VNIVERSIDAD NACIONAL
AVTONOMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERIA DE LA COMPUTACION

CONSTRUCCIÓN Y EVALUACIÓN DE UNA HERRAMIENTA QUE IMPLEMENTA AGREGACIONES HORIZONTALES EN SQL.

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN CIENCIAS (COMPUTACIÓN)

PRESENTA:

ROGELIO MONTERO CAMPOS

TUTOR: DR. JAVIER GARCÍA GARCÍA
FACULTAD DE CIENCIAS
UNAM

MÉXICO, D. F. MARZO 2014.

Agradecimientos

Para Lucy
"Estoy seguro que serás muy feliz por que mi mayor satisfacción será ver que logres todo lo que te propongas."

Agradezco a todas las personas que me ayudaron a lograr esta meta en mi vida profesional, a todos lo que me dieron alguna palabra de apoyo o que siempre confiaron en mí. Especialmente a mis padres Rogelio y Angelina que me dieron todo su apoyo y confianza en todo lo que me he propuesto, estos valores que ahora me toca transmitir y enseñar. Al amor de mi vida: Tere, por apoyarme como nadie, dándome la confianza y tiempo que requerí para terminar este trabajo tan importante para mí, para ella mi más sincera admiración y agradecimiento. Quiero agradecer también a tus padres Mercedes y Bernabe por apoyar a nuestra nueva familia y por todos sus valiosos consejos.

A todos mis viejos amigos que conforme pasan los años se vuelvan más valiosos y me hacen recordar cuando todo esto empezó. Gracias por estar siempre junto a mí.

A mis nuevos amigos de la maestría por compartir esta grata y exhausta experiencia, por todos esos días de estudio, discusión y apoyo mutuo.

Al personal del Posgrado en Ciencia e Ingeniería en Computación por permitirme enfocarme en los estudios, su trabajo es realmente valioso. A los profesores del posgrado que me impartieron clase y que compartieron sus buenas y malas experiencias. A mis sinodales por la revisión de mí investigación, por sus comentarios y observaciones que enriquecieron mi trabajo. A todos ustedes gracias.

Agradezco profundamente al Dr. Javier García y a su familia por abrirme las puertas de su casa y por brindarme su confianza. Gracias por la larga amistad y por exigir siempre mi mejor esfuerzo.

A la Dra. Lourdes García y a todo el equipo de colaboración del INSP por su aprecio y su amistad, ha sido un gusto trabajar con ustedes y aportar mis conocimientos para el avance de nuestra unidad de investigación.

Gracias también a Edith, Ara, Gus, Pau, Mauricio, Cesar, Rodro, Pilar, Edi, Josué, March, Isaias, Roncio, Adri, Adan, Pablo, Lety, Lupita, Inora y a la Dra. Liz por los buenos momentos de los últimos dos años los cuales los hicieron más llevaderos.

Pronostico que en 100 años las computadoras serán dos veces más potentes, diez mil veces más grandes y tan costosas que sólo los cinco reyes más ricos de Europa las tendrán.
– Profesor Frink.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción.

En los últimos años el desarrollo de la tecnología ha permitido que cualquier dispositivo genere o capture datos que por lo regular son almacenados en un Sistema Manejador de Bases de Datos (SMBD) ya que estos sistemas ofrecen características importantes como seguridad, integridad y eficiencia de acceso. Una vez almacenados, los datos deben ser consultados o analizados para proveer información a los usuarios. Dependiendo del tipo de usuario, la búsqueda de información puede ser compleja o sencilla. Las bases de datos operacionales en la actualidad tienden a crecer a un ritmo acelerado lo cual es un factor que aumenta la complejidad para extraer la información.

Las organizaciones invierten gran cantidad de recursos en adquirir o desarrollar distintas herramientas o estrategias enfocadas a la administración y recuperación de conocimiento mediante el análisis de los datos existentes en su empresa. El proceso de descubrimiento del conocimiento consta de siete etapas de manipulación de los datos, las cuales involucran: la Limpieza, la Integración, la Selección, la Transformación, el Minado, la Evaluación y la Presentación.

En el área de descubrimiento del conocimiento, las investigaciones recientes se enfocan en la minería de datos y el desarrollo de algoritmos eficientes para obtener patrones a partir de grandes cantidades de datos. Las etapas de limpieza, integración, selección y transformación, de aquí en adelante referidas como *Preparación de los datos*, involucran muchas veces el conocimiento y uso de herramientas específicas y se requiere un trabajo significativo y un esfuerzo manual por parte del usuario encargado de esta tarea.

Dentro de un SMBD el trabajo de preparación requiere desarrollar consultas complejas que deben de ser eficientes para manipular grandes cantidades de datos. La complejidad para expresar el resultado requerido se puede disminuir por medio de bibliotecas que se encargan de leer y escribir datos en un SMBD y que permiten manipular los datos con más libertad fuera de estos sistemas. Esto resulta costoso en tiempo y se exponen los datos en un entorno donde su integridad y/o seguridad pueden ser violados. Es por ello que se han realizado esfuerzos en integrar herramientas que permitan preparar los datos dentro del SMBD de forma eficiente.

Las funciones definidas por el usuario, los procedimientos almacenados y distintas extensiones de SQL ya se han creado para la preparación de los datos en un SMBD. Incluso con estas herramientas existen operaciones o funciones que no están definidas dentro del lenguaje de consulta SQL como la transposición de los datos ya que no forma parte del modelo relacional.

Las investigaciones sobre estos temas han llevado al desarrollo de herramientas que permitan hacer una manipulación de los datos dentro del SMBD de forma eficiente y de una forma estandarizada [25][3][20]. Tomando como base el estándar del lenguaje de consulta de datos SQL, cada SMBD introduce cláusulas que permiten al usuario tener un mayor poder de consulta, ya sea definiendo nuevos tipos de datos o funciones.

Una reciente investigación [18, 21, 22] se ha planteado el objetivo de definir una extensión de SQL que permita obtener una disposición horizontal de datos a los cuales se les aplica una función de agregación. Dicha investigación se ha enfocado en generar código SQL estándar que permita consultar los datos con una disposición diferente a la acostumbrada en un SMBD. Esta presentación horizontal de los datos a través de funciones de agregación es el tema de estudio del presente trabajo de investigación, cuyos objetivos han sido implementar, experimentar, evaluar, proponer mejoras y exponer ejemplos reales en los cuales se muestre la eficacia y eficiencia de este método de preparación y análisis de datos.

1.1. Objetivos.

Tomando en cuenta la importancia de desarrollar herramientas que permitan preparar los datos dentro del SMBD, la eficiencia es una característica clave pues la cantidad de datos que deben procesar puede ser muy grande. La operación de transposición de datos junto con la agregación de columnas son operaciones importantes ya que permiten obtener una vista tabular y cruzada de los datos la cual es complicada de obtener con cláusulas estándar de SQL. El objetivo central de esta investigación consiste en mejorar una clase novedosa de funciones de agregación a través de la construcción de una herramienta de software tomando como base el prototipo presentado en [22]. A continuación se mencionan los objetivos que, unas vez alcanzados, permitan llegar al objetivo principal.

1. Estudiar sobre la importancia de la preparación de datos dentro de un SMBD.
2. Llevar a cabo un estudio de la historia del arte sobre las agregaciones horizontales.
3. Proponer optimizaciones distintas a las del trabajo existente.
4. Diseñar e implementar una herramienta que se encargue de evaluar consultas con agregaciones horizontales de manera óptima.
5. Experimentar con la herramienta para definir el impacto de las optimizaciones propuestas.
6. Mostrar y ejemplificar la utilidad de estas funciones con datos reales.

1.2. Motivación.

La motivación inicial fue el trabajo de programación que se realizó para desarrollar el software descrito en [22]. En este trabajo se obtuvo un conocimiento profundo acerca de las agregaciones horizontales. Parte de esta motivación fue la oportunidad de colaborar en una investigación científica, seria y de nivel internacional.

Al programar parte del prototipo mencionado surgieron ideas acerca de optimizaciones o mejoras en la ejecución de consultas con agregaciones horizontales. Fue así como se definieron varios objetivos de este trabajo.

Durante la experimentación se buscaron varios escenarios en los cuales además de evaluar y comparar las optimizaciones propuestas se mostrara la utilidad de las funciones de

agregación horizontales, lo cual fue difícil pues la base de datos utilizada tiene un esquema limpio e integrado. Una colaboración con una unidad médica de un Instituto de Salud Público en México (INSP[1]) permitió aplicar o probar la herramienta sobre datos reales, obteniendo resultados favorables para la investigación lo cual motivó más el desarrollo de la parte experimental. Estos resultados se incluyen como parte de la experimentación pues consideramos muy importante la aplicación del producto de esta tesis en un entorno real.

Por último las motivación principal y final de este trabajo es aportar en el desarrollo de las funciones de agregación horizontal mediante su implementación y ejecución óptima mostrando su eficiencia y utilidad en problemas del mundo real.

1.3. Aportación.

Las aportaciones de este trabajo están vinculadas al campo de preparación de los datos para su análisis o minado.

1. Proponer y evaluar optimizaciones a las funciones de agregación horizontales diferentes a las de [18, 21].
2. Crear una herramienta de software para la ejecución optimizada de funciones de agregación horizontales.
3. Experimentar con datos sintéticos y reales que permitan estudiar el desempeño y la utilidad de la herramienta en la preparación de datos dentro de un SMBD.

1.4. Trabajo relacionado.

El tema de la investigación es la preparación de los datos, principalmente la transformación e integración. La preparación de los datos es fundamental en la minería de datos y en la creación de un almacén de datos.

En cuanto a la preparación de los datos, el minado de reglas de asociación es uno de los problemas más recurrentes en OLAP [25], donde las funciones de agregación son utilizadas en el cálculo del soporte de un *itemset* [28]. Existen algoritmos que se han integrado dentro del SMBD como K-MEANS [20] donde la presentación de los datos de forma horizontal facilita su cómputo.

Otras investigaciones se han enfocado en mostrar cómo la obtención de funciones de agregación y una vista tabular de los datos son útiles, por ejemplo, en la creación y manipulación de cubos de datos [8]. La manipulación de cubos de datos también se ha estudiado y tratado de implementar en los sistemas relacionales (ROLAP), un ejemplo de ello son las cláusulas PIVOT y UNPIVOT de SQL SERVER[3].

Un tema con mayor relación es el de agregaciones porcentuales horizontales [19], donde se requiere una agregación en dos niveles de agrupación junto con manejo de división

[1] Instituto Nacional de Salud Pública

de números. Una agregación horizontal puede tratarse como una generalización de éstas funciones y servir como una operación primitiva para calcular estos porcentajes.

Por último se tienen tres trabajos recientes que introducen, experimentan e implementan las funciones de agregación horizontal. El planteamiento, definición y optimizaciones de estas funciones se expone en [18]. La demostración de la equivalencia entre las distintas optimizaciones, su experimentación y evaluación son parte de lo realizado en [21]. El prototipo de un sistema para evaluar consultas con agregaciones horizontales se expone en [22], en él se analiza una consulta, se genera el código SQL correspondiente a alguna de las tres optimizaciones y se despliega al usuario el código SQL generado, el plan de ejecución y alternativamente el resultado.

Capítulo 2
Conceptos y definiciones.

El lenguaje de consulta SQL provee de varias cláusulas que permiten obtener, operar y relacionar los datos para obtener información. Algunas de estas cláusulas son los operadores de agregación o funciones de agregación las cuales obtienen información a partir de conjuntos de registros de las tablas de la base de datos. Estas funciones reciben como parámetro un conjunto de datos y devuelven un solo valor. La importancia y utilidad de estas funciones es amplia pues con ellas se puede obtener información resumida de los datos lo que puede facilitar el análisis al ser un menor número de datos. Las funciones de agregación también han sido útiles en el desarrollo de herramientas de análisis de datos como OLAP donde la creación de cubos está pensada precisamente en la vista resumida de los datos considerando varias dimensiones. En este capítulo se hablará del comportamiento de las funciones de agregación definidas en SQL ya que su estudio nos permitirá comprender y relacionar mejor el concepto de una agregación horizontal, tema central de éste trabajo.

Por otro lado se ha hablado de la manipulación eficiente de los datos y de la capacidad de consulta del lenguaje SQL. A pesar de ello, el modelo relacional no es la única forma en la que se pueden almacenar u operar los datos, es por ello que existen otras herramientas donde la visualización o disposición de los datos es diferente lo cual permite analizarlos también de formas diversas o más eficientes. Una estructura alternativa a la tabla tradicional es una tabla de contingencia donde las filas son marcadas con valores de una o más variables y las columnas son nombradas con valores de otra(s).

El conocimiento puede definirse como aquella información útil y novedosa que permite tomar decisiones. La generación de esta información es un punto importante hoy en día para las organizaciones o empresas que dedican grandes recursos a la búsqueda de esta información.

En los sistemas de cómputo las capacidades de generar y recolectar datos han incrementado rápidamente. Esto se debe a los últimos avances y la explotación de nuevas tecnologías como Internet y los dispositivos móviles. El creciente almacenamiento de los datos ha generado la necesidad de desarrollar nuevas técnicas automatizadas que nos puedan asistir, de forma "inteligente", en transformar las enormes cantidades de datos en información útil.

Esta extracción de conocimiento desde los datos involucra varias disciplinas pero principalmente se basa en el almacenamiento y consulta de datos. Desde hace más de 50 años las bases de datos han evolucionado y han pasado de ser simples repositorios de información a ser uno de los sistemas más complejos e importantes de la actualidad. Desde 1980 la concepción de los almacenes de datos y el procesamiento analítico en línea (OLAP por sus siglas en inglés) han buscado la forma de facilitar la extracción del conocimiento a partir de los datos.

Los datos pueden almacenarse en diferentes clases de bases de datos y repositorios de información los cuales pueden carecer de una estructura homogénea haciendo que la extracción sea más compleja. El análisis efectivo y eficiente de datos no homogéneos se convierte en

una tarea desafiante.

La extracción o descubrimiento del conocimiento a partir de los datos es un proceso que consiste en una secuencia iterativa de pasos que se repite hasta la obtención del conocimiento.

2.1. Funciones de agregación

La agregación significa la unión o adición de una parte a un todo. En bases de datos, la unión de un conjunto de datos puede ser representada por una función aplicada al conjunto, al ser esta función una representación de estos datos se puede definir como una función de agregación. Las funciones de agregación son utilizadas para resumir información a partir de múltiples registros en uno solo. La agrupación es utilizada para crear subgrupos de registros antes de resumir los datos. La agrupación y la agregación son requeridas en muchas aplicaciones por lo que el estándar SQL incorpora las funciones de agregación: COUNT, SUM, MAX, MIN y AVG [6].

La función COUNT regresa el número de registros o valores especificados en una consulta. Las funciones SUM, MAX, MIN y AVG pueden ser aplicadas a un conjunto de valores numéricos. Estas funciones pueden ser utilizadas en las clausulas SELECT y HAVING. Las funciones MAX y MIN pueden ser utilizadas también con atributos de dominios no numéricos si los valores del dominio tienen un orden total entre unos y otros.

Los subgrupos de registros están basados en los valores de algunos atributos, para ello es necesario particionar la tabla en subconjuntos disjuntos de registros, cada grupo o partición consiste en registros que tienen el mismo valor para algún o algunos atributos, llamados atributos de agrupación. Se puede aplicar la función de agregación a cada grupo independientemente para producir información resumida de cada grupo. La cláusula `GROUP BY` especifica los atributos de agrupación, los cuales al aparecer en la cláusula `SELECT` hacen que la aplicación de cada función de agregación a un grupo de registros aparezca junto con el valor de los atributos de agrupación.

Si existen NULOS en un atributo de agrupación, entonces un grupo separado se crea para todos los registros con valor nulo en el atributo de agrupación. (Esto no quiere decir que los valores nulos sean iguales)

Cuando existen NULOS se complica el procesamiento de los operadores de agregación. El estándar dice que el operador SUM debe ignorar los valores NULOS en su entrada. En general las funciones de agregación tratan los NULOS de acuerdo a la siguiente regla: Toda función de agregación excepto COUNT(*) ignora los valores NULOS en su entrada. Como resultado, la entrada puede ser vacía. El conteo de un conjunto vacío es 0 y todos los otros operadores de agregación regresan NULL cuando son aplicados a un conjunto vacío[26].

En OLAP, una *medida* es una propiedad numérica de un cubo multidimensional que captura información que puede resumirse con una agregación. Para su cálculo estas medidas pueden ser categorizadas de acuerdo a la clase de función de agregación utilizada[9].

Una función de agregación f es distributiva si puede ser calculada como sigue:

1. Particionar los datos en n conjuntos.

2. Aplicar la función f a cada partición, obteniendo n valores agregados.

3. Aplicar una función f' a los n valores agregados para obtener el valor agregado final.

Las funciones COUNT(), SUM(), MAX() y MIN() son distributivas ya que pueden ser calculadas de forma distributiva. Si f es SUM, MAX y MIN, $f' = f$, pero si f=COUNT, entonces f'=SUM.

Una función de agregación es algebraica si puede ser calculada por una operación algebraica con M argumentos (donde M es un entero positivo mayor a cero) donde cada argumento es obtenido por una función distributiva. La función AVG() puede ser calculada por la operación algebraica de la división SUM()/COUNT() donde el numerador y denominador son resultado de aplicar funciones de agregación distributivas.

Una función es holística si no existe una función de agregación algebraica con M argumentos (donde M es constante) que caracterice el cálculo. Por ejemplo la funciones *mediana* y *moda* son holísticas.

Esta clasificación de las funciones permite evaluarlas de distintas maneras que pueden ser más eficientes. Por ejemplo, en un sistema paralelo, las funciones distributivas tienen un mejor desempeño pues cada partición puede ser calculada al mismo tiempo.

2.2. Visualización de los datos

El modelo tradicional de un SMBD relacional se basa en tablas, cada una con su propio esquema, en el que cada fila se compone de varias celdas correspondientes a cada variable definida en el esquema de la tabla. Cualquier operación sobre una tabla devuelve una tabla, esto puede limitar o complicar la manipulación de los datos ya que el usuario debe ajustarse a la estructura ofrecida por el sistema.

Una forma distinta de visualizar los datos puede ser una tabla de contingencia. Donde inicialmente para dos variables, una columna contiene los valores de una variable y una fila contiene los valores de otra variable, la intersección contiene el valor de una característica que es compartida por dos variables cuando ellas tienen el valor desplegado en la cabecera de las columna y fila correspondiente [11]. Esta estructura puede extenderse a más de dos variables para lo cual las filas o columnas son agrupadas por valores de atributos adicionales.

Un ejemplo ilustrativo es mostrado en la Figura 2.1 donde los datos originales tienen una visualización tradicional en los SMBD (a) y alternativamente la tabla de contingencia de estos datos (b) los presenta de una forma más comoda para leer. Los valores NULOS representan la no existencia de tuplas con las combinaciones (BORA,Rojo) y (Chevy,Rojo) en la tabla original.

Modelo	Color	Precio
Tsuru	Rojo	20
Tsuru	Azul	40
Pick Up	Azul	30
Pick Up	Rojo	30
Sentra	Rojo	30
Sentra	Azul	50
Bora	Azul	20
Chevy	Azul	40
Chevy	Azul	20

(a): Datos originales

Modelo	Azul	Rojo
Tsuru	40	20
Pick Up	30	30
Sentra	30	50
Bora	20	NULL
Chevy	60	NULL

(b): Tabla de contingencia para los datos originales

Figura 2.1: Ejemplo de visualización de una tabla de contingencia.

Tal disposición alternativa permite al usuario una forma más intuitiva y ordenada de analizar los datos. La transformación entre ambos esquemas ya ha sido estudiada y se han propuesto distintas soluciones. Una de ellas es la cláusula PIVOT que permite obtener una tabla de contingencia aplicando una función de agregación.

```
 SELECT <columna no dinamizada>,
    [primera columna dinamizada] AS <nombre de columna>,
    [segunda columna dinamizada] AS <nombre de columna>
    ...
    [última columna dinamizada] AS <nombre de columna>
FROM
    (<la consulta SELECT que genera los datos>)
    AS <alias de la consulta de origen>
PIVOT
(
    <función de agregación>(<columna que se agrega>)
FOR
[<columna que contiene los valores que se convertirán en encabezados de columna>]
    IN ([primera columna dinamizada], [segunda columna dinamizada]
    ... [última columna dinamizada])
) AS <alias de la tabla dinamizada>
<cláusula ORDER BY opcional>;
```

En esta solución se pueden presentar dos escenarios para los cuales no es muy útil. Al especificar las columnas dinamizadas se tiene que colocar el valor, por lo que esta consulta es susceptible a errores de escritura por parte del usuario que pueden producir resultados indeseados. Otro problema surge cuando la cantidad de columnas a transponer no es un número pequeño, digamos que se necesita transponer una columna que contiene más de 100 valores de nuestro interés. En este caso el usuario deberá conocer y escribir la consulta insertando manualmente estos valores aumentando así la probabilidad de errores.

Adicionalmente a la cláusula PIVOT existe su contraparte, la cláusula UNPIVOT que se encarga precisamente de obtener casi la operación inversa de PIVOT. UNPIVOT no reproduce el resultado de la expresión con valores de tabla original porque las filas se han combinado a través de la función de agregación.

En resumen, la visualización alternativa de los datos es de mucho interés a la comunidad pero es necesario mejorarla o proponer soluciones eficientes y que ayuden al usuario a evitar equivocaciones.

2.3. Proceso de extracción de conocimiento en bases de datos.

En general este proceso involucra distintas etapas que inician con las distintas fuentes de datos y terminan con la presentación del conocimiento obtenido. La Figura 2.2 muestra las etapas principales para la extracción del conocimiento a partir de los datos.

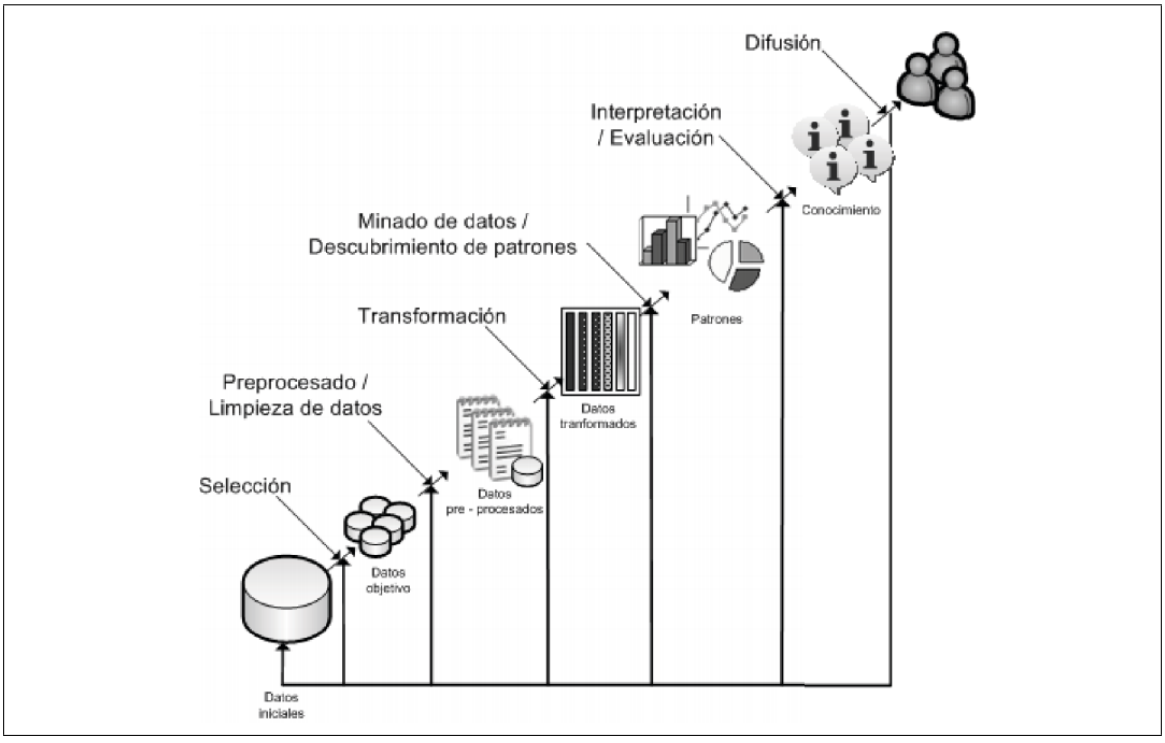


Figura 2.2: Proceso de extracción del conocimiento (Basada en [7]).

1. Selección de los datos: Obtener los datos relevantes para el análisis.
2. Limpieza de los datos: Eliminar datos inconsistentes o ruidosos.
3. Integración de los datos: Combinar múltiples fuentes de datos.
4. Transformación de los datos: Consolidar los datos de una forma apropiada para su análisis o minado.

5. Minería de datos: Procesamiento esencial aplicado a los datos para la extracción de patrones.

6. Evaluación de patrones: Identificar aquellos patrones interesantes y novedosos que representan el conocimiento.

7. Presentación del conocimiento: Técnicas de visualización y representación del conocimiento utilizadas para mostrar los patrones extraídos al usuario.

La minería de los datos es una fase fundamental y se ve influenciada directamente por todas las fases anteriores, en particular por las primeras cuatro fases que se encargan de la preparación de los datos [24]. La etapa de preparación de los datos consume aproximadamente ente el 60 % y 80 % del tiempo utilizado para la extracción del conocimiento [4]. Este fenómeno puede ser atribuido a distintos aspectos como pueden ser:

- La enorme cantidad de datos: Los almacenes de datos consolidan información que puede ser del orden de Gigabytes o Petabytes.
- La cantidad de atributos o campos: Este número aumenta debido a la conjunción de diversas fuentes de datos.
- Semántica de los datos: El significado de cada dato dentro de un repositorio, no necesariamente se preserva durante la integración. Esto involucra una verificación.
- Identificación de datos erróneos: Depende de la semántica definida por el tipo de información que se desea extraer.
- Complejidad de algoritmos de preparación: No todos los algoritmos tienen una complejidad lineal por lo cual impactan en el tiempo total.

El disminuir o eliminar el tiempo utilizado en estas fases del proceso permite enfocarse en otras aspectos que reditúen en una mayor eficiencia o eficacia dentro del descubrimiento del conocimiento.

El procesamiento individual de cada uno de los datos también requiere tener mayor conocimiento sobre cada una de las fuentes de datos el cual muchas veces no es posible de adquirir por los tiempos destinados para el análisis de datos.

Al tener distintas fuentes de datos una forma inteligente de prepararlos es normalizar el entorno en el que se modificarán. Esto ayuda a crear procesos estandarizados y automatizados para su manipulación. En general los SMBD incorporan herramientas que permiten la importación de datos desde distintas fuentes.

2.4. Preparación de los datos.

Aun cuando los datos sean incorporados dentro de un mismo SMBD quedan muchas cuestiones a resolver ya que los datos pueden ser heterogéneos. Dos problemas recurrentes son la integración y la transformación de los datos. La primer tarea involucra estandarizar los

datos de forma que los datos que tengan el mismo significado puedan ser consultados en conjunto, al tener distintas fuentes de datos esta tarea puede ser muy compleja pues cada fuente puede tener esquemas o paradigmas de almacenamiento distintos. Una vez superada la integración es necesario preparar los datos que servirán como entrada a los algoritmos de minado o análisis. Esta preparación puede llevarse a cabo conjuntamente con la transformación pero muchas veces los datos son analizados por distintos algoritmos por lo que cada proceso de minado requiere la manipulación de los datos para adecuarlos a la entrada requerida. Esta tarea puede ser tan simple como la proyección y selección hasta ser tan compleja como la agregación o transposición de los datos [9].

2.4.1. Integración.

En cuanto a la integración de bases de datos podemos encontrar algunos de los siguientes problemas técnicos [9][23]:

- Identificadores distintos.
- Tipos de datos incompatibles.
- Formato de datos distinto.
- Las relaciones entre los datos pueden ser diferentes.
- Las tablas y los campos son referidos con nombres diferentes.
- Los datos pueden almacenarse en tablas diferentes.
- La disposición de los datos no coincide (horizontal o vertical).

Un SMBD permite resolver muchos de los problemas mencionados con relativa facilidad mediante SQL o funciones de conversión de datos y manejo de cadenas. Algunos problemas como la transposición de tablas no son tan fáciles de resolver y a veces presentan limitaciones técnicas. [1]

2.4.2. Transformación.

Esta tarea depende del tipo de análisis que se requiere realizar. Incluso una vez integrados los datos es necesario modificarlos a fin de que puedan ser procesados por los algoritmos de minado. Algunos problemas comunes [9] pueden ser:

- Datos resumidos o agregados.
- Selección de los datos.
- Re-codificación de los campos.

[1] En muchos SMBD el número de columnas máximo que se pueden definir en una tabla es limitado, por lo que la transposición de una tabla completa no sería factible.

- Transposición de los datos.
- Reunión de tablas.

De igual forma un SMBD puede resolver estos problemas de forma sencilla mediante SQL. Un problema importante e interesante es la generación de datos resumidos o agregados. Esta transformación permite tener una visión general de la información e incluso puede utilizarse para la generación de reportes o análisis sencillos. En los sistemas OLAP está es una de las herramientas más poderosas pues a través de la generación de cubos de datos se puede hacer un análisis eficiente de grandes volúmenes de información de acuerdo a las distintas dimensiones que definen el cubo. Una de las operaciones OLAP importantes es PIVOT, la cual provee una presentación alternativa de los datos y permite mejorar la exploración para la minería de datos.

Capítulo 3
Agregaciones horizontales.

Basándose en la necesidad de proveer una forma alternativa y estándar para la preparación de los datos antes de su análisis, el autor en [18] define un tipo nuevo de funciones de agregación. La aplicación de este tipo de funciones de agregación devuelven un conjunto de datos que tienen dos características principales, por un lado obtiene una disposición transpuesta de los datos y al mismo tiempo resume los datos mediante una agregación. Estas dos operaciones permiten tener un tratamiento de los datos que no pertenece a un estándar pero que resulta ser de gran utilidad. La vista resumida o agregada de los datos es fundamental en un lenguaje de consulta de datos como SQL, en cambio, la transposición de tablas es utilizada en sistemas más sofisticados u orientados al análisis de datos pues en ellos es necesario poder manipular los datos de distintas formas para su análisis, por ejemplo sistemas OLAP. Dado que ésta visualización transpuesta de los datos es importante, algunos SMBD han implementado sus propias funciones o sentencias para lograr este cometido. Estos avances son funcionales pero son difíciles de utilizar pues tienen limitaciones prácticas además de que requieren tener un conocimiento previo de los datos y son poco intuitivos. Tomando esto en cuenta, las agregaciones horizontales presentan características que mejoran la transparencia y se basan en un estándar además de ser eficientes para obtener una vista transpuesta (horizontal) y resumida (agregada) de los datos. Este capítulo describe el funcionamiento y la utilidad de un nuevo tipo de funciones de agregación que por sus características son denominadas: funciones de agregación horizontales.

3.1. Conceptos y definiciones.

Basándonos en la nomenclatura de [18, 21] Sea F una tabla con una llave primaria K, m atributos o columnas categóricas y un atributo numérico A. Cada atributo categórico describe una característica y cada valor numérico es una medida. En términos OLAP, los atributos categóricos son dimensiones y son utilizadas para agrupar filas y totalizar los atributos medida, por lo que la tabla F puede manipularse como un cubo con m dimensiones [9]. El tamaño de la tabla F es N, es decir $|F| = N$.

La cláusula GROUP BY junto con las funciones de agregación de SQL devuelven un resultado vertical de los datos, es decir, por cada grupo la función de agregación devuelve un único resultado. Sean $j + k$ columnas de agrupación y A el atributo al que se le aplica una función de agregación. El resultado es una tabla F_V con $j + k + 1$ atributos, donde los atributos j y k son clave candidata, es decir, identifican cada fila de la tabla de forma única, debido a la agrupación. La meta de las funciones de agregación horizontal es transformar F_V en una tabla F_H con una visualización horizontal con n filas y $j + d$ columnas, donde d representa el número de combinaciones únicas de los valores de las k columnas de agrupación.

Las funciones de agregación estándar serán referidas a lo largo del trabajo como funciones

de agregación verticales para diferenciarlas de las funciones de agregación horizontales. Para diferenciar ambos tipos de agregaciones es necesario introducir una extensión de la notación SQL para una función de agregación donde se busca expresar la transposición y agregación de columnas. Esta pequeña modificación será transformada en una consulta SQL estándar, la generación de este código SQL es uno de los objetivos de la herramienta de software que se propone en este documento.

Considérese la consulta en SQL estándar con la cláusula GROUP BY sobre las columnas $L_1, ..., L_m$ de la tabla F.

SELECT $L_1, ..., L_m$, $f(A)$ FROM F GROUP BY $L_1, ..., L_m$;

Esta consulta produce una tabla con $m+1$ columnas, con un grupo por cada combinación única de los valores de $L_1, ..., L_m$ y un valor totalizado $f(A)$ por cada grupo. Una agregación horizontal tiene como meta transponer la agregación de la columna A por las columnas $\{R_1, ..., R_k\} \subset \{L_1, ..., L_m\}$. Es decir, la lista de columnas de agrupación se dividirá en dos sub-listas, una que produce cada grupo (j columnas $L_1, ..., L_j$) y otra que transpone los valores agregados (k columnas $R_1, ..., R_k$). La intersección de estas sub-listas es vacía. Cada combinación de valores distintos de $\{R_1, ..., R_k\}$ producirá una columna. Por ejemplo si $k = 1$ entonces cada valor en R_1 se convertirá en una columna que almacena el resultado de una función de agregación.

Entonces para una función de agregación horizontal es necesario definir cinco parámetros:

1. La tabla de entrada F.
2. La lista de atributos o columnas de agrupación $L_1, ..., L_j$.
3. La función de agregación f.
4. La columna A a la que se le aplica la función de agregación.
5. La lista de atributos o columnas de transposición $R_1, ..., R_k$

Una vez definidos estos cinco parámetros se define la modificación a la sintaxis SQL que permite de manera intuitiva expresar una función de agregación horizontal. Esta extensión modifica la sintaxis de una función de agregación vertical introduciendo una cláusula o palabra reservada "BY" seguida de la lista de columnas de transposición. La sintaxis propuesta en [18, 21] es la siguiente:

SELECT $L_1, ..., L_j$,
$f(A\ BY\ R_1, ..., R_k)$ FROM F
GROUP BY $L_1, ..., L_j$;

Esta modificación necesita definir un comportamiento para su evaluación. La función de agregación $f()$ representa una función de agregación estándar de SQL (*sum(), count(), max(), min(),avg()*). El número de filas está definido por las columnas $L_1, ..., L_j$, este conjunto puede ser vacío, como en el estándar de SQL donde la cláusula GROUP BY es opcional. El número de columnas está definido por las distintas combinaciones de los valores de las columnas $R_1, ..., R_k$. Cuando la cláusula GROUP BY no está presente el

resultado será una sola fila. La cláusula BY es opcional, cuando no está presente, la agregación horizontal se reduce a una agregación vertical, por otro lado cuando está presente la lista $R_1, ..., R_k$ es obligatoria. Las agregaciones horizontales pueden ser combinadas con agregaciones verticales u otras agregaciones horizontales en la misma consulta, todas las agregaciones compartirán la lista de columnas de agrupación $L_1, ..., L_j$. Cuando una función $f()$ es utilizada más de una vez se deben tener diferentes listas de columnas de transposición para no repetir nombres de columnas.

Una vez definida la sintaxis y el comportamiento de las funciones de agregación horizontales en [18, 21] se proponen distintas formas de generar código SQL estándar que devuelvan el resultado definido por este tipo de funciones.

3.2. Optimizaciones.

De acuerdo a lo mostrado en [21] se conocen tres tipos de consultas equivalentes u optimizaciones que generan el resultado esperado de una agregación horizontal. Al ser equivalentes, cualquiera de estas consultas es útil como base en el desarrollo de la herramienta propuesta pero el factor del desempeño resulta tener un gran impacto y marca una diferencia que permite establecer cuál de estas optimizaciones es la mejor.

Las optimizaciones son referidas como SPJ (SELECT PROJECT JOIN), CASE y PIVOT. Para una consulta genérica con agregación horizontal como la siguiente se mostrarán las tres optimizaciones.

SELECT $L_1, ..., L_j$,
$f(A\ BY\ R_1, ..., R_k)$ FROM F
GROUP BY $L_1, ..., L_j$;

3.2.1. Optimización SPJ.

−−Obtener las d combinaciones de valores de $R_1, ..., R_k$ *:*
SELECT DISTINCT $R_1, ..., R_k$ FROM F;
−−Generar una tabla base que contiene los grupos definidos por las columnas L_i
INSERT INTO F_0 (SELECT DISTINCT $L_1, ..., L_j$ FROM F);
−−Para cada combinación de valores $R_1, ..., R_k$ *se obtiene una tabla* F_I
−−y se agregan los datos para esa combinación:
INSERT INTO F_I (
SELECT $L_1, ..., L_j$,
$f(A)$ AS A FROM F
WHERE $R_1 = v_{1I}$ AND ... AND $R_k = v_{kI}$ GROUP BY $L_1, ..., L_j$);

−−Las tablas son reunidas por la izquierda con la tabla base $F0$
−−para obtener todas las columnas:
SELECT $F_0.L_1, ..., F_0.L_j$,
$F1.A, F2.A, ..., Fd.A$
FROM F_0 LEFT OUTER JOIN F_1

ON $F_0.L_1 = F_1.L_1$ AND ... AND $F_0.L_j = F_1.L_j$
...
LEFT OUTER JOIN F_d
ON $F_0.L_1 = F_d.L_1$ AND ... AND $F_0.L_j = F_d.L_j$

Esta optimización tiene la desventaja de leer $d+1$ veces la tabla F y además reunir todos los resultados parciales, este método es el menos eficiente aunque utiliza solamente código SQL estándar y se tiene una equivalencia en álgebra relacional lo cual proporciona una base teórica fuerte para las funciones de agregación horizontales.

3.2.2. Optimización PIVOT.

Para esta optimización se utiliza la cláusula PIVOT [3], la cual ya se ha implementado en algunos SMBD comerciales. Esta cláusula solamente permite transponer los datos de acuerdo a una sola columna, es decir, $k = 1$. $--$*Obtener las d combinaciones de valores de* $R_1, ..., R_k$ *:*
SELECT DISTINCT $R_1, ..., R_k$ FROM F);
SELECT $L_1, ..., L_j, v_1, ..., v_d$ INTO F_t
INTO F_t FROM
(SELECT $L_1, ..., L_j, R_1, A$ FROM F) AS F_t
PIVOT ($f(A)$ FOR R_1 IN $(v_1, ..., v_d)$) AS P;

SELECT $L_1, ..., L_j, f(v_1), ..., f(v_d)$
FROM F_t
GROUP BY $L_1, ..., L_j$;

El primer paso es obtener las d combinaciones de valores distintos y a continuación se procede a transponer los datos y hacer una agregación parcial, la segunda parte ejecuta la agrupación y calcula las agregaciones finales. Esta optimización lee la tabla dos veces la tabla F pero está limitada al transponer solamente una columna.

3.2.3. Optimización CASE.

$--$*Obtener las d combinaciones de valores de* $R_1, ..., R_k$ *:*
SELECT DISTINCT $R_1, ..., R_k$ FROM F;
SELECT $L_1, ..., L_j$,
f(CASE WHEN $R_1 = v_{11}$ and ... and $R_k = v_{1d}$ THEN A ELSE *null* END) ,
...
f(CASE WHEN $R_1 = v_{1d}$ and ... and $R_k = v_{kd}$ THEN A ELSE *null* END) ,
FROM F_V GROUP BY $L1, ..., Lj$;

Esta optimización es más simple que las anteriores y lee la tabla F dos veces para obtener el resultado final. La cláusula CASE con múltiples opciones (WHEN) permite que $k >= 1$ a diferencia de la optimización PIVOT. Esta sintaxis de la cláusula CASE pertenece al estándar por lo que puede ser utilizada en distintos sistemas a diferencia de la cláusula PIVOT que al no pertenecer al estándar puede no funcionar correctamente en otro SMBD.

3.3. Evaluación de las optimizaciones.

En todas las optimizaciones la primer consulta obtiene las combinaciones únicas de las columnas $R_1, ..., R_k$, por lo que también contamos esto como una lectura de la tabla F. Esta lectura extra puede ser eliminada al construir una tabla F_V y ejecutar todas las consultas sustituyendo F por F_V:

SELECT $L_1, ..., Lj, R_1, ..., R_k, f(A)$
INTO F_V FROM F
GROUP BY $L_1, ..., Lj, R_1, ..., R_k$;

Esta optimización solamente elimina la lectura extra y es la que se utilizó para evaluar y comparar las 3 optimizaciones. Esta comparación se realizó sobre la base de datos TPCH [27], misma que se utilizará en el presente trabajo. De los resultados experimentales en [21] se obtuvieron los siguientes resultados:

- El método SPJ fue el más lento.
- La optimizaciones PIVOT y CASE fueron las más eficientes y se obtuvieron comportamientos muy similares.
- En general se realizaron pruebas donde $d << n$, un escenario común en minería de datos.
- El factor n[1] es el que influye en el desempeño de las tres optimizaciones, para SPJ el factor d también tiene un peso importante.
- En la experimentación, los planes de ejecución mostraban paralelismo para la optimización CASE.

De acuerdo a esta evaluación y a las características de las tres optimizaciones, el método CASE es uno de los más eficientes junto con PIVOT ya que permite definir transposición por más de una columna y las cláusulas SQL utilizadas pertenecen a un estándar. Es por ello que el método CASE será utilizado como base de la herramienta propuesta. Esta optimización será implementada y se buscará mejorar su tiempo de ejecución a través de otros mecanismos.

[1] Recordemos que n es el numero de filas del resultado, esta definido por el número de combinaciones de valores distintos de $L_1, ..., L_j$

Capítulo 4
Optimizaciones propuestas

En este capítulo se exponen las optimizaciones implementadas en la herramienta de software. Para cada optimización propuesta se expondrán las condiciones bajo las cuales puede ser aplicada a consultas con agregaciones horizontales y se mostrarán los resultados obtenidos por la consulta optimizada y la no optimizada.

4.1. Optimización de llave foránea.

En una base de datos el uso de llaves foráneas son necesarias para mantener la integridad de los datos, en particular para definir la integridad referencial que ayuda a mantener la consistencia de las relaciones definidas por la lógica del negocio.

Las agregaciones horizontales crean nuevas columnas a partir de los valores distintos de un conjunto de columnas existentes en una tabla. En SQL la cláusula `DISTINCT` nos ayuda a obtener estos valores, para ello es necesario leer todos los registros en la columnas de las cuales se requieren los valores distintos. Esta operación depende del tamaño de la tabla, lo que puede ser costoso y por ello resulta importante preguntarse si existe una forma más eficiente de obtener estos valores.

La optimización propuesta trata de mejorar la recuperación de las de combinaciones de d valores distintos cuando las columnas que los contienen son una llave foránea, pues en este caso podemos obtenerlos mediante una proyección de la llave primaria a la cual están relacionados.

```
SELECT L,f(A BY R) FROM T GROUP BY L;
```

Figura 4.1: Consulta genérica para ejemplificar la optimización de llave foránea.

Sea la consulta de la Figura 4.1 con una agregación horizontal en la cual se cumplen las siguientes condiciones

- R es llave foránea de la tabla T
- R está relacionada a la llave primaria P de una tabla S.

4.1.1. Consulta CASE

De acuerdo a [18] la consulta en SQL equivalente es la consulta de la Figura 4.2

```
--Se obtienen los d valores distintos v1,v2,...,vd
 SELECT DISTINCT R FROM T;
--Se calcula la agregación horizontal
 SELECT L,
   f(CASE WHEN R=v1 THEN A ELSE null END) as f_R_v1,
   ...
   f(CASE WHEN R=vd THEN A ELSE null END) as f_R_vd,
FROM T GROUP BY L;
```

Figura 4.2: Optimización CASE para la consulta de la Figura 4.1

De esta consulta podemos observar que la tabla T se leerá dos veces, una para obtener los valores distintos de la columna R y otra para obtener el resultado final el cual tendrá $d+1$ columnas.

4.1.2. Optimización propuesta

Nuestra propuesta toma en cuenta la existencia de la relación entre las tablas T y S a través de la llave foránea definida en la columna R de la tabla T. La optimización consiste en obtener los valores distintos a través de la proyección de la llave primaria P de la tabla S y así evitar una lectura de la tabla T. La consulta optimizada se muestra en la Figura 4.3.

```
--Se obtienen e valores distintos v1,v2,...,vd,...,ve
--de la tabla S
 SELECT P FROM S;
--Se calcula la agregación horizontal
 SELECT L,
   f(CASE WHEN R=v1 THEN A ELSE null END) as f_R_v1,
   ...
   f(CASE WHEN R=vd THEN A ELSE null END) as f_R_vd,
   ...
   f(CASE WHEN R=ve THEN A ELSE null END) as f_R_ve,
FROM T GROUP BY L;
```

Figura 4.3: Optimización de llave foránea para la consulta de la Figura 4.1.

La ejecución involucra una lectura de la tabla S para obtener los e valores distintos pertenecientes a la llave primaria P y también una lectura de la tabla T para obtener el resultado final.

En la consulta de la Figura 4.3 podemos observar que los valores distintos son e y no d, esto es por que en la columna R de la tabla T no necesariamente se tienen todos los valores de la llave primaria P. Por lo tanto el resultado final tendrá $e+1$ columnas.

Supongamos que el valor ve de la llave primaria P no es referido en la tabla T por la columna R, entonces al ejecutar la consulta con las clausulas `CASE` la expresión `f(CASE WHEN R=ve`

`THEN A ELSE null END`) regresará siempre un valor nulo pues ningún registro cumplirá tal condición. En general, con esta optimización siempre obtendremos $e - d$ columnas donde todos los registros tendrán un valor nulo, por lo cual el usuario podrá omitir estas columnas como parte del resultado final.

La optimización propuesta evita una segunda lectura de la tabla T por lo que uno de los parámetros importantes es el número de registros n de esta tabla.

En la práctica, la definición de una llave primaria crea de forma inherente un índice por lo que la proyección de la llave primaria P no tiene que leer la tabla S completamente, sólo el archivo del índice, pues ahí se encuentran los valores de la llave primaria.

Por otro lado, una de las limitaciones prácticas de las agregaciones horizontales es el número de columnas que pueden generar. En los SMBD actuales, existe un límite λ para el número de columnas en una tabla, generalmente 1024. Este límite favorece a la optimización pues querría decir que el máximo número de valores distintos d también tendría un límite igual a $\lambda - |L|$ donde L son las columnas de agrupación de la consulta. Al ser P llave primaria de la tabla S, la máxima cantidad de valores en P también estaría acotada por lo que el número de registros en S será muy pequeño en comparación con el número de registros que puede almacenar la tabla T. Esto beneficia la optimización, que en el peor caso tendría que hacer una proyección de $\lambda - |L|$ valores, por el contrario, la consulta sin optimizar tendría que buscar $\lambda - |L|$ valores distintos en n registros, donde n no está acotado y en la práctica podría ser del orden de millones.

4.2. Optimización asíncrona.

En un SMBD la lectura a disco es una de las operaciones más lentas. Una consulta que necesita consultar una gran cantidad de datos puede tomar demasiado tiempo por el simple hecho de leerlos desde el disco. Una vez entregados los datos al usuario el sistema da por terminado el procesamiento de la consulta. Si la misma consulta es procesada, el sistema puede almacenar en memoria el resultado y devolverlo en un tiempo mucho menor que la primera vez[1]. Si la consulta no es la misma pero se requieren los mismos datos que la consulta anterior, el sistema debe procesar esta nueva petición leyendo[2] y procesando los nuevos datos. Si el sistema almacena en su buffer los datos únicamente, entonces la evaluación de la consulta sería lo único que habría que procesar nuevamente.

Con este enfoque se expone una optimización que pretende reducir el número de lecturas así como el tiempo de procesamiento para consultas con agregaciones horizontales. De acuerdo al trabajo relacionado, la forma de obtener el resultado de una agregación horizontal se hace mediante SQL estándar. Al igual que el SMBD, la herramienta de software que se implementó utiliza un esquema de almacenamiento de resultados intermedios para consultas eventuales. Esta optimización aprovecha aspectos como la agrupación en SQL y la propiedad distributiva de algunas funciones de agregación.

[1] Esto puede deberse al cache del buffer de datos del SMBD, el cual puede almacenar el resultado de las consultas o los datos leídos del disco.

[2] Esto ocurre si el SMBD sólo almacena los resultados de las consultas, pues al ser una consulta diferente el resultado de la consulta anterior puede no ser útil. También puede haber lecturas a disco si es que la cantidad de datos no puede ser almacenada en el buffer del sistema.

Sea una consulta con agregación horizontal como la que se muestra en la Figura 4.4. En esta consulta tenemos m columnas de agrupación "vertical" y k columnas de agrupación "horizontal". El resultado de esta consulta tendrá $m + d$ columnas.

```
SELECT L1,...,Lm,f(A BY R1,...,Rk)
  FROM T GROUP BY L1,...,Lm;
```

Figura 4.4: Consulta para ejemplificar la optimización asíncrona.

Si el usuario ahora ejecuta la consulta mostrada en la Figura 4.5, donde $h < m$ ó $i < k$, el sistema no podrá devolver el resultado alamacenado en el buffer de la consulta de la Figura 4.4, ya que el número de columnas de la consulta mostrada en 4.5 es $h+d' < m+d$, incluso cuando $h = m$ y $i = k$, si g es diferente a f el resultado de la de ejecutar el código de la Figura 4.4 no sería correcto. Si el sistema almacena en buffer los datos de la tabla T, tendrá que calcular nuevamente la agrupación y la función g, en el caso que la cantidad de datos de T no puedan ser almacenados en el buffer, el sistema leerá los datos restantes del disco.

```
SELECT L1,...,Lh,f(A BY R1,...,Ri)
  FROM T GROUP BY L1,...,Lh;
```

Figura 4.5: Consulta para ejemplificar la optimización asíncrona

4.2.1. Consulta CASE

De acuerdo a [18] podemos obtener el resultado de cada una de las consultas de la siguiente forma:

1. Generar una consulta que agrupe por los atributos L y R aplicando la función de agregación. Materializar el resultado en una tabla FV.

2. Obtener los d valores distintos de las columnas R a partir de la tabla FV.

3. Aplicar la consulta con las clausulas `CASE` para obtener el resultado final

De acuerdo a los pasos anteriores las consultas de las Figuras 4.4 y 4.5, éstas se traducen al código moestrado en las Figuras 4.6 y 4.7 respectivamente.

```
--Se materializa la tabla FV
 SELECT L1,...,Lm,R1,...,Rk,f(A) as f_A
 INTO FV
 FROM T GROUP BY L1,...,Lm,R1,...,Rk;
--Se obtienen los d valores distintos (v11,...,vk1), (v1d,...,vkd)
 SELECT DISTINCT R1,...,Rk FROM FV;
--Se construye la consulta CASE y se evalúa
 SELECT L1,...,Lm,
   f(CASE WHEN R1=v11 and ... and Rk=v1d THEN f_A ELSE null END) ,
   ...
   f(CASE WHEN R=v1d and ... and Rk=vkd THEN f_A ELSE null END) ,
FROM FV GROUP BY L1,...,Lm;
```

Figura 4.6: Optimización CASE para evaluar la consulta de la Figura 4.4.

```
 --Se materializa la tabla FV
 SELECT L1,...,Lh,R1,...,Ri,g(A) as g_A
 INTO FV
 FROM T GROUP BY L1,...,Lh,R1,...,Ri;
--Se obtienen los d valores distintos (v11,...,vi1), (v1d,...,vid)
 SELECT DISTINCT R1,...,Ri FROM FV;
--Se construye la consulta CASE y se evalúa
 SELECT L1,...,Lh,
   f(CASE WHEN R1=v11 and ... and Ri=v1d THEN g_A ELSE null END) ,
   ...
   f(CASE WHEN R=v1d and ... and Ri=vid THEN g_A ELSE null END) ,
FROM FV GROUP BY L1,...,Lh;
```

Figura 4.7: Optimización CASE para evaluar la Consulta de la Figura 4.5.

Como podemos observar ambas consultas generan su tabla FV por lo que podemos borrar esta tabla antes de evaluar cualquier consulta o alternativamente nombrarlas de forma distinta. La ejecución de estas consultas en tiempos distintos involucran en teoría dos lecturas de la tabla T, una para cada consulta con agregación horizontal.

4.2.2. Optimización propuesta

En esta optimización se aprovecharán los resultados parciales de las consultas utilizadas para generar el resultado final.

Para ejecutar la consulta de la Figura 4.4 con agregación horizontal procederemos de la siguiente forma:

1. Verificar si existe la tabla FV, si existe continuamos en el paso 5, en otro caso continuamos con el paso 2.

2. Generar una consulta que agrupe por los atributos L y R de la tabla T aplicando la función de agregación y materializar el resultado en la tabla FV.
3. Obtener los valores distintos de las columnas R a partir de la tabla FV.
4. Aplicar la consulta con las clausulas `CASE` a FV para obtener el resultado final
5. Si $\{L_1, ...L_m, R_1, ..., R_k\} \subseteq C$ donde C es el conjunto de columnas de la tabla FV y f es una función distributiva de tal forma que: $f(A) = f(f'(A))$ para $f'(A)$ renombrada como F'_A la función de agregación calculada en FV continuamos al paso 6, en otro caso regresamos al paso 2.
6. Obtener los valores distintos de las columnas $R_1, ..., R_k$ a partir de la tabla FV.
7. Aplicar la consulta con las clausulas `CASE` a FV para obtener el resultado final

Si ejecutamos las consultas de las Figuras 4.4 y 4.5 con este algoritmo obtenemos la secuencia de consultas en SQL estándar mostradas en la Figura 4.8.

```
--1) En un inicio no existe la tabla FV
--2) Se materializa la tabla FV a partir de la tabla T
 SELECT L1,...,Lm,R1,...,Rk,f(A) as f_A
 INTO FV
 FROM T GROUP BY L1,...,Lm,R1,...,Rk;
--3) Se obtienen los d valores distintos (v11,...,vk1), (v1d,...,vkd)
 SELECT DISTINCT R1,...,Rk FROM FV;
--4) Se construye la consulta CASE y se evalúa
 SELECT L1,...,Lm,
   f(CASE WHEN R1=v11 and ... and Rk=v1d THEN f_A ELSE null END) ,
   ...
   f(CASE WHEN R=v1d and ... and Rk=vkd THEN f_A ELSE null END) ,
FROM FV GROUP BY L1,...,Lm;

--Para la ejecución de la Consulta 2
--1) Existe la tabla FV creada por la consulta 1
--5) Se cumplen las restricciones
--6) Se obtienen los d valores distintos (v11,...,vi1), (v1d,...,vid)
 SELECT DISTINCT R1,...,Ri FROM FV;
--Se construye la consulta CASE y se evalúa
 SELECT L1,...,Lh,
   g(CASE WHEN R1=v11 and ... and Ri=v1d THEN f_A ELSE null END) ,
   ...
   g(CASE WHEN R=v1d and ... and Ri=vid THEN f_A ELSE null END) ,
FROM FV GROUP BY L1,...,Lh;
```

Figura 4.8: Optimización asincrona para las consultas de las figuras 4.6 y 4.7

Las restricciones del paso 5) aseguran que por una parte todas las columnas de agrupación para la consulta 2 se encuentren en la tabla FV, de otra manera la consulta de la Figura 4.5

no puede aprovechar los resultados obtenidos por la consulta de la figura 4.4. La restricción sobre la función de agregación es más interesante pues al pedir que sea distributiva, esta consulta puede aprovechar el procesamiento que ya se ha realizado.

La optimización en la consulta en la Figura 4.7 es asincrona pues su tabla FV fue calculada en un tiempo anterior, la ventaja que busca esta optimización es principalmente reducir el número de lecturas de la tabla T y adicionalmente disminuir el tiempo de procesamiento mediante el comuto de la función distributiva. La optimización no necesariamente funciona para dos consultas, por ejemplo, una tercera consulta podría seguir aprovechando los resultados de la consulta de la Figura 4.4 almacenados en la tabla FV si es que cumple las restricciones del paso 5).

Una desventaja de esta optimización es el espacio necesario para almacenar las tablas intermedias FV. Cuando la cantidad de datos es muy grande puede ser preferible sacrificar el almacenamiento a fin de disminuir el tiempo de respuesta.

En general esta optimización se ve afectada por el número de registros en la tabla T y la cantidad de datos almacenados en las tablas FV, ya que si la tabla FV tiene un tamaño muy cercano a $|T|$ entonces las consultas sobre la tabla FV pueden tardar tanto como consultar a la tabla T. En general este caso es poco útil pues al agrupar datos lo que se busca es resumir los datos originales para tener una perspectiva de los datos en conjunto.

4.3. Optimización de consulta anidada

En distintas consultas la tabla de datos es una tabla derivada, es decir, una sub-consulta. En este tipo de tablas los nombres de las columnas no puede repetirse y si se trata de expresiones, incluidas las funciones de agregación, debe haber un renombre (clausula `AS`). La sub-consulta también debe tener un renombre para evitar ambigüedad. En el caso de las consultas con funciones de agregación horizontal como tablas derivadas se presentan algunos problemas los cuales se explicarán a continuación.

Sea la consulta de la Figura 4.9 con dos niveles de anidación. Nos referiremos a la consulta no anidada como consulta externa, la cual tiene un nivel de anidación cero. La tabla de la consulta externa es una tabla derivada $S1$, cuya consulta se encuentra en el primer nivel de anidación;a su vez $S1$ tiene una tabla derivada $S2$ que está en el segundo nivel de anidación respecto a la consulta externa y por último se tiene la tabla T sobre la cual se evaluará la consulta $S2$. Los valores distintos son d_1 y d_2 para los conjuntos de atributos $R_1, ..., R_k$ y $R_1, ..., R_l$ respectivamente. Además $k \leq l$ por lo que $d_1 \leq d_2$. Por último $m \leq p$.

```
SELECT L1,...,Lm,R1,...,Rk,f2_f1_A FROM (
  (SELECT L1,...Lm,R1,...,Rk, f1(f2_A) as f1_f2_A
    (SELECT L1,...Lp,R1,...,Rl, f2(A) as f2_A
     FROM T GROUP BY L1,...,Lp,R1,...,Rl) as S2
  FROM T GROUP BY L1,...,Lm,R1,...,Rk) as S1
)
```

Figura 4.9: Consulta con dos niveles de anidación.

El resultado de la consulta $S2$ tendrá $p + l + 1$ columnas, $S1$ devolverá $m + k + 1$ al igual

que la consulta externa. A su vez $m + k + 1 \leq p + l + 1$ y en cada consulta las referencias de atributos de una tabla derivada no presentan ningún problema pues cada que se hace referencia a una expresión de una sub-consulta se sabe que el resultado de dicha expresión se guarda en una sola columna.

A continuación estudiamos el caso en el que las consultas contengan agregaciones horizontales como se puede observar en la Figura 4.10. Primero definiremos el comportamiento de una consulta que contiene una expresión con una referencia a una agregación horizontal de una tabla derivada. Para este tipo de consultas, toda expresión con referencia a una agregación horizontal generará d expresiones con referencias a cada una de las columnas resultantes de la evaluación de la agregación horizontal, donde d es el número de valores distintos del conjunto de atributos R para la agregación horizontal. Por ejemplo en la Figura 4.10 la consulta $S1$ tiene la función de agregación horizontal `f1(f2_A BY R1,...,Rk)`, el atributo medida `f2_A` es una referencia a una función de agregación horizontal de la sub-consulta $S2$, para el conjunto $R_1, ..., R_l$ las combinaciones de valores distintos son d_2, por lo que en $S1$ en lugar de tener una agregación horizontal se tendrán ahora d_2 agregaciones, esto se puede ver en la consulta equivalente de la misma Figura.

```
--Consulta con tabla derivada y agregación horizontal
SELECT L1,...,Lm,R1,...,Rk,f1_f2_A FROM (
  (SELECT L1,...Lm, f1(f2_A BY R1,...,Rk) as f1_f2_A
   FROM
    (SELECT L1,...Lp, f2(A BY R1,...,Rl) as f2_A
     FROM T GROUP BY L1,...,Lp) as S2
  FROM T GROUP BY L1,...,Lm) as S1
)
--Consulta equivalente
SELECT L1,...,Lm,
  f2_f1_A_1_1,...,
  f2_f1_A_1_d1,...,
  f2_f1_A_d2_1,...,
  f2_f1_A_d2_d1
  FROM (
  (SELECT L1,...Lm,
    f1(f2_A_1 BY R1,...,Rk) as f1_f2_A_1 ,
    ...,
    f1(f2_A_d2 BY R1,...,Rk) as f1_f2_A_d2
   FROM
    (SELECT
      L1,...,Lp, f2(A BY R1,...,Rl) as f2_A
     FROM T GROUP BY L1,...,Lp) as S2
  FROM T GROUP BY L1,...,Lm) as S1
)
```

Figura 4.10: Consulta anidada y su expresión equivalente.

Al evaluar cada sub-consulta tendremos que para $S2$ el número de columnas es $p + d_2$ pues

la agregación horizontal genera d_2 columnas. Para $S1$ será $m + d_2 * d_1$ pues se generaron d_2 agregaciones horizontales, cada una con el conjunto de atributos $R_1, ..., R_k$ donde el número de combinaciones distintas es d_1, por último la consulta externa sólo hace la proyección de $S1$ y aquí podemos observar como la referencia inicial `f1_f2_A` se expandió a $d_1 * d_2$ columnas. Esto se muestra en la consulta equivalente de la Figura 4.10.

Esta consulta equivalente pareciera ser errónea pues en $S1$ las d_2 referencias no aparecen en la consulta $S2$, aunque existirán al evaluarse. Primero se evalúa $S2$, la consulta más anidada, ésta devolverá su resultado a $S1$ la cual servirá como la tabla de datos de la consulta externa. Como se puede ver en este ejemplo, es necesario saber de antemano los valores de los atributos R para cada agregación horizontal, pues esto permitirá generar la consulta equivalente que obtiene el resultado final.

El problema puede ser que si solamente tomamos en cuenta la consulta anidada en el nivel de anidación siguiente podríamos perder referencias pues nada nos asegura que las consultas anidadas no tendrán más columnas, como resultado de una agregación horizontal.

Para resolver este problema es necesario obtener los valores de los conjuntos de atributos R de todas las consultas con agregaciones horizontales y comenzar a sustituir desde la consulta más anidada hasta la consulta externa. El siguiente algoritmo se encarga de transformar una consulta con anidación y funciones de agregación horizontal en una consulta equivalente donde todas las consultas tengan las referencias adecuadas para su evaluación.

Algoritmo 1 Algoritmo para la ejecución de consultas anidadas con agregaciones horizontales

1: Si la consulta Q no tiene una tabla derivada con agregación horizontal, evaluar normalmente y terminar.
2: Sea S la tabla derivada de Q con la función de agregación horizontal $f(A\ BY\ R)\ as\ f_A$ sobre la tabla T .
3: Si T no es una tabla derivada, almacenar en M los pares $< f_A, rename(v_i) >$ para $i = 1, ..., d.$ donde v es uno de los valores distintos de R.
4: Si T es una tabla derivada almacenar en M los pares resultantes al evaluar T (paso recursivo)
5: Generar en Q las expresiones tales que correspondan con un valor x de $< x, y >$ en M. Si la expresión corresponde a una agregación horizontal $g(B\ BY\ R')\ as\ g_B$ agregar a M los pares $< g_B, g_B + y + rename(u_i) >$ para $i = 1, ..., d'$, donde u corresponde a los d' valores distintos de R'.
6: Evaluar y materializar S.
7: Cambiar la tabla derivada de Q por la tabla materializada de S
8: Evaluar Q normalmente y terminar.

De acuerdo a este algoritmo, para cada función de agregación horizontal deben obtenerse los valores distintos de las columnas que se van a transponer. En la práctica esto puede ser muy costoso pues por cada agregación se obtienen los valores distintos lo cual requiere un escaneo completo de la tabla.

Para esta optimizar esta ejecución se tiene el siguiente algoritmo.

Algoritmo 2 Algoritmo optimizado para la ejecución de consultas anidadas con agregaciones horizontales

1: Si la consulta Q no tiene una tabla derivada con agregación horizontal, evaluar normalmente y terminar.
2: Para toda consulta anidada, se obtendrán sus columnas de agrupación L, y en caso de haber funciones de agregación horizontal, las columnas de transposición R y las funciones de agregación $f(A)$.
3: Se obtiene la tabla base T sobre la que se ejecutan las sub-consultas.
4: Construir con estas columnas y esta tabla la siguiente consulta y materializarla:

```
INSERT INTO FV (SELECT L,R, f1(A),...,fj(B) FROM T GROUP BY L,R);
```

5: Sustituir T por FV en toda la consulta Q.
6: Aplicar el algoritmo 1 sobre Q.

La optimización en este caso es crear una tabla que contenga una agrupación parcial de los datos respecto a la tabla T. Esta tabla también obtiene las agregaciones parciales, tales agregaciones deben ser distributivas o en caso de ser algebraicas pueden descomponerse en agregaciones distributivas las cuales se deberán operar para obtener el resultado final.

La optimización dependerá de la relación entre el tamaño de T y el tamaño de FV ya que si tienen un tamaño parecido entonces la optimización no dará muy buenos resultados. Por el contrario, si la tabla FV es muy pequeña en comparación con T la optimización se vera beneficiada pues el algoritmo 1 obtendrá eficientemente tanto las combinaciones distintas de los atributos R así como el resultado de las agregaciones pues éstas ya han sido pre calculadas en la tabla FV.

Capítulo 5
Implementación de la herramienta.

En este capítulo se expone el diseño de la herramienta la cual está basada en el prototipo presentado en [22], éste permite elegir únicamente una de las tres optimizaciones : CASE,SPJ o PIVOT [21]. La mejora realizada al prototipo fue la elección de la optimización CASE como método principal debido a su eficiencia comparado con los otros métodos y a que utiliza clausulas estándar de SQL. Adicionalmente la consulta es analizada y mediante los catálogos del SMBD se procede a aplicar las nuevas optimizaciones, descritas en el Capítulo 4.

5.1. Desarrollo del sistema

El desarrollo del sistema se realizó en un solo ciclo aplicado a 3 fases: Diseño, Implementación y Pruebas. El SMBD que se utilizó es SQL SERVER [17] y la herramienta se programó en JAVA. Por un lado SQL Server es uno de los manejadores comerciales más utilizados actualmente e integra la cláusula PIVOT con la que se pudo comparar la optimización CASE[21, 22].El lenguaje de programación elegido fue **Java** ya que este cuenta con una biblioteca de conectividad JDBC para este SMBD y en comparación con otros lenguajes o ambientes de programación que pudieran tener más compatibilidad con el SMBD (como **.Net**) se eligió **Java** ya que para programar las optimizaciones y generar el código SQL se necesita principalmente manejo de cadenas de caracteres y no se ocupa ninguna característica de compatibilidad pues el código generado es SQL estándar. Por el contrario, utilizar un lenguaje de programación o ambiente más compatible con el SMBD podría limitar el uso de la herramienta si se requiere adaptar para otro SMBD o incluso a otro Sistema Operativo como Linux. A continuación enumeramos la serie de objetivos que debe cumplir la aplicación para su implementación:

1. Recibir como entrada una consulta SQL con agregaciones horizontales.
2. Analizar la consulta y verificar que la expresión esté bien formada y tenga sentido semánticamente en SQL estándar así como para la nueva extensión que define una función de agregación horizontal.
3. Una vez analizada y validada la consulta se debe inferir qué optimizaciones son aplicables a la consulta, para ello se utilizará el catalogo del sistema del SMBD.
4. Después se procede a la generación del código SQL estándar basándose en la optimización CASE y en las optimizaciones aplicables del paso anterior.
5. El código SQL estándar servirá para obtener el resultado de la ejecución de la consulta y para obtener el plan de ejecución elegido por el SMBD.
6. La herramienta debe proporcionar al usuario:

a) El código SQL generado por la herramienta.

b) El plan de ejecución elegido por el SMBD.

c) Los datos resultantes de la ejecución de la consulta.

d) El tiempo de análisis, optimización y ejecución.

5.2. Diseño.

El diseño de este sistema se separó en tres módulos principales, cada uno encargado de cumplir con los objetivos planteados para el correcto funcionamiento de la herramienta. El primer módulo se encargara del análisis sintáctico y semántico de la consulta recibida como entrada. El módulo dos se encargara de generar la optimización en SQL estándar de la consulta recibida como entrada. Por último el módulo tres tomará el código SQL optimizado para obtener el plan de ejecución y el resultado de su evaluación. Además en cada módulo se medirá el tiempo de cada tarea para proporcionar al usuario los tiempos de cada parte del procesamiento de la consulta.

Este diseño permite simplificar la arquitectura de la herramienta, la cual puede verse en el diagrama de componentes mostrado en la Figura 5.1.

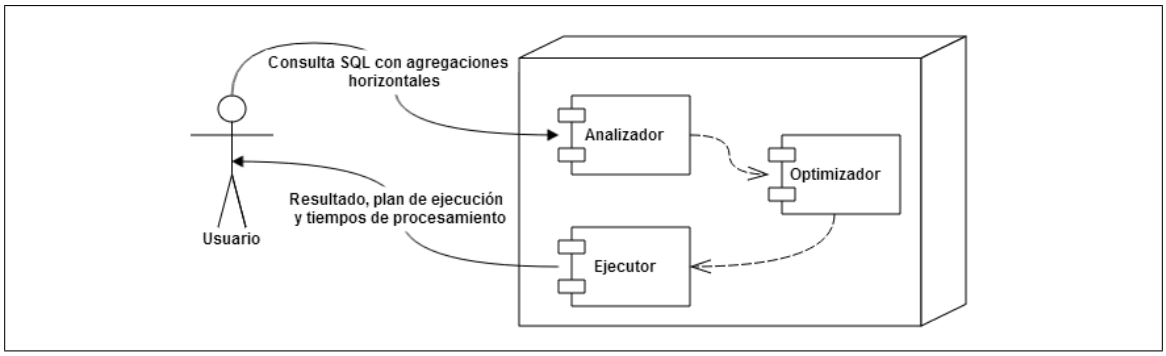


Figura 5.1: Diagrama de componentes de la herramienta.

Este diagrama muestra la dependencia existente entre los componentes, este diseño se basa en el procesamiento de consultas de un SMBD real. En las siguientes secciones mostraremos el diseño de cada uno de estos módulos y también se definirá cual será la interacción entre ellos.

5.2.1. Módulo de análisis.

Este módulo recibe una consulta con agregaciones horizontales y debe validar la sintaxis SQL junto con la extensión para agregaciones horizontales. Una vez validada se debe verificar la semántica de las expresiones, en particular para las agregaciones horizontales se verifica que la intersección de los atributos de transposición y los atributos de agrupación (si es que la cláusula GROUP BY esta presente) es vacía. Para verificar este tipo de restricciones se define un modelo orientado a objetos, Figura 5.2 que permite acceder a los distintos elementos de la consulta. Este modelado servirá también al módulo de optimización

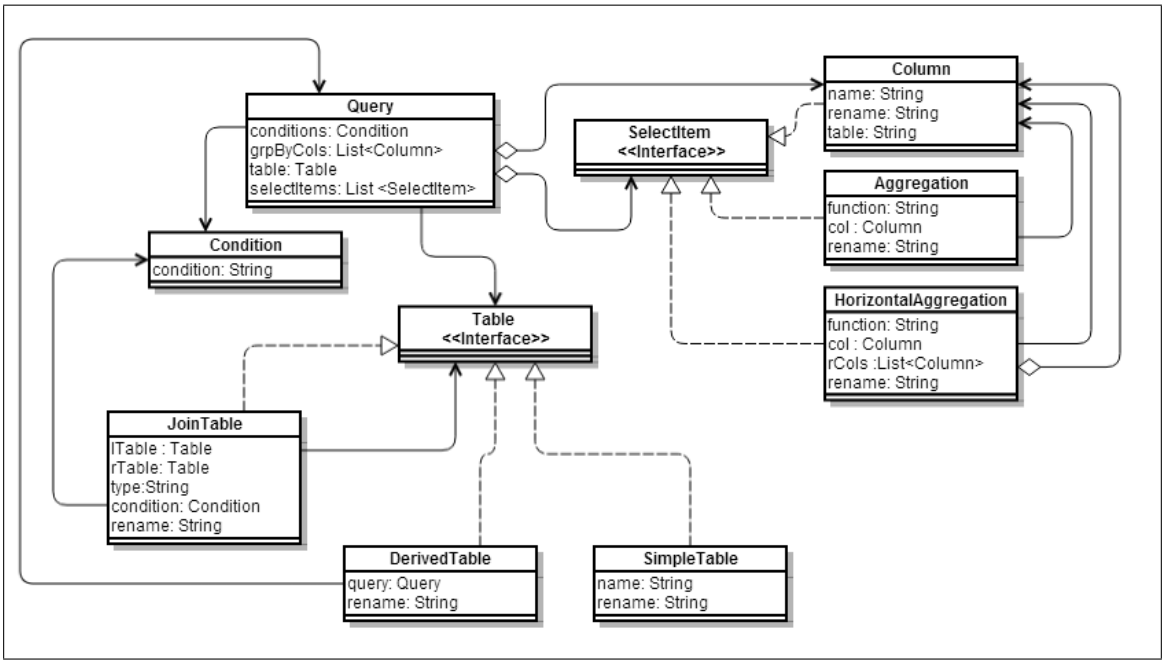


Figura 5.2: Diagrama de clases del modelo orientado a objetos de una consulta SELECT de SQL.

el cual manipulará esta estructura para crear una consulta SQL estándar optimizada para su evaluación.

Este módulo obtendrá como salida una instancia de la clase **Query** que representara la consulta introducida por el usuario. Dicha instancia será utilizada por el siguiente módulo para la generación de código SQL estándar.

5.2.2. Módulo de optimización.

Este módulo se encarga de transformar una instancia de la clase **Query** en código SQL estándar optimizado. Para ello primero debe de verificar cuales de las optimizaciones pueden aplicarse. Este módulo define 4 clases encargadas de aplicar cada una de las optimizaciones.

El diagrama de clases de la Figura 5.3 muestra las clases involucradas en la optimización de la consulta. La clase principal Optimizer recibe la instancia de la clase Query creada por el módulo de análisis, esta clase ejecuta su método **verifyOptimizations()** el cual deberá verificar mediante instancias de las clases **AsyncOptimizer**, **FkOptimizer** y **NestedOptimizer** si alguna optimización puede aplicarse y en dado caso que así sea deberá modificar la estructura de la instancia de la clase **Query**. La última verificación será la de la optimización asíncrona, la cual generará el código SQL final a través de la clase **CaseOptimizer**. En la Figura 5.4 se muestran las interacciones entre las instancias de los objetos involucrados en la optimización de una consulta con agregaciones horizontales.

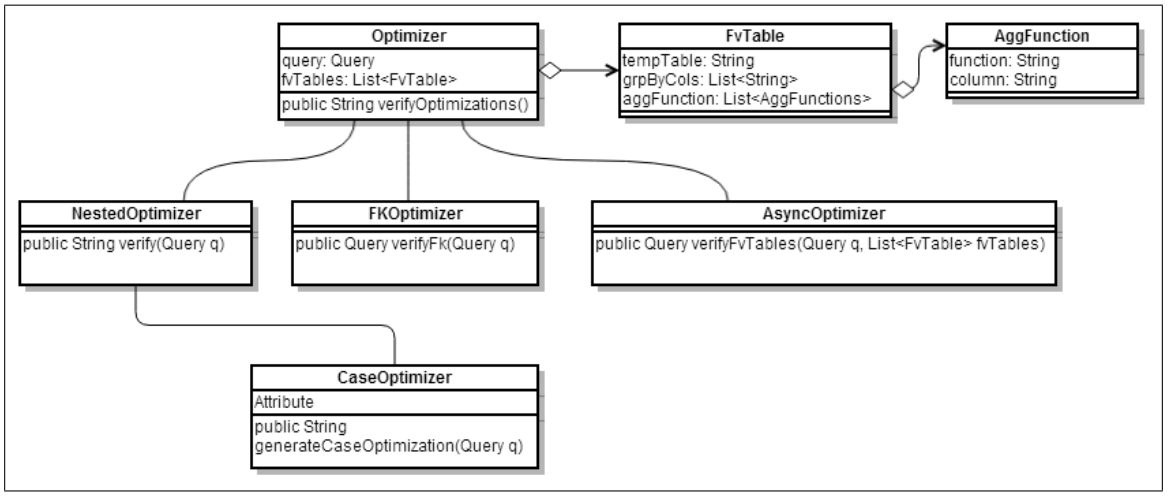


Figura 5.3: Diagrama de clases del módulo de optimización.

5.2.3. Módulo de ejecución

Por último el módulo de ejecución debe utilizar el código generado por el modulo anterior y ejecutarlo en el SMBD. Para ello se define una simple clase que implementa varios métodos que permiten interactuar con el SMBD. En particular debe definirse una interfaz de conexión a la base de datos. Esta interfaz se implementó para el SMBD **SQL Server 2008** pero pueden implementarse clases que permitan la interacción con otros SMBD.

La interfaz **Connection** define los métodos necesarios para poder ejecutar la consulta. Si se desea utilizar otro SMBD se puede implementar esta interfaz especificando las particularidades de dicho SMBD.

Por último los dos métodos que obtienen los resultados se encuentran en la clase **Executor**. Ambos métodos utilizan a su vez las funciones de la instancia de **Connection** definida en esta clase.

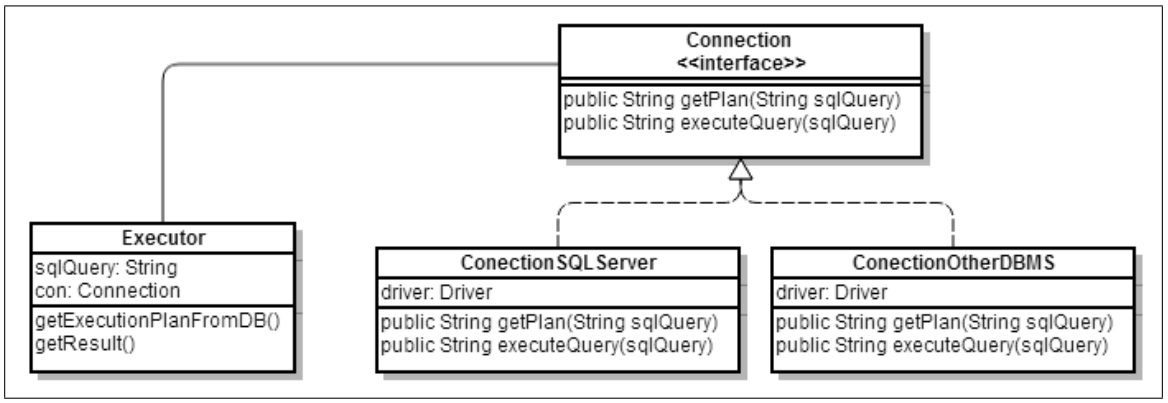


Figura 5.5: Diagrama de clases para la ejecución de la consulta en SQL estándar.

De esta forma la herramienta consta de 3 módulos o paquetes, de los cuales la entrada es el módulo de análisis y la salida es el módulo de ejecución. Esto permite ocultar el módulo de

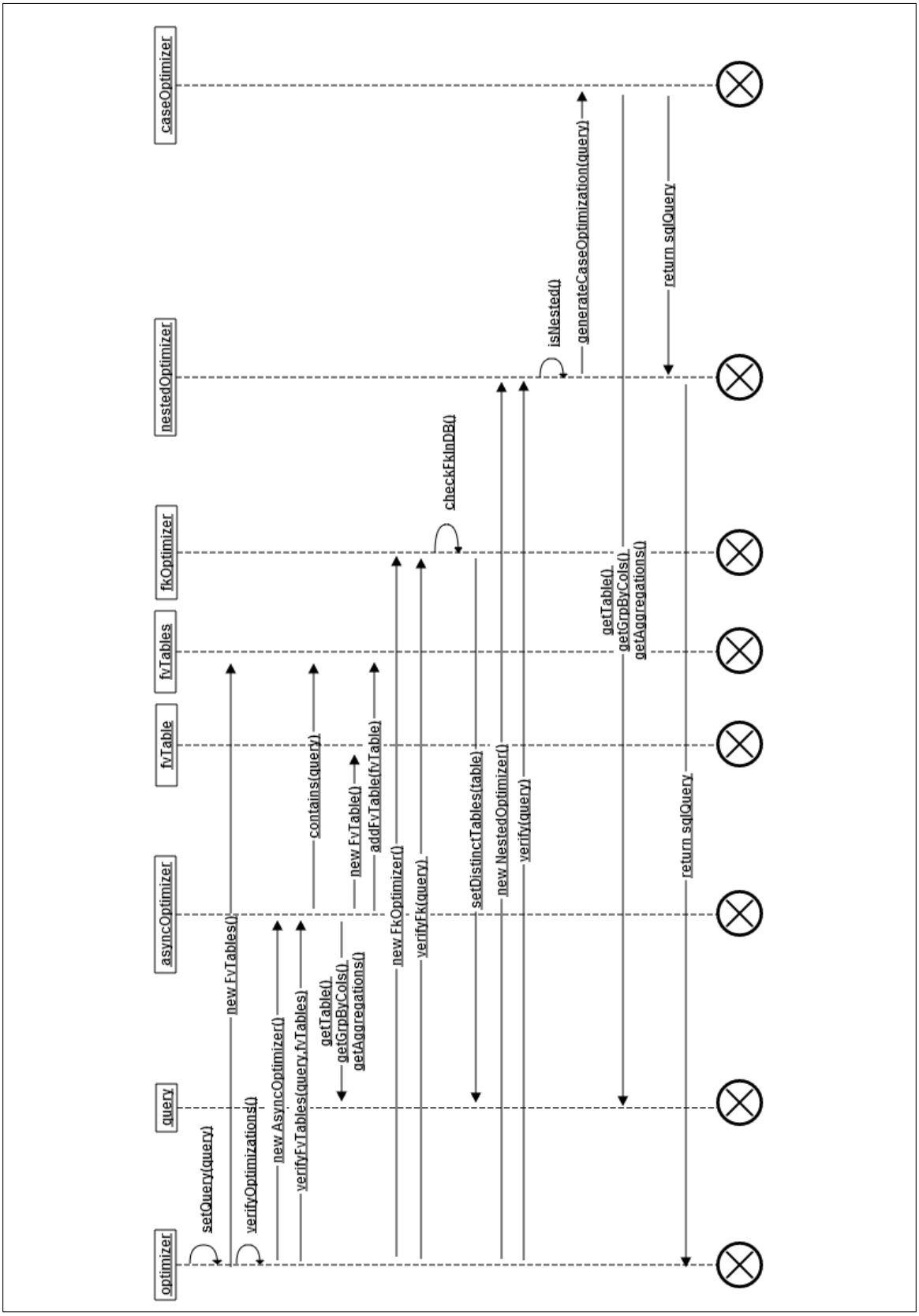


Figura 5.4: Diagrama de actividades para la aplicación de optimizaciones.

optimización de tal forma que la ejecución de una consulta con agregaciones horizontales optimizada es transparente, lo cual le proporciona facilidad de uso pero también ofrece la posibilidad de analizar el código SQL generado y el plan de ejecución utilizado para obtener el resultado.

5.3. Implementación

En esta sección se tratarán algunos aspectos de la programación de la herramienta que pueden servir como pauta para la implementación en otro SMBD. El SMBD utilizado durante el desarrollo fue SQL Server 2008 R2 y se utilizó un Driver JDBC para la conectividad con JAVA. La implementación se realizó con la premisa de no manipular los datos de la base de datos dentro del programa, sólo cuando esto fuera necesario y de estar manera hacer más eficiente la herramienta.

5.3.1. Análisis de la consulta.

Primeramente fue necesario implementar el módulo de análisis que acepta una consulta con agregaciones horizontales y en general consultas válidas de SQL. Para aceptar todo o parte del dialecto SQL del SMBD se tendría que escribir un analizador sintáctico que también permita y valide las expresiones de agregaciones horizontales. Esto último no es el objetivo de la tesis y por lo tanto se utilizó otra técnica para analizar y validar las consultas.

La idea es que la consulta de entrada sea analizada buscando expresiones de agregaciones horizontales $HAGG(ABY R_1, R_2, ..., R_d)$, si la consulta contiene estas expresiones se sustituyen las agregaciones horizontales por una agregación genérica `COUNT(A)` y se añade a la cláusula Y los atributos $R_1, R_2, ..., R_d$. Por ejemplo:

```
SELECT L_ORDERKEY,SUM(L_EXTENDEDPRICE BY L_LINENUMBER)
    FROM LINEITEM GROUP BY L_ORDERKEY;
```

Cambia por

```
SELECT L_ORDERKEY,COUNT(L_EXTENDEDPRICE)
    FROM LINEITEM GROUP BY L_ORDERKEY,L_LINENUMBER;
```

Esta segunda consulta está escrita en SQL estándar por lo que el SMBD puede analizarla sintáctica y semánticamente. Esto se puede realizar sin ejecutar la consulta, basta con pedir el plan de ejecución de dicha consulta para verificar si ésta bien formada o no.

Una vez que el SMBD valida por nosotros la consulta es necesario convertirla al modelo orientado a objetos el cuál permitirá validar las restricciones referentes a las agregaciones horizontales.

5.3.2. Optimización de la consulta

Una vez recibido el modelo orientado a objetos se procede a aplicar las optimizaciones. La interacción con el SMBD tendrá dos objetivos, primero acceder al catálogo del sistema para la lectura de restricciones de llaves foráneas así como los nombres de tablas y columnas y los tipos de datos; en segundo lugar para obtener los d valores por los cuales se transpondrá la tabla y que corresponden a los valores de las columnas de transposición R especificadas en la agregación horizontal.

Estos serán los únicos datos que se leerán de la base de datos a la herramienta pues ellos son necesarios para construir la optimización CASE. Los tipos de datos de las columnas de transposición son necesarios pues recordemos que sus valores pasarán a ser nombres de columnas y en los SMBD se tienen reglas sobre esta nomenclatura. Cuando los atributos sean algún tipo de cadena de caracteres deberemos verificar la longitud, para las fechas y números en general se deberá hacer una conversión a cadena de caracteres. Los valores deberán ser sanitizados para eliminar cualquier símbolo no valido (como espacios y puntos). En cuanto a los números, se debe establecer un caractér no numérico al inicio pues un nombre de una columna no puede iniciar con un número. En todos estos eventos se eligió el símbolo de guión bajo (_) para ayudar a sanitizar los nombres este símbolo es válido en los nombres de columnas.

Una vez leídos los datos de las columnas de transposición se deberán aplicar las optimizaciones propuestas anteriormente. Las tablas intermedias también deberán tener una nomenclatura especial, en SQL Server una tabla temporal comienza con ##. La ejecución de las consultas deberá siempre ser almacenada en una tabla temporal para ser reutilizada por alguna optimización y para no hacer la herramienta muy lenta ya que los resultados pueden tener demasiados datos como para caber en memoria y se puede alentar el sistema por falta de memoria.

Tanto la lectura del catálogo como la lectura de los datos de transposición no representan un alto costo ya que debemos tener en cuenta que el sistema no puede transponer una tabla a más de 1024 columnas que es el límite en este SMBD. Leer 1024 datos primitivos dentro de la herramienta no es una cantidad que pueda comprometer de manera significativa el desempeño de la herramienta.

5.3.3. Ejecución de la consulta

En cuanto a la ejecución no se tienen demasiadas consideraciones de programación. Por seguridad la herramienta almacenará los resultados en una tabla temporal para evitar perder tiempo de procesamiento si es que se trata de una cantidad considerable de datos. El sistema ofrece una opción al usuario para devolverle los datos bajo su responsabilidad de la inestabilidad del sistema causado por la posible falta de memora principal para mostrar los datos.

Otro aspecto referente a la ejecución será la obtención del plan de ejecución de la optimización. El plan de ejecución puede obtenerse de manera fácil mediante la biblioteca de conectividad para este SMBD. Por último se debe considerar el tiempo de ejecución de la consulta para fines informativos al usuario acerca del desempeño de la ejecución. El lenguaje de programación JAVA tiene distintas herramientas que proporcionan esta infor-

mación como la clase `System` y su método `currentTimeMillis()` el cual es utilizado antes y después de la evaluación de la consulta para medir su tiempo de ejecución.

Capítulo 6
Experimentación.

Los experimentos que se muestran en esta Sección tienen como objetivo exponer la eficiencia de las nuevas optimizaciones así como ejemplificar la utilidad de las funciones de agregación horizontales en trabajos de preparación y transformación de los datos. Por un lado probaremos la eficiencia de las optimizaciones en una base de datos sintética ya que podemos controlar la cantidad de datos y modificar el esquema para ejemplificar más escenarios. En cuanto a la utilidad se trabajaron bases de datos reales de las cuales se presentan dos tipos de problemas distintos y que son resueltos con agregaciones horizontales, el tamaño de esta base de datos es pequeña en comparación con la base de datos sintética, por esta razón no se hicieron experimentos de eficiencia sobre los datos reales.

Separamos los experimentos dependiendo de las bases de datos y equipos donde fueron ejecutadas se especifican en cada Sección.

6.1. Experimentos en una base de datos sintética

A continuación se presentan una serie de experimentos y resultados en una base de datos generada artificialmente. Esta base de datos puede ser generada con un factor de escala el cual permite manipular el número de registros en algunas de sus tablas. Este factor nos sirvió para experimentar y observar el comportamiento de las optimizaciones a través de diferentes tamaños de las tablas de datos.

6.1.1. Hardware

El hardware en el que se realizaron los experimentos consta de un equipo de cómputo con las siguientes características:

- Procesador Intel i7-2600k @3.40GHz.

  Cache L2: 256 KB por núcleo.

  Cache L3: 8MB compartidos por todos los núcleos.

  4 núcleos y 8 hilos de ejecución.
- 4 GB en memoria principal (RAM) con frecuencia de 677 MHz.
- 366 GB en memoria secundaria (disco duro).

6.1.2. Software

El equipo de cómputo tiene instalado el sistema operativo Windows 7 Ultimate y el sistema manejador SQL Server 2008 R2 Enterprise Edition[17]. El equipo está configurado para

proveer la máxima cantidad de recursos al SMBD para que los experimentos no se vean afectadas por procesos externos que involucren un alto uso de recursos del sistema.

En este SMBD es posible especificar dos parámetros en cuanto a la memoria y el número de procesadores:

- *MAXDOP: Max Degree Of Parallelism*, el cual acepta valores desde 0 hasta p, donde p es el número de procesadores en el sistema. El valor 0 indica utilizar todos los procesadores disponibles.

- *Maximum server memory (MB)*: La cantidad de memoria principal (RAM) que el sistema podrá utilizar para procesar las consultas.

Estos parámetros permitieron configurar distintos escenarios en los experimentos. Los experimentos fueron ejecutados con p=1,2 y 4 además de variar el tamaño de M (Maximum server memory) en 1,2 y 4 GB.

El lenguaje de programación utilizado es Java y se utilizan solamente las bibliotecas estándar junto con la biblioteca de conectividad a la base de datos JDBC.

Cada experimento se realizó varias veces bajo las mismas circunstancias, para ello después de cada ejecución es necesario indicarle al SMBD que escriba los datos del cache del buffer hacia el disco mediante la instrucción `CHECKPOINT` y que el sistema vacíe la caché de los planes de ejecución y remueva los buffers limpios de la memoria a través de las instrucciones `DBCC DROPCLEANBUFFERS`, `DBCC FREEPROCCACHE`, `DBCC FLUSHPROCINDB` [14] o alternativamente reiniciar todo el SMBD[1]. En nuestro caso la segunda opción fue la mejor ya que al limpiar los buffers y el cache del SMBD a través de estas instrucciones no se desocupa la memoria principal realmente ya que el monitor de uso de memoria del sistema operativo se mantenia igual, haciendo que el sistema operativo se volviera más lento en ejecuciones posteriores. En distintos foros de expertos se consultó este comportamiento y en muchos de ellos se coincidía en que la única solución de liberar completamente la memoria era mediante el reinicio del servidor [5]. Esta opción fue la que permitió mantener ambientes comparables para cada ejecución y así obtener tiempos más justos entre las ejecuciones.

6.1.3. Base de datos TPCH

Como se menciona en la introducción se utilizó una base de datos construida artificialmente. La base de datos utilizada fue creada y es mantenida por el consorcio TPC[27] dedicado a modelar medidas de desempeño para comparar distintos sistemas manejadores de bases de datos en distintos ambientes.

Se eligió esta base de datos ya que al ser orientada a OLAP puede ser generada con una gran cantidad de datos. El esquema de esta base de datos es un esquema tipo estrella el cual tiene como tabla de hechos a la tabla *LINEITEM*, esta tabla es la más grande dentro de esta base de datos. El esquema lógico de la base de datos *TPCH* se muestra en la Figura 6.1.

[1] El reinicio del SMBD se realizo con las intrucciones `net stop MSSQLSERVER` y `net start MSSQLSERVER`

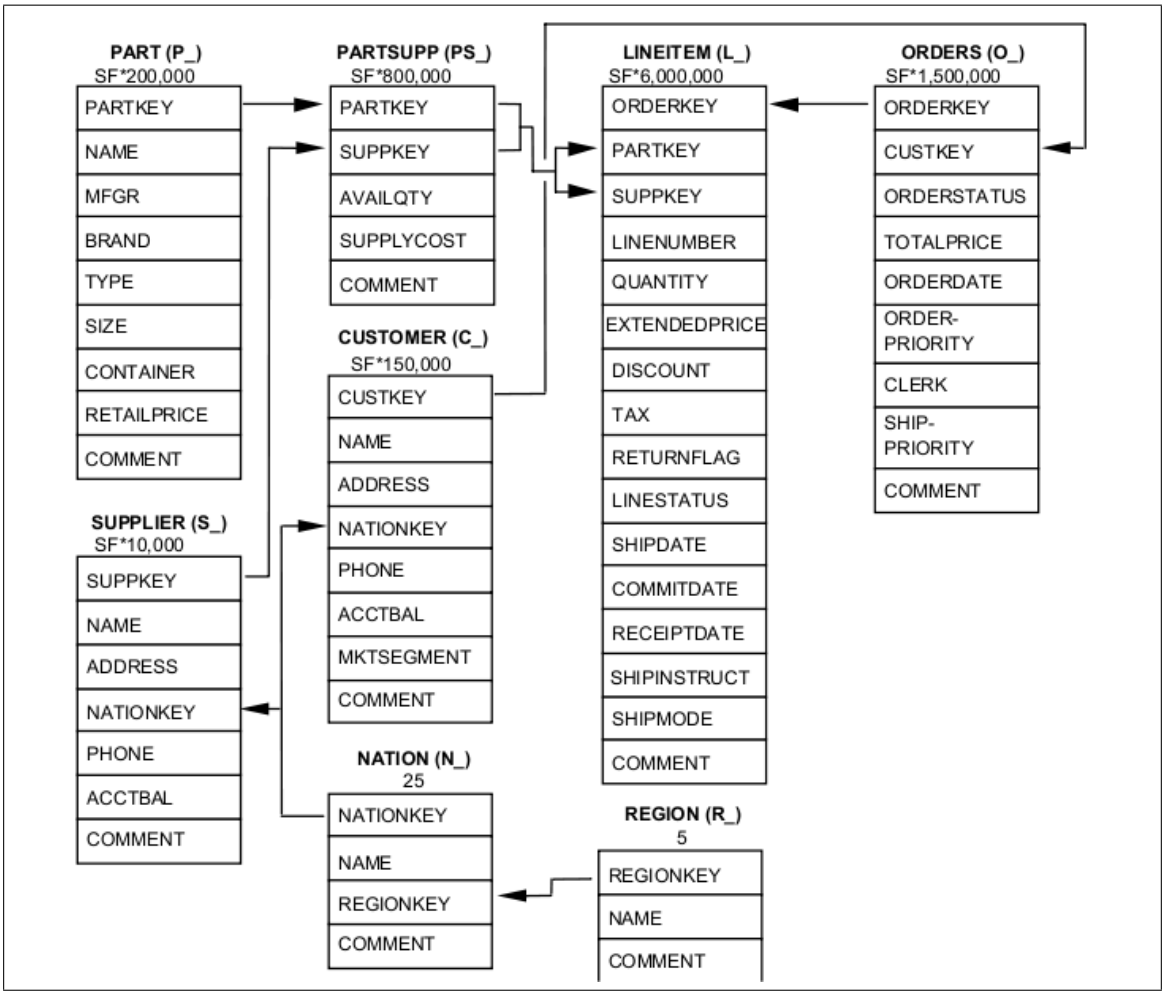


Figura 6.1: Esquema lógico de la base de datos sintética *TPCH*

En la Figura anterior cada tabla tiene en su parte superior las siglas *SF* correspondientes a *Scale Factor* (Factor de escala) el cual permite variar el número de registros que contiene cada tabla. A su vez este diagrama permite saber cuál es el tipo de relación y la multiplicidad entre las tablas.

En los experimentos se utilizan principalmente 5 tablas:

- *LINEITEM*
  - Es la tabla más grande.
  - El número de registros en esta tabla con el factor de escala 1 es de alrededor de seis millones.
  - Su clave primaria es (L_ORDERKEY,L_LINENUMBER), ambos campos de tipo integer.
  - Sus demás atributos son de tipo integer, float, char, date y varchar.
  - Tiene dos llaves foráneas.
    1. (L_ORDERKEY) que la relaciona con *ORDERS*.

2. (L_PARTKEY,L_SUPPKEY) que la relaciona con *PARTSUP*.

- *ORDERS*
  - Es segunda tabla más grande.
  - El número de registros en esta tabla con el factor de escala 1 es de cerca de un millón y medio.
  - La relación con *LINEITEM* es en promedio de 1 a 4, es decir, cada registro en *ORDERS* está relacionado en promedio con 4 registros en *LINEITEM*.
  - Su clave primaria es (L_ORDERKEY).
  - Sus atributos son de tipo integer, float, char, date y varchar
  - Tiene una llave foránea (O_CUSTKEY) que la relaciona con *CUSTOMER*.
- *PARTSUPP*
  - Es la tercera tabla más grande.
  - El número de registros en esta tabla con el factor de escala 1 es de cerca de ochocientos mil.
  - La relación con *LINEITEM* es en promedio de 1 a 7.
  - Su clave primaria es (PS_PARTKEY,PS_SUPPKEY).
  - Sus atributos son de tipo integer, float y varchar
  - Tiene dos llaves foráneas.
    1. (PS_PARTKEY) que la relaciona con *PART*.
    2. (PS_SUPPKEY) que la relaciona con *SUPPLIER*.
- *PART*
  - El número de registros en esta tabla con el factor de escala 1 es de cerca de doscientos mil.
  - Su clave primaria es (P_PARTKEY).
  - Sus atributos son de tipo integer, float y varchar
  - La tabla a la que está relacionada es *PARTSUPP* pero puede relacionarse también con *LINEITEM* en una proporción promedio de 1 a 30.
- *SUPPLIER*
  - El número de registros en esta tabla con el factor de escala 1 es de cerca de diez mil.
  - Su clave primaria es (S_SUPPKEY).
  - Sus atributos son de tipo integer, float y varchar
  - La tabla a la que está relacionada es *PARTSUPP* pero puede relacionarse también con *LINEITEM* en una proporción promedio de 1 a 600.

Se generaron 3 bases de datos, cada una con un factor de escala diferente. El factor de escala fue igual a 1,2 y 3 por lo que la tabla *LINEITEM* tuvo 6 millones, 12 millones y 18 millones respectivamente.

6.1.4. Optimización de consulta anidada

En esta serie de experimentos se estudia el comportamiento de la optimización respecto a la consulta no optimizada bajo distintos escenarios de hardware y software descritos en la Sección anterior. El objetivo principal es saber si la optimización es mejor en todos los escenarios propuestos así como medir en qué porcentaje es mejor respecto a la consulta no optimizada.

Los parámetros de los experimentos fueron:

- n: El tamaño de la tabla.
- p: El número de procesadores utilizados por el SMBD.
- M: La cantidad de memoria máxima que utiliza el SMBD.

La consulta con la que se realizó la experimentación es la siguiente:

```
SELECT  L1.L_LINESTATUS,
L1.L_SHIPINSTRUCT,
L1.L_RETURNFLAG,
SUM(L1.SUM_L_TAX_1 BY L1.L_LINENUMBER) AS SUM_L_TAX_2
  FROM (
SELECT  L_LINESTATUS,
L_SHIPINSTRUCT,
L_RETURNFLAG,
L_LINENUMBER,
SUM(L_TAX BY L_SHIPMODE) AS SUM_L_TAX_1
  FROM LINEITEM
  GROUP BY  L_LINESTATUS,
    L_SHIPINSTRUCT,
    L_RETURNFLAG,
    L_LINENUMBER
) L1
  GROUP BY L1.L_LINESTATUS,
   L1.L_SHIPINSTRUCT,
   L1.L_RETURNFLAG ;
```

Esta consulta se ejecuta sobre la tabla *LINEITEM* y contiene dos funciones de agregación horizontal, además se tiene solamente una consulta anidada, esta consulta interna L1 generará los datos sobre los que se ejecuta la consulta externa que tiene una función de agregación horizontal donde el atributo medida L1.SUM_L_TAX_1 hace referencia al resultado de una agregación horizontal de la consulta interna L1.

En la consulta no optimizada se tendrán que realizar dos lecturas de la tabla *LINEITEM* ya que por un lado se deben obtener los d valores distintos correspondientes al atributo L_SHIPMODE y por otro se debe ejecutar la consulta L1, a continuación se muestra la serie de pasos que ejecuta las consulta no optimizada.

Consulta no optimizada:

- Obtener los d valores distintos para L_SHIPMODE de la tabla *LINEITEM*.
- Sustituir las referencias de L1.SUM_L_TAX_1 en la consulta externa.
- Ejecutar la consulta L1 y almacenar su resultado.
- Ejecutar la consulta externa en base al resultado de la consulta L1.

Para mejorar la ejecución de esta consulta se busca reducir el número de lecturas de *LINEITEM*, esto se logra obteniendo una tabla reducida que contenga los valores distintos de L_SHIPMODE y los datos agregados L_TAX. Al ser una tabla materializada y reducida se obtiene un mejor desempeño pues los datos se encuentran en memoria principal. Se describen los pasos para evaluar la consulta de forma optimizada.

Consulta optimizada:

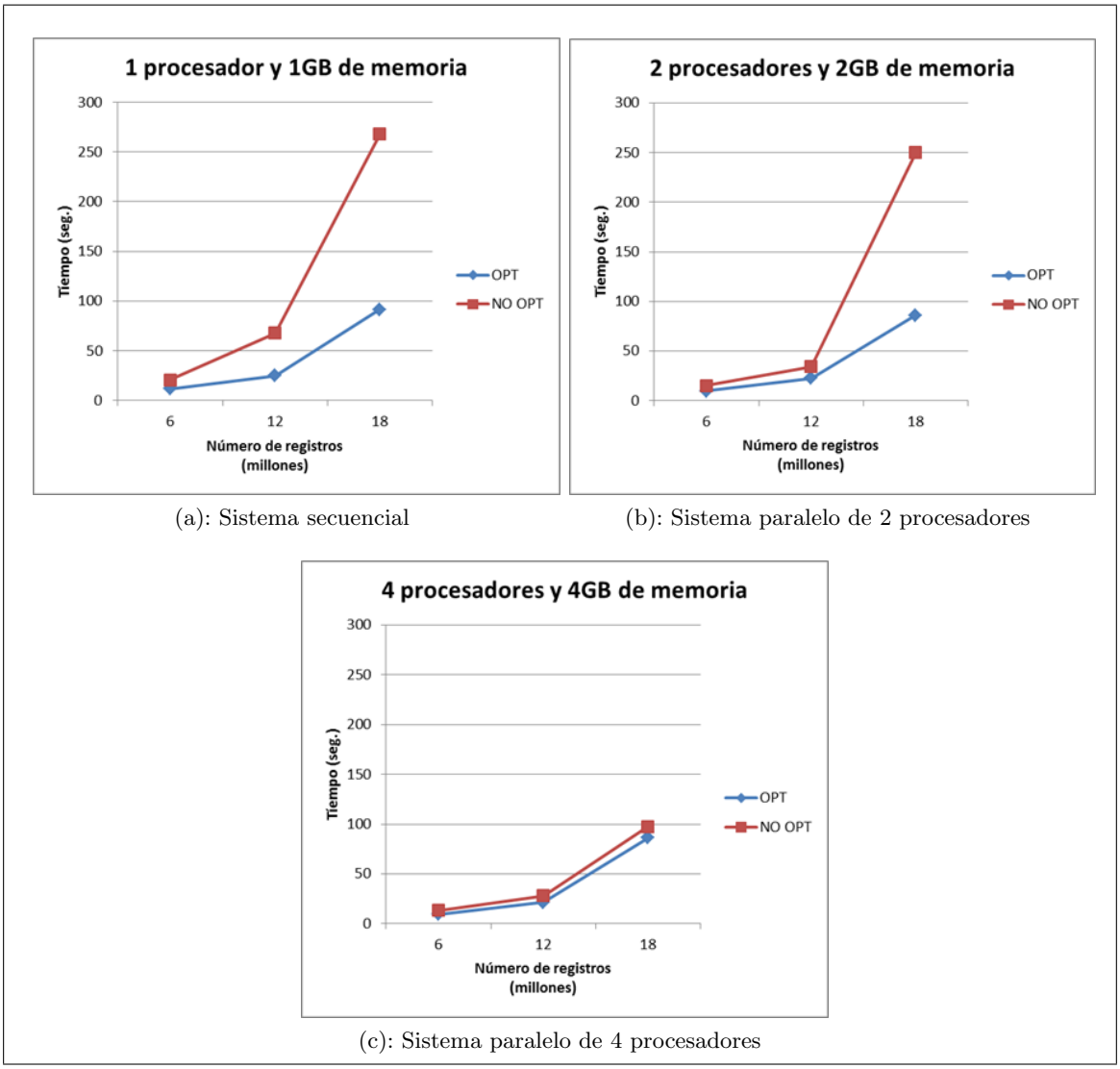
- Leer *LINEITEM* para construir una tabla temporal F_V donde se calculará una función de agregación SUM(L_TAX) agrupando por los atributos L_LINESTATUS, L_SHIPINSTRUCT, L_RETURNFLAG, L_LINENUMBER, L_SHIPMODE.
- Obtener los d valores distintos para L_SHIPMODE de la tabla F_V.
- Sustituir las referencias de L1.SUM_L_TAX_1 en la consulta externa.
- Ejecutar la consulta L1 a partir de la tabla F_V y almacenar su resultado.
- Ejecutar la consulta externa en base al resultado de la consulta L1.

La Tabla 6.1 muestra los resultados de la ejecución de ambas consultas bajo distintos escenarios de las variables n,p y M.

n (millones)	Recursos	OPT(seg.)	NO OPT(seg.)	Mejora
6	p=1, M=1	11.5	20.5	43.90 %
	p=2, M=2	9.8	15.2	35.50 %
	p=4, M=4	9.4	13.2	28.80 %
12	p=1, M=1	24.8	67.6	63.30 %
	p=2, M=2	22.6	34.2	33.90 %
	p=4, M=4	21.2	28.1	24.60 %
18	p=1, M=1	91.6	268.0	65.80 %
	p=2, M=2	86.1	250.0	65.60 %
	p=4, M=4	86.4	97.5	11.40 %

Tabla 6.1: Resultados de experimentos de optimización de consulta anidada

En todos los escenarios la optimización obtuvo un mejor desempeño que la consulta no optimizada. La mejora va del 11 % al 65 %, siendo el mejor escenario un sistema secuencial con 1GB (a), por el contrario, el escenario en donde se obtuvo una menor mejoría fue el sistema con cuatro procesadores y 4GB (c). Los resultados completos de la Tabla se pueden ver de forma gráfica en la Figura 6.2.



(a): Sistema secuencial

(b): Sistema paralelo de 2 procesadores

(c): Sistema paralelo de 4 procesadores

Figura 6.2: Resultados de experimentos de optimización de consulta anidada.

Uno de los factores que influyen de manera significativa es el parámetro M, el tamaño de la memoria del sistema. Recordemos que uno de los mecanismos de optimización de lectura a disco es el cache del buffer de datos del sistema, para el SMBD el tamaño de este cache depende de la máxima cantidad de memoria disponible. Esto quiere decir que si se tiene más memoria entonces las lecturas a una tabla serán más eficientes pues los datos (o algunos de ellos) se podrán acceder mediante la memoria principal en lugar de realizar lecturas a memoria secundaria, esto reduce las lecturas a disco pero aumenta las lecturas a memoria principal.

Este efecto se puede apreciar en los resultados de la Figura 6.2. Como se explica en la Sección referente a la base de datos, TPCH con factores de escala 1, 2 y 3 genera la tabla *LINEITEM* con un tamaño de 0.7GB ,1.5GB y 2.1 GB respectivamente.Esto quiere decir, por ejemplo, que la tabla *LINEITEM* en TPCH con factor de escala 2 podrá ser almacenada en cache si el sistema cuenta con 2GB de memoria o más.

Observamos que cuando $n = 6$ el comportamiento en los tres sistemas es similar para cada consulta (Figuras 6.2: a,b, c) ya que en los tres la tabla *LINEITEM* puede almacenarse en el cache del buffer[2]. Pero si $n = 12$ entonces M influye en la ejecución en el sistema con $M = 1$ ya que al no poder almacenar todos los datos en el cache, es necesario hacer más lecturas a disco. Esto puede verse cuando $n = 18$ y $M = 4$ (c) ya que al tener todos los datos en cache, la consulta no optimizada baja de manera considerable el tiempo de ejecución pues hace más lecturas lógicas[3] que lecturas físicas[4] en comparación con los otros sistemas.

Factor	Lecturas anticipadas	Lecturas físicas	Lecturas lógicas
p=2 y M=2GB			
1	104,516	996	313,578
2	626,007	4,724	626,910
3	939,039	9,403	940,374
p=4 y M=4GB			
1	104,516	1,712	314,804
2	208,957	3,461	629,056
3	939,026	12,308	942,831

(a) Consulta anidada sin optimización.

Factor	Lecturas anticipadas	Lecturas físicas	Lecturas lógicas
p=2 y M=2GB			
1	104,516	25	104,550
2	208,957	54	208,994
3	313,457	1,282	313,482
p=4 y M=4GB			
1	104,516	121	104,700
2	208,957	234	209,140
3	313,462	3,438	314,266

(b) Consulta anidada optimizada.

Tabla 6.2: Lecturas físicas y logicas para la optimización de consulta anidada.

En la Tabla 6.2 se muestran las lecturas físicas y lógicas para las primeras dos configuraciones. La primera columna corresponde al factor de escala de la base de datos, para este caso el factor es muy similar al tamaño de la base de datos, es decir, un factor de 1 significa que la base de datos ocupa alrededor de 1GB de espacio. En cada Tabla podemos ver en la columna de lecturas físicas que conforme aumentamos el tamaño de la base de datos el número de lecturas físicas aumenta pues para M=1GB solamente la base de datos con factor 1 puede almacenarse en memoria. Cuando se aumenta M=2GB las bases de datos con factores 1 y 2 pueden ser leídas por completo desde la memoria principal, no así para el factor 3. Siguiendo con cada tabla por separado podemos observar que las lecturas lógicas se mantienen muy similares sin importar el factor, esto sucede porque una vez que se tienen los datos en memoria se deben acceder para computar el resultado mediante el mismo plan de ejecución.

Si ahora comparamos ambas Tablas, podremos observar una disminución en las lecturas lógicas. La disminución de las lecturas físicas es considerable pero en este caso es necesario remitirse también a la columna **Lecturas anticipadas**[5]. Ya que las lecturas anticipadas

[2]Un buffer es una pagina de 8KB en memoria principal. El cache del buffer es dividido en paginas de 8KB [13].

[3]Una lectura lógica es el número de páginas leidas del cache[16].

[4]Una lectura física es el número de páginas leidas desde el disco[16].

[5]Una lectura anticipada es el número de páginas colocadas en cache para la ejecución de la consulta[16]. Este mecanismo de optimización del SMBD realiza lecturas dispersas y coloca los datos en cache, se puede ver como una lectura física optimista que coloca los datos en memoria principal antes de ser requeridos por la consulta. Al ser dispersa puede que algunos datos necesarios para el procesamiento de la consulta no se hayan leído con antelación y para obtenerlos se debe hacer una lectura física.

pueden haber obtenido muchos de los valores necesarios para la consulta, las lecturas físicas se ven disminuidas. Al tener una M mayor se pueden realizar más lecturas anticipadas disminuyendo la probabilidad de realizar lecturas físicas extra. Esto se puede ver en ambas tablas. En general las lecturas lógicas son las que más describen el comportamiento de la optimización pues en ellas se refleja una menor cantidad de lecturas para obtener el resultado.

A pesar de que la optimización no aprovecha el aumento de los recursos del sistema la diferencia en el tiempo de ejecución con la consulta no optimizada es significativa pues se hacen menos lecturas de los datos. Estos hallazgos nos muestran que no importa que tan rápido se procesen los datos en memoria principal, el mayor tiempo consumido por una consulta de este tipo es en la lectura de los datos.

6.1.5. Optimización asíncrona

Para esta clase de optimización los experimentos también se realizaron con distintos valores de n, M y p en los cuales se busco nuevamente saber el grado de mejora. Para poder comparar el desempeño de la optimización se definirán dos consultas y se comparará la ejecución de ambas por separado contra la ejecución optimizada que propuesta.

```
--Consulta 1
SELECT
      L_SHIPINSTRUCT,
      L_LINESTATUS,
      L_RETURNFLAG,
      SUM(L_QUANTITY BY L_SUPPKEY)
  FROM LINEITEM
  GROUP BY
      L_SHIPINSTRUCT,
      L_LINESTATUS,
      L_RETURNFLAG ;
--Consulta 2
SELECT
      L_SHIPINSTRUCT,
      L_RETURNFLAG,
      SUM(L_QUANTITY BY L_SUPPKEY)
  FROM LINEITEM
  GROUP BY
      L_SHIPINSTRUCT,
      L_RETURNFLAG ;
```

Figura 6.3: Consultas para la experimentación de optimización asíncrona.

En la Figura 6.3 tenemos las dos consultas en donde podemos observar que los atributos de agrupación (vertical y horizontal) de la *Consulta 2* son un subconjunto de los atributos de agrupación de la *Consulta 1*, además la función de agregación es la misma y sobre el mismo atributo.

Ejecución no optimizada:

- La *Consulta 1* lee la tabla *LINEITEM* para procesar la consulta con agregación horizontal.
- La *Consulta 2* lee la tabla *LINEITEM* para procesar la consulta con agregación horizontal.

Como podemos observar para ejecutar estas consultas se lee y se procesa, al menos, 2 veces la tabla *LINEITEM*.

Ejecución optimizada

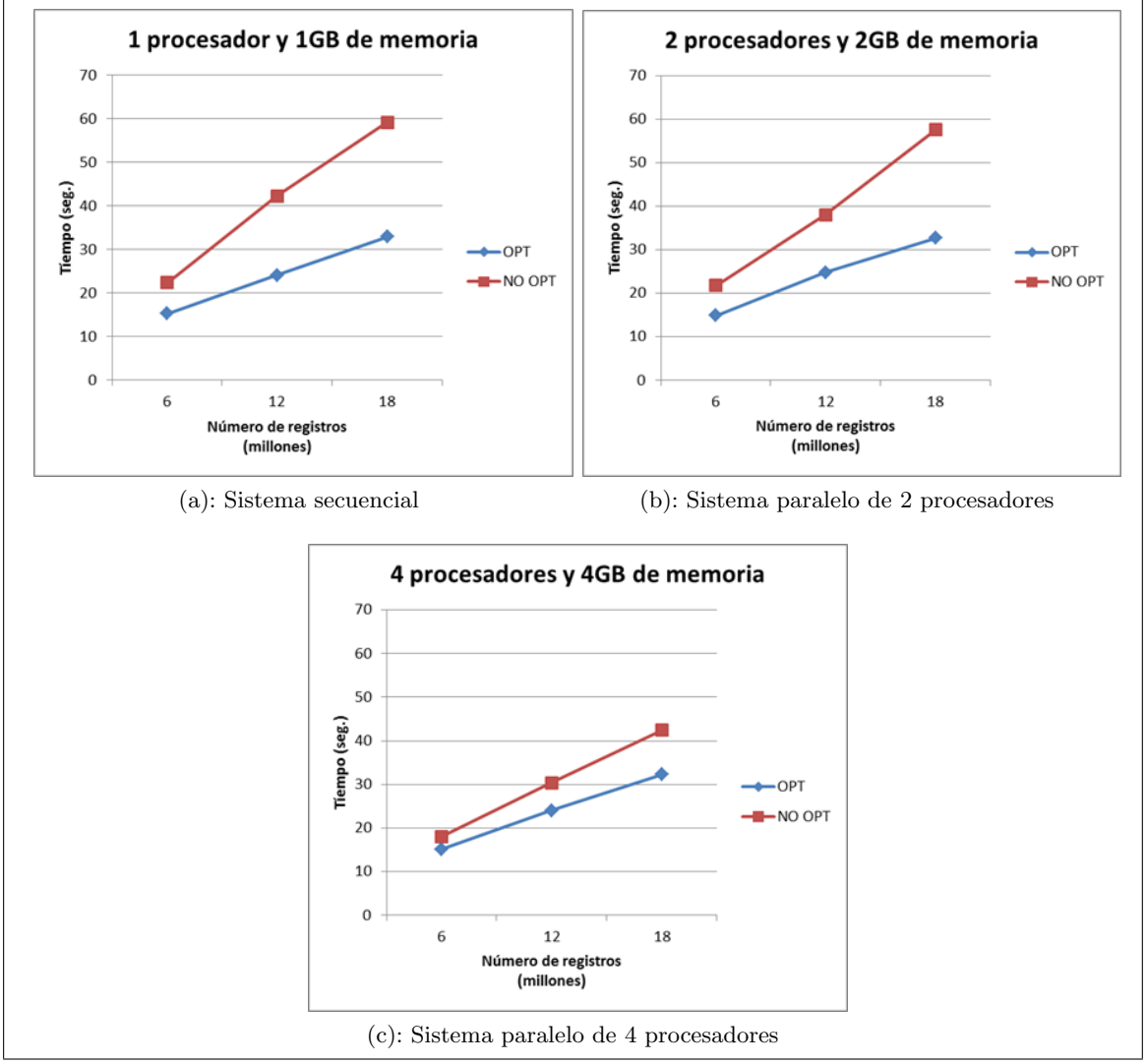
- La *Consulta 1* lee la tabla *LINEITEM* para procesar la consulta con agregación horizontal. Recordemos que para este procesamiento se creó una tabla F_V la cual almacena una agregación parcial.
- La *Consulta 2* lee la tabla F_V de la *Consulta 1*, calculada asíncronamente, la cual contiene la agregación parcial requerida para la consulta actual.

Esta forma de procesar las consultas pretende acelerar tanto la lectura de los datos así como el cómputo de la agregación.

n (millones)	Recursos	NO OPT(seg.)	OPT(seg.)	Mejora
6	p=1, M=1	22.3	15.3	31.40 %
	p=2, M=2	21.8	14.9	31.70 %
	p=4, M=4	18.0	15.1	16.10 %
12	p=1, M=1	42.2	24.0	43.10 %
	p=2, M=2	38.1	24.8	34.90 %
	p=4, M=4	30.4	24.0	21.10 %
18	p=1, M=1	59.2	32.9	44.40 %
	p=2, M=2	57.6	32.7	43.20 %
	p=4, M=4	42.4	32.3	23.80 %

Tabla 6.3: Resultados de experimentos de optimización asíncrona.

La Tabla 6.3 muestra los resultados de los experimentos. La optimización tuvo una mejora entre 30 % y 40 % a excepción del escenario donde se tienen $n = 4$ y $M = 4$. Los resultados de esta Tabla pueden verse de forma gráfica en la Figura 6.4. Los tiempos suman el tiempo de ejecución de la *Consulta 1* y de la *Consulta 2*.



(a): Sistema secuencial

(b): Sistema paralelo de 2 procesadores

(c): Sistema paralelo de 4 procesadores

Figura 6.4: Resultados de experimentos de optimización asíncrona.

De esta experimentación se pudo constatar que la ejecución optimizada mejora en mucho el tiempo de respuesta. El cache del buffer del sistema también influye pues en la ejecución normal, al tener dos lecturas de la tabla *LINEITEM*, algunos datos se mantienen en memoria. Por otra parte la ejecución optimizada funciona como el cache del SMBD ya que al mantener resultados parciales de las consultas anteriores reduce el tiempo de lectura de los datos y además reduce también el tiempo de procesamiento.

En la Tabla 6.4 podemos observar las lecturas físicas y lógicas hechas por la ejecución de las consultas *Consulta 1* y *Consulta 2* por separado y con la optimización asíncrona. En estas Tablas podemos observar que la optimización disminuye las lecturas lógicas en promedio un 50 % ya que aprovechamos el resultado parcial de la primer consulta. Por otro lado, el número de lecturas físicas varía por que depende de las lecturas anticipadas. Nuevamente podemos observar que al tener más memoria

se realizan más lecturas anticipadas lo cual impacta en el numero de lecturas físicas. En conclusión esta optimización muestra una reducción de las lecturas lógicas lo cual proporciona una mejora en la ejecución de consultas similares ahorrando tiempo de lectura y procesamiento de los datos.

Factor	Lecturas anticipadas	Lecturas físicas	Lecturas lógicas
p=2 y M=2GB			
1	104,308	300	209,010
2	208,525	908	417,460
3	623,518	393	625,984
p=4 y M=4GB			
1	104,308	1,638	209,010
2	208,525	3,275	417,460
3	312,795	4,921	625,984

(a) Consultas sin la optimización.

Factor	Lecturas anticipadas	Lecturas físicas	Lecturas lógicas
p=2 y M=2GB			
1	104,308	459	104,769
2	208,525	588	208,994
3	312,795	143	313,256
p=4 y M=4GB			
1	104,308	1,638	104,769
2	208,525	3,277	208,994
3	312,795	4,921	313,256

(b) Consultas con la optimización.

Tabla 6.4: Lecturas físicas y logicas para la optimización asíncrona.

6.1.6. Optimización de llave foránea

Para esta optimización es necesario tener dos tablas y que entre ellas se defina una llave foránea. Las tablas utilizadas en este experimento son *LINEITEM* y *SUPPLIER*.

Una agregación horizontal tiene un límite práctico en el número de valores distintos d que se pueden transponer. Este límite lo establece el SMBD el cual no permite generar tablas o vistas donde el número de columnas sea mayor a 1024 [15]. Para esta experimentación se modificará un poco la base de datos de tal manera que nos permita experimentar con este límite.

De acuerdo a la Sección 6.1.3 la tabla *SUPPLIER* tiene *SF**10000 registros y *LINEITEM* no tiene definida directamente una llave foránea a *SUPPLIER*. Las modificaciones serán las siguientes:

- Dejaremos en *SUPPLIER* sólo los primeros 1023 registros, los cuales tendrán valores desde 1 hasta 1023.
- En *LINEITEM* el valor de *L_SUPPKEY* sera igual a $L\_SUPPKEY \ mod \ 1023$ [6].
- Se definirá una llave foránea en el atributo *L_SUPPKEY* de *LINEITEM* que tenga referencia a la llave primaria *S_SUPPKEY* de *SUPPLIER*.

[6]Esto hará que una agregación horizontal sobre *L_SUPPKEY* genere 1023 columnas, dejando la posibilidad de tener una columna de agrupación vertical

Con estas modificaciones se obtuvo un escenario en el cual se pudo comparar el desempeño de la optimización de llave foránea en la base de datos TPCH. La Figura 6.5 muestra la consulta utilizada para esta experimentación.

```
SELECT
      L_ORDERKEY,
      SUM(L_EXTENDEDPRICE BY L_SUPPKEY)
FROM LINEITEM
GROUP BY  L_SUPPKEY,
  L_ORDERKEY;
```

Figura 6.5: Consulta para la experimentación de llave foránea

Ejecución no optimizada:

- Se obtienen los d valores distintos con la cláusula DISTINCT sobre la llave foránea L_SUPPKEY en LINEITEM.
- Se construye y se ejecuta la consulta CASE.

En esta ejecución leemos dos veces LINEITEM, una para obtener los valores distintos y otra para calcular el resultado de la agregación horizontal. La segunda lectura puede mejorarse por el uso del cache del sistema como se ha mencionado en los experimentos anteriores, aunque esto depende de la cantidad de memoria del sistema.

Ejecución optimizada:

- Se obtienen los d valores distintos de la llave primaria S_SUPPKEY en *SUPPLIER*.
- Se construye y se ejecuta la consulta CASE.

Para la ejecución optimizada, *LINEITEM* solamente fue leída una vez, por lo que esta ejecución pretende ser más eficiente que la anterior. La lectura de *SUPPLIER* no aumenta mucho el tiempo pues al hacer la proyección de la llave primaria se utiliza un índice lo cual hace que no se lea por completo la tabla *SUPPLIER*.

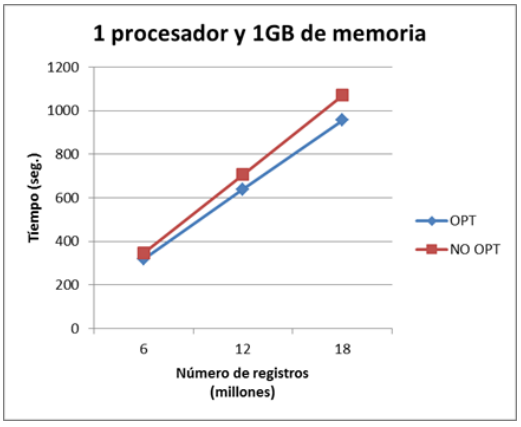
La consulta mostrada en la Figura 6.5 tiene una agregación horizontal en el atributo *L_SUPPKEY* y agrupada por *L_ORDERKEY*, el resultado de esta consulta genera una tabla de 1024 columnas, 1023 correspondientes al resultado de la agregación horizontal en L_SUPPKEY y la columna perteneciente a L_ORDERKEY. Dado que el resultado está dentro del límite del número de columnas en una tabla, no hubo problemas en la experimentación.

Los resultados de las ejecuciones de la consulta mostrada en la Figura 6.5 variando los parámetros n, M y p se muestran en la Tabla 6.5. El tiempo de ejecución mejoró alrededor del 10 % en todos los casos.

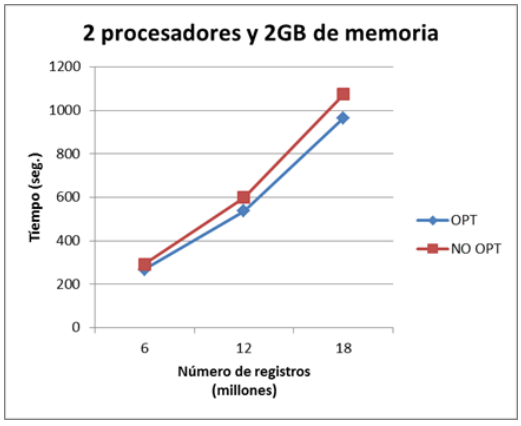
n (millones)	Recursos	NO OPT(min.)	OPT(min.)	Mejora
6	p=1, M=1	5.8	5.3	8.60 %
	p=2, M=2	4.9	4.5	8.20 %
	p=4, M=4	4.8	4.4	8.30 %
12	p=1, M=1	11.8	10.6	10.20 %
	p=2, M=2	10.0	8.9	11.00 %
	p=4, M=4	9.9	8.8	11.10 %
18	p=1, M=1	17.8	16.0	10.10 %
	p=2, M=2	17.9	16.1	10.10 %
	p=4, M=4	14.9	13.1	12.10 %

Tabla 6.5: Resultados de experimentos de optimización de llave foránea en minutos.

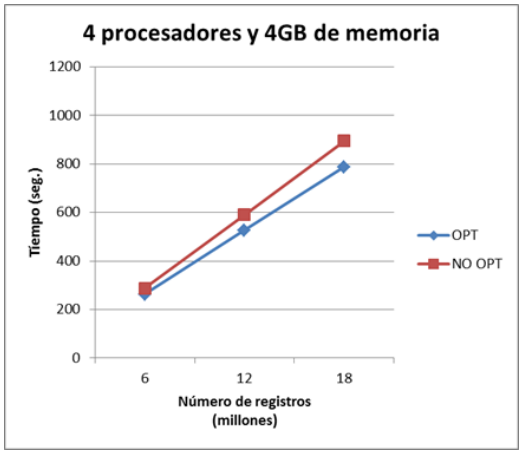
La Figura 6.6 muestra de forma gráfica los resultados de la Tabla 6.5. En estas gráficas podemos observar que si hay una diferencia entre la ejecución optimizada y la ejecución normal.



(a): Sistema secuencial



(b): Sistema paralelo de 2 procesadores



(c): Sistema paralelo de 4 procesadores

Figura 6.6: Resultados de experimentos de optimización de llave foránea.

Con estos resultados se pudo observar que la optimización mantiene una ganancia en todos los escenarios alrededor del 10 %. En general esta optimización se ve beneficiada con el crecimiento de la tabla que tiene la llave foránea pues el aumentar el numero de registros no influye en la obtención de los valores distintos obtenidos de la llave primaria a la cual se hace referencia.

6.2. Experimentos en una base de datos real

Los experimentos realizados en estas bases de datos corresponden a la colaboración realizada con la unidad de tuberculosis de un Instituto de Salud Público en México (INSP[7]) sobre tuberculosis [2] y [10]. En estos proyectos se realizan análisis estadísticos de las relaciones latentes entre la tuberculosis, otras enfermedades y distintos factores sociales. El análisis requerido por los investigadores necesita que los datos sean preparados antes de analizarlos, principalmente se necesita reunir tablas, seleccionar campos y crear columnas que recodifican los datos de otros campos.

Estos experimentos tienen como finalidad mostrar la utilidad de las funciones de agregación horizontales en tareas de pre-procesamiento. Las bases de datos utilizadas en esta experimentación no cuentan con muchos registros en comparación con la base de datos sintética por lo que los experimentos de esta sección se enfocan en la utilidad en lugar de mostrar la eficiencia.

Los ejemplos presentados en esta Sección corresponden al trabajo de preparación en los pasos de integración y transformación de datos. Los ejemplos omiten la información privada de los pacientes y solamente se muestran datos ilustrativos sobre la problemática y solución planteada.

6.2.1. Hardware

El hardware en el que se realizaron los experimentos consta de un equipo de cómputo con las siguientes características:

- Procesador Intel i5-3470 @3.20GHz.

  Cache L2: 256 KB por núcleo.

  Cache L3: 6MB compartidos por todos los núcleos.

  4 núcleos y 4 hilos de ejecución.

- 4 GB en memoria principal (RAM) con frecuencia de 677 MHz.

- 500 GB en memoria secundaria (disco duro).

[7] Instituto Nacional de Salud Pública

6.2.2. Software

El equipo de cómputo tiene instalado el sistema operativo Windows 8 Pro y el sistema manejador SQL Server 2008 R2 Enterprise Edition[17].

En cuanto a los parámetros de *MAXDOP* y *Maximum server memory* se tienen configurados los valores que proveen un mayor desempeño al SMBD, p=4 y M=4GB.

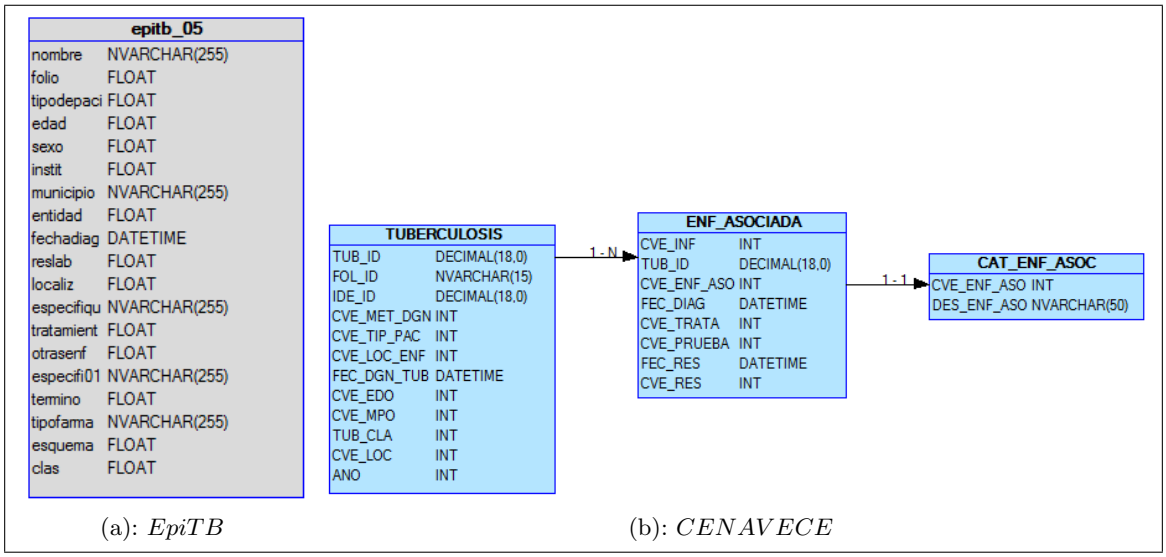
El entorno de ejecución de JAVA que se tiene instalado corresponde a la versión 1.7 y no presenta ningún tipo de incompatibilidad

Los resultados obtenidos fueron utilizados por el programa de análisis estadístico STATA en su versión 10 [12] en el cual se desarrolló el análisis de los datos.

6.2.3. Bases de datos

Las bases de datos reales a utilizar son versiones distintas de la misma base de datos. Las versiones difieren mucho pues fueron creadas con distintas herramientas de software.

La primera versión de esta base de datos fue desarrollada con un software especializado en epidemiología. Esta primera versión fue utilizada para la captación y seguimiento de pacientes entre los años 2000 y 2005. El esquema de esta primera versión de la base de datos consta de una sola tabla que contiene la información personal y médica de los pacientes. La Figura 6.7.a muestra un esquema resumido de esta tabla.



(a): *EpiTB* (b): *CENAVECE*

Figura 6.7: Esquemas de las bases de datos del caso de estudio.

Por otro lado, la versión de la base que almacena la información de los pacientes captados entre 2007 y 2012 tiene un mejor diseño ya que algunas tablas se encuentran normalizadas. El uso de tablas de catálogos hace que la búsqueda o manipulación de esta base de datos sea más fácil pues se tiene un mejor entendimiento de los datos

que almacena. Un diseño resumido de la base de datos en la que nos enfocaremos se muestra en la Figura 6.7 subfigura b.

6.2.4. Integración de bases de datos.

Para las bases de datos *EPI TB* y *CENAVECE* se requiere tener una tabla que cumpla con las siguientes características:

- Cada paciente debe aparecer una sola vez.
- Por cada paciente, se deben tener 12 columnas correspondientes a las enfermedades asociadas, el valor de cada columna es 1 si el paciente presenta la enfermedad y 0 en otro caso.

Obtener esta información en cada base de datos es diferente debido a que los esquemas difieren mucho. Por un lado en la base de datos *EPI TB* la información de las enfermedades asociadas se almacena solamente en una columna, es decir, a cada paciente solamente se le puede asociar una enfermedad. En la base de datos *CENAVECE* cada paciente puede tener muchas enfermedades asociadas.

Para ello en ambas bases de datos se deben transponer los datos de las enfermedades asociadas y en caso de presentarse esta enfermedad asignar el valor 1. Una función de agregación horizontal permite obtener los resultados deseados ya que se puede transponer la tabla de acuerdo a la enfermedad y junto con la aplicación de la función *COUNT(FOLIO)* se pueden asignar los valores requeridos ya que al ser el *FOLIO* una llave candidata, al agrupar los datos por el *FOLIO*, esta función devolverá siempre 1.

La siguiente consulta, utilizando funciones de agregación, resuelve el problema planteado:

```
SELECT FOLIO, COUNT(FOLIO BY otrasenf) AS c FROM EPITB_05 group by FOLIO
UNION ALL
SELECT T.TUB_ID,COUNT(FOLIO BY CVE_ENF_ASO) FROM TUBERCULOSIS T
LEFT OUTER JOIN ENF_ASOCIADA E ON T.TUB_ID=E.TUB_ID
```

Esta consulta transpone las enfermedades asociadas y dado que los valores de la combinación de *FOLIO* y enfermedad es única entonces la evaluación de la función de agregación obtiene 1 en caso de la presencia de la enfermedad y NULL en caso de no presentarla. Para la tabla *EPITB_05* ya que solamente se puede almacenar una enfermedad asociada entonces únicamente una de las columnas tendrá 1. Por otro lado para la base de datos *CENAVECE* un paciente podría presentar varias enfermedades por lo que varias columnas podrían tener un valor igual a 1. El resultado de ambas consultas se une para obtener la información deseada. Este ejemplo muestra una forma de integración de bases de datos donde los esquemas son totalmente diferentes.

La consulta en SQL estándar se muestra a continuación:

```
if OBJECT_ID('tempdb..##Fv','U') IS NOT NULL drop table ##Fv;
Select OTRASENF,FOLIO,COUNT(FOLIO) AS  C INTO ##Fv FROM EPITB_05
GROUP BY OTRASENF,FOLIO;
SELECT FOLIO,
SUM(CASE WHEN OTRASENF='1'THEN C ELSE NULL END)
  AS COUNT_FOLIO_BY_OTRASENF__1,
SUM(CASE WHEN OTRASENF='2'THEN C ELSE NULL END)
  AS COUNT_FOLIO_BY_OTRASENF__2,
SUM(CASE WHEN OTRASENF='3'THEN C ELSE NULL END)
  AS COUNT_FOLIO_BY_OTRASENF__3,
SUM(CASE WHEN OTRASENF='4'THEN C ELSE NULL END)
  AS COUNT_FOLIO_BY_OTRASENF__4,
SUM(CASE WHEN OTRASENF='5'THEN C ELSE NULL END)
  AS COUNT_FOLIO_BY_OTRASENF__5,
SUM(CASE WHEN OTRASENF='6'THEN C ELSE NULL END)
  AS COUNT_FOLIO_BY_OTRASENF__6,
SUM(CASE WHEN OTRASENF='7'THEN C ELSE NULL END)
  AS COUNT_FOLIO_BY_OTRASENF__7,
SUM(CASE WHEN OTRASENF='8'THEN C ELSE NULL END)
  AS COUNT_FOLIO_BY_OTRASENF__8,
SUM(CASE WHEN OTRASENF='9'THEN C ELSE NULL END)
  AS COUNT_FOLIO_BY_OTRASENF__9,
SUM(CASE WHEN OTRASENF='10'THEN C ELSE NULL END)
  AS COUNT_FOLIO_BY_OTRASENF__10,
SUM(CASE WHEN OTRASENF='11'THEN C ELSE NULL END)
  AS COUNT_FOLIO_BY_OTRASENF__11,
SUM(CASE WHEN OTRASENF='12'THEN C ELSE NULL END)
  AS COUNT_FOLIO_BY_OTRASENF__12,
FROM ##Fv  GROUP BY FOLIO;
--UNION ALL
if OBJECT_ID('tempdb..##Fv','U') IS NOT NULL drop table ##Fv;
Select CVE_ENF_ASO,TUB_ID,COUNT(TUB_ID) AS h_COUNT_TUB_ID
INTO ##Fv FROM  TUBERCULOSIS T LEFT OUTER JOIN ENF_ASOCIADA E
ON T.TUB_ID=E.TUB_ID GROUP BY CVE_ENF_ASO,TUB_ID;
SELECT TUB_ID,
SUM(CASE WHEN CVE_ENF_ASO='1' THEN h_COUNT_TUB_ID
         ELSE NULL END) AS COUNT_TUB_ID_BY_CVE_ENF_ASO_1,
SUM(CASE WHEN CVE_ENF_ASO='2' THEN h_COUNT_TUB_ID
         ELSE NULL END) AS COUNT_TUB_ID_BY_CVE_ENF_ASO_2,
SUM(CASE WHEN CVE_ENF_ASO='3' THEN h_COUNT_TUB_ID
         ELSE NULL END) AS COUNT_TUB_ID_BY_CVE_ENF_ASO_3,
SUM(CASE WHEN CVE_ENF_ASO='4' THEN h_COUNT_TUB_ID
         ELSE NULL END) AS COUNT_TUB_ID_BY_CVE_ENF_ASO_4,
SUM(CASE WHEN CVE_ENF_ASO='5' THEN h_COUNT_TUB_ID
         ELSE NULL END) AS COUNT_TUB_ID_BY_CVE_ENF_ASO_5,
SUM(CASE WHEN CVE_ENF_ASO='6' THEN h_COUNT_TUB_ID
```

```
         ELSE NULL END) AS COUNT_TUB_ID_BY_CVE_ENF_ASO_6,
SUM(CASE WHEN CVE_ENF_ASO='7' THEN h_COUNT_TUB_ID
         ELSE NULL END) AS COUNT_TUB_ID_BY_CVE_ENF_ASO_7,
SUM(CASE WHEN CVE_ENF_ASO='8' THEN h_COUNT_TUB_ID
         ELSE NULL END) AS COUNT_TUB_ID_BY_CVE_ENF_ASO_8,
SUM(CASE WHEN CVE_ENF_ASO='9' THEN h_COUNT_TUB_ID
         ELSE NULL END) AS COUNT_TUB_ID_BY_CVE_ENF_ASO_9,
SUM(CASE WHEN CVE_ENF_ASO='10' THEN h_COUNT_TUB_ID
         ELSE NULL END) AS COUNT_TUB_ID_BY_CVE_ENF_ASO_10,
SUM(CASE WHEN CVE_ENF_ASO='11' THEN h_COUNT_TUB_ID
         ELSE NULL END) AS COUNT_TUB_ID_BY_CVE_ENF_ASO_11,
SUM(CASE WHEN CVE_ENF_ASO='12' THEN h_COUNT_TUB_ID
         ELSE NULL END) AS COUNT_TUB_ID_BY_CVE_ENF_ASO_12
FROM ##Fv  GROUP BY TUB_ID;
```

Como se puede apreciar el código generado es complicado de leer. Una de las ventajas que presentan las agregaciones horizontales es la de expresar de mejor manera o por lo menos de manera resumida una consulta que es compleja. Por otro lado sin la utilización de agregaciones horizontales, el usuario debería de construir la consulta y pensar y aplicar las optimizaciones, como la de la tabla F_V. Para este experimento la utilización de agregaciones horizontales fue de gran ayuda pues permitió integrar la información de ambas bases de datos de manera rápida y eficiente.

6.2.5. Transformación de una base de datos.

En el siguiente experimento resuelve un problema de transformación de datos, en la Figura 6.8 se presenta un esquema de la base de datos llamada *TUBERCULOSIS*. En este esquema cada paciente tienen una tratamiento que dura alrededor de 1 año donde cada cierto periodo debe realizarse un seguimiento de la enfermedad y del tratamiento mediante tomas de muestras de laboratorio. Uno de los análisis requiere observar a los pacientes a lo largo de todo el tratamiento. La transformación de los datos que se requiere debe de cumplir con lo siguiente:

- Cada paciente debe aparecer una sola vez.
- Se deben tener todos los datos de las pruebas de laboratorio *BACILOS* (baciloscopias) a lo largo del tratamiento donde cada seguimiento debe considerar si el paciente tenía tos o no al momento de la revisión.
- Un paciente puede tener de 1 hasta 54 seguimientos. En caso de que un paciente no termine el tratamiento o falte algún seguimiento, el resultado de la prueba médica será nulo.

Figura 6.8: Esquema resumido de la tabla de seguimientos.

La información solicitada puede ser obtenida de la tabla *SEGUIMIENTOS* a través de las columnas *ID*, *N_VISITA*, *TOS* y *BACILOS*.

Uno de los desafíos en este trabajo es realizar la transposición de los resultados de laboratorio ya que algunos pacientes abandonaron el tratamiento por distintas causas. Otra cuestión es la consideración de un síntoma como lo es la *TOS* en cada revisión.

Nuevamente la utilización de funciones de agregación horizontal puede ayudar a transformar los datos de manera clara y eficaz. La siguiente consulta con una agregación horizontal sobre la tabla de seguimientos obtiene la vista transpuesta de los datos. La transposición se hace con las columnas *N_VISITA* y *TOS*, esto generará tantas columnas como el numero de combinaciones de los valores de ambas columnas. Si consideramos que la columna *TOS* solamente tiene dos valores entonces para un seguimiento determinado se tendrán dos columnas, una que indica el resultado del laboratorio en ese seguimiento cuando el síntoma de tos estaba presente y otra columna donde indica el resultado de la prueba para ese seguimiento cuando no había tos.

```
SELECT ID, SUM(BACILOS BY N_VISITA, TOS) FROM SEGUIMIENTOS GROUP BY ID;
```

Dado que *ID*, *N_VISITA* y *TOS* son clave candidata (*ID* y *N_VISITA* son llave primaria) de la tabla *SEGUIMIENTOS*, entonces los grupos definidos por los valores de estos tres atributos contendrán solamente una tupla por lo que el uso de la función de agregación *SUM()* es útil en este caso.

El código SQL generado es el siguiente:

```
if OBJECT_ID('tempdb..##Fv','U') IS NOT NULL drop table ##Fv;
Select ID,TOS,N_VISITA,SUM(BACILOS) AS h_SUM_BACILOS
INTO ##Fv FROM SEGUIMIENTO GROUP BY
ID,TOS,N_VISITA;

SELECT ID,
SUM(CASE WHEN N_VISITA='3' AND TOS='8' THEN
 h_SUM_BACILOS ELSE NULL END) AS
SUM_BACILOS_BY_N_VISITA_3_TOS_8,
SUM(CASE WHEN N_VISITA='1' AND TOS='2' THEN
 h_SUM_BACILOS ELSE NULL END) AS
```

```
SUM_BACILOS_BY_N_VISITA_1_TOS_2,
SUM(CASE WHEN N_VISITA='10' AND TOS='8' THEN
 h_SUM_BACILOS ELSE NULL END) AS
SUM_BACILOS_BY_N_VISITA_10_TOS_8,
SUM(CASE WHEN N_VISITA='1' AND TOS='0' THEN
 h_SUM_BACILOS ELSE NULL END) AS
SUM_BACILOS_BY_N_VISITA_1_TOS_0,
SUM(CASE WHEN N_VISITA='1' AND TOS='1' THEN
 h_SUM_BACILOS ELSE NULL END) AS
SUM_BACILOS_BY_N_VISITA_1_TOS_1,
SUM(CASE WHEN N_VISITA='5' AND TOS='8' THEN
 h_SUM_BACILOS ELSE NULL END) AS
SUM_BACILOS_BY_N_VISITA_5_TOS_8,
SUM(CASE WHEN N_VISITA='7' AND TOS='0' THEN
 h_SUM_BACILOS ELSE NULL END) AS
SUM_BACILOS_BY_N_VISITA_7_TOS_0,
SUM(CASE WHEN N_VISITA='2' AND TOS='0' THEN
 h_SUM_BACILOS ELSE NULL END) AS
SUM_BACILOS_BY_N_VISITA_2_TOS_0,
SUM(CASE WHEN N_VISITA='8' AND TOS='8' THEN
 h_SUM_BACILOS ELSE NULL END) AS
SUM_BACILOS_BY_N_VISITA_8_TOS_8,
SUM(CASE WHEN N_VISITA='7' AND TOS='8' THEN
 h_SUM_BACILOS ELSE NULL END) AS
SUM_BACILOS_BY_N_VISITA_7_TOS_8,
SUM(CASE WHEN N_VISITA='3' AND TOS='0' THEN
 h_SUM_BACILOS ELSE NULL END) AS
SUM_BACILOS_BY_N_VISITA_3_TOS_0,
SUM(CASE WHEN N_VISITA='1' AND TOS='8' THEN
 h_SUM_BACILOS ELSE NULL END) AS
SUM_BACILOS_BY_N_VISITA_1_TOS_8,
SUM(CASE WHEN N_VISITA='2' AND TOS='8' THEN
 h_SUM_BACILOS ELSE NULL END) AS
SUM_BACILOS_BY_N_VISITA_2_TOS_8,
SUM(CASE WHEN N_VISITA='6' AND TOS='8' THEN
 h_SUM_BACILOS ELSE NULL END) AS
SUM_BACILOS_BY_N_VISITA_6_TOS_8,
SUM(CASE WHEN N_VISITA='4' AND TOS='8' THEN
 h_SUM_BACILOS ELSE NULL END) AS
SUM_BACILOS_BY_N_VISITA_4_TOS_8,
SUM(CASE WHEN N_VISITA='9' AND TOS='8' THEN
 h_SUM_BACILOS ELSE NULL END) AS
SUM_BACILOS_BY_N_VISITA_9_TOS_8
FROM ##Fv  GROUP BY ID;
```

El resultado obtenido por la función de agregación horizontal permite analizar la relación entre el número de bacilos y la tos a lo largo del tratamiento, incluso permite observar a simple vista un panorama más general de los datos pues de tener una tabla de 1340 registros con 4 columnas se obtuvo una tabla de 176 filas con 17 columnas.

La ventaja de esta expresión frente a clausulas existentes como PIVOT son dos:

1. Se pudieron utilizar dos columnas de transposición, la clausula PIVOT sólo permite especificar los valores correspondientes a una columna.

2. No fue necesario conocer los valores distintos de cada combinación de los atributos de transposición. Si bien estos valores fueron obtenidos de la base de datos con la clausula DISTINCT, para el usuario fue transparente.

La experimentación con estas base de datos sirvió para darnos cuenta del poder de consulta de éstas expresiones ya que se pudieron mostrar ejemplos útiles que no necesariamente necesitaban calcular una función de agregación como se ha mostrado en la Sección 6.1, sino que además ayudan a integrar y transformar los datos para dar a los usuarios una mayor oportunidad de análisis.

Capítulo 7
Conclusiones y trabajo futuro.

En el presente trabajo uno de los objetivos principales ha sido el de mejorar las investigaciones que se han venido realizando sobre preparación de los datos mediante agregaciones horizontales. Nuestras propuestas son optimizaciones a los métodos existentes y se ha mostrado a través de la experimentación que mejoran el desempeño de la ejecución de consultas con este tipo de agregaciones. Las optimizaciones buscan reducir el número de lecturas a memoria secundaria ya que el acceso a los datos es lento en comparación con la memoria principal.

Las tres optimizaciones propuestas mostraron una reducción en el tiempo de ejecución. La reducción fue del 10 al 60 % dependiendo del tipo de optimización, como se puede observar en las Tablas 6.1,6.3 y 6.5. La optimización asíncrona es la mejor pues con ella se obtiene una reducción de hasta el 60 % además de que es posible reutilizarla para otras consultas. Dicha optimización es similar al uso del cache del buffer del SMBD con la diferencia de que además de reducir el tiempo de lectura se reduce también el tiempo de cómputo. Durante la experimentación se observó que al aumentar la cantidad de datos se tenía un mayor porcentaje de mejora en cada optimización. Esta propiedad es importante ya que en la actualidad la cantidad de datos generados o capturados aumenta de forma importante y los sistemas deben contar con las herramientas necesarias para manipular grandes cantidades de datos eficientemente.

La extensión SQL presentada para definir una función de agregación horizontal es clara e intuitiva lo que facilita al usuario su integración a las aplicaciones o programas existentes. Además la extensión ofrece al usuario transparencia en cuanto a la forma en la que se evalúa por lo que su implementación eficiente es una pieza clave. Modificar el SMBD para la evaluación y optimización de estas funciones es un trabajo laborioso y/o costoso debido a que el código fuente no está disponible y que al ser un sistema tan complejo puede ser necesario invertir demasiado tiempo en su integración. La herramienta desarrollada tiene como primer propósito ofrecer a la comunidad una herramienta optimizada para la transposición y agregación de datos dentro de un SMBD. Como segundo propósito permite entender los problemas asociados con la evaluación y optimización de estas funciones de agregación.

La herramienta permite integrar la funcionalidad de transposición de datos a partir de código SQL estándar. Se han mostrado problemas reales donde las funciones de agregación horizontales han sido útiles en la integración y transformación de bases de datos. En general se ha mostrado que la implementación de estas funciones provee una herramienta de gran utilidad que puede ser utilizada en sistemas donde se requiere eficiencia así como en sistemas donde se busca facilidad y flexibilidad en la manipulación de los datos. Finalmente podemos concluir que la herramienta descrita a lo largo de este trabajo incorpora mucho del conocimiento sobre las funciones de

agregación horizontal y que es una herramienta útil para la preparación y análisis de datos además de ser implementada con varias optimizaciones para proveer al usuario transparencia acerca de la manipulación interna de los datos.

El trabajo futuro sobre agregaciones horizontales está enfocado a aspectos que permitan la integración de estas funciones dentro de un SMBD. El software y las optimizaciones expuestas en esta investigación tratan varios aspectos técnicos de la implementación que deben ser considerados para lograr esta integración, pero existen otros aspectos que también deben ser tomados en cuenta, por ejemplo, el bloqueo de tablas y niveles de aislamiento, optimizaciónes en tiempo de compilación de las consultas y problemas referentes al estandar o dialecto particular SQL del SMBD.

En investigaciones recientes se han desarrollado diferentes tipos de SMBD como VERTICA [1] que es un SMBD que tienen un manejo distinto del almacenamiento y acceso a los datos y es orientado a análisis de grandes cantidades de datos. Experimentar con sistemas que realizan la consulta a través de SQL es un campo abierto de investigación para las agregaciones horizontales pues cada sistema basado en SQL podría en principio [1] utilizar este nuevo tipo de funciones para el análisis y preparación interna de los datos.

[1] Con las implicaciones técnicas inherentes de cada sistema.

Apéndices

Apéndice A
Aplicación

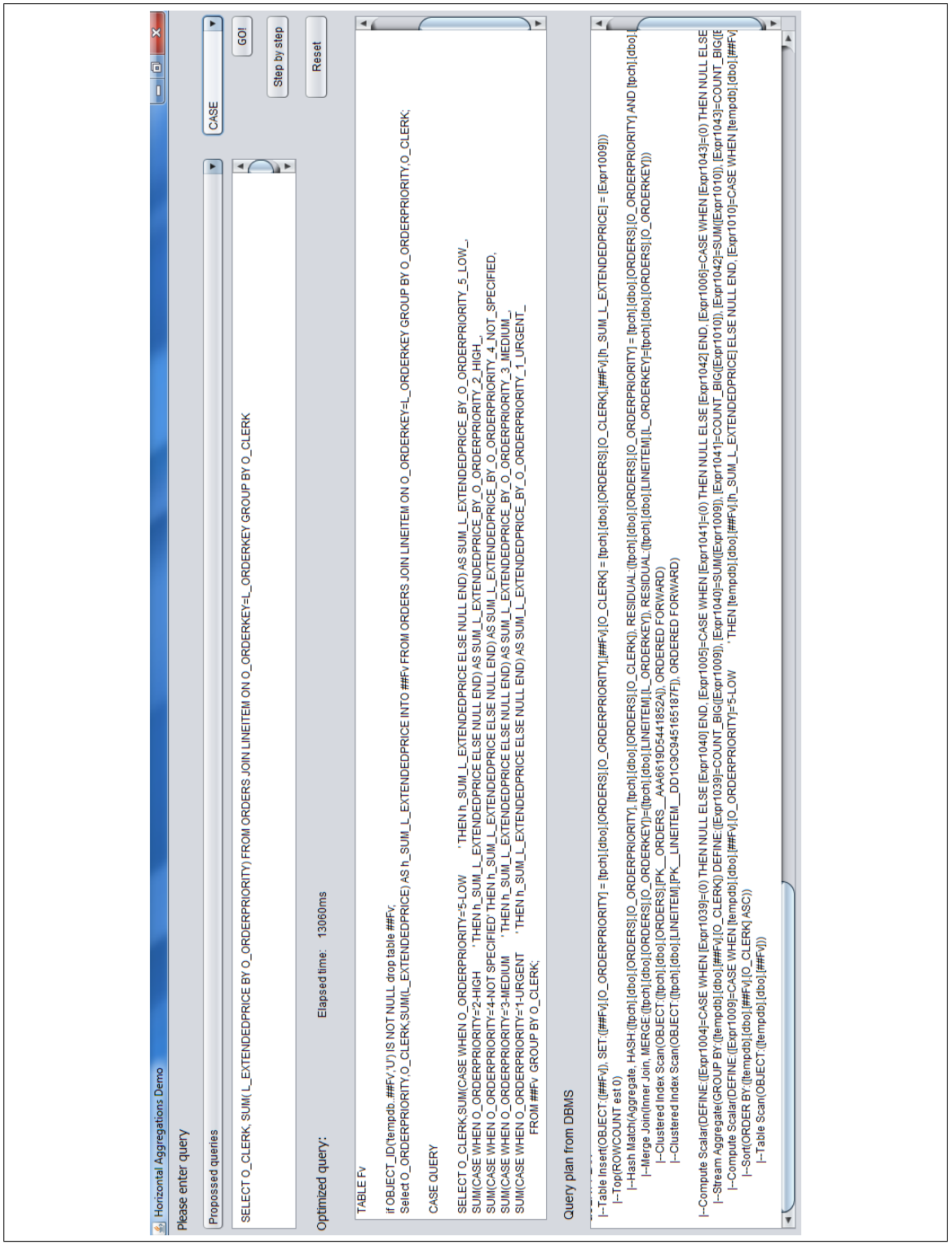
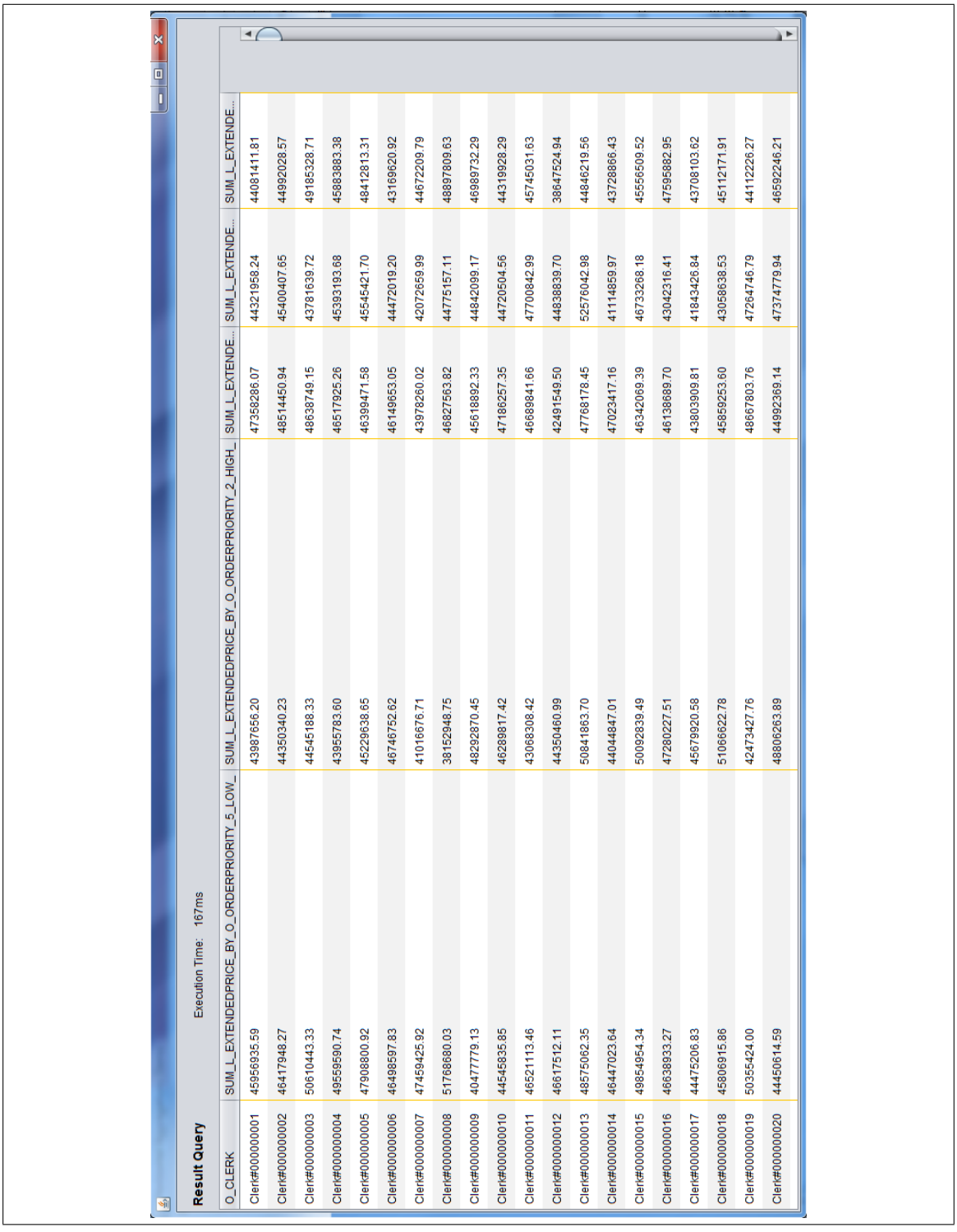


Figura A.1: Interfaz de escritorio de la herramienta.



Result Query Execution Time: 167ms

O_CLERK	SUM_L_EXTENDEDPRICE_BY_O_ORDERPRIORITY_5_LOW_	SUM_L_EXTENDEDPRICE_BY_O_ORDERPRIORITY_2_HIGH_	SUM_L_EXTENDE...	SUM_L_EXTENDE...	SUM_L_EXTENDE...
Clerk#000000001	45956935.59	43987656.20	47358286.07	44321958.24	44081411.81
Clerk#000000002	46417948.27	44350340.23	48514450.94	45400407.65	44992028.57
Clerk#000000003	50610443.33	44545188.33	48638749.15	43781639.72	49185328.71
Clerk#000000004	49559590.74	43955783.60	46517925.26	45393193.68	45883883.38
Clerk#000000005	47908800.92	45229638.65	46399471.58	45545421.70	48412813.31
Clerk#000000006	46498597.83	46746752.62	46149653.05	44472019.20	43169620.92
Clerk#000000007	47459425.92	41016676.71	43978260.02	42072659.99	44672209.79
Clerk#000000008	51768680.03	38152948.75	46827563.82	44775157.11	48897809.63
Clerk#000000009	40477779.13	48292870.45	45618892.33	44842099.17	46989732.29
Clerk#000000010	44545835.85	46289817.42	47186257.35	44720504.56	44319928.29
Clerk#000000011	46521113.46	43068308.42	46689841.66	47700842.99	45745031.63
Clerk#000000012	46617512.11	44350460.99	42491549.50	44838839.70	38647524.94
Clerk#000000013	48575062.35	50841863.70	47768178.45	52576042.98	44846219.56
Clerk#000000014	46447023.64	44044847.01	47023417.16	41114859.97	43728866.43
Clerk#000000015	49854954.34	50092839.49	46342069.39	46733268.18	45556509.52
Clerk#000000016	46638933.27	47280227.51	46138689.70	43042316.41	47595882.95
Clerk#000000017	44475206.83	45679920.58	43803909.81	41843426.84	43708103.62
Clerk#000000018	45806915.86	51066622.78	45859253.60	43058638.53	45112171.91
Clerk#000000019	50355424.00	42473427.76	48667803.76	47264746.79	44112226.27
Clerk#000000020	44450614.59	48806263.89	44992369.14	47374779.94	46592246.21

Figura A.2: Resultado de ejecutar la consulta con agregaciones horizontales.

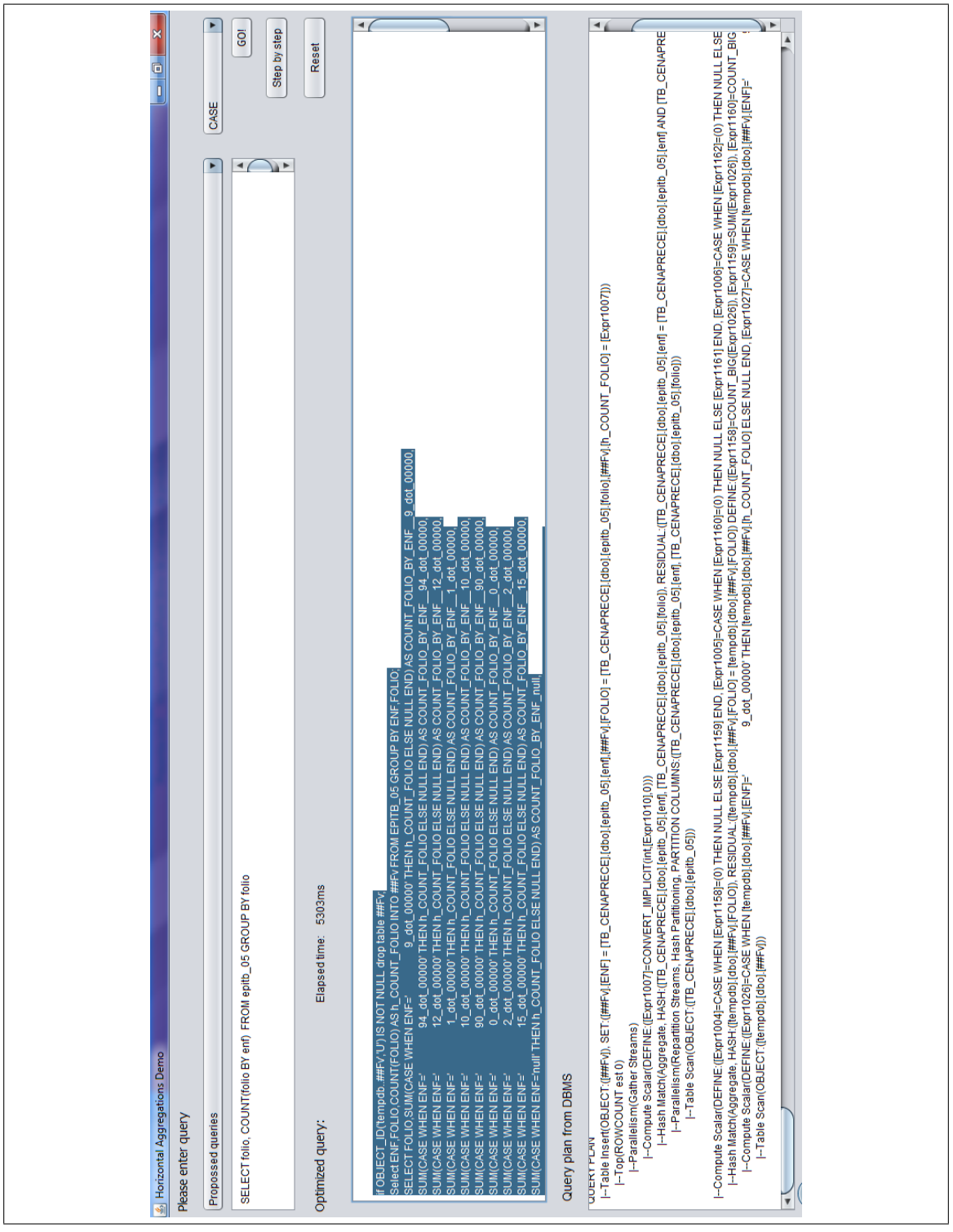


Figura A.3: Utilizando la herramienta en los experimentos con bases de datos reales.

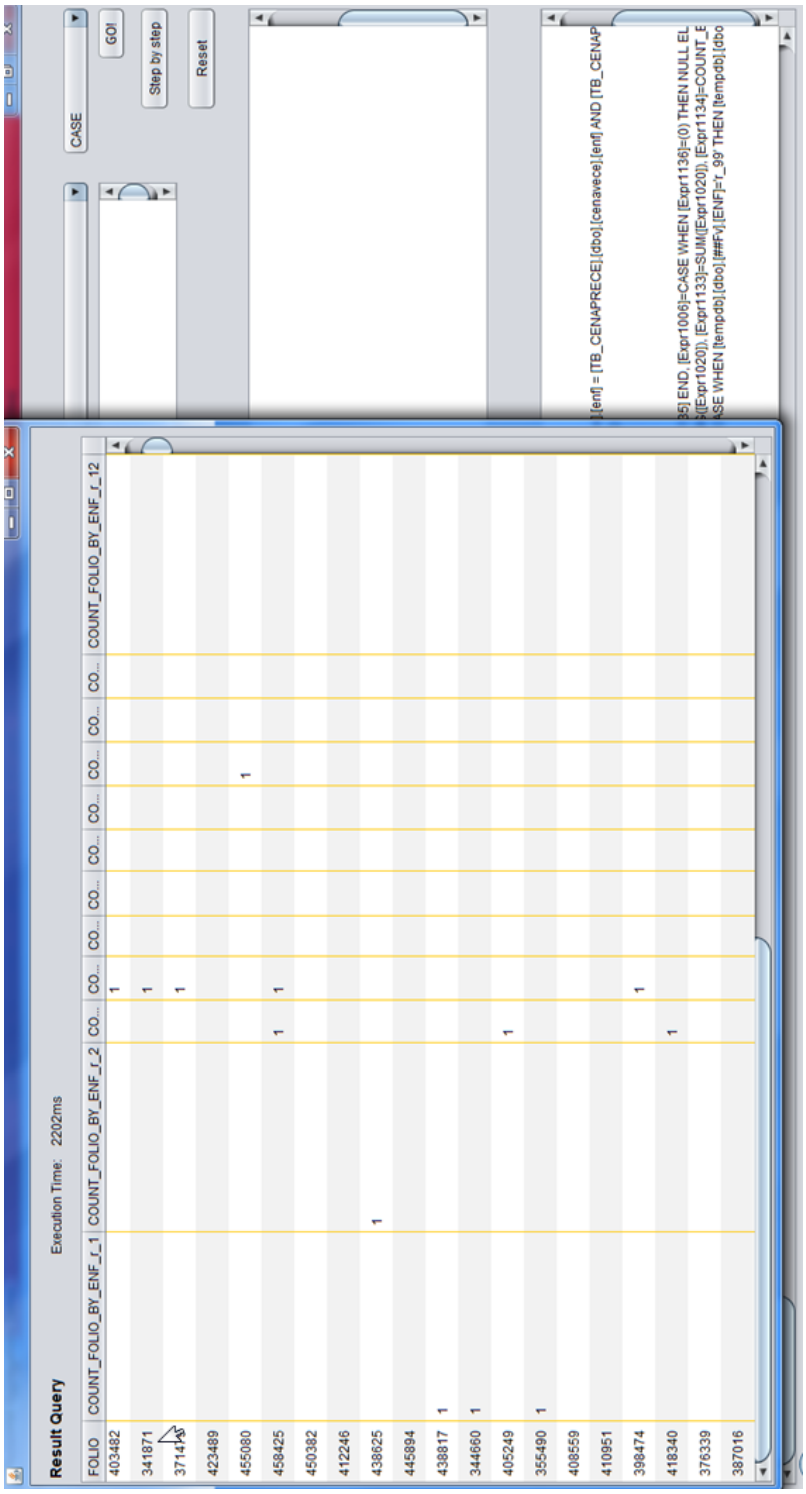


Figura A.4: Resultado de un experimento con una base de datos real.

Bibliografía

[1] Chuck Bear, Andrew Lamb, and Nga Tran. The vertica database: Sql rdbms for managing big data. In *Proceedings of the 2012 Workshop on Management of Big Data Systems*, MBDS '12, pages 37–38, New York, NY, USA, 2012. ACM.

[2] Luis Pablo Cruz-Hervert, Lourdes García-García, Leticia Ferreyra-Reyes, Miriam Bobadilla-del Valle, Bulmaro Cano-Arellano, Sergio Canizales-Quintero, Elizabeth Ferreira-Guerrero, Renata Báez-Saldaña, Norma Téllez-Vázquez, Ariadna Nava-Mercado, et al. Tuberculosis in ageing: high rates, complex diagnosis and poor clinical outcomes. *Age and ageing*, 41(4):488–495, 2012.

[3] Conor Cunningham, César A. Galindo-Legaria, and Goetz Graefe. Pivot and unpivot: optimization and execution strategies in an rdbms. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 998–1009. VLDB Endowment, 2004.

[4] Tamraparni Dasu and Theodore Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley & Sons, Inc., New York, NY, USA, 1 edition, 2003.

[5] Pinal Dave. Sql authority blog: Clean cache and clean buffer. http://blog.sqlauthority.com/2007/03/23/sql-server-stored-procedure-clean-cache-and-clean-buffer/.

[6] Ramez Elmasri and Shamkant Navathe. *Fundamentals of Database Systems*. Addison-Wesley Publishing Company, USA, 6th edition, 2010.

[7] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. Advances in knowledge discovery and data mining. chapter From Data Mining to Knowledge Discovery: An Overview, pages 1–34. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.

[8] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Min. Knowl. Discov.*, 1(1):29–53, January 1997.

[9] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[10] María Eugenia Jiménez-Corona, Luis Pablo Cruz-Hervert, Lourdes García-García, Leticia Ferreyra-Reyes, Guadalupe Delgado-Sánchez, Miriam Bobadilla-del Valle, Sergio Canizales-Quintero, Elizabeth Ferreira-Guerrero, Renata Báez-Saldaña, Norma Téllez-Vázquez, et al. Association of diabetes and tuberculosis: impact on treatment and post-treatment outcomes. *Thorax*, 68(3):214–220, 2013.

[11] Ling Liu and M. Tamer Zsu. *Encyclopedia of Database Systems*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[12] StataCorp LP. Sata. http://www.stata.com/stata10/.

[13] Microsoft. Buffer managment: Sql server 2008 r2 x64. http://msdn.microsoft.com/en-us/library/aa337525(v=sql.105).aspx.

[14] Microsoft. Dbcc: Sql server 2008 r2 x64. http://msdn.microsoft.com/en-us/library/ms188796(v=sql.105).aspx.

[15] Microsoft. Especificaciones de capacidad máxima: Sql server 2008 r2 x64. http://technet.microsoft.com/en-us/library/ms143432(v=SQL.105).aspx.

[16] Microsoft. Io statistics: Sql server 2008 r2 x64. http://msdn.microsoft.com/en-us/library/ms184361(v=sql.105).aspx.

[17] Microsoft. Sql server 2008 r2 x64. http://msdn.microsoft.com/en-us/library/ms187875(v=sql.105).aspx.

[18] Carlos Ordonez. Horizontal aggregations for building tabular data sets. In *Proceedings of the 9th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, DMKD '04, pages 35–42, New York, NY, USA, 2004. ACM.

[19] Carlos Ordonez. Vertical and horizontal percentage aggregations. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, pages 866–871, New York, NY, USA, 2004. ACM.

[20] Carlos Ordonez. Integrating k-means clustering with a relational dbms using sql. *IEEE Trans. on Knowl. and Data Eng.*, 18(2):188–201, February 2006.

[21] Carlos Ordonez and Zhibo Chen. Horizontal aggregations in sql to prepare data sets for data mining analysis. *IEEE Trans. on Knowl. and Data Eng.*, 24(4):678–691, April 2012.

[22] Carlos Ordonez, Javier García-García, and Zhibo Chen. Dynamic optimization of generalized sql queries with horizontal aggregations. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 637–640, New York, NY, USA, 2012. ACM.

[23] P. S. Patil, S. Rao, and S. B. Patil. Data integration problem of structural and semantic heterogeneity: Data warehousing framework models for the optimization of the etl processes. In *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*, ICWET '11, pages 500–504, New York, NY, USA, 2011. ACM.

[24] Dorian Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[25] Sunita Sarawagi, Shiby Thomas, and Rakesh Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. *Data Min. Knowl. Discov.*, 4(2-3):89–125, July 2000.

[26] A. Silberschatz, H. Korth, and S. Sudarshan. *Database System Concepts*. Connect, learn, succeed. McGraw-Hill Education, 2010.

[27] Benchmark TPC-H. Transaction processing performance council. http://www.tpc.org/tpch/default.asp, 2005.

[28] Haixun Wang, Carlo Zaniolo, and Chang Richard Luo. Atlas: a small but complete sql extension for data mining and data streams. In *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '03, pages 1113–1116. VLDB Endowment, 2003.