciones R y S sobre el atributo A en R y B en S, almacenadas en los sitios 1 y 2 respectivamente, puede ser realizada reemplazando una o las dos relaciones por un semijoin con la otra relación:

$$
\begin{aligned}
R\ \mathbf{JN}_{A=B}\ S \quad & \leftrightarrow (R\ \mathbf{SJ}_{A=B}\ S)\ \mathbf{JN}_{A=B}\ S \\
& \leftrightarrow (R\ \mathbf{SJ}_{A=B}\ \mathbf{PJ}_{B} S)\ \mathbf{JN}_{A=B}\ S \qquad (1) \\
& \leftrightarrow R\ \mathbf{JN}_{A=B}\ (S\ \mathbf{SJ}_{B=A}\ R) \\
& \leftrightarrow R\ \mathbf{JN}_{A=B}\ (S\ \mathbf{SJ}_{B=A}\ \mathbf{PJ}_{A} R) \qquad (2) \\
& \leftrightarrow (R\ \mathbf{SJ}_{A=B}\ S)\ \mathbf{JN}_{A=B}\ (S\ \mathbf{SJ}_{B=A}\ R) \\
& \leftrightarrow (R\ \mathbf{SJ}_{A=B}\ \mathbf{PJ}_{B} S)\ \mathbf{JN}_{A=B}\ (S\ \mathbf{SJ}_{B=A}\ \mathbf{PJ}_{A} R) (3)
\end{aligned}
$$

La proyección utilizada ya que si, observando en (1), S le transmite a R su columna B, está minimizando la transmisión. Pues B es lo único que necesita A para poder hacer semijoin.

La selección entre una de las tres estrategias de semijoin requiere estimar sus respectivos costos.

En el caso de (1) sólo las tuplas de R que sobreviven al semijoin son transmitidas a S.



5.3 Agrupamiento Distribuido y Evaluación de Función Agregada

Consiste en agrupar tuplas en subconjuntos disyuntos de relaciones y hacer la evaluación de funciones agregadas sobre ellos. Lo denotaremos en el álgebra relacional mediante la expresión $\mathbf{GB}_{G,AF}R$, donde GB significa group by, G es el atributo o atributos objeto de agrupamiento y AF es la función a evaluar. Consideremos las siguientes instrucciones SQL con su correspondiente expresión en álgebra relacional:

1. SELECT AVG(cant) FROM part_desp
   WHERE pnum='p14';

   $\mathbf{GB}_{AVG(cant)}\ \mathbf{SL}_{PNUM='p14'}$ part_desp

2. SELECT cnum,pnum,SUM(cant) FROM part_desp
   GROUP BY cnum,pnum;

   $\mathbf{GB}_{cnum,pnum,SUM(cant)}$ part_desp

3. SELECT cnum,pnum,SUM(cant) FROM part_desp
   GROUP BY cnum,pnum
   HAVING SUM(cant)>300;

   $\mathbf{SL}_{SUM(cant)>300}\mathbf{GB}_{cnum,pnum,SUM(cant)}$ part_desp

En este último caso, la operación interna es el agrupamiento, la cual genera una tabla temporal y como tal se le hace una operación de selección (equivalente al *having* en SQL).

Propiedades de la operación GROUP BY

$\mathbf{GB}_{G,AF}$ (R1 **UN** R2) → ($\mathbf{GB}_{G,AF}$ R1) **UN** ($\mathbf{GB}_{G,AF}$ R2), $\forall i,j\ Gi \subseteq Rj$ ó $Gi \cap Rj = \varnothing$

Ejemplo 7: Sea la relación PART_DESP, tal como se definió antes. Supongamos que se ha fragmentado en R1 y R2, así:

R1			
cnum	**pnum**	**dnum**	**cant**
C1	P14	25	10
C1	P14	28	2
C1	P27	25	10

R2			
cnum	**pnum**	**dnum**	**cant**
C2	P14	8	5
C2	P14	9	10
C2	P29	4	6

Los grupos son:

G1: C1 P14 10
C1 P14 2

G2: C1 P27 10

G3: C2 P14 5
C2 P14 10

G4: C2 P29 6

Al analizar las contenencias e intersecciones obtenemos:

$G1 \subseteq R1$, $G1 \cap R2 = \emptyset$, $G3 \subseteq R2$, $G3 \cap R1 = \emptyset$, ..., lo que nos permite concluir que la transformación se puede aplicar.

Ejemplo 8: Sea la relación Envíos basada fragmentada de la siguiente manera:

ENV1			
prov	**prod**	**cant**	**ciudad**
S1	P1	10	Bogotá
S1	P1	2	Pereira
S2	P1	8	Bogotá
S2	P1	6	Cartagena

ENV2			
prov	**prod**	**cant**	**ciudad**
S1	P1	5	Madrid
S1	P1	3	Cádiz
S1	P2	10	Valencia
S2	P1	1	Madrid

Los grupos son:

G1: S1 P1 10
S1 P1 2
S1 P1 5
S1 P1 3

G2: S2 P1 8
S2 P1 6
S2 P1 1

G3: S1 P2 10

La propiedad no se puede aplicar ya que, por ejemplo, ni $G1 \subseteq ENV1$ ni $G1 \cap ENV1 = \emptyset$.

Ejercicio propuesto:

Supongamos que se tienen las relaciones:

MPIOS(COD_MPIO,NOMBRE,COD_DPTO), y los fragmentos:

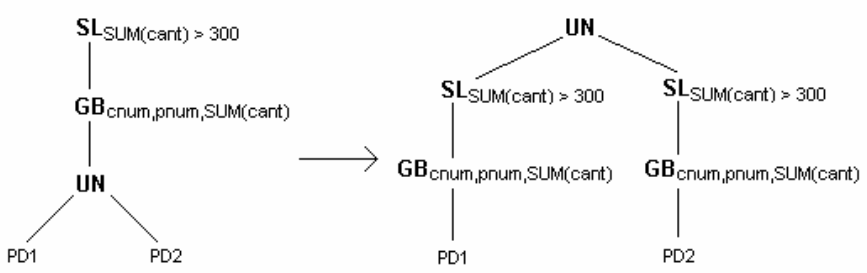
MPIO1: los municipios de la región Andina.
MPIO2: los municipios de la región Atlántica.

Verificar la aplicabilidad de esta propiedad enunciada del GROUP BY.

Criterio 6 Para distribuir agrupamiento y evaluaciones de función agregada en una relación global, las uniones (que representan conjuntos de fragmentos) deben ser llevadas hacia arriba, más allá de la operación de grupo correspondiente.

Ejemplo 9: Para cada producto despachado a cada cliente, calcular la suma total despachada hasta el momento, con la condición de listar las sumas superiores a 300 productos.

$\mathbf{SL}_{SUM(cant)>300}\mathbf{GB}_{cnum,pnum,SUM(cant)}$ (PD1 **UN** PD2) →
($\mathbf{SL}_{SUM(cant)>300}\mathbf{GB}_{cnum,pnum,SUM(cant)}$ PD1) **UN**
($\mathbf{SL}_{SUM(cant)>300}\mathbf{GB}_{cnum,pnum,SUM(cant)}$ PD2).



COMPUTACIÓN DISTRIBUIDA DE UNA FUNCIÓN AGREGADA F

Una función agregada F tiene una computación distribuida si para cualquier multiconjunto S y cualquier descomposición de S en multiconjuntos S_1, S_2, ..., S_n, es posible determinar un conjunto de funciones agregadas F_1, F_2, ..., F_m, y una expresión $E(F_1,F_2,...,F_m)$, tales que:

$F(S) = E(F_1(S_1), ...F_1(S_n), F_2(S_1), ..., F_m(S_n))$

Algunas funciones de computación distribuida son, por ejemplo:

$min(s)=min(min(s_1),... min(s_N))$
$max(s)=max(max(s_1),... max(s_N))$
$sum(s)=sum(sum(s_1),... sum(s_N))$
$count(s)= sum(count(s_1),... count(s_N))$

$avg(s)=sum(sum(s_1),... sum(s_N))/ sum(count(s_1),... count(s_N))$

La computación distribuida de las funciones agregadas es una propiedad importante ya que los resultados parciales de las funciones F_1, F_2, ..., F_m, pueden ser transmitidos a un sitio común donde la expresión E puede ser evaluada, aprovechando de esta manera el paralelismo.

Ejemplo 10: Obtener el promedio despachado de la parte 'p14'

$\mathbf{GB}_{AVG(cant)}\ \mathbf{SL}_{PNUM='p14'}$part_desp →
$\mathbf{GB}_{SUM(cant),COUNT}\ \mathbf{SL}_{PNUM='p14'}$PD1
$\mathbf{GB}_{SUM(cant),COUNT}\ \mathbf{SL}_{PNUM='p14'}$PD2

Si obtenemos en S1 y C1 los valores del sitio donde reside PD1, y en S2 y C2 los valores del sitio donde reside PD2, en el sitio origen de la consulta global se calcula E: AVG(cant)=(S1+S2)/(C1+C2)

Las **consultas parametrizadas** son consultas con fórmulas que incluyen parámetros cuyos valores no son conocidos en el momento de la compilación. Por ejemplo, para obtener la información de las dependencias cuyos valores son parámetros sería:

$\mathbf{SL}_{DNUM=\$P1\ OR\ DNUM=\$P2}$DEPEND

En principio, no hay necesidad de entenderse con simplificaciones en ejecución en forma diferente a las realizadas en tiempo de compilación.

5.4 Optimización de Estrategias de Acceso

El objetivo de una estrategia de acceso para una consulta es minimizar el costo. No es un proceso exacto con unas técnicas de optimización perfectamente definidas.

Las técnicas de optimización no obtienen optimalidad. En general, podría decirse que son *buenas estrategias de acceso*.

Las transformaciones vistas en las secciones anteriores constituyen una premisa fundamental para la optimización de queries. Hay dos situaciones a considerar:

1. Las que efectivamente dan beneficio:
    a. Factorización de subexpresiones y eliminación de subexpresiones comunes.
    b. Empujar las operaciones unarias hacia las hojas del árbol de operadores.

2. Las transformaciones que generan alternativas que deben ser comparadas sobre una base del costo:
    a. La conmutatividad de joins y uniones
    b. La transformación de joins en algoritmos semijoins.

Aspectos involucrados por una estrategia de optimización

1. Determinar las copias físicas de los fragmentos sobre los cuales se ejecuta la consulta.
2. El orden de ejecución de las operaciones (sobre todo join, semijoin y unión). Ascender de las hojas a la raíz no siempre da la mejor solución.
3. El método para cada operación (el más difícil es el mejor método para evaluar las operaciones de joins).

El costo de transmisión de un dato de tamaño x equivale a sumar el costo fijo de iniciar una transmisión entre dos sitios más el costo de transmisión del dato dado su tamaño:

$TC(x)=C_0+xC_1$, donde C_0 es el costo fijo de iniciarla y C_1 es el costo de transmisión unitaria.

El retardo de transmisión se define de manera similar:

$TD(x)=D_0+xD_1$, donde D_0 es el tiempo fijo de establecer la conexión y D_1 es la tasa de transferencia unitaria.

Objetivos en optimización de consultas

En el proceso de optimización se busca determinar las siguientes medidas:

- ✓ Operaciones de entrada-salida
- ✓ Operaciones de CPU
- ✓ Cantidad de transmisión de datos entre sitios

Estas permiten obtener las sumatorias de los costos de transmisión más las sumatorias de los costos de procesamiento local.

En redes WAN con bajo ancho de banda, las tasas de transmisión son muy bajas comparadas con, por ejemplo, las tasas de transferencia disco-memoria. Por consiguiente se descartan los costos de procesamiento local.

El uso de semijoins es conveniente. En tal caso, se debe encontrar la mejor secuencia de semijoin para reducir relaciones antes de transmitirlas.

En redes LAN con alto ancho de banda, las tasas de transmisión son comparables con las tasas de transferencia disco-memoria. Por tal motivo, se tienen en cuenta los costos de procesamiento local.

Se puede establecer el uso de joins como táctica y seleccionar la mejor técnica para ejecutar los joins.

EJERCICIOS PROPUESTOS

Ejercicio 1

Traduzca el query global en un query fragmentado y luego transfórmelo en un query más eficiente usando los criterios de simplificación (justificando en cada paso) y aplicando el álgebra de relaciones cualificadas.

PACIENTE(PNUM,NOMBRE,DEPT,TRATAMIENTO)
CUIDADO(PNUM,DROGA,CANT)

PACIENTE1=$\mathbf{SL}_{DEPT='CIRUGIA'}$(PACIENTE)
PACIENTE2=$\mathbf{SL}_{DEPT='PEDIATRIA'}$(PACIENTE)
CUIDADO1=CUIDADO$\mathbf{SJ}_{PNUM=PNUM}$PACIENTE1
CUIDADO2= CUIDADO$\mathbf{SJ}_{PNUM=PNUM}$PACIENTE2

Listar los pacientes que usan aspirina en su cuidado:

$\mathbf{PJ}_{NOMBRE}$(PACIENTE **NJN** $\mathbf{SL}_{DROGA='ASPIRINA'}$CUIDADO

Ejercicio 2

A partir de las siguientes esquemas y fragmentos:

VENDEDOR(CODV,NOMBRE,ZONA)
PRODUCTO(CODPROD,CODV,CANT,VALOR)
MEDICO(CODM,NOMBRE,ESPEC,TRATAMIENTO,ZONA)
CONSULTORIO(CNUM,CODM,NUMCITAS,HORAINI)

VENDEDOR1=$\mathbf{SL}_{ZONA='NORTE'}$(VENDEDOR)
VENDEDOR2=$\mathbf{SL}_{ZONA='SUR'}$(VENDEDOR)
PROD1=PRODUCTO$\mathbf{SJ}_{CODV=CODV}$VENDEDOR1
PROD2=PRODUCTO$\mathbf{SJ}_{CODV=CODV}$VENDEDOR2
MEDICO1=$\mathbf{SL}_{ESPEC='CARDIOLOGIA'}$(MEDICO)
MEDICO2=$\mathbf{SL}_{ESPEC='NEUROLOGIA'}$(MEDICO)
CONSULTORIO1=CONSULTORIO$\mathbf{SJ}_{CODM=CODM}$MEDICO1
CONSULTORIO2= CONSULTORIO$\mathbf{SJ}_{CODM=CODM}$MEDICO2

1. Describa en palabras precisas

a) PROD1:
_____

b) MEDICO2:
_____

c) CONSULTORIO1:
_____

d) VENDEDOR2:
_____

2. Traduzca el query global en el query fragmentado más eficiente que resulte usando transformaciones y criterios de simplificación.

Listar los nombres de los vendedores que han entregado el producto 3150:

$\mathbf{PJ}_{NOMBRE}$*(VENDEDORES* **NJN** $\mathbf{SL}_{CODPROD=3150}$*PRODUCTO)*

a) QUERY TRANSFORMADO:
_____

Listar los nombres de los médicos que atienden en el consultorio 924:

$\mathbf{PJ}_{NOMBRE}(MEDICO\ \mathbf{NJN}\ \mathbf{SL}_{CNUM=924'}CONSULTORIO)$

b) QUERY TRANSFORMADO:

_____

Listar los nombres de los vendedores que trabajan en la zona NORTE pero no venden el producto 1538:

$\mathbf{PJ}_{NOMBRE}(\mathbf{SL}_{ZONA='NORTE'}(VENDEDOR)\ \mathbf{DF}$
$(VENDEDOR\ \mathbf{NSJ}(\mathbf{SL}_{CODPROD=1538}(PRODUCTO))))$

c) QUERY TRANSFORMADO:

_____

Listar los vendedores del norte que atienden sólo a los cardiólogos:

$\mathbf{SL}_{ZONA='NORTE'}(VENDEDOR)\ \mathbf{NJN}\ (\mathbf{SL}_{ESPEC='CARDIOLOGIA'}(MEDICO))$

d) QUERY TRANSFORMADO:

_____

6

Manejo de Transacciones Distribuidas

Hasta este punto, se ha considerado lo que es una consulta, como procesarla, como fragmentarla y algunos aspectos de optimización. Sin embargo, nunca hemos considerado lo que pasa cuando dos consultas intentan actualizar los mismos datos, o que pasa cuando ocurre una falla durante la ejecución de una consulta. Para las consultas, ninguna de las dos situaciones es problemática pero para las actualizaciones sí.

Los mismos problemas de los ambientes centralizados no son ajenos en un ambiente distribuido, sólo que aquí las situaciones son de mayor complejidad.

Se busca entender la relación entre control de concurrencia, mecanismos de recuperación y la estructura de todo el sistema distribuido. Igualmente mostrar la técnica 2-phase commit para recuperación y la técnicas 2-phase lock y 3-phase lock para control de concurrencia.

6.1 Modelo de Transacciones Distribuidas

El modelamiento de las transacciones distribuidas cubre aspectos como las características, los objetivos y la definición. Las características así como son propias de las bases de datos centralizadas, también lo son para las bases de datos distribuidas.

Atomicidad → Las transacciones son atómicas, esto es, todas sus operaciones se hacen o ninguna se ejecuta. Esto implica que las operaciones parciales de una transacción se deben deshacer. A este proceso se le denomina **rollback** y sucede en los siguientes casos:

- ✓ La transacción por si misma o el usuario lo ejecuta
- ✓ Puede ser forzado por una sobrecarga del sistema
- ✓ En caso de un abrazo mortal

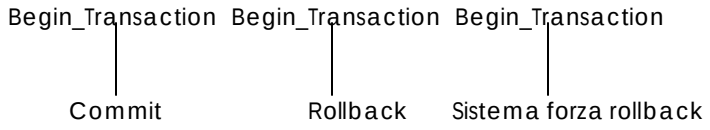
En estos casos, se hace una recuperación (hacia atrás) de la transacción.

Cuando hay una caída del sistema, hay recuperación a una caída o *crash recovery*.

Toda transacción comienza con una primitiva **begin_transacrion** (que puede ser implícita). La terminación de una transacción se

hace a través de una operación llamada **commit** o compromiso (terminación exitosa) o a través de un **rollback** (aborto de la transacción).

Los escenarios posibles para la terminación de una transacción son los siguientes:



Durabilidad → Después del commit, el DBMS debe garantizar la durabilidad de sus resultados, es decir, estos no se perderán. En caso de una falla posterior al commit, existe la garantía de recuperación de la base de datos.

Seriabilidad → En acceso concurrente, el resultado de las transacciones es igual a si se ejecutaran serialmente en algún orden. La característica de garantizar la seriabilidad es llamada control de concurrencia.

Aislamiento → Una transacción no revela resultados intermedios a otra. Esto es necesario para evitar abortos en cascada o efecto dominó.

OBJETIVOS EN EL MANEJO DE TRANSACCIONES DISTRIBUIDAS

Fundamentalmente las transacciones persiguen:

- ✓ Eficiencia
- ✓ Confiabilidad
- ✓ Concurrencia

Hay otros objetivos, de tipo estratégico, que bien valen la pena considerar:

- ✓ Bajar la utilización de CPU y memoria RAM: La ejecución concurrente de muchas operaciones de entrada-salida puede generar un cuello de botella en memoria RAM o en consumo de CPU. Muchos procesos para muchas transacciones puede generar mucho swapping.

- ✓ Bajar la cantidad de mensajes de control entre sitios: No deben ser usados para transferir datos sino para controlar la ejecución de una aplicación. Hay que tener en cuenta que en el costo

del mensaje hay que considerar el costo de transmisión más el costo de CPU (cantidad de instrucciones para el envío del mensaje).

- ✓ Bajar el tiempo de respuesta: Es importante debido al tiempo adicional requerido por la comunicación entre sitios diferentes.

- ✓ Aumentar la disponibilidad: Aún en la presencia de fallas e los sitios.

DEFINICIÓN DE UNA TRANSACCIÓN DISTRIBUIDA

Hay varios elementos que delimitan una transacción distribuida y permiten definirla. Ellos son:

Begin_Transaction: Primitiva que indica el comienzo de la transacción y a partir todas las acciones deben ser ejecutadas hasta que haya un commit o un rollback.

Commit: Indica la terminación exitosa de la transacción.

Rollback: Acción que permite anular los cambios hechos hasta el momento por la transacción.

Agentes: Procesos de una aplicación que han de ejecutarse en otros sitios. Para cooperar en la ejecución de una operación global requerida por la aplicación, los agentes tienen que comunicarse (a través de mensajes).

Agente Raíz: Es el encargado de iniciar la transacción, normalmente en el sitio origen de la transacción.

El agente raíz, en la mayoría de los sistemas, tiene la responsabilidad de iniciar Begin_Transaction, Commit y Rollback.

En algunos sistemas, sólo el agente raíz puede solicitar la creación de un nuevo agente.

Ejemplo: Sea la relación cuenta definida por el siguiente esquema: cuenta(cta_num, saldo). Sobre ella vamos a realizar una transferencia de fondos. Las rutinas aparecen a continuación.

NIVEL GLOBAL

Input($valor, $cta_orig, $cta_dest);
begin_transaction;

```
SELECT saldo INTO $saldo_orig
    FROM cuenta
    WHERE cta_num=$cta_orig;
if $saldo_orig < $valor
    rollback;
else begin
    UPDATE cuenta
    SET saldo=saldo - $valor
    WHERE cta_num=$cta_orig;
    UPDATE cuenta
    SET saldo=saldo + $valor
    WHERE cta_num=$cta_dest;
    commit;
end;
```

CON AGENTES

Agente_raíz:

```
Input($valor, $cta_orig, $cta_dest);
begin_transaction;
SELECT saldo INTO $saldo_orig
    FROM cuenta
    WHERE cta_num=$cta_orig;
if $saldo_orig < $valor
    rollback;
else begin
    UPDATE cuenta
        SET saldo=saldo - $valor
        WHERE cta_num=$cta_orig;
    create Agente1;
    send to Agente1($valor, $cta_dest);
    commit;
end;
```

Agente1:

```
receive from Agente_raiz($valor,$cta_dest);
UPDATE cuenta
    SET saldo=saldo + $valor
    WHERE cta_num=$cta_dest;
```

Soporte de un DDMBS

Un sistema manejador de bases de datos distribuidas debe implementar primitivas globales begin_transaction, commit,

Rollback y asumir en cada sitio un Manejador Local de Transacciones (LTM) que aproveche los sistemas existentes.

6.2 Recuperación de Transacciones Distribuidas

RECUPERACIÓN EN SISTEMAS CENTRALIZADOS

1. Tipos de fallas
   - **i)** Fallas sin pérdida de información: Toda la información en memoria está disponible para recuperación. Ejemplo: Un rollback de una transacción por algún error.
   - **ii)** Fallas con pérdida de información volátil: La información del disco no es afectada. Estas fallas ocurren con las caídas del sistema.
   - **iii)** Fallas con pérdida de información no volátil: La información del disco se pierde debido a fallas en los medios de almacenamiento.
   - **iv)** Fallas con pérdida de almacenamiento estable: Son fallas simultáneas de dispositivos de almacenamiento no volátil que soportan almacenamiento estable. Estas tienen una probabilidad muy baja de que ocurran.

   Almacenamiento Estable: Aquel que bajan la probabilidad de falla replicando en varios discos con MODOS DE FALLA INDEPENDIENTES, incluyendo una estrategia de reemplazo cuidadosa.

2. Archivos de Log. Técnica básica para implementar transacciones en presencia de fallas. Almacenan la información suficiente para deshacer resultados parciales y para rehacer resultados definitivos .

   Con las fallas tipo ii) se aprovecha la información de los archivos Log la cual consta de:
   - **i)** Transacciones comprometidas que no quedaron escritas.
   - **ii)** Transacciones no comprometidas que quedaron y no quedaron escritas.

   Para la recuperación de las transacciones se utilizan las propiedades:
   - **i)** undo(undo(... (acción)...)) = undo(acción)
   - **ii)** redo(redo(... (acción)...)) = redo(acción)

   Contenido de los archivos de Log:

- Identificador de transacción
- Identificador de fila
- Tipo de acción (insert, update, delete)
- Valor anterior
- Valor nuevo
- Información auxiliar (dirección del registro anterior y otra)
- Primitivas begin_transaction, commit, Rollback.

Antes de un checkpoint, al menos la porción *undo* del Log en memoria debe escribirse en almacenamiento estable.

Antes de escribir un *commit*, todos los registros Log de la transacción deben escribirse en almacenamiento estable, en los archivos de Log. Esto garantiza la recuperación hacia delante.

3. Checkpoint. Consiste en escribir en almacenamiento estable, los registros de log en memoria y los cambios en memoria (en los buffers de la base de datos), en el lugar donde se encuentra físicamente la base de datos.

   Escribir el registro checkpoint (en Log) en almacenamiento estable consiste en registrar las transacciones activas en el momento del checkpoint).

4. Procedimientos de recuperación.
   - **i)** Buscar y leer, en el archivo de Log, el último registro checkpoint.
   - **ii)** Incluir en una lista *Undo* todas las transacciones en el momento del checkpoint. Iniciar una lista *Redo* vacía.
   - **iii)** Recorrer el archivo de Log hasta el final haciendo lo siguiente: Si hay *Begin_transaction* registrar en la lista *Undo*. Si hay *Commit* registrar en la lista *Redo*.
   - **iv)** Recorrer las listas haciendo undo y redo.

5. Archivos de Log fuera de linea. Son Logs de transacciones sobre los que no se puede hacer *undo* ni *redo*. Son utilizados en caso de fallas tipo iii), las cuales pueden ser:
   - **i)** Fallas en las cuales la información de la base de datos se pierde pero los Logs fuera de línea están seguros.
   - **ii)** Fallas con información de Logs fuera de línea perdida.

Estos mecanismos son aplicables en los sitios donde operan las bases de datos locales que se integran en una base de datos distribuida.

FALLAS DE COMUNICACIÓN EN BASES DE DATOS DISTRIBUIDAS

Para mencionar las fallas, conviene mencionar los requerimientos de una red para que dos sitios A y B puedan intercambiar mensajes:

- El sitio A recibe un reconocimiento positivo después de un tiempo menor que un parámetro DMAX.
- El mensaje es entregado en B conservando la secuencia.
- El mensaje es correcto.

Los tipos de errores pueden ser **mensajes perdidos** (ya sea de datos o de confirmación) o **particiones** (A no se comunica con B pero se comunica con otros sitios, los cuales tampoco se comunican con B).

Si B no recibe reconfirmación (ACK) después de un tiempo (**mensaje perdido**), una de las siguientes situaciones pudo haber ocurrido:

- El sistema está lento.
- Una falla de comunicación ocurrió antes de la entrega del mensaje en B.
- Una falla de comunicación después de la entrega en B.
- Una caída del sitio B antes de la entrega en B.
- Una caída del sitio B después de la entrega en B.

Niveles de algoritmos de confiabilidad

CLASE 1: Sólo atienden fallas de sitios. Son ideales para redes completamente confiables.

CLASE 2: Atienden fallas de sitios y mensajes perdidos.

CLASE 3: Atienden fallas de sitios, mensajes perdidos y particiones.

Sistemas k-elásticos o k-resilientes

Son sistemas que pueden tolerar simultáneamente k fallas, como k sitios caídos o k particiones generadas.

MODELO DE REFERENCIA PARA EL MANEJO DE TRANSACCIONES DISTRIBUIDAS

Los procesos que se involucran en el manejo de las transacciones distribuidas son:

- LTM (Manejador Local de Transacciones): implementa las primitivas correspondientes a las subtransacciones en cada uno de los sitios. Los LTMs no se comunican entre sí. Ellos implementan la Interface 1 (**local_begin**, **local_commit**, **local_Rollback** y **local_create**).

- DTM (Manejador de la Transacción Distribuida): el cual tiene como requisitos:
    - Asegurar la atomicidad de las subtransacciones por parte del LTM
    - Escribir algunos registros (nuevos tipos de registros) en almacenamiento estable, por parte del LTM, a nombre del DTM

  El DTM es implementado por un conjunto de agentes_DTM locales que se comunican entre sí. Implementa la interface 2 (**begin_transaction**, **commit**, **rollback**, **create** –remoto-). El DTM debe saber cuáles agentes forman una transacción distribuida.

- Agentes de la Transacción Distribuida: Constituido por el agente raíz y los otros agentes. El agente raíz es quien utiliza la interface 2 o DTM.

Para asegurar la atomicidad de una transacción distribuida:

1. En cada sitio, o todas las acciones son ejecutadas o ninguna se ejecuta (A cargo de cada LTM).
2. Todos los sitios deben tomar la misma decisión (todos hacen commit o todos hacen rollback)

¿Cómo funciona la Transacción Distribuida con este modelo?

- El agente Raíz inicia el begin_transaction.
- El agente DTM solicita un local_begin al LTM local (en el sitio origen). En este sitio el LTM registra un begin_transaction en el Log local.
- Opcionalmente, se pueden registrar operaciones en el sitio origen.
- El agente Raíz encuentra la instrucción *Create Agente2*, para activar un agente en el sitio 2. Lo hace a través del DTM.
- El agente DTM envía la solicitud de *Create* al agente DTM del sitio 2.
- El agente DTM del sitio 2 solicita un *local_create* al LTM del sitio 2.

- El LTM del sitio 2 crea el proceso Agente2. A partir de este momento, el agente Raíz y el agente2 ya se pueden comunicar.
- El agente DTM solicita un local_begin al LTM del sitio 2, donde el LTM registra un begin_transaction en el su Log local.
- El agente2 (en el sitio 2) entra en estado de espera mediante su primera instrucción *Receive from agente_raiz(parámetros)*.
- El agente Raíz transmite al agente2 los parámetros.
- El agente Raíz inicia el commit mediante el protocolo 2PC, a través del DTM, quien solicita *local_commit* a los LTMs donde hay subtransacciones activas.

Tanto el commit como el rollback son iniciados por el agente Raíz. La principal dificultad del protocolo commit global es que todas las subtransacciones hagan commit aún en caso de fallas. Es el proceso más difícil y costoso.

PROTOCOLO 2-PHASE-COMMIT

Se apoya en dos fases: La primera es obtener una decisión común y la segunda implementar la decisión. Se apoya en un proceso Coordinador y varios procesos Participantes.

Fase 1:	**Coordinador**	Registra *prepare* en el LOG Transmite *prepare* a los participantes Activa TimeOut
	Participante	Espera mensaje *prepare* Si decide hacer commit Entonces Registra operaciones en LOG Registra *ready* en LOG Transmite *ready* al coordinador Sino Registra *rollback* en LOG Transmite *rollback* al coordinador
	Coordinador	Espera mensaje *ready* ó *rollback* de todos los Participantes o TimeOut
Fase 2:	**Coordinador**	Si TimeOut o por lo menos un rollback Registra *global_rollback* en LOG Transmite *rollback* a los participantes Sino Registra *global_commit* en LOG Transmite *commit* a los participantes
	Participante	Espera mensaje *rollback* o *commit* Registra *rollback* ó *commit* en LOG Transmite *Ack* al coordinador

	Ejecuta *rollback* ó *commit*
Coordinador	Espera mensaje *Ack* de los participantes
	Registra *complete* en LOG

Comportamiento del protocolo 2PC en presencia de fallas de sitios

- Un participante falla antes de escribir el registro *ready* en el LOG o antes de transmitirlo al coordinador.
  - Se activa el TimeOut del coordinador. El coordinador transmite *rollback* a los participantes.
  - Cuando el participante se recupera, simplemente se aborta la transacción.

- Un participante falla después de escribir *ready* en el LOG y transmitirlo.
  - El 2PC sigue su curso normal, los otros sitios terminan la transacción (commit o rollback).
  - El coordinador no recibe el Ack del participante caído. Cuando este se recupera, consulta al coordinador y otro participante el resultado de la transacción y lo ejecuta (commit o rollback).

- El coordinador falla después de escribir el *prepare* en el LOG y transmitirlo pero antes de registrar un *global_commit* ó *global_rollback* en el LOG.
  - Los participantes que hayan hecho *ready* esperan la recuperación del coordinador.
  - Cuando el coordinador se recupera, lee la identidad de los participantes del *prepare*, transmite el *prepare* nuevamente a los participantes y continúa.

- El coordinador falla después de *global_commit* ó *global_rollback* en el LOG y transmitirlo, pero antes de escribir el *complete* en el LOG.
  - Cuando el coordinador se recupera, envía el *global_commit* ó *global_rollback* nuevamente.

- El coordinador falla después de escribir el registro *complete* en el LOG.
  - La transacción está concluida.

Comportamiento del protocolo en presencia de mensajes perdidos

- Un mensaje *prepare* se pierde.
  - Se activa el TimeOut del coordinador. El coordinador registra *rollback* y lo transmite a los participantes.

- Un mensaje *ready* o *rollback* de un participante.
  - Se activa el TimeOut del coordinador. El coordinador registra *rollback* y lo transmite a los participantes.

- Un mensaje *commit* o *rollback* del coordinador.
  - Introducir el mecanismo de TimeOut en el participante, el cual debería solicitar repetición de la decisión.

- Un mensaje *Ack* de un participante.
  - Introducir un nuevo TimeOut en el coordinador, el permitiría enviar el comando de nuevo (*commit* o *rollback*).

Comportamiento del protocolo en presencia de particiones de red

Si la falla se presenta en la partición donde se encuentra el coordinador, para la otra partición el comportamiento es como si fuera una falla del coordinador.

Si la falla se presenta en la partición donde no se encuentra el coordinador, para el coordinador el comportamiento es como si fueran fallas de los participantes.

Características del 2PC

1. Capacidad de abortar unilateralmente hasta justo antes de la respuesta *ready*.
2. Bloqueo por falla del coordinador o de la red.
3. Posibilidad de eliminar el mensaje *prepare* pero mayor probabilidad de bloqueo y la autonomía de cada sitio se ve reducida.
4. Incremento de eficiencia usando *commit presumido* o *rollback presumido*, propio del Sistema R*.
5. Tratamiento al problema de la recuperación remota solicitando información a
   a. El coordinador
   b. Otro participante
   c. Mediante un spool de mensajes en cada sitio

6.3 Control de concurrencia basado en Transacciones Distribuidas

CONTROL DE CONCURRENCIA EN SISTEMAS CENTRALIZADOS

En los sistemas centralizados, el control de concurrencia está basado en bloqueos y en la correcta definición de transacciones.

1. Modos de bloqueo:
    a. Bloqueo Compartido (**S** o Shared) donde un dato puede ser accesado por varias transacciones.
    b. Bloqueo Exclusivo (**X** o eXclusive) donde un dato puede ser accesado por sólo una transacción.

2. Transacciones bien formadas, con bloqueo **S** para lectura y bloqueo **X** para escritura.

3. Conflicto entre dos transacciones, el cual se da cuando hay dos modos incompatibles de bloqueo: **S** con **X** y **X** con **X**.

Para que haya ejecución concurrente de transacciones se requiere de:

1. Transacciones bien formadas
2. Reglas de compatibilidad de bloqueo
3. Las transacciones no deben requerir bloqueos después de liberados. Son 2PL (2-Phase Lock):
    a. Fase creciente: de bloqueos
    b. Fase decreciente: de liberaciones de bloqueos

Ejecución de transacciones garantizando aislamiento

El 2PL no garantiza el aislamiento. Para garantizarlo se debe seguir la siguiente secuencia:

1. begin_transaction
2. bloqueos
3. operaciones
4. commit
5. liberar bloqueos

CONTROL DE CONCURRENCIA EN SISTEMAS DISTRIBUIDOS

Consiste en aprovechar las facilidades de los DBMSs locales (LTMs) para construir los mecanismos distribuidos. Estos permiten bloquear y liberar datos locales.

En el nivel 1, los LTMs implementan las primitivas **local_lock_shared**, **local_lock_exclusive** y **local_unlock**.

En el nivel 2, los DTMs implementan las primitivas **lock_shared**, **lock_exclusive** y **unlock**.

Problemas con datos replicados

Aparecen cuando dos transacciones tienen conflicto de bloqueo en dos copias del mismo dato, almacenado en sitios diferentes. Y lo peor es que no se dan cuenta del conflicto. Esto se convierte en un Bloqueo Inútil.

La solución consiste en hacer bloqueos locales por parte de los LTMs locales en todos los sitios del dato replicado. Esto implica que una primitiva de bloqueo es traducida en muchas primitivas de bloqueo.

Alternativas de solución:

1. Bloqueo de escritura en todos, bloqueo de lectura en uno: Un conflicto es detectado siempre. Si es compartido-exclusivo (**S** con **X**), el conflicto es detectado en el sitio del bloqueo **S**. Si es exclusivo-exclusivo (**X** con **X**), el conflicto es detectado en todos los sitios.

   T_1 → S
   T_2 → X X X X X
   A A A A A

2. Bloqueo por mayoría: El conflicto es detectado en al menos un sitio. Supongamos que el dato A está replicado en cinco sitios y las transacciones T_1 y T_2 trata de bloquearlo.

   T_1 T_1 T_1
   A A A A A

   T_2 no puede bloquear el dato A.

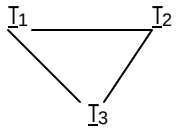
3. Bloqueo de copia primaria: Los conflictos son descubiertos en el sitio donde reside la copia primaria.

Problema del Abrazo Mortal

Puede involucrar dos o más transacciones. Un sistema lo puede descubrir mediante un grafo wait-for y analizando si hay ciclos en él.

En transacciones distribuidas, un grafo de espera global debe ser construido por el DTM. Dicha construcción corresponde a algoritmos complejos.

T_1	T_2	T_3
A		
	B	
B		C
	C	
		A



En vista de la complejidad, la mayoría de los sistemas usan TimeOuts para detección del abrazo mortal.

Si el TimeOut es corto, muchas transacciones que no entran en abrazo mortal pueden ser abortadas. Si es largo, podría haber desperdicio de tiempo por transacciones en abrazo mortal.

El mecanismo de TimeOut no es conveniente para sistemas congestionados por el efecto en cascada que esto puede generar.

El problema de la disponibilidad con 2PL

En caso de bloqueo de todas las copias de un dato, si el sitio con una copia del dato se cae, la transacción se bloquea. En este caso, la redundancia disminuye la disponibilidad del sistema para la transacción.

Una subtransacción bloqueada después de bloquear datos se apropia de recursos que no pueden ser usados por otras transacciones. La solución es abortar las subtransacciones bloqueadas, pero antes de enviar el *ready*.

La recuperación y el 2PL

La fase creciente del 2PL y el abrazo mortal deben ser cancelados antes de que un participante escriba *ready* en el LOG.

Si una falla bloquea la ejecución del 2PC y algún participante ha registrado *ready*, el participante debe conservar los bloqueos de la transacción, hasta que sea informado de la decisión.

MODELOS DE PROCESOS Y DE COMUNICACIÓN EN TRANSACCIONES DISTRIBUIDAS

Son los enfoques para la organización de la computación y de la comunicación de la transacciones.

Modelo de Procesos

Consiste en tener un proceso del sistema operacional por cada transacción.

Usa tablas de bloqueo compartidas por los procesos. Hay una fuerte correspondencia entre transacciones y procesos.

La desventaja aparece cuando hay una entrada-salida infructuosa de una transacción, lo cual implica cambio de proceso dando lugar a mucho trabajo de CPU. Muchos procesos (transacciones) concurrentes demandan mucha memoria.

Muchos DBMSs tienen su propio planificador de procesos e implementan más convenientemente sus estrategias de planificación.

Las Transacciones Distribuidas en el Modelo de Procesos

Una transacción distribuida implica la ejecución de varios procesos en diferentes sitios.

Los agentes de una transacción distribuida son procesos locales separados, aunque hay variaciones:

- Crear un proceso para servir cada solicitud, generándose varios procesos.
- Crear sólo un proceso para la primera solicitud y retenerlo durante toda la transacción.

Modelo de Servidores

Los procesos servidores existen independientes de las transacciones.

Las transacciones usan los servicios del proceso servidor por medio de mensajes.

Un proceso servidor atiende en forma dinámica las transacciones.

Un registro LOG es etiquetado con el identificador de transacción y no con el identificador de servidor.

Hay baja sobrecarga en creación y en conmutación de procesos.

Una transacción distribuida solicita la ejecución de una función remota enviando un mensaje de solicitud a un servidor en el sitio remoto. Este mensaje es etiquetado con el identificador de transacción.

El proceso servidor es el agente de la transacción solo por el tiempo que se ejecuta a nombre de esa transacción. Después se convierte en agente de otra transacción.

Sesiones para comunicación de procesos

El enfoque de procesos usando sesiones es más conveniente. Pues se requiere que el proceso por cada sitio sea retenido en lugar de crear uno nuevo cada vez que haya una solicitud. Hay un proceso simple para todas las transacciones en cada sitio, además es una estructura computacional distribuida estable.

Se justifica por los siguientes motivos:

- La autenticación e identificación se hace una sola vez.
- El intercambio de muchos mensajes durante toda la sesión.

Datagramas

Con las siguientes características:

- La comunicación, entre procesos o servidores, tiene un patrón irregular.
- Es no-orientado a conexión.
- Todo el protocolo de autenticación e identificación se ejecuta para cada mensaje.

Topologías en Transacciones Distribuidas

ESTRUCTURA CENTRALIZADA de Transacciones Distribuidas:

- El agente raíz activa y controla los otros agentes (esto simplifica el control de concurrencia y la recuperación).
- Los otros agentes no deberían comunicarse entre sí, pero para evitar sobrecarga transmitida al agente raíz, se permite algún grado de comunicación.

ESTRUCTURA JERARQUICA de Transacciones Distribuidas:

- Cada agente puede activar otros agentes, creando un árbol de agentes.
- Hay comunicación directa entre agentes.

Estructuras de comunicación para protocolos commit

CENTRALIZADA → La comunicación es entre coordinador y participantes.

JERARQUICA → Cada agente DTM, que es interno al árbol, recoge mensajes de sus hijos o los dispersa a ellos.

LINEAL → Hay ordenamiento de sitios. Cada sitio tiene un predecesor y un sucesor, excepto el primero y el último.

DISTRIBUIDO → Cada participante requiere comunicarse con los otros participantes para efectos de commit o rollback.

EJERCICIOS RESUELTOS

Completar:

1. Los procesos que permiten dar integración al DDBMS son _DTMs_

2. En el manejo de Transacciones Distribuidas sobre una WAN lenta, los tipos de costos que no se tienen en cuenta son los de __transmisión_

3. Si en un servidor con RAID 5 fallan 2 discos de los 4 que posee, hay una falla __con pérdida de almacenamiento estable_

4. Si en un servidor con RAID 5 falla 1 disco de los 4 que posee, hay una falla __ninguna_

5. Si en un servidor con RAID 1 falla 1 disco, hay una falla __ninguna_

6. Si el DDBMS aborta una transacción de un usuario por un error de overflow hay falla __sin pérdida de información

7. Si el sistema es desconectado accidentalmente hay una falla __con pérdida de información volátil

8. Si el Participante1 ha transmitido *ready* y previamente había hecho *lock_shared(A)* y otra transacción requiere leer el dato *A* qué tiene que esperar?__Nada_

9. Si el Participante1 ha transmitido *ready* y previamente había hecho *lock_shared(A)* y otra transacción requiere modificar el dato *A* qué tiene que esperar?__Desbloqueo del dato_ Cuándo?__global_commit o global_abort

10. En una liquidación de una nómina con una intenso intercambio de datos entre dos ciudades durante 30 minutos quincenalmente, el enfoque de comunicación de procesos adecuado es __Sesión_

11. Qué secuencia de ejecución se da en las siguientes operaciones (hay bloqueos explícitos previos) sabiendo que T1 inicia primero pero T2 inicia antes que T1 actualice Y y T3 inicia de último pero antes que T2 actualice Z: (Ejemplo: T1-X T1-Y T3-Z ...)

    T1(update(X),update(Y))
    T2(update(Y),update(Z))
    T3(update(Z),update(W))

    T1-X T2-Y T3-Z T3-W T2-Z T1-Y

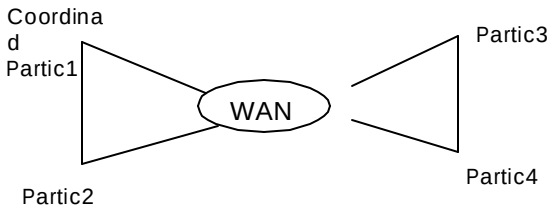
12. Qué secuencia de ejecución se da en las siguientes operaciones (hay bloqueos explícitos previos) sabiendo que T1 inicia primero pero T2 inicia antes que T1 actualice Y y T3 inicia de último pero antes que T2 actualice Z:

    T1(update(X),update(Y))
    T2(update(Y),update(Z))
    T3(update(Z),update(X))

    Abrazo Mortal_

EJERCICIOS PROPUESTOS

Suponga la transacción T de acuerdo con los procesos mostrados del 2PC:



2. Cuál es el resultado final de la transacción si se daña la WAN después de que Partic3 y Partic4 registran el *ready* en el LOG y lo transmiten por la WAN.

3. Cuál es el resultado final de la transacción si el Coordinador recibe el mensaje *ready* del Partic3 pero se cae antes de recibir el mensaje de Partic4.

Supóngase que las transacciones T1 y T2 están ejecutándose en el tiempo y sean A1, A2, A3 y A4 replicas de la tabla A en los sitios 1, 2, 3 y 4:

Tiempo	T1	T2
a)	lock_X(A1)	lock_X(A2)
b)	lock_X(A3)	lock_X(A4)
c)	lock_X(A2)	lock_X(A1)
d)	lock_X(A4)	lock_X(A3)

4. Explique si funciona o no el bloqueo por mayoría.

5. Explique si funciona o no el bloqueo de escritura en todas.

6. Explique si funciona o no el bloqueo de copia primaria, suponiendo que el sitio de la copia primaria es el 1.

7. Una transacción bloquea el dato X replicado en n sitios. Uno de los participantes se cae después de que todos envían el *ready*. ¿Qué ocurre con la transacción y por qué?

8. Durante el día, en una corporación, hay en promedio 20 transferencias de fondos entre dos cuentas de un cliente especial de la corporación. Las cuentas están en Armenia y Santafé de Bogotá respectivamente. ¿Cuál enfoque de comunicación de procesos es el adecuado y por qué?

9. Durante el día, en una empresa nacional, hay una transacción con un procesamiento en batch que dura aproximadamente 1 hora. ¿Cuál enfoque de comunicación de procesos es el adecuado y por qué?

7

Control de Concurrencia Distribuida

El control de concurrencia se entiende con las propiedades de aislamiento y consistencia de las transacciones. El mecanismo de control de concurrencia distribuida asegura que la consistencia de la base de datos sea mantenida en un ambiente distribuido multiusuario. Si las transacciones son internamente consistentes, es decir, no violan restricciones de integridad y consistencia, la manera más simple de lograr su objetivo es ejecutar cada transacción completamente sola, una después de otra. Sin embargo, esta alternativa no es práctica ya que minimizaría el rendimiento del sistema. El nivel de concurrencia es entonces el parámetro más importante en sistemas distribuidos. Por consiguiente, su mecanismo intenta encontrar un punto adecuado entre mantener la consistencia de la base de datos y un alto grado de concurrencia.

SERIABILIDAD EN UNA BASE DE DATOS CENTRALIZADA

Dos transacciones T_i y T_j se ejecutan serialmente en un itinerario S (secuencia de operaciones), si la última operación de T_i precede la primera operación de T_j, o viceversa. De lo contrario, se ejecutan concurrentemente.

Definamos Serial(S) como una secuencia de transacciones de un itinerario serial S. Si un itinerario S es no serial, Serial(S) no está definido. Ejemplo:

$S = R_i(x)\ R_i(y)\ W_i(z)\ R_j(z)\ R_k(x)\ R_k(y)\ R_k(z)$, donde los subíndices identifican la transacción, R una operación de lectura y W una de escritura.

$Serial(S) = T_i\ T_j\ T_k \Rightarrow T_i < T_j < T_k$ (T_i precede a T_j y T_j precede a T_k)

La ejecución serial de transacciones descrita por un itinerario serial es por definición correcta.

Ejecución Correcta de Transacciones en presencia de Concurrencia

Un itinerario es correcto si es seriable, es decir, computacionalmente equivalente a un itinerario serial.

Un <u>Mecanismo de Control de Concurrencia es correcto</u> si permite que las transacciones ejecuten operaciones de manera tal que se produzcan itinerarios seriables.

Equivalencia de Itinerarios

Sean dos itinerarios S_1 y S_2. Se dice que S_1 es equivalente a S_2 si para cada par de operaciones en conflicto O_i y O_j tales que $O_i < O_j$ en S_1 entonces $O_i < O_j$ en S_2. Ejemplo:

S_1: $R_i(x)$ $R_j(x)$ $W_j(y)$ $W_i(x)$
S_2: $R_j(x)$ $R_i(x)$ $W_i(x)$ $W_j(y)$
S_3: $R_j(x)$ $W_j(y)$ $R_i(x)$ $W_i(x)$

La precedencia de las transacciones depende sólo del orden de las operaciones en conflicto. En los tres itinerarios $R_j(x) < W_i(x)$, es decir, la transacción j lee el dato x antes de que la transacción i lo modifique, y eso precisamente debe conservarse en todos los itinerarios.

SERIABILIDAD EN UNA BASE DE DATOS DISTRIBUIDA

Itinerario Local

Es una secuencia de operaciones ejecutadas por las transacciones en un sitio.

Ejecución Correcta de una Transacción Distribuida

La ejecución correcta de las transacciones T_1, T_2, ..., T_N correspondientes a una transacción distribuida se define de la siguiente manera:

1. Cada itinerario local S_k es seriable.
2. Existe un ordenamiento total de las transacciones tales que: Si $T_i<T_j$ en el ordenamiento total, entonces hay un itinerario serial $S_{k'}$ tal que S_k es equivalente a $S_{k'}$ y $T_i < T_j$ en Serial($S_{k'}$) para todo k, donde ambas transacciones han ejecutado alguna acción.

Otra forma de definirla:

Sean T_1, T_2, ..., T_N un conjunto de transacciones. Sea E una ejecución de estas transacciones modeladas por los itinerarios S_1, S_2, ..., S_M (Matriz de MxN). E es correcto (seriable) si existe un ordenamiento total de las transacciones tales que para cada par de operaciones en conflicto O_i y O_j de T_i y T_j, $O_i < O_j$ en cualquier itinerario S_1, S_2, ..., S_M si y solo si $T_i < T_j$ en el ordenamiento total.

Un Mecanismo de Control de Concurrencia es correcto si permite solamente ejecuciones correctas de transacciones distribuidas.

2PL es un Método de Control de Concurrencia Distribuido

Si todas las transacciones distribuidas son 2PL entonces los itinerarios locales son seriables.

2PL es un método correcto de control de concurrencia distribuido.

2PL garantiza que todas las ejecuciones de transacciones sean seriables pero no garantiza que todas las ejecuciones seriables se produzcan.

EJERCICIO PROPUESTO

Sean T1 y T2 transacciones concurrentes. Produzca un itinerario seriable equivalente a las siguientes operaciones:

$Read_{T1}(X)$, $Read_{T1}(Y)$, $Read_{T2}(X)$, $Write_{T2}(X)$, $Write_{T1}(Y)$

8

Bases de Datos Distribuidas en Oracle

El nivel de sofisticación de la tecnología de bases de datos distribuidas ha aumentado mientras que el entendimiento por parte del usuario promedio no. Es importante que, con las facilidades de distribución de Oracle, los desarrolladores tengan una herramienta y una guía rápida a la mano. Introduciremos varios aspectos, desde la definición del esquema de la base de datos y la distribución de los datos en Oracle mediante Links o enlaces, hasta la generación de los formularios en Oracle, pasando por bloques simples y luego por formas maestro-detalle.

8.1 Oracle y el Modelo OSI

Net8 es el protocolo de red que Oracle provee para soportar las comunicaciones con una base de datos Oracle sobre una red.

Aunque las transacciones son ejecutadas sobre el servidor de la base de datos, ellas usualmente no se inician en él. Una transacción se puede iniciar mediante un click de mouse sobre una página web o una lectura de código de barras en un supermercado.

Hay diferentes componentes Oracle para redes asociados con los niveles 5, 6 y 7. En los niveles inferiores del modelo OSI se manejan las características físicas de la red y el enrutamiento. Analizaremos los niveles superiores.

Nivel de Aplicación

Este nivel está formado por las interfaces de usuario como por ejemplo, los web browsers, las aplicaciones desarrolladas en Forms o, inclusive, lectores de códigos de barras. La aplicación inicia la solicitud en el lado del cliente. Estas pueden ser de conexión, de consulta o de actualización.

Las solicitudes se hacen a través del OCI (Oracle Call Interface), el cual contiene llamadas API para hacer lo siguiente:

- Conectarse a la base de datos y desconectarse
- Análisis sintáctico de las instrucciones SQL

- Abrir y cerrar cursores
- Ejecutar instrucciones SQL
- Manejar excepciones (errores)
- Obtener filas de datos

Nivel de Presentación

En este nivel reside el Net8, el cual es usado por el OCI. Net8 es requerido para establecer comunicación entre servidores y clientes o entre servidores con otros servidores. Net8 es también responsable de traducir diferencias entre conjuntos de caracteres o representaciones de datos que puedan existir a nivel del sistema operacional.

Net8 opera en un ambiente homogéneo aunque se tengan diferentes sistemas operacionales y diferentes arquitecturas de procesador. Tiene tres componentes básicos:

- El Cliente: Software que inicia la conexión, lo cual puede ser desde una página web o desde otro servidor Oracle.
- El Servidor: Software al cual el cliente se conecta. Puede ser un servidor Oracle.
- El Oyente (Listener): Crea puntos de escucha (o direcciones) que alberga el servidor Oracle.

Es claro que estas conexiones se apoyan en el nivel de sesión, encargado real de establecer la comunicación entre los procesos.

Nivel de Sesión

Está constituido por el TNS (Transport Network Substrate), componente Oracle encargado de:

- Establecer y finalizar conexiones entre bases de datos y transportar datos y solicitudes.
- Traducir mensajes OCI en mensajes SEND. Estos mensajes OCI pudieron haber sido traducidos en el nivel de presentación o pudieron haber pasado directamente del nivel de aplicación en el caso de un ambiente completamente homogéneo.

- Pasar mensajes RECEIVE en formato OCI hacia el nivel de aplicación pasando, si es necesario, por el nivel de presentación.

- Intercambiar datos con el adaptador de protocolos de Oracle, el cual formatea los datos para el nivel de transporte, dependiendo del protocolo a utilizar.

- Proveer manejo de errores e interrupciones.

La siguiente es una situación típica Oracle en el modelo OSI:

CLIENTE	NIVEL	SERVIDOR
Aplicación Forms	Aplicación	Servidor Oracle
Net8	Presentación	Net8
TNS	Sesión	TNS
TCP	Transporte	TCP
IP	Red	IP
Ethernet	Enlace	Ethernet
	Físico	

8.2 Definición del esquema y creación de la base de datos distribuida

La base de datos distribuida estará definida de la siguiente manera:

Esquema Global:

CLIENTE (CNUM, NOMBRE, CIUDAD)
PART_DESP (CNUM, PNUM, DNUM, CANT)

Esquema de Fragmentación:

CLIEN1 = $\mathbf{SL}_{\text{CIUDAD='Armenia'}}$CLIENTE
CLIEN2 = $\mathbf{SL}_{\text{CIUDAD='Pereira'}}$CLIENTE
PD1 = PART_DESP $\mathbf{SJ}_{\text{CNUM=CNUM}}$ CLIEN1
PD2 = PART_DESP $\mathbf{SJ}_{\text{CNUM=CNUM}}$ CLIEN2

Esquema de Asignación:

Sitio 1, con los fragmentos CLIEN1 y PD1
Sitio 2, con los fragmentos CLIEN2 y PD2

En Oracle no existe ningún objeto que se denomine *fragmento*. Lo que conceptualmente conocemos como fragmentos será representado por tablas.

En el sitio 1, la máquina se llamará **Maq1** los fragmentos se crearán así desde SQL*Plus:

```
SQL>CREATE TABLE clien1 (
                    cnum      VARCHAR2(2) NOT NULL,
                    nombre    VARCHAR2(30) NOT NULL,
                    ciudad    VARCHAR2(10));

SQL>CREATE TABLE pd1 (
                  cnum      VARCHAR2(2) NOT NULL,
                  pnum      VARCHAR2(2) NOT NULL,
                  dnum      NUMBER(2) NOT NULL,
                  cant NUMBER(3));
```

Similarmente, en el sitio 2, la máquina se llamará **Maq2** y se definirán los fragmentos así:

```
SQL>CREATE TABLE clien2 (
                    cnum      VARCHAR2(2) NOT NULL,
                    nombre    VARCHAR2(30) NOT NULL,
                    ciudad    VARCHAR2(10));

SQL>CREATE TABLE pd2 (
                  cnum      VARCHAR2(2) NOT NULL,
                  pnum      VARCHAR2(2) NOT NULL,
                  dnum      NUMBER(2) NOT NULL,
                  cant NUMBER(3));
```

En estas instrucciones DDL se pueden incluir restricciones de chequeo, de tal manera que CLIEN1 sólo acepte clientes de Armenia y CLIEN2 acepte no más que de Pereira. También se pueden incluir restricciones de integridad referencial de forma que PD1 esté relacionado con CLIEN1 y PD2 con CLIEN2, utilizando CNUM como llave foránea.

El usuario en el sitio 1 se llamará **user1** y el del sitio 2 se llamará **user2**. Podríamos suponer que en el sitio 1 la base de datos local está montada sobre Windows 2000 Server y la del sitio 2 sobre Linux Mandrake.

El objetivo es poder consultar, desde una aplicación la información de la base de datos, utilizando sólo las relaciones del esquema global por parte del programador. Esta aplicación se instalará en el sitio 1.

Para lograr ver el esquema global de manera transparente los pasos a realizar son los siguientes:

1. Crear desde el sitio 1 un servicio de conexión a la base de datos que se encuentra en el sitio 2. Esto se hace mediante el programa Oracle Net8 Easy Config, de la misma manera en que se logra conectar Oracle Forms con una base de datos local.

2. Crear un enlace a la base de datos del sitio 2 usando el servicio de conexión a dicha base de datos.

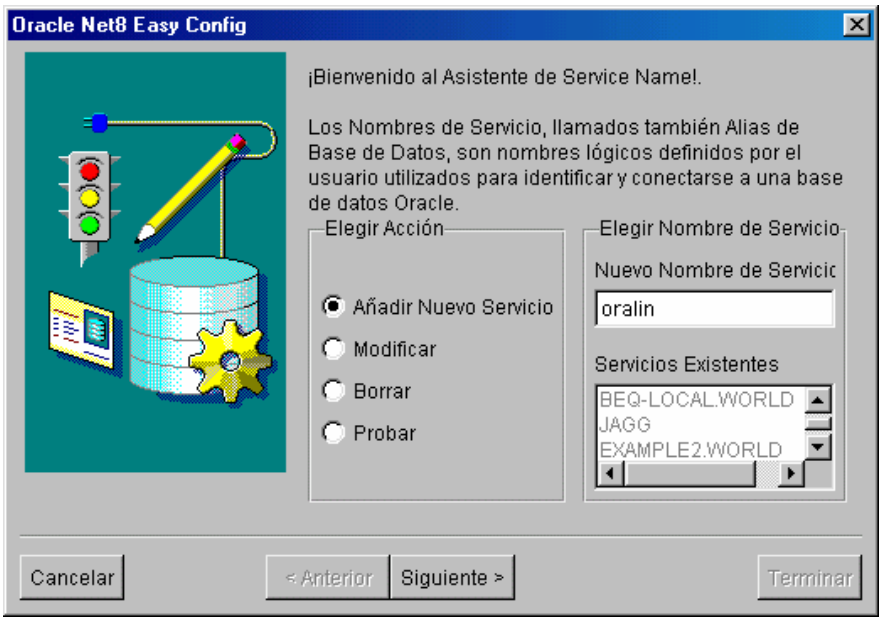
3. Crear un sinónimo en el sitio 1 por cada tabla del sitio 2. Los sinónimos son objetos de la base de datos y son referenciados como si fueran tablas.

4. Reconstruir las relaciones globales a partir de los sinónimos. Estas relaciones globales se representarán mediante vistas.

Creación del Servicio de Conexión

Vamos a suponer que la base de datos del sitio 1 recibe el nombre de *ora_w* (oracle sobre windows) y la del sitio 2 el de *ora_l* (oracle sobre linux).

Desde el sitio 1, ingresar al programa Oracle Net8 Easy Config y agregar el servicio **oralin**, para accesar la base de datos *ora_l*.



Después de elegir `Siguiente`, ingresar la siguiente información:

- Protocolo de Comunicaciones: **TCP/IP** u otro.
- Nombre del Host: Dirección IP (p.e. **200.31.205.19**) o nombre de máquina.
- Número de Puerto: **1521** por defecto
- SID de Base de Datos: Nombre de la base de datos del sitio 2. (p.e. **ora_l**).

Creación del Link y de los Sinónimos

En Maq1, ingresar a SQL*Plus, desde user1 y ejecutar la siguiente instrucción:

```
SQL> CREATE DATABASE LINK ora_l
       CONNECT  TO user2 IDENTIFIED BY user2
       USING 'oralin';

SQL> CREATE  PUBLIC  SYNONYM  clie2  FOR clie2@ora_l;
SQL> CREATE  PUBLIC  SYNONYM  pd2  FOR pd2@ora_l;
```

En este momento, en el sitio 1, se podrán ver en el catálogo las tablas clie1 y pd1 (fragmentos del sitio 1) y los sinónimos clie2 y pd2 (fragmentos del sitio 2), accesándolos como si fueran locales.

Creación de las Relaciones Globales

En el sitio 1, se crearán dos vistas con los nombres CLIENTE y PART_DESP, desde SQL*Plus:

```
SQL> CREATE VIEW cliente AS
       (SELECT * FROM clien1) UNION (SELECT * FROM clien2);

SQL> CREATE VIEW part_desp AS
       (SELECT * FROM pd1) UNION (SELECT * FROM pd2);
```

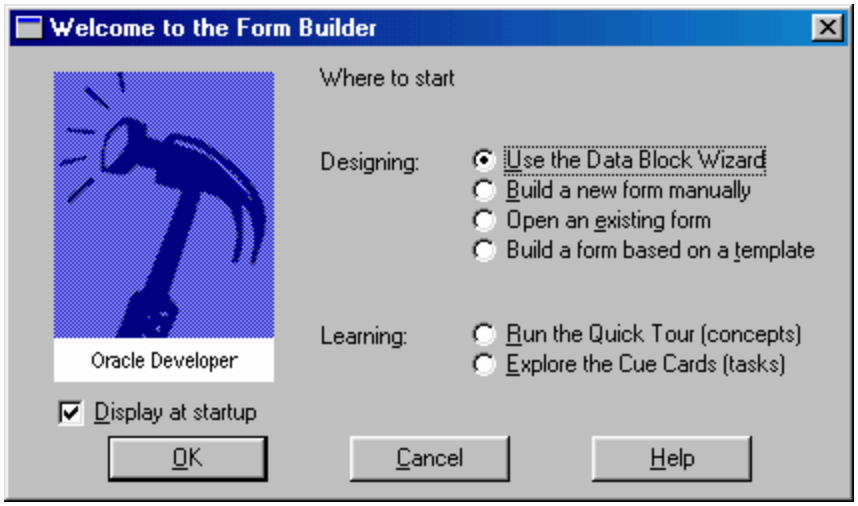
De esta manera hemos obtenido un esquema global para que sea accesado con transparencia para efectos de consulta por parte del usuario y para desarrollo de aplicaciones de consulta por parte del programador.

8.3 Introducción a Oracle Forms

Antes de ingresar a Oracle Forms, se debe subir la base de datos, tarea que hace el DBA, o si es monousuario (Windows) lo hace el programador, mediante el programa **Start Database**.

Para que Oracle Forms pueda interactuar con la base de datos se requiere de un programa que los comunique, denominado **lsnrctl**, el cual, en NT se ejecuta automáticamente, mientras que en Windows 98 y Linux hay que hacerlo manualmente:

Al ingresar a Oracle Forms (Forms Builder 6.0), aparece la pantalla de bienvenida *Wecome to the Form Builder*.



Al cancelar o seleccionar la opción **Build a new form manually**, se podrá trabajar con uno de los componentes de Forms denominado *Object Navigator*, el cual muestra todos los elementos de la Forma como Bloques de Datos, Triggers, Unidades de Programa, Ventanas y Alertas, entre otros.

Además muestra elementos que no constituyen parte de la Forma como Librerías PL/SQL y Objetos de la Base de Datos.

Son cinco categorías de elementos en Object Navigator:

Forms: Contiene todas las formas (p.e. Module1) y los objetos de cada una, tales como triggers, alertas, bloques de datos y canvas y unidades de programa.

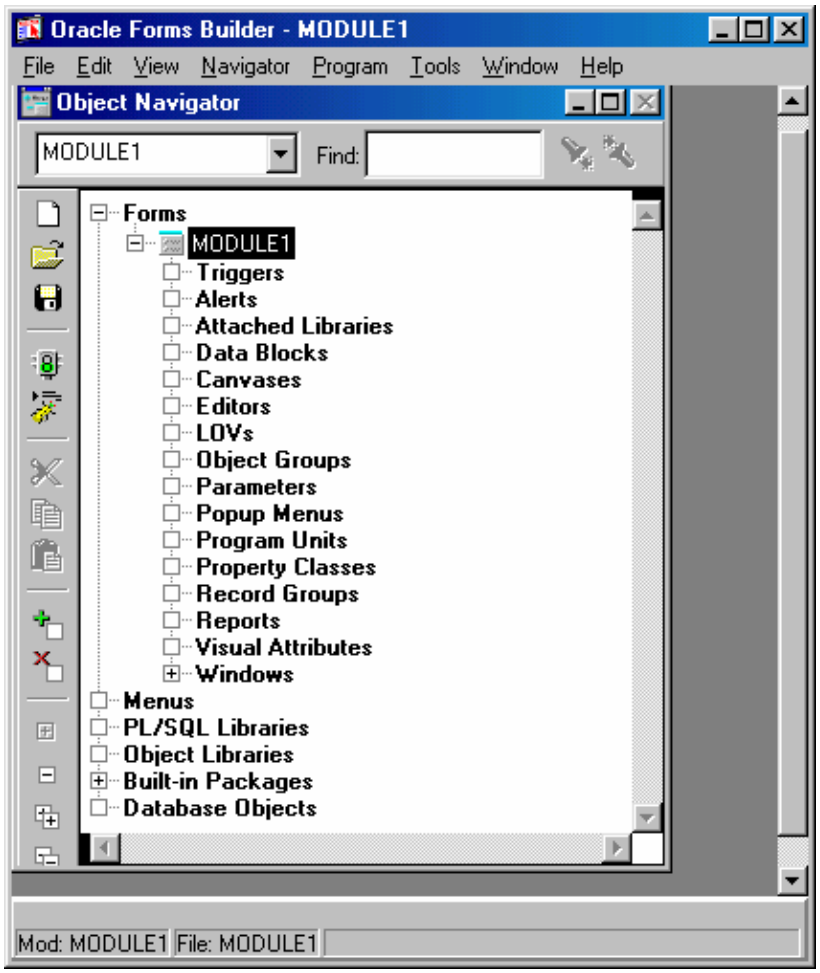
Menus: Contiene los menús y los objetos de cada uno tales como items, parámetros y unidades de programa.

PL/SQL Libraries: Son librerías, ya sea almacenadas en la base de datos o en archivos. Pueden ser compartidas por las formas y por los desarrolladores.

Object Libraries: Son librerías que pueden contener colecciones de objetos diversos.

Built-in Packages: Son paquetes almacenados por defecto para las formas, reportes y gráficas.

Database Objects: Contiene todos los objetos de la base de datos como tablas, vistas, unidades de programa almacenadas, etc.



El menú que aparece en la barra superior da acceso a casi todas las características de Oracle Forms:

File tiene opciones para crear, abrir, grabar e imprimir formas. También permite conectarse a la base de datos y compilar formas.

Edit permite cortar, copiar y pegar objetos, invocar un editor y deshacer una acción.

View permite intercambiar entre un despliegue de los elementos meramente visuales (visual view) y uno con todos los elementos del usuario (ownership view).

Navigator tiene opciones para colapsar y expandir el árbol de los elementos relacionados con la forma. También es usado para crear y borrar objetos.

Program maneja opciones para compilar, generar y ejecutar la forma. Permite controlar triggers, procedimientos y otros tipos de código.

Tools permite conmutar entre diferentes pantallas como el Layout editor (editor de presentación), el Menu editor y el Object Navigator. Tiene varios asistentes (wizards) que pueden activarse como el Data Block wizard y el Layout wizard.

Windows muestra las pantallas activas y en estilos como cascada u otros. Por defecto solo muestra el Object Navigator.

Help usado para ayudas.

Oracle Forms normalmente se refiere a una forma con el término **módulo**. Cuando Forms Builder se inicia, un módulo por defecto se crea: MODULE1.

El primer paso para trabajar con la base de datos a través de Forms Builder es conectarse a ella a través de la opción Connect del menú File.

Mediante una forma en Oracle Forms podemos hacer operaciones DML sobre una o varias tablas.

En general, una forma está constituida de uno o más bloques de datos. Un bloque de datos, normalmente, corresponde a una tabla base de la base de datos. Para cada tabla que se tenga que mostrar en la forma, debe crearse un nuevo bloque de datos. Un bloque de datos está formado por varios campos. Un campo dentro del bloque de datos puede corresponder a una columna de la tabla base, aunque no siempre.

Existen cuatro tipos de formas que pueden diseñarse mediante Oracle Forms:

1. Formas simples: Constan de un solo bloque de datos correspondiente a una tabla de la base de datos.

2. Formas simples con campos *lookup*: Contienen un solo bloque de datos correspondiente a una tabla con uno o más campos

que muestran datos de otras tablas, denominados *looked up*, lo que quiere decir, que son buscados en otra tabla.

3. Formas maestro-detalle: Contienen dos bloques de datos que corresponden a dos tablas que tienen una relación 1:N.

4. Formas maestro-detalle con campos *lookup*: Son similares a las formas maestro-detalle, pero adicionalmente tienen campos *lookup*, ya sea en el bloque maestro o en el bloque detalle.

8.4 Creación de formas simples

Crearemos un solo bloque de datos para la tabla (vista en nuestro caso) CLIENTE. Un bloque de datos aparece sobre un canvas, el cual está contenido en una ventana. Se pueden llegar a tener múltiples canvas y múltiples ventanas por forma. En este caso se hará con un solo canvas y una sola ventana.

Una forma se puede crear manualmente o de manera asistida. Oracle Forms provee dos asistentes para crear bloques de datos:

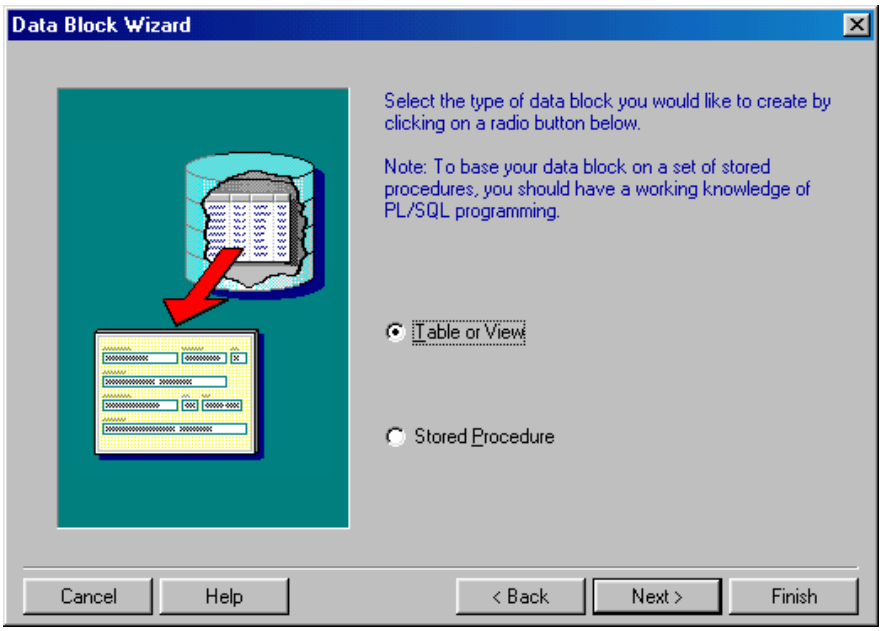
- El Asistente de bloque de datos (Data Block wizard), el cual permite escoger la tabla base y las columnas.

- El Asistente de presentación de datos (Layout wizard), el cual permite organizar la tabla base y las columnas sobre la forma.

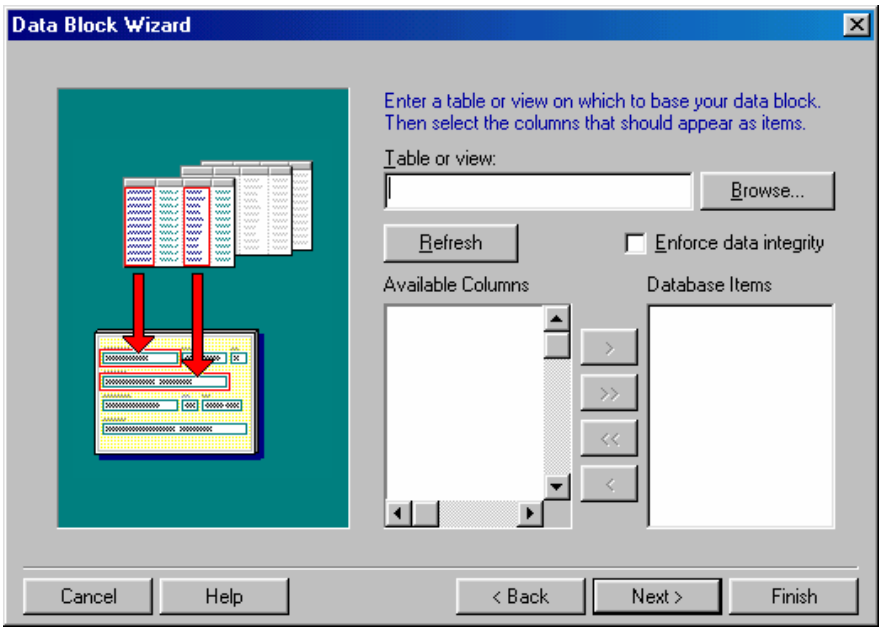
Para crear un bloque nuevo, de la barra superior seleccione el menú de herramientas y luego podrá activar la opción de Asistente de bloque de datos, para obtener una ventana con un mensaje de bienvenida. Aquí sencillamente se procede con el siguiente paso. En resumen, la secuencia es la siguiente:

`Tools → Data Block wizard → Next`

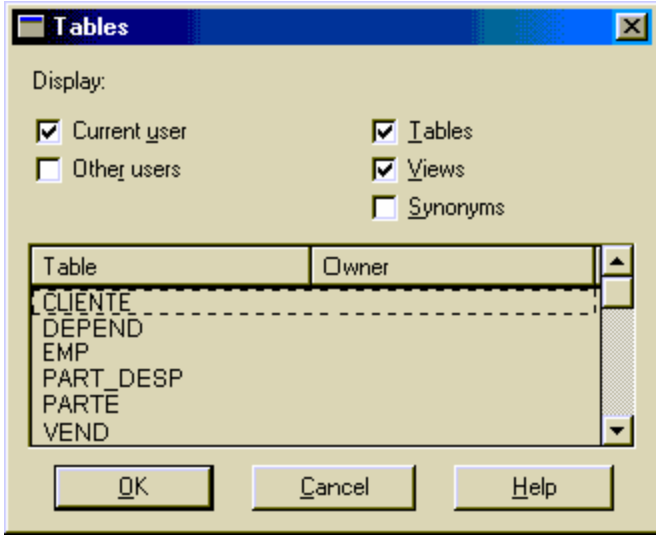
Luego aparecerá la siguiente caja de diálogo:



En este punto se selecciona `Table or View` y se hace click en el botón `Next`, apareciendo la siguiente caja de diálogo:

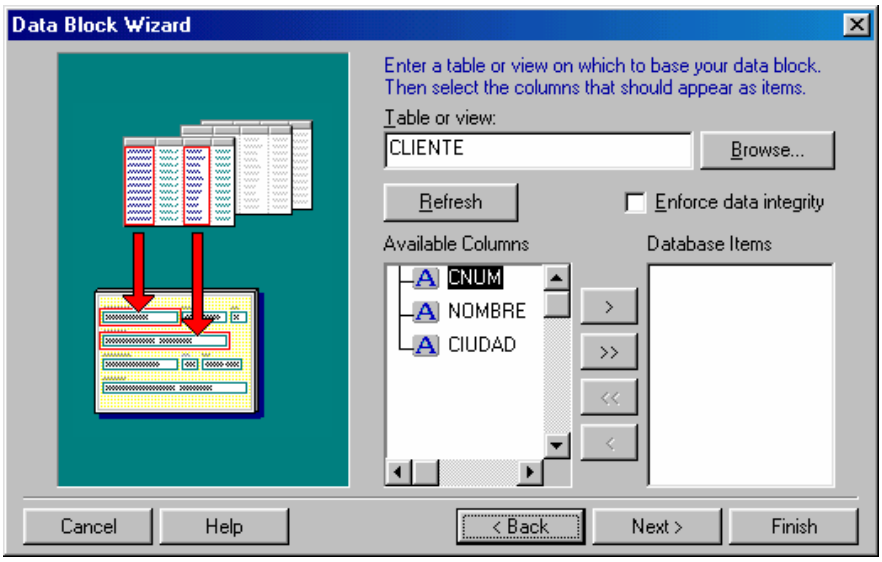


Si está seguro se entra el nombre de la vista o tabla base, de lo contrario se busca con Browse, en cuyo caso aparece la siguiente caja de diálogo:

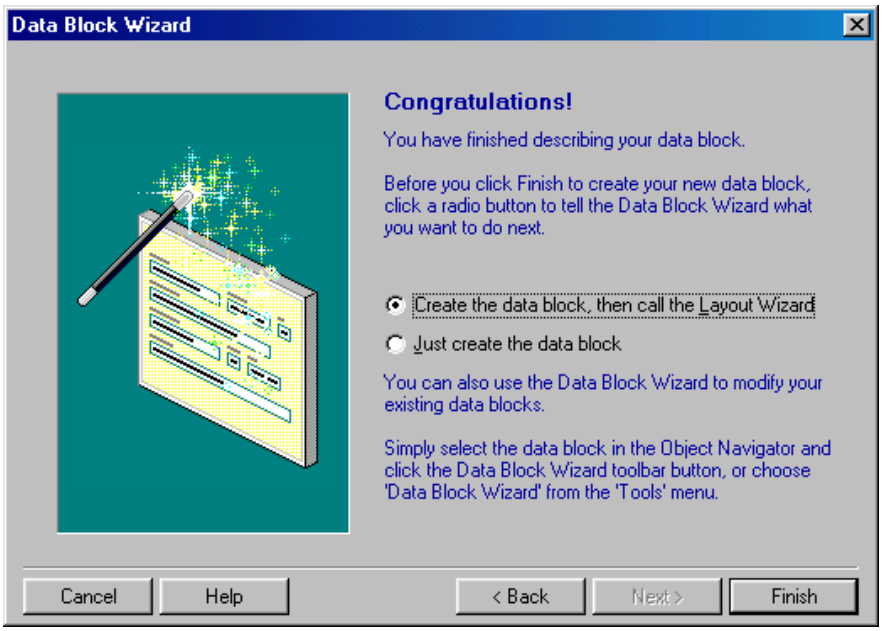


Hay que seleccionar la tabla base. En nuestro caso debemos hacer click en Current User y en Views y luego escoger la vista global CLIENTE. Después de resaltada se hace click en OK.

El asistente reaparecerá con la tabla (o la vista) seleccionada y las columnas disponibles.



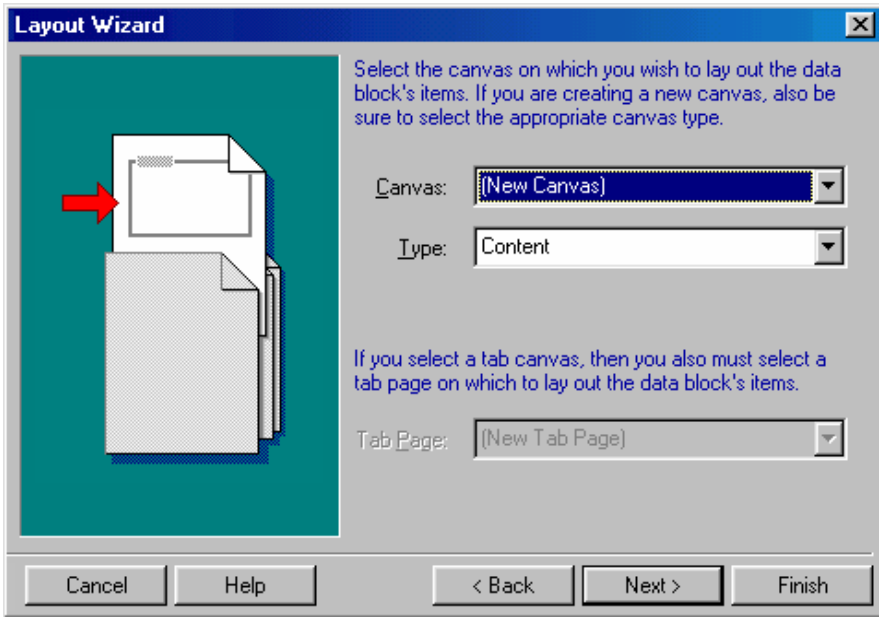
De estas columnas disponibles se escogerán las que formen parte del bloque de datos de la forma, resaltándolas y trasladándolas, una por una, a la derecha (a Database Items) mediante el botón **>** o desplazándolas todas mediante **>>**. Se debe hacer click en Next, para que aparezca la caja de diálogo final del asistente del bloque de datos.



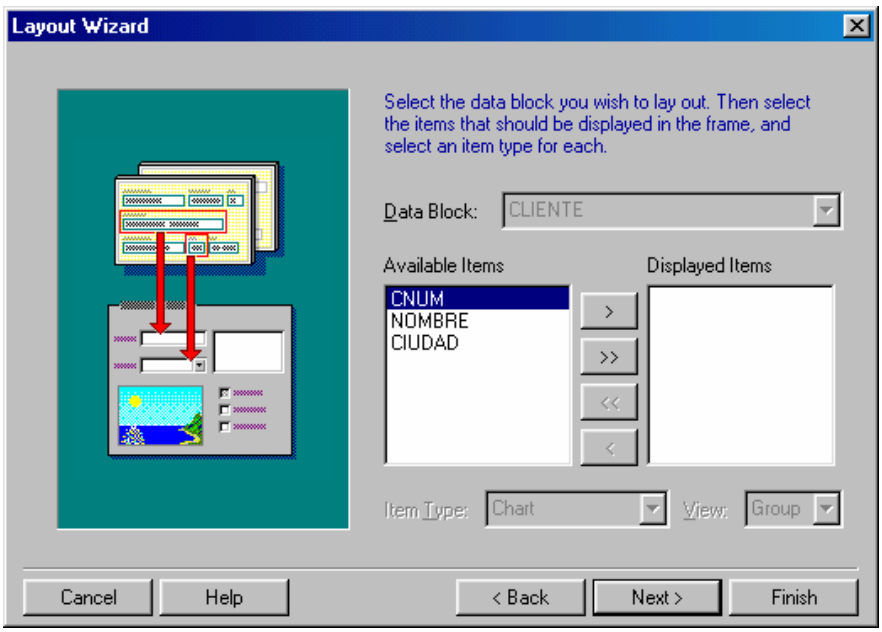
Para efectos de organización y distribución de la forma, se debe proseguir con el asistente de presentación, escogiendo la opción Create the data block, then call the Layout Wizard y hacer click sobre Finish. El bloque de datos es creado y luego el mensaje de bienvenida del asistente de presentación (Layour Wizard) aparecerá. Haga click sobre el botón Next para continuar con el asistente.

Un canvas es como el *lienzo* sobre el que se *pintan* los items del bloque de datos.

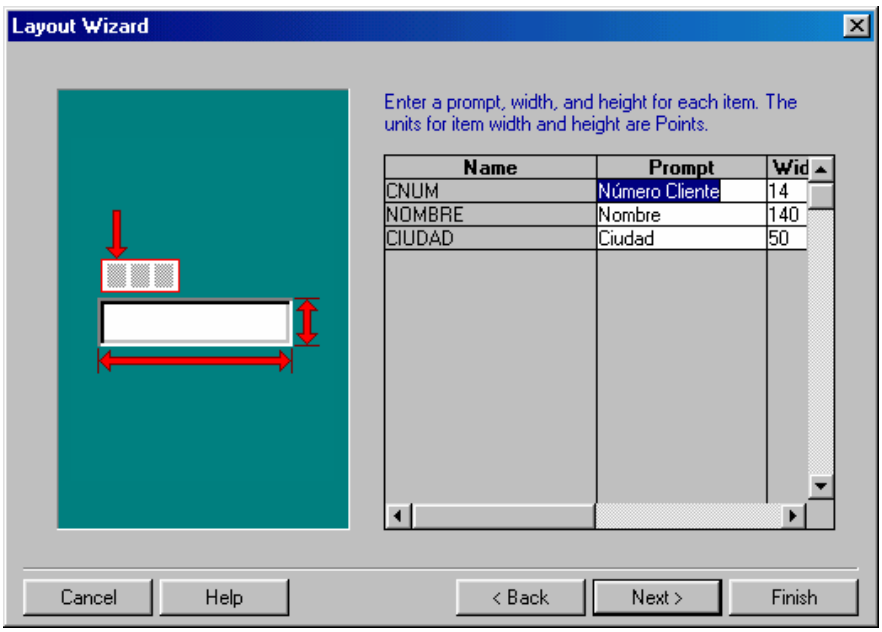
Hay diferentes tipos de canvas: content, stacked, Vertical Toolbar, Horizontal Toolbar y Tab.



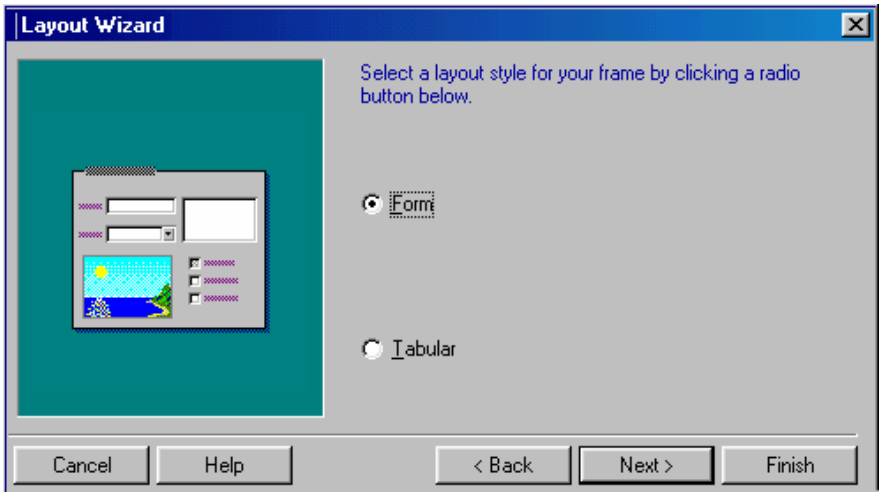
La presentación de un bloque de datos se puede localizar en un canvas. Al no haber canvas, la única opción es (New Canvas) y el tipo, en este caso, será Content. Presione Next. Aparecerá la siguiente caja de diálogo:



Movemos los items disponibles (Available Items) hacia el área de items a mostrar (Displayed Items) mediante **>>**. Se presiona Next.



En este punto, se puede cambiar el texto del prompt o etiqueta de cada uno de los items para dar mayor claridad a quien lee la interfaz. (Se cambió Cnum por Número Cliente). Los tamaños de los campos y la altura de cada uno también se pueden modificar. Se presiona Next. Aparecerá la siguiente caja de diálogo:

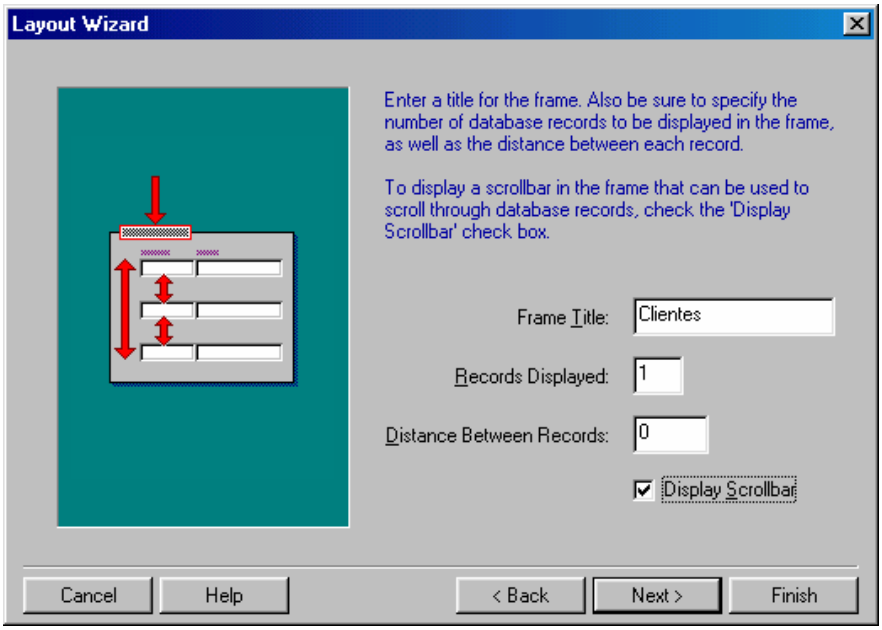


Existen dos estilos de presentación para las formas:

- Form: Ubica las etiquetas de los campos a la izquierda del campo. Normalmente se utiliza para un registro mostrado a la vez.

- Tabular: Utilizado para una lista de registros, ubicando las etiquetas de los campos en la parte superior.

Debe seleccionarse Form y presionar Next. Viene la siguiente caja de diálogo:



- Título del Marco (Frame Title): Clientes.

- Cantidad de Registros mostrados (Records Displayed). Para el estilo Form debe ser 1. (En el Tabular debe ser superior a 1).

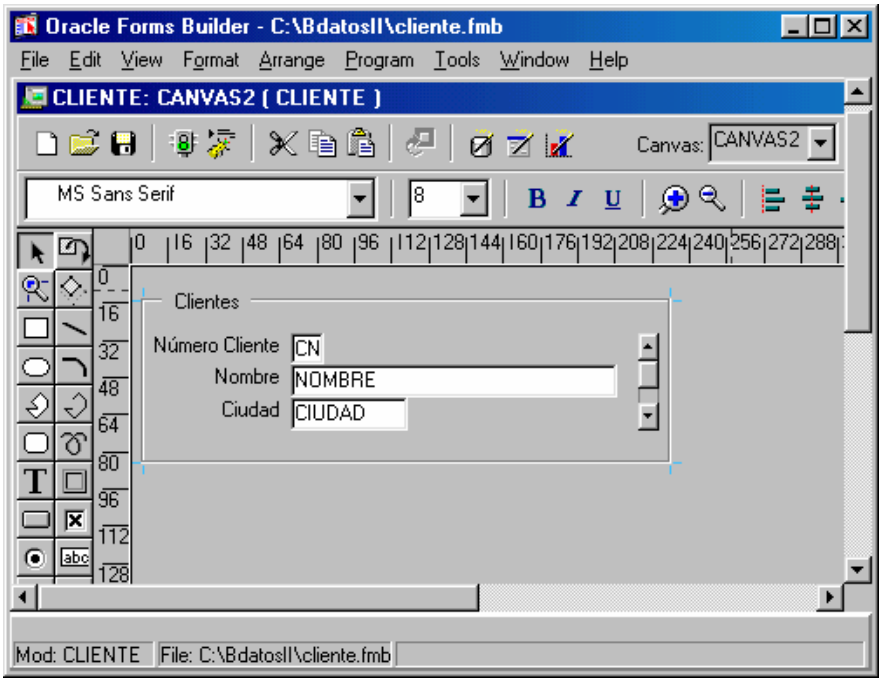
- Distancia entre registros: Puede ser 0.

En esta ventana debe chequearse la opción Display Scrollbar. Debe hacerse click en Next para que aparezca la ventana final, donde se debe seleccionar el botón Finish para crear la presentación.

El Editor de Presentación

En este punto, el Editor de Presentación (Layout Editor) será mostrado con la forma, tal como va desarrollada, y el bloque de datos creado hasta el momento.

En caso de que la forma no esté exhibida por el Editor de Presentación, se activa mediante Tools → Layout Editor.



En el Editor de Presentación, los campos y las etiquetas se pueden mover haciendo click y arrastrando con el mouse. Si se desea adicionar texto, líneas, marcos u otros elementos, se puede utilizar la barra de herramientas de la izquierda.

Si se desea cambiar el texto de una etiqueta, se debe seleccionar el botón de Texto T de la barra de herramientas, dar click en la etiqueta y proceder a cambiar el texto. Se confirma haciendo click por fuera de la etiqueta. También se puede adicionar una etiqueta con texto seleccionando el botón de texto, haciendo click en un área abierta y digitando el texto.

Para cambiar el tipo de letra de un campo o etiqueta, use el Apuntador para señalar dicha etiqueta, vaya al menú Format, luego escoja la opción Font y finalmente seleccione el tipo de letra. También se puede lograr señalando el campo o la etiqueta y

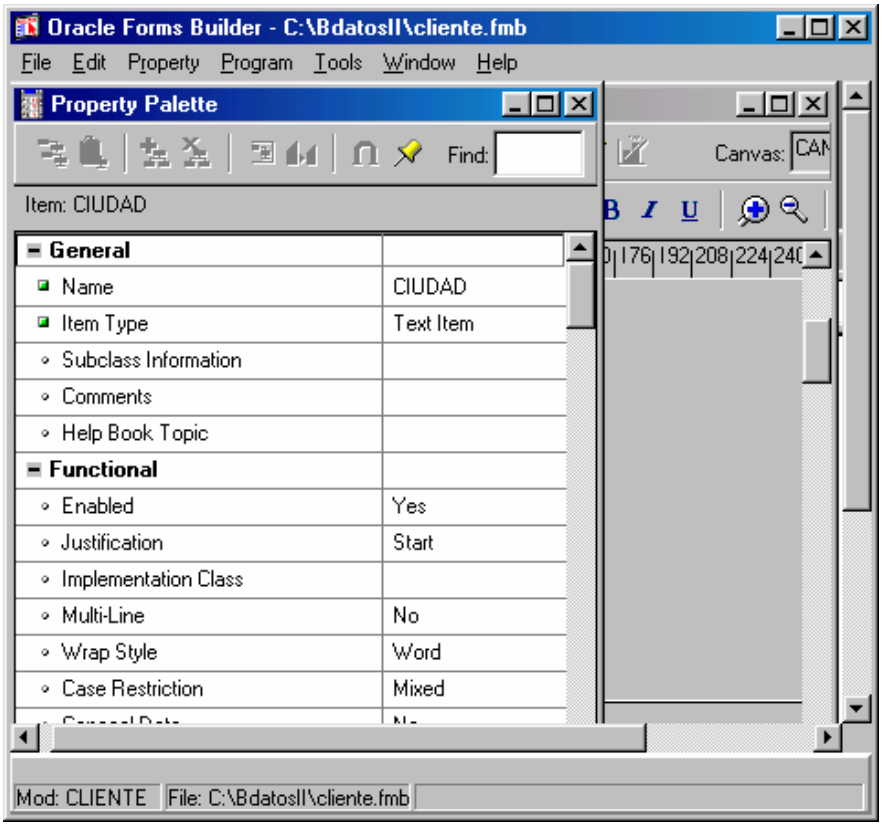
seleccionando directamente el tipo de letra desde la barra de tipos de letras presente en el Editor de Presentación.

Si se requiere cambiar el color de un texto, use el botón Apuntador, señalando la etiqueta o el campo y haciendo click sobre el botón de Color de Texto para finalmente seleccionar el color.

Para cambiar el tamaño de un campo, seleccione el campo, arrastre con el mouse uno de los lados y determine el tamaño.

La Paleta de Propiedades

Estando en el Editor de Presentación y haciendo doble click en algún objeto de la forma, aparece la Paleta de Propiedades con las características del objeto. También se logra haciendo click derecho sobre el objeto y escogiendo, en el recuadro que aparece, la opción de `Property Palette`.



Por ejemplo, la Paleta de Propiedades del item Ciudad muestra que su nombre es Ciudad, el tipo de item es de Texto y se justifica al comienzo del campo (no al final ni al centro).

Para ver la Paleta de Propiedades del bloque, estando en el Object Navigator se aplica click derecho sobre el bloque. También se puede hacer en el Editor de Presentación marcando la barra de desplazamiento vertical (scroll bar) y haciendo click derecho.

La Paleta de Propiedades del bloque muestra, en el grupo de propiedades de **Database**, algunas características importantes como las siguientes:

- Enforce Primary Key: Para forzar llave primaria en el bloque. Esto obliga que haya un campo dentro del bloque con esta misma característica.
- Query Allowed: Para permitir consultas
- WHERE Clause: Para especificar una cláusula WHERE con el fin de hacer selección sobre las filas de la tabla base.
- ORDER BY Clause: Para ordenar los registros.
- Insert Allowed: Para permitir insertar en la tabla base.
- Update Allowed: Para permitir actualizar la tabla base.
- Delete Allowed: Para permitir borrar filas de la tabla base.

Grabar, Compilar y Ejecutar

Como acto final, después de todos los ajustes requeridos, la forma se debe grabar en medio magnético, escogiendo del menú la opción File y luego Save. Mediante la caja de diálogo se escoge el directorio donde quedarán las formas de la aplicación que se está desarrollando y se graba, por ejemplo, con el nombre cliente. Deberá llevar la extensión *fmb*.

El proceso de compilación genera un archivo con extensión *fmx*, el cual es usado para ejecutar la forma. Los errores aparecerán en un archivo con extensión *err*.

Para compilar la forma vaya al menú y siga la siguiente secuencia: File → Administration → Compile File. Sin embargo, Oracle Forms por defecto compila la forma automáticamente cada vez que es ejecutada. Otra opción de compilación en el menú es a través de Program → Compile.

Para ejecutar la forma se debe ir al menú y seguir la secuencia Program → Run Form → Client/Server. En Windows 98 utilizando

la combinación Ctrl-F1, se conoce el uso de las teclas de función para controlar la forma y manejar los datos a través de la forma.

8.5 Creación de Formas Maestro-Detalle

Una forma maestro-detalle es aquella que tiene dos bloques dispuestos en una relación maestro-detalle, es decir, en una relación 1:N entre entidades, tal como lo muestra el diagrama Entidad-Relación. El primer bloque de la tabla corresponde a la tabla maestra (o tabla padre) y el segundo bloque corresponde a la tabla detalle (o tabla hija).

Las formas no afectan el esquema de la base de datos, en el sentido de crear, eliminar o reforzar restricciones de integridad referencial.

La relación que se va a mostrar es la existente entre la relación global (vista en el caso de Oracle) CLIENTE y la relación global PART_DESP (vista utilizada para las partes despachadas a los clientes), donde a un cliente corresponderán varias partes despachadas.

Creación del Bloque Maestro

Se procede a crear un bloque simple (el mismo creado en la sección anterior) para la relación global CLIENTE. Cuando hay que crear una nueva forma y ya se está trabajando en Oracle Forms se debe usar la secuencia `File` → `New` → `Form`. En nuestro caso, continuaremos con la misma forma.

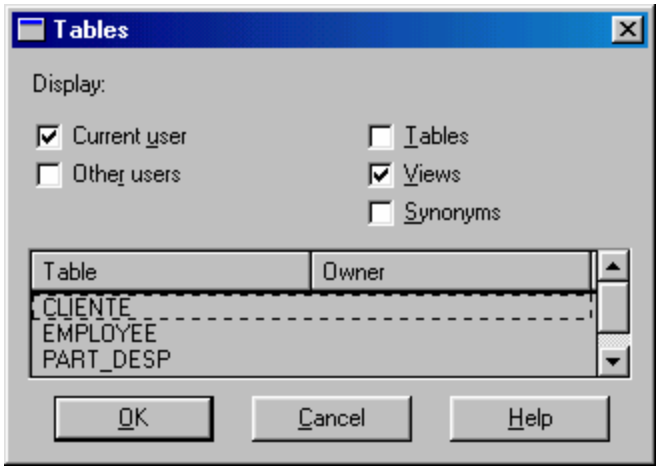
Después de creado el bloque CLIENTE se procede a crear un bloque detalle PART_DESP y se asocia con el bloque maestro.

Creación del Bloque Detalle

En el árbol de Object Navigator se debe seleccionar Data Blocks. Ir al menú `Tools` y luego a la opción `Data Block Wizard`. Confirmar con `Next`.

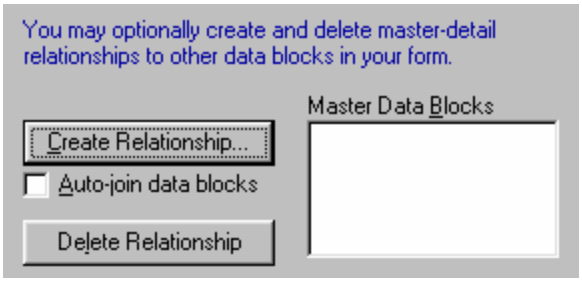
Igual que con el Bloque Simple, se selecciona `Table or View` y se hace click en el botón `Next`.

Por medio del botón Browse, se hace una exploración de los objetos del usuario, chequeando en la caja de diálogo que aparece la opción Current User y la opción Views.



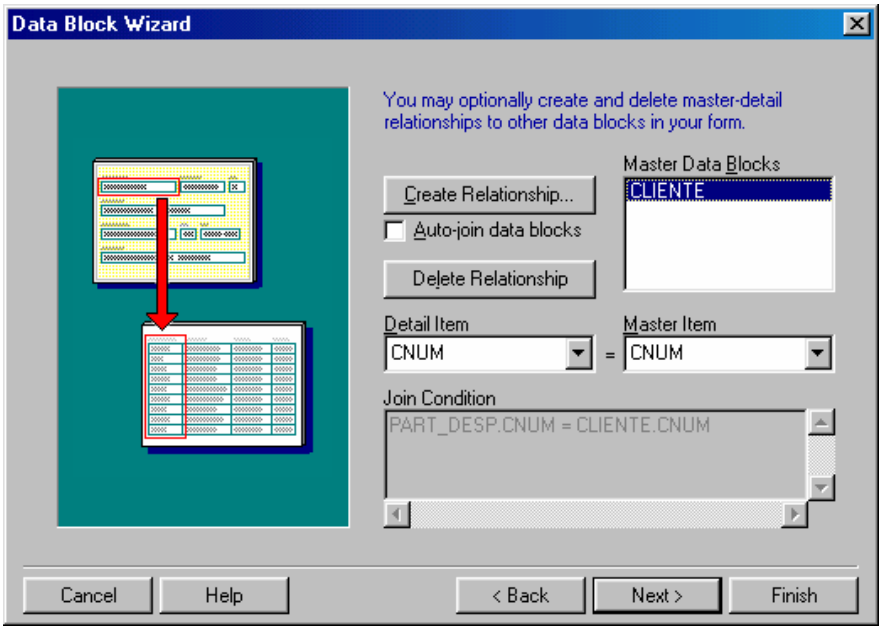
Luego se selecciona la vista PART_DESP y se incluyen todas las columnas (Cnum, Pnum, Dnum y Cant).

Cuando aparece la ventana con el botón Create Relationship, se deshabilita la opción Auto-join data blocks.



Se selecciona el botón Create Relationship, luego la opción Based on a join condition y se confirma con OK.

Se selecciona el bloque de datos CLIENTE (bloque maestro), se escogen los items relacionadores (Cnum en el Item Detalle y Cnum en el Item Maestro) para que se establezca la condición del join PART_DESP.CNUM = CLIENTE.CNUM y se confirma con Next.



Escoja la opción Create the data block, then call the Layout Wizard y hacer click sobre Finish.

Al canvas creado para el bloque CLIENTE, el sistema le dio el nombre de CANVAS2. Sobre este canvas se ubicarán los items de datos del bloque.

Se seleccionan los items de datos Pnum, Dnum y Cant y se omitirá Cnum, ya que éste está ubicado en el bloque maestro.

Se puede cambiar el texto del prompt o etiqueta de cada uno de los items.

Se selecciona el estilo Tabular para la presentación de los datos del bloque.

Se ingresa el título del marco (Frame Title): PARTES DESPACHADAS y la cantidad de registros a mostrar (p.e. 5).

Se graba la forma, se compila y se ejecuta.