Bases de Datos Distribuidas

1. Evolución de sistemas de bases de datos a bases de datos distribuidas
2. Niveles de distribución de los datos y procesos
3. Características de transparencia de la base de datos distribuida
4. Diseño de bases de datos distribuidas
5. Traslación de consultas globales a consultas fragmentadas
6. Administración de transacciones distribuidas
7. Seguridad
8. Multibases de Datos

1 Evolución de sistemas de bases de datos a bases de datos distribuidas
Objetivo: El alumno explicará la evolución de las bases de datos y la importancia de las Bases de Datos Distribuidas

1.1 Introducción

Existen dos fuerzas que han impulsado la evolución de los sistemas de bases de datos. Por un lado los **usuarios** como parte de organizaciones más complejas han **demandado** una serie de capacidades que se han ido incorporando en los sistemas de bases de datos. Un ejemplo de esto es la **necesidad de integrar información proveniente de fuentes diversas**. Por otro lado, la **tecnología ha hecho posible** que algunas facilidades inicialmente imaginadas solo en sueños se conviertan en realidad. Por ejemplo, las transacciones en línea que permite el sistema bancario actual no hubiera sido posible sin el desarrollo de los equipos de comunicación.

a) La presión de los usuarios por datos distribuidos

Casi cualquier organización que ha incorporado sistemas de información para su funcionamiento ha experimentado dos fases:

En la primera fase, se ha agrupando **toda la información en un solo luga**r. La idea original era que todos los accesos a datos podrían ser integrados en un solo lugar usando herramientas de bases de datos tales como lenguajes de descripción de datos, lenguajes de manipulación de datos, mecanismos de acceso, verificadores de restricciones y lenguajes de alto nivel. Para poder tener estos mecanismos de almacenamiento y recuperación de información, las organizaciones hicieron fuertes inversiones en equipos computacionales sofisticas y con grandes capacidades. Sin **embargo**, después de experimentar por un tiempo con este enfoque, muchas organizaciones encontraron que el sistema completo era satisfactorio, en algún grado, para un buen número de usuarios pero muy pocos obtenían un servicio óptimo. Más aún, **bajo este esquema centralizado los "propietarios" u originadores de la información específica perdieron el control sobre el manejo de su información ya que ésta no se almacenaba en sus lugares de trabajo.**

En la segunda fase se promovió la descentralización de los sistemas de bases de datos corporativos. Entonces, se **empezaron a adquirir sistemas de software y hardware departamentales.** Este enfoque presentó grandes beneficios para el control de la seguridad de la información y la disponibilidad de la misma. Permitió que los esquemas de mantenimiento y planeación de los sistemas de información afectara en menor medida al funcionamiento general de la organización.

Sin embargo, muy pronto empezaron a aparecer inconvenientes con este enfoque. **Se presentaron problemas de consistencia de la información entre los sistemas locales y central y se hallaron dificultados al transferir información de entre departamentos diferentes de una corporación.**

En una tercera fase (la cual aún no ha concluido) se ha tratado de formalizar la descentralización de las bases de datos y de sus funciones manteniendo la integridad de la información y quizá algún tipo de control centralizado o distribuido.

a) La presión de la tecnología a distribuir sobrecarga de I/O y la velocidad de redes

Existen buenas razones técnicas para distribuir datos. La más obvia es la referente a la sobrecarga de los canales de entrada y salida a los discos en donde se almacena finalmente la información. **Es mucho mejor distribuir los accesos a la información sobre diferentes canales que concentrarlos en uno solo**. Otra razón de peso es que las redes de computadoras empezaron a trabajar a velocidades razonables abriendo la puerta a la distribución del trabajo y la información.

1.2 Bases de Datos Distribuidas contra Bases de Datos Centralizadas

Las Bases de Datos Distribuidas corresponden a una Base de datos virtual compuesta por varias Bases de Datos reales que se encuentran ubicadas en lugares físicos diferentes. Cada sitio tiene su base de datos, sus usuarios, su manejador de base de datos con capacidad de procesamiento autónomo, sin embargo, también soportan transacciones distribuidas que garantizan la consistencia cuando las aplicaciones accedan más de una sola base de datos en tiempo real. Esto se logra usando el protocolo de compromiso de dos fases (2PC).

Las Bases de Datos Distribuidas fueron diseñadas para soportar una sola aplicación en donde los datos están físicamente distribuidos y tomar ventaja de los sistemas de comunicación, de las facilidades del particionamiento, replicación, y los mecanismos de control de concurrencia y procesamiento distribuido entre otros, proporcionando un mejor rendimiento y tiempo de respuesta. En este tipo de sistemas de bases de datos es en donde existe mayor interdependencia entre las bases de datos componentes y se diseñan estos componentes desde el principio, utilizando el mismo Sistema Manejador de Bases de Datos en los sitios implicados a fin de evitar traducciones de un modelo de datos a otro.

El hacer una descentralización de la información se justifica desde el punto de vista tecnológico por las siguientes razones:

- Para permitir autonomía local y promover la evolución de los sistemas y los cambios en los requerimientos de usuario.
- Para proveer una arquitectura de sistemas simple, flexible y tolerante a fallas.
- Para ofrecer buenos rendimientos.

Aunque la idea de distribución de datos es bastante atractiva, su realización conlleva la superación de una serie de dificultades tecnológicas entre las que se pueden mencionar:

- Asegurar que el acceso entre diferentes sitios o nodos y el procesamiento de datos se realice de manera eficiente, presumiblemente óptima.
- Transformar datos e integrar diferentes tipos de procesamiento entre nodos de un ambiente distribuido.
- Distribuir datos en los nodos del ambiente distribuido de una manera óptima.
- Controlar el acceso a los datos disponibles en el ambiente distribuido.
- Soportar la recuperación de errores de diferentes módulos del sistema de manera segura y eficiente.
- Asegurar que los sistemas locales y globales permanezcan como una imagen fiel del mundo real evitando la interferencia destructiva que pueden ocasionar diferentes transacciones en el sistema.

Así también, la aplicación de técnicas de distribución de información requiere de superar algunas dificultades de índole organizacional y algunas otras relacionadas con los usuarios. Entre ellas se puede mencionar:

- El desarrollo de modelos para estimar la capacidad y el tráfico esperado en el sistema distribuido.
- Soportar el diseño de sistemas de información distribuidos. Por ejemplo, ayudar a decidir donde localizar algún dato particular o donde es mejor ejecutar un programa de aplicación.
- Considerar la competencia que habrá por el uso de los recursos entre nodos diferentes.

Aun cuando las dificultades mencionadas son importantes, las ventajas de la distribución de información han promovido su aplicación en ambientes del presente y del futuro.

Resumen

1.2.1 Ventajas

Ventajas de las bases de datos distribuidas con respecto a las centralizadas

Ventajas de las Bases de Datos Distribuidas

- **Compartimiento de datos**. Los usuarios de un nodo son capaces de acceder a los datos de otro nodo.
- **Autonomía**. Cada nodo tiene cierto grado de control sobre sus datos, en un sistema centralizado, hay un administrador del sistema responsable de los datos a nivel global. Cada administrador local puede tener un nivel de autonomía local diferente.
- **Disponibilidad.** Si en un sistema distribuido falla un nodo, los nodos restantes pueden seguir funcionando. Si se duplican los datos en varios nodos, la transacción que necesite un determinado dato puede encontrarlo en cualquiera de los diferentes nodos.

1.2.2 Desventajas

Ventajas de las bases de datos centralizadas con respecto a las distribuidas

Inconvenientes de la Bases de Datos Distribuidas

- **Coste de desarrollo del software**. La complejidad añadida que es necesaria para mantener la coordinación entre nodos hace que el desarrollo de software sea más costoso.
- **Mayor probabilidad de errores.** Como los nodos que constituyen el sistema funcionan en paralelo, es más difícil asegurar el funcionamiento correcto de los algoritmos, así como de los procedimientos de recuperación de fallos del sistema.

- **Mayor sobrecarga de procesamiento.** El intercambio de mensajes y ejecución de algoritmos para el mantenimiento de la coordinación entre nodos supone una sobrecarga que no se da en los sistemas centralizados.

2 Niveles de distribución de los datos y los procesos

Objetivo: El alumno explicará cómo se deben de distribuir los datos y que factores se deben de considerar para ello

Los datos se pueden localizar en una sola base de datos ubicada en un solo sitio dentro de un disco local.
Los datos se pueden localizar en una sola base de datos ubicada en un solo sitio dentro de diferentes discos locales.
Los datos se pueden localizar en una sola base de datos ubicada en diferentes sitios y por tanto dentro de diferentes discos locales.

2.1 Procesamiento en un solo sitio, datos en un solo sitio (SPSD)

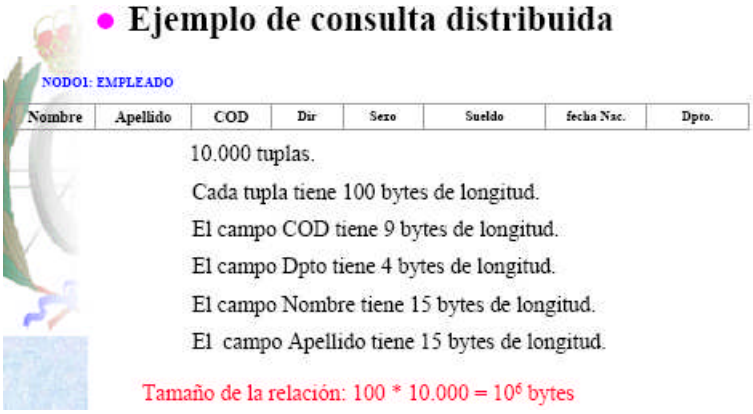
Todo el procesamiento se realiza en un solo CPU, y todos los datos se guardan en el disco local de la computadora anfitriona. El procesamiento no puede ser realizado del lado del usuario del sistema. Ej. Mainframe o minicomputadoras. EL DBMS se localiza en la computadora anfitriona, la cual es accesada por terminales no inteligentes. Primera generación de BD de microcomputadora para un solo usuario.

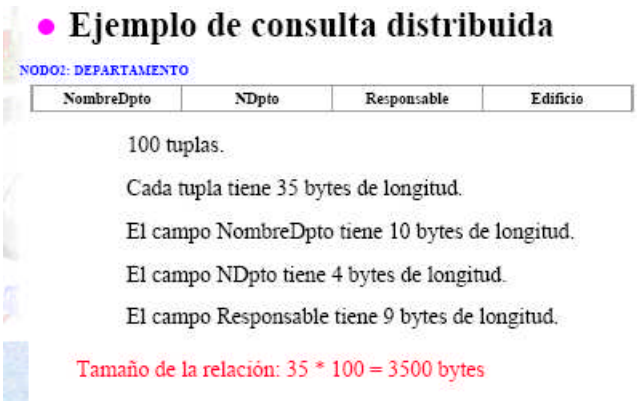
2.2 Procesamiento en sitios múltiples, datos en un solo sitio (MPSD)

Se realizan procesos multiples en diferentes computadoras que comporten una sola base de datos. Se requiere un servidor de archivos de red que ejecuta aplicaciones convencionales que son accesadas por LAN.

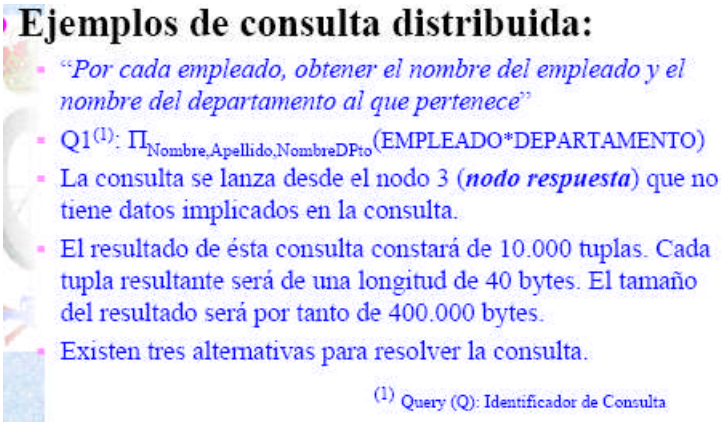
2.3 Procesamiento en sitios múltiples, datos en sitios múltiples (MPMD)

Se realizan procesos multiples en diferentes computadoras que comporten varios dépositos de datos.





Ejemplos de consulta distribuida:



Primera alternativa

Transferir tanto la relación EMPLEADO como la relación DEPARTAMENTO al nodo respuesta (nodo 3) y realizar allí mismo la operación de join. En este caso se transfieren 1000000 + 3500 = 1003500 bytes

Segunda alternativa
Transferir la relación EMPLEADO al nodo 2, ejecutar el join en este nodo y enviar el resultado al nodo 3. Esto implica transferir 1000000 + 400000 (resultado) = 1400000 bytes

Tercera alternativa
Transferir la relación DEPARTAMENTO al nodo 1, ejecutar el join en este nodo y enviar el resultado al nodo 3. En este caso se transfieren 3500 + 40000 (resultado) 403500 bytes

Esta última alternativa es la óptima: Origen del semijoin
Proceso distribuido de consultas usando el semijoin
Reducción del número de tuplas antes de ser transferidas a otro nodo
Se envía la columna con la que se va a realizar el join de una relación R al nodo donde se encuentra la otra relación, ahí se realiza el join con la otra relación S
Se envían las columnas implicadas en el resultado al nodo inicial y se vuelve a realizar el join con R.
Solo se transfieren las columnas de R que intervienen en la realización del join en una dirección y el subconjunto de columnas de S resultantes en la otra

Proceso distribuido de consultas utilizando semijoin
Semijoin de las consultas Q1 y Q2 Paso 1
Consulta Q1: proyección de DEPARTAMENTO sobre atributos que van a intervenir en la operación de join y transferencia al nodo 1
Consulta Q1 FI: *Π* nodepto (DEPARTAMENTO)
Resultado 4 bytes del atributo nodepto por 100 registros de DEPARTAMENTO = 400 bytes transferidos
Consulta Q2 F2: *Π*jefe (DEPARTAMENTO)
Resultado 9 bytes del atributo jefe por 100 registros de DEPARTAMENTO = 900 bytes transferidos
Semijoin de las consultas Q1 y Q2 Paso 2

Consulta Q1: realización del join de las tuplas transferidas en el paso anterior. Transferencia del resultado del join de nuevo al nodo 1. Se transfieren solo los atributos necesarios para realizar el join final
R1: *Π* depto, Nombre, Apellido (Fx Empleado)
Tamaño (4+15+15)* 10000 = 340 000 bytes transferidos

Para la consulta Q2
R2: *Π* jefe, Nombre, Apellido (Fx Empleado)
Tamaño (9+15+15)* 100 = 3900 bytes transferidos

3 Características de transparencia de la base de datos distribuida
Objetivo: El alumno explicará la importancia de la transparencia en la distribución, transacción, desempeño y consultas en Bases de datos distribuidas.

Una base de datos requiere características funcionales que pueden ser agrupadas y descritas como características de transparencia. Estas tienen la propiedad de hacer pensar al usuario final que él o ella

está trabajando con un sistema de base de datos centralizado, todas las complejidades de una base de datos distribuida son ocultas o transparentes al usuario.

3.1 Transparencia de distribución

Permite a una base de datos distribuida ser tratada como una sola base de datos lógica, de tal forma que el usuario no necesita saber

Que los datos están fragmentados (transparencia de fragmentación)

Que los datos pueden estar replicados en varios sitios (transparencia de ubicación)

La ubicación de los datos

El nivel de transparencia soportados por un DDBMS varia de un sistema a otro. Se presentan a continuación tres niveles de transparencia:

a) **Transparencia de Fragmentación** es el nivel más alto de transparencia. El usuario final o programador no necesitan saber si la base de datos esta particionada. Por tanto, ni los nombres de los fragmentos ni la ubicación de cada fragmento son especificados al accesar los datos.

b) **Transparencia de ubicación** existe cuando el usuario final o programador debe especificar los nombres de los fragmentos de la base de datos pero no necesitan especificar en donde se encuentran esos fragmentos.

c) **Transparencia de mapeo local** existe cuando el usuario final o programador debe especificar tanto los nombres de los fragmentos como sus ubicaciones.

Ejemplo, sea el siguiente esquema:

EMPLEADOS (NOEMP, NOMBRE, FECHANAC, DIRECCION, DEPARTAMENTO Y SALARIO)

Los datos del empleado están distribuidos en tres estados: Hidalgo, Guerrero y Morelos correspondientes a los nodos HGO, GRO Y MRL. La tabla esta divida por estado, esto es todos los empleados en Hidalgo se encuentran en el fragmento EMPLEADOS_H1, los empleados de Guerrero están almacenados en el fragmento EMPLEADOS_G1 y todos los empleados que trabajan en Morelos se almacenan en el fragmento EMPLEADOS_M1.

Ahora suponga que un usuario desea obtener todos los empleados con fecha de nacimiento anterior al 18 de Noviembre de 1967.

Consideraciones:

a) Los fragmentos de la tabla EMPLEADOS son únicos quiere decir no hay dos registros en los fragmentos que sean únicos.

b) No existe alguna porción de la base de datos replicada en ningún otro nodo o sitio de la red.

Dependiendo del nivel de distribución soportado se pueden obtener tres casos.

CASO A: La BD soporta Transparencia de Fragmentación

La consulta no contiene ningún elemento de distribución en la base de datos. Esto es no especifica nombres de los fragmentos o ubicaciones. Por tanto la consulta seria como sigue:

```
SELECT * FROM EMPLEADOS
WHERE FECHANAC < 18/11/67
```

CASO B: La BD soporta Transparencia de ubicación

En la consulta se debe especificar los nombres de las particiones, pero no se necesita especificar la ubicación de las particiones. Por tanto la consulta seria como sigue:

```
SELECT * FROM EMPLEADOS_H1
WHERE FECHANAC < 18/11/67
UNION
SELECT * FROM EMPLEADOS_G1
WHERE FECHANAC < 18/11/67
UNION
SELECT * FROM EMPLEADOS_M1
WHERE FECHANAC < 18/11/67
```

CASO C: La BD soporta Transparencia de mapeo local

En la consulta se debe especificar los nombres de las particiones y la ubicación de las particiones. Por tanto la consulta seria como sigue:

```
SELECT * FROM EMPLEADOS_H1 NODE HGO
WHERE FECHANAC < 18/11/67
UNION
SELECT * FROM EMPLEADOS_G1 NODE GRO
WHERE FECHANAC < 18/11/67
UNION
SELECT * FROM EMPLEADOS_M1 NODE MRL
WHERE FECHANAC < 18/11/67
```

La transparencia de distribución es soportada a través del diccionario de datos distribuidos (DDD) o un catalogo de datos distribuidos (DDC). EL DDC contiene la descripción de la base de datos completa como es vista por el administrador de la base de datos.
La descripción de la base de datos, conocida como el esquema global distribuido (distributed global schema) es el esquema de la base de datos usado por los procesadores de transacciones locales (TP) para traducir los requerimientos de los usuarios en subconsultas (requerimientos remotos) que serán procesaros por diferentes nodos de red. Entonces el DDC debe mantener consistencia al actualizar todos los sitios. Debemos recordar que algunas implementaciones de DDBMS imponen limitaciones en el soporte de niveles de transparencia. Por tanto se puede distribuir una base de datos, pero no una tabla a lo largo de varios sitios. Tal condición indica transparencia en la ubicación, pero no transparencia en la fragmentación.

3.1 Transparencia de transacción

Permite a una transacción actualizar datos en varios sitios o nodos en la red. La transparencia de transacciones asegura que la transacción será completada satisfactoriamente o bien abortada, manteniendo así la integridad en la base de datos. En otras palabras, la transparencia en la transacción asegura que la transacción será completada solamente si todos los nodos de la base de datos participantes terminaron satisfactoriamente su parte de la transacción.
Transacciones Locales: cuando se accede a los datos del único emplazamiento donde se inició la transacción.
Transacciones Globales: Cuando se accede a datos de emplazamientos distintos al nodo donde se inició la transacción.

3.1.1 Solicitudes/Requerimientos y transacciones distribuidas

El que una transacción sea distribuida o no, se forma por uno o más requerimientos de base de datos. La diferencia entre una transacción no distribuida y una transacción distribuida es que la última puede actualizar o pedir datos de varios sitios remotos en la red.
Una transacción distribuida se divide en varias subtransacciones, uno por cada sitio a accesar, una transacción se puede representar por un agente.
Consideraciones:
Existe transparencia de ubicación

Un **requerimiento remoto** permite accesar datos a ser procesados por un solo procesador de base de datos remoto. En otras palabras, una sentencia SQL puede referenciar datos a un solo sitio remoto.

```
SELECT * FROM CLIENTES
WHERE ESTADO = MORELOS
```

Una **transacción remota** compuesta por varios requerimientos puede accesar datos a un solo sitio.

```
BEGIN TRANSACTION
UPDATE CLIENTES
SET BALANCE = BALANCE +120
WHERE NOCLIENTE = ‘5432’
INSERT INTO ORDENES (NOCLIENTE, FECHA, TOTAL)
VALUES (‘5432’, ’21-06-2009’, 3000.00)
COMMIT TRAN
```

a) La transacción actualiza las tablas CLIENTES y ORDENES
b) Ambas tablas están ubicadas en el sitio B
c) La transacción puede referenciar solamente un DP remoto
d) Cada sentencia SQL o requerimiento puede referenciar solamente un (el mismo) DP remoto a un tiempo, y la transacción completa puede referenciar y es ejecutada en un solo DP remoto.

Una **transacción distribuida** es aquella que se refiere a varios sitios locales o remotos. De tal forma que cada requerimiento puede referenciar a un procesador de datos remotos (DP), la transacción como un todo puede referenciar múltiples DPs porque cada requerimiento puede referenciar a un sitio diferente.

```
BEGIN TRAN
SELECT *
  FROM PRODUCTOS
  WHERE NOPROD = '98765'
UPDATE CLIENTES
  SET BALANCE = BALANCE +120
  WHERE NOCLIENTE = '5432'
INSERT INTO ORDENES (NOCLIENTE, FECHA, TOTAL)
  VALUES ('5432', '21-06-2009', 3000.00)
COMMIT TRAN
```

a) La transacción referencia dos sitios remotos B y C
b) El primer requerimiento (sentencia SELECT) es procesada por el procesador de datos DP en el sitio remoto B, y los siguientes requerimientos (UPDATE e INSERT) son procesados por el DP ubicado en C
c) Cada requerimiento puede accesar solamente un sitio remoto a un tiempo
La tercera característica puede crear problemas. Por ejemplo suponga que la tabla PRODUCTOS está dividida en dos fragmentos o particiones PRODB y PRODC ubicados en los sitios B y C respectivamente. Dado este escenario, la transacción distribuida no puede ser ejecutada porque el requerimiento no puede accesar los datos de más de un sitio remoto. Entonces el DBMS debe poder soportar un requerimiento distribuido.

```
SELECT * FROM PRODUCTOS
WHERE NOPROD = '98765'
```

Un **requerimiento distribuido** permite referenciar datos de varios procesadores de datos DP ubicados en sitios remotos. Dado que cada requerimiento puede accesar datos de uno o más DPs. Una transacción puede accesar varios sitios. La habilidad para ejecutar un requerimiento distribuido proporciona capacidades de procesamiento distribuido de base de datos completo. Porque puede:
a) Particionar una tabla de base de datos en varios fragmentos
b) Referenciar uno o más de estos fragmentos con un solo requerimiento. En otras palabras proporciona transparencia de fragmentación.
La ubicación y partición de los datos debe ser transparente al usuario final.

```
BEGIN TRAN
SELECT  NOCLIENTE, TOTAL
  FROM CLIENTES, ORDENES
  WHERE NOCLIENTE = '9876'
  AND  ORDENES.NOCLIENTE = CLIENTES.NOCLIENTE
UPDATE CLIENTES
  SET BALANCE = BALANCE +120
  WHERE NOCLIENTE = '5432'
INSERT INTO ORDENES (NOCLIENTE, FECHA, TOTAL)
  VALUES ('9876', '21-06-2009', 3000.00)
COMMIT TRAN
```

La transacción usa una sola sentencia SELECT para referenciar dos tablas CLIENTES Y ORDENES. Las dos tablas están ubicadas en dos sitios diferentes MTY y MRL.
El requerimiento distribuido permite a un solo requerimiento referenciar a una tabla particionada físicamente. Por ejemplo, suponga que la tabla CLIENTES está dividida en dos fragmentos CLIENTES_MTY y CLIENTES_MRL ubicados en los sitios MTY y MRL respectivamente.
Supongamos que un usuario final quiere obtener la lista de todos los clientes cuyos balances exceden los 5000. Vea la siguiente figura de requerimiento distribuido donde existe un soporte de transparencia de fragmentación completa, la cual la proporciona el DDBMS que soporta requerimientos distribuidos.

```
SELECT *
FROM CLIENTES
WHERE BALANCE > 3000
```

El entender los diferentes tipos de requerimientos a base de datos en sistemas de base de datos distribuidos ayuda a lidiar mejor con la transparencia de transacción.
La transparencia de transacción asegura que las transacciones distribuidas sean tratadas como transacciones centralizadas, asegurando seriabilidad de transacciones. Es decir, la ejecución de transacciones concurrentes, ya sean o no distribuidas, llevara la base de datos de un estado consistente a otro.
La transparencia transaccional en una base de datos distribuida es complicada debido a que se deben contemplar la fragmentación, la ubicación, y los esquemas de replicación.
Consideremos dos aspectos más de transparencia, Transparencia en la concurrencia y la transparencia en las fallas.

3.1.2 Control de concurrencia distribuido

Transparencia en la concurrencia
En un DBMS centralizado, esto significa que todas las transacciones que se ejecuten concurrentemente deben ejecutarse independientemente y deben estar consistentes con los resultados que se obtuvieran si las transacciones se ejecutaran una a un tiempo, en cualquier orden arbitrario. Sin embargo, el DDBMS debe asegurarse que las transacciones locales y globales no interfieran entre sí. De igual manera, el DDBMS debe asegurar la consistencia de todas las subtransacciones de una transacción global.

El control de concurrencia distribuido es muy importante en los ambientes de base de datos distribuidos porque las operaciones entre múltiples sitios y múltiples procesos tienen a generar inconsistencias en los datos y problemas de abrazos mortales (deadlocks) en las transacciones más que en los ambientes de un solo sistema. Por ejemplo, un procesador de transacciones TP componente de un DDBMS debe asegurarse que todas las partes de la transacción, en todos los sitios, se terminen satisfactoriamente antes que el COMMIT final se ejecute para comprometer la transacción.

Suponga que cada operación de la transacción se compromete por cada procesador de datos DP local, pero uno no puede comprometer los resultados de la transacción. Este escenario generaría un estado inconsistente. La solución a este problema se da con el Protocolo de Compromiso de dos fases.

3.2.3 Protocolo Commit de dos fases

Transparencia a Fallas
En una DBMS centralizado éste debe proporcionar un mecanismo de recuperación que asegure que en caso de fallas, las transacciones deben ser atómicas y durables (ACID properties). Sin embargo, **en un DDBMS, este debe de poder recuperarse en caso de fallas como perdidas de mensajes, fallas en ligas de comunicación, falla de un sitio, particionado de red,** etc. Y asegurar la atomicidad y durabilidad de transacciones globales.
Las bases de datos centralizadas requieren solamente un solo procesador de datos (DP), todas las operaciones de base de datos se realizan en un solo sitio, y los resultados de las operaciones en la base de datos son conocidas de inmediato por el DBMS. En contraste, las base de datos distribuidas hacen posible esto a través de que la transacción accesa a los datos en diversos sitios, un COMMIT final no debe darse hasta que todos los sitios han comprometido sus partes de la transacción.

El protocolo de dos fases garantiza que si una porción de una operación de la transacción no pudo ser comprometida, todos los cambios realizados en los otros sitios participantes en la transacción serán deshechos para mantener la consistencia en el estado de la base de datos.
Cada procesador de datos tiene su propia bitácora de transacciones (transactions log). El protocolo de dos fases requiere registrar una entrada en cada bitácora de transacciones de su procesador de datos (DP) antes de que el fragmente sea realimente actualizado. Por tanto, el protocolo requiere usar el protocolo (hacer-deshacer y rehacer) DO-UNDO-REDO y el protocolo (escritura anticipada) write-ahead.
Protocolo DO-UNDO-REDO

a) Do ejecuta la correspondiente operación y registra sus valores antes y después de la operación (before and after image) en la bitácora de transacciones.
b) UNDO deshace la operación, aplicando los valores anteriores guardados en la bitácora de transacciones por la fase DO.
c) REDO re-ejecuta la operación, aplicando los valores posteriores guardados en la bitácora de transacciones por la fase DO.

Para asegurar que las operaciones (hacer-deshacer y rehacer) puedan recuperar de una falla en el sistema cuando requieran ejecutarse, se usa el protocolo write-ahead. Este protocolo forza que las entradas en el log de transacciones se guarden en disco antes que la operación tome lugar.

El protocolo commit de dos fases define las operaciones entre dos tipos de nodo, el coordinador y uno o más subordinados o conjuntos. El protocolo es implementado en dos fases
Fase de Preparación

1. El coordinador manda un mensaje PREPARE COMMIT a todos los subordinados
2. Los subordinados reciben el mensaje, escriben en su bitácora de transacciones, usando el protocolo write-ahead y mandan un mensaje de confirmación (acknowledge) (YES o PREPARED TO COMMIT) o (NO o NOT PREPARED) al coordinador.
3. El coordinador se asegura que todos los nodos estén listos para comprometer la transacción (commit) o este abortara la transacción.

Si todos los nodos están preparados para comprometer su parte de la transacción se iniciara la fase 2.
Si uno o más nodos no están preparados, el coordinador mandara un mensaje a todos (BROADCAST) los subordinados de ABORT.
Fase El COMMIT FINAL

1. El coordinador manda mensaje a todos (BROADCAST) los subordinados de COMMIT y espera por las respuestas.
2. Cada subordinado recibe el mensaje COMMIT y procede a actualizar la base de datos usando el protocolo DO (hacer).
3. Los subordinados responden con un mensaje COMMITED o un NOT COMMITED al coordinador.

Si uno o más subordinados no comprometen su parte de la transacción, el coordinador manda mensaje de ABORT, forzándolos a UNDO (deshacer) los cambios.
El objetivo del commit de dos fases es asegurarse que todos los nodos comprometieron su parte de la transacción, o la transacción se aborta.

3.3 Transparencia de desempeño y optimización de consultas

Estas características permiten al sistema tener un rendimiento como si fuera un DBMS centralizado. El sistema no sufrirá ninguna degradación en su rendimiento derivado de su uso en una red o derivado de las diferencias en la plataforma de red. Esto aseguro que el sistema encontrara la ruta más efectiva en costos para accesar datos remotos.
Uno de las funciones de base de datos más importantes es la disponibilidad de los datos. S todos los datos residen un solo sitio en una base de datos centralizada, el DBMS debe evaluar todos los requerimientos de datos y encontrar la mejor ruta de acceso a los datos locales. En contraste, **los Sistemas de base de datos distribuidas hacen posible la partición de una base de datos en varios fragmentos, entonces la traducción de la consulta es más complicada porque los DDBMS deben decidir en qué fragmento de la base de datos deben accesar. Además los datos pueden también estar replicados en diferentes sitios. La replicación de datos accesa el problema de acceso a los datos más complejo, porque ahora la base de datos debe decidir cual copia de los datos accesar.** El DDBMS usa técnicas de optimización de consultas para lidiar con tales problemas y asegurar rendimiento en la base de datos aceptable.

El objetivo de la optimización de consultas es minimizar los costos totales asociados con la ejecución de un requerimiento. Los costos asociados con el requerimiento son una función de:

a) El costo del tiempo de acceso (I/O) requerido en accesar los datos físicamente en disco.
b) El costo de uso de red asociado con la transmisión de los datos.
c) El costo tiempo de procesamiento (CPU)

Dado que los costos son clasificados ya sea por comunicación o por procesamiento, no todos los algoritmos de optimización de consultas usan los mismos parámetros para la evaluación de costos.
Para evaluar la optimización de consultas, debemos tener en mente que el procesador de transacciones TP debe recibir los datos del procesador de datos DP, sincronizarlos, generar la respuesta a partir de las consultas en cada sitio y presentarlas al usuario final o a la aplicación. A pesar de que este proceso es estándar, debemos considerar que una consulta en particular puede ser ejecutada en uno de los diferentes sitios. El tiempo de respuesta asociado con los sitios remotos no puede ser fácilmente predeterminado porque algunos nodos pueden terminar su parte de la consulta más rápido que otros.
La optimización de consultas en base de datos distribuidas debe ofrecer transparencia de distribución y transparencia de réplica.
Transparencia de replica se refiere a la habilidad del DDBMS de ocultar al usuario la existencia de múltiples copias de datos.
La mayoría de los algoritmos propuestos por la optimización de consultas están basados en dos principios: a) La selección del orden de ejecución óptimo y b) la selección de los sitios a ser accesados para minimizar los costos de uso de red. Dentro de estos dos principios, el algoritmo de optimización de consultas debe ser evaluado con base de su modo de operación o el tiempo de su optimización.
Los modos de operación pueden ser clasificados como manuales o automáticos.

a) La optimización de consulta automática significa que el DDBMS tiene cuidado de encontrar la ruta de acceso con el costo más efectivo sin intervención del usuario.

b) La optimización de consulta manual requiere que la optimización sea selecciona y calendarizada por el usuario final o el programados.

Otra clasificación de optimización de consulta puede ser de acuerdo a cual optimización se realiza.
Clasificación de optimización dependiendo del momento en que se realiza, el algoritmo de optimización de consulta puede ser estático o dinámico.

c) La optimización de consulta estática tiene lugar al tiempo de compilación. Este tipo de optimización es común cuando las sentencias SQL están inmersas en un lenguaje de programación procedural. Cuando el programa se compila, este crea el plan de acceso necesario para accesar a la base de datos. Cuando el programa es ejecutado, el DBMS usa el plan de acceso para accesar la base de datos.

d) La optimización de consulta dinámica toma lugar al tiempo de ejecución. La estrategia de acceso es definida cuando el programa es ejecutado. Por tanto, la estrategia de acceso es dinámicamente determinada por el DBMS al tiempo de ejecución, usando la información más actualizada acerca de la base de datos. A pesar de que la optimización de consultas dinámica puede ser eficiente, su costo es medido por el tiempo de ejecución (run time processing overead). La mejor estrategia es determinada cada vez que la consulta es ejecutada, esto puede pasar varias veces en el mismo programa.

Otra clasificación de optimización de consultas es dada de acuerdo al tipo de información que es usada para optimizar la consulta. Por ejemplo las consultas pueden estar basadas en algoritmos basados en estadísticas o en reglas.

e) Los algoritmos basados en estadísticas usan información estadística de la base de datos, como el número de registros en las tablas, tiempo de acceso promedio, etc. Las estadísticas pueden ser actualizadas automáticamente o manualmente.

f) El algoritmo basado en reglas está basado en un conjunto de reglas definidas por el usuario o el administrador de la base de datos.

3.4 Forzamiento de la integridad en Bases de Datos Distribuidas

Integridad de los datos en sistemas de bases de datos distribuidas.
En los sistemas de bases de datos distribuidas las transacciones dejan de ser secuencias lineales y ordenadas de las acciones sobre los datos, quiere decir que como los datos están distribuidos, las actividades de la transacción pueden tener lugar en diferentes sitios y puede ser difícil mantener un orden de tiempo entre las acciones.
El problema más común lo tenemos cuando dos (o más) transacciones se ejecutan al mismo tiempo y ambas requieren acceso al mismo registro de datos con vistas a completar su procesamiento. El problema de concurrencia en sistemas distribuidos es mayor que en sistemas centralizados, puesto que puede
haber varias copias de un mismo registro, y todas las copias deben tener los mismos valores o de lo contrario las transacciones trabajarían con datos imprecisos.
Para implementar el control de concurrencia se debe conocer lo siguiente
1. El tipo de algoritmo de planificación utilizado.
2. La localización del planificador.
3. Cómo se controlan los datos duplicados.
Anomalías de procesamiento concurrente.

Cuando se ejecutan transacciones concurrentes en bases de datos distribuidas pueden interferir unas con otras, lo que puede afectar la integridad y consistencia de la base de datos. Los diferentes problemas que se pueden presentar son los siguientes:
1. Problema de actualización perdidas.
2. Violación de integridad.
3. Lectura o recuperación inconsistente.

El Problema de actualización perdidas ocurre cuando un usuario aparentemente completa una actualización con éxito y puede ser sobre escrita por otro usuario, es decir cuando dos o más transacciones acceden concurrentemente sobre un mismo registro de datos, una transacción actualiza un registro de datos y otra transacción escribe este mismo registro. En el siguiente ejemplo se ejecutan 2 transacciones T1 y T2 de forma concurrente, la transacción T1 resta 500 unidades de registro X y la transacción T2 incrementa 200 unidades del registro X, como podemos observar se pierde la actualización de la transacción T2.

Ej. Actualizaciones perdidas.

	Valor del Registro X	
T1	500	T2
		Begin T2
Begin T1		Read X
Read X		Inc(X,200)
Dec(X,500)	700	Write(X)
Write X	0	Commit T2
Commit T1		

Violación de integridad
Se obtiene de violación de integridad en el manejo de la base de datos, cuando se permite la ejecución de dos transacciones concurrentemente sin previo sincronismo.

Para el siguiente ejemplo tenemos una base profesores con los campos: nombre, departamento, asignatura. También otra base con la información de un grupo de una facultad que le llamaremos Grupo_1 que posee los campos: turno, asignatura, nombre_P. Inicialmente estas bases se encuentran en siguiente estado:

Profesores

Nombre	Departamento	Asignatura
Pepe	Computación	DDB
Pepe	Computación	DB
Juan	Computación	DB

Grupo_1

Turno	Asignatura	Nombre_P
1er	DB	Pepe
2da	Análisis	Pedro
3ra	MOC	Tomas

Tenemos una transacción T4 que decide cambiar la asignatura DB por DDB y actualiza el nombre del profesor en la tabla Grupo_1, Por razones de trabajo Pepe no puede impartir la asignatura DB en el 1er turno y la transacción T5 cambia el profesor que imparte esa asignatura por Juan.

T4	T5
update Grupo_1	
set Asignatura = DDB,	update Grupo_1
Nombre_P = Nombre	set Nombre_P = Juan
where Asignatura=DB	where Asignatura=DB
	AND Turno= 1er

Donde finalmente la base de datos Grupo_1 quedaría en el siguiente estado:

Grupo_1

1er	DDB	Juan
2da	Análisis	Pedro
3ra	MOC	Tomas

Que sería una violación de integridad

Problemas de recuperaciones inconsistentes.

La mayoría del trabajo de control de concurrencia se concentra en las transacciones que actualizando la base de datos pueden interferir unas con otras y provocar datos erróneos. Sin embargo, transacciones que solo lean la base de datos pueden obtener resultados inexactos si se permite leer resultados parciales de transacciones incompletas que actualizan la base de datos, lo que algunas veces se hace, obteniéndose lecturas erróneas.
Veamos el siguiente ejemplo:
Tenemos un base de datos S con los salarios de los profesores de la universidad y la referencia al mismo se hace mediante la siguiente forma para Pepe es S_Pepe, para Juan es S_Juan, La transacción T5 actualiza los salarios de los profesores que han cambiado de categoría, en este caso S_Juan, T6 suma el salario de todos los profesores a pagar por la Universidad.

El resultado final es que a Juan no le pagarán el salario correcto.

Ej Problemas de recuperaciones inconsistente.

T5	T6
	Begin T6
	Total = 0
Begin T5 Do	while not end (S)
Read (S, S_Juan)	Read S_Juan
Inc(S_juan, 15$)	Inc(Total, S_Juan)
Write(S, S_Juan)	
Commit T5	
	Read S_Pepe
	Inc(Total, S_Pepe)
	:
	Commit T6

Estos factores proporcionan las bases para la construcción de reglas que determinen cuándo un algoritmo de control de concurrencia es correcto. Las reglas se basan en la teoría de la serialización(Theory of serializability). Una transacción consiste en una secuencia de lecturas y escrituras en la base de datos.

4 Diseño de bases de datos distribuidas

Objetivo: El alumno diseñara, un bosquejo, una Base de Datos Distribuida

En el presente capítulo se mostrará los aspectos importantes referentes al diseño de una base de datos distribuida.

El problema de diseño

El problema de diseño de bases de datos distribuidos se refiere, en general, a hacer decisiones acerca de la ubicación de datos y programas a través de los diferentes sitios de una red de computadoras. Este problema debería estar relacionado al diseño de la misma red de computadoras. Sin embargo, en estas notas únicamente el diseño de la base de datos se toma en cuenta. La decisión de donde colocar a las aplicaciones tiene que ver tanto con el software del SMBDD como con las aplicaciones que se van a ejecutar sobre la base de datos.

El diseño de las bases de datos centralizadas contempla los dos puntos siguientes:

1. Diseño del "esquema conceptual" el cual describe la base de datos integrada (esto es, todos los datos que son utilizados por las aplicaciones que tienen acceso a las bases de datos).
2. Diseño "físico de la base de datos", esto es, mapear el esquema conceptual a las áreas de almacenamiento y determinar los métodos de acceso a las bases de datos.

En el caso de las bases de datos distribuidas se tienen que considerar los dos problemas siguientes:

3. **Diseño de la fragmentación**, este se determina por la forma en que las relaciones globales se subdividen en fragmentos horizontales, verticales o mixtos.
4. **Diseño de la asignación de los fragmentos**, esto se determina en la forma en que los fragmentos se mapean a las imágenes físicas, en esta forma, también se determina la solicitud de fragmentos.

3.1 Transparencia de distribución

Objetivos del Diseño de la Distribución de los Datos.

En el diseño de la distribución de los datos, se deben de tomar en cuenta los siguientes objetivos:

* **Procesamiento local.** La distribución de los datos, para maximizar el procesamiento local corresponde al principio simple de colocar los datos tan cerca como sea posible de las aplicaciones que los utilizan. Se puede realizar el diseño de la distribución de los datos para maximizar el procesamiento local agregando el número de referencias locales y remotas que le corresponden a cada fragmentación candidata y la localización del fragmento, que de esta forma se seleccione la mejor solución de ellas.

* **Distribución de la carga de trabajo**. La distribución de la carga de trabajo sobre los sitios, es una característica importante de los sistemas de cómputo distribuidos. Esta distribución de la carga se realiza para tomar ventaja de las diferentes características (potenciales) o utilizaciones de las computadoras de cada sitio, y maximizar el grado de ejecución de paralelismo de las aplicaciones. Sin embargo, la distribución de la carga de trabajo podría afectar negativamente el procesamiento local deseado.

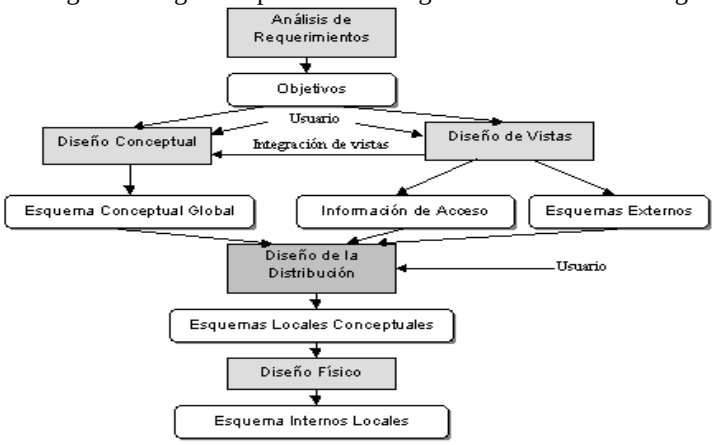
* **Costo de almacenamiento y disponibilidad**. La distribución de la base de datos refleja el costo y disponibilidad del almacenamiento en diferentes sitios. Para esto, es posible tener sitios especializados en la red para el almacenamiento de datos. Sin embargo el costo de almacenamiento de datos no es tan relevante si éste se compara con el del CPU, I/O y costos de transmisión de las aplicaciones.

Enfoques al problema de diseño de bases de datos distribuidas

Existen dos estrategias generales para abordar el problema de diseño de bases de datos distribuidas:

El enfoque de arriba hacia abajo (top-down). Este enfoque es más apropiado par**a aplicaciones nuevas y para sistemas homogéneos**. Consiste en partir desde el análisis de requerimientos para definir el diseño conceptual y las vistas de usuario. A partir de ellas se define un esquema conceptual global y los esquemas externos necesarios. Se prosigue con el diseño de la fragmentación de la base de datos, y de aquí se continúa con la localización de los fragmentos en los sitios, creando las imágenes físicas. Esta

aproximación se completa ejecutando, en cada sitio, "el diseño físico" de los datos, que se localizan en éste. En la siguiente Figura se presenta un diagrama con la estructura general del enfoque top-down.



El diseño de abajo hacia arriba (bottom-up). Se utiliza particularmente a partir de bases de datos existentes, generando con esto bases de datos distribuidas. En forma resumida, el diseño bottom-up de una base de datos distribuida requiere de la selección de un modelo de bases de datos común para describir el esquema global de la base de datos. Esto se debe a que es posible que se utilicen diferentes SMBD. Después se hace la traducción de cada esquema local en el modelo de datos común y finalmente se hace la integración del esquema local en un esquema global común.

El diseño de una base de datos distribuida, cualquiera sea el enfoque que se siga, debe responder satisfactoriamente las siguientes preguntas:

* Por qué hacer una fragmentación de datos?
* Cómo realizar la fragmentación?
* Qué tanto se debe fragmentar?
* Cómo probar la validez de una fragmentación?
* Cómo realizar el asignación de fragmentos?
* Cómo considerar los requerimientos de la información?

El diseño de las base de datos distribuidas introduce dos nuevos problemas:

a) Como Particionar la base de datos en fragmentos que puedan ser replicados en diferentes sitios.
b) Donde ubicar esos fragmentos

La fragmentación de datos y la replicación de datos atacan el primer problema y la ubicación de los datos ataca el segundo problema.

La fragmentación de los datos nos permite romper un objeto en uno o más segmentos o fragmentos. El objeto puede ser almacenado en cualquier sitio en la red de computadoras. La información de la fragmentación de datos es almacenada en el catalogo de datos distribuidos DDC, al cual lo acceso el procesador de transacciones TP para procesar las peticiones de usuarios.

4.1.1 Fragmentación Horizontal

La fragmentación horizontal se realiza sobre las tuplas de la relación. Cada fragmento será un subconjunto de las tuplas de la relación.

Fragmentación horizontal:

- Una tabla T se divide en subconjuntos, T1, T2, ...Tn. Los fragmentos se definen a través de una operación de selección y su reconstrucción se realizará con una operación de unión de los fragmentos componentes.
- Cada fragmento se sitúa en un nodo.
- Pueden existir fragmentos no disjuntos: combinación de fragmentación y replicación.

Algoritmos de particionamiento

Ejemplo práctico: En Adaptive Server Enterprise

- Round Robin
- Semántico
    - Por rangos
    - Por listas de valores
    - Por valores discretos

Ejemplo: Suponga que una compañía de Administración requiere información acerca de sus clientes en los tres estados pero la compañía en cada estado (Hidalgo, Morelos y Guerrero) requiere solamente los datos correspondientes a los clientes locales. Con base en tales requerimientos, se decide distribuir los datos por Estado. Por tanto, se definen particiones horizontales conforme a la estructura que se muestra a continuación.

CLIENTES

NOCLIENTE	NOMBRE_CLI	DIRECCION	ESTADO	LIMITE	BALANCE	PUNTAJE	DUE
10	NISSAN		HGO	3500	2700	3	1245
11	FORD		MRL	6000	1200	1	NULL
12	CHRYSLER		GRO	4000	3500	3	3400
13	GENERAL MOTORS		HGO	6000	5890	3	1090
14	MAZDA		GRO	1200	550	1	NULL
15	TOYOTA		MRL	2000	350	2	50

Condiciones para la fragmentación horizontal:

FRAGMENTO	UBICACIÓN	CONDICION	NODO	CLIENTES	NO. REGS.
PARTHGO	PACHUCA	ESTADO=HGO	NODO1	10,13	500
PARTGRO	ACAPULCO	ESTADO=GRO	NODO2	12,14	700
PARTMRL	CUERNAVACA	ESTADO=MRL	NODO3	11,15	600

Cada fragmento horizontal puede tener diferente número de registros, pero debe tener los mismos campos

Los fragmentos de las tablas creados son las siguientes:

PARTHGO

NOCLIENTE	NOMBRE_CLI	DIRECCION	ESTADO	LIMITE	BALANCE	PUNTAJE	DUE
10	NISSAN		HGO	3500	2700	3	1245
13	GENERAL MOTORS		HGO	6000	5890	3	1090

PARTGRO

NOCLIENTE	NOMBRE_CLI	DIRECCION	ESTADO	LIMITE	BALANCE	PUNTAJE	DUE
12	CHRYSLER		GRO	4000	3500	3	3400
14	MAZDA		GRO	1200	550	1	NULL

PARTMRL

NOCLIENTE	NOMBRE_CLI	DIRECCION	ESTADO	LIMITE	BALANCE	PUNTAJE	DUE
11	FORD		MRL	6000	1200	1	NULL
15	TOYOTA		MRL	2000	350	2	50

Existen dos variantes de la fragmentación horizontal: la primaria y la derivada.

La fragmentación horizontal primaria de una relación se desarrolla empleando los predicados definidos en esa relación.

Consiste del particionamiento en tuplas de una relación global en subconjuntos, donde cada subconjunto puede contener datos que tienen propiedades comunes y se puede definir expresando cada fragmento como una operación de selección sobre la relación global.

Ejemplo Considere la relación global

SUPPLIER(SNUM, NAME, CITY)

entonces, la fragmentación horizontal puede ser definida como:

SUPPLIER1 = SLcity == "SF"SUPPLIER

SUPPLIER2 = SLcity == "LA"SUPPLIER

--Esta fragmentación satisface la condición de completes si "SF" y "LA" son solamente los únicos valores posibles del atributo CITY.

--La condición de reconstrucción se logra con: SUPPLIER = SUPPLIER1 union SUPPLIER2

--La condición de disjuntos se cumple claramente en este ejemplo.

La fragmentación horizontal derivada consiste en dividir una relación partiendo de los predicados definidos sobre alguna otra. En la fragmentación horizontal derivada es importante recordar dos cosas:

1. La conexión entre las relaciones de propietario y miembro están definidas como un equi-join.
2. Un equi-join puede ser implementado por su significado de semijoins.

La fragmentación derivada horizontal se define partiendo de una fragmentación horizontal.
En esta operación se requiere de Semi-junta (Semi-Join) el cual nos sirve para derivar las tuplas o registros de dos relaciones.
Ejemplo Las siguientes relaciones definen una fragmentación horizontal derivada de la relación SUPPLY.
SUPPLY1 = SUPPLYSJsnum == snumSUPPLIER1
SUPPLY2 = SUPPLYSJsnum == snumSUPPLIER2

Revisión de correcciones.
Ahora debemos revisar los algoritmos de fragmentación respecto a los 3 criterios de corrección descritos antes:
Completitud: La completitud de una fragmentación horizontal primaria está basada en los criterios de selección usados ya que la base del algoritmo de fragmentación es un conjunto de predicados completos y mínimos. La completitud de una fragmentación horizontal derivada es un poco más difícil de definir.
Reconstrucción: la reconstrucción de una relación global a partir de sus fragmentos es llevada a cabo por el operador Unión tanto en la fragmentación horizontal primaria como en la derivada.
Disyunción: es fácil establecer la disyunción de una fragmentación primaria que la de una fragmentación derivada. En una fragmentación primaria la disyunción está garantizada mientras que los predicados de minterms de la fragmentación sean mutuamente excluyentes.
En la fragmentación derivada hay un semi-join involucrado que agrega una complejidad considerable. La disyunción puede ser garantizada si el join es simple de no ser así será necesario investigar los valores actuales de las tuplas.

4.1.2 Fragmentación Vertical

La fragmentación vertical produce fragmentos los cuales contienen un subconjunto de los atributos así como la clave principal de la relación. El objetivo de este tipo de fragmentaciones es particionar una relación en un conjunto de relaciones más pequeñas tal que otras de las aplicaciones de los usuarios se ejecutarán en solo un fragmento.

- Una tabla T se divide en subconjuntos, T1, T2, ...Tn. Los fragmentos se definen a través de una operación de proyección.
- Cada fragmento debe incluir la clave primaria de la tabla. Su reconstrucción se realizará con una operación de join de los fragmentos componentes.
- Cada fragmento se sitúa en un nodo.
- Pueden existir fragmentos no disjuntos: combinación de fragmentación y replicación

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

Ejemplo Considere la siguiente relación global:

EMP(empnum, name, sal, tax, mgrnum, depnum)

Una fragmentación vertical de esta relación puede ser definida como:

EMP1 = PJempnum, name, mgrnum, depnum EMP
EMP2 = PJempnum, sal, tax EMP

La reconstrucción de la relación EMP puede ser obtenida como:

EMP = EMP1 (JN empnum) EMP2 porque empnum es una clave de EMP

Ejemplo Considere la tabla CLIENTES que se muestra a continuación

NOCLIENTE	NOMBRE_CLI	DIRECCION	ESTADO	LIMITE	BALANCE	PUNTAJE	DUE
10	NISSAN		HGO	3500	2700	3	1245
11	FORD		MRL	6000	1200	1	NULL
12	CHRYSLER		GRO	4000	3500	3	3400
13	GENERAL MOTORS		HGO	6000	5890	3	1090
14	MAZDA		GRO	1200	550	1	NULL
15	TOYOTA		MRL	2000	350	2	50

Suponga que la Compañía está dividida en dos departamentos, el departamento de Servicios y el departamento de Distribución. Cada departamento está ubicado en edificios separados, y cada departamento tiene intereses distintos con respecto a los Clientes.
Cada fragmento vertical tiene el mismo número de registros, pero contiene diferentes atributos

FRAGMENTO	UBICACIÓN	NODO	ATRIBUTOS
PARTUNO	SERVICIOS	SRVC	NOCLIENTE, NOMBRE_CLI,DIRECCION,ESTADO
PARTDOS	DISTRIBUCION	DIST	NOCLIENTE,LIMITE,BALANCE,RATING,DUE

Los fragmentos se muestran a continuación.
Fragmento PARTUNO

NOCLIENTE	NOMBRE_CLI	DIRECCION	ESTADO
10	NISSAN		HGO
11	FORD		MRL
12	CHRYSLER		GRO
13	GENERAL MOTORS		HGO
14	MAZDA		GRO
15	TOYOTA		MRL

Fragmento PARTDOS

NOCLIENTE	LIMITE	BALANCE	PUNTAJE	DUE
10	3500	2700	3	1245
11	6000	1200	1	NULL
12	4000	3500	3	3400
13	6000	5890	3	1090
14	1200	550	1	NULL
15	2000	350	2	50

Nótese que la llave de la tabla original CLIENTES se mantiene común en ambos fragmentos.

Requerimientos de información de una fragmentación vertical.
La información principal requerida por una fragmentación vertical es la relacionada a las aplicaciones. Ya que el particionamiento vertical coloca en un fragmento aquellos atributos los cuales son accesados juntos. Los principales datos requeridos relacionados a las aplicaciones son sus frecuencias de acceso.

Algoritmo de agrupamiento
La tarea fundamental en el diseño de un algoritmo de fragmentación vertical es encontrar algunas formas de agrupamiento de los atributos de una relación basada en la afinidad de los atributos. El algoritmo de agrupamiento es considerado apropiado por las siguientes razones.
1. Es diseñado específicamente para determinar grupos de elementos similares, en comparación con ordenarlos linealmente.
2. Los agrupamientos finales son intensivos en el orden en el cual los elementos son presentados al algoritmo, 3. El tiempo de computación del algoritmo es razonable ***n2*** donde **n** es el número de atributos.
4. Las relaciones secundarias entre los grupos de atributos agrupados son identificables.

Revisando las correcciones
Completitud: la completitud es garantizada por el algoritmo de partición ya que cada atributo de la relación global es asignado a uno de los fragmentos.

Reconstrucción*:* hemos mencionado ya que la reconstrucción puede darse mediante la aplicación de una operación de unión. Por lo tanto mientras que cada ***Ri*** sea completo, la operación de unión reconstruirá apropiadamente ***R.***
Disyunción*:* como indicamos antes la disyunción de fragmentos no es importante para la fragmentación vertical como lo es en la horizontal.

4.1.3 Fragmentación Mixta

En la mayoría de los casos una simple fragmentación horizontal o vertical de un esquema de base de datos no será suficiente para satisfacer los requerimientos de una aplicación de usuario. En este caso una fragmentación vertical debería ir seguida por una horizontal, o viceversa, produciendo una partición con estructura de árbol.
La fragmentación mixta requiere particionamiento de las tablas tanto horizontal como vertical. Suponga que una Compañía opera en tres diferentes lugares. En cada uno de estos lugares los departamentos de servicio y distribución operan en diferentes edificios . Cada uno de los departamentos requiere accesar información acerca de sus clientes locales. La estructura de la Compañía requiere entonces que los datos sean fragmentados horizontalmente para contener la información de los diferentes lugares y dentro de cada lugar, los datos deben estar fragmentados verticalmente para contener los diferentes departamentos. En este caso, se requiere fragmentación mixta.

La fragmentación mixta es un procedimiento de dos pasos:

a) Se realiza fragmentación horizontal por cada lugar, con base en la ubicación dentro de cada estado. La fragmentación horizontal genera los subconjuntos de registros de clientes (fragmentos horizontales) que serán ubicados en cada sitio.
b) Dado que los departamentos están localizados en diferentes edificios. Se requiere fragmentación vertical dentro de cada fragmento horizontal para dividir los atributos, proporcionando así la información que se requiere en cada subsitio.

Ejemplo

Las condiciones de fragmentación se muestran a continuación

FRAGMENTO	UBICACIÓN	CRITERIO HORIZONTAL	NODO	REGISTROS EN SITIO	CRITERIO VERTICAL
HGOUNO	HIDALGO-SERVICIOS	ESTADO= HIDALGO	UNOSRVC	10,13	NOCLIENTE, NOMBRE_CLI,DIRECCION,ESTADO
HGODOS	HIDALGO-DISTRIBUCION	ESTADO= HIDALGO	UNODIST	10,13	NOCLIENTE,LIMITE,BALANCE,RATING,DUE
GROUNO	GUERRERO-SERVICIOS	ESTADO= GUERRERO	DOSSRVC	12,14	NOCLIENTE, NOMBRE_CLI,DIRECCION,ESTADO
GRODOS	GUERRERO-DISTRIBUCION	ESTADO= GUERRERO	DOSDIST	12,14	NOCLIENTE,LIMITE,BALANCE,RATING,DUE
MRLUNO	MORELOS-SERVICIOS	ESTADO= MORELOS	TRESSRVC	11,15	NOCLIENTE, NOMBRE_CLI,DIRECCION,ESTADO
MRLDOS	MORELOS-DISTRIBUCION	ESTADO= MORELOS	TRESDIST	11,15	NOCLIENTE,LIMITE,BALANCE,RATING,DUE

Los Fragmentos mixtos quedarían como se muestra a continuación:

Fragmento HGOUNO, ubicación Hidalgo, Departamento Servicios

NOCLIENTE	NOMBRE_CLI	DIRECCION	ESTADO
10	NISSAN		HGO
13	GENERAL MOTORS		HGO

Fragmento HGODOS, ubicación Hidalgo, Departamento Distribución

NOCLIENTE	LIMITE	BALANCE	PUNTAJE	DUE
10	3500	2700	3	1245
13	6000	5890	3	1090

Fragmento GROUNO, ubicación Guerrero, Departamento Servicios

NOCLIENTE	NOMBRE_CLI	DIRECCION	ESTADO	LIMITE	BALANCE	PUNTAJE	DUE
12	CHRYSLER		GRO	4000	3500	3	3400
14	MAZDA		GRO	1200	550	1	NULL

Fragmento GRODOS ubicación Guerrero, Departamento Distribución

NOCLIENTE	LIMITE	BALANCE	PUNTAJE	DUE
12	4000	3500	3	3400
14	1200	550	1	NULL

Fragmento MRLUNO, ubicación Morelos, Departamento Servicios

NOCLIENTE	NOMBRE_CLI	DIRECCION	ESTADO
11	FORD		MRL
15	TOYOTA		MRL

Fragmento MRLDOS ubicación Morelos, Departamento Distribución

NOCLIENTE	LIMITE	BALANCE	PUNTAJE	DUE
11	6000	1200	1	NULL
15	2000	350	2	50

Ejemplo: Considere la relación global

EMP(empnum, name, sal, tax, mrgnum, depnum)

Las siguientes son para obtener una fragmentación mixta, aplicando la vertical seguida de la horizontal:

EMP1 = SL depnum <= 10 PJempnum, name, mgrnum, depnum EMP
EMP2 = SL 10 < depnum <= 20 PJempnum, name, mgrnum, depnum EMP
EMP3 = SL depnum > 20 PJempnum, name, mgrnum, depnum EMP
EMP4 = PJ empnum, name, sal, tax EMP

La reconstrucción de la relación EMP es definida por la siguiente expresión:

EMP = UN(EMP1, EMP2, EMP3)JNempnum = empnumPJempnum, sal, tax EMP4

Finalmente, como otro ejemplo considere el siguiente esquema global

EMP(EMPNUM, NAME, SAL, TAX, MGRNUM, DEPNUM)
DEP(DEPNUM, NAME, AREA, MGRNUM)
SUPPLIER(SNUM, NAME, CITY)
SUPPLY(SNUM, PNUM, DEPNUM, QUAN)

Después de aplicar una fragmentación mixta se obtiene el siguiente esquema fragmentado

EMP1 = Sldepnum <= 10 PJempnum, name, mgrnum, depnum (EMP)
EMP2 = SL 10 < depnum <= 20 PJempnum, name, mgrnum, depnum (EMP)
EMP3 = SL depnum > 20 PJempnum, name, mgrnum, depnum (EMP)

EMP4 = PJ empnum, name, sal, tax (EMP)
DEP1 = SL depnum <= 10 (DEP)
DEP2 = SL 10 < depnum <= 20 (DEP)
DEP3 = SL depnum > 20 (DEP)

SUPPLIER1 = SL city == "SF" (SUPPLIER)
SUPPLIER2 = SL city == "LA" (SUPPLIER)
SUPPLY1 = SUPPLYSJsnum == snumSUPPLIER1
SUPPLY2 = SUPPLYSJsnum == snumSUPPLIER2

4.1.4 Localización de los fragmentos

Por razones de rendimiento, se pueden realizar copias de los fragmentos.

4.2 Replicación de Datos

Réplica: El sistema conserva varias copias o réplicas idénticas de una tabla. Cada réplica se almacena en un nodo diferente.

La replicación de datos se refiere al almacenamiento de varias copias de datos en múltiples sitios conectados por una red de computadoras. Las copias de los fragmentos pueden estar almacenadas en diferentes sitios dependiendo de ciertos requerimientos de información específicos. Dado que existen

copias de los fragmentos la disponibilidad de los datos y el tiempo de respuesta pueden ser mejores, reduciendo tiempo de red y costos de consultas.
Suponga una base de datos X dividida en dos fragmentos X1 y X2. Dentro de un escenario de base de datos distribuida replicada, es posible que el fragmento X1 este almacenado en los sitios A1 y A2, mientras que el fragmento X2 este almacenado en los sitios A2 y Z3.
Los datos replicados están sujetos a una regla de mutua consistencia. Esta regla requiere que todos los fragmentos de datos deben ser idénticos. Así se mantiene la consistencia de los datos entre las replicas. Un DDBMS debe asegurar que una actualización a la base de datos se realice en todos los sitios donde las replicas existen.
A pesar de que la replicación tiene sus beneficios, también impone carga de procesamiento al DDBMS, dado que al mantenimiento de cada copia. Por ejemplo, considere los procesos que un DDBMS debe realizar para usar la base de datos:

- Si la base de datos está fragmentada, el DDBMS debe descomponer la consulta en subconsultas para accesar los correspondientes fragmentos.
- Si la base de datos esta replicada, el DDBMS debe decidir cual copia accesar. Una operación de lectura selecciona la copia más cercada para satisfacer la transacción. Una operación de escritura requiere que todas las copias deban ser seleccionadas y actualizadas para satisfacer la regla de mutua consistencia.
- El procesador de transacciones TP manda un requerimiento de datos a cada procesador de datos DP para su ejecución.
- El procesador de datos DP recibe y ejecuta cada requerimiento y manda los datos de regreso al TP.
- El TP ensambla las respuestas del DP.

El problema se complica si se toma en consideración factores adicionales como la topología de red, los throughputs de comunicación, etc.
Existen tres condiciones de replicación, completamente replicada, parcialmente replicada o no replicada.

a) Una base de datos completamente replicada: La cual almacena múltiples copias de cada fragmento de base de datos en múltiples sitios. En este caso todos los fragmentos de base de datos están replicados. Las base de datos completamente replicadas tienen a ser poco practicas por la cantidad de carga (overead) que representa para el sistema.
b) Una base de datos parcialmente replicada: La cual almacena múltiples copias de algunos fragmentos de base de datos en múltiples sitios. La mayoría de los DDBMSs son capaces de administrar y manipular la base de datos parcialmente replicada bien.
c) Una base de datos que no es replicada almacena cada fragmento de base de datos en un solo sitio. Por tanto, no existen fragmentos duplicados.

Existen dos factores que permiten tomar decisión sobre utilizar datos replicados o no.

a) El tamaño de la base de datos
b) Frecuencia de utilización

Cuando la frecuencia de utilización de datos ubicados remotamente es alta y la base de datos es grande, la replicación de datos puede ayudar a reducir los costos de acceso a datos. La información de los datos replicados se encuentra en el catalogo de los datos distribuidos DDC, cuyo contenido es usado por el procesador de transacciones TP para decidir cual copia de fragmento de base de datos accesar.

“Una tabla puede fragmentarse horizontalmente (por registros), horizontalmente (por atributos) o mixto (registros y atributos) y cada fragmento puede ser replicado”

Otros tipos de replicación de datos
La réplica de las claves de la relación en los fragmentos es una característica de la fragmentación vertical que permite la reconstrucción de la relación global. Por tanto, la escisión considera únicamente aquellos atributos que no son parte de la clave primaria.

La réplica de los atributos clave supone una gran ventaja, a pesar de los problemas que pueda causar. La ventaja está relacionada con el esfuerzo para mantener la integridad semántica. Tenga en cuenta que cada dependencia (funcional, multivaluada ...) es, de hecho, una restricción que influye sobre el valor de los atributos de las respectivas relaciones en todo momento. También muchas de estas dependencias implican a los atributos clave de una relación. Si queremos diseñar una base de datos tal que los atributos clave sean parte de una fragmento que está ubicado en un sitio, y los atributos relacionados sean parte de otro fragmento asignado a un segundo sitio, cada petición de actualización provocará la verificación de integridad que necesitará de una comunicación entre esos sitios. La réplica de los atributos clave de cada fragmento reduce esta problemática, pero no elimina toda su complejidad, ya que la comunicación puede

ser necesaria para las restricciones de integridad que implican a las claves primarias, así como para el control de concurrencia.

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

Ventaja de la Replicación:
Disponibilidad: el sistema sigue funcionando aún en caso de caída de uno de los nodos.
Aumento del paralelismo: Varios nodos pueden realizar consultas en paralelo sobre la misma tabla. Cuantas más réplicas existan de la tabla, mayor será la posibilidad de que el dato buscado se encuentre en el nodo desde el que se realiza la consulta, minimizando con ello el tráfico de datos entre nodos.

Desventaja de la Replicación:
Aumento de la sobrecarga en las actualizaciones: El sistema debe asegurar que todas las réplicas de la tabla sean consistentes. Cuando se realiza una actualización sobre una de las réplicas, los cambios deben propagarse a todas las réplicas de dicha tabla a lo largo del sistema distribuido.

4.2 Colocación de Datos

Para la ubicación de los datos se deben observar tres estrategias de alojamiento:

a) **Centralizado:** Toda la base de datos esta almacenada en un solo sitio.
b) **Particionada:** La base de datos está dividida en varios fragmentos y almacenados en diversos sitios.
c) **Replicada:** Copias de uno o más fragmentos de base de datos están almacenados en diversos sitios.

La distribución de los datos a lo largo de una red de computadoras se logra a través de la partición de los datos, a través de la replicación de los datos o bien a través de una combinación de ambos. El alojamiento de los datos está muy relacionado con la forma en que la base de datos está dividida o fragmentada. La mayoría de los estudios de ubicación o colocación de los datos se enfoca en DONDE poner QUE.

Los algoritmos de colocación de datos toman en consideración una variedad de factores como:

a) Requerimientos de rendimiento y disponibilidad de datos
b) Tamaño, numero de registros y el número de relaciones que una entidad mantiene con otras entidades.
c) Tipos de transacciones a ser aplicados a la base de datos, los atributos accesados por cada una de estas transacciones, etc.

Algunos algoritmos incluyen datos externos, como la topología de la red, el throughput de la red. No existe un algoritmo universalmente aceptado aun y pocos algoritmos han sido implementados hasta la fecha.

TAREA: OBTENER LAS 12 REGLAS DE CHRIS DATE PARA BASE DE DATOS DISTRIBUIDAS

Las 12 reglas de Chris Date para Base de datos Distribuidas

Las reglas de Date describen una base de datos completamente, y por tanto, no todos los Manejadores de bases de Datos Distribuidas DDBMS están conforme a ellas. Por tanto estas reglas son un buen criterio para seleccionar un buen Manejador de Base de Datos Distribuidas.

1. Regla de independencia del Sitio Local o Autonomía Local

Cada sitio local puede actuar como un Sistema Manejador de Base de Datos independiente, autónomo, centralizado. Cada sitio es responsable por su seguridad, control de concurrencia, respaldo y recuperación.

2. Independencia de Sitio Central

No debe existir un sitio en la red que dependa de algún sitio central o en cualquier otro sitio. Todos los sitios tienen las mismas capacidades.

3. Independencia a fallos
4. El sistema no se ve afectada por fallas en nodos.

El sistema está en operación continua aun en el caso de falla de algún nodo o expansión de la red.

5. Transparencia de ubicación

El usuario no necesita saber la ubicación de los datos para poder consultarlos.

6. Transparencia de replicación

Los usuarios ben solamente una sola base de datos lógica. El DDBMS transparentemente selecciona que fragmento e datos accesa. El DDBMS administra y maneja todos los fragmentos de forma transparente al usuario.

7. Procesamiento de consultas distribuido

Una consulta distribuida debe ser ejecutada por procesadores de datos DP en diferentes sitios. La optimización de consultas se realiza transparentemente por el DDBMS.

8. Procesamiento de transacciones distribuido

Una transacción puede actualizar datos a lo largo de diversos sitios. La transacción se ejecuta de manera transparente en por diferentes procesadores de datos diversos sitios.

9. Independencia de Hardware

El sistema debe corre en cualquier plataforma de hardware.

10. Independencia de Sistema Operativo

El sistema debe correr en cualquier plataforma de sistema operativo

11. Independencia de Red

El sistema debe correr en cualquier plataforma de red.

12. Independencia de Base de datos

El sistema debe soportar cualquier proveedor de Manejadores de Base de datos

5 Traslación de consultas globales a consultas fragmentadas PROCESAMIENTO DE CONSULTAS

Objetivo: El alumno explicará cómo se realizan de consultas en una base de datos distribuida

Arquitectura del procesamiento de consultas

5.1 Equivalencia de transformación de consultas

Cando una base de datos se encuentra en multiples servidores y distribuye a un numero determinado de nodos tenemos:

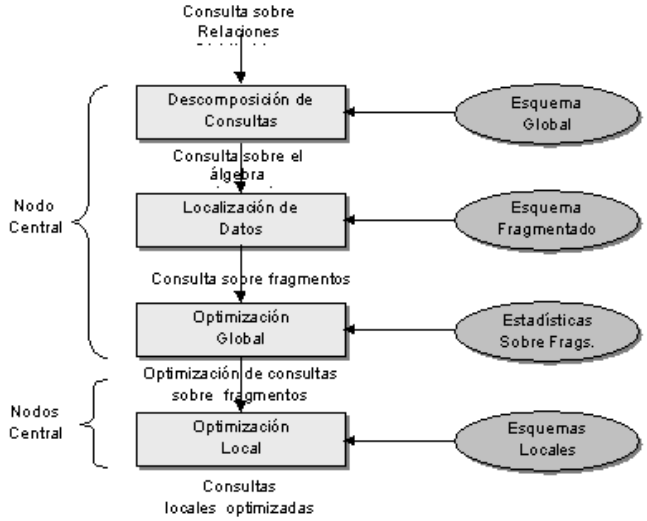
1.-el servidor recibe una peticion de un nodo
2.-el servidor es atacado por el acceso concurrente a la base de datos cargada localmente
3.-el servidor muestra un resultado y le da un hilo a cada una de las maquinas nodo de la red local.

Cuando una base de datos es accesada de esta manera la técnica que se utiliza es la de fragmentación de datos que puede ser hibrida, horizontal y vertical.

En esta fragmentación lo que no se quiere es perder la consistencia de los datos, por lo tanto se respetan las formas normales de la base de datos.

Para realizar una transformación en la consulta primero desfragmentamos siguiendo los estandares marcados por las reglas formales y posteriormente realizamos el envio y la maquina que recibe es la que muestra el resultado pertinente para el usuario, de esta se puede producir una copia que sera la equivalente a la original.

El problema de procesamiento de consultas se puede descomponer en varios subproblems que corresponden a diferentes niveles. En la Figura 4.4, se presenta un esquema por niveles genérico para el procesamiento de consultas. Para simplificar la discusión, suponga que se tiene un procesador de consultas estático semicentralizado en donde no se tienen fragmentos replicados. Cuatro capas principales están involucradas en mapear una consulta a una base de datos distribuida en una secuencia optimizada de operaciones locales, cada una de ellas actuando en una base de datos local. **Las cuatro capas principales son: descomposición de consultas, localización de datos, optimización global de consultas y optimización local de consultas.**
Las primeras tres se realizan en un nodo central usando información global. La cuarta capa se realiza en cada nodo local.



5.2 Transformación de consultas globales a consultas fragmentadas y 5.3 Consultas parametrizadas

*****Descomposición de consultas**

La primera capa descompone una consulta en el cálculo relacional en una consulta en el álgebra relacional que opera sobre relaciones globales. Consiste de cuatro partes:

1. **Normalización.** Involucra la manipulación de los cuantificadores de la consulta y de los calificadores de la misma mediante la aplicación de la prioridad de los operadores lógicos.

2. **Análisis.** Se detecta y rechazan consultas semánticamente incorrectas.
3. **Simplificación.** Elimina predicados redundantes.
4. **Reestructuración**. Mediante reglas de transformación una consulta en el cálculo relacional se transforma a una en el álgebra relacional. Se sabe que puede existir más de una transformación. Por tanto, el enfoque seguido usualmente es empezar con una consulta algebraica y aplicar transformaciones para mejorarla.

1. **Normalización**. Involucra la manipulación de los cuantificadores de la consulta y de los calificadores de la misma mediante la aplicación de la prioridad de los operadores lógicos.

El objetivo de la normalización es transformar una consulta a una forma normalizada para facilitar su procesamiento posterior. Existe un procedimiento para obtener la forma normal, conjuntiva o disyuntiva, de cualquier expresión en el cálculo de proposiciones. Para la aplicación que estamos considerando, la forma normal conjuntiva es más práctica puesto que incluye más operadores AND (Ù) que operadores OR (Ú). Típicamente, los operadores OR se transforman en uniones de conjuntos y los operadores AND se transforman en operadores de junta o selección.

Considere la siguiente consulta:
"Encuentre los nombres de los empleados que han trabajado en el proyecto J1 12 o 24 meses"

La consulta expresada en SQL es:

```
SELECT ENAME
FROM E, G
WHERE E.ENO = G.ENO AND G.JNO = "J1" AND DUR = 12 OR DUR = 24
```

La consulta en forma normal conjuntiva es: AND (Ù)
E.ENO = G.ENO Ù G.JNO = "J1" Ù (DUR = 12 Ú DUR = 24)

La consulta en forma normal disyuntiva es: OR (Ú).
(E.ENO = G.ENO Ù G.JNO = "J1" Ù DUR = 12) Ú
(E.ENO = G.ENO Ù G.JNO = "J1" Ù DUR = 24)

En esta última forma, si se tratan las conjunciones en forma independiente se puede incurrir en trabajo redundante si no se eliminan las expresiones comunes.

2. **Análisis.** Se detecta y rechazan consultas semánticamente incorrectas.

El análisis de consultas permite rechazar consultas normalizadas para los cuales no se requiere mayor procesamiento. Una consulta se puede rechazar si alguno de sus atributos o nombres de relación no están definidas en el esquema global. También se puede rechazar si las operaciones que se aplican a los atributos no son del tipo adecuado. Considere la siguiente consulta:

"Encuentre los nombres y responsabilidades de los programadores que han estado trabajando en el proyecto de CAD/CAM por más de tres años y el nombre de su administrador"

La consulta expresada en SQL es:

```
SELECT ENAME, RESP
FROM E, G, J
WHERE E.ENO = G.ENO AND G.JNO = J.JNO AND JNAME = "CAD/CAM"
AND DUR ³ 36 AND TITLE = "Programador"
```

La gráfica de la consulta anterior se presenta en la Figura 4.5a. En la Figura 4.5b se presenta la gráfica de juntas para la gráfica de la Figura 4.5a.

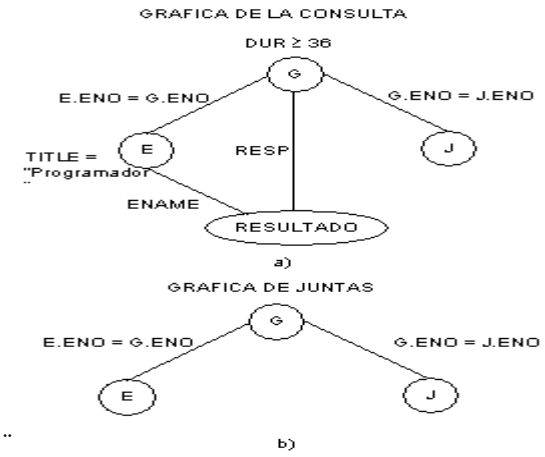


Figura 4.5. a) Gráfica de una consulta. b) Gráfica de juntas.

La gráfica de la consulta es útil para determinar la correctitud semántica de una consulta conjuntiva con múltiples variables sin negaciones. Tal consulta es semánticamente incorrecta si su gráfica no es conectada. En este caso, una o más subgráficas están desconectadas de la gráfica que contiene la relación RESULTADO.

Ejemplo Considere la siguiente consulta:

SELECT ENAME, RESP
FROM E, G, J
WHERE E.ENO = G.ENO **AND** JNAME = "CAD/CAM"
AND DUR ³ 36 **AND** TITLE = "Programador"

La gráfica de la consulta anterior se presenta en la Figura 4.6, la cual se ve claramente que es no conectada.

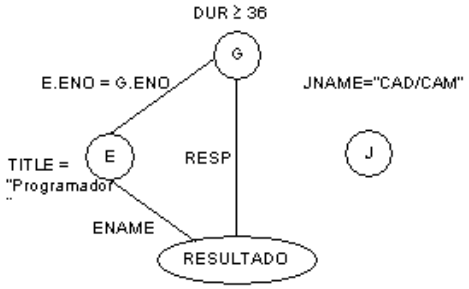


Figura 4.6. Gráfica de una consulta semánticamente incorrecta.

3. **Simplificación.** Elimina predicados redundantes. ***

La consulta en forma normal conjuntiva puede contener predicados redundantes. Una evaluación directa de la consulta con redundancia puede llevarnos a realizar trabajo duplicado. La redundancia puede ser eliminada aplicando sucesivamente las siguientes reglas de idempotencia:

1. $p \wedge p \Leftrightarrow p$
2. $p \vee p \Leftrightarrow p$
3. $p \wedge true \Leftrightarrow p$
4. $p \vee false \Leftrightarrow p$
5. $p \wedge false \Leftrightarrow false$
6. $p \vee true \Leftrightarrow true$
7. $p \wedge \neg p \Leftrightarrow false$
8. $p \vee \neg p \Leftrightarrow true$
9. $p_1 \wedge (p_1 \vee p_2) \Leftrightarrow p_1$
10. $p_1 \vee (p_1 \wedge p_2) \Leftrightarrow p_1$

Ejemplo La siguiente consulta en SQL

```
SELECT TITULO
FROM E
WHERE (NOT (TITULO = "Programador"))
AND (TITULO = "Programador"
OR TITULO = "Ingeniero Eléctrico")
AND NOT (TITULO = "Ingeniero Eléctrico")
OR ENOMBRE = "J. Doe"
```

Puede ser simplificada usando las reglas anteriores a

```
SELECT TITULO
FROM E
WHERE ENOMBRE = "J. Doe"
```

4. **Reestructuración**. Mediante reglas de transformación una consulta en el cálculo relacional se transforma a una en el álgebra relacional. Se sabe que puede existir más de una transformación. Por tanto, el enfoque seguido usualmente es empezar con una consulta algebraica y aplicar transformaciones para mejorarla.

El último paso en la descomposición de consultas reescribe la consulta en el álgebra relacional. Esto se hace típicamente en los siguientes pasos:

1. Una transformación directa del cálculo relacional en el álgebra relacional
2. Una reestructuración de la consulta en el álgebra relacional para mejorar la eficiencia

Por claridad es costumbre representar la consulta en el álgebra relacional por un árbol del álgebra relacional, el cual es un árbol en donde una hoja representa a una relación almacenada en la base de datos y un nodo no hoja es una relación intermedia producida por una operación del álgebra relacional.

La transformación de una consulta en el cálculo relacional en un árbol del álgebra relacional se puede hacer como sigue. Primero, se crea una hoja diferente para cada variable de tuplo diferente. En SQL, las hojas están disponibles de forma inmediata en la cláusula FROM. Segundo, el nodo raíz se crea como una operación de proyección involucrando a los atributos resultantes. Estos se encuentran en la cláusula SELECT de una consulta en SQL. Tercero, la calificación (WHERE) de una consulta se traduce a una secuencia apropiada de operaciones relacionales (select, join, union, etc.) yendo de las hojas a la raíz. La secuencia se puede dar directamente por el orden de aparición de los predicados y operadores.

Ejemplo La consulta

"Encuentre los nombres de empleados diferentes de "J. Doe" que trabajaron en el proyecto de CAD/CAM por uno o dos años" Se puede expresar en SQL como sigue:

```
SELECT ENAME
FROM J, E, G
WHERE E.ENO = G.ENO
AND G.JNO = J.JNO
AND ENAME ≠ "J. Doe"
AND JNAME = "CAD/CAM"
AND (DUR = 12 OR DUR = 24)
```

Se puede mapear de manera directa al árbol del álgebra relacional de la Figura 4.7.

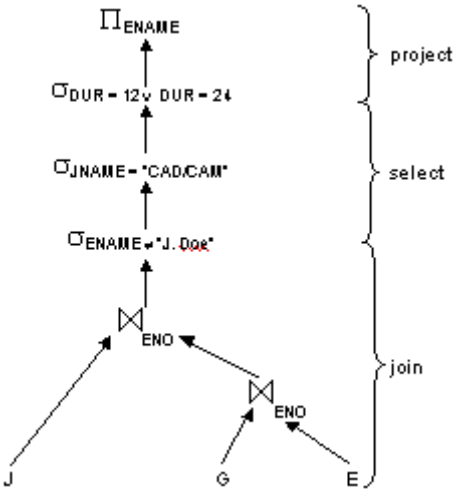


Figura 4.7 Ejemplo de un árbol del álgebra relacional.

Aplicando las reglas de transformación que se verán a continuación, muchos árboles diferentes se pueden encontrar por el procedimiento descrito antes. Las seis reglas de equivalencia más útiles, las cuales están relacionadas a las operaciones del álgebra relacional se presentan a continuación. En lo que sigue, R, S y T son relaciones donde R se define sobre los atributos $A = \{ A_1, A_2, ..., A_n \}$ y S se define sobre los atributos $B = \{ B_1, B_2, ..., B_n \}$.

1. Conmutatividad de operaciones binarias:
   $R \times S = S \times R$
   $R \rhd\lhd S = S \rhd\lhd R$
   $R \cup S = S \cup R$
2. Asociatividad de operaciones binarias:
   $R \times (S \times T) \Leftrightarrow (R \times S) \times T$
   $R \rhd \lhd (S \rhd \lhd T) \Leftrightarrow (R \rhd \lhd S) \rhd \lhd T$
3. Idempotencia de operaciones unarias:
   $\Pi_A (\Pi_A (R)) \Leftrightarrow \Pi_A (R)$
   $\sigma_{p1(A1)} (\sigma_{p2(A2)} (R)) \Leftrightarrow \sigma_{p1(A1) \wedge p2(A2)} (R)$
   donde $R[A]$ y $A \subseteq A$, $A \subseteq A$y $A \subseteq A$.
4. Conmutando selección con proyección
   $\Pi_{A1, ., An} (\sigma_{p(Ap)} (R)) \Leftrightarrow \Pi_{A1, ., An} (\sigma_{p(Ap)} (\Pi_{A1, ., An, Ap} (R)))$

5. Conmutando selección con operaciones binarias
   $\sigma_{p(A)} (R \times S) \Leftrightarrow (\sigma_{p(A)} (R)) \times S$
   $\sigma_{p(Ai)} (R \rhd\lhd_{(Aj, Bk)} S) \Leftrightarrow (\sigma_{p(Ai)} R) \rhd\lhd_{(Aj, Bk)} S$
   $\sigma_{p(Ai)} (R \cup T) \Leftrightarrow \sigma_{p(Ai)} (R) \cup \sigma_{p(Ai)} (T)$
   donde A_i pertenece a R y a T.
6. Conmutando proyección con operaciones binarias
   $\Pi_C (R \times S) \Leftrightarrow \Pi_A (R) \times \Pi_B (S)$
   $\Pi_C (R \rhd \lhd_{(Aj, Bk)} S) \Leftrightarrow \Pi_A (R) \rhd \lhd_{(Aj, Bk)} \Pi_B (S)$
   $\Pi_C (R \cup S) \Leftrightarrow \Pi_C (R) \cup \Pi_C (T)$
   donde $R[A]$ y $S[B]$; $C = A \cup B$ donde $A \subseteq A$ y $B \subseteq B$.

Ejemplo En la Figura 4.8 se presenta un árbol equivalente al mostrado en la Figura 4.7.

La reestructuración del árbol de la Figura 4.7 se presenta en la Figura 4.9. El resultado es bueno en el sentido de que se evita el acceso repetido a la misma relación y las operaciones más selectivas se hacen primero. Sin embargo, este árbol aún no es óptimo. Por ejemplo, la operación de selección E no es muy útil antes de la junta dado que no reduce grandemente el tamaño de la relación intermedia.

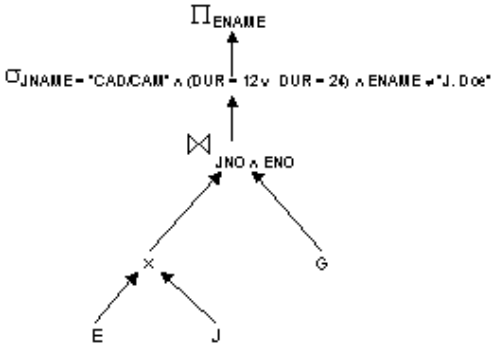


Figura 4.8. Arbol equivalente al de la Figura 4.7.

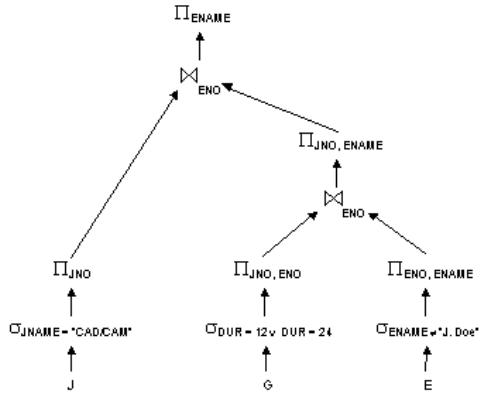


Figura 4.9. Arbol reestructura a partir del de la Figura 4.7.

Localización de Datos

La entrada a esta capa es una consulta algebraica definida sobre relaciones distribuidas. El objetivo de esta capa es localizar los datos de la consulta usando la información sobre la distribución de datos. Esta capa determina cuales fragmentos están involucrados en la consulta y transforma la consulta distribuida en una consulta sobre fragmentos.

Este es el papel de la capa de localización, la cual traduce una consulta hecha sobre relaciones globales a una consulta algebraica expresada en fragmentos físicos. La localización utiliza información almacenada en el esquema de fragmentación. Por simplicidad en esta sección no se considera el caso de fragmentos replicados.

La fragmentación de una relación se define a través de las reglas de fragmentación, las cuales pueden ser expresadas como consultas relacionales. Una relación global puede ser reconstruida aplicando las reglas de reconstrucción y derivando un programa en el álgebra relacional cuyos operandos son los fragmentos. A este programa se le conoce como *programa de localización*. Una forma simple de localizar una consulta distribuida es generar una consulta donde cada relación global es sustituida por su programa de localización. Esto puede ser visto como el reemplazo de las hojas del árbol del álgebra relacional de la consulta distribuida con subárboles que corresponden a los programas de localización. A la consulta obtenida por esta forma se le conoce como una *consulta genérica*.

En general, el enfoque anterior puede ser ineficiente dado que varias simplificaciones y reestructuraciones de la consulta genérica aún pueden ser realizadas. En lo que sigue de esta sección, por cada tipo de fragmentación se presentan técnicas de reducción que general consultas simples y optimizadas.

Reducción para fragmentación horizontal primaria

La fragmentación horizontal distribuye una relación basada en predicados de selección. El ejemplo siguiente será usado a lo largo de esta sección.

Ejemplo . La relación E(ENO, ENOMBRE, TITULO) puede ser dividida en tres fragmentos horizontales E_1, E_2 y E_3, definidos como sigue:

$$E_1 = s_{ENO £ "E3"} (E)$$
$$E_2 = s_{"E3" < ENO £ "E6"} (E)$$
$$E_3 = s_{ENO > "E6"} (E)$$

El programa de localización para fragmentación horizontal es la unión de los fragmentos. Aquí se tiene que:

$$E = E_1 È E_2 È E_{13}$$

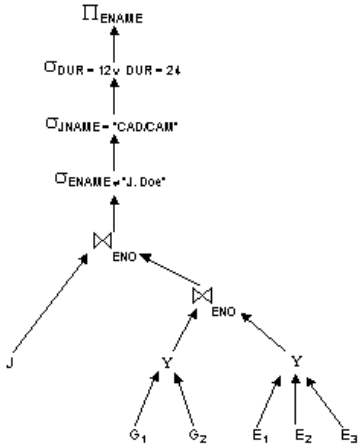
La relación G puede ser dividida en dos fragmentos horizontales G_1 y G_2 definidos como sigue:

$$G_1 = s_{ENO £ "E3"} (G)$$
$$G_2 = s_{ENO > "E3"} (G)$$

El programa de localización para G es la unión de los fragmentos. Aquí se tiene que:

$$G = G_1 È G_2$$

El árbol genérico se presenta a continuación:



Reducción con selección

Dada una relación *R* que ha sido fragmentada horizontalmente como R_1, R_2, ..., R_w, donde $R_j = s_{pj}(R)$, la regla puede ser formulada como sigue

Regla 1: $s_{pj}(R_j) = f$ si " *x* en *R*: Ø ($p_i(x)$ Ù $p_j(x)$)

donde $p_i(x)$ y $p_j(x)$ son predicados de selección, *x* denota a un tuplo, y $p(x)$ denota que el predicado *p* se satisface para *x*.

Ejemplo Considere la siguiente consulta

```
SELECT *
FROM E
WHERE ENO = "E5"
```

Aplicando el enfoque directo para localizar E a partir de E_1, E_2 y E_3, se obtiene la consulta genérica de la Figura a. Conmutando la selección con la operación de unión, es fácil detectar que el predicado de selección contradice los predicados de E_1 y E_3, produciendo relaciones vacías. La consulta reducida es simplemente aplicada a E_2 como se muestra en la Figura b.

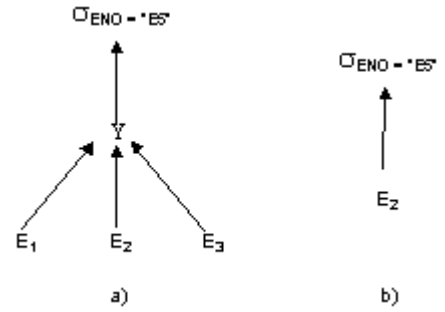


Figura Reducción para fragmentación horizontal con selección.

Reducción con junta

Juntas en relaciones fragmentadas horizontalmente pueden ser simplificadas cuando las relaciones juntadas están fragmentadas de acuerdo al atributo de la junta. La simplificación consiste en distribuir las juntas sobre las uniones y eliminar juntas inútiles. La distribución de una junta sobre una unión puede ser establecida como

$$(R_1 \text{ È } R_2) > < R_3 = (R_{1> <} R_3) \text{ È } (R_2 > < R_3)$$

donde R_i son fragmentos. Con esta transformación las uniones pueden ser movidas hacia arriba en el árbol de consulta de manera que todas las posibles juntas de fragmentos son exhibidas. Juntas inútiles de fragmentos pueden ser determinadas cuando los predicados de fragmentos juntados son contradictorios. Suponga que los fragmentos R_i y R_j están definidos de acuerdo a los predicados p_i y p_j, respectivamente, sobre el mismo atributo, la regla de simplificación puede formularse como sigue:

Regla 2: $R_{1> <} R_3$ = f si " x en R_i, " y en R_j: Ø ($p_i(x)$ Ù $p_j(y)$)

Ejemplo Considere la fragmentación de la relación G del Ejemplo 4.11 junto con la fragmentación de E. E_1 y G_1 están definidos de acuerdo al mismo predicado. Más aún, el predicado que define a G_2 es la unión de los predicados que definen E_2 y E_3. Ahora considere la consulta con junta

```
SELECT *
FROM E, G
WHERE ENO = G.ENO
```

La consulta genérica equivalente se presenta en la Figura 4.12a. La consulta reducida se puede observar en la Figura 4.12b.

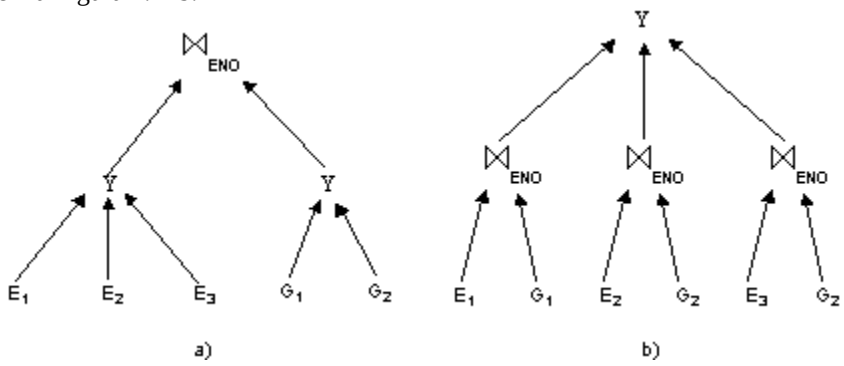


Figura Reducción para fragmentación horizontal con junta.

4.7.2 Reducción para fragmentación vertical

La fragmentación vertical distribuye una relación de acuerdo a los atributos de proyección. Dado que el operador de reconstrucción para la fragmentación vertical es la junta, el programa de localización para una relación fragmentada verticalmente consiste de la junta de los fragmentos sobre el atributo común.

Ejemplo Considere la relación E dividida en dos fragmentos verticales donde el atributo ENO sirve como atributo común.

$E_1 = P_{ENO,ENOMBRE}(E)$
$E_2 = P_{ENO,TITULO}(E)$

El programa de localización es

$E = E_1 > < E_2$

La reducción de consultas sobre fragmentos verticales se hace determinando relaciones intermedias inútiles y eliminando los subárboles que las producen. Las proyecciones sobre fragmentos verticales que no tienen atributos en común con los atributos de proyección (excepto la llave de la relación) producen relaciones inútiles aunque probablemente no vacías.

Dada una relación *R* definida sobre los atributos $A = \{ A_1, A_2, ..., A_n \}$, la cual se fragmenta verticalmente como $R_i = P_A(R)$, donde A Í A, la regla para determinar relaciones intermedias inútiles se puede formular como sigue:

Regla 3: $P_{D,K}(R)$ es inútil si el conjunto de atributos de proyección *D* no está en *A*.

Ejemplo 4.13. Considere de nuevo la relación E dividida en fragmentos verticales como en el Ejemplo 4.12. Considere también la siguiente consulta en SQL:

SELECT ENAME
FROM E

$E_1 = P_{ENO,ENOMBRE}(E)$

$E_2 = P_{ENO,TITULO}(E)$

La consulta genérica equivalente se presenta en la Figura a. Conmutando la proyección con la junta, se puede ver que la proyección sobre E_2 es inútil dado que ENOMBRE no está en E_2. Por lo tanto, la proyección necesita aplicarse únicamente a E_1, como se presenta en la Figura b.

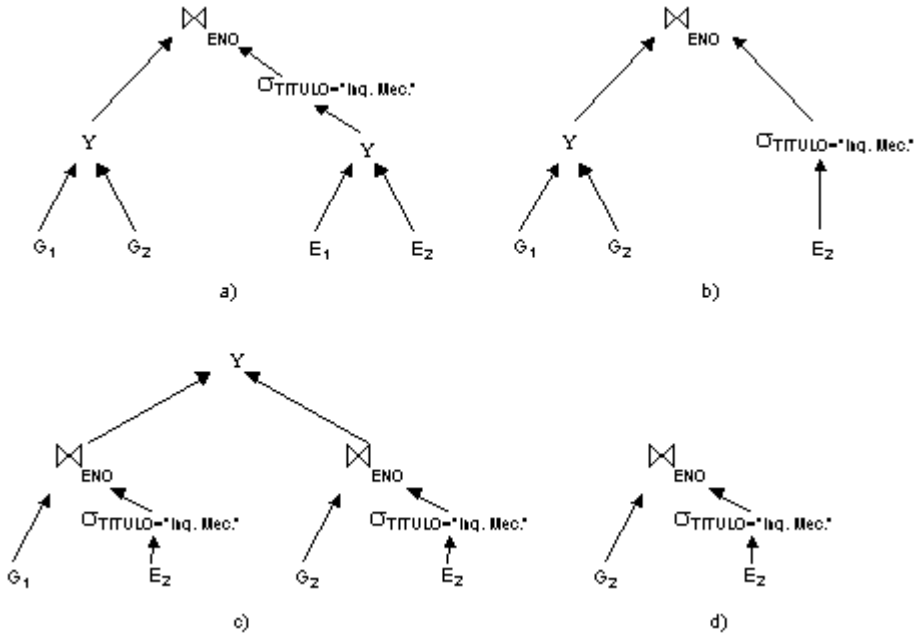


Figura Reducción para fragmentación vertical.

Reducción para fragmentación horizontal derivada

Si una relación *R* es sometida a una fragmentación horizontal derivada con respecto a *S*, los fragmentos de *R* y *S* que tienen el mismo valor del atributo para la junta se localizan en el mismo nodo. Así, *S* puede ser fragmentado de acuerdo al predicado de selección. Dado que los tuplos de *R* se colocan de acuerdo a los tuplos de *S*, la fragmentación horizontal derivada debe ser usada solo para relaciones uno-a-muchos de la forma *S R*, donde un tuplo de *S* se asocia con *n* tuplos de *R*, pero un tuplo de *R* se asocia exactamente con uno de *S*.

Ejemplo Dado una relación uno-a-muchos de E a G, la relación G[ENO, JNO, RESPONSABLE, DUR] se puede fragmentar indirectamente de acuerdo a las siguientes reglas:

$G_1 = G >< _{ENO} E_1$
$G_2 = G >< _{ENO} E_2$

La relación E es fragmentada horizontalmente de la siguiente manera:

$E_1 = s_{TITULO="Programador"} E$
$E_2 = s_{TITULO^1 "Programador"} E$

El programa de localización para G es
$G = G_1 È G_2$

Las consultas en fragmentos derivados también se pueden reducir. Dado que este tipo de fragmentación es útil para optimizar consultas con juntas, una transformación útil es distribuir las juntas sobre las uniones (usadas en los programas de localización) y aplicar la regla 2 presentada antes. Ya que las reglas de fragmentación indican cómo se asocian las tuplas, ciertas juntas producirán relaciones vacías si los predicados de la fragmentación son inconsistentes.

Ejemplo Considere la siguiente consulta en SQL sobre la fragmentación definida en el Ejemplo 4.14:

SELECT *
FROM E, G
WHERE G.ENO = E.ENO AND TITLE = "Ingeniero Mecánico"

La consulta genérica de se presenta en la Figura a. Aplicando primero la selección sobre los fragmentos E_1 y E_2, el predicado de la selección es inconsistente con el de E_1, por lo tanto, E1 puede ser eliminado y la consulta se reduce a la mostrada en la Figura b. Ahora se distribuyen las juntas sobre las uniones produciendo el árbol de la Figura c. El subárbol de la izquierda junta los fragmentos G_1 y E_2, sin embargo, sus predicados son inconsistentes (TITULO = "Programador" de G_1 es inconsistente con TITULO ¹ "PROGRAMADOR" en E_2). Así, el subárbol de la izquierda produce una relación vacía por lo que puede ser eliminado obteniendo la consulta reducida de la Figura d.

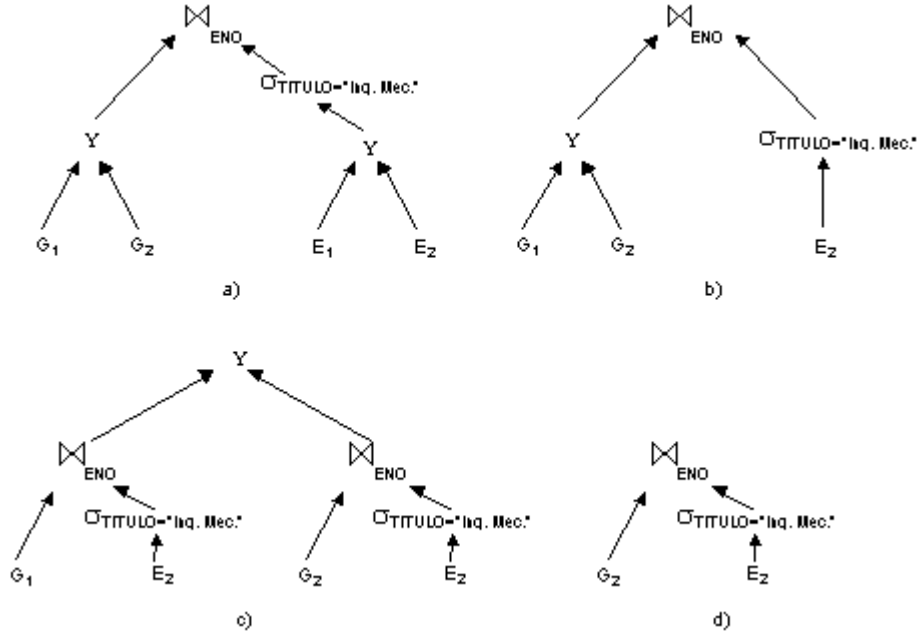


Figura 4.14. Reducción para fragmentación horizontal derivada.

Reducción para fragmentación híbrida

La fragmentación híbrida se obtiene combinando fragmentación horizontal y vertical. El objetivo de la fragmentación híbrida es ejecutar de manera eficiente consultas que involucran selección, proyección y junta. El programa para fragmentación híbrida utiliza uniones y juntas de fragmentos. Las consultas en fragmentos híbridos se pueden reducir combinando las reglas usadas para fragmentación horizontal primaria, fragmentación vertical y fragmentación horizontal derivada. Estas se pueden resumir de la manera siguiente:

1. Remover las relaciones vacías generadas para por selecciones contradictorias en fragmentos horizontales.
2. Remover las relaciones intermedias inútiles generadas por proyecciones en fragmentos verticales.
3. Distribuir juntas sobre uniones a fin de aislar y remover juntas inútiles.

Ejemplo Considere la siguiente fragmentación híbrida de la relación E:

$E_1 = s_{ENO £ "E4"} (P_{ENO,ENOMBRE} (E))$

$E_1 = s_{ENO > "E4"} (P_{ENO,ENOMBRE} (E))$

$E_3 = P_{ENO,TITULO} (E)$

El programa de localización de para la relación E es

$E = (E_1 È E_2) >< _{ENO} E_3$

Considere ahora la siguiente consulta en SQL

```
SELECT ENAME
FROM E
WHERE E.ENO = "E5"
```

La consulta genérica se presenta en la Figura a. Si se mueve hacia abajo la operación de selección se puede eliminar E_1. Más aún, si, una vez eliminando E_1, se mueve hacia abajo la operación de proyección, entonces, se puede eliminar E_3. Así, la junta ya no es necesaria y se obtiene la consulta reducida de la Figura b.

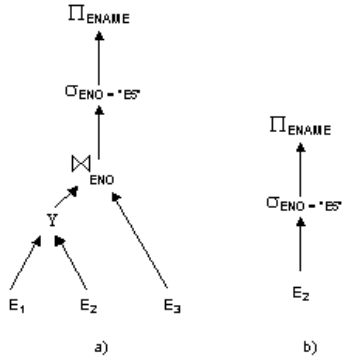


Figura Reducción para fragmentación horizontal derivada

5.4 Optimización Global de Consultas

Dada una consulta algebraica sobre fragmentos, el objetivo de esta capa es hallar una estrategia de ejecución para la consulta cercana a la óptima. La estrategia de ejecución para una consulta distribuida puede ser descrita con los operadores del álgebra relacional y con primitivas de comunicación para transferir datos entre nodos. Para encontrar una buena transformación se consideran las características de los fragmentos, tales como, sus cardinalidades. Un aspecto importante de la optimización de consultas es el ordenamiento de juntas, dado que algunas permutaciones de juntas dentro de la consulta pueden conducir a un mejoramiento de varios órdenes de magnitud. La salida de la capa de optimización global es una consulta algebraica optimizada con operación de comunicación incluidas sobre los fragmentos.

La optimización global de consultas es la tercera etapa del procesamiento de consultas distribuidas de acuerdo a la Figura 4.4. Dada una consulta algebraica sobre fragmentos, el objetivo de esta capa es hallar una estrategia de ejecución para la consulta cercana a la óptima. La salida de la capa de optimización global es una calendarización de una consulta optimizada en el álgebra relacional la cual indica el orden en que se ejecutan los operadores e incluye operaciones de comunicación sobre los fragmentos.

Ya que la selección del ordenamiento óptimo de operadores para una consulta es computacionalmente intratable, el objetivo del optimizador es encontrar la mejor estrategia, no necesariamente óptima, para ejecutar la consulta. La selección de la "mejor" estrategia requiere, por lo general, predecir los costos de ejecución de diferentes alternativas para ejecutar la consulta. El costo de ejecución se expresa como la combinación pesada de los costos de entrada/salida, de CPU y de comunicación.

Definiciones básicas

Antes de empezar los conceptos involucrados en la optimización de consultas, se revisarán brevemente algunos conceptos básicos en el proceso de optimización. En un proceso de optimización es necesario optimizar (minimizar o maximizar) una función objetivo, en nuestro caso llamada *función de costo*. El proceso de optimización selecciona el mejor miembro *x* de un conjunto que optimiza la función de costo. A este conjunto de posibles soluciones se le conoce como el conjunto solución. Dada la forma en que trabajan los algoritmos de optimización, al conjunto solución también se le conoce como el *espacio de búsqueda*. Los algoritmos, llamados de búsqueda, se mueven de elemento a elemento en este espacio hasta encontrar una buena solución. Existen diversas técnicas para el desarrollo de algoritmos de búsqueda: búsqueda ávida, programación dinámica, algoritmos heurísticos, mejoramiento iterativo, recocido simulado, algoritmos genéticos, búsqueda tabú, etc.

Modelo de costo

El costo de una estrategia de ejecución distribuida se puede expresar con respecto al costo total (tiempo de ejecución) o al tiempo de respuesta. El costo total es la suma de todas sus componentes (I/O, CPU y comunicación). El tiempo de respuesta es el tiempo transcurrido desde el inicio de la consulta hasta su terminación. Ambas estrategias de optimización son diferentes. Mientras que en la función de costo total se trata de reducir el costo de cada componente haciendo que éstas sean tan rápidas como sea posible, en la función del tiempo de respuesta se trata de hacer tantas cosas en forma simultánea (paralela) como sea posible lo que puede conducir a un incremento en el tiempo total de ejecución de la consulta.

Costo Total

El costo total es la suma de todos los factores de costo y puede expresarse como

Costo total = *costo de* I/O + *costo de* CPU + *costo de comunicación*

en donde,

costo de CPU = *costo de una instrucción * no. de instrucciones*
costo de I/O = *costo unitario de una operación de* I/O *a disco * no. de accesos*
costo de comunicación = (*tiempo de iniciación + tiempo de transmisión*) * *no. de mensajes*

Los diferentes factores pueden tener pesos diferentes dependiendo del ambiente distribuido en el que se trabaje. Por ejemplo, en las redes de área amplia (WAN), normalmente el costo de comunicación domina dado que hay una velocidad de comunicación relativamente baja, los canales están saturados y el trabajo adicional requerido por los protocolos de comunicación es considerable. Los costos de iniciación y transmisión de mensajes en una WAN son relativamente altos relativos a los tiempos de procesamiento local. Por otra parte, un rango común entre el costo de comunicación y el costo de I/O es 20:1. Así, los algoritmos diseñados para trabajar en una WAN, por lo general, ignoran los costos de CPU y de I/O. En redes de área local (LAN), por otro lado, el costo de comunicación no es tan dominante, así que se consideran los tres factores con pesos variables.

Tiempo de Respuesta

El tiempo de respuesta es el tiempo transcurrido desde el inicio de la consulta hasta su terminación y se puede expresar como

Costo total = *costo de* I/O + *costo de* CPU + *costo de comunicación*

en donde,

costo de CPU = *costo de una instrucción * no. de instrucciones secuenciales*
costo de I/O = *costo unitario de una operación de* I/O *a disco *no. de accesos secuenciales*
costo de comunicación = (*tiempo de iniciación + tiempo de transmisión*) * *no. de mensajes secuenciales*

Ejemplo Considere el sistema distribuido mostrado en la Figura 4.16, en la cual se procesa la respuesta a la consulta en el nodo 3 con datos que provienen de los nodos 1 y 2. Por simplicidad se considera únicamente el costo de comunicación. Para este caso,

costo total = 2 * tiempo de iniciación + tiempo unitario de transmisión * ($x + y$)

Por otro lado, el tiempo de respuesta es

tiempo de respuesta = max{ tiempo para enviar de 1 a 3,
tiempo para enviar de 2 a 3 }

donde

tiempo para enviar de 1 a 3 = tiempo de iniciación + tiempo unitario de transmisión * x
tiempo para enviar de 2 a 3 = tiempo de iniciación + tiempo unitario de transmisión * y

Estadísticas de la base de datos

El factor principal que afecta la eficiencia de la ejecución de una estrategia es el tamaño de las relaciones intermedias que son producidas durante la ejecución. Cuando una operación subsecuente se localiza en un nodo diferente, la relación intermedia debe ser transmitida sobre la red. Por lo tanto, es de un interés especial el poder estimar el tamaño de las relaciones intermedias para poder minimizar el tamaño de las transferencias de datos. Esta estimación se basa en información estadística acerca de las relaciones básicas para predecir las cardinalidades de los resultados de las operaciones relacionales. Para un relación R definida sobre los atributos $A = \{ A_1, A_2, ..., A_n \}$ y fragmentada como $R_1, R_2, ..., R_r$, los datos estadísticos son típicamente:

1. La longitud de cada atributo: *length*(A_i)
2. El número de valores distintos para cada atributo en cada fragmento: card(Π_{Ai}, R_j)
3. Los valores máximo y mínimo en el dominio de cada atributo: *min*(A_i), max(*Ai*)
4. Las cardinalidades de cada dominio: *card*(*dom*[A_i])
5. Las cardinalidades de cada fragmento: *card*(*dom*[R_j])

En algunas ocasiones es útil aplicar el *factor de selectividad* al realizar una junta. Este se define como sigue:

$$SF_{><}(R,S) = \frac{card(R >< S)}{card(R) * card(S)}$$

El tamaño de una relación intermedia R es

size(R) = *card*(R) * *length*(R)

donde *length*(R) es la longitud en bytes de un tuplo de R calculada a partir de sus atributos. La estimación de *card*(R), el número de tuplos en R, requiere de las fórmulas de la siguiente sección.

Cardinalidades de las Relaciones Intermedias

Las cardinalidad de las operaciones son las siguientes:

Selección: *card*($\Pi_F(R)$) = $SF_S(F)$ * *card*(R)

donde $SF_S(F)$ es el factor de selección de la fórmula F calculada como sigue, donde $p(A_i)$ y $p(A_j)$ indican predicados sobre los atributos A_i y A_j, respectivamente.

$$SF_S(A = valor) = \frac{1}{card(\Pi_A(R))}$$

$$SF_S(A > valor) = \frac{max(A) - valor}{max(A) - min(A)}$$

$$SF_S(A < valor) = \frac{valor - min(A)}{max(A) - min(A)}$$

$$SF_S(p(A_i) \wedge p(A_j)) = SF_S(p(A_i)) * SF_S(p(A_j)$$

$$SF_S(p(A_i) \vee p(A_j)) = SF_S(p(A_i)) + SF_S(p(A_j) - (SF_S(p(A_i)) * SF_S(p(A_j))$$

$$SF_S(A \in \{valores\}) = SF_S(A = valor) * card(\{valores\})$$

Proyección: *card*($\Pi_A(R)$) = *card*(R)

Producto Cartesiano: *card*($R \times S$) = *card*(R) * *card*(S)

Unión:

cota superior: *card*($R \cup S$) = *card*(R) + *card*(S)

cota inferior: *card*($R \cup S$) = max{*card*(R), *card*(S)}

Diferencia de conjuntos:

cota superior: *card*($R - S$) = *card*(R)

cota inferior: *card*($R - S$) = 0

Junta:

caso general: *card*($R >< S$) = $SF_{><}$ * *card*(R) * *card*(S)

caso especial: A es una llave de R y B es una llave externa de S;

A es una llave externa de R y B es una llave de S

card($R >< _{=B} S$) = *card*(R)

Semijunta: *card*($R >< _A S$) = $SF_{><}$ ($S.A$) * *card*(R)

donde $SF_{><}$ ($R >< _A S$) = $SF_{><}$ ($S.A$) = $\frac{card(\Pi_A(S))}{card(dom[A])}$

Optimización centralizada de consultas

En esta sección se revisa el proceso de optimización de consultas para ambientes centralizados. Esta presentación es un prerequisito para entender la optimización de consultas distribuidas por tres razones. Primero, una consulta distribuida se transforma a varias consultas locales cada una de las cuales se procesa en forma centralizadas. Segundo, las técnicas para optimización distribuida son, por lo general, extensiones de técnicas para optimización centralizada. Finalmente, el problema de optimización centralizada es mucho más simple que el problema distribuido.

Las técnicas de optimización más populares se deben a dos sistemas de bases de datos relacionales: INGRES y System R. Ambos sistemas tienen versiones para sistemas distribuidos pero sus técnicas difieren sustancialmente. Por un lado, INGRES utiliza una técnica dinámica la cual es ejecutada por un intérprete. Por otra parte, System R utiliza un algoritmo estático basado en búsqueda exhaustiva. Los sistemas más comerciales utilizan variantes de la búsqueda exhaustiva por su eficiencia y compatibilidad con la compilación de consultas.

Por brevedad, en esta sección se revisará brevemente el esquema de optimización de INGRES. Posteriormente, se revisará con mayor detenimiento el esquema de System R.

Algoritmo de INGRES

INGRES usa un algoritmo de optimización dinámico que recursivamente divide una consulta en el cálculo relacional en piezas más pequeñas. Combina dos fases de la descomposición cálculo-álgebra relacional y optimización. No requiere información estadística de la base de datos. Las dos fases generales del algoritmo son:

1. Descompone cada consulta con múltiples variables en una secuencia de consultas con una sola variable común.
2. Procesa cada consulta mono-variable mediante un procesador de consultas de una variable el cual elige, de acuerdo a algunas heurísticas, un plan de ejecución inicial. Posteriormente, ordena las operaciones de la consulta considerando los tamaños de las relaciones intermedias.

La primera parte se hace en tres pasos como se indica a continuación:

- Reemplaza una consulta *q* con *n* variables en una seria de consultas

  $$q_1 , q_2, \dots , q_n$$

  donde q_i usa el resultado de q_{i-1}.
- La consulta *q* se descompone en *q1 q2*, donde *q1* y *q2* tienen una variable en común la cual es el resultado de *q*
- Se reemplaza el valor de cada tupla con los valores actuales y se simplifica la consulta

  $$q(V_1, V_2, ..., V_n) \ (q(t_1, V_2, ..., V_n), t_1 \ R)$$

Ejemplo Para ilustrar el paso de desacoplamiento del algoritmo de INGRES considere la siguiente consulta:

"Encuentre los nombres de los empleados que trabajan en el proyecto CAD/CAM"

Esta consulta se puede expresar en SQL de la siguiente forma:

q_1**: SELECT** E.ENOMBRE
FROM E, G, J
WHERE E.ENO = G.ENO **AND** G.JNO = J.JNO **AND** JNAME = "CAD/CAM"

q_1 es reemplazada por q_{11} seguida de *q* en donde JVAR es una relación intermedia.

q_{11}**: SELECT** J.JNO **INTO** JVAR
FROM J
WHERE JNAME = "CAD/CAM"
q11: **SELECT** E.ENOMBRE
FROM E, G, JVAR
WHERE E.ENO = G.ENO **AND** G.JNO = JVAR.JNO

La aplicación sucesiva de este paso a *q* puede generar

q_{12}**: SELECT** G.ENO **INTO** GVAR
FROM G, JVAR
WHERE G.JNO = JVAR.JNO
q_{13}**: SELECT** E.ENOMBRE
FROM G, GVAR
WHERE E.ENO = GVAR.ENO

Así, q_1 ha sido reducido a la secuencia de consultas q_{11} , q_{12} , q_{13}. La consulta q_{11} es monovariable. Sin embargo, las consultas q_{12} y q_{13} no son monovariables. Veamos como transformar q_{13}. La relación definida por la variable GVAR es sobre un solo atributo (ENO). Supongamos que contiene únicamente dos tuplos: <E1> y <E2>. La sustitución de GVAR genera dos consultas de una sola variable:

q_{131}: **SELECT** E.ENOMBRE
FROM E
WHERE E.ENO = "E1"
q_{132}: **SELECT** E.ENOMBRE
FROM E
WHERE E.ENO = "E2"

Algoritmo de System R

System R ejecuta una optimización de consultas estática basada en búsqueda exhaustiva del espacio solución. La entrada del optimizador es un árbol del álgebra relacional que resulta de una consulta en SQL. La salida es un plan de ejecución que implementa el árbol del álgebra relacional "óptimo".

El optimizador asigna un costo a cada árbol candidato y obtiene aquel con costo menor. Los árboles candidatos se obtienen permutando el orden de las juntas de las *n* relaciones de una consulta usando las reglas de conmutatividad y asociatividad. Para limitar el costo de optimización, el número de alternativas se reduce utilizando programación dinámica. El conjunto de estrategias alternativas se construye de forma dinámica, de manera que, cuando dos juntas son equivalentes por conmutatividad, se queda solo con la de costo más bajo. Más aún, cada vez que aparecen productos Cartesianos se trata de eliminar la estrategia correspondiente.

El costo de una estrategia candidato es una combinación pesada de costos de I/O y CPU. La estimación de tales costos (en tiempo de compilación) se basa en un modelo de costo que proporciona una fórmula de costo para cada operación de bajo nivel. Para la mayoría de las operaciones, esas fórmulas de costo se basan en las cardinalidades de los operandos. La información de las cardinalidades se obtiene del mismo System R. La cardinalidad de las relaciones intermedias se estima de acuerdo a los factores de selectividad de las operaciones.

El algoritmo de optimización consiste de dos grandes pasos:

1. Las consultas simples se ejecutan de acuerdo al mejor camino de acceso (basada en un predicado de selección).
2. Para cada relación *R*, se estima el mejor ordenamiento de juntas, en donde *R* se accesa primero usando el mejor método de acceso a una sola relación. Así, se determinan todos los posibles ordenamientos de juntas, se determina el costo de cada ordenamiento y se elige aquel con el costo mínimo.

Para la junta de dos relaciones, la relación cuyos tuplos se leen primero se llama *externa*, mientras que la otra, cuyos tuplos se encuentran de acuerdo a los valores obtenidos de la relación externa, se llama relación *interna*. Una decisión importante es determinar el camino de acceso menos costoso a la relación interna.

Al considerar las juntas, hay dos algoritmos disponibles. El primero, llamado de *ciclos anidados*, compone el producto de las dos relaciones y se puede observar mediante el siguiente pseudocódigo.

Para cada tuplo de la relación externa (con cardinalidad n_1)
Para cada tuplo de la relación interna (con cardinalidad n_2)
junta los dos tuplos si el predicado de la junta es verdadero
end
end

Claramente, este método tiene complejidad $n_1 * n_2$.

El segundo método, llamado *junta-mezcla*, consiste en mezclar dos relaciones ordenadas sobre el atributo de la junta. Los índices del atributo de la junta se pueden usar como caminos de acceso. Si el criterio de junta es igualdad, el costo de juntar dos relaciones de n_1 y n_2 páginas, respectivamente, es proporcional a $n_1 + n_2$.

Ejemplo Considere la consulta q_1 del Ejemplo . La gráfica de juntas de se muestra en la Figura Suponga que se tienen los siguientes índices:

E tiene un índice en ENO
G tiene un índice en GNO
J tiene un índice en JNO y un índice en JNOMBRE

Supongamos que se seleccionan los siguientes caminos de acceso mejores para cada relación:

E: recorrido secuencial (ya que no hay selección sobre E)
G: recorrido secuencial (ya que no hay selección sobre G)
J: índice sobre JNOMBRE (ya que hay una selección en J basada en JNOMBRE)

La construcción dinámica del árbol de estrategias alternativas se presenta en la Figura. Note que el número de ordenamientos posibles de juntas es 3! los cuales son:

E > < G > < J
G > < J > < E
J > < E > < G
J > < G > < E

G > < E > < J
E ´ J > < G
J ´ E > < G

La operación marcadas como "podadas" se eliminan en forma dinámica. El primer nivel del árbol indica los mejores caminos de acceso a una sola relación. El segundo nivel indica, el mejor método de junta con cualquier otra relación. Las estrategias que utilizan el pruducto Cartesiano son inmediatamente podadas. Suponga que (E > < G) y (G > < J) tienen un costo más alto que (G > < E) y (J > < G), respectivamente. Así ellas pueden ser podadas ya que existe un orden equivalente con costo menor. Las dos posibilidades restantes están dadas en el tercer nivel del árbol. El mejor orden de las juntas es el de costo menor entre ((G > < E) > < J) y ((J > < G) > < E). El último es el único que tienen un índice útil en el atributo de selección y acceso directo a los tuplos de junta entre G y E. Por lo tanto, éste es elegido con el siguiente método de acceso:
Selecciona J usando el índice sobre JNOMBRE
Junta con G usando el índice sobre JNO
Junta con E usando el índice sobre ENO

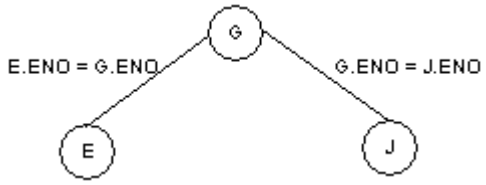


Figura Gráfica de juntas.

Ordenamiento de juntas para consultas fragmentadas

Como se puede apreciar en la sección anterior, el ordenamiento de juntas es un aspecto importante para la optimización de consultas centralizadas. El ordenamiento de las juntas en ambientes distribuidos es aún más importante ya que las juntas entre fragmentos pueden incrementar los costos de comunicación.

Consideremos inicialmente el caso más simple de transferencia de operandos en una junta simple. La consulta es R > < S en donde R y S son relaciones almacenadas en nodos diferentes. La elección obvia es transferir la relación más pequeña al nodo con la relación más grande como se ilustra en la Figura

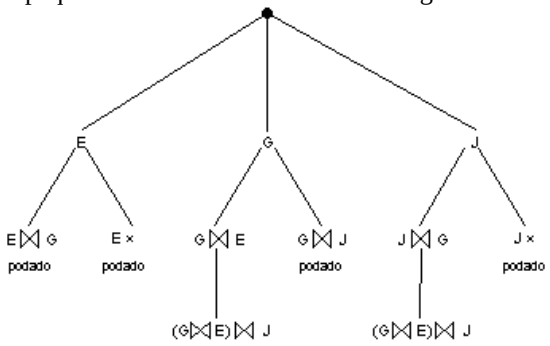


Figura Alternativas para el ordenamiento de juntas.

Consideremos ahora el caso en donde hay más de dos relaciones en la junta. El objetivo sigue siendo transferir los operandos más pequeños. La dificultad aparece del hecho de que las operaciones de juntas pueden reducir o incrementar el tamaño de los resultados intermedios. Así, estimar el tamaño de los resultados de las juntas es obligatorio pero difícil. Una solución es estimar el costo de comunicación de todas las estrategias posibles y elegir la de costo menor. Sin embargo, el número de estrategias crece rápidamente con el número de relaciones. Este enfoque, utilizado por System R*, hace que la optimización sea costosa pero se puede amortizar si la consulta se ejecuta con frecuencia.

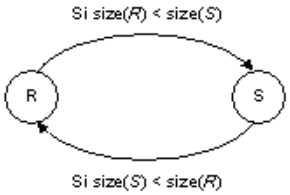


Figura Transferencia de operandos en una operación binaria.

Ejemplo . Considere la siguiente consulta expresada en el álgebra relacional:

$J >< _{JNO} E >< _{ENO} G$

cuya gráfica de juntas se presenta en la Figura 4.20. Note que se han hecho suposiciones acerca de la ubicación de cada relación. Esta consulta se puede ejecutar en al menos cinco estrategias diferentes. R ® nodo *j* denota que la relación R se transfiere al nodo *j*.

1. E ® nodo 2
   Nodo 2 calcula E = E > < G
   E ® nodo 3
   Nodo 3 calcula E > < J
2. G ® nodo 1
   Nodo 1 calcula E = E > < G
   E ® nodo 3
   Nodo 3 calcula E > < J
3. G ® nodo 3
   Nodo 3 calcula G = G > < J
   G ® nodo 1
   Nodo 1 calcula G > < E
4. J ® nodo 2
   Nodo 2 calcula J = J > < G
   J ® nodo 1
   Nodo 1 calcula J > < E
5. E ® nodo 2
   J ® nodo 2

Nodo 2 calcula E > < J > < G

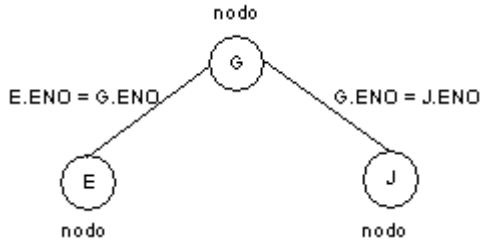


Figura Gráfica de juntas distribuida.

Optimización de consultas distribuidas

En esta sección se ilustrará brevemente el uso de las técnicas presentadas previamente en las extensiones distribuidas de INGRES y System R.

Algoritmo Distribuido de INGRES

El algoritmo de optimización de consultas para INGRES distribuido se deriva del algoritmo usado en INGRES centralizado. La función objetivo del algoritmo pretende minimizar la combinación del costo de comunicación junto con el tiempo de respuesta. Dado que estos criterios pueden presentar conflictos entre sí se considera una combinación pesada de ambos factores. Este algoritmo ignora el costo de transmisión de los datos al nodo de resultados. El algoritmo también toma ventaja de la fragmentación, pero únicamente considera fragmentación horizontal. El algoritmo utiliza mensajes de tipo broadcast, de uno a todos.

Ejemplo Considere la consulta

E > < G > < J

y suponga que la relación E esta fragmentada donde la ubicación de cada fragmento se muestra a continuación:

Nodo 1	Nodo 2
E_1	E_2
G	J

Existen varias estrategias posibles de ejecución. Dos de ellas son las siguientes:

1. Ejecutar la consulta completa E > < G > < J transmitiendo E_1 y G al nodo 2.
2. Ejecutar (E > < G) > < J transfiriendo E_1 > < G y G al nodo 2.

La elección entre las dos posibilidades requiere de una estimación del tamaño de los resultados intermedios. Por ejemplo, si *size*(E_1 > < G) > size(E_1), la estrategia 1 es mejor.

Algoritmo R*

El algoritmo de optimización de consultas distribuidas R* es una extensión sustancial a las técnicas desarrolladas para el optimizador de System R. La función de costo considera el costo de procesamiento local así como el costo de comunicación. Sin embargo, no utiliza fragmentos sino relaciones completas como unidades de distribución. La compilación de la consulta es una tarea distribuida en R*, la cual es coordinada por un nodo *maestro*. El optimizador del maestro hace todas las decisiones entre nodos, tales como la selección de los nodos de ejecución así como el método de transferencia de datos. Los nodos *aprendices* que son todos los otros nodos involucrados en la consulta hacen únicamente decisiones locales, tales como, el orden de las juntas en la consulta local y generan planes de acceso local para la consulta.

Para juntar dos relaciones existen tres nodos candidatos: el nodo de la primera relación, el nodo de la segunda relación o un tercer nodo (por ejemplo, el nodo donde se hará una junta con una tercera relación). En R*, se utilizan dos métodos para las transferencias de datos entre nodos.

1. *Transfiere todo*. La relación completa es transferida al nodo donde se realiza la junta y almacenada en una relación temporal antes de hacer la junta. Si el algoritmo de la junta es por medio de una mezcla, la relación no necesita ser almacenada y el nodo que ejecuta la junta puede procesar los tuplas conforme llegan. Este método requiere de grandes transferencias de datos, pero utiliza pocos mensajes. Es adecuado si las relaciones son relativamente pequeñas.
2. *Lee conforme lo necesita*. La relación externa es recorrida secuencialmente y, para cada tuplo, el valor de la junta se envía al nodo de la relación interna, el cual selecciona los tuplos internos que satisfacen el predicado de la junta y envía los tuplos seleccionados al nodo con la relación externa. La cantidad de mensajes es proporcional al tamaño de la relación externa. Este método es adecuado si las relaciones son grandes y la selectividad de la junta es buena.

Dada la junta de una relación externa *R* con una relación interna *S* en el atributo *A*, existen cuatro estrategias para realizar la junta. A continuación se describirá cada estrategia en detalle y se proporcionará en cada caso una versión simplificada de la fórmula de costo. *LC* denota el costo de procesamiento local (I/O + CPU) y CC denota el costo de comunicación.

Estrategia 1. Se transfiere la relación externa completa al nodo con la relación interna. En este caso los tuplos externos se pueden juntar con *S* conforme llegan. Así se tiene

Costo total = *LC*(recuperar *card*(*R*) tuplos de *R*)
+ *CC*(*size*(*R*))
+ *LC*(recuperar *s* tuplos de *S*)**card*(*R*)

Estrategia 2. Se transfiere la relación interna completa al nodo con la relación externa. En este caso los tuplos internos no se pueden juntar conforme llegan y se tienen que almacenar en una relación temporal *T*.

Costo total = *LC*(recuperar *card*(*S*) tuplos de *S*)
+ *CC*(*size*(*S*))
+ *LC*(almacenar *card*(*S*) tuplos en *T*)
+ *LC*(recuperar *card*(*R*) tuplos de *R*)
+ *LC*(recuperar *s* tuplos de *T*)**card*(*R*)

Estrategia 3. Se obtienen los tuplos de la relación interna conforme se necesitan para cada tuplo de la relación externa. En este caso, para cada tuplo de *R*, el atributo de junta se envía al nodo de *S*. Luego *s* tuplos de *S* que satisfacen el atributo son recuperados y envíados al nodo de *R* para ser juntados conforme llegan.

Costo total = *LC*(recuperar *card*(*R*) tuplos de *R*)
+ *CC*(*length*(*A*)) * *card*(*R*)
+ *LC*(recuperar *s* tuplos de *S*)**card*(*R*)
+ *CC*(*s* * *length*(*S*)) * *card*(*R*)

Estrategia 4. Se mueven las relaciones a un tercer nodo y se calcula la junta ahí. En este caso la relación interna se mueve al tercer nodo y se almacena en una relación temporal *T*. Luego la relación externa se mueve al tercer nodo y sus tuplos se juntan con *T* conforme llegan.

Costo total = *LC*(recuperar *card*(*S*) tuplos de *S*)
+ *CC*(*size*(*S*))
+ *LC*(almacenar *card*(*S*) tuplos en *T*)
+ *LC*(recuperar *card*(*R*) tuplos de *R*)
+ *CC*(*size*(*R*))
+ *LC*(recuperar *s* tuplos de *T*)**card*(*R*)

Optimización Local de Consultas

El trabajo de la última capa se efectúa en todos los nodos con fragmentos involucrados en la consulta. Cada subconsulta que se ejecuta en un nodo, llamada consulta local, es optimizada usando el esquema local del nodo. Hasta este momento, se pueden eligen los algoritmos para realizar las operaciones relacionales. La optimización local utiliza los algoritmos de sistemas centralizados.

El último paso en el esquema de procesamiento de consultas distribuidas es la optimización de consultas locales. Después del material presentado en este capítulo, debe quedar claro que para ello se utilizan las técnicas de optimización de consultas centralizadas. El propósito es seleccionar el mejor camino de acceso para una consulta local.

6 Administración de transacciones distribuidas

Objetivo: El alumno explicará la importancia de controlar y administrar transacciones distribuidas

El modulo Administrador de transacciones distribuidas se encarga de asegurar la atomicidad de las transacciones globales y de cada componente de la subtransaccion. El administrador de transacciones coordina las transacciones generadas por los programas de aplicación, a través de una comunicación con el scheduler, el modulo responsable de implementar una estrategia particular para el control de concurrencia. El scheduler es una secuencia de las operaciones para un conjunto de transacciones concurrentes que preserva el orden de las operaciones en cada una de las transacciones individuales. El objetivo del scheduler (horario) es maximizar la concurrencia sin permitir la ejecución concurrente de transacciones que interfieran unas con otras y que por tanto comprometan la consistencia de la base de datos. En el caso de alguna falla el modulo de recuperación asegura que la base de datos se restaure a el estado inmediato anterior a la transacción fallida y por ende a un estado consistente. En los DDBMS estos módulos existen en cada DBMS local, pero además existe un administrador de transacciones globales o coordinador de transacciones en cada sitio que coordina la ejecución de las transacciones globales y locales iniciadas en ese sitio.
La comunicación entre los sitios se da por el componente de comunicación de datos (es decir los administradores de transacciones en diferentes sitios no se comunican directamente entre sí.

6.1 Atomicidad

Centralizado: Las sentencias que se ejecutan en una transacción deben ejecutarse como un todo.
Distribuido: Las subtransacciones que se ejecutan en una transacción global deben ejecutarse como un todo.

6.2 Control de Concurrencia

El control de concurrencia y la recuperación en un entorno distribuido, debe presentar soluciones a problemas que no surgen en ambientes centralizados. Por ejemplo:

a) Manejar multiples copias de los elementos de datos: El método de control de concurrencia tiene a obligación de mantener la consistencia entre estas copias. El método de recuperación debe cuidar que una copia sea consistente con las demás si el sitio en el que la copia estaba almacenada falla y se recupera posteriormente.
b) Fallos de sitios individuales: EL Manejador de BDD debe continuar operando con sus sitios activos, si es posible, cuando fallen uno o mas sitios individuales. Cuando un sitio se recupere, su bd local se deberá poner al dia con los demás sitios antes de que se incorpore al sistema.
c) Fallo de enlaces de comunicación: El sistema debe ser capaz de manejar el fallo de uno o mas de los enlaces de comuinicaciones que conectan los sitios. Un caso extremo de este problema es que puede haber partición de la red. Esto divide los sitos en dos o mas particiones, dentro de las cuales los sitios pueden comunicarse entre si, pero no con sitios de otras particiones.
d) Confirmacion distribuida: Puede haber problemas para confirmar una transacción que esta teniendo acceso a base de datos almacenadas en multiples sitios si algunos de estos fallan durante el proceso de confirmación. A menudo se utiliza el protocolo de confirmación endos fases (tema ya visto) para resolver este problema.
e) Bloqueo mortal distribuido: Puede ocurrir un bloqueo mortal (deadlock) entre varios sitios, lo que hace necesario extender las técnicas de manejo de deadlocks.

Control de concurrencia distribuido basado en una copia distinguida.
Se han diseñado varios métodos de control de concurrencia para manejar los elementos de datos replicados en una base de datos distribuida, que se basan en la extensión de las técnicas de control de concurrencia de las BD centralizadas.
La idea es designar una copia determinada de cada elemento de base de datos como copia distinguida. Los bloqueos para este elemento se asocian a la copia distinguida y todas las solicitudes de bloqueo o desbloqueo se envían al sitio que contiene esa copia.

Tecnica de sitio primario: En este método se designa un solo sitio primario com sitio coordinador para todos los elementos de la base de datos. Por tanto todos los bloqueos se mantienen en ese sitio, y todas las solicitudes de bloqueo y desbloqueo se envían a ese sitio. Asi pues este método es una extenssion del bloqueo centralizado. Por ej. Si todas las transacciones siguen e protocolo de candado de dos fases, la seriabilidad esta garantizada. La ventaja de este enfoque es que es una simple extensión del esquema de candados centralizado y no es complejo. Sin embargo como todas las solicitudes de candado se van a un sitio este sitio se sobrecarga. Si este sitio falla, se paraliza todo el sistema.

Sitio primario con sitio de respaldo: Este enfoque busca subsanar la segunda desventaja del método anterior designando un segundo sitio como sitio de respaldo. Toda la información de bloqueo se mantiene al tanto en el sitio primario com en el de respaldo. En caso de fallar el sitio primario el de respaldo asume sus funciones y el procesamiento puede reanudarse una vez que se elija el nuevo sitio de respaldo y la información de estado de los candados se copie en el. Sin embargo, el proceso de adquisición de candados se hace mas lento, porque todas las solicitudes de candados se deben guardar tanto en el sitio primario como en el de respaldo antes de enviar una respuesta a la transacción solicitante.

Técnica de copia primaria: Este método intenta distribuir la carga de la coordinación de los candados entre varios sitios manteniendo las copias distinguidas de diferentes elementos de datos almacenados en diferentes sitios. El fallo de un sitio afecta a todas las transacciones que estén teniendo acceso a candados sobre los elementos de bd cuyas copias primarias residan en ese sitio, pero las demás transacciones no resultan afectadas. Este método también puede usar sitios de respaldo para elevar la fiabilidad y disponibilidad.

Elección de un nuevo sitio coordinador en caso de fallo. Siempre que un sitio coordinador falle en cualquiera de las técnicas anteriores, los sitios siguen activos deberán elegir un nuevo coordinador. En el caso del enfoque del sitio primerio sin sitio de respaldo, será preciso abortar y reiniciar todas las transacciones en ejecución, y el proceso de recuperación será bastante tedioso. Parte de dicho proceso implica elegir un nuevo sitio primario y crear un proceso administrador de candados y un registro de toda la información de los candados en ese sitio. En los métodos que usan sitios de respaldo, el procesamiento de transacciones se suspende mientras el sitio de respaldo se designa como nuevo sitio primario, se escoge un nuevo sitio de respaldo y se envían a el copias de toda la información de los candados del nuevo sitio primario.
Si el sitio de respaldo X esta a punto de convertirse en el nuevo sitio primario, X puede escoger el nuevo sitio de respaldo entre los sitios activos del sistema. Sin embargo, si no hay sitio de respaldo, o si están caidos tanto el sitio primario como el de respaldo, se puede seguir el proceso denominado elección para escoger el nuevo sitio coordinador. En este proceso, cualquier sitio Y que intente comunicarse repetidamente con el sitio coordinador y fracase puede suponer que el coordinador esta caído e iniciar el proceso de elección enviando un mensaje a todos los sitios activos en el cual proponga que Y se convierta en el nuevo coordinador. Tan pronto como Y reciba una mayoría de votos afirmativos, puede declararse nuevo coordinador.

Control de concurrencia distribuido basado en la votación
En el método de votación, no hay copia distinguida, sino cada solicitud de candado se envía a todos los sitios que incluyan una copia del elemento de datos. Cada copia mantiene su propio candado y puede conceder o rechazar la solicitud. Si la mayoría de las copias otorgan un candado a la transacción que lo solicita, esta poseera el candado e informara a todas las copias que le ha sido concedido. Si una transacción no recibe la mayoría de los votos de concesión del candado durante un cierto periodo de tiempo predefinido, cancelara su solicitud e informara de ello a todos los sitios.
El método de votación se considera un método de control de concurrencia verdaderamente distribuido, ya que la responsabilidad de la decisión recae en todos los sitios implicados. Sin embargo requiere un trafico mas alto de mensajes entre sitios.

LO QUE ESTA EN ITALICA TAL VEZ NO SE DA, CREO QUE YA SE DIO EN EL 3ER CAPITULO
El control de concurrencia administra la ejecución de operaciones simultáneas sin permitir que estas operaciones interfieran unas con otras. El control de concurrencia se encarga de maximizar la concurrencia y garantizar la consistencia y atomicidad de las transacciones, debe evitar o manejar errores de concurrencia como el interbloqueo.
Recordemos que se debe dar la Transparencia en la concurrencia
En un DBMS centralizado, esto significa que todas las transacciones que se ejecuten concurrentemente deben ejecutarse independientemente y deben estar consistentes con los resultados que se obtuvieran si las transacciones se ejecutaran una a un tiempo, en cualquier orden arbitrario. Sin embargo, el DDBMS debe asegurarse que las transacciones locales y globales que se ejecutan concurrentemente no interfieran entre sí. De igual manera, el DDBMS debe asegurar la consistencia de todas las subtransacciones de una transacción global.

Los mecanismos de control de concurrencia deben de
a) poder manejar fallas de comunicación y de sitios.
b) Permitir paralelismo para satisfacer requerimientos de rendimiento.

c) *Tener un buen rendimiento a pesar de problemas de comunicaciones*

Un problema típico de ambientes distribuidos corresponde al problema de consistencia en múltiples copias, el cual ocurre cuando un dato es replicado en diferentes sitios. Para mantener la consistencia de la base de datos global, cuando un dato se modifica, todas las copias replicadas deben reflejar este cambio. Si no se refleja la actualización en las copias, existe un problema de consistencia en la base de datos global. Cuando la actualización de un dato se realiza en todas las copias a un mismo tiempo, se dice que las actualizaciones se realizan de manera síncrona, como parte de una transacción que las comprenda. Pero también puede ser ejecutado de manera asíncrona, es decir en algún momento posterior a la actualización del primer dato.

*****************OPCIONAL *********

El scheduler (calendarizador de horario) es una secuencia de las operaciones para un conjunto de transacciones concurrentes que preserva el orden de las operaciones en cada una de las transacciones individuales. El objetivo del scheduler (horario) es maximizar la concurrencia sin permitir la ejecución concurrente de transacciones que interfieran unas con otras y que por tanto comprometan la consistencia de la base de datos.

Un schedule (horario) S consiste de una secuencia de operaciones de un conjunto de n transacciones sujetos a la restricción de que el orden de las operaciones por cada transacción se preserva en el Schedule. Esto es cada transacción Ti en un Schedule S, el orden de las operaciones en Ti debe ser el mismo en Schedule S.

Horario Serial: Un horario donde las operaciones de cada transacción son ejecutadas consecutivamente sin alguna operación interpolada de otras transacciones. La ejecución serial evita muchos problemas, sin importar cual horario seriado se escoja, la base de datos nunca quedara en estado inconsistente.

Horario no-serial: Un horario donde las operaciones de un conjunto concurrente de transacciones son interpoladas.

Si un conjunto de transacciones se ejecuta concurrentemente se dice que el horario no serial es correcto si produce los mismos resultados como alguna ejecución serial. Un horario así se le llama serializadle.

El objetivo de la serializabilidad es encontrar horarios no seriales que permitan a las transacciones ejecutarse concurrentemente sin interferencia entre ellas.

En la serializabilidad el orden de las operaciones de lectura y escritura es importante.

a) *Si dos transacciones solo leen un objeto, no existe conflicto y el orden no es importante.*
b) *Si dos transacciones leen o escriben en entidades completamente separadas, no hay conflicto y el orden no importa.*
c) *Si una transacción escribe a un objeto de bd. Y otro lee o escribe en el mismo objeto, el orden de la ejecución es importante.*

******************* Hasta aquí fue opcional ****************

Serializacion distribuida

Si la calendarización de la ejecución de las transacciones en cada sitio fuera serializable, entonces el horario global (global scheduler, la unión de todos los horarios locales seria también serial. Esto requiere que todas las subtransacciones aparezcan en el mismo orden en el horario serial de todos los sitios.

Es decir, si la subtransaccion de Ti en el sitio S1 se denota como T^1_i *, debemos asegurar que si* $T^1_i < T^1_j$, *entonces*

$T^x_i < T^x_j$ *para todos los sitios* S_X *en el cual* T_i *y* T_j *tienen subtransacciones.*

La solución al control de concurrencia en un ambiente distribuido se basa en dos mecanismos.

El mecanismo de candados y el mecanismo de marcas de tiempo.

Por lo tanto, dado un conjunto de transacciones a ser ejecutadas de manera concurrente, se tiene que

a) *El mecanismo de candados garantiza que la ejecución concurrente es equivalente a alguna ejecución impredecible serial de aquellas transacciones.*
b) *El mecanismo de marcas de tiempo garantiza que la ejecución concurrente es equivalente a una ejecución específica serial de aquellas transacciones, correspondiente al orden de las marcas de tiempo.*

Si la base de datos es centralizada o fragmentada pero no replicada y todas las transacciones ya sea locales o pueden ejecutarse en un solo sito remoto entonces se pueden controlar la concurrencia a través de los mecanismos propios de ambiente centralizado (horario, horario serial y horario no serial)

Los mecanismos de control de concurrencia deben ser extendidos para sistemas de base de datos distribuidas y los datos son replicados o las transacciones requieren acceso a datos en varios sitios.

6.2.1 Mecanismos de candados

El protocolo Two phase locking se utiliza para el manejo de candados distribuidos, existe diversos tipos de 2PL

a) *Candado de dos fases centralizado C 2PL*
   Un solo sitio mantiene toda la información de candados
   Un solo calendarizador de horarios o manejador de candados que libera o asigna candados.
   1.- El coordinador de transacciones en sitio S1 divide la transacción en varias subtransacciones, usando información del catalogo global. El coordinador asegura la consistencia, debe asegurarse de que todas las copias se reflejen. Para esto, el coordinador hace requerimientos de candados exclusivos en todas las copias antes de actualizar cada copia y liberar los candados.
   2.- Los manejadores de transacciones locales correspondientes a la transacción global requiere y liberan candados del manejador de candados usando las mismas reglas del 2PL.
   3. - El manejador de candados centralizado checa que un requerimiento por candado en cada dato sea compatible con los candados que actualmente existen. De ser así y si hay recursos lo otorga, de lo contrario lo pone en cola de espera por asignación de candado.
   Ventaja sencillo, desventaja contención en el manejador de candados porque este es el que controla lo de toda la base de datos global, y baja confiabilidad porque en caso de falla en el sitio donde está el manejador de candados se provocan fallas del sistema distribuido global.
b) *Copia primaria de Candado de dos fases*
   Varios manejadores de candados
   Cada manejador de candados es responsable de otorgar y controlar los candados de un conjunto de sitios.
   Por cada objeto replicado, una copia se selecciona como copia primaria y las otras serán secundarias o esclavas.
   La diferencia entre el anterior es que cuando un objeto se actualiza el coordinador de transacciones debe determinar donde está la copia primaria para mandar el requerimiento de candado al correspondiente manejador de candados. Se requiere candado exclusivo sobre la copia primaria que será actualizada. Una vez que esta copia sea actualizada, el cambio se propaga a las esclavas. No es necesario realizar todas las actualizaciones de manera atómica y por tanto puede haber lecturas de datos no actualizados, por tanto solo se garantiza que los datos primarios se encuentren siempre actuales. Es viable cuando no hay muchas actualizaciones en datos replicados.
c) *Candado de dos fases distribuido*
   Manejadores de candado en cada sitio en todos los sitios
   Cada Manejador de candado se encarga de los candados de su sitio
   Si no hay datos replicados entonces es igual al primary copy 2PL
   Si hay datos replicados existe un protocolo de control de replica ROWA Rea done write all. Esto significa que cualquier copia de un objeto replicado puede ser usado para la operación de lectura, pero todas las copias deben de tener candado exclusivo antes de que el objeto se actualizado. El manejo de interbloqueos es más complejo.

6.2.2 Marcas de tiempo

Mecanismos de marcas de tiempo

El objetivo del timestamping es ordenar las transacciones globalmente de tal forma que las transacciones más Viejas tengan marcas de tiempo más pequeñas. Dado que así, obtienen mayor prioridad en caso de conflicto. En caso de ambiente distribuido se sigue requiriendo generar timestamps únicos local y globalmente. El usar el reloj del sistema o un contador de eventos en cada sitio no sería recomendable como en el caso de los ambientes centralizados. Los relojes en diferentes sitios podrían no está sincronizados, o se podría generar el mismo valor de contador en sitios diferentes.

Se usa la concatenación del tiempo en el sitio local con un identificador único del sitio. < marcadetiempo, idsitio> (primero el timestamp y luego el identificador del sitio)

Cada sitio sincroniza su timestamp. Cada sitio incluye su timestamp en los mensajes entre sitios.

Ejemplo

Sitio 1 con timestamp <10,1> manda mensaje al sitio 2 con el timestamp actual <15,2>, después el sitio 2 no cambiaría su timestamp. Por otro lado, si el timestamp actual del sitio 2 fuera <5,2>, entonces el sitio dos cambiaria su timestamp a <11,2>

Cualquier mecanismo de manejo de candados y algunas marcas de tiempo pueden resultar en interbloqueos

6.2.3 Deadlocks

Interbloqueo distribuido: se presenta cuando se desarrolla una espera circular entre dos transacciones y cada una de estas solicita una actualización sobre la misma tabla, no permite a otros usuarios el recurso hasta que termine el proceso. Comúnmente se maneja este problema de concurrencia escogiendo una víctima y cancelándola para que el otro proceso termine satisfactoriamente. La victima será aquel proceso con menor tiempo de CPU ejecutado al momento del conflicto.

T1 iniciado en S1 y creando un agente en el sitio S2
T2 iniciado en S2 y creando un agente en el sitio S3
T3 iniciado en S3 y creando un agente en el sitio S1

Las transacciones asignan candados compartidos (lectura) y candado exclusivos (escritura) como se ilustra a continuación donde el read_lock(ti,xj) denota un candado compartido por la transacción ti en el objeto xj y el write_lock(ti,xj) denota un candado exclusivo por la transacción Ti en objeto xj.

Tiempo	S1	S2	S3
T1	read_lock(t1x1)	write_lock(t2,y2)	read_lock(ti3,z3)
T2	write_lock(t1,y1)	write_lock(t2,z2)	
T3	write_lock(t3,x1)	write_lock(t1,y2)	write_lock(t2,z3)

Individualmente no hay ciclos T3→T1, T1→T2, T2→T3, sin embargo al juntarlos si existe el ciclo T1→T2→T3→T1

Existen tres métodos comunes para la detección y manejo de interbloqueos.

a) Detección de interbloqueos centralizado
b) Detección de interbloqueos jerárquico
c) Detección de interbloqueos distribuido

TAREA: TRAER LOS TRES MÉTODOS DE DETECCIÓN DE INTERBLOQUEOS PG. 742 DE COLLONY

7 Seguridad

Objetivo: El alumno explicará la importancia de mantener la seguridad en una base de datos distribuida y como se lleva a cabo.

7.1 Conceptos básicos

Diseñar un sistema confiable que se pueda recuperar de fallas requiere identificar los tipos de fallas con las cuales el sistema tiene que tratar. Así, los tipos de fallas que pueden ocurrir en un SMBD distribuido son:

1. **Fallas de transacciones**. Las fallas en transacciones se pueden deber a un error debido a datos de entrada incorrectos así como a la detección de un interbloqueo. La forma usual de enfrentar las fallas en transacciones es abortarlas. Experimentalmente, se ha determinado que el 3% de las transacciones abortan de manera anormal.
2. **Fallas del sistema**. En un sistema distribuido se pueden presentar fallas en el procesador, la memoria principal o la fuente de energía de un nodo. En este tipo de fallas se asume que el contenido de la memoria principal se pierde, pero el contenido del almacenamiento secundario es seguro. Típicamente, se hace diferencia entre las fallas parciales y fallas totales del nodo. Una falla total se presenta en todos los nodos del sistema distribuido. Una falla parcial se presenta solo en algunos nodos del sistema.
3. **Fallas del medio de almacenamiento**. Se refieren a las fallas que se pueden presentar en los dispositivos de almacenamiento secundario que almacenan las bases de datos. Esas fallas se pueden presentar por errores del sistema operativo, por errores del controlador del disco, o del disco mismo.
4. **Fallas de comunicación**. Las fallas de comunicación en un sistema distribuido son frecuentes. Estas se pueden manifestar como pérdida de mensajes lo que lleva en un caso extremo a dividir la red en varias subredes separadas.

TRANSACCIONES

7.2 Protocolo Commit

Como con los protocolos de recuperación local, las versiones distribuidas ayudan a mantener la atomicidad y durabilidad de las transacciones. La ejecución de los comandos **begin_transaction**, **read** y **write** no provoca ningún cambio significativo. Sin embargo, la ejecución de las operaciones **commit**, **abort** y **recover** requieren del desarrollo de un protocolo especial para cada una de ellas en el cual participan todos los nodos de la red que intervienen en una transacción. De manera breve, a continuación se presentan algunos problemas que aparecen en sistemas distribuidos y que no se presentan en sistemas centralizados.

Cómo ejecutar un comando **commit** para transacciones distribuidas?
Cómo asegurar la atomicidad y durabilidad de las transacciones distribuidas?
Si ocurre una falla, cómo pueden, los nodos que permanecen funcionando, seguir operando normalmente?

Idealmente, la ocurrencia de la falla en un nodo no debe forzar a los otros nodos a tener que esperar a que se repare la falla en el nodo para terminar una transacción. A esta propiedad se le conoce como *no-bloqueante*.

De manera similar, cuando ocurre una falla, cómo terminar una transacción que se estaba ejecutando al momento de la falla? Idealmente un nodo con falla debería poder terminar una transacción sin tener que consultar a los otros nodos. Esta es una propiedad de *independencia* del nodo con falla con respecto a los nodos sin falla. Una recuperación independiente supone que existe una estrategia no-bloqueante en el manejo de fallas.

Para facilitar la descripción de los protocolos de confiabilidad distribuida se supondrá que en el nodo que se origina una transacción hay un proceso, llamado el *coordinador*, que ejecuta las operaciones de la transacción. El coordinador se comunica con procesos *participantes* en los otros nodos los cuales lo ayudan en las operaciones de la transacción.

7.2.1 Compromisos de dos fases /Commit de dos fases

El compromiso de dos fases (two-phase commit) es un protocolo muy simple y elegante que asegura la atomicidad de las transacciones distribuidas. Extiende los efectos de una operación local de commit a transacciones distribuidas poniendo de acuerdo a todos los nodos involucrados en la ejecución de una transacción antes de que los cambios hechas por la transacción sean permanentes.
Las fases del protocolo son las siguientes:

- *Fase 1*: el coordinador hace que todos los participantes estén listos para escribir los resultados en la base de datos.
- *Fase 2*: Todos escriben los resultados en la base de datos.

La terminación de una transacción se hace mediante la *regla del compromiso global*:

1. El coordinador aborta una transacción si y solamente si al menos un participante vota por que se aborte.
2. El coordinador hace un commit de la transacción si y solo si todos los participantes votan por que se haga el commit.

La operación del compromiso de dos fases entre un coordinador y un participante en ausencia de fallas se presenta en la Figura en donde los círculos indican los estados y las líneas entrecortadas indican mensajes entre el coordinador y los participantes. Las etiquetas en las líneas entrecortadas especifican la naturaleza del mensaje.
En la Figura se presentan las dos fases de comunicación para realizar un commit en sistemas distribuidos. El esquema presentado en la figura se conoce como centralizado ya que la comunicación es solo entre el coordinador y los participantes; los participantes no se comunican entre ellos.

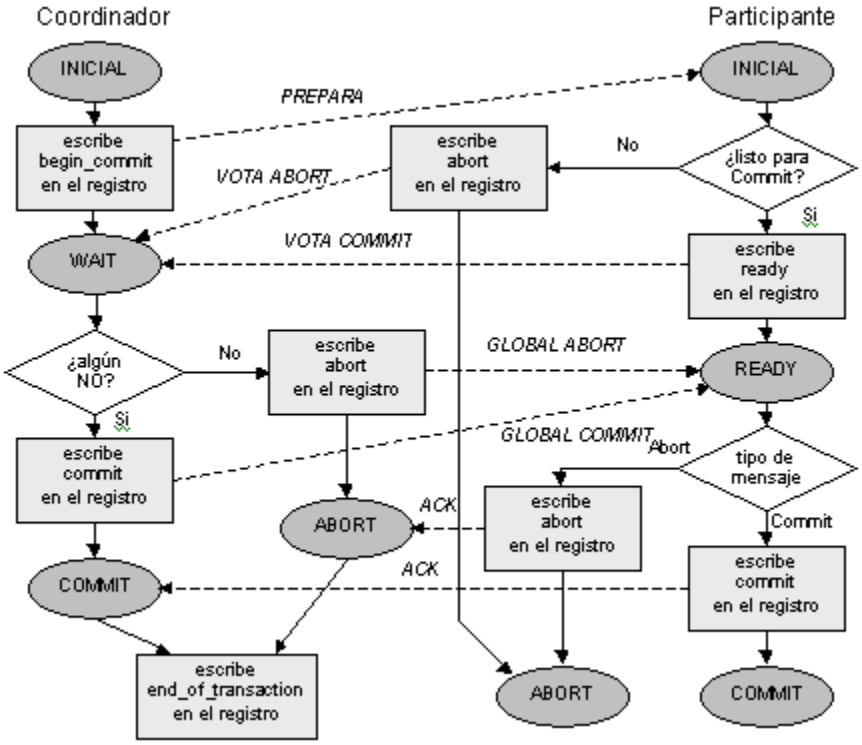


Figura Acciones del compromiso de dos fases.

Otra alternativa es una comunicación lineal, en donde los participantes se pueden comunicar unos con otros. Existe un ordenamiento entre los nodos del sistema. La estructura se presenta en la Figura 7.11. Un participante, *P*, recibe un mensaje vote-abort o vote-commit de otro participante, *Q*. Si P no está listo para hacer un commit, envía un vote-abort al siguiente participante, *R*, y la transacción se aborta en este punto. Si P el participante está de acuerdo en hacer un commit, envía un vote-commit a *R* y entra al estado de LISTO. Este proceso continúa hasta el último participante poniendo fin a la primera fase. En la segunda fase, si el último participante está de acuerdo en hacer un commit envía un global-commit al participante anterior. En caso contrario, envía un global-abort. Así en la segunda fase, *P* recibe un mensaje de *R* informándole si se hace un global-commit o un global-abort. Este mensaje se propaga a *Q* y así hasta alcanzar al coordinador.

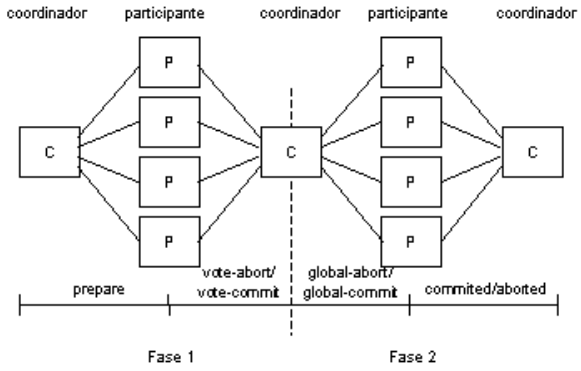


Figura 7.10. Estructura centralizada de comunicaciones para el compromiso de dos fases.

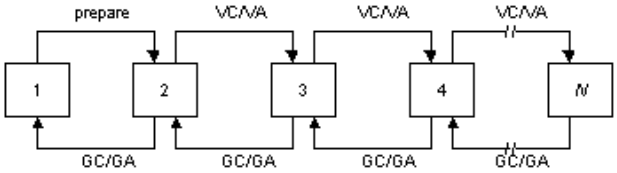


Figura 7.11. Estructura lineal de comunicaciones para el compromiso de dos fases.

Otro estructura de comunicación usual para implementar los compromisos de dos fases involucra la comunicación entre todos los participantes durante la primera fase del protocolo de manera que todos ellos alcanzan su punto de terminación en forma independiente. Esta versión, llamada la *estructura distribuida*, no requiere la segunda fase. En la Figura 7.12 se presenta la estructura distribuida en la cual el coordinador envía el mensaje de preparación a todos los participantes. Cada participante, entonces, envía su decisión a todos los otros participantes y al coordinador indicándola como un vote-commit o vote-abort. Cada participante espera los mensajes de todos los otros participantes y toma su decisión de acuerdo a la regla de compromiso global.

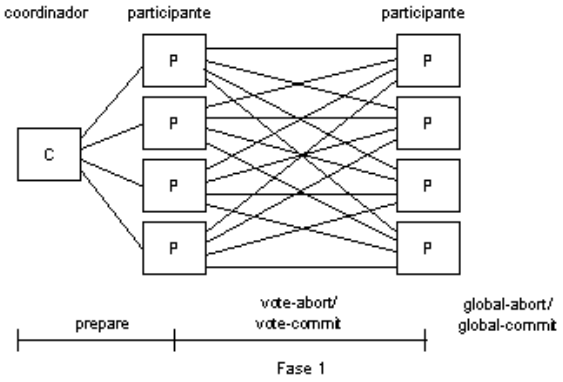


Figura 7.12. Estructura distribuida de comunicaciones para el compromiso de dos fases

SISTEMA

Información de recuperación

En esta sección se asumirá que ocurren únicamente fallas del sistema. Más adelante se considerará el caso de los otros tipos de fallas. Cuando una falla del sistema ocurre, el contenido de la base de datos volátil se pierde. Por lo tanto, el DBMS tiene que mantener cierta información acerca de su estado en el momento de la falla con el fin de ser capaz de llevar a la base de datos al estado en el que se encontraba antes de la falla. A esta información se le denomina *información de recuperación*.

La información de recuperación que el sistema mantiene depende del método con el que se realizan las actualizaciones. Existen dos estrategias para efectuarlas: en el lugar (*in place*) y fuera de lugar (*out-of-place*). En el primer caso, cada actualización se hace directamente en los valores almacenados en las páginas de los buffers de la base de datos. En la segunda alternativa, al crear un nuevo valor para un dato, éste se almacena en forma separada del valor anterior. De esta manera, se mantiene los valores nuevo y anterior.

Recuperación in-place

Dado que las actualización *in-place* hacen que los valores anteriores se pierdan, es necesario mantener suficiente información de los cambios de estado en la base de datos. Esta información se mantiene, por lo general, en el *registro de la base de datos* (database log). Así cada actualización, no solo cambia la base de datos, sino es también guardada en el registro de la base de datos (Figura 7.4).

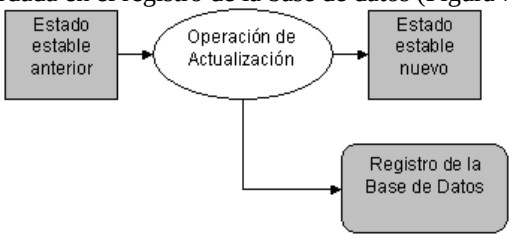


Figura 7.4. Ejecución de una operación de actualización.

El registro de la base de datos contiene información que es utilizada por el proceso de recuperación para restablecer la base de datos a un estado consistente. Esta información puede incluir entre otras cosas:

- el identificador de la transacción,
- el tipo de operación realizada,
- los datos accesados por la transacción para realizar la acción,
- el valor anterior del dato (imagen anterior), y
- el valor nuevo del dato (imagen nueva).

Considere el escenario mostrado en la Figura 7.5. El DBMS inicia la ejecución en el tiempo 0 y en el tiempo *t* se presenta una falla del sistema. Durante el periodo [0,*t*] ocurren dos transacciones, T_1 y T_2. T_1 ha sido concluida (ha realizado su commit) pero T_2 no pudo ser concluida. La propiedad de durabilidad requiere que los efectos de T_1 sean reflejados en la base de datos estable. De forma similar, la propiedad de atomicidad requiere que la base de datos estable no contenga alguno de los efectos de T_2.

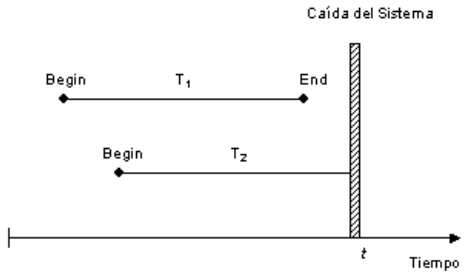


Figura 7.5. Ejemplo de una falla del sistema.

A pesar que T_1 haya sido terminada, puede suceder que el buffer correspondiente a la página de la base de datos modificada no haya sido escrito a la base de datos estable. Así, para este caso la recuperación tiene que volver a realizar los cambios hechos por T_1. A esta operación se le conoce como REDO y se presenta en la Figura 7.6. La operación de REDO utiliza la información del registro de la base de datos y realiza de nuevo las acciones que pudieron haber sido realizadas antes de la falla. La operación REDO genera una nueva imagen.

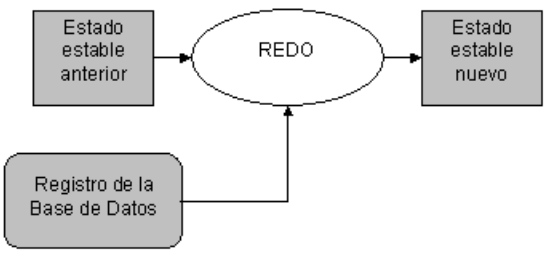


Figura 7.6. Operación REDO.

Por otra parte, es posible que el administrador del buffer haya realizado la escritura en la base de datos estable de algunas de las páginas de la base de datos volátil correspondientes a la transacción T_2. Así, la información de recuperación debe incluir datos suficientes para permitir deshacer ciertas actualizaciones en el nuevo estado de la base de datos y regrasarla al estado anterior. A esta operación se le conoce como UNDO y se muestra en la Figura 7.7. La operación UNDO restablece un dato a su imagen anterior utilizando la información del registro de la base de datos.

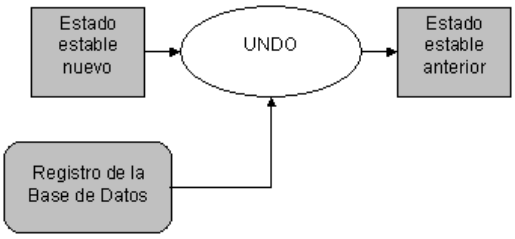


Figura 7.7. Operación UNDO.

De forma similar a la base de datos volátil, el registro de la base de datos se mantiene en memoria principal (llamada los *buffers de registro*) y se escribe al almacenamiento estable (llamado *registro estable*). Las páginas de registro se pueden escribir en el registro estable de dos formas: *síncrona* o *asíncrona*. En forma síncrona, también llamada un registro forzado, la adición de cada dato en el registro requiere que la página del registro correspondiente se mueva al almacenamiento estable. De manera asíncrona, las páginas del registro se mueven en forma periódica o cuando los buffers se llenan.

Independientemente de que la escritura del registro sea síncrona o asíncrona, se debe observar un protocolo importante para el mantenimiento del registro de la base de datos. Considere el caso donde las actualizaciones a la base de datos son escritas en el almacenamiento estable antes de que el registro sea modificado en el registro estable para reflejar la actualización. Si una falla ocurre antes de que el registro sea escrito, la base de datos permanecerá en forma actualizada, pero el registro no indicará la actualización lo que hará imposible recuperar la base de datos a un estado consistente antes de la actualización. Por lo tanto, el registro estable siempre debe ser actualizado antes de la actualización de la base de datos. A este protocolo se le conoce como registro antes de la escritura (write-ahead logging) y se puede especificar de la manera siguiente:

1. Antes de que la base de datos estable sea actualizada, las imágenes anteriores deben ser almacenadas en el registro estable. Esto facilita la realización de un UNDO.
2. Cuando la transacción realiza un commit, las imágenes nuevas tienen que ser almacenadas en el registro estable antes de la actualización de la base de datos estable. Esto facilita la realización de una operación REDO.

La interfaz completa del registro de la base de datos volátil y estable se presenta en la Figura 7.8.

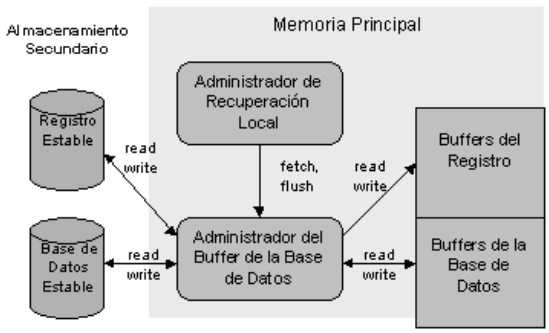


Figura 7.8. Interfaz entre el registro de la base de datos volátil y estable.

Recuperación out-of-place

Las técnicas de recuperación más comunes son de tipo in-place. Por lo tanto, aquí se presenta solo una breve descripción de las técnicas out-of-place.

1. **Shadowing**. Cuando ocurre una actualización, no se cambia la página anterior, sino se crea una página sombra con el nuevo valor y se escribe en la base de datos estable. Se actualizan los caminos de acceso de manera que los accesos posteriores se hacen a la nueva página sombra. La página anterior se retiene para propósitos de recuperación, para realizar una operación UNDO.
2. **Archivos diferenciales**. Para cada archivo F se mantiene una parte de solo lectura (FR), un archivo diferencial que consiste de la parte de inserciones DF+ y la parte de supresiones DF-. Así, el archivo completo consistirá de la unión de la parte de lectura más la parte de inserciones y a todo esto se le eliminan las supresiones realizadas.

   F = (FR È DF+) DF-

   Todas las actualizaciones se tratan como la supresión de un valor anterior y la inserción de un nuevo valor. Periódicamente, el archivo diferencial tiene que ser mezclado con el archivo base de solo lectura.

EJECUCION DE LOS COMANDOS DEL LRM

Existen cinco comandos que forman la interfaz al LRM. Ellos son: **begin_transaction**, **read**, **write**, **commit** y **abort**. En esta sección se discutirá cada uno de ellos y se presentará el comando **recover** cuya necesidad debe ser aparente después de la discusión anterior. La ejecución de los comandos **abort**, **commit** y **recover** es completamente dependiente de los algoritmos de recuperación que se usen. Por otra parte, los comandos **begin_transaction**, **read** y **write** son independientes del LRM.

La decisión fundamental en el diseño de administrador de recuperación local, el administrador del buffer y la interacción entre estas componentes es si el administrador de buffers obedece instrucciones del LRM tales como cuando escribir las páginas del buffer de la base de datos al almacenamiento estable. Específicamente, existen dos decisiones fundamentales:

1. Puede el administrador de buffers escribir las páginas del buffer actualizadas por una transacción en el almacenamiento estable durante la ejecución de la misma, o tiene que esperar las instrucciones del LRM para escribirlas en la base de datos estable? A este tipo de estrategia se le conoce como *fix*/*no-fix*.
2. Puede el LRM forzar al administrador del buffer para que escriba ciertas páginas del buffer en la base de datos estable al final de la ejecución de una transacción? A este tipo de estrategia se le conoce como *flush*/*no-flush*.

De acuerdo a lo anterior, existe cuatro posibles alternativas: no-fix/no-flush, no-fix/flush, fix/no-flush y fix/flush.

- *Begin_transaction*. Hace que el LRM escriba un comando begin_transaction en el registro de la base de datos.
- *Read*. El LRM trata de leer los datos especificados de las páginas del buffer que pertenecen a la transacción. Si el dato no se encuentra en esas páginas, envía un comando **fetch** al administrador del buffer para que los datos sean disponibles.
- *Write*. Si el dato está disponible en los buffers de la transacción, el valor se modifica en los buffers de la base de datos. Si no se encuentra en los buffers, se envía un comando **fetch** al administrador del buffer y, entonces, se hace la actualización en la base de datos volátil.

Discutiremos ahora la ejecución de las instrucciones restantes de acuerdo a la estrategia que se siga.

No-fix/No-flush

- *Abort*. El administrador del buffer pudo haber escrito algunas páginas actualizadas en la base de datos estable. Por lo tanto, el LRM ejecuta un UNDO de la transacción.
- *Commit*. El LRM escribe un "end_of_transaction" en el registro de la base de datos.
- *Recover*. Para aquellas transacciones que tienen tanto un "begin_transaction" como un "end_of_transaction" en el registro de la base de datos, se inicia una operación parcial REDO por el LRM. Mientras que para aquellas transacciones que tienen solo un "begin_transaction" en el registro, se ejecuta un UNDO global por el LRM.

No-fix/Flush

- *Abort*. El administrador del buffer pudo haber escrito algunas páginas actualizadas en la base de datos estable. Por lo tanto, el LRM ejecuta un UNDO de la transacción.
- *Commit*. El LRM envía un comando **flush** al administrador del buffer para todas las páginas actualizadas. Se escribe un comando "end_of_transaction" en el registro de la base de datos.
- *Recover*. Para aquellas transacciones que tienen tanto un "begin_transaction" como un "end_of_transaction" en el registro de la base de datos, no se requiere una operación REDO. Mientras que para aquellas transacciones que tienen solo un "begin_transaction" en el registro, se ejecuta un UNDO global por el LRM.

Fix/No-flush.

- *Abort*. Ninguna de las páginas actualizadas ha sido escrita en la base de datos estable. Por lo tanto, solo se liberan las páginas correspondientes a la transacción enviándoles un comando **fix**.

- *Commit*. Se escribe un comando "end_of_transaction" en el registro de la base de datos. El LRM envía un comando **unfix** al administrador del buffer para todas las páginas a las que se les envío un comando **fix** previamente.
- *Recover*. Para aquellas transacciones que tienen tanto un "begin_transaction" como un "end_of_transaction" en el registro de la base de datos, se inicia una operación parcial REDO por el LRM. Mientras que para aquellas transacciones que tienen solo un "begin_transaction" en el registro, no se necesita un UNDO global.

Fix/Flush

- *Abort*. Ninguna de las páginas actualizadas ha sido escrita en la base de datos estable. Por lo tanto, solo se liberan las páginas correspondientes a la transacción enviándoles un comando **fix**.
- *Commit*. De manera atómica se tienen que hacer las siguientes acciones. Se envía un comando **flush** al administrador del buffer para todas las páginas a las que se les aplicó un comando **fix**. El LRM envía un comando **unfix** al administrador del buffer para todas las páginas a las que se les aplicó un comando **fix**. El LRM escribe un "end_of_transaction" en el registro de la base de datos.
- *Recover*. No requiere hacer trabajo alguno.

Verificación

La operación de recuperación requiere recorrer todo el registro de la base de datos. Así, el buscar todas las transacciones a las cuales es necesario aplicarles un UNDO o REDO puede tomar una cantidad de trabajo considerable. Para reducir este trabajo se pueden poner puntos de verificación (checkpoints) en el registro de la base de datos para indicar que en esos puntos la base de datos está actualizada y consistente. En este caso, un REDO tiene que iniciar desde un punto de verificación y un UNDO tiene que regresar al punto de verificación más inmediato anterior.

COMUNICACION

En esta sección se considera cuando un nodo falla en la red. El objetivo es desarrollar un protocolo de terminación no-bloqueante que permita desarrollar protocolos de recuperación independiente. En una red puede haber muchas fallas de comunicación debido a colisiones, comunicación intermitente por interferencia, sobrecarga de trabajo, etc. La única forma de suponer que existe una falla de nodo es cuando después de un cierto periodo de tiempo (timeout) no se obtiene respuesta del mismo. Así, la discusión de los protocolos de terminación y recuperación considera el caso en que las fallas de nodo se manifiestan por ausencia de comunicación con éste durante intervalos de tiempo.

Protocolos de terminación

Las fallas de nodo se pueden manifestar tanto en el coordinador como en los participantes.

- **Timeouts del coordinador**

1. *Timeout en el estado WAIT*. El coordinador está esperando por las decisiones locales de los participantes. El coordinador no puede de manera unilateral realizar un commit. Sin embargo, él puede decidir abortar globalmente la transacción. En este caso, escribe un comando abort en el registro de la base de datos y envía un mensaje global-abort a todos los participantes.
2. *Timeout en los estados COMMIT o ABORT*. En este caso el coordinador no esta seguro que los procedimientos de commit o abort se han terminado por los administradores de recuperación local en todos los nodos participantes. Así, el coordinador envía mensajes global-commit o global-abort de manera repetida a todos los nodos que no han respondido y espera por su reconocimiento.

- **Timeouts de los participantes**

1. *Timeout en el estado INITIAL*. En este estado el participante está esperando por un mensaje prepare. El coordinador pudo haber fallado en el estado INITIAL por lo que el participante puede abortar de manera unilateral la transacción. Si el mensaje prepare llega en un tiempo posterior al participante, esto se puede manejar de dos formas. Puede responder con un vote-abort o simplemente ignorar el mensaje y dejar que ocurra el timeout en el lado del coordinador.

2. *Timeout en el estado READY*. En este estado el participante ha votado por realizar un commit pero desconoce la decisión global del coordinador. No se puede de manera unilateral ni hacer un commit ni hacer un abort. Así permanecerá bloqueado hasta que pueda conocer de alguien, el coordinador u otro participante, cual fue la decisión final sobre la transacción.

Protocolos de recuperación

En la parte anterior se discutió como implementar los compromisos de dos fases cuando se presentan fallas en nodos pero desde la perspectiva de los nodos que se mantienen trabajando. En esta parte, se examinará el otro punto de vista, esto es, como un coordinador o participante se pueden recuperar cuando sus nodos fallas y tienen que reiniciar su trabajo.

- **Fallas del coordinador**

1. *El coordinador falla en el estado INITIAL*. Esto es antes de que el coordinador ha iniciado el procedimiento para hacer un commit. Por lo tanto, cuando el coordinador reinicia tiene que empezar el proceso para hacer el commit.
2. *El coordinador falla en el estado de WAIT*. En este caso el coordinador ha enviado el comando prepare y después falla. Para recuperarse de una falla, tendrá que reiniciar el proceso de commit para la transacción desde el envío nuevamente del mensaje prepare.
3. *El coordinador falla en los estados COMMIT o ABORT*. En este caso el coordinador ha tomado una decisión y la ha informado a los participantes. Así, si todos los reconocimientos se han recibido cuando se trata de recuperar, entonces no se hace nada. De otra manera, se invoca al protocolo de terminación.

Fallas de los participantes

1. *Un participante falla en el estado INICIAL*. En este caso se aborta la transacción de manera unilateral cuando se trata de recuperar. Lo anterior es aceptable ya que el coordinador de la transacción debe estar en el estado INITIAL o WAIT. Si está en el primero, enviará un mensaje prepare y pasará al estado de WAIT en donde no recibirá respuesta del participante que ha fallado y eventualmente abortará globalmente la transacción.
2. *Un participante falla en el estado READY*. En este caso el participante ha informado su decisión al coordinador. Cuando se recupere, el participante en el nodo fallido puede tratar esto como un timeout en el estado de READY y manejar la transacción incompleta sobre su protocolo de terminación.
3. *Un participante falla en el estado ABORT o COMMIT*. Esos estados representan las condiciones de terminación, de manera que, cuando se recupere, el participante no necesita tomar acciones especiales.

7.3 Seguridad y Control de concurrencia

7.4 Detección y resolución de inconsistencias

7.5 Puntos de chequeo y restauración en frío

Como se mencionó en la etapa de verificación, la operación de recuperación requiere recorrer todo el registro de la base de datos. Para la reducción de este proceso se pueden poner puntos de verificación (checkpoints) en el registro de la base de datos para indicar que en esos puntos la base de datos está actualizada y consistente. La colocación de puntos de verificación se realiza con las siguientes acciones:

1. Se escribe un "begin_checkpoint" en el registro de la base de datos.
2. Se recolectan todos los datos verificados en la base de datos estable.
3. Se escribe un "fin_de_checkpoint" en el registro de la base de datos.

Restauración en frío

La recuperación a un estado consistente puede ser:

a) Automática: Se establecen algoritmos para detectar quien falla y que hacer para recuperarse de la situación (vistos anteriormente) ejemplos:

1.- Fallos locales a la transacción, que detecta el propio código de la aplicación.
- Son responsabilidad del Programador al dar commit o rollback.

2. Fallos locales a la transacción, que no detecta el propio código de la aplicación.
- El SGBD realiza Rollback de la Transacción.

3. Fallos del sistema que no dañan la BD.
- "Recuperación en caliente“: consulta de la bitácora para deshacer y rehacer Transacciones. REDO –UNDO

b) Manual: El Manejador requiere asistencia para recuperar a un estado consistente a la base de datos a partir de respaldos (se verán a continuación).

4. Fallos del sistema que dañan la BD.
- "Recuperación de emergencia“: Copia de Seguridad +Recuperación en base a la bitácora.

5. Error Fatal (Pérdida de archivo(s) de bitácora)
- "Recuperación en frío“: Copia de Seguridad. Pérdida de Ultimas Transacciones

8 Multibases de Datos

Objetivo: El alumno explicará los diferentes tipos y clasificaciones de base de datos distribuidas.

8.1 Introducción

Un Sistema Multibases de Datos soporta operaciones en múltiples sistemas pre-existentes de bases de datos. Cada sistema de base de datos componente es gestionada por un Sistema Manejador de Bases de Datos (DBMS) componente. Un sistema de base de datos componente puede estar centralizado o distribuido y puede residir en la misma computadora o en varias computadoras conectadas por un sistema de comunicación Sheth et.al (1990).
Un Sistema Multibases de datos permite operaciones de acceso y modificación de datos entre diversos sistemas de bases de datos de manera concurrente y transaccional.
La diferencia entre las bases de datos distribuidas y los sistemas Multibases de datos es la autonomía local. No se puede forzar a los esquemas locales pre-existentes a que se ajusten a un esquema estándar.

8.2 Clasificación por Autonomía de Multibases de Datos

La autonomía local en los sistemas de múltiples bases de datos es que cada sistema manejador de base de datos retiene el control completo sobre los datos locales y su procesamiento. En este caso, ni los cambios globales, como la adición de otros componentes tienen efecto en los sistemas de base de datos locales.
Si existe completa autonomía local, entonces los sistemas Multibases de datos proporcionan un servicio de solo lectura, porque los sistemas manejadores de base de datos no colaboran entre sí para procesar transacciones distribuidas.

En el caso de poca o nula autonomía, existe una sola imagen de la base de datos completa disponible a todos los usuarios. Todos los sitios trabajan juntos para completar una transacción.
Se puede realizar la propagación de actualizaciones a lo largo de las bases de datos componentes con datos semánticamente equivalentes a través de un control a nivel distribuido y
local por manejo de transacciones y control de concurrencia.

En el caso de semi autonomía, los sistemas de base de datos pueden operar independientemente, pero han decidido participar en una federación para compartir sus datos locales con otros sistemas de base de datos.

El tipo de autonomía de los sistemas multibase de bases de datos determinará si pueden formar un Sistema Federado cuando éstos se integran para compartir información.

8.2.1 Sistemas Federados

Los Sistemas de Base de Datos Federados se forman a partir de un conjunto de sistemas cooperativos pero autónomos de base de datos que se unen para poder compartir e intercambiar información. Una federación se refiere a un conjunto de base de datos que constituyen una base de datos federada.

Los Sistemas Federados también son conocidos como Sistemas Multibases de datos autónomos y heterogéneos.

8.2.2 Sistemas No federados

Los Sistemas de Bases de Datos no Federados resultan de la integración de sistemas gestores de base de datos que no son autónomos. Todos los sistemas de base de datos están completamente integrados a un solo esquema global. Los sistemas Multibases de datos unificados lógicamente se parecen a un Sistema de Base de Datos Distribuido.

8.3 Clasificación por Variedad de Sistemas de Manejadores Componentes

8.3.1 Multibases de datos homogéneas

Sistema multibase de datos homogéneo es aquel en donde el sistema manejador de base de datos de todas las bases de datos componentes es el mismo.

8.3.2 Multibases de datos heterogéneas

Por tanto, cuando los manejadores de las bases de datos componentes son diferentes se trata de un Sistema multibase de datos heterogéneo. Existen dos casos de heterogeneidad a nivel Manejador de base de datos:

a) El Sistema Multibase de datos está compuesto por diferentes Manejadores de base de Datos, pero pertenecen al mismo modelo de datos. Por ejemplo, Sybase y Oracle ofrecen Manejadores de Bases de Datos Relacionales.
b) El Sistema Multibases de datos está compuesto por diferentes Manejadores de Base de Datos pertenecientes a diferentes modelos de datos, como el relacional y el objeto-relacional con Sybase Adaptive Server Enterprise y Versant Object Database por ejemplo.

Tipos de heterogeneidad

Los tipos de heterogeneidad a nivel sistemas de base de datos pueden dividirse en aquellos derivados de las diferencias en los Manejadores de Base de Datos y aquellos derivados de las diferencias en la semántica de los datos.
Heterogeneidades por Diferentes Manejadores de Datos

a) Componentes del Modelo de datos: Estructura, restricciones, lenguajes de consulta
b) Implementación propia del manejador: manejo de transacciones, control de concurrencia, protocolos de respaldo y recuperación, etc.
c) Diseño lógico de base de datos

Heterogeneidad semántica

d) Homónimos, sinónimos, conflictos de dominio, manejo de información faltante, conflictos de valores por omisión.

RESUMEN:

Una base de datos distribuida almacena datos relacionados lógicamente en dos o más sitios físicamente independientes conectados vía una red de computadoras. La base de datos es dividida en fragmentos, los cuales pueden ser horizontal (por registros) o vertical (conjuntos de atributos). Cada fragmento puede estar alojado en un nodo diferente en la red.

El procesamiento distribuido se refiere a la división lógica del procesamiento de una base de datos a lo largo de dos o más nodos de red. Las bases de datos distribuidos requieren procesamiento distribuido.

Un Sistema Manejador de base de datos distribuida gobierna el procesamiento y almacenamiento de datos relacionados vía un sistema de computadoras interconectadas.

Los principales componentes de un DDBMS son el procesador de transacciones TP y el procesador de datos DP. El primero es el software que reside en un nodo de computadora que requiere datos. El procesador de datos es el software que reside en cada computadora que almacena y obtiene datos.

Los sistemas de base de datos actuales pueden ser clasificados por su soporte a la distribución de datos y procesamiento.

a) Procesamiento y Datos en un solo sitio. (SPSD)
b) Procesamiento multi-sitio, datos en un solo sitio (MPSD)
c) Procesamiento multi-sitio, datos en múltiples sitios (MPMD)

Un sistema de base de datos distribuido homogéneo integra solamente un tipo de manejador de base de datos en la red.

Un sistema de base de datos distribuido heterogéneo integra diferentes tipos de manejadores de base de datos en la red.

Conjunto de transparencias que describen a un DDBMS y que tienen como objetivo hacer que una base de datos distribuida se comporte como si fuera una base de datos centralizada.

a) Transparencia de distribución: Permite manejar datos distribuidos como si estuvieran centralizados, existen tres niveles de transparencia
    i. Fragmentación
    ii. Localización
    iii. Mapeo local
b) Transparencia de transacción: Asegura que la ejecución de una transacción distribuida mantenga la integridad y la consistencia de la base de datos distribuida.
c) Transparencia a fallos: Asegura la operación continua de los DDBMS aun en el caso de falla en uno o más nodos en la red.
d) Transparencia a heterogeneidad: Asegura que los DDBMS será capaces de integrar diferentes tipos de Manejadores de base de datos en un solo esquema común o global.
e) Transparencia en rendimiento: Asegura que el sistema se comportara como un sistema centralizado y que el rendimiento del sistema no se verá afectado significativamente por la distribución de los datos en diversos sitios.

Una transacción se forma por uno o más requerimientos de base de datos. Una transacción no distribuida actualiza u obtiene datos de un solo sitio. Una transacción distribuida puede actualizar o consultas datos ubicados en diversos sitios.

El control de concurrencia distribuido se requiere en una red de base de datos distribuidas. El protocolo de compromiso de dos gases se usa para asegurar que todas las partes de la transacción se completan.

Un DBMS distribuido evalúa todos los requerimientos de datos para encontrar la ruta de acceso óptima en una base de datos distribuida. El DDBMS debe optimizar la consulta para reducir costos de acceso. CPU y red asociados a la consulta.

El diseño de una base de datos debe tomar en consideración la fragmentación y replicación de datos. Como alojar cada uno de los fragmentos o replicas para poder obtener el mejor tiempo de respuesta y asegurar disponibilidad de datos al usuario final.

Bibliografía básica:

ROB, Peter; CORONEL, Carlos (todos los temas)
Sistemas de bases de datos
México
Thomson, 2004

CONOLLY, Thomas M; BEGG, Carolyn E. (todos los temas)
Sistemas de bases de datos: Un enfoque practico para diseño, implementación y gestión
Editorial Pearson, 4ª Edición, Madrid, 2005.
ISBN: 8478290753

Bibliografía complementaria:

CERI, Estefano; PELAGATT, Giuseppe Todos
Distributed databases (principles & systems)
New York
Mc Graw Hill, 1985}