VNIVERSIDAD NACIONAL
AVTNOMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

ESTUDIO Y EXPERIMENTACIÓN DE LOS FACTORES QUE INCIDEN EN EL DESEMPEÑO DE OPERACIONES COMPLEJAS DENTRO DE UN SISTEMA MANEJADOR DE BASES DE DATOS

TESIS

QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN INGENIERÍA (COMPUTACIÓN)

PRESENTA:
ISAIAS ZUÑIGA JOSE

TUTOR:
DR. JAVIER GARCÍA GARCÍA
FACULTAD DE CIENCIAS
UNAM

MÉXICO, D. F. JUNIO 2014

Resumen

Este documento recopila los resultados obtenidos en el trabajo de investigación, los resultados consisten en los factores que inciden en el desempeño de operaciones complejas dentro de un sistema manejador de bases de datos. Para ello se plantearon metas a alcanzar así como las delimitaciones correspondientes. Se desarrolló primeramente una recopilación bibliográfica que respalda la base teórica del trabajo a realizar, se recurrió tanto a los libros más citados en el área de base de datos y específicamente sobre desempeño de operaciones realizadas dentro de un sistema manejador de bases de datos, así como a la búsqueda en artículos de investigación cuyos objetivos están en correspondencia con lo que aquí se plantea.

Posteriormente se realizó un trabajo experimental en el que se analizó el desempeño de operaciones desde diferentes perspectivas como por ejemplo: experimentar con los niveles de aislamiento, pruebas con escalamiento de memoria, pruebas con reuniones (JOIN) anidadas, etc. En esa parte se muestran los resultados logrados, los cuales algunos unos fueron los esperados de acuerdo a las fuentes consultadas, mientras que otros constituyeron hallazgos originales como por ejemplo la experimentación que se realizó con la memoria caché.

Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Planteamiento del problema

En la actualidad hay una necesidad de almacenar y recuperar los datos que se producen continuamente, esta generación de datos se da tanto en el ámbito comercial, académico, industrial, entretenimiento. Estos datos alcanzan tamaños que van desde el orden de los kilobytes llegando a crecer hasta los petabytes de almacenamiento con los consecuentes costos de recuperación, mantenimiento[Borthakur13].

Para cumplir con la demanda se deben conocer y solventar las dificultades que se presentan, entre las que se encuentran: lecturas repetidas las cuales incrementan la carga trabajo del CPU cuando se ejecutan pasos redundantes en las operaciones, cuellos de botella generados al enviar grandes cantidades de datos a memoria, incremento de lecturas a disco, etc. Por lo anterior, se debe conocer a detalle el funcionamiento de los algoritmos de exploración de grandes tablas de datos. Estos algoritmos incluyen tareas de extracción o de traslado de datos desde dispositivos de almacenamiento de memoria secundaria, como por ejemplo discos magnéticos o de estado sólido hasta la memoria primaria.

Por otro lado la utilización concurrente de las bases de datos en grandes proporciones, es el común denominador en la operación de los sistemas de bases de datos de alcance global. Por lo cual se requiere analizar el desempeño de las consultas tan-

to grupalmente como de forma individual[Duggan11]. Este aspecto es cada vez más recurrente pues cada día hay una mayor integración de la población con la tecnología. Esto genera enormes cantidades de datos que se traducen en una utilización intensiva de bases de datos tanto en almacenamiento, recuperación, actualizaciones y explotación para toma de decisiones.

Aunque el lenguaje de consulta SQL es un estándar, en cada sistema manejador de bases de datos (SMBD) se encuentra implementado de manera diferente. De forma significativa, cada SMBD tiene características únicas, las cuales no son estándar como por ejemplo escaneo reverso de índices, restricciones de información, etc.[Teorey11]. Se requiere, por lo tanto, evaluar las operaciones en distintos SMBD existentes[1], así como conocer las variables que se pueden manipular, esto con el objetivo de detectar cual es la combinación de estas variables que lleva a un desempeño óptimo. Entendemos por variables tanto las inherentes al SMBD, así como otras ligadas al hardware y al software con el cual se trabaja.

[1]En este trabajo se evaluará el comportamiento de operaciones entre SMBDs, pero no se pretende evaluar el desempeño de un SMBD en particular contra otro.

1.2. Objetivos de la Tesis

1.2.1. Objetivo general

Conocer los factores inciden en el desempeño de operaciones complejas en un sistema manejador de bases de datos a través de diferentes enfoques, los cuales incluyen aspectos relacionados con el hardware, aspectos relacionados con variables inherentes a los SMBDs, como por ejemplo: almacenamiento, velocidad de procesamiento, soporte a transacciones concurrentes, entre otros.

1.2.2. Objetivos particulares

- Estudiar y experimentar de los factores que inciden en el desempeño de operaciones complejas utilizando el lenguaje SQL dentro de un sistema manejador de bases de datos.
- Determinar la forma más eficiente para expresar sentencias en diferentes SMBDs.
- Analizar el comportamiento de consultas en SQL para diferentes niveles de aislamiento y determinar las condiciones para un mejor desempeño de las mismas.
- Conocer el desempeño de operaciones que involucren el cálculo de funciones de agregación(sum(), avg(),min()).
- Conocer el desempeño de sentencias SQL de manipulación de datos como son las operaciones inserción (insert) y modificación (update),
- Analizar el comportamiento de consultas SQL y su comportamiento concurrente, considerando diferentes niveles de aislamiento.
- Realizar pruebas de escalabilidad y aceleración y medir el desempeño de las consultas al variar la cantidad de memoria y cantidad de procesadores disponibles.
- Analizar el comportamiento de sentencias SQL cuando se inhibe la memoria caché.

1.2.3. Alcances y limitaciones

Este trabajo se abordará dentro de los límites que a continuación se mencionan.

Se toma el desempeño obteniendo el tiempo de ejecución, para ello se usan segundos como unidad de medida llegando hasta las décimas de segundo.

Se utilizará una versión comercial de hardware(que en el capítulo de Experimentación se especifica), es decir no se usan servidores de alto desempeño, el software requerido será el de libre distribución explotándolo hasta donde las licencias del mismo lo permitan.

En las pruebas de escalabilidad como más adelante se detalla, se utilizarán datos generados de forma sintética y el tamaño de la base de datos será en el orden de gigabytes.

Para determinar los factores que deben ser sometidos a nuestro análisis, se hará una revisión bibliográfica y a partir de este estudio se realizará trabajo experimental.

1.3. Trabajos relacionados

Para la realización de este trabajo de tesis se realizó primeramente una búsqueda de los trabajos relacionados en lo referente a optimización de consultas, experimentación con bases de datos extensas, análisis realizados en cuanto a desempeño de operadores, entre otros estudios. Como primer paso se tiene una base de artículos de investigación en los cuales han publicado hasta la fecha diversos resultados. En [Chaudhuri98] Surajit Chaudhuri: "An overview of query optimization in relational systems" se exponen fundamentos de eficiencia en ejecución de consultas así como ejemplos de costo y estimación, utilizando SMBDs tradicionales y usando estrategias de ejecución en sistemas tanto distribuidos como en paralelo.

Un elemento que motivó principalmente este trabajo es el artículo [Ordonez10] Carlos Ordoñez and Javier García-García "Evaluating JOIN performance on relational database systems" aquí se estudió ampliamente el desempeño del operador JOIN desde diferentes perspectivas y para gran cantidad de registros en las tablas, tomando en cuenta restricciones de integridad referencial, ello se llevó a cabo en tres diferentes SMBDs relacionales; en los experimentos que se obtuvieron se utilizó el tiempo como medida para clasificar el desempeño de las perspectivas analizadas. Entre los resultados obtenidos en esa investigación se muestra que son más eficientes los hash-joins para tratar los nulos e inválidos que los sort-merge JOINs; recomienda utilizar un índice secundario para columnas de llave foránea con valores inválidos y/o nulos, también la creación de vistas ya que si la tabla tiene gran cantidad de estos valores la inclusión de esta vista ayudará a acelerar la ejecución.

En este mismo trabajo se muestran los resultados del desempeño del operador JOIN bajo condiciones de llaves nulas, válidas e inválidas. Entre los resultados interesantes se encontró que el número de filas con valores nulos o con llaves no válidas no influye en el plan de ejecución, incluso se encontró que los valores inválidos de llave no influyen en el desempeño del JOIN. En los experimentos que ahí se realizaron, se encontró que la utilización de un segundo índice trae consigo una aceleración.

Otra referencia fundamental que se tomó es [Graefe93] "Query evaluation tech-

niques for large databases". En este trabajo los autores tuvieron como objetivos los siguientes: hacer experimentos con respecto al comportamiento de consultas con gran cantidad de datos, se dan a conocer una serie de elementos para el diseño, implementación de consultas en SMBDs así como técnicas de evaluación de consultas, evaluación de planes de ejecución complejos y ejecución paralela de algoritmos de consulta.

Donald Kossmann [Kossmann00] explica la arquitectura del procesamiento de consultas y muestra una serie de técnicas de ejecución de consultas básicas, las cuales son utilizadas en sistemas de bases de datos distribuidos; entre esas técnicas menciona las referentes a JOIN, al aprovechamiento de paralelismo, técnicas para reducir los costos de comunicación, así como explotar el cacheo y replicación de los datos. Se destaca en el artículo cómo beneficiarse de las capacidades de las consultas en entornos que nos son cliente servidor con un sistema heterogéneo[2]

En [Ioannidis96] el autor logró un mejoramiento en el desempeño de ejecución de consultas complejas re-optimizando los planes de ejecución, a través de una estrategia estadística que permite la reubicación eficiente de recursos compartidos de las tablas intermedias. Presenta un resumen de la arquitectura de un optimizador de consultas, enfatiza las características recurrentes en los SMBDs más usados, también presenta un panorama de problemas avanzados en optimización de consultas, a los cuales no se les ha encontrado una solución.

Un trabajo que trata la optimización en un SMBD comercial[Shankar12], describe la forma en la que está implementado el optimizador de consultas en este gestor en particular en la variante "servidor de almacén de datos paralelo" implementa un optimizador basado en costo para ejecución de consultas distribuidas, no se paraleliza el mejor plan encontrado si no que considera más alternativas, eligiendo alguno basado en costo para el entorno de ejecución distribuida. Se describe que su optimizador de consultas no se queda solo en un reordenamiento de consultas sino que implementa técnicas tales como detección de contradicciones, eliminación de JOINs redundantes, subconsultas no anidadas, re-ordenamiento de JOINs externos.

[2]Sistemas que están conformados de muchos componentes de bases de datos autónomos con diferentes esquemas, capacidades de procesamiento variantes e interfaces de programación de aplicaciones (APIs)

En [Ngamsuriyaroj10] los autores hacen un estudio de desempeño de un cluster MySQL utilizando la base de datos TPCH, para ello los autores diseñaron un middelware llamado vParNDB en el cual logran un mayor desempeño sobre el cluster MySQL a través de la combinación del paralelismo intra-query[3] e inter-query [4]; se realizaron experimentaciones como reescritura de consultas y estrategias utilizando SQL para optimizar los planes de ejecución de las consultas para que puedan ser ejecutadas como subconsultas en cada nodo MySQL antes de que el resultado final sea agregado en el nodo front-end. Se encontró que el desempeño de 22 consultas TPC-H que fueron evaluadas en términos de aceleración y escalamiento mostraron ganancia con respecto a la forma del cluster original.

[3]Intra-query: Forma de paralelismo en la que una sola consulta es descompuesta en pequeñas tareas que se ejecutan en múltiples procesadores.[Liu09]

[4]Inter-query:Forma de paralelismo en la que varias consultas diferentes se ejecutan simultáneamente para mejorar el rendimiento general del sistema.[Liu09]

1.4. Descripción de Capítulos

Este documento se ha estructurado de la siguiente forma:

Al inicio se exponen los objetivos de este trabajo, se comenzó con el objetivo general y se desglosó en cuanto a lo que se pretendió alcanzar. Se expresaron las razones por las cuales se realizó del mismo.

En el capítulo 2 se mostraran los conceptos relacionados a la investigación que se hace, se hizo una búsqueda bibliográfica para enriquecer la parte teórica del documento; encontramos aquí las definiciones de los conceptos utilizados posteriormente, se incluye la explicación de los algoritmos que se usarán. Se anexa también información acerca de transacciones, concurrencia y memoria caché, ya que realizaremos experimentos que tienen que ver con estos temas.Además los conocimientos mencionados son recurrentes en la actualidad por las constantes mejoras en el hardware de los equipos.

En el capítulo 3 de experimentación se muestran los resultados obtenidos, las diferentes pruebas desde las perspectivas que se planearon, así como las comparativas correspondientes. Se menciona en esa parte tanto el hardware y software utilizado, las operaciones realizadas y los programas creados para llevar a cabo dichos experimentos; los resultados se muestran utilizando tablas.

En el 4º capítulo se redactaron las conclusiones que se tienen después de realizar este trabajo, aquí se explican los logros obtenidos así como la interpretación que se les da. Posteriormente se exponen las posibles áreas de oportunidad que están relacionadas con este tema.

Capítulo 2

Marco Teórico

A continuación se definen los conceptos que serán utilizados de aquí en adelante.

2.1. Sistema manejador de bases de datos

Consiste en una base de datos (una colección de información, usualmente organizada en registros con campos compuestos) y capacidades de inserción, actualización, recuperación y obtención de informes sobre los datos[Henderson08].

La tendencia en bases de datos en la actualidad son los de sistemas manejadores de bases de datos relacionales, los cuales son el núcleo de la mayoría de las aplicaciones en el mundo[Hellerstein07]. Los sistemas manejadores de base de datos (relacionales) sirven como depósitos de registros en prácticamente todas las transacciones en linea.

Tareas del sistema manejador de base de datos

Un sistema manejador de base de datos es el encargado de tareas tales como: definición, construcción, manipulación y gestión bases de datos entre usuarios y aplicaciones.

En la tarea de definición se especifican los tipos de datos, estructuras y restricciones de tales datos que se almacenan en la base de datos; lo cual implica la construcción de un diccionario de datos. Por construcción se entiende como guardar los datos en

un medio de almacenamiento administrado por el SMBD. La manipulación se refiere a funciones de consulta de datos en la base de datos, actualización y obtención de reportes. La gestión permite el acceso simultaneo de usuarios y programas a la misma base de datos, tener interfaces de comunicación, lenguajes de acceso a bases de datos e interfaces de programación de aplicaciones.

Entre otras funciones se tiene la protección de los datos contra el acceso no autorizado y contra fallos de hardware o software[Coronel09].

2.2. Ejecución de consultas

El procesador de consultas es el grupo de componentes de un SMBD que convierte las consultas del usuario y los comandos de manipulación de datos en una secuencia de operaciones de base de datos y ejecuta tales operaciones. SQL nos permite expresar consultas a un nivel de abstracción alto, comprensible para el ser humano, el procesador de consultas debe suministrar una gran cantidad de detalles en cuanto a cómo la consulta se va a ejecutar. Además, una estrategia de ejecución mal planteada para una consulta puede llevar a un algoritmo a ejecutar la consulta en más tiempo de lo necesario. Una consulta puede abarcar varios métodos con el objeto de ejecutar operaciones del álgebra relacional, las cuales son de alto nivel de abstracción. Estos métodos son diferentes en su estrategia básica, donde las estrategias principales son: exploración total, algoritmos hash, ordenación e indexación. Los métodos también difieren en cuanto a la cantidad de memoria principal disponible, ya que algunos algoritmos suponen que hay suficiente memoria principal y está disponible para almacenar al menos una de las relaciones involucradas en una operación. Otros asumen que los argumentos de la operación son demasiado grandes para caber en la memoria y estos algoritmos tienen costos y estructuras significativamente diferentes.

La compilación de la consulta se divide en tres pasos principales:

a) Parsing: en el que un árbol de análisis sintáctico representa la consulta y como está estructurada.

b) Query rewrite: en donde el árbol de análisis sintáctico se convierte a un plan de

ejecución inicial, que es generalmente una representación algebraica de la consulta. Este plan inicial se transforma entonces en un plan equivalente que se espera requiera menos tiempo para ejecutarse.

c) Physical plan generation: el plan de consulta abstracto a partir de (b), a menudo llamado un plan de consulta lógica, se convierte en un plan de consulta física mediante la selección de algoritmos para implementar cada uno de los operadores del plan lógico resolviéndose mediante la selección de una orden de ejecución para estos operadores. El plan físico, como el resultado del análisis sintáctico y el plan lógico, pueden ser representados gráficamente como un árbol de expresión. El plan físico también incluye detalles tales como la forma en que se accede a las relaciones consultadas, cuando se accede a ellas y si una relación debe ser ordenada.

Las partes (b) y (c) conforman lo que se llama el optimizador de consultas y éstas son las partes complicadas de la compilación de las mismas.

Los planes de ejecución son construidos a partir de operadores, cada uno implementa un paso del plan; los operadores físicos son implementaciones de operadores del álgebra relacional, pero también se requieren operadores físicos que no involucren operadores del álgebra relacional.

Se podría pensar que lo mejor es tener toda la tabla en memoria, lo cual podría ser necesario cuando se ejecuta un JOIN. Se tienen alternativas como solamente almacenar las tuplas que se requieren, también acceder a ellas utilizando un índice.

En algunas ocasiones se requerirá que se ordenen los datos, incluso algunos algoritmos requieren que parte de sus argumentos estén ordenados. El algoritmo decide la forma en la cual se recorrerán los datos ordenados esta decisión se hará tomando en cuenta el tamaño de las tablas según si son pequeñas o grandes o inclusive dependiendo si un argumento diferente al requerido en la consulta está ordenado no.

2.2.1. Operadores físicos.

Una consulta es generalmente una serie de pasos del álgebra relacional y el plan físico está regularmente compuesto de operadores físicos, los cuales son a su vez im-

plementaciones de operadores lógicos de álgebra relacional. Usando operadores físicos se logra un buen procesamiento de consultas, pero se debe estimar el costo de cada operador que se use, recurriendo a medidas como número de escrituras/lecturas (de aquí en adelante E/S), se parte sabiendo que los datos están en disco pero el resultado de la operación permanecerá en memoria; algunas veces se almacena en disco pero otras es mostrada como una salida formateada.

Definiciones.

Sean:

R*: una relación, es un conjunto de tuplas almacenados en una base de datos como una tabla.*

S*: una relación equivalente a* R*, pero su tamaño es como máximo el número de páginas de memoria.*

B(R)*: el número de bloques que se requieren para manejar todas la tuplas de la relación* R*.*

T(R)*: el número de tuplas totales de* R*.*

V(R,*a*)*: el número de valores distintos en una columana para una relación* R*.*

M-1*: el número de páginas disponibles en memoria para la operación a realizar.*

En cuanto al costo de la operación se mide en relación al número de E/S a disco, esto hace que el costo del resultado dependa del tamaño de los datos de entrada, por lo tanto cuando solo se trabaja en memoria sin escribir a disco el costo es cero.

Estimar el costo es esencial, el optimizador debe determinar cuál de los posibles planes de ejecución es probable que se ejecute más rápido.[Garcia-Molina09]

Si una relación R está agrupada (cluster) el número **B** de E/S a disco para una lectura completa de tabla, es aproximadamente igual a la cantidad de bloques.[Garcia-Molina09] Lo contrario sucede a si la relación no está agrupada.

Una búsqueda de índices requerirá menos de **B** lecturas, pero si se hace una lectura completa requerirá un número mayor de E/S que una búsqueda de índices, ya que además del índice necesita examinar la tabla completa.

2.2.2. Medición de costos de ejecución.

Se utilizan parámetros para medir costos de ejecución en una consulta, entre estos se tiene: tamaño de memoria, tamaño de las páginas, bloques de memoria. Aunque la memoria se puede usar totalmente, en ocasiones debe compartirse, así el tamaño requerido es menor, también el numero de páginas en que se divide la memoria puede estar predeterminado o se decide durante la ejecución. Regularmente en el proceso de lectura se accede un bloque de datos a la vez, aunque en la práctica se pueden leer más bloques, tal estrategia hace que se acelere el proceso de llevar los datos a memoria principal.

Se manejan tres parámetros para realizar esta medición: **B**, **T** y **V**. **B**: es el número de bloques que se requieren para manejar todas la tuplas de una relación. Se denota **B**(R), para lo cual se supondrá además que **B** está almacenado en forma agrupada. **T**: en **T**(R) indica el número de tuplas totales de R, y cuando se use **T/B** es el número de tuplas que puede contener un solo bloque. **V**(R,a): es el número de valores distintos en una columana para una relación R

Costo de operaciones de lectura

La medida que se utiliza para conocer el costo es el número de E/S, si una relación R esta agrupadaen disco entonces un búsqueda en tabla toma aproximadamente **B** lecturas, si R cabe en memoria principal entonces se puede implementar un ordenamiento en memoria con solo **B** accesos a disco. Si R no está agrupada el número de E/S requeridas es mucho mayor, si R esta distribuida el costo tiende a **T**. En el caso de una lectura de índices en general se tiene un costo menor que **B**(R) bloques, pero si se requiere examinar todos los índices entonces aumenta el número de E/S necesarios puesto que se requerirá examinar la relación y el índice.

2.3. Algoritmos de una pasada

En una operación que consta de dos argumentos y uno de ellos está en memoria, se puede leer el otro argumento desde el dispositivo de almacenamiento secundario. Se ejecuta la operación dejando en memoria el argumento más pequeño y leyendo el otro un bloque a la vez. Estos algoritmos son los siguientes:

- **Algoritmos para operaciones de una tupla a la vez:** no requiere que la relación entera esté en memoria, leen un bloque a la vez y usan las páginas de memoria principal. Las operaciones que se ejecutan de esta manera son la operación de selección $\sigma(R)$ y la operación de proyección $\pi(R)$; para ello se leen bloques de la relación R, se colocan en una página, se ejecuta la operación y se manda el resultado a salida, el número de E/S es **B** si R está agrupada y es **T** si no lo está.

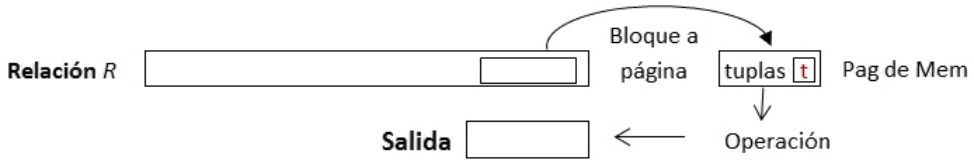

Figura 2.1: Una tupla a la vez, operaciones de selección y proyección

- **Algoritmos para operaciones unarias de relaciones completas:** requieren ver todas o la mayoría de las tuplas en memoria, por lo que está limitado al número de páginas disponibles en memoria, se aplica en eliminación de duplicados, $\delta(R)$ agrupamiento, $\gamma(R)$.

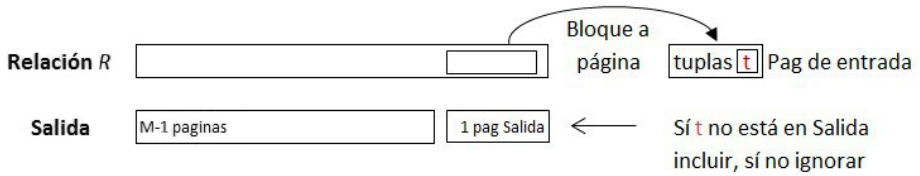

Figura 2.2: Eliminación de duplicados

- **Eliminación de duplicados:** se lee cada bloque de R uno cada vez, entonces para cada tupla: sí es la primera vez que se ha visto, se manda la salida, pero si ya ha sido vista antes se ignora; se necesita por lo regular utilizar los **M** páginas de memoria, uno para almacenar el bloque a analizar y **M-1** para almacenar las nuevas apariciones.

  El agrupamiento se refiera a la reunión de valores en un atributo de agrupación y un valor acumulado, que pueden ser: mínimo min(), máximo max() en estos dos casos hace un recorrido y va actualizando el mínimo o máximo encontrado. Contador count(), suma sum() y promedio avg(): en la función promedio se realiza un acumulado y a la vez cuenta las tuplas y al final con estos dos valores se calcula el promedio.

- **Algoritmos para operaciones binarias:** requiere que al menos uno de los argumentos de la operación tenga como tamaño máximo el número de páginas de memoria, de esta forma se resuelve en una pasada, a este argumento menor se le llama S y el argumento más grande se le llama R. Aplica sobre las siguientes operaciones: *unión* el argumento S permanece en memoria y luego solo se copiarán las tuplas de R que no estén ya en S; *intersección* solo se copiarán en la salida las tuplas que estén tanto en S como en R; *diferencia, producto y reunión natural (NATURAL JOIN)* donde se debe tomar en cuenta que: la salida deseada solo aceptará los registros que tengan los mismos valores para la columna en común de S y R, el número de E/S es $\mathbf{B}(R) + \mathbf{B}(S)$ (asumiendo que S es el más pequeño y cabe en las páginas de memoria principal). Estos algoritmos requieren como máximo **M-1** páginas.

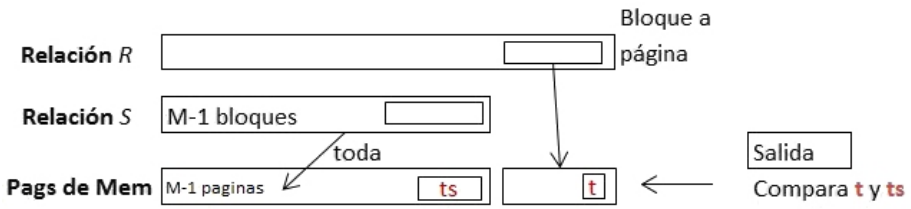

Figura 2.3: Operaciones binarias

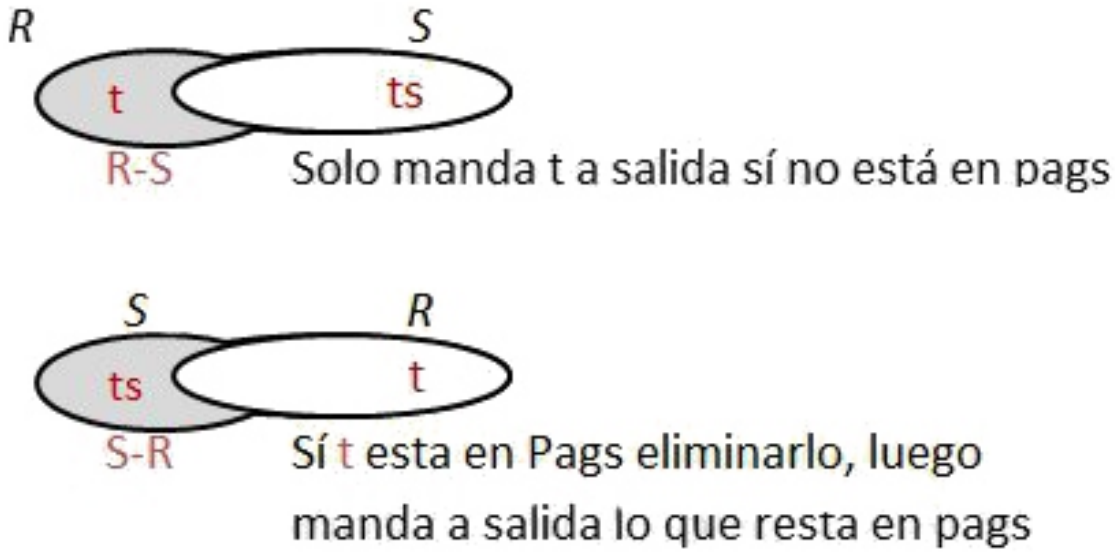

Figura 2.4: Diferencia

2.4. JOIN de ciclo anidado

Estos algoritmos leen tanto como pueden de la relación más pequeña y compara completamente con el otro argumento, se repite hasta que toda la relación más pequeña ha estado en memoria. Se estima que se resuelven en aproximadamente una y media pasadas.

- **JOIN de ciclo anidado basado en tupla:** para esta variación de dos relaciones R y S, se lee una tupla de S y se recorre cada tupla de R, se verifica que coincidan en la columna deseada y en caso de ser así se manda la tupla

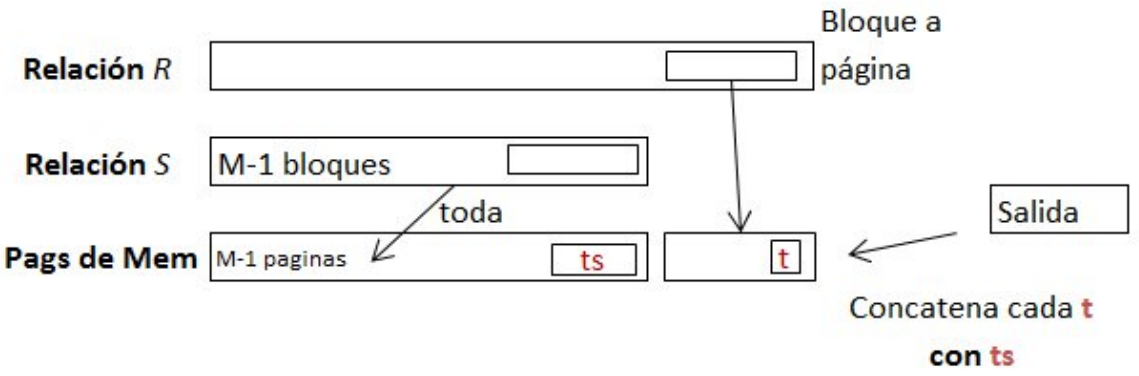

Figura 2.5: Producto

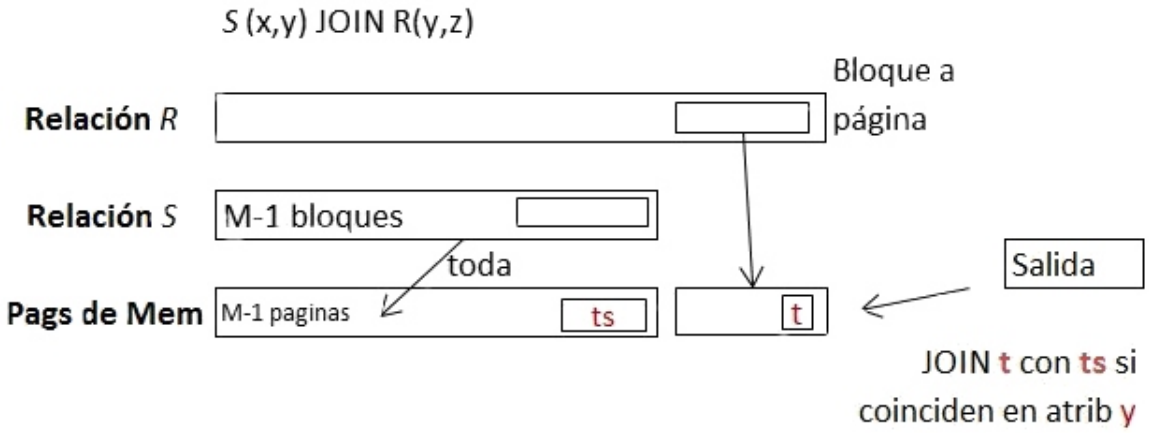

Figura 2.6: Natural JOIN

generada a la salida, se repite hasta que ambos han sido recorridos, por lo que el número de E/S requeridas es $\mathbf{T}(R)$*$\mathbf{T}(S)$.

- **JOIN de ciclo anidado basado en bloques:** en este algoritmo se van colocando en memoria los registros en bloques, se trasladan tantos bloques de S como **M-1** páginas haya, luego se leen bloques de R, en cada bloque se va comparando el registro de R y mandando a salida el que coincida en la columna deseada, es decir se hace el JOIN. El costo en lecturas es $\mathbf{B}(S)$*(**M-1** + $\mathbf{B}(R)$)/**M-1**, esto porque primeramente se requieren $\mathbf{B}(S)$/**M-1** lecturas para recorrer toda la relación S, cuando todos los bloques contenidos en **M-1** han

sido recorridos se tiene que leer el siguiente grupo de bloques de S y compararlos con el siguiente bloque (de tamaño **1**). El costo de ejecución es (**M-1**+**B**(R)).

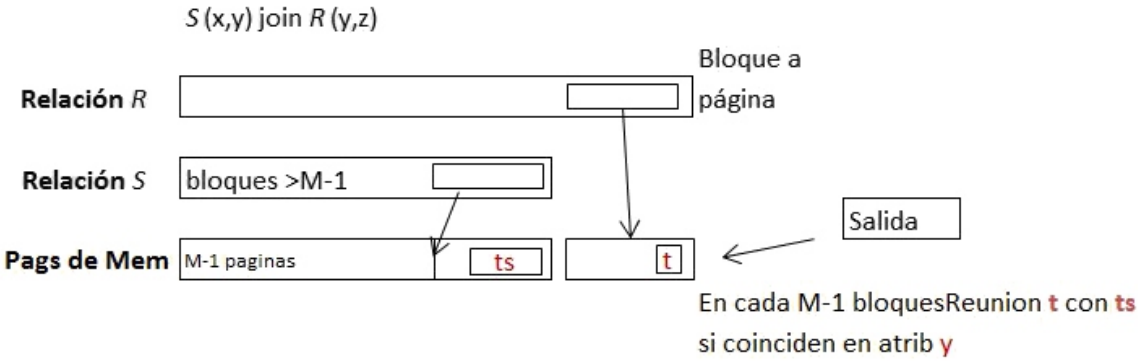

Figura 2.7: JOIN de ciclo anidado.

2.5. Algoritmos de dos pasadas

Estos incluyen algoritmos como los de ordenamiento, basados en hash o basados en índices. Se aplican cuando los argumentos son demasiado grandes para ponerlos en memoria; los datos de las relaciones son llevados a memoria principal donde se les aplica la operación, se escribe el resultado nuevamente en disco y se releen para completar la tarea.

2.5.1. Operaciones basadas en ordenamiento

En este tipo de operaciones se divide la memoria en trozos de tamaño definido y en ellas se colocan sub-listas ordenadas, en una segunda pasada estas partes se unen. Las dos fases en que se realiza es: teniendo **M** páginas, en primer lugar se leen bloques de la relación R por sublistas de tamaño **M** y se van ordenando, después se escriben e disco; en la segunda fase se hace la combinación en donde se traen **M-1** sublistas ordenadas en la fase anterior, posteriormente de ellas se van extrayendo los valores menores y se van colocando en páginas de salida, de los registros restantes en las sublistas se extrae el siguiente menor hasta que se haya llenado la página de salida,

cuando se llena se escribe a disco y se continúa, al agotarse los valores en la sublistas se traen nuevas sublistas ordenadas y se repite el proceso.

A continuación se muestra el funcionamiento de algunos algoritmos basados en ordenamiento.

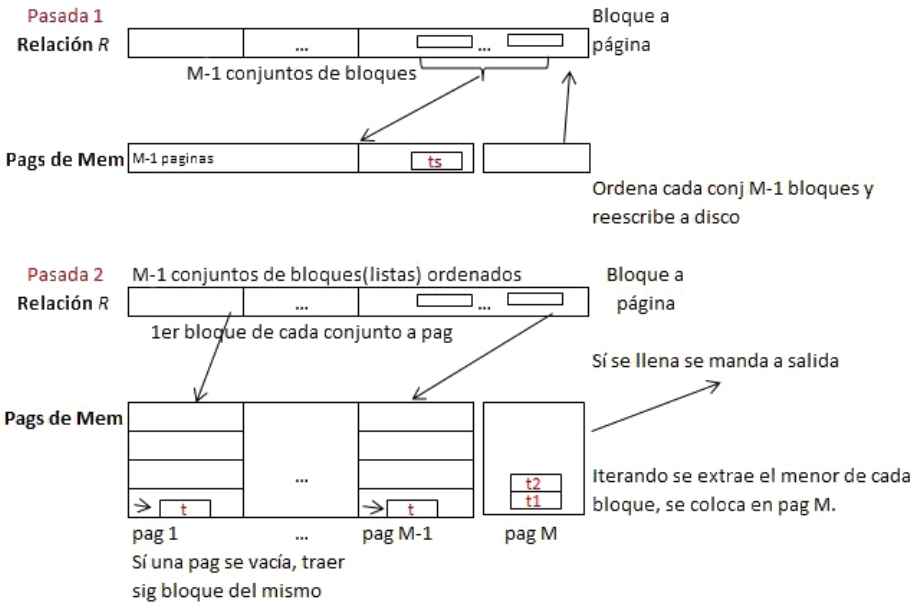

Figura 2.8: Merge sort de dos pasadas

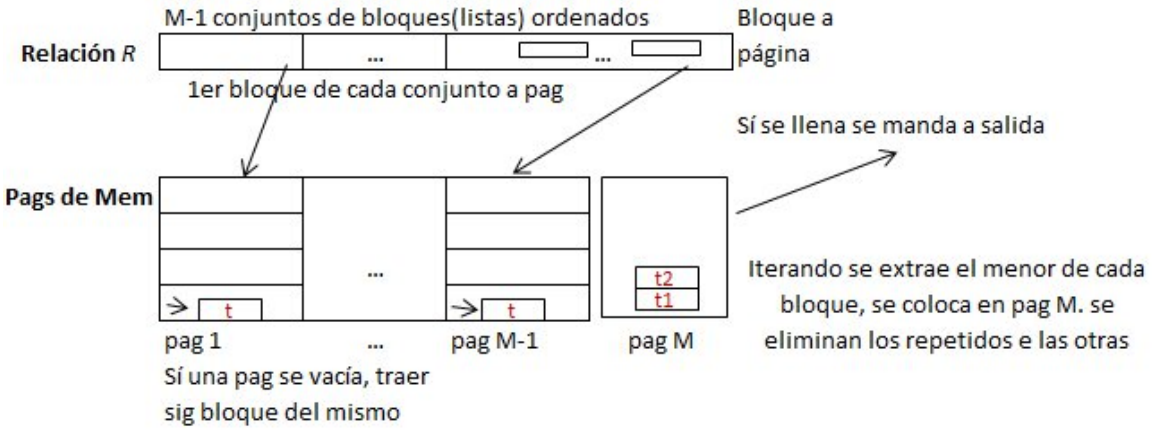

Figura 2.9: Eliminación de registros repetidos dentro de bloques previamente ordenados

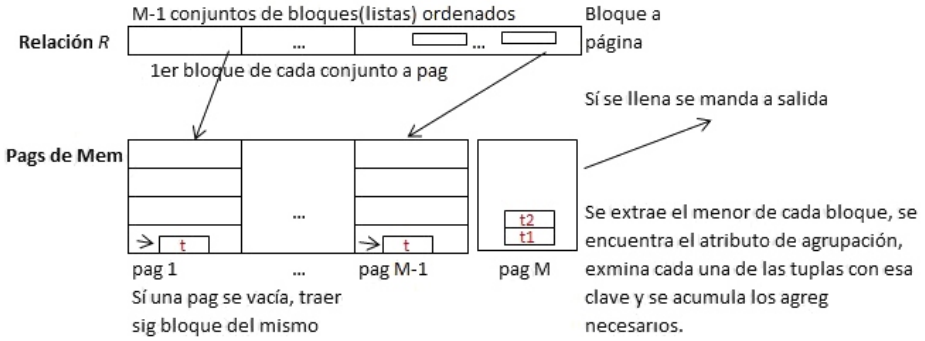

Figura 2.10: Aplicando agregación, extrayendo los menores

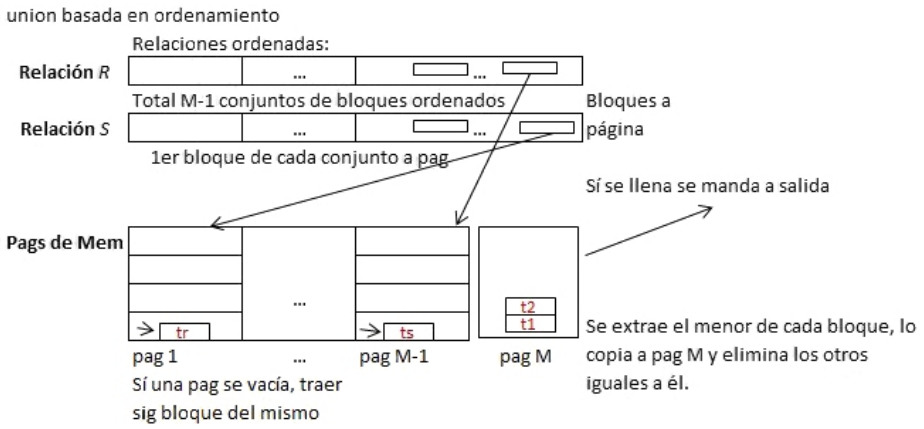

Figura 2.11: Unión de conjunto de bloques ordenados para dos relaciones

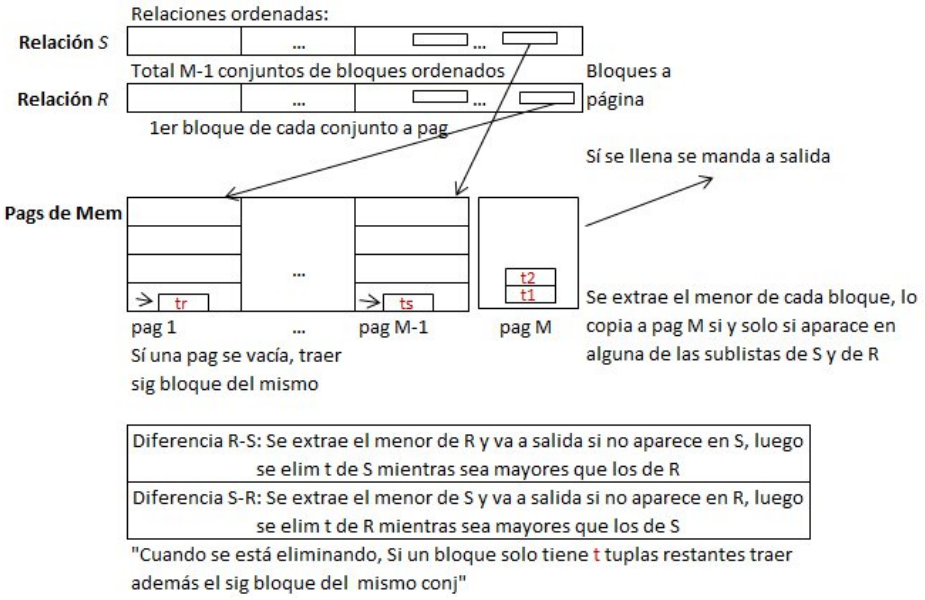

Figura 2.12: Algoritmo de intersección y diferencia

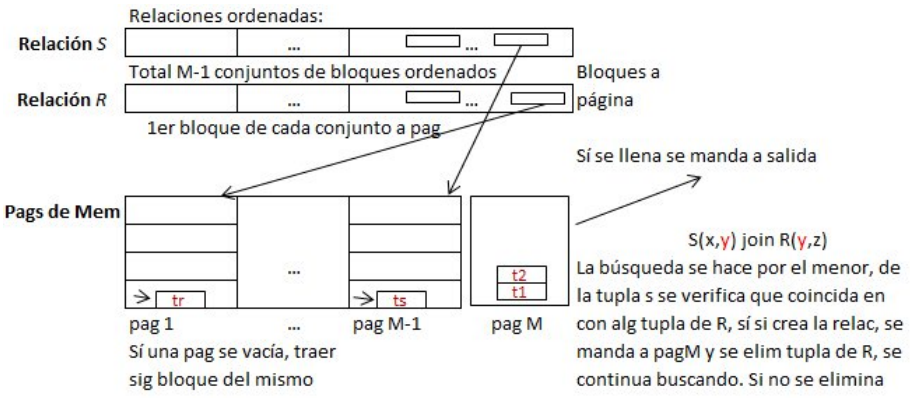

Figura 2.13: JOIN basado en ordenamiento

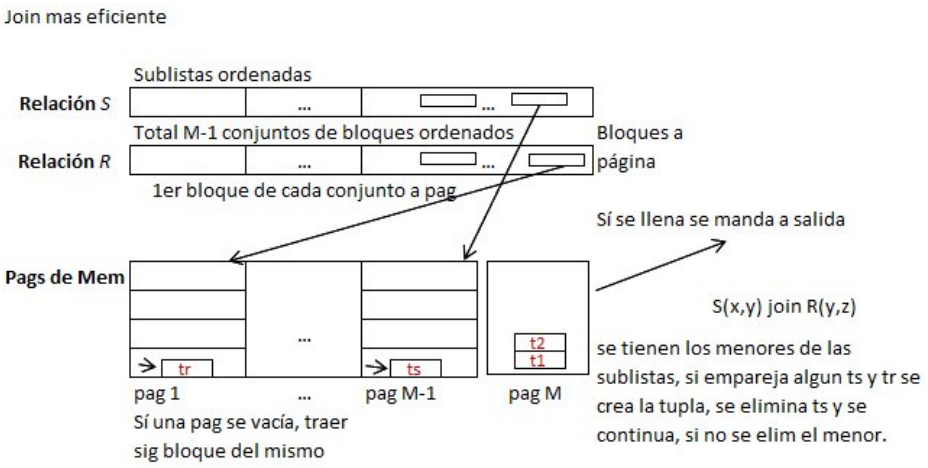

Figura 2.14: JOIN con mayor eficiencia

2.5.2. Algoritmos basados en hash

Estos algoritmos utilizan una función hash para dividir el argumento en secciones, a estas secciones les es aplicada la operación, ya sea individual o en pares de acuerdo a lo requerido por la sentencia SQL. La división en secciones es necesaria puesto que se trabaja con relaciones grandes que no caben en los páginas de memoria, se les aplica una operación hash a todas la tuplas del argumento(s) con alguna llave hash, entonces se ejecuta la operación.

- **Unión, intersección y diferencia:** se requiere aplicar la misma función hash a ambos argumentos y entonces se llenan nuevamente **M-1** páginas con argumentos de R y se les aplica la operación, tanto en el caso de la unión, intersección y diferencia a los bloques contenidos en las páginas, se les aplica los algoritmos de una pasada por lo que en la primera pasada el número de E/S requeridas es $\mathbf{B}(R)+\mathbf{B}(S)$, tomando en cuenta la escritura y la segunda lectura da un total de $3(\mathbf{B}(R)+\mathbf{B}(S))$

- **Algoritmo de hash JOIN:** al calcular una operación R(x, y) JOIN S(y,z) se aplica la llave hash a la columna común (en este caso **Y**) si los valores coinciden se obtendrá el mismo valor de hash y así corresponderán para hacer el JOIN en esas tuplas.

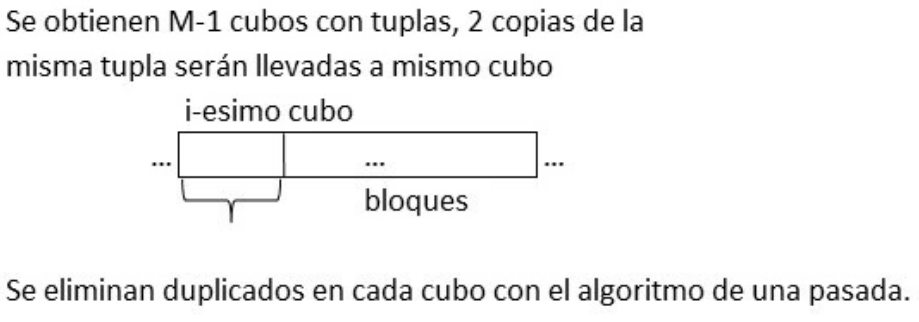

Figura 2.15: Eliminación de duplicados

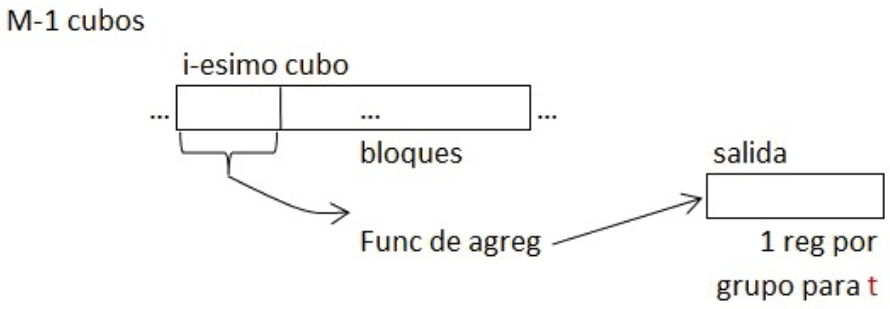

Figura 2.16: Agrupación y Agregación

2.6. Administrador de páginas de memoria

El administrador de páginas se encarga de mantener la disponibilidad de los bloques de memoria necesarios, así como las políticas de reemplazo y almacenamiento a disco. Es difícil predecir el número de páginas que serán necesarios más adelante, cuando se logra esta predicción se puede reducir el número de páginas. Se inicia suponiendo que los páginas están disponibles para los operadores, pero no siempre se tendrá la disponibilidad por adelantado, ya que la cantidad de páginas cambia dependiendo del sistema. Pueden suceder dos cosas: que administre las páginas directamente la memoria principal, que trabaje en memoria virtual y el sistema operativo sea el que maneje el uso de memoria principal. El sistema operativo puede mantener las demás páginas en la zona de intercambio del disco duro. Entre las decisiones que debe tomar el administrador de páginas se encuentran: saber en que momento un bloque debe ser vaciado porque se necesita el espacio, entre las soluciones propuestas están: el menos usado recientemente LRU, FIFO primero en entrar primero en salir, la estrategia del reloj este último es parecido al LRU, pero en este caso se le coloca una bandera a cada página tras un segundo uso, el que provocará que sean menos propensos a ser vaciados.

2.7. Algoritmos multipasada

Este tipo de algoritmos se utilizan cuando aquellos de dos pasadas: ordenamiento o hash trabajan en forma recursiva. Toman tres o más pasadas ya que además trabajan sobre gran cantidad de datos. Para el caso del ordenamiento multi-pasada, se hace una generalización del algoritmo de tres pasadas, cuando la relación R no cabe en memoria principal ésta se divide en **M** grupos donde **M** es el número de páginas disponibles de memoria, luego se ordena cada grupo de forma recursiva y se mezclan las sublistas como si fuera un algoritmo de dos pasadas

2.8. Compilador de consultas

A continuación se describe el proceso de compilación de consultas.
El procesador de consultas ejecuta tres pasos básicos:

a) La consulta escrita es analizada sintácticamente, para ello utiliza un árbol de análisis donde se representa la estructura de la consulta.

b) El árbol de análisis es transformado en una expresión de álgebra relacional, el cual vemos como el plan de ejecución lógico.

c) Este plan de ejecución lógico es transformado en un plan de ejecución físico, se indican no solo las operaciones ejecutadas sino el orden, los algoritmos usados y la forma en que los datos son obtenidos.

2.8.1. Análisis sintáctico y pre-procesamiento

Comienza con un análisis sintáctico, el objetivo es tomar la sentencia escrita en lenguaje SQL y convertirla en un árbol de análisis sintáctico, en este árbol sus nodos corresponden a *átomos* los cuales son elementos léxicos de SQL y *categorías sintácticas*, que son nombres que se le dan a subpartes de una consulta. Si el nodo es un *átomo* entonces no tiene hijos, en caso contrario el hijo es descrito por las reglas sintácti-

cas para el lenguaje, las cuales son reglas que describen un pequeño subconjunto de consultas SQL, a continuación se muestran estas reglas.

Select

[Select-list] ::= [Atributo] , [Select-list]

[Select-list] ::= [Atributo]

From-Lists

[FromList]::=[Relacion],[FromList]

[FromList] ::=[Relacion]

Condiciones

[Condicion] ::= [Condicion] AND [Condicion]

[Condicion] ::= [Atributo] IN ([Query])

[Condicion] ::= [Atributo] = [Atributo]

[Condicion] ::= [Atributo] LIKE [Patron]

El preprocesador

Es un programa que identifica las sentencias de base datos y las extrae para que el SMBD las procese[Elmasri10]. También es responsable del análisis semántico [1]de cada sentencia que aunque sea válida sintácticamente, puede violar las reglas semánticas; por lo cual debe:

a) Verificar las relaciones usadas.

b Verificar y resolver atributos usados (cada atributo en la consulta debe aparecer en la relación).

c) Verificar tipos de datos.

[1]semantic checking.

2.8.2. Planes de ejecución lógicos

Una vez que se construye el árbol de análisis, es necesario convertir este árbol en el plan de ejecución lógico.

Primeramente se reemplazan los nodos y estructuras del árbol de análisis, y se incorpora(n) el operador(es) de álgebra relacional correspondientes.

Posteriormente, se toman las expresiones de álgebra relacional producidas por el paso anterior y se transforman en expresiones, se espera que estas expresiones sean convertidas en un plan de ejecución físico más eficiente.

Transformación en álgebra relacional

Existen algunas reglas para hacer esta transformación.

Cuando se tiene una consulta sin que tenga subconsultas, se puede reemplazar la construcción entera por una expresión de álgebra relacional, la cual consiste en lo siguiente: el producto de todas las relaciones que aparecen en <fromList>, una selección $\sigma_C(R)$ donde C es la <condicion> y una proyección $\pi_L(R)$ donde L es la lista de atributos de <Select-list>.

Cuando se tienen subconsultas, se utiliza una estrategia que se le conoce como selección de dos argumentos; la cual se deberá representar como un árbol de análisis transformado por un nodo sin parámetros, este nodo se etiqueta como σ, bajo este nodo, está del lado izquierdo un nodo hijo que representa la relación R sobre la que se está realizando la selección y a la derecha otro nodo hijo, el cual es una expresión para la condición aplicada a cada tupla de R. Ambos argumentos pueden ser representados como árboles de análisis, como árboles de expresión o una combinación de los dos.

Se puede hacer una mejora al plan de ejecución. Para ello se tiene la siguiente estrategia. Se obtiene un plan lógico de ejecución tentativo, luego se reescribe usando leyes algebraicas, pero se pueden generar planes alternativos. Se tienen algunas reglas para mejorar el plan de ejecución lógico.

Las selecciones[2] pueden ser apiladas en el árbol de expresión. Si contienen un AND

[2]select

con muchas condiciones, entonces se divide cada condición y se apila cada parte del árbol de forma separada. Análogamente las proyecciones pueden ser apiladas al árbol o agregar nuevas.

Las eliminaciones repetidas pueden ser removidas o trasladadas a una mejor posición en el árbol, aunque lo recomendable es combinar estas dos estrategias.

Tratamiento de operadores asociativo/conmutativos

Un operador que es asociativo/conmutativo[3] puede tener cualquier numero de operadores, la estrategia es reordenarlo para que se ejecute en pares. Para ello en cada parte del sub-árbol que tenga nodos con el mismo operador asociativo o conmutativo, se agruparán los nodos similares en uno con muchos hijos.

2.8.3. Costo de ejecución

Después del análisis sintáctico y transformación a plan de ejecución, se continúa con un plan de ejecución físico, de los posibles planes generados se hace una estimación de costo[4], se elige la menor entre las estimaciones, se toman en cuenta para ello: el tamaño de las relaciones intermedias, el tamaño del la proyección, la cantidad de atributos en JOIN, la cantidad de relaciones y otras operaciones (unión, intersección, diferencia, agrupación y agregación).

Elección de un plan de ejecución

El procesador de consultas debe transformar el árbol de análisis, a expresiones de álgebra relacional. Esta transformación es un tanto simple excepto para las subconsultas, esto se resuelve con la utilización de una estrategia de selección de doble argumento en donde se pone la subconsulta en una condición especial de selección y entonces se aplican transformaciones para casos especiales. El procesador de consultas debe elegir el plan de ejecución más eficiente, también elegir la mejor forma de

[3]Por ejemplo Natural JOIN, UNIÓN e INTERSECCIÓN
[4]cost-based enumeration

agrupar operadores asociativos y conmutativos.

2.9. Transacciones

Una transacción es un unidad lógica de procesamiento de base de datos que consta de una o más operaciones, estas operaciones incluyen: inserción, borrado, modificación o recuperación de datos. Las operaciones de base de datos que forman una transacción pueden estar embebidas dentro de un programa de aplicación, también pueden ser especificadas interactivamente a través de un lenguaje de alto nivel como SQL[Elmasri10].

Una transacción también puede ser entendida como: un grupo de sentencias SQL que son tratadas de forma atómica, es decir como una sola unidad de trabajo. Si el motor de la base de datos puede ejecutar el grupo completo de sentencias a la base de datos, se aplican, pero si cualquiera de ellas no puede hacerse por algún problema u otra razón, entonces ninguna de ellas es aplicada[Schwartz12].

En las transacciones se tienen tareas como lectura, borrado, inserciones, actualizaciones en los datos de la base de datos; se requiere mantener la coherencia en tales datos. Se deben cumplir las restricciones que son: aquellas implícitas o basadas en el modelo(se especifican en el lenguaje de definición de datos) y las restricciones semánticas o reglas de negocio.

Comúnmente en los SMBDs las transacciones están formadas por dos o más peticiones a la base de datos. Una petición de base de datos es el equivalente a una sola sentencia SQL en un programa de aplicación o transacción. Por ejemplo: si una transacción está compuesta por dos sentencias UPDATE y una sentencia INSERT, la transacción usa tres peticiones de base de datos. Cada petición a la base de datos genera muchos operaciones de E/S que leen y/o escriben en un medio de almacenamiento físico[Coronel09].

2.9.1. Propiedades de las transacciones

Para asegurar que se mantenga la consistencia de una base de datos, las transacciones deben poseer un conjunto de propiedades, las cuales son conocidas como ACID por sus siglas en inglés (Atomicity, Consistency, Isolation, Durability), deben ser cumplidas por el SMBD a través del control de concurrencia.

Atomicidad: una transacción es una unidad completa de procesamiento, que se ejecuta de forma total o no se ejecuta en absoluto.[Elmasri10].

Consistencia: se mantiene la consistencia si su ejecución total se lleva de un estado coherente a otro.[Elmasri10].

Aislamiento: la transacción debe dar la apariencia de estar ejecutándose en forma independiente a las demás, la ejecución de una de ellas no debe afectar la realización de ninguna otra.[Elmasri10].

Durabilidad: los cambios que se apliquen por una transición confirmada deben quedar registrados en la base de datos. Cambios que no deben perderse por fallas del sistema.[Elmasri10].

2.9.2. Niveles de aislamiento

La restricción de una transacción SQL previene que: ninguna acción de una alguna otra ejecución pueda afectar la visibilidad de los datos mientras esa transacción dure. Por ejemplo si se ejecuta una consulta durante una transacción, se realiza otra tarea durante la espera y después se ejecuta la misma consulta una segunda vez. El resultado debe ser el mismo si la transacción no ha cambiado los datos. Esta capacidad de poder recuperar una fila durante una transacción es la de más alto nivel, pero también es la de mayor costo en términos de bloqueo de la base de datos. Para reducir el costo de ejecución, se utilizan niveles de aislamiento menos restrictivos

Lectura no comprometida (READ UNCOMMITTED): es el nivel de aislamiento menos restrictivo. En este nivel se pueden dar lecturas no confirmadas y lecturas inconsistentes. El objetivo es permitir que se puedan ejecutar lecturas y escrituras intercaladas [Ozsu91].

Lectura comprometida (READ COMMITED): a una transacción no se le permite ver las actualizaciones que aún no han sido confirmadas por otras transacciones, por lo cual no ocurren problemas de pérdida de actualizaciones y datos no confirmados. Aunque dos lecturas para las misma transacción pueden dar diferentes resultados[Ozsu91].

Lectura repetible (REPEATABLE READ): es un nivel de aislamiento más restrictivo que READ COMMITED. Incluye READ COMMITED, especifica que ninguna otra transacción puede modificar ni eliminar datos que la transacción actual haya escrito mientras no se confirmen. El paralelismo es menor que en READ COMMITED, durante la transacción se mantienen bloqueos compartidos en los datos leídos en lugar de liberarlos al final de cada instrucción[Liu09].

Nivel serializable (SERIALIZABLE): el nivel de aislamiento más restrictivo de todos. En este nivel se realiza un bloque completo a las otras transacciones. Con la intención de que la transacción sea verdaderamente serializable[Liu09].

2.10. Control de concurrencia

En esta sección se explican algunas técnicas para manejar la concurrencia, ya que al ejecutar dos o más transacciones simultaneas se deben tener mecanismos para garantizar que no hayan interferencia, lo cual daría lugar a estados inconsistentes en la base de datos.

2.10.1. El bloqueo de dos fases

Un bloqueo es una variable asociada a un elemento de datos, describe el estado de ese elemento respecto a posibles operaciones que se le pueden aplicar[Elmasri10]. Un bloqueo para cada elemento de datos en la base datos, este bloqueo se requiere para lograr la sincronización del acceso de transacciones concurrentes a la base de datos.

Bloqueo binario se compone de dos estados que son *bloqueado* y *debloqueado*, para indicarlos se utilizan más como 1 y 0. Si algún objeto posee el valor 1 no es permitido a otras operaciones accederlo, lo cual si es permitido cuando se encuentra en 0; sus dos

operaciones son *bloquear* y *desbloquear*. Cuando se solicita una operación de bloquear y el objeto ya está bloqueado se tiene la obligación de esperar hasta que el valor cambie a 0. Es necesario que esta parte de la transacción ya sea bloquear o desbloquear se ejecute de forma atómica, es decir desde que inicie hasta que termine debe hacerse en forma completa.

En el bloqueo compartido exclusivo, únicamente se permite el acceso de varias transacciones a un mismo elemento cuando la operación es para leer (compartido), en caso de una escritura se debe realizar un bloqueo(exclusivo) a ese elemento. Para lograr esto se tienen tres tipos de operaciones: bloquearLectura(), bloquearEscritura() y desbloquear(), lo cual conlleva a tres estados que son *bloqueado para lectura*, *bloqueado para escritura* y *desbloqueado*.

2.11. Mecanismos de recuperación y registro histórico

El registro histórico es una secuencia de registros que mantiene un control de todas las actividades de actualización de la base de datos[Silberschatz05]. Un control de actualización del registro histórico describe una única escritura en la base de datos, tiene los siguientes campos:

- El identificador de la transacción: es un identificador único de la transacción que realiza la operación escribir.
- El identificador del elemento de datos: es un identificador único del elemento que se escribe. Normalmente suele coincidir con la ubicación del elemento de datos en el disco.
- El valor anterior: es el que tenía el elemento de datos antes de la escritura.
- El valor nuevo: es el que tendrá el elemento de datos después de la escritura.

La recuperación es necesaria, el sistema es responsable de garantizar que todas las operaciones de la transacción se completen satisfactoriamente, que su efecto se grabe

permanentemente en la base de datos, que la transacción no afecte a la base de datos o a cualquier otra transacción. El SMBDs no debe permitir que solo algunas operaciones de una transacción se apliquen a la base de datos ignorando otras; esto puede ocurrir si una transacción falla después de ejecutar algunas de sus operaciones, pero antes de ejecutar todas ellas.[Elmasri10]

En el caso de SQLServer, los modelos de recuperación se han diseñado para controlar el mantenimiento del registro de transacciones. Existen tres modelos de recuperación: simple, completa y por medio de registros de operaciones masivas. Normalmente, en los SMBDs se usa el modelo de recuperación completa o el modelo de recuperación simple

2.12. Memoria caché

La memoria caché es un área de memoria, la cual tiene un acceso relativamente rápido, en donde los datos pueden ser almacenados de manera anticipada antes que sean necesitados por el procesador[Henderson08]. El caché se maneja en dos conceptos, el cache de disco y el caché de CPU [5].

2.12.1. Caché de CPU

Cuando se utiliza el caché del CPU se tienen ventajas, los datos e instrucciones pueden ser precargados desde el caché que está cerca o junto del CPU, datos e instrucciones que son traídos desde la memoria RAM. Un algoritmo analiza las instrucciones que son ejecutadas actualmente por el procesador e intenta predecir que instrucciones y datos serán requeridos en el futuro. Estas instrucciones y datos que son previstas se transfieren de memoria principal al caché, mientras el procesador está aún ejecutando las instrucciones anteriores. Sí la predicción fue correcta, el CPU carga las instrucciones y datos que ya están disponibles en la memoria caché de alta velocidad, como consecuencia, se tiene un incremento en la velocidad del proceso sin requerir incrementar la frecuencia de reloj. La efectividad del procesador depende de dos cosas: la combinación de datos e instrucciones y de la ubicación de la memoria cache. Si se usan grandes secuencias repetitivas de instrucciones o datos, se tendrá una aceleración; un caché dentro del mismo CPU llamado L1[6] es más rápido que un L2[7], que esta en un chip separado.

Los cambios en datos son escritos al caché (no a memoria) hasta que se llena, luego son enviados a memoria para mantener la coherencia. [8]

[5]Unidad central de proceso
[6]Level 1
[7]Level 2
[8]tomado de: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1

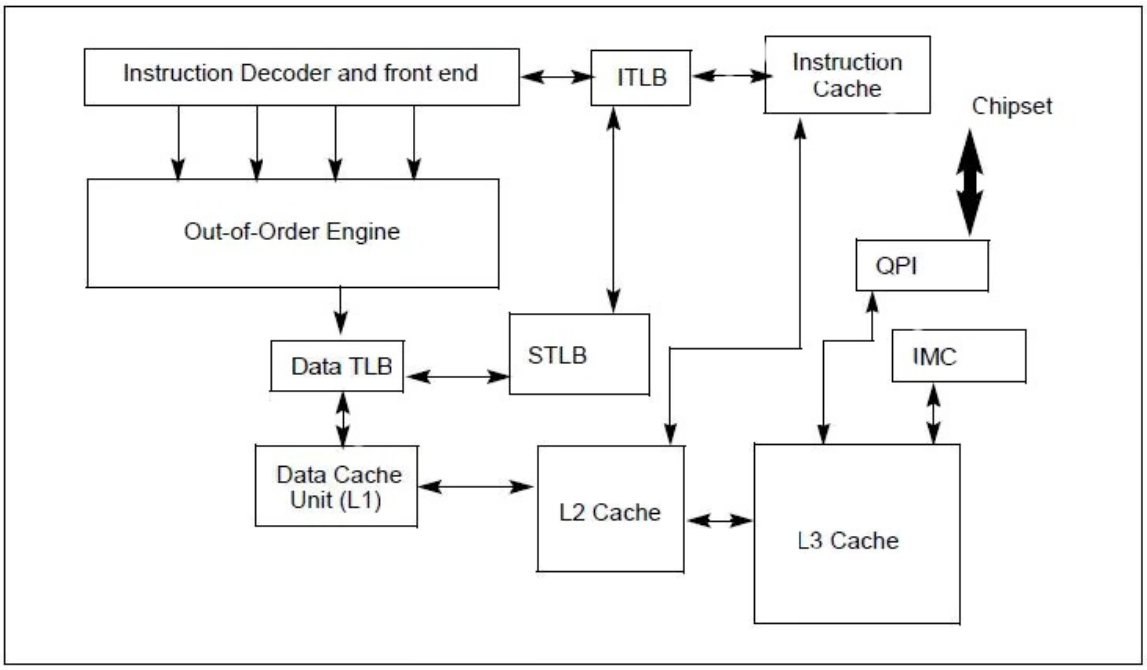

Figura 2.17: Estructura del caché en procesador Intel i7

2.12.2. Cache de disco.

Aquí se utiliza una parte de la memoria RAM[9] principal o una RAM incrustada en el disco, ya que la tasa de E/S a disco es lenta en relación a memoria principal. Al requerirse datos del disco, el caché lee uno o más sectores completos de disco no sólo el que se le está pidiendo, de esta forma si la aplicación continúa pidiendo datos secuenciales, estos serán leídos del caché del alta velocidad en lugar de la unidad de disco. Al escribir datos a disco estos pueden ser acumulados en el cache, escribirlos de vuelta a la unidad de disco en conjuntos de bloques. Es uso del caché incrementa la eficiencia, pero un problema de suministro de energía o algún otro borra y/o corrompe el contenido del caché ya no estará disponible, lo cual podría causar corrupción en una base de datos[Henderson08].

[9]Memoria de acceso aleatorio.

2.13. Desempeño de una base de datos

El desempeño de un sistema de base de datos es afectado por funcionalidades complejas e interdependientes, por ejemplo: el manejo de transacciones, contención de concurrencia, administración de caché y contención de disco[Osman12].

Se tendrá una mejor ejecución cuando el harware y el software estén correctamente optimizados, pero siempre existen restricciones internas y externas. Por tanto los componentes del sistema deben ser optimizados para obtener el mejor desempeño posible[Teorey11].

2.13.1. Desempeño de un sistema de procesamiento de transacciones

El desempeño de un sistema de procesamiento de transacciones o de forma más general un SMBDs es medido en sistemas operacionales, prototipos y test de desempeño. El análisis probabilístico y de colas ha sido usado para obtener una perspectiva del rendimiento del sistema de procesamiento de transacciones[Liu09]; se consideran aspectos como análisis de encolado en procesadores y discos, modelos de redes de encolado, técnicas para estimar la tasa de pérdidas en páginas de base de datos, factores que afectan el desempeño de RAID[10], métodos para gran concurrencia de datos y su análisis.

[10]Conjunto redundante de discos independientes

Capítulo 3

Experimentación

Existe la necesidad de extraer los datos almacenados en una base de datos. Ya que la cantidad de esos datos tiende a crecer, se busca constantemente que la extracción de datos sea rápida. Los datos obtenidos se transfieren a la aplicación que la va a utilizar o en su defecto se envían directamente al usuario a través de algunas de las interfaces que el gestor ofrece.

Para la extracción de la información se ejecutan peticiones a la base de datos, estas pueden ser consultas muy elaboradas que se componen en muchos casos de numerosos argumentos, tales consultas deben ejecutarse dentro del motor de la base de datos y posteriormente retornar la información. Este proceso implica un descomposición en otras operaciones, estos factores impactan en el tiempo de respuesta.

Este trabajo muestra el comportamiento de diferentes sentencias de ejecución. En este capítulo se presentan los resultados obtenidos de las diferentes pruebas realizadas.

3.1. Ejecución

Se refiere a la ejecución de alguna(s) sentencia(s) que son efectuadas en los SMBD que se utilizan, ya sea que envía sentencias directamente en la interfaz proporcionada por el propio gestor y/o a través de una aplicación, se utilizó un programa desarrollado para este fin; la interfaz desarrollada se hizo en lenguaje java, misma que se explica

mas adelante.

Las pruebas.

Hardware. El hardware utilizado para la realización de estas pruebas es:

Procesador: Intel i7, a 2.2gigahertz con cuatro núcleos.

Caché: nivel 1 de 32kbytes, caché nivel 2 de 256kb y caché nivel 3 de 6mb.

Memoria: 6144Mb de memoria RAM DDR3 y un bus de 667mhz.

Almacenamiento: disco duro 596Gb, se trabajó en una partición de 500gb ntfs en ejecuciones para S.O. windows y 12GB ext 4 en GNU/Linux

Software

Sistema operativo: Estas pruebas se hacen sobre tres SMBDs en sus versiones más actualizadas, para ello se busca que los tres se ejecuten en igualdad de condiciones, por lo que se hacen en el mismo sistema operativo.

Sistemas manejadores de base de datos:

Microsoft SQL Server 2008

MySQL Server 5.5

PostgreSQL 9.2

Base de datos

Para la realización de experimentos se recurre a datos generados, se utiliza la base de datos del proyecto TPC-H, es una de las bases de datos utilizadas para pruebas de desempeño tanto en la industria como en la investigación académica. Los datos se generaron en un factor de escalación de 1, por factor de escalación se entiende que se generan datos en el orden del millón de registros, se obtuvieron en la tabla más grande

6 millones de filas. Las consultas y datos para poblar la base de datos se eligieron por tener una amplia relevancia a nivel industrial[1].

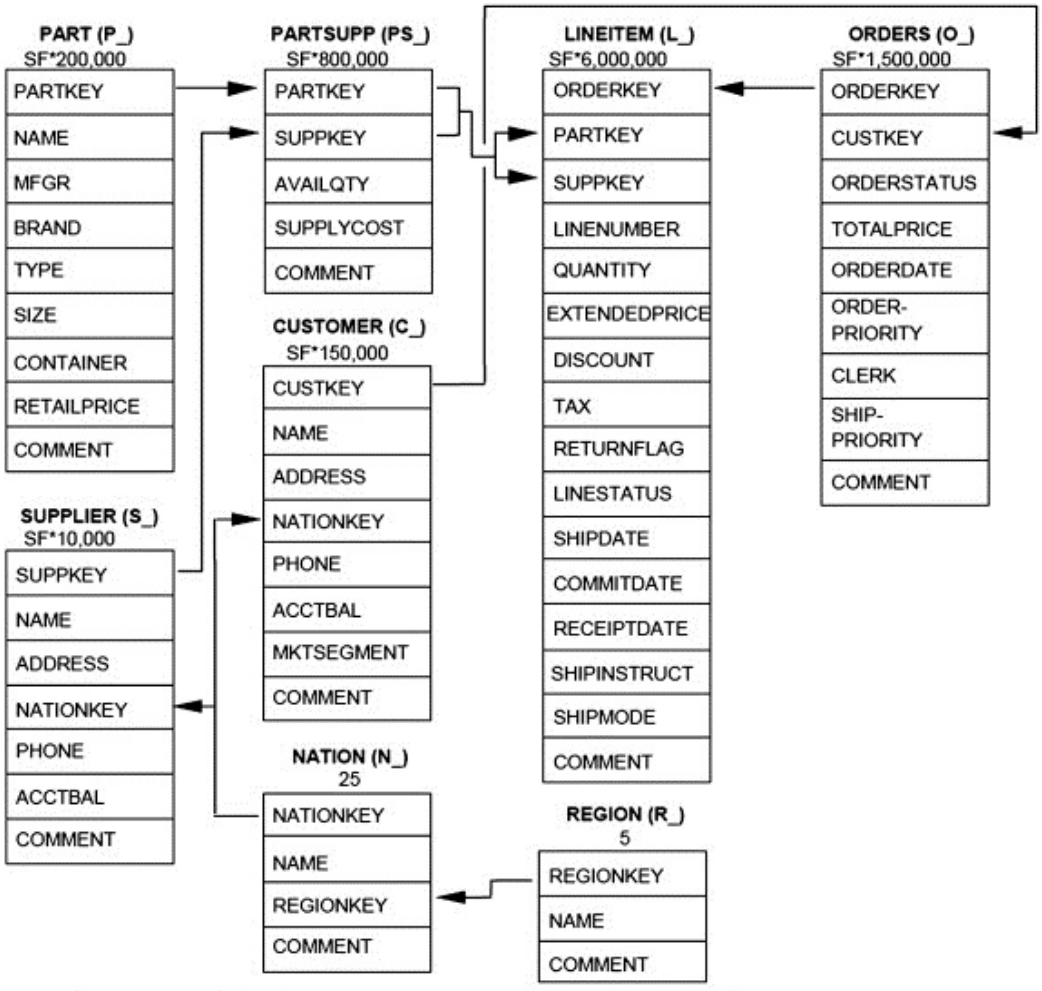

Figura 3.1: Base de datos TPC-H.

[1]En la biblioteca digital ACM se encontraron alrededor de 410 entradas relacionadas, en la biblioteca digital Springer se tiene 160 resultados; esto a la fecha de realización de este trabajo.

Interfaz para ejecución de experimentos.

Se desarrolló una interfaz para la medición del desempeño, con la opción de elegir cada una de las opciones desde las cuales se hace el análisis. El código de ellas se incluye en el apéndice, la idea de esta interfaz es automatizar la tarea, posteriormente extraer tendencias y/o hacer comparaciones y verificar la ejecución de los diferentes enfoques, ver apéndice: C.

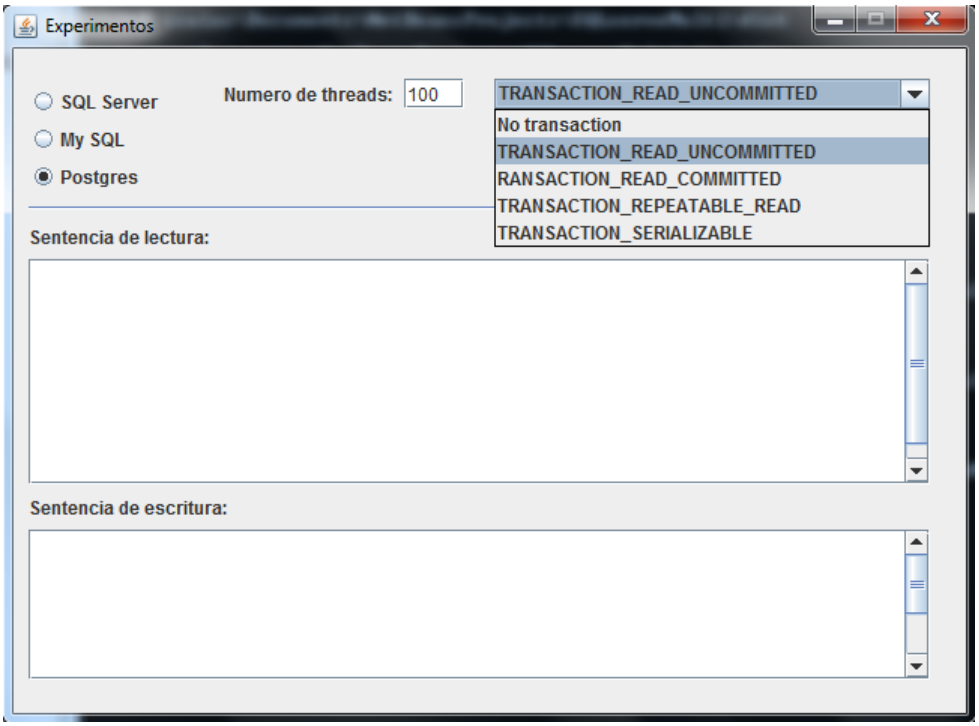

Figura 3.2: Ventana para ejecución de consultas concurrentes.

3.2. Desarrollo de experimentos.

Estas pruebas se hacen sobre tres SMBDs en sus versiones más actualizadas a la fecha, para ello se busca que los tres se ejecuten en igualdad de condiciones, por lo que se hacen en el mismo sistema operativo y en el mismo hardware 3.1.

Las sentencias siguen el orden de clasificación de algoritmos que expone [Garcia-Molina09]

en el capítulo Query Execution, así que se inició la ejecución de consultas de relaciones unarias con selección y proyección, posteriormente se hizo un JOIN entre las tablas LINEITEM y PARTSUPP, lo cual arroja un total de $4.8*10^{12}$ combinaciones de resultados, lo que se midió fue el tiempo en calcular tales combinaciones. Se observó que los tiempos se decrementan cuando se hace una JOIN únicamente sobre las llaves primarias.

Tabla 3.1: Ejecución de sentencias.

Prueba\servidor	Mysql	Postgres	Sql server
Select * from LINEITEM where L_PARTKEY=199997;	24.2	52.4	32
Select L_ORDERKEY from LINEITEM where L_PARTKEY=199997;	22.2	55.4	26
Select distinct L_ORDERKEY from LINEITEM where L_PARTKEY=199997;	23	52	30
Select distinct L_PARTKEY from LINEITEM where L_PARTKEY=199997;	24	31	26
Select L_PARTKEY, L_ORDERKEY from LINEITEM where L_ORDERKEY between 199995 and 199997 group by L_PARTKEY, L_ORDERKEY;	21	29	27
select count(*) from PARTSUPP inner JOIN LINEITEM on L_SUPPKEY=PS_PARTKEY;	193	103.4	35
select COUNT(*) from ORDERS left JOIN LINEITEM on L_ORDERKEY = O_ORDERKEY left JOIN PARTSUPP on PS_PARTKEY=L_PARTKEY group by O_ORDERSTATUS;	35	32	80
select distinct PS_PARTKEY from PARTSUPP cross JOIN PART where PS_PARTKEY in (select L_PARTKEY from LINEITEM)	38	40	32

Las unidades que se muestran en las tablas se refieren a unidades de tiempo, estas unidades están representadas en segundos.

La ejecución de esta prueba generó un notable aumento de tiempo, y en un momento dado provocó que el sistema la detuviera, la misma consulta se utilizó posteriormente solo con agregarle la palabra clave distinct, en ese caso la consulta termina en el tiempo mostrado.

Por lo tanto se tiene que: en el caso de retornar gran cantidad de datos el mayor tiempo no lo consumirá el SMBD en el procesamiento mismo, sino al presentar los datos en salida.

Se observa que incluso en la segunda consulta: tomó más tiempo solo el procesamiento de los datos de este caso en particular, pues incluye además la restricción del distinct, pero al evitar la salida de datos duplicados requerirá una menor cantidad de memoria.

3.3. Pruebas con ejecución concurrente y niveles de aislamiento.

Para esto se realizó una aplicación con la técnica de multihilos, ejecuta las sentencias para cada uno de los tres gestores utilizados. Los tiempos se muestran en segundos, para la realización de estas pruebas se utilizan las consultas mostradas abajo. Se utilizó la interfaz 3.1 para que aleatoriamente fuese eligiendo alguna sentencia, de tal manera que la asignación se distribuyese de forma uniforme entre la cantidad de hilos de ejecución.

Aquí sin embargo se debe tomar en cuenta que existe un límite en la búsqueda de incremento de desempeño de un DBMS, como se menciona en esta referencia[Schwartz12] dos factores fundamentales pueden impactar en la búsqueda de eficiencia, entre los cuales se tiene la saturación: del CPU y de las entradas/salidas.

Estas pruebas de ejecución de consultas concurrentes se hacen probando los cuatro niveles de aislamiento en el SMBD 2.9.2, en los resultados obtenidos se observa que los niveles más restrictivos dan tiempos de ejecución más grandes, en este caso serializable contra los menos restrictivos. Para el nivel READ UNCOMMITTED se obtienen incluso lecturas inconsistentes, en un momento dado lee registros cuya permanencia en la base de datos no ha sido confirmada. Por lo tanto se tiene que: en aplicaciones donde se requiera tener tiempos de respuesta inmediatos, aunque implique una lectura de datos no actualizados, ahí se recomienda usar el nivel de aislamiento menos restrictivo.

Consultas utilizadas

Sentencia 1:

```
Select distinct l_partkey from lineitem where l_partkey
in ( Select p_partkey from part where p_container='SM CASE')
group by l_partkey having avg(l_quantity) > 30;
```

Sentencia 2:

```
update part set p_container='facultad' where p_container=
'SMCASE';
```

Tabla 3.2: concurrencia con niveles de aislamiento.

Sql server	Mysql	Postgres	No Procesos	Nivel aislamiento
15.6	42.4	5.29	10	READ UNCOMMITTED
15.8	41.9	6.5*	10	READ COMMITTED
33*	75.7	7.5	10	REPEATABLE READ
27.9*	2.7*	34.8	10	SERIALIZABLE
15.5	177.9	52.5	50	READ UNCOMMITTED
17.9	176.8	32.7*	50	READ COMMITTED
17.8*	361.8	27.6*	50	REPEATABLE READ
20.25	27.4	30.2	50	SERIALIZABLE

Se observó que se produjo un deadlock[2] en la base de datos. Este ciclo resulta de interacciones entre aplicaciones centralizadas de base de datos y los SMBDs[Grechanik12], hasta la fecha los deadlocks de base de datos son detectados por los SMBDs[3] utilizan diferentes algoritmos y logran la ruptura del ciclo de espera.

[2]Interbloqueo: Cuando dos o más transacciones están mutuamente reteniendo y pidiendo bloqueo del mismo recurso, crean un ciclo de dependencia.

[3]Postgres solo soporta tres niveles de aislamiento, ya que el READ UNCOMMITED lo trata igual que READ COMMITED

3.4. Experimentación con escalamiento de memoria

En este caso se muestra el desempeño de los gestores cuando se les restringió la cantidad memoria, para ello se realizó una prueba de 10 ejecuciones concurrentes con 128mb, 256mb y 512mb para los datos en memoria en cada uno de los SMBD. A continuación se muestra el tiempo obtenido no el resultado de la consulta. Cabe recordar que las sentencias que se utilizan son las mismas que para las anteriores pruebas expuestas y el tiempo se muestra en segundos.

Tabla 3.3: Escalación de memoria.

Memoria	No Procesos	sql server	MySQL	Postgres
128Mb	10	36.8	34.4	9.4
256Mb	10	17.0	40.1	6.09
512Mb	10	16.1	37.3	11.2
128Mb	50	263.7	127.1	23.2
256Mb	50	72.7	148.2	24.8
512Mb	50	31.7	125.9*	20.3

Como se menciona en[Smith10], generalmente agregando más memoria RAM incrementará el desempeño, sin embargo hay situaciones donde no se mejorará, como en el caso en el que la cantidad de datos sea demasiado pequeña. En otro caso cuando hay que hacer muchas lecturas, el mejoramiento se obtiene al usar discos más rápidos en lugar de incrementar la cantidad de memoria.

3.5. Experimentos realizados inhabilitando la memoria caché

Hasta aquí se han realizado una serie de experimentos sobre distintos escenarios de configuración de software, pero se tiene la necesidad de realizar estudio acerca del comportamiento cuando alguna variable de hardware utilizado sea manipulada. Se sabe que: el mayor tiempo de respuesta es consumido al realizar las correspondientes

lecturas en el dispositivo de almacenamiento secundario, que en el caso de este trabajo es un disco magnético, pero no se debe dejar de lado que la velocidad de procesamiento del CPU, así como la memoria principal y su velocidad también impactan en el tiempo de respuesta.

Con lo cual se decidió llegar hasta el más alto nivel en la jerarquía de memoria y realizar pruebas de desempeño, anteriormente se expusieron resultados cuando la cantidad de memoria principal asignada al SMBD aumenta, de la cual se describieron los resultados arrojados; en este caso se quiere conocer de manera experimental: ¿cual es el impacto de la presencia/ausencia de memoria caché en los tiempo de respuesta?. Conocer el desempeño en estas condiciones abre un campo de oportunidad en la cuestión de: conocer la cantidad óptima de memoria caché que se requiera para un equipo que alberga un SMBD.

En esta parte se realizó por medio de software la inhibición de la memoria caché2.12. Para realizar esto se recurre al manual de Intel, en el cual se indica la forma en la que se debe realizar de acuerdo a la arquitectura del CPU. Primeramente se debe mover el registro del control CR0 a un registro de propósito general EAX, en este registro se realizará la modificación del bit que CD[4] que se encuentra en la posición 30 de su valor 0 a 1.

```
1 asm("push %rax");                          //; save eax
2 asm("cli");                                        //;
   disable interrupts while it do this
3 asm("movq %cr0, %rax");              //; read CR0
4 asm("or  $0x40000000, %rax");//; set CD bit of CR0
5 asm("movq %rax, %cr0");              //; cache is now disabled
6 asm("wbinvd");                             //flush
7 asm("or $0x20000000, %rax");//; now set the NW bit
8 asm("movq %rax, %cr0");//; turn off the cache entirely
9 asm("pop %rax");          //; restore eax
```

[4]Cache disable, activar desactivar globalmente la memoria caché

Resultados obtenidos

En esta tabla se muestran una serie de ejecuciones, en ellas se mide el desempeño en los dos modos, estas pruebas se realizaron sobre dos SMBDs, los cuales se instalaron en un sistema operativo Debian Linux, en el cual se pudieron manipular los bits del registro de control, ver apéndice: D. Se obtuvo el comportamiento que a continuación se muestra.

Tabla 3.4: Pruebas con manipulación de caché.

Ejecución\Escenario	PG	PG s/c	MyS	MyS s/c
select L_ORDERKEY from LINEITEM group by L_ORDERKEY order by L_ORDERKEY	4.4	7.33	12.35	58
select distinct l_PARTKEY from LINEITEM group by l_PARTKEY order by L_PARTKEY	2.3	2.85	5.47	33.6
select avg(l_PARTKEY) from LINEITEM group by L_PARTKEY	3.9	10.43	7.156	11.8
select max(l_PARTKEY) from LINEITEM group by L_PARTKEY	2.85	5.04	5.56	8.85
select min(l_PARTKEY) from LINEITEM group by L_PARTKEY	3.1	4.91	5.63	14.11
select atan(avg(l_PARTKEY)) from LINEITEM group by L_PARTKEY	3.65	18	7.25	57.65
select max(atan(l_PARTKEY)) from LINEITEM	2.25	3.96	3.38	6.34
select max(atan(l_PARTKEY)) from LINEITEM group by L_ORDERKEY	5	21.7	32.61	28.8

PG: Postgres en ejecución normal, PG s/c: Postgres sin caché, MyS:MySQL ejecución normal, MyS s/c: MySQL sin caché.

Estas pruebas se hicieron con funciones de agregación las cuales requieren más procesamiento en memoria2.9, esto debido a que no se limita a tareas de recuperación de información y envío para consumo de otra aplicación o presentar a usuario. Lo que hace al tener los datos en memoria es implementar la operación correspondiente, dicha operación requiere procesamiento en memoria, que al incluir un paso de ejecución extra impacta en el tiempo global de ejecución. La presencia de memoria caché debe

tener un efecto de disminución de acceso a los datos por parte del procesador en esta tarea, luego entonces se midió que efectivamente ocurriera. Por lo cual en base a los resultados obtenidos se concluye que: un factor en el desempeño de ejecución de operaciones dentro de un SMBD es efectivamente la memoria caché.

Experimentos usando una configuración de hardware diferente

Estas mismas pruebas se realizaron en una computadora con diferentes características: procesador intel Core Duo2, 2gb de memoria Ram DDR2, almacenamiento en disco de 120gb 5400rpm, igualmente se obtuvieron los dos escenarios de ejecución con y sin acceso a memoria caché. Aquí se muestran los resultados.

Tabla 3.5: Hardware diferente

Ejecución\Escenario	PG	PG s/c	MyS	MyS s/c
select * from LINEITEM group by L_ORDERKEY order by L_ORDERKEY	16.4	205	41.4	186
select distinct l_PARTKEY from LINEITEM group by l_PARTKEY order by L_PARTKEY	35.8	42.1	19.3	218.8
select avg(l_PARTKEY) from LINEITEM group by L_PARTKEY	29.9	38.8	21.1	87.5
select max(l_PARTKEY) from LINEITEM group by L_PARTKEY	30.5	39.2	13.7	157.9
select min(l_PARTKEY) from LINEITEM group by L_PARTKEY	31.2	40.4	12.9	117.8
select atan(avg(l_PARTKEY)) from LINEITEM group by L_PARTKEY	28.6	44.4	16.7	92
select max(atan(l_PARTKEY)) from LINEITEM	5.2	5.6	7.2	52.8
select max(atan(l_PARTKEY)) from LINEITEM group by L_ORDERKEY	23.2	41.1	17.2	35.6

PG: Postgres en ejecución normal, PG s/c: Postgres sin caché, MyS:MySQL ejecución normal,MyS s/c: MySQL sin caché.

3.6. Síntesis de factores estudiados.

A continuación y de acuerdo a los resultados obtenidos, se muestran en síntesis los factores que incidieron en el desempeño de las operaciones dentro del sistema manejador de base de datos, dentro de los límites para los cuales fue acotado éste trabajo.

- Se mostró que al manipular los niveles de aislamiento 2.9.2, se obtienen cambios en el tiempo de ejecución con sentencias en el SMBD, pero en algunos gestores afectó la visualización arrojando resultados inconsistentes. Se obtiene decremento de tiempo de ejecución al utilizar niveles de aislamiento menos restrictivos, esto también se probó con ejecución de operaciones concurrentes.

- Se experimentó con transacciones concurrentes 2.9, obteniendo los resultados que se mostraron anteriormente, se observa que se generan deadlocks, lo que hace abortar algunas de las operaciones. La combinación de lecturas con escrituras simultaneas y el tener que cumplir con las propiedades de las transacciones, provoca que se acumule el tiempo de ejecución del grupo de sentencias.

- Cada SMBD implementa de diferente manera sus algoritmos para realizar las operaciones, por lo que se obtienen diferentes tiempos de respuesta para mismas condiciones de hardware con sistema operativo en una misma consulta 2.2.

- La velocidad de E/S en el dispositivo de almacenamiento permanente es la que más impacta en el rendimiento de un SMBD, ya que los tiempos de acceso son en el orden de los mili-segundos, mientras que en memoria principal son de nano-segundos [Garcia-Molina09].

- La disposición y velocidad de acceso a memoria principal tiene efecto en el tiempo de ejecución para tareas aritméticas, funciones de agregación e impresión de datos.

Capítulo 4

Conclusiones

4.1. Conclusiones Generales

Al haber realizado este trabajo se tiene lo siguiente:

Existe una gran cantidad de algoritmos implementados dentro de los sistemas gestores de bases de datos, los cuales son el resultado de años de investigación en el área.

Las transacciones son operaciones que en un sistema gestor de base de datos causan un impacto en el desempeño cuando crece el número de peticiones concurrentes, ya que el SMBD debe hacer cumplir las propiedades de la misma.

La disposición de más recursos de hardaware para mismas condiciones de software y una misma complejidad de la operación, disminuyen el tiempo de ejecución.

Los niveles de aislamiento garantizan la visibilidad de los datos, en las pruebas que se hicieron, se sacrificó esta propiedad con niveles de aislamiento de menor restricción, con estas pruebas se obtuvieron tiempos de respuesta más cortos.

Aunque la complejidad de una operación de consulta de bases de datos es impactada mayormente por la velocidad de escritura a dispositivo de almacenamiento, se observó que la memoria igualmente puede provocar atrasos, la disponibilidad de memoria caché en especial para operaciones que se realizan en memoria tiene un efecto sobre el desempeño del sistema manejador.

4.2. Trabajos Futuros

Los trabajos que posteriormente se pueden realizar son en realidad muchos, a continuación se listan algunos de ellos.

1. Realizar análisis del comportamiento con bases de datos en memoria (Por sus siglas en inglés IMBD) con hardware de alto desempeño y utilizando datos en el orden de los teabyes.

2. Efectuar un análisis en bases de datos distribuidas trabajando con diferentes niveles de aislamiento simultaneamente.

3. Someter a pruebas la utilización de discos de estado sólido. Esto podría acercarse a la velocidad de acceso en memoria principal sin perder la capacidad de almacenamiento permanente.

4. Determinar el tamaño óptimo de caché de disco que permita la mejor relación desempeño/costo.

Referencias

[Borthakur13] Borthakur, D. Petabyte scale databases and storage systems at facebook. *En Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, págs. 1267–1268. ACM, New York, NY, USA, 2013. ISBN 978-1-4503-2037-5. doi:10.1145/2463676.2463713.
URL `http://doi.acm.org/10.1145/2463676.2463713`

[Chaudhuri98] Chaudhuri, S. An overview of query optimization in relational systems. *En Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '98, págs. 34–43. ACM, New York, NY, USA, 1998. ISBN 0-89791-996-3. doi:10.1145/275487.275492.
URL `http://doi.acm.org/10.1145/275487.275492`

[Coronel09] Coronel, C., Morris, S., y Rob, P. *Database Systems: Design, Implementation, and Management.* Course Technology Press, Boston, MA, United States, 9ª ed$^{\text{ón}}$., 2009. ISBN 0538469684, 9780538469685.

[Duggan11] Duggan, J., Cetintemel, U., Papaemmanouil, O., y Upfal, E. Performance prediction for concurrent database workloads. *En Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, págs. 337–348. ACM, New York, NY, USA, 2011. ISBN 978-1-4503-0661-4. doi:

10.1145/1989323.1989359.
URL http://doi.acm.org/10.1145/1989323.1989359

[Elmasri10] Elmasri, R. y Navathe, S. *Fundamentals of Database Systems*. Addison-Wesley Publishing Company, USA, 6ª ed^{ón}., 2010. ISBN 0136086209, 9780136086208.

[Garcia-Molina09] Garcia-Molina, H., Ullman, J. D., y Widom, J. *Database systems - the complete book (2. ed.)*. Pearson Education, 2009. ISBN 978-0-13-187325-4.

[Graefe93] Graefe, G. Query evaluation techniques for large databases. *ACM Comput. Surv.*, 25(2):73–169, jun. 1993. ISSN 0360-0300. doi: 10.1145/152610.152611.
URL http://doi.acm.org/10.1145/152610.152611

[Grechanik12] Grechanik, M. The curse of database deadlocks: the problem with no good solution. *SIGSOFT Softw. Eng. Notes*, 37(5):13–14, sep. 2012. ISSN 0163-5948. doi:10.1145/2347696.2347719.
URL http://doi.acm.org/10.1145/2347696.2347719

[Hellerstein07] Hellerstein, J. M., Stonebraker, M., y Hamilton, J. *Architecture of a Database System*. Now Publishers Inc., Hanover, MA, USA, 2007. ISBN 1601980787, 9781601980786.

[Henderson08] Henderson, H. *Encyclopedia of Computer Science and Technology*. FACTS ON FILE, 2008. ISBN 0816063826.

[Ioannidis96] Ioannidis, Y. E. Query optimization. *ACM Comput. Surv.*, 28(1):121–123, mar. 1996. ISSN 0360-0300. doi:10.1145/234313.234367.
URL http://doi.acm.org/10.1145/234313.234367

[Kossmann00] Kossmann, D. The state of the art in distributed query processing. *ACM Comput. Surv.*, 32(4):422–469, dic. 2000. ISSN 0360-0300.

doi:10.1145/371578.371598.
URL http://doi.acm.org/10.1145/371578.371598

[Liu09] Liu, L. y Zsu, M. T. *Encyclopedia of Database Systems*. Springer Publishing Company, Incorporated, 1ª edón., 2009. ISBN 0387355448, 9780387355443.

[Ngamsuriyaroj10] Ngamsuriyaroj, S. y Pornpattana, R. Performance evaluation of tpc-h queries on mysql cluster. *En Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, págs. 1035 –1040. april 2010. doi: 10.1109/WAINA.2010.167.

[Ordonez10] Ordonez, C. y García-García, J. Evaluating join performance on relational database systems. *JCSE*, 4(4):276–290, 2010.

[Osman12] Osman, R. y Knottenbelt, W. J. Database system performance evaluation models: A survey. *Perform. Eval.*, 69(10):471–493, oct. 2012. ISSN 0166-5316. doi:10.1016/j.peva.2012.05.006.
URL http://dx.doi.org/10.1016/j.peva.2012.05.006

[Ozsu91] Ozsu, M. T. y Valduriez, P. *Principles of Distributed Database Systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991. ISBN 0-13-691643-0.

[Schwartz12] Schwartz, B., Zaitsev, P., y Tkachenko, V. *High Performance MySQL: Optimization, Backups, and Replication*. O'Reilly Media, Inc., 3ª edón., 2012. ISBN 1449314287, 9781449314286.

[Shankar12] Shankar, S., Nehme, R., Aguilar-Saborit, J., Chung, A., Elhemali, M., Halverson, A., Robinson, E., Subramanian, M. S., DeWitt, D., y Galindo-Legaria, C. Query optimization in microsoft sql server pdw. *En Proceedings of the 2012 ACM SIGMOD International*

Conference on Management of Data, SIGMOD '12, págs. 767–776. ACM, New York, NY, USA, 2012. ISBN 978-1-4503-1247-9. doi:10.1145/2213836.2213953.
URL `http://doi.acm.org/10.1145/2213836.2213953`

[Silberschatz05] Silberschatz, A., Korth, H. F., y Sudarshan, S. *Database System Concepts, 5th Edition*. McGraw-Hill Book Company, 2005. ISBN 978-0-07-295886-7.

[Smith10] Smith, G. *PostgreSQL 9.0 High Performance*. Packt Publishing, Limited, 2010. ISBN 9781849510301.
URL `http://books.google.ch/books?id=gCi3cQAACAAJ`

[Teorey11] Teorey, T. J., Lightstone, S. S., Nadeau, T., y Jagadish, H. V. *Database Modeling and Design: Logical Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5ª ed$^{\underline{\text{ón}}}$., 2011. ISBN 0123820200, 9780123820204.

Apéndice A

Código para ejecución concurrente

consultas.java

```
1  /**
2   *
3   * @author Isaias
4   */
5  /*
6   * To change this template, choose Tools | Templates
7   * and open the template in the editor.
8   */
9
10 public class consultas{
11         int countQuery=0;
12         Thread fifo[];//fifo
13         public String arriveSequence="";
14         short maxNumThreads;//Num consultas/inserciones
               simultaneas
15         long tiempo=0;
16         String listaSQLInterf[]=new String[2];//resultado
               de interfaz.
```

```
17          String sentenciaL,sentenciaE;
18
19          short serverSelected;
20          short nivelSelected;
21
22  public consultas() {//En modo main
23          maxNumThreads=5;
24          fifo=new Thread [maxNumThreads];
25          tiempo=System.currentTimeMillis();
26      }
27  public consultas(short nHilos, short serverSelected,
       String sentenciaL, String sentenciaE, short
       nivelSelectedInterf) {//invocado desde win.java
28          maxNumThreads=nHilos;
29          this.serverSelected=serverSelected;
30          this.sentenciaL=sentenciaL;
31          this.sentenciaE=sentenciaE;
32          this.nivelSelected=nivelSelectedInterf;
33
34          tiempo=System.currentTimeMillis();
35          fifo=new Thread [maxNumThreads];
36          armaSentencias();
37      }
38  void armaSentencias(){
39      listaSQLInterf[0]=sentenciaL;
40      listaSQLInterf[1]=sentenciaE;
41  }
42
43      public static void main(String[] args) {
44          consultas sharedQuery=new consultas();
```

```
45
46          sharedQuery.crearHilos();
47          sharedQuery.LanzaHilos();//threads ya creados son
               ejecutados
48      }
49
50      public void resultThreads(){
51        //  System.out.println("\n=========\nOrden en que
             fueron creados:\n"
52            //      +arrive Sequence+"\n______\nUso del
                 recurso:\n");
53      }
54
55  public void crearHilos(){
56
57      while(countQuery<maxNumThreads){
58                              fifo[countQuery]=new querySQL(
                                   countQuery,listaSQLInterf,
                                   serverSelected, nivelSelected
                                   );//queue
59                  arriveSequence+="\nNueva consulta: "+
                       countQuery+" ha llegado";
60                  countQuery++;
61              }
62    }
63      /**Metodo que hace start() en los hilos ya creados,
          esto de forma aleatoria
64      para que compitan por el recurso toilet **/
65  public void LanzaHilos(){
66      for(Thread PilaConsultass:fifo)
```

```
67          {PilaConsultass.start();}
68
69  boolean someAlive=true;
70      while(someAlive){//mientras NO todos hayan parado
71              someAlive=false;//presupone que ya todos
                    pararon
72              for (Thread someThread: fifo)
73                  if(someThread.isAlive())
74                      someAlive=true; //encontro algun vivo,
                            continua ciclo
75          }
76      resultThreads();//Finalmente imprime resultado de
            ejecucion
77     tiempo=System.currentTimeMillis()-tiempo;
78      System.out.println(maxNumThreads+" sentencias; Tiempo
            total: "+tiempo+"ms");
79      }
80  }
81  }
```

Apéndice B

Ejecución aleatoria de consultas

querySQL.java Codigo que ejecuta aleatoriamente alguna consulta de acuerdo al servidor, nivel de aislamiento

```
1  import java.util.Random;
2  /*
3   * To change this template, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  /**
7   *
8   * @author ZJ
9   */
10 public class querySQL extends Thread {
11     int valRandomUpd=new Random().nextInt(500);
12     int numThread;
13     int difQueryPropuestas;
14     short serverSelected;//tres opciones
15     short nivelSelected;
16        String listaSQL[];
17
```

```
18      public querySQL(int numThread, String listaSQLInterf
           [], short serverSelected, short  nivelSelected) {
19          this.numThread=numThread;
20
21          this.listaSQL=listaSQLInterf;//Vendran de la
              interfaz
22          difQueryPropuestas=listaSQL.length;
23
24          this.serverSelected=serverSelected;
25          this.nivelSelected=nivelSelected;
26      }
27      public void run(){
28
29
30            Conexion conBase;
31            conBase=new Conexion(nivelSelected);
32            valRandomUpd=new Random().nextInt(500);
33          try{
34            switch(serverSelected){
35          case 0:conBase.conectaDB('s',"jdbc:jtds:sqlserver
               ://localhost", "TPCH1;intance=", "sa", "***");
               //sql server
36              break;
37          case 1:conBase.conectaDB('m',"jdbc:mysql://
               localhost", "TPCH", "root", "***");//mysql
38              break;
39          case 2:conBase.conectaDB('p',"jdbc:postgresql://
               localhost", "tpch1", "postgres", "***");//
               postgres
40              break;
```

```
41          default:break;//No selecciono boton
42          }
43
44              try {sleep((long)(Math.random() * 1000)); }
45              catch (InterruptedException e){}
46               String []tupla;
47              String tuplaPrint="";
48                  //Elige aleatoriamente alguna de las 'n'
                       sentencias
49
50                                  int nSQL=new Random().
                                       nextInt(10);//
51                                  if(nSQL<6)//60%
52                                          nSQL=0;
53                                  else//30%
54                                          nSQL=1;
55
56              try{
57                 // System.out.println("------------\
                       nworking with SMBD thread: "+numThread);
58                 // System.out.println(" ++++++++++++++++
                       consulta: "+nSQL);
59                  if(nSQL==0){//consulta 1
60                      tupla=conBase.queryRow(listaSQL[0],1);
61
62                     for (String tmpString:tupla)//Formatean
                           la salida
63                          {tuplaPrint+=" | "+tmpString.
                               toString();}
64                     if(nivelSelected!=0)
```

```
65                          {conBase.con.rollback();}
66                  System.out.println("Resultado consulta: \n
                       \t"+tuplaPrint);
67                  }
68                  else//consulta 1
69                  {
70                      tuplaPrint=""+conBase.executeUpdate(
                           listaSQL[1]);//Retorna 'int'
71                      if(nivelSelected!=0)
72                          {conBase.con.rollback();}
73                  System.out.println("Resultado
                       Actualizacion:\t"+tuplaPrint);
74                  }
75                  }catch(Exception eConecta){ eConecta.
                       printStackTrace();//System.out.println(
                       listaSQL[nSQL]);
76                   }
77
78              try {sleep((long)(Math.random() * 100)); }
79                  catch (InterruptedException e){}
80                  finally{
81                  }//end sleep
82               }
83          catch(Exception eConecta){eConecta.printStackTrace
               ();
84          //System.out.println("-cerrado-Fallo: "+numThread)
               ;
85
86          //sharedQuery.conBase.cerrar();
87          }
```

```
88          finally{//System.out.println("-cerrado-Normal: "+
               numThread);
89          conBase.cerrar();
90          }
91  }
92  }
```

Apéndice C

Interfaz de usuario

win.java Esta clase fue creada para hacer automatizada la ejecución de consultas concurrentes, se obtiene flexibilidad al cambiar entre niveles de aislamiento, servidor de BD y número de hilos de ejecución.

```
1  import java.awt.*;
2  import java.awt.event.MouseAdapter;
3  import java.awt.event.KeyEvent;
4  import java.awt.event.KeyListener;
5  import javax.swing.text.MaskFormatter;
6  import javax.swing.*;
7  import javax.swing.event.*;
8  public class win extends JFrame implements ChangeListener{
9      String listaSGBD[]={"SQL Server","MySQL","PostgreSQL"
           };
10     Short serverSelected=-1;//toma valores 0,1,2; sqlser,
           mysq, postgres
11     Short nivelSelected =0; //0=ninguno,
           TRANSACTION_READ_UNCOMMITTED,
           TRANSACTION_READ_COMMITTED,
           TRANSACTION_REPEATABLE_READ,
```

```
        TRANSACTION_SERIALIZABLE
12      private JRadioButton sqlserverOp,mysqlOp,postgresOp;
13      private ButtonGroup bg;
14      private JTextArea sentenciaE,sentenciaL;
15      private javax.swing.JComboBox nivelesAislamiento;
16
17      javax.swing.JScrollPane  jScrollPane1,jScrollPane2 =
           new javax.swing.JScrollPane();
18      JButton runQuery = new JButton("Run Query");
19      JFormattedTextField nHilos= new JFormattedTextField(
           createFormatter("###"));
20      JSeparator lineaS= new JSeparator(JSeparator.
           HORIZONTAL);
21
22      KeyListener keyListener = new KeyListener() {//Ttext
           area
23          public void keyPressed(KeyEvent keyEvent) {    }
24
25            public void keyReleased(KeyEvent keyEvent) {
26            runQuery.setEnabled(verificarTodasLasEntradas
                 ());
27            }
28        public void keyTyped(KeyEvent keyEvent) {     }
29      };
30
31      public win() {
32          super("Experimentos");
33          setLayout(null);
34
35          setDefaultCloseOperation(javax.swing.
```

```
            WindowConstants.EXIT_ON_CLOSE);
36          bg=new ButtonGroup();
37          sqlserverOp=new JRadioButton("SQL Server");
38                      sqlserverOp.setBounds
                          (10,20,100,30);
39                      sqlserverOp.addChangeListener(this
                          );
40                      add(sqlserverOp);
41          bg.add(sqlserverOp);
42          mysqlOp=new JRadioButton("My SQL");
43                      mysqlOp.setBounds(10,45,100,30);
44                      mysqlOp.addChangeListener(this);
45                      add(mysqlOp);
46          bg.add(mysqlOp);
47          postgresOp=new JRadioButton("Postgres");
48                      postgresOp.setBounds(10,70,100,30)
                          ;
49                      postgresOp.addChangeListener(this)
                          ;
50                      add(postgresOp);
51          bg.add(postgresOp);
52
53                      lineaS.setBounds(10, 105, 600,
                          300);
54          add(lineaS, BorderLayout.LINE_START);
55
56                      JLabel label2=new JLabel("
                          Sentencia de lectura:");
57                      label2.setBounds(10,110,200,30);
58                  add(label2);
```

```
59                          sentenciaL=new JTextArea(null, 10,
                                15);//transaction...selec
60                          sentenciaL.addKeyListener(
                               keyListener);
61                          jScrollPane1=new JScrollPane(
                               sentenciaL);
62                          jScrollPane1.setBounds
                               (10,140,600,150);
63          add(jScrollPane1);
64                  JLabel label3=new JLabel("Sentencia de
                       escritura:");
65                          label3.setBounds(10,290,200,30);
66                  add(label3);
67                          sentenciaE=new JTextArea(null, 10,
                                15);
68                          sentenciaE.addKeyListener(
                               keyListener);
69                          jScrollPane2=new JScrollPane(
                               sentenciaE);//tupdate..
70                          jScrollPane2.setBounds
                               (10,320,600,100);
71          add(jScrollPane2);
72
73                  JLabel label=new JLabel("Numero de threads
                       :");
74                          label.setBounds(140, 20, 200, 20);
75                  add(label);
76
77                          nHilos.setBounds(260,20,40,20);
78                          nHilos.setText("000");
```

```
79                          add(nHilos);
80                          nivelesAislamiento=new  JComboBox
                               ();
81                          nivelesAislamiento.setModel(new
                               javax.swing.
                               DefaultComboBoxModel(new String
                               [] {
82                              "No transaction","
                                   TRANSACTION_READ_UNCOMMITTED
                                   ", "
                                   TRANSACTION_READ_COMMITTED"
                                   , "
                                   TRANSACTION_REPEATABLE_READ
                                   ","TRANSACTION_SERIALIZABLE
                                   " }));
83                          nivelesAislamiento.
                               setKeySelectionManager(null);
84                          nivelesAislamiento.addItemListener
                               (new java.awt.event.
                               ItemListener() {
85
86                              @Override
87                              public void itemStateChanged(
                                   java.awt.event.ItemEvent
                                   evt) {
88                              nivelSelected=(short)
                                   nivelesAislamiento.
                                   getSelectedIndex();
89
90                              }});
```

```
91                          nivelesAislamiento.setBounds
                               (320,20,290,20);
92                          add(nivelesAislamiento);
93
94                          runQuery.setEnabled(false);
95                          runQuery.setBounds(510,50,100,20);
96                  runQuery.addMouseListener(new java.awt.
                       event.MouseAdapter() {
97                        public void mouseClicked(java.awt.
                             event.MouseEvent evt) {
98
99                      if (verificarTodasLasEntradas())
100                             {ejecuarConsultas();}
101                     }
102                     });
103           add(runQuery);
104     }
105 void ejecuarConsultas(){
106     short nHilos=Short.parseShort(this.nHilos.getText());
107
108         consultas sharedQuery=new consultas( nHilos,
               serverSelected,  sentenciaL.getText(),
               sentenciaE.getText(), nivelSelected);
109         System.out.println("\nSe ejecutarÃ¡ con nivel de
               aislamiento: "+nivelesAislamiento.
               getSelectedItem().toString()
110                  +"\tServidor: "+listaSGBD[serverSelected])
                        ;
111         sharedQuery.crearHilos();
112         sharedQuery.LanzaHilos();
```

```
113 }
114
115     public void stateChanged(ChangeEvent e) {//los
            radioButtons
116         runQuery.setEnabled(verificarTodasLasEntradas());
117     }
118 protected MaskFormatter createFormatter(String s) {
119                 MaskFormatter formatter = null;
120                 try {
121                         formatter = new MaskFormatter(s);
122                 } catch (java.text.ParseException exc) {
123                         System.err.println("formatter is
                            bad: " + exc.getMessage());
124                         System.exit(-1);
125                 }
126                 return formatter;
127 }//end method
128
129 boolean verificarTodasLasEntradas(){
130         String caja=sentenciaE.getText();
131
132         if(caja.length()==0){
133             return false;}
134         caja=sentenciaL.getText();
135          if(caja.length()==0){
136             return false;}
137
138          caja=nHilos.getText();
139          if(caja.length()==0)
140             return false;
```

```
141 //Si hasta aqui uno estÃ¡ selecc retorna true
142         if (sqlserverOp.isSelected()) {
143             serverSelected=0;
144             return true;
145         }
146         if (mysqlOp.isSelected()) {
147             serverSelected=1;
148             return true;
149         }
150         if (postgresOp.isSelected()) {
151             serverSelected=2;
152             return true;
153         }
154             return false;//ningun botton estaba selecc
155         }//end method
156 
157     public static void main(String[] ar) {
158         win formulario1=new win();
159         formulario1.setBounds(0,0,640,470);
160         formulario1.setResizable(false);
161         formulario1.setVisible(true);
162     }
163 }
```

Apéndice D

Código para ejecución en modo cache habilitado/inhabilitado

noCache.java

```
1  import java.sql.*;
2  public class noCacheLote{
3
4          private final String DRIVER = "com.mysql.jdbc.
               Driver"; String port=":3306";//Mysql
5          //private final String DRIVER = "org.postgresql.
               Driver";String port="";//postgres
6          String servidor;
7          String base;
8          String usuario;
9          String password;
10         public Connection con;
11     private Statement stmt;
12     private ResultSet rs;
13     private int tiempo[]=new int[100];
14         String []listaSQL={
```

```
15          "select * from LINEITEM group by L_ORDERKEY order
              by L_ORDERKEY",
16          "select  distinct l_PARTKEY from LINEITEM group by
               l_PARTKEY order by L_PARTKEY;",
17          "select avg(l_PARTKEY) from LINEITEM group by
              L_PARTKEY;",
18          "select max(l_PARTKEY) from LINEITEM group by
              L_PARTKEY;",
19          "select min(l_PARTKEY) from LINEITEM group by
              L_PARTKEY;",
20          "select ATAN(avg(l_PARTKEY)) from LINEITEM group
              by L_PARTKEY;",
21          "select MAX(ATAN(l_PARTKEY)) from LINEITEM;",
22          "select MAX(ATAN(l_PARTKEY)) from LINEITEM group
              by L_ORDERKEY;"};
23  public static void main(String lote[]){
24                  new noCacheLote().lanzar();
25  }
26
27  public void lanzar(){
28          try{
29                  conecta("jdbc:mysql://localhost", "TPCH",
                      "root", "***");//mysql
30                  //conectaDB("jdbc:postgresql://localhost",
                       "tpch1", "postgres", "***");//postgres
31                  }catch(Exception eConecta){ eConecta.
                      printStackTrace();return;}
32
33                  String []tupla=new String[100*2];//
                      Resultados
```

```
34                  String algunQuerySimple;
35                  int cont=0;
36                  try{
37                          for (String sqlQuery:listaSQL ){
38                                  tiempo[cont]=(int)System.
                                       currentTimeMillis();
39                                        tupla=queryRow(
                                             sqlQuery,1);//
                                             Una consulta
40                                  tiempo[cont]=(int)System.
                                       currentTimeMillis()-
                                       tiempo[cont];//nuevo -
                                       viejo
41                                  algunQuerySimple=
                                       querySimple("FLUSH
                                       QUERY CACHE;");
42                                  algunQuerySimple=
                                       querySimple("FLUSH
                                       TABLES;");
43                          cont++;
44                          }
45                  }catch(Exception eConecta){ eConecta.
                        printStackTrace();
46                      }
47                  finally{
48          cerrar();
49          }
50                  int u;
51                  for(u=0;u<listaSQL.length;u++){
52                          System.out.println(u+" "+tiempo[u
```

```
                                ]);
53                  }
54  }
55
56          public void conecta(String servidor, String base,
               String usuario, String password)throws
               Exception {
57          try {
58              Class.forName(DRIVER).newInstance();
59              con=DriverManager.getConnection(servidor +port+
                  "/" + base, usuario, password);
60                          stmt = con.createStatement(
                               ResultSet.
                               TYPE_SCROLL_INSENSITIVE,
                               ResultSet.CONCUR_READ_ONLY);
61
62          } catch (Exception e) {
63              throw e;
64          }
65          }//End method conecta
66          public String[] queryRow(String query, int numArgs
               ) {
67          try {
68              rs = stmt.executeQuery(query);
69              ResultSetMetaData rsmd = rs.getMetaData();
70              String[] datos = new String[rsmd.
                   getColumnCount()];
71              int i = 0;
72              while (rs.next()) {
73                  while (i < numArgs) {
```

```
74                      datos[i] = rs.getString(i + 1);
75                      i++;
76                  }
77              }
78              return datos;
79          } catch (Exception e) {
80              e.printStackTrace();
81              return null;
82          }
83      }//End method queryRow
84          public String querySimple(String query) {
85          try {
86              rs = stmt.executeQuery(query);
87                  return rs.toString();
88          } catch (Exception e) {
89              e.printStackTrace();
90              return null;
91          }
92      }//end query simple
93           public void cerrar() {
94          try {
95              this.con.close();
96              this.con.setAutoCommit(true);
97          } catch (Exception e) {  }
98      }//End cerrar()
99  }
```