**SELF-STUDY 1: Fragmentación**

**4. Distributed Database Design**

Este documento discute el diseño de bases de datos distribuidas, centrándose en la fragmentación de datos como un aspecto crucial. La fragmentación de datos implica descomponer relaciones grandes en piezas más pequeñas para mejorar la eficiencia y el rendimiento de un sistema de base de datos distribuido. Aborda el desafío de cómo asignar datos a diferentes sitios para optimizar varios parámetros, como minimizar el consumo de recursos para el procesamiento de consultas. Se tratan dos problemas principales en este contexto: la fragmentación de datos y la asignación de datos.

**4.1 Fragmentación de Datos:**

Razones para la Fragmentación de Datos:

La fragmentación de datos es esencial porque una relación no siempre es una unidad de procesamiento eficiente. Los usuarios y las aplicaciones suelen acceder solo a un subconjunto de las relaciones disponibles en la base de datos. Sin fragmentación de datos, existen dos opciones principales: colocar toda la relación en un solo nodo, aumentando los accesos remotos y los posibles cuellos de botella, o replicarla en cada nodo donde pueda ser necesaria. Sin embargo, optimizar la localidad de los datos es un enfoque mejor, donde los fragmentos que probablemente se utilicen juntos se colocan juntos para minimizar la sobrecarga de comunicación y la replicación innecesaria.

Beneficios de la Fragmentación de Datos:

Localidad de Datos: La fragmentación permite que se coloquen juntos los datos relacionados, reduciendo la sobrecarga de comunicación en la red y la replicación innecesaria.

Concurrencia y Paralelismo: Los datos fragmentados permiten que diferentes transacciones se ejecuten de manera concurrente y mejoran el rendimiento del sistema, facilitando el paralelismo, que es vital para manejar bases de datos muy grandes.

Desafíos de la Fragmentación de Datos:

Impacto en el Rendimiento: Fragmentar una relación puede conducir a un rendimiento deficiente cuando es necesario recuperar y manipular múltiples fragmentos durante el procesamiento de consultas, especialmente cuando existen requisitos conflictivos.

Verificación de la Integridad: Dividir atributos con dependencias entre fragmentos puede complicar la verificación de la integridad, aumentando la sobrecarga de comunicación entre nodos.

Tipos de Fragmentación de Datos:

Existen dos enfoques principales para la fragmentación de datos: horizontal y vertical.

Fragmentación Horizontal: En este enfoque, se utiliza un predicado de selección para crear diferentes fragmentos basados en valores de atributos. Cada tupla se coloca en el fragmento correspondiente. Por ejemplo, puede fragmentar la relación de proyectos por ciudad, lo que resulta en un fragmento para cada ciudad.

Fragmentación Vertical: La fragmentación vertical implica crear diferentes proyecciones de la relación, donde cada fragmento contiene un conjunto diferente de atributos. Por ejemplo, puede almacenar ciertos atributos en cada nodo y el resto en una ubicación central.

Fragmentación Híbrida: Este enfoque combina tanto la fragmentación horizontal como la vertical para fragmentar aún más los fragmentos producidos por estrategias anteriores. Permite un control más detallado sobre la distribución de datos.

Propiedades de una Fragmentación Correcta:

Una fragmentación se considera correcta si satisface tres propiedades:

Completitud: Cada elemento de datos de la relación original se puede encontrar en al menos un fragmento, garantizando que los datos no se pierdan durante la fragmentación.

Disyunción: Los datos colocados en un fragmento no deben aparecer en ningún otro fragmento para evitar redundancias. La replicación de datos se considera por separado durante la asignación.

Reconstrucción: Siempre se puede reconstruir la relación original a partir de los fragmentos utilizando operadores algebraicos relacionales.

Fragmentación Híbrida: Esta combinación anidada de estrategias de fragmentación se considera correcta si todas las estrategias de fragmentación subsiguientes aplicadas son correctas.

En resumen, la fragmentación de datos es un paso crucial en el diseño de bases de datos distribuidas. Su objetivo es optimizar el rendimiento, la concurrencia y el paralelismo mientras se mantiene la integridad de los datos. La fragmentación horizontal y vertical son las estrategias principales, y la fragmentación híbrida combina estos enfoques para una distribución de datos más matizada. Una fragmentación correcta garantiza la completitud, la disyunción y la reconstrucción de los datos.

**4.1.1. Horizontal Fragmentation**

El texto se centra en la fragmentación horizontal de datos en una base de datos distribuida. La fragmentación horizontal implica dividir una relación en función de sus tuplas. Cada fragmento se define mediante predicados (es decir, selecciones sobre cualquier atributo de la relación). El texto ilustra una fragmentación horizontal para la relación de proyectos en un ejemplo en curso. Cada fragmento contiene un subconjunto de las tuplas de la relación.

Formalmente, una relación R se fragmenta horizontalmente en n fragmentos mediante un predicado de fragmentación Ri, donde 1 ≤ i ≤ n y Fi es el predicado de fragmentación que define el fragmento Ri. Típicamente, cada fragmento se representa mediante el predicado utilizado para crearlo. En el ejemplo proporcionado: HF1: city = Barcelona, HF2: city = Hong Kong, HF3: city = Portland, HF4: city = Frankfurt y HF5: city = Glasgow.

El texto luego aborda cuándo un administrador de bases de datos (DBA) debe realizar una fragmentación horizontal en una relación dada. En general, un sistema distribuido se beneficia de la fragmentación horizontal cuando necesita reflejar datos geográficamente distribuidos (cada nodo accede principalmente a datos relacionados con sí mismo), facilitar la recuperación y el paralelismo, reducir la profundidad de los índices (ya que cada fragmento tiene su propio índice, el número de índices aumenta pero su tamaño se reduce) y reducir la contención. En el ejemplo proporcionado, cada fragmento es obviamente más pequeño que la relación completa (siempre que se sepa que hay al menos una tupla en cada nodo). Por lo tanto, un índice sobre la clave principal (pno) resultaría en cinco índices diferentes (uno por fragmento), que son más pequeños en tamaño (ya que cada índice solo contendrá entradas para las tuplas en ese fragmento). La contención se reduce claramente, ya que varios usuarios mutuamente excluyentes, que trabajan en nodos diferentes, pueden acceder a diferentes fragmentos simultáneamente y no causarán conflictos. Además, las consultas sobre toda la relación se pueden resolver mediante paralelismo.

Luego, se discute hasta qué punto se debe fragmentar. Se menciona que la fragmentación puede variar desde un extremo (sin fragmentación en absoluto) hasta el otro (colocando cada tupla en un nodo diferente). Se destaca la importancia de identificar qué predicados (sobre qué atributos) son de interés en la base de datos. Se sugiere que se debe prestar atención a los usuarios o aplicaciones más activos, ya que suelen generar la mayoría de los accesos a la base de datos. Para cada uno de estos predicados, se describen los pasos a seguir, que incluyen recopilar todas las cláusulas simples utilizadas en cualquier predicado de consulta relacionado con la relación dada, completar las cláusulas faltantes de acuerdo con la semántica del operador y determinar conjuntos de cláusulas relevantes.

**Pasos a seguir:**

La conclusión de que la fragmentación ilustrada en la Tabla 2 es la más apropiada se basa en el proceso de análisis de los predicados de fragmentación y sus características en el contexto del sistema distribuido. El texto proporciona un ejemplo práctico paso a paso de cómo se llega a esta conclusión. Aquí está el proceso resumido:

Recopilación de Predicados: Se recopilan todos los predicados utilizados en las consultas emitidas por los usuarios más activos. En el ejemplo, se obtienen los siguientes predicados: city = 'Barcelona', city = 'Glasgow', city = 'Frankfurt' y budget > 10000.

Completar Predicados Faltantes: Se completan los predicados faltantes según la semántica de los operadores utilizados. Por ejemplo, se agregan predicados para cada valor en el dominio de city (en este caso, city = 'Portland' y city = 'Hong Kong') y se completa el rango de valores para budget (en este caso, budget <= 10000).

Determinación de la Relevancia: Se evalúa si cada predicado es relevante para al menos un nodo en el sistema distribuido. En este caso, dado que la organización está geográficamente distribuida, todos los predicados relacionados con el atributo city son relevantes. Sin embargo, los predicados relacionados con el atributo budget solo son relevantes para el nodo Glasgow.

Generación de Combinaciones: Se generan todas las combinaciones posibles de predicados. En este ejemplo, solo quedan predicados relacionados con el atributo city, ya que los relacionados con budget se descartaron debido a su falta de relevancia para la mayoría de los nodos.

Eliminación de Predicados Semánticamente Inconsistentes: Se eliminan los predicados complejos que son semánticamente inconsistentes. Por ejemplo, si hubiera un predicado contradictorio como city = 'Barcelona' AND country = 'USA', se eliminaría.

Selección de los Predicados de Fragmentación: Finalmente, entre los predicados restantes, se seleccionan los predicados de fragmentación. En este caso, los predicados relacionados con city son seleccionados, ya que son los únicos relevantes y coherentes.

Por lo tanto, la conclusión de que la fragmentación horizontal basada en los predicados de city es la más apropiada se deriva de un proceso de análisis sistemático que tiene en cuenta la relevancia, la coherencia y la eficiencia en el contexto de la organización y las consultas de los usuarios más activos. Esta fragmentación se considera apropiada porque refleja la estructura geográfica de la organización y permite un acceso eficiente a los datos relacionados con las ciudades.

Finalmente, se presenta un ejemplo práctico en el que se aplican estos pasos al proyecto de relación, y se llega a la conclusión de que la fragmentación ilustrada en la Tabla 2 es la más apropiada. El texto concluye enfatizando que una fragmentación horizontal debe cumplir con tres propiedades: completitud, disyunción y reconstrucción. La fragmentación propuesta en el ejemplo satisface estas propiedades, ya que los predicados de fragmentación son completos, mutuamente excluyentes y la relación original se puede reconstruir a través del operador de unión.

**4.1.2. Derived Horizontal Fragmentation**

El texto trata sobre la fragmentación horizontal derivada en el contexto de bases de datos distribuidas. La fragmentación horizontal derivada es un enfoque que considera las relaciones entre diferentes tablas (relaciones) en una base de datos. Aquí está la explicación paso a paso del texto:

* Fragmentación Horizontal Primaria: El texto menciona que la fragmentación horizontal primaria se refiere a la fragmentación de una sola relación (tabla) a la vez. En este enfoque, se divide una tabla en fragmentos más pequeños en función de algún criterio, como los valores de un atributo. Por ejemplo, podríamos fragmentar la tabla de proyectos en fragmentos basados en la ciudad.
* Relaciones y Dependencias: Sin embargo, en una base de datos, las tablas suelen estar relacionadas entre sí a través de claves foráneas (foreign keys) y claves primarias (primary keys). Estas relaciones son fundamentales para consultar la base de datos. Por ejemplo, una relación R podría estar relacionada con una relación S a través de una relación de "muchos a uno" (many-to-one), donde no se permiten valores NULL en el extremo de S. Esto se implementa típicamente como una restricción de clave foránea y clave primaria.
* Uso de Relaciones en Consultas: El texto señala que, en muchas ocasiones, las relaciones se consultan juntas al realizar una unión (join) utilizando las claves primarias y foráneas. Por lo tanto, para optimizar el rendimiento, es beneficioso colocar estas relaciones relacionadas en el mismo nodo en un sistema de base de datos distribuida.
* Propietario y Relación Dependiente: Para aplicar esta estrategia, se debe identificar una relación propietaria (owner) y una relación dependiente (member). Estos términos son cuestiones de notación para resaltar el hecho de que una relación dependiente está de alguna manera vinculada a una relación propietaria. El propietario decide cómo fragmentar la relación dependiente.
* Ejemplo: Se proporciona un ejemplo en el que se considera la fragmentación horizontal de las relaciones de "assigned" y "employee" con respecto a la relación de "project". En este caso, "project" es el propietario y "assigned" es el miembro. Las relaciones se unen a través de la relación entre las claves primarias y foráneas (pno - projectNo y id - employeeId).
* Cuándo Utilizar Fragmentación Horizontal Derivada: La fragmentación horizontal derivada es útil cuando las relaciones propietarias necesitan combinarse con fragmentos de relaciones dependientes mediante claves de unión coincidentes. En otras palabras, cuando la relación miembro depende claramente de la relación propietaria según las consultas de la base de datos.
* Definición de Fragmentación Horizontal Derivada: La fragmentación horizontal derivada se define como la creación de fragmentos de una relación dependiente R en relación con una relación propietaria S mediante una relación r. La definición se expresa como: Ri = R ⨝ Si, donde ⨝ representa una semijoin. El resultado de este semijoin son las tuplas en R para las cuales hay al menos una tupla en Si con una clave de unión coincidente (se consideran como claves de unión los atributos de R y S en r).
* Múltiples Relaciones de Dependencia: En algunos casos, las relaciones propietarias y dependientes pueden estar relacionadas por varias relaciones. En tales casos, se deben considerar criterios adicionales para decidir cuál de las relaciones de dependencia se utilizará para la fragmentación. Esto puede incluir la frecuencia de las consultas y la optimización del procesamiento de consultas distribuidas y el paralelismo.
* Requisitos Adicionales para la Fragmentación Horizontal Derivada: Además de las propiedades discutidas previamente para la fragmentación horizontal (completitud, disyunción y reconstrucción), la fragmentación horizontal derivada debe cumplir con dos requisitos adicionales:
* Completitud: La relación utilizada para realizar el semijoin debe aplicar una restricción de integridad referencial.
* Disyunción: El atributo de unión debe ser la clave del propietario.

En resumen, la fragmentación horizontal derivada es un enfoque que considera las relaciones entre tablas en una base de datos distribuida. Su objetivo es optimizar la colocación de tablas relacionadas en los mismos nodos para mejorar el rendimiento.

**4.1.3. Vertical Fragmentation**

El texto se refiere a la fragmentación vertical en el contexto de bases de datos distribuidas. La fragmentación vertical es un enfoque en el que una tabla se divide en fragmentos más pequeños mediante la proyección de ciertos atributos de la relación en cada fragmento. Aquí está una explicación paso a paso del texto:

* Fragmentación Vertical: La fragmentación vertical implica dividir una tabla en subconjuntos más pequeños proyectando algunos de sus atributos en cada fragmento. Cada fragmento contiene un subconjunto de los atributos de la relación, pero todos ellos contienen la clave primaria (primary key) de la tabla. La clave primaria se replica en cada fragmento, y la justificación de esta replicación se discutirá más adelante en el texto.
* Definición Formal de la Fragmentación Vertical: Formalmente, una relación R se fragmenta verticalmente en n fragmentos mediante proyecciones: VPij, donde 1 ≤ j, k ≤ m. Aquí, m es el número de atributos en R y PK es la clave primaria de R.
* Determinación de la Necesidad de Fragmentación Vertical: El texto comienza abordando la cuestión de cuándo es apropiada la fragmentación vertical. Históricamente, la fragmentación vertical no se utilizaba ampliamente en sistemas transaccionales debido a que a menudo empeoraba los tiempos de inserción y actualización. Sin embargo, con la llegada de sistemas de consulta (como sistemas de toma de decisiones en los que los usuarios solo consultan datos), la fragmentación vertical se convirtió en una alternativa poderosa para reducir la cantidad de atributos que se deben leer de una tabla.
* Beneficios de la Fragmentación Vertical: La fragmentación vertical es beneficiosa cuando solo se requiere un subconjunto de atributos de una tabla en consultas específicas. Esto reduce significativamente la cantidad de datos irrelevantes que se leen, mejorando así el rendimiento. También reduce la contención en el sistema, facilita la recuperación de datos y permite un mayor paralelismo en consultas. Sin embargo, tiene desventajas, como el aumento en la cantidad de índices y un potencial empeoramiento en los tiempos de inserción y actualización.
* Agrupación de Atributos: La decisión de cómo agrupar los atributos en cada fragmento no es sencilla. Para abordar esto, se pueden utilizar enfoques como la agrupación de atributos con alta afinidad. Esto implica analizar consultas relevantes en el sistema y calcular la afinidad entre cada par de atributos. La afinidad se mide según cuántas veces dos atributos aparecen juntos en las consultas ponderadas por su frecuencia.
* Ejemplo de Afinidad de Atributos: El texto proporciona un ejemplo con una relación "project" y consultas relevantes. Se calcula la afinidad entre atributos y se crea una matriz de afinidad. Por ejemplo, si "pno" y "city" aparecen juntos con alta frecuencia, se agruparían en el mismo fragmento.
* Propiedades de los Fragmentos Verticales: Al igual que con cualquier otro tipo de fragmentación, los fragmentos verticales deben cumplir con ciertas propiedades:
* Completitud: La unión de los atributos proyectados en cada fragmento debe producir la relación original.
* Disyunción: Cada atributo (excepto la clave primaria) solo puede utilizarse para crear un fragmento. En otras palabras, cada atributo aparece en un solo fragmento.
* Reconstrucción: Para garantizar la reconstrucción de la relación original, la clave primaria se replica en cada fragmento. Esto permite unir los fragmentos mediante la clave primaria replicada.
* Ejemplo de Fragmentación Vertical: En el ejemplo proporcionado, se crean dos fragmentos verticales, VP1 y VP2, que contienen diferentes conjuntos de atributos de la relación "project". Estos fragmentos cumplen con las propiedades de completitud, disyunción y reconstrucción, y la clave primaria "pno" se replica en cada fragmento.

Este ejemplo se centra en la relación "project" y cómo se pueden crear fragmentos verticales a partir de esta relación.

Primero, recordemos que la fragmentación vertical implica dividir una tabla en subconjuntos más pequeños mediante la proyección de atributos específicos en cada fragmento. En este caso, estamos trabajando con la relación "project," que puede tener múltiples atributos.

El ejemplo propone crear dos fragmentos verticales, llamados VP1 y VP2, para la relación "project." Aquí hay una descripción más detallada de cada fragmento:

VP1: Este fragmento contiene los siguientes atributos de la relación "project":

* pno (primary key)
* name
* city
* country

VP2: El segundo fragmento, VP2, contiene los siguientes atributos de la relación "project":

* pno (primary key)
* budget
* category
* income
* productivityRatio

Ahora, veamos cómo se justifica esta fragmentación:

* Compleción: Ambos fragmentos, VP1 y VP2, juntos cubren todos los atributos de la relación "project." Esto significa que al unir VP1 y VP2, obtendrás todos los atributos originales de la relación. Esto cumple con la propiedad de completitud.
* Disyunción: Cada atributo (excepto la clave primaria "pno") aparece en solo uno de los fragmentos. Por ejemplo, "name," "city," y "country" aparecen en VP1, mientras que "budget," "category," "income," y "productivityRatio" aparecen en VP2. Esto garantiza que no haya redundancia de atributos en diferentes fragmentos, cumpliendo así con la propiedad de disyunción.
* Reconstrucción: Para asegurar que los fragmentos se puedan reconstruir en la relación original, se replica la clave primaria "pno" en cada fragmento (tanto en VP1 como en VP2). Esto es fundamental porque permite realizar uniones (joins) entre los fragmentos utilizando la clave primaria para recuperar la relación original. La propiedad de reconstrucción se cumple, ya que puedes combinar VP1 y VP2 mediante una operación de unión (join) en la clave primaria "pno" para obtener la relación "project" completa.

En resumen, este ejemplo de fragmentación vertical divide la relación "project" en dos fragmentos, cada uno conteniendo un conjunto diferente de atributos. Estos fragmentos cumplen con las propiedades de completitud, disyunción y reconstrucción, y la clave primaria se replica en cada fragmento para garantizar la reconstrucción adecuada de la relación original. Esto permite optimizar el rendimiento de las consultas al reducir la cantidad de atributos que se deben leer en cada consulta.

En resumen, la fragmentación vertical es un enfoque que divide una tabla en fragmentos basados en atributos para optimizar el rendimiento de consultas al reducir la cantidad de datos irrelevantes que se leen. Esto es especialmente útil en sistemas de consulta donde los usuarios solo necesitan ciertos atributos en lugar de la tabla completa.