UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Construcción y aplicaciones de las Bases de Datos Columnares

TESINA

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A N

Oscar Alfredo Constantino Mota

Jesús Emiliano García Montes

DIRECTOR DE TESIS

Ing. Héctor Bautista Vázquez

Ciudad Universitaria, Cd. Mx., 2016

CONTENIDO

1. PRESENTACIÓN

1.1. INTRODUCCIÓN

El mundo que vivimos hoy en día está gobernado por aplicaciones computacionales y por los datos que se generan en cada momento de nuestras vidas, incluso existen personas que afirman que estamos viviendo una tercera revolución industrial, donde el combustible de esta era son los datos. La información contenida en este universo de datos tiene un papel muy importante en la sociedad, para empresas, gobiernos, y personas. Para poder manejar todo el volumen de datos que se genera, se han diseñado y utilizado diferentes tipos de herramientas, desde simples archiveros en papel hasta grandes sistemas donde se administra de manera más eficiente la información.

La tecnología de bases de datos relacionales es la que se utiliza actualmente en los sistemas de manejo de bases de datos más populares, sin embargo, como revisaremos en este trabajo existen otras tecnologías de bases de datos que pueden ser más eficientes en el manejo de grandes volúmenes de datos. Consideremos también que todos esos datos deben ser procesados para poder obtener información relevante. Este procesamiento puede resultar muy costoso tanto en tiempo como en los recursos que se requieren, por lo que obtener dicha información resulta ser un reto para los sistemas manejadores de base de datos basados en modelos relacionales.

Para poder resolver correctamente estos problemas se decidió desarrollar tecnologías alternativas de manejo de bases de datos que puedan responder de una manera eficiente a las consultas de grandes cantidades de datos. Entre estas tecnologías alternativas, se ubican las Bases de Datos Columnares (BDC), las cuales manejan los datos de forma distinta a los manejadores de bases de datos relacionales, con el objetivo de mejorar el desempeño en las consultas de grandes cantidades de registros. Sin embargo, como veremos durante el desarrollo de esta tesis, las Bases de Datos Columnares, son eficientes en ciertos ambientes operacionales, mientras que para otros pudieran no ser la mejor opción.

En este trabajo se presentará el sustento tecnológico y cómo se pueden utilizar de una manera eficiente las Bases de Datos Columnares. Se inicia con un análisis de los modelos de datos anteriores para establecer el sustento teórico de este trabajo que consiste en una descripción general de las Bases de Datos Columnares y sus orígenes.

En la segunda parte se analizará el detalle tecnológico del manejo de datos de las BDC, su descripción, el almacenamiento y acceso a los datos. Así mismo, nos enfocamos en analizar a detalle el proceso de transacciones, es decir, se describe el funcionamiento de las operaciones de inserción, actualización y eliminación de datos, para presentar una comparación de rendimiento en relación a las bases de datos por renglón ya que el almacenamiento por renglón es la forma de almacenar datos más utilizada actualmente. También realizamos la descripción de las funcionalidades más importantes que soportan en general las bases de datos columnares que son la compresión de los datos, la paralelización de procesos, es decir, dividir un proceso y atender cada proceso simultáneamente y el manejo de índices. Dicha comparación nos permitirá conocer el funcionamiento de las Bases de Datos Columnares.

Por último se identificarán los proyectos en los cuales es conveniente utilizar una base de datos columnar, señalando las ventajas que proporcionan, qué proveedores existen al día de hoy, y sobre todo, conocer las características necesarias en los sistema para utilizar de manera óptima este tipo de bases de datos.

1.2. OBJETIVO

Comparar las bases de datos columnares con las bases de datos de almacenamiento por renglón, a fin de identificar su conveniencia en diferentes escenarios. Así mismo se muestran las características, funcionalidades y aplicaciones de una base de datos columnar.

1.3. HIPÓTESIS

Al realizar la comparación de desempeño entre las bases de datos columnares con las bases de datos de almacenamiento por renglón, esperamos que en ambientes transaccionales (OLTP) el mejor desempeño lo obtenga el almacenamiento por renglón, mientras que en ambientes de data warehouse y soporte a toma de decisiones (OLAP, DSS) el almacenamiento por columnas superará al almacenamiento por renglón.

1.4. JUSTIFICACIÓN

Decidimos elaborar la presente tesis sobre las bases de datos columnares ya que a pesar de que tienen más de veinte años en la industria, la mayoría de los administradores de bases de datos no saben de su existencia, o bien no cuentan con un amplio conocimiento sobre ellas. Además, esta tecnología puede ser de gran ayuda para ciertos tipos de proyectos, pero si no se saben implementar de manera adecuada no se podrán aprovechar todas sus capacidades.

Por otro lado, se ha identificado que no existen muchas investigaciones o documentos generados en México que hablen sobre estos temas. Partiendo de éstas dos razones, este documento podrá ser empleado como base para que más profesionistas conozcan sobre este tipo de bases de datos.

1.5. METODOLOGÍA

Para realizar este trabajo utilizamos el método de análisis-síntesis.

Es un método que consiste en la separación de las partes de un todo para estudiarlas en forma individual (Análisis), y la reunión racional de elementos dispersos para estudiarlos en su totalidad (Síntesis).

Separamos el tema en los antecedentes, en cómo se definen las bases de datos columnares, cuales son las características tecnológicas y por último sus aplicaciones en diferentes condiciones.

Análisis

Del griego analizas: descomposición, fragmentación de un cuerpo en sus principios constitutivos. Método que va de lo compuesto a lo simple. Proceso cognoscitivo por medio del cual una realidad es descompuesta en partes para su mejor comprensión. Separación de un todo en sus partes constitutivas con el propósito de estudiar éstas por separado, así como las relaciones que las unen (François, J.L., s.f, pág. 3).

Síntesis

Del griego síntesis: método que procede de lo simple a lo compuesto, de las partes al todo, de la causa a los efectos, del principio a las consecuencias. Composición de un todo por la reunión de sus partes. Reunión de las partes o elementos para analizar, dentro de un todo, su naturaleza y comportamiento con el propósito de identificar las características del fenómeno observado (François, J.L., s.f, pág. 3).

2. Bases de Datos Columnares

En el presente capítulo se hablará sobre las BDC, sus características, antecedentes y su relación con las bases de datos de almacenamiento por renglón.

2.1. Antecedentes de las bases de datos

Durante toda la historia de la humanidad se han utilizado herramientas para poder archivar datos. Con el avance de la tecnología y la creación de las computadoras se empezaron a guardar grandes cantidades de archivos en discos, cintas disquetes, etc. Sin embargo, al principio no se tenía un control adecuado de toda esa información.

Durante la década de los 60 se le empezó a dar más importancia a la información, combinado con algunos factores como son la mayor potencia de los sistemas operativos y el aumento en la capacidad de la memoria principal, se generaron los primeros sistemas de administración de bases de datos (Guillenson, 2006, p. 13).

2.1.1. Bases de Datos Jerárquicas

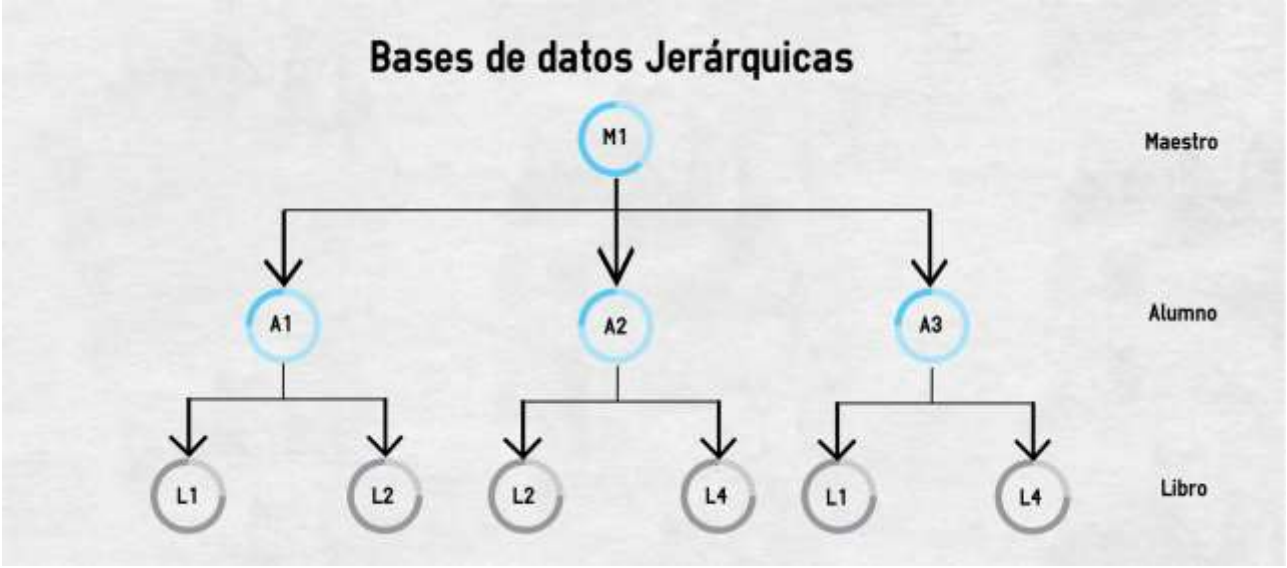
Las bases de datos jerárquicas fue el primer modelo empleado en un Sistema de administración de bases de datos (DBMS). “Se conoce como enfoque navegable debido a la forma en que los programas tienen que ‘navegar’ a través de jerarquías y redes de datos para encontrar los datos que se necesitan” (Guillenson, 2006, p. 78).

Uno de los sistemas que lo implementó fue el IMS (Information Management System) de IBM que se utilizará como referencia en los siguientes puntos.

2.1.1.1. Representación

Se representan como un árbol, cumpliendo con varias de sus características:

- Debe tener un nodo raíz, el cual no puede tener padre.
- Un nodo no tiene límite de hijos, pero cada nodo sólo puede tener un padre.
- Los nodos que no tienen hijos se conocen como hojas.



Img.1 Ejemplo de modelo jerárquico.

Estos árboles consisten de una colección de **registros** que se conectan mediante relaciones entre los datos, también denominadas **ligas**.

Cada registro contiene una colección campos o atributos, estos campos definen los datos que se van a guardar, ya que solo puede guardar un solo tipo de dato y un solo valor a la vez.

Una liga es la relación entre los registros, estas ligas deben de asociar a dos únicos registros y permiten asociar datos de diferentes registros, funciona mediante el uso de punteros para poder unir los nodos del árbol.

Las ligas o relaciones son de gran utilidad de las bases de datos, porque permite que se pueda obtener más información de un solo registro.

Algunas características importantes de los diagramas de árbol es la representación de las relaciones uno a uno, uno a muchos y muchos a muchos.

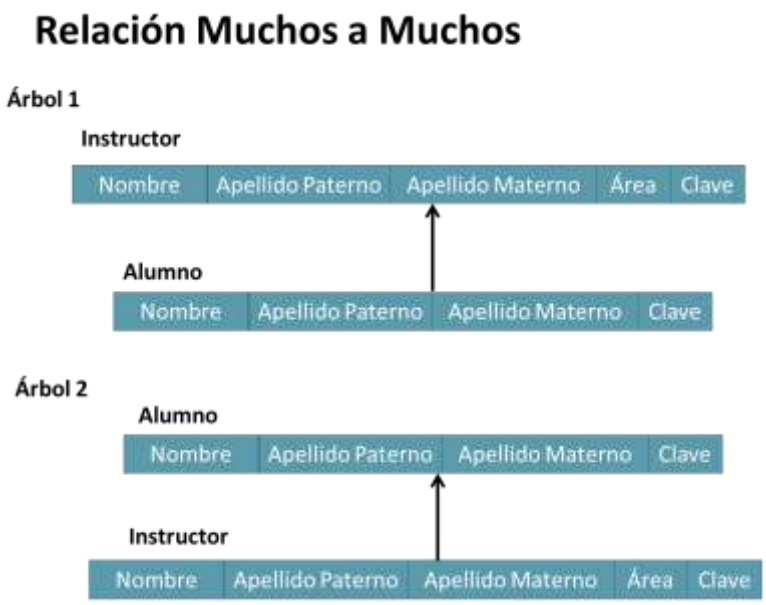
Para las relaciones uno a uno se representa por una flecha por ambas direcciones de los registros (Abraham Silberschatz, S. Sudarshan, Henry F. Korth, 1986, pag 4).

Img.2 Ejemplo de relación Uno a Uno.

Para las relaciones uno a muchos se utiliza una flecha en la que se dirige del nodo hijo al nodo padre, pero si otro registro utiliza el mismo nodo hijo, el nodo hijo se tendrá que duplicar (Abraham Silberschatz, S. Sudarshan, Henry F. Korth, 1986, pag 4).

Img.3 Ejemplo de relación Uno a Muchos.

Para el caso de las relaciones muchos a muchos, se tiene que generar un árbol extra en el que se tienen los mismos registros, el primer árbol tendrá como nodo raíz el registro padre y el segundo árbol tendrá como nodo raíz el registro hijo (Abraham Silberschatz, S. Sudarshan, Henry F. Korth, 1986, pag 4).

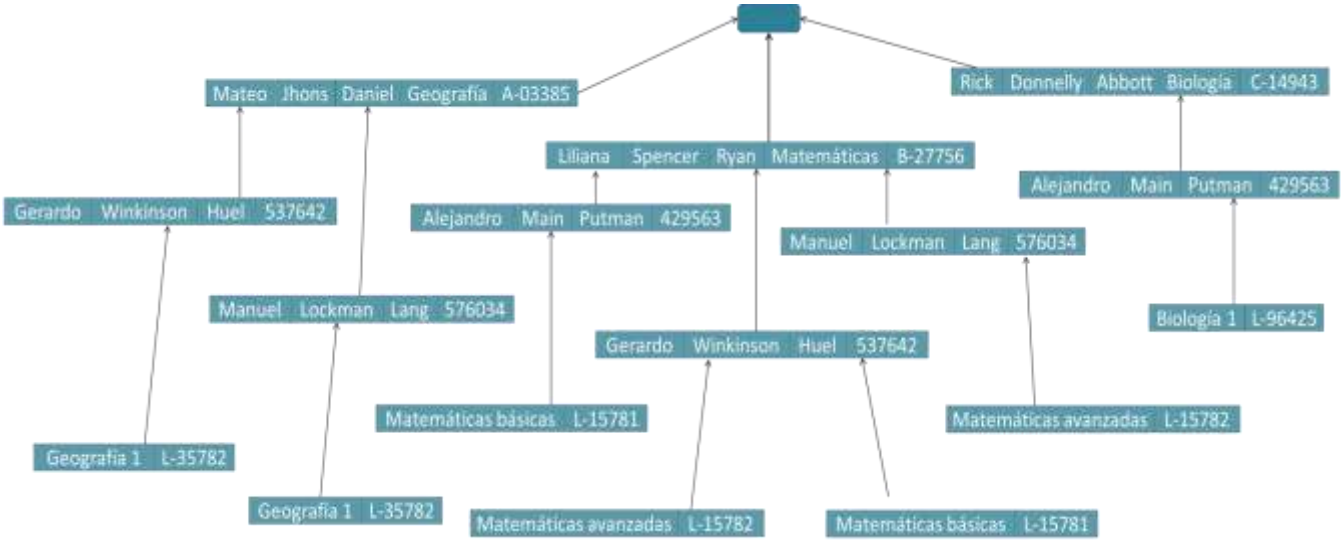


Img.4 Ejemplo de relación Muchos a Muchos.

Por ejemplo, la siguiente figura muestra las estructura del modelo de la **Img 1.**:

Img.5 Ejemplos de estructuras.

Un ejemplo en la base de datos de este modelo sería como el mostrado en **Img 6**, en la que podemos observar cómo se comportan los registros en las bases de datos jerárquicas.



Img.6 Ejemplos de una base de datos.

Podemos observar puntos interesantes, una de éstas es la duplicidad de registros, por ejemplo, para indicar que un libro está relacionado con varios alumnos cada alumno tendrá un registro repetido del mismo libro, debido a la restricción sobre los registros con un solo padre. Otro punto que también está relacionado con la duplicidad de registros, es la alta posibilidad de inconsistencias en la información. También se puede notar que se tiene que un nodo raíz *dummy* que nos ayuda a tener varios registros relacionados.

2.1.1.2. Consulta de datos

Para representar cómo se consulta una base de datos usaremos como lenguaje de consulta DL/I (Data Language/Interface), que es el lenguaje que utiliza el manejador IMS.

Cuando se quiere almacenar o recuperar datos de la base de datos, IMS utiliza funciones de DL/I. El cual se puede procesar de dos diferentes maneras, secuencial o aleatoria (tutorialspoint, 2016).

- **Procesamiento secuencial**

  Durante este proceso se recorre la estructura desde el nodo raíz, después se continúa con su primer hijo izquierdo, si ese segmento también tiene hijos, entonces se continúa con el primer hijo izquierdo y así sucesivamente, cuando se está en el nivel más bajo, se recupera todos los elementos repetidos del registro y comienza a subir por los registros a la derecha. Si el registro a la derecha también tiene hijos, entonces se empieza a recorrer de igual manera a su primer hijo a la izquierda.

  Para poder localizar estos segmentos o nodos, utiliza su posición que utiliza en la base de datos.

- **Procesamiento aleatorio**

  Este proceso también conocido como procesamiento directo, para esto se requiere que cada nodo contenga su ruta o llave hasta el nodo raíz, es decir tiene la referencia de todos los nodos necesarios para llegar hasta el nodo raíz.

  Al realizar la búsqueda se debe de indicar el camino o los nodos que se debe de recorrer para llegar al nodo deseado.

  En el procesamiento real el DL/I de IMS utiliza una combinación de ambos métodos.

2.1.1.3. ALMACENAMIENTO

La longitud máxima de cualquier segmento es de 32,000 bytes, con un apartado para el índice de 240 bytes y si se adiciona con prefijos para los datos, puede llegar a 32,258 bytes. Cada segmento solo puede tener 15 niveles de jerarquía.

Para almacenar datos IMS proporciona una longitud fija para los segmentos raíz, pero se puede indicar la longitud de la raíz desde un principio.

Algunas características que se pueden observar en este modelo es que no se tienen un dueño de los segmentos, en este tipo de estructuras no se insertan los datos, solo se actualiza la estructura, otra característica es que al almacenar segmentos que están

relacionados, se realizan de manera jerárquica, es decir que se encuentran continuas, lo que nos permite una recuperación de datos mucho más rápida (IBM, s.f.).

2.1.1.4. Observaciones

Algunas observaciones de este modelo son las siguientes (Rodríguez, Campos, 2013):

- Una vez creada la jerarquía, ya no se puede modificar.
- En relaciones M:N existe redundancia.
- En búsquedas particulares se tiene que recorrer todo el árbol.
- Si se elimina una entidad que tiene nodos hijos, estos también se eliminan.

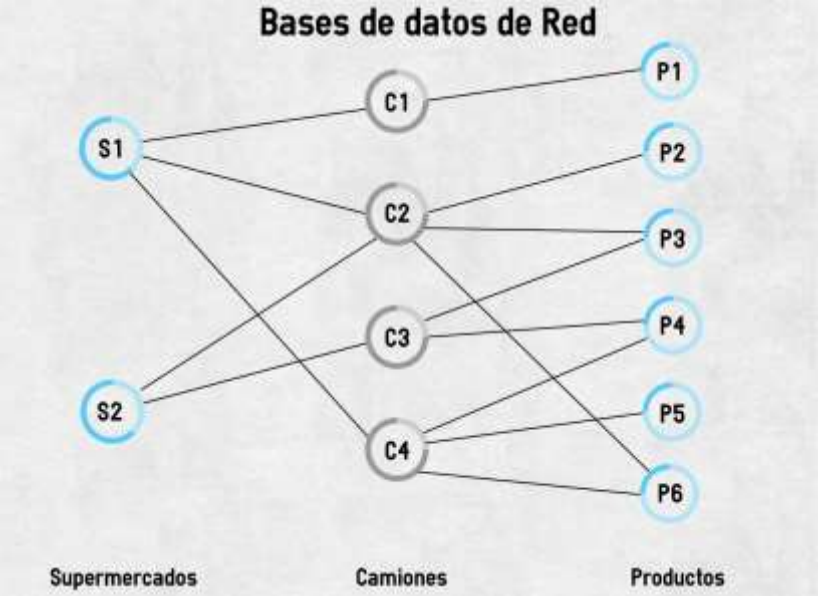
2.1.2. Bases de Datos de Red

Este modelo de bases de datos es similar a las bases de datos jerárquicas, con la diferencia de que en lugar de representarse como un árbol, se hace como un grafo. Esta diferencia le permite que un nodo pueda tener varios padres, con lo que se pueden manejar de una manera más adecuada las relaciones entre las entidades. Sin embargo, sigue teniendo algunos problemas, por ejemplo, según Gálvez (s.f, p. 4, http://www.lcc.uma.es/~galvez/ftp/bdst/Tema2.pdf) un registro del elemento A sólo puede estar ligado a un elemento del registro B.

2.1.2.1. Representación

Estos modelos consisten de la colección de registros conectados unos a otros mediante ligas.

Los registros son muy parecidos al modelo Jerárquico y al modelo Entidad Relación, ya que consisten en una colección de campos, los cuales contienen un valor único. Una liga es la asociación de solo dos registros.



Img 7. Modelo de datos de red

Estos tipos de diagrama de estructura nos ayudan a poder modelar de manera más sencilla un caso, luego poder implementarlo en una base de datos.

Para las relaciones Uno a Uno, la representación es muy parecida a la del modelo Jerárquico, se representa con una flecha apuntando a los dos registros con una flecha apuntando en cada una de los registros (Abraham Silberschatz, S. Sudarshan, Henry F. Korth, 1986, pág 3)

Img 8. Representación de una relación Uno a Uno

Para las relaciones Uno a Muchos se representa mediante una flecha que va dirigido del registro hijo al registro padre, en este ejemplo un alumno está relacionado con varios libros, por la dirección de la flecha está dirigida al alumno. (Abraham Silberschatz, S. Sudarshan, Henry F. Korth, 1986, pág 3).

Img 9. Representación de una relación Uno a Uno

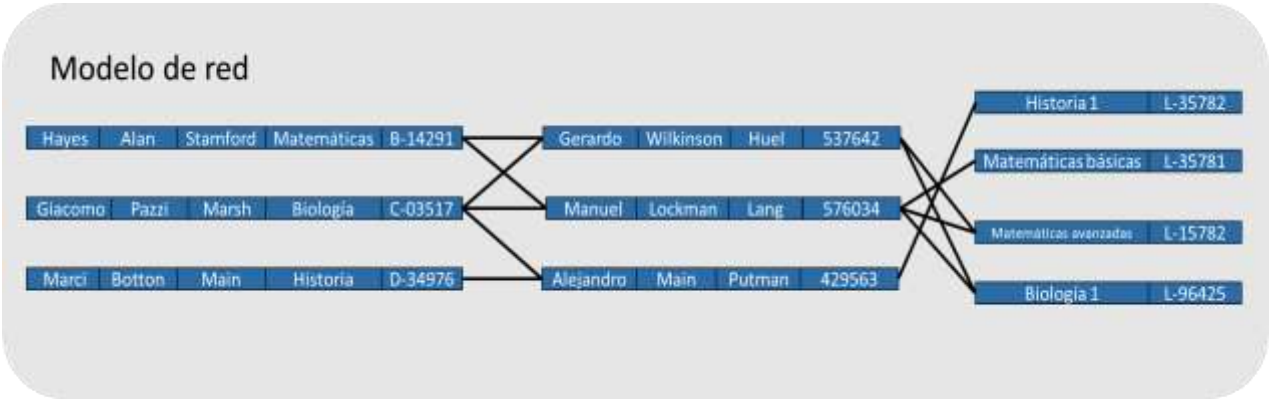
Para las relaciones muchos a muchos se representa mediante una línea que une a los dos registros, pero sin ninguna flecha en los registros (Abraham Silberschatz, S. Sudarshan, Henry F. Korth, 1986, pág 3).

Img 10. Representación de una relación Muchos a Muchos

Si se requiere tener atributos entre las dos relaciones, como en las relaciones débiles en el modelo relacional, se requiere realizar un tercer registro el cual tendrá los atributos que se requieren y las referencias de los registros a los que pertenece.

Img 11. Representación de una relación Muchos a Muchos

Como un ejemplo de la implementación de la base de datos, se muestra la estructura de la **Img 5** en la que se observa el comportamiento de los registros en las bases de datos de red.



Img 12. Ejemplo en la base de datos de un modelo de Red

Se puede observar que a diferencia del modelo jerárquico, este no requiere de duplicar registros, ahora los registros pueden tener la referencia de más de un solo padre, pero en una relación muchos a muchos si se quiere tener atributos de esa relación es un poco diferente al modelo relacional.

2.1.2.2. CONSULTA DE DATOS

Para representar el modo por el cual una base de datos de red recupera los datos, se utiliza como ejemplo el manejador IDMS/R (*Integrated Database Management System/Relational*) que utiliza el modelo de base de datos de red (Mainframe Tutorials, 2009).

IDMS/R utiliza dos métodos para recuperar los datos, uno de ellas es la de acceso aleatorio, el cual también es llamado recuperación CALC. El segundo método es el que accede al registro mediante sus relaciones.

- CALC

Cuando se utiliza el método CALC se brinda una llave con el cual se va a identificar el registro, se debe conocer la llave exacta para traer un único registro. Después la llave es utilizada para localizar la página en la base de datos, una vez que se localiza el registro en la página regresa el registro a un subesquema donde se recogen los resultados de las búsquedas (Mainframe Tutorials, 2009).

- Acceso mediante relaciones

Cuando se elige encontrar registros mediante sus relaciones existen dos maneras de realizarlo. Una es recuperar las relaciones hijas de un padre y la otra es buscar la relación padre mediante las hijas. Primero utilizando el método CALC se recupera un registro en específico. Cuando se tiene el registro se procede a realizar la búsqueda de sus relaciones.

Para realizar esto se utilizan los apuntadores que se guardan de los demás registros, sean padres o hijas. Con esto se pueden recuperar los datos o indicar que no se tiene ninguna relación (Mainframe Tutorials, 2009).

2.1.2.3. ALMACENAMIENTO

IDMS implementa la base de datos de manera física (en disco) mediante divisiones llamadas áreas, también conocidas por ser la división más grande localizable almacenada en la base de datos. Estas áreas contienen registros de una o varias estructuras, pero todos los registros de la misma estructura deben de estar localizadas en la misma área.

Un área puede estar a su vez dividida en páginas, que se conoce como la unidad lógica más pequeña de almacenamiento de la base de datos. La página es la unidad de datos que contiene los registros recuperados que serán movidos del buffer del sistema a la

base de datos. Estas páginas contienen los registros recuperados e información de control (Mainframe Tutorials, 2009).

IDMS/R divide sus páginas en cuatro segmentos:

- Encabezado: segmento de 16 bytes, el cual contiene información de la página.
- Cuerpo: segmento de 32 bytes, de los cuales se optimiza para tener el máximo espacio útil.
- Índices: segmento de 8 bytes, el cual apuntan a los diferentes registros almacenados.
- Pie de página: segmento de 16 bytes, contiene información de la página.

Cada página está numerada consecutivamente en un área, esta numeración es única para la página.

2.1.2.4. Observaciones

- No se tiene duplicidad en los registros
- Aún no se tiene una estructura de tablas, se representa como una estructura de red, y sus relaciones como ligas entre los registros.
- Las relaciones M:N son complicadas de implementar.

2.1.3. Bases de Datos Relacionales

Este es el modelo más utilizado en la actualidad, su principio es modelar los datos en tablas y relaciones entre esas tablas. La ventaja de este modelo es que se basa en relaciones matemáticas de conjuntos que ya están definidas para poder trabajar con los datos (Gálvez, s.f, p. 6, http://www.lcc.uma.es/~galvez/ftp/bdst/ Tema2.pdf).

2.1.3.1. Representación

Las bases de datos relacionales representan el conjunto de las relaciones de las tablas y los datos que contienen.

Las bases de datos relacionales utilizan tablas como estructuras lógicas donde se almacenan los datos. Estas tablas pueden ser vistas como archivos planos, las cuales buscan que los registros no tengan duplicidad y sean siempre consistentes. Pero se tiene la flexibilidad de ser tan estrictos en la duplicidad de los datos como se desee, por cuestiones de rendimiento e implementaciones físicas.

A diferencia de las representaciones anteriores se tienen claves o llaves para indicar restricciones sobre la tabla, estas llaves son de tipo:

- Llave primaria: Es una clave única que define a los demás atributos de la tabla.
- Llave foránea: Es una referencia a una clave única de otra tabla y determina la relación entre dos tablas.
- Índice: Es una estructura de datos la cual minimiza el número de lecturas de datos en disco.

Un concepto que tomamos en cuenta al modelar es el de dependencia de existencia, decimos que una relación es opcional o independiente si un registro en la entidad hija no necesita de la existencia de un registro en la entidad padre, una relación es obligatoria o dependiente cuando se necesita de un registro en la entidad padre para que exista un registro en la entidad hija.

Además se tienen diferentes tipos de relaciones, la identificativa y no identificativa o también conocidas como relaciones fuertes y débiles, estos nos indican como es la relación entre las dos entidades.

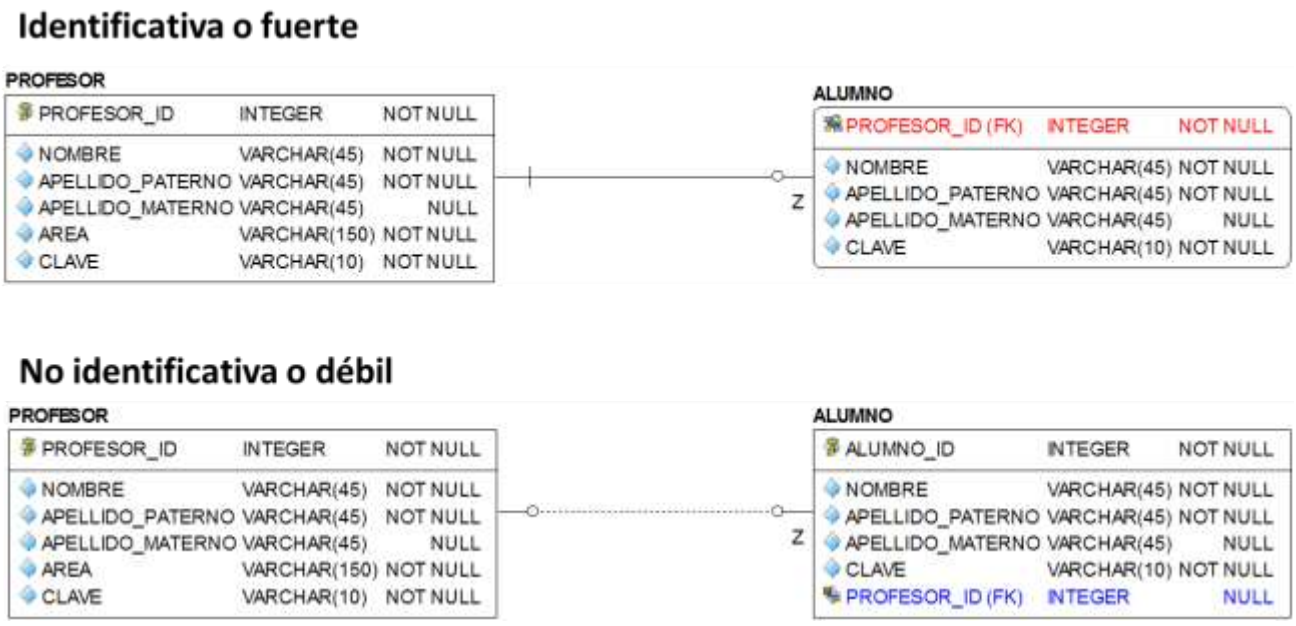
Las relaciones identificativas o fuertes indican que uno de los registros de la entidad padre va a identificar a una entidad hija, al realizar esto la llave primaria de la entidad padre pasa como llave primaria y foránea a la entidad hija, con esta regla se asegura que exista un registro en la entidad padre para tener un registro en la entidad hija y que sea una relación única. Cuando solo la llave primaria de la entidad hija es la llave primaria del padre se tiene una relación uno a uno entre las dos entidades, en caso de que la entidad hija tenga como llave primaria una llave compuesta, es decir que se

identifique por dos o más llaves primarias, se tiene una relación uno a muchos. Este tipo de relación se puede utilizar cuando se quieran agregar atributos a un registro en particular o de varios registros de diferentes entidades en el caso de las llaves compuestas.

Las relaciones no identificativas o débiles nos indican que un registro de una entidad hija dependiendo del caso pueda necesitar de un registro en la entidad padre o existir sin que en la entidad padre se tenga un registro que se asocie, también nos permite tener relaciones únicas entre dos registros o relaciones de un registro de una entidad con varios registros de otra, cuando sucede esto la llave primaria de la entidad padre pasa como una llave foránea en la entidad hija.

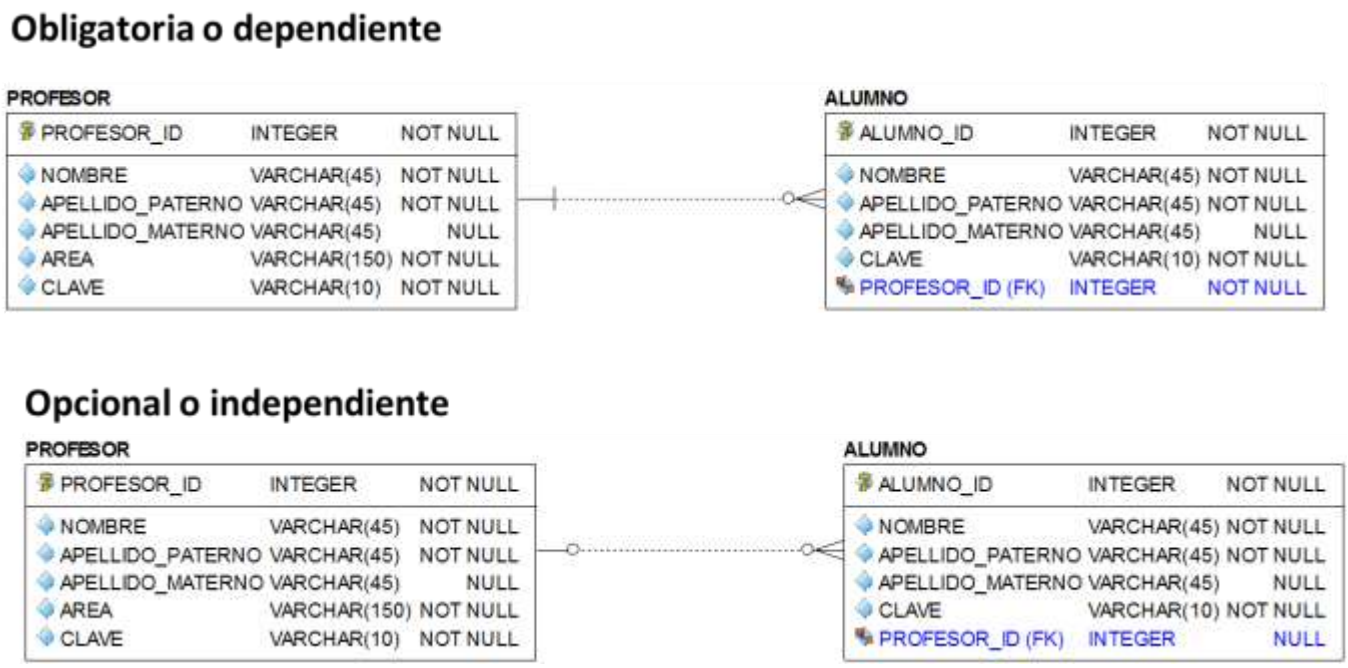
Estas características permiten obtener un mejor modelo, es decir definen de manera correcta la distribución y reglas que se le aplicarán a los datos sin importar el problema, esto además permite tener un mejor control sobre la consistencia de los datos.

Supongamos que tenemos una entidad Profesor y una entidad Alumno como se muestra en la imagen 13. En una se muestra una relación uno a uno identificativa y en otro una relación uno a uno no identificativa. En los dos tipos un registro de la entidad Profesor tiene relación únicamente con un registro de la entidad Alumno, pero en la relación identificativa debe existir un registro de la entidad Profesor para tener un registro en la entidad Alumno, en la no identificativa puede aplicar esta regla o no, es decir que pueden existir registros de la entidad Alumno si la necesidad de registros en Profesor.



Img 13. Ejemplo relación uno a uno identificativa y no identificativa en el modelo relacional

Como se muestra en el ejemplo de la imagen 14, en las relaciones uno a muchos un registro de la entidad Profesor se puede relacionar con muchos registros de la entidad Alumno, sin embargo, un registro de la entidad Alumno únicamente puede relacionarse con un registro de la entidad Profesor, también se muestra dos relaciones uno a muchos en el cual uno es obligatoria y la otra es opcional, en el caso opcional nos indica que se pueden guardar registros de Alumnos sin necesidad de tener registros de Profesores, mientras que en el caso de los obligatorios no se pueden guardar registros de Alumnos sin antes tener un registro de Profesor que los relacione.

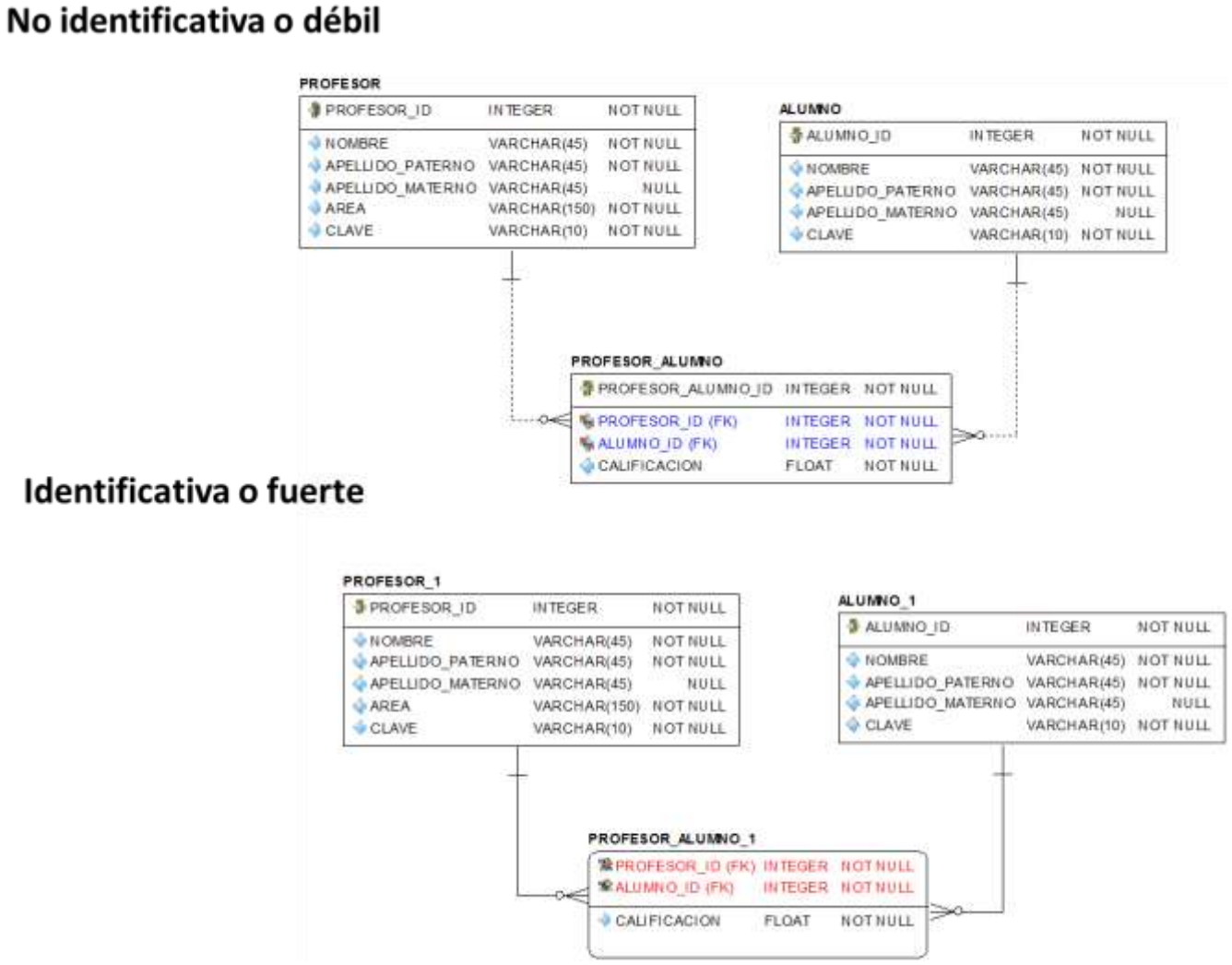


Img 14. Ejemplo relación uno a muchos en el modelo relacional

La relación muchos a muchos es la más compleja debido a que un registro de la entidad A se puede asociar a muchos registros de la entidad B y un registro de la entidad B se puede asociar a muchos registros de la entidad A.

Estas relaciones se pueden modelar de varias maneras dependiendo de las reglas de negocio y de las habilidades del diseñador. En este ejemplo utilizaremos el concepto de entidad débil, en el cual en la tercera tabla que los relaciona se tiene las llaves foráneas de cada tabla y además se agregan campos.

Como se observa en la imagen 15 se tienen dos ejemplos de relaciones muchos a muchos, en el caso de la relación identificativa solo se podría tener los registros de la calificación de un alumno por profesor, básicamente se tendrá una relación uno a muchos en donde se busque que los registros de profesores y alumnos sean únicos, en el otro ejemplo en el tipo de relación no identificativa se pueden tener varios registros de calificaciones de un mismo Profesor a un Alumno. Como vemos dependiendo de las reglas que nos indiquen se podría utilizar cualquiera de las dos.



Img 15. Ejemplo relación muchos a muchos identificativa y no identificativa en el modelo relacional

2.1.3.2. CONSULTA DE DATOS

Los manejadores de bases de datos relaciones, a diferencia de los modelos anteriores, manejan el lenguaje SQL (Structured Query Language) para realizar operaciones de creación y modificación de objetos con sentencias de tipo DDL (Data Definition Language) y de manipulación de datos con sentencias de tipo DML (Data Modification Language). Este lenguaje permite realizar operaciones algebraicas y el cálculo relacional entre tablas.

Cuando se realiza una consulta los manejadores de bases de datos relacionales realizan los siguientes pasos (Microsoft, 2016).

Se analiza la instrucción, se divide la sentencia y se traducen los diferentes bloques obtenidos al álgebra relacional, al realizar esto se obtienen los operadores, valores que se utilizan en la sentencia y el formato de salida requerido.

Se construye los árboles de consulta, en los que se describen los pasos lógicos que se van a desarrollar dentro de la sentencia, una sentencia puede generar más de un árbol de consulta, cada árbol es una expresión del álgebra relacional, generalmente el analizador sintáctico genera los árboles de consulta ordenando sus operaciones de la siguiente forma, primero se realizan los productos cartesianos, luego se ejecutan los joins, luego las operaciones de selección y por último las proyecciones. Como podemos notar el orden que genera por omisión es ineficiente por que realiza operaciones de join sobre datos que luego se eliminarán (A. Jaime, 2005).

Se analizan los posibles caminos a tomar y se decide por el procedimiento más eficiente modificando el orden de las operaciones de los árboles de consulta, eliminando registros innecesarios mediante la aplicación de las restricciones antes de hacer otra operación, para realizar esto las RDBMS utilizan reglas de transformación o también conocidas como reglas de equivalencia que modifican el orden de las operaciones, estas reglas son:

1. Una selección conjuntiva, es decir una selección dividida por varios ANDs en serie, se pueden dividir en una secuencia de selecciones individuales.
2. La operación de selección es conmutativa entre sí.
3. Solo se utilizan las últimas operaciones de una secuencia de proyecciones, las demás se pueden omitir.
4. Las operaciones de proyección y selección son conmutativas si se cumple que la selección sea sobre los campos establecidos en la proyección.
5. Los productos cartesianos son conmutativos entre sí, así como las operaciones de join.
6. Una selección es conmutativa a una operación de join si se aplica la selección a los atributos correspondientes a las entidades, si no se tienen ningún

atributos en una entidad solo se aplica la selección a la entidad que posee atributos.

7. Las operaciones de proyección son conmutativas a las operaciones de join y producto cartesiano, si se aplican las proyecciones de los atributos a sus entidades correspondientes. Cuando se tiene un join en el que el parámetro de asociación no está dentro de los atributos de la proyección, entonces se realiza una proyección del resultado obtenido.
8. Las operaciones de unión e intersección son conmutativas.
9. Las operaciones de diferencia no son conmutativas.
10. Las operaciones de join, producto cartesiano, unión e intersección son asociativas entre sí.
11. Operaciones de selección son conmutativas con operaciones de unión, intersección y diferencia.
12. Operaciones de proyección son conmutativas a las operaciones de unión.
13. Transformar condiciones en otra equivalente, por ejemplo aplicar leyes de DeMorgan.

También se aplican otras reglas como el costo de cada operador que aparece el árbol de consulta, para ello se estima cantidad de operaciones a disco que se utilizan. Para calcular el costo se toman en cuenta los siguientes parámetros:

- Tamaño de la entidad, es decir el número de registros de una entidad
- Espacio en disco de un registro, la cantidad de Bytes que ocupa un registro.
- Número de bloques de hoja que tiene una entidad.
- Cantidad de registros por bloque
- Los índices que se tienen.
- Cantidad de valores distintos de una entidad
- El tiempo de uso del CPU, cantidad de memoria principal utilizada y disponible.
- La comunicación de envió y recuperación de los resultados

El objetivo de estas operaciones son la reducción del número de bloques a recuperar, aun cuando se realizan operaciones sobre los mismos esquemas el costo puede determinar el orden en la ejecución de la sentencia, aunque este parámetro no siempre es el más óptimo, ya que el tiempo que se requiere para realizar el cálculo del costo es considerable (Oracle, 2002).

Se empieza a ejecutar el plan de ejecución, se recuperan los datos y se les aplican las operaciones requeridas, en el orden que proporciona el plan de ejecución y por último se devuelven los resultados obtenidos en el formato indicado.

Cada manejador tiene un método por el cual optimiza las peticiones para obtener el mejor desempeño en tiempo de respuesta, para poder decidir qué opciones tomar las RDBMS construyen el plan de ejecución.

El RDBMS maneja el concepto de plan de ejecución, el cual nos muestra cómo ejecutar una sentencia SQL de manera que el procesamiento sea más eficiente. El plan de ejecución se puede representar de dos maneras: como una tabla y en forma de diagrama de árbol que tienen por elementos las operaciones que se realizan sobre las tablas con el número de registros que se obtienen de la operación. Un diagrama se lee de abajo hacia arriba y de izquierda a derecha hasta llegar a lo más alto del árbol (Oracle,2012).

```
SQL> SELECT plan_table_output
FROM
TABLE(DBMS_XPLAN.DISPLAY('plan_table',null,'typical'));  2    3

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 1277587874

-------------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          |     5 |   150 | 27707   (2)| 00:00:02 |
|   1 |  SORT ORDER BY      |          |     5 |   150 | 27707   (2)| 00:00:02 |
|   2 |   HASH GROUP BY     |          |     5 |   150 | 27707   (2)| 00:00:02 |
|*  3 |    TABLE ACCESS FULL| LINEITEM |  5935K|   169M| 27402   (1)| 00:00:02 |
-------------------------------------------------------------------------------


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("L_SHIPDATE"<='1998-09-16')

15 filas seleccionadas.

SQL>
```

Img16 . Ejemplo del plan de ejecución en Oracle

El plan de ejecución decide el orden para ejecutar de manera física las operaciones sin que se pierda el orden lógico de la consulta, por ejemplo, si se realiza primero una operación de JOIN antes se aplican las condiciones de reducción WHERE sobre las tablas de entrada, si no se hiciera esto se harían muchas más iteraciones de las necesarias, ya que la condición reduce el número de registros a iterar y la operación de join es más eficiente. Si se utilizará un OUTER JOIN entonces el manejador ejecuta la reducción WHERE después de recuperar las tablas para no perder el orden lógico de la sentencia.

Para decidir qué operaciones realizar se utiliza la información estadística. Entre esta información se encuentra la cardinalidad, es decir el número de registros estimados que se recuperan en cada operación, el tipo de acceso que se utiliza para recuperar los datos, es decir si se realiza una búsqueda completa sobre la tabla o se utiliza algún índice, los métodos join que utilizan, el tipo de join que se requiere, el orden en que se

realiza el join, si se divide una tabla en partes más pequeñas o si se requiere paralelizar la operación; para obtener esta información se utiliza el diccionario de datos.

El diccionario de datos contiene tablas y vistas que definen cómo se encuentran distribuidos los datos y objetos que componen a la base de datos, esto nos permite tener información precisa de los datos. Entre los datos que almacena se encuentran los nombres y tipos de objetos que contiene, el número de registros que contiene una tabla, el espacio que ocupa en disco, control de acceso a los objetos, la región física donde se encuentran los datos, la estructura de las tablas, entre otras.

Cuando se tiene la información se ejecuta el plan que se generó, accede a las tablas, aplica las operaciones en el orden indicado y finalmente se obtiene la información.

2.1.3.3. Región de Control

Cada RDBMS administra sus datos de distinta manera, en el caso de Oracle para gestionar las peticiones, recuperación de los datos y proporcionar servicios, se utilizan instancias las cuales son estructuras que se almacenan en memoria y que contienen procesos para desarrollar las tareas que se piden (Oracle, 2002).

Una instancia en Oracle se divide en dos grandes áreas el SGA (*Área Global de Sistema*) y el PGA (*Área Global de Programa*) (Sánchez, Jorge, 2004).

Img 17. Ejemplo de instancia RDBMS

- El SGA contiene los datos de control de la instancia, los bloques de datos que se leen de disco, el buffer para recuperar los logs de transacciones y los planes de ejecución generados. El SGA está formado de los siguientes componentes (Oracle, s.f):
  - Shared pool: es un área de memoria en la cual se almacena, divide, interpreta y ejecuta las sentencias SQL y PL/SQL, parámetros del sistema e información del diccionario de datos. El shared pool se divide además en las siguientes subregiones:
    - Library Cache**:** es una estructura que almacena las instrucciones SQL y PL/SQL que se ejecutaron, además cuenta con estructuras de control como candados y administradores de la memoria, en este caso en particular utiliza el algoritmo LRU (*Último Recientemente Usado*), la utilidad principal de esta estructura es la de reusar código previamente ejecutado, si se ejecuta una misma sentencia en lugar

de generar el mismo código utiliza el que se tiene previamente almacenado.

- Data Dictionary Cache**:** es un espacio de memoria que contiene las referencias de la información de tablas y vistas, para esto accede al Diccionario de Datos, cuando se realiza una instrucción y se valida que el objeto exista en el diccionario de datos y si la verificación fue exitosa, se almacena el nombre del objeto en esta región para que cuando se utilice nuevamente el mismo objeto no se realice una nueva comprobación.
- Server Result Cache**:** almacena el conjunto de resultados que se obtuvo de realizar una consulta SQL o función PL/SQL.
- Reserved Pool**:** es una región de memoria en la que se puede almacenar largos pedazos contiguos de memoria. Esto permite limitar la posibilidad de que la memoria se quede sin espacio de memoria por desperdiciar memoria por fragmentación.

➢ Database Buffer Cache: es el área de memoria que almacena copias de bloques de datos leídos de los archivos de datos. La utilidad de esta área es la de mantener los bloques de datos más usados en memoria y escribir en disco aquellos bloques que se utilicen menos frecuentemente. También para tener un mejor desempeño en llamadas al disco después de realizar un COMMIT la base de datos realiza una operación llamada *lazy write* en la cual guarda las peticiones en buffers de memoria y luego la escribe dentro del disco cuando encuentra un momento adecuado.

➢ Redo Log Buffer**:** es un buffer circular que almacena los cambios a realizar sobre la base de datos. Estos contienen la información suficiente para restaurar los cambios realizados por sentencias DML o DDL. Luego de ir recibiendo las peticiones se tiene un proceso llamado *log writer* que escribe en disco las peticiones realizadas.

➢ Large Pool**:** es un área de memoria opcional que se utiliza cuando se realizan operaciones que requieren más memoria de la que se tiene en el

share pool, por ejemplo para operaciones de respaldo de datos o como memoria de sesión.

- ➢ Java Pool**:** área de memoria que almacena código Java.
- ➢ Streams Pool**:** almacena mensajes del buffer y provee memoria para procesos y aplicaciones de Oracle Streams.
- ➢ Fixed SGA**:** área que contiene información general sobre el estado de la base de datos y la instancia necesarias para los procesos en segundo plano, además de información para la comunicación de procesos, por ejemplo los candados establecidos en los registros.

❖ PGA**:** área de memoria que se ocupa específicamente para cada proceso o hilo de ejecución, esta área es privada a cada proceso. En este espacio se almacena los datos del proceso, como variables, status, etc. El PGA está compuesto de la siguiente manera (Oracle, s.f).

- ➢ Sort Area**:** espacio que se utiliza para operaciones de ordenamiento de registros.
- ➢ Hash Area**:** espacio de memoria que se utiliza para construir la *tabla hash* de una operación hash join.
- ➢ Bitmap Merge Area**:** área de memoria para reunir los datos recuperados de realizar una búsqueda con múltiples índices bitmap.
- ➢ Session Memory**:** es un espacio de memoria reservado para guardar variables de sesión como la información de conexión, permisos del usuario y otra información requerida para la sesión de la base de datos.
- ➢ Persistent Area**:** área que contiene valores de las variables de vinculación, estos valores son proporcionados a las sentencias SQL que se ejecutan.
- ➢ Run-Time Area**:** una área que contiene la información del estado de ejecución de las consultas, por ejemplo el número de registros encontrados, etc.

2.1.3.4. ALMACENAMIENTO

Los manejadores de bases de datos, suelen almacenar la información en disco en unidades físicas llamadas *"bloques de disco"* o *"páginas"* las cuales se localizan en diferentes posiciones del disco.

Una base de datos relacional separa los datos en dos capas independientes, lógica y física, esto quiere decir que no se tiene que interactuar directamente una con la otra para administrar los datos, esto nos permite realizar cambios en la parte lógica sin alterar directamente a la base de datos. Cada aplicación organiza los datos en función de sus necesidades, permite realizar cambios en la estructura física sin modificar las aplicaciones, por ejemplo cambiar el nombre de una base de datos no cambia el nombre de las tablas. En el Oracle, la parte física lo componen los datafiles, redo log y control files; la estructura lógica lo conforman los tablespace y los objetos, es decir, tablas, índices, vistas, etc (Fernández, Jesualdo, s.f).

Específicamente en Oracle cada tablespace es una colección de objetos y está asociada con diferentes bloques en disco. El tamaño de los bloques puede ser de 2 KB, 4 KB, 8 KB, 16 KB y 32 KB, el cual se puede fijar dependiendo de los usos y desempeño requerido. Al conjunto de bloques continuos se le conoce como extent también conocido como una unidad lógica, al conjunto de extents se le conoce como segmento, a cada instancia se le asigna un segmento y si requiere de más espacio, se le puede asignar varios segmentos (Vegas, Jesús, 1998).

2.1.3.5. OBSERVACIONES

- Se pueden modelar problemas de la vida real mucho más fácil, este tipo de modelado nos permite representar de mejor manera el comportamiento de los diferentes tipos de relaciones.
- El estándar SQL facilita el modo en que se construyen y utilizan las bases de datos.
- Debido a que se basa en un modelo matemático, se pueden realizar diversas operaciones de teoría de conjuntos sobre los datos.

- Se tiene una gama muy variada de manejadores de base de datos que implementan el modelo relacional, los cuales proporcionan diversas configuraciones que permiten tener un mejor rendimiento dependiendo del problema que se tenga.

2.1.4. Bases de Datos Orientadas a Objetos

Las Bases de Datos Orientadas a Objetos (BDOO) siguen los conceptos básicos de la Programación Orientada a Objetos (POO) al modelar los datos como clases con sus atributos. Entre las clases hay relaciones que definen el comportamiento que tendrán las clases entre ellas.

Este tipo de bases de datos surgen de la problemática de utilizar un lenguaje de programación orientado a objetos con una base de datos relacional (Gálvez, XXXX, p. 9, http://www.lcc.uma.es/~galvez/ftp/bdst/Tema2.pdf). Al intentar hacer esa combinación nos encontramos con muchos problemas porque no son compatibles completamente debido a que tratan los datos de diferente manera.

2.1.4.1. Representación

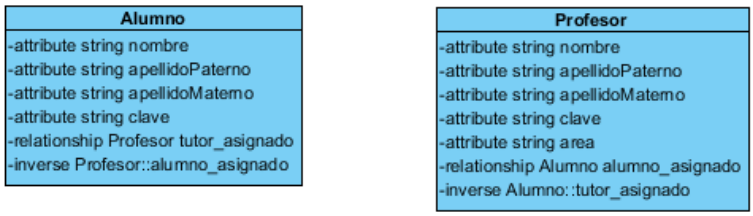
Este modelo utiliza el concepto de objetos para la administración de los datos. Un objeto es una representación de una entidad, es decir que modela cualquier cosa del mundo real. Los objetos cuentan con un comportamiento y característica que definen su estado.

En las BDOO cada objeto se identifica por un simple IDO (*Identificador de objeto*), mediante este identificador los objetos se pueden localizar y construyen redes de objetos. En cada objeto se implementan sus relaciones, incluyendo los identificadores de los objetos que lo relaciona, estos IDOs son únicos incluso entre las relaciones, a diferencia de las llaves foráneas que tienen el valor de su referencia (Bertino, Elisa y Martino Lorenzo, 1993).

Además estos identificadores son asignados automáticamente por el sistema y se implementan a bajo nivel por lo que su construcción es más rápida.

Una característica importante de los objetos es que solo se puede consultar las relaciones predefinidas en cada objeto, por lo que no es posible realizar tareas tan extensas de búsqueda. Por otro lado el rendimiento para realizar búsquedas específicas es mucho mejor.

Para representar una relación uno a muchos, se define un atributo en el objeto hijo, o el que tiene la relación, este atributo tendrá el valor del IDO del padre. Por su parte el padre o el que provee la relación, tiene como atributo el conjunto de IDOs que confirman sus hijos (Marqués, Merche, 2012).



Img 18. Ejemplo relación uno a muchos en el modelo orientado a objetos

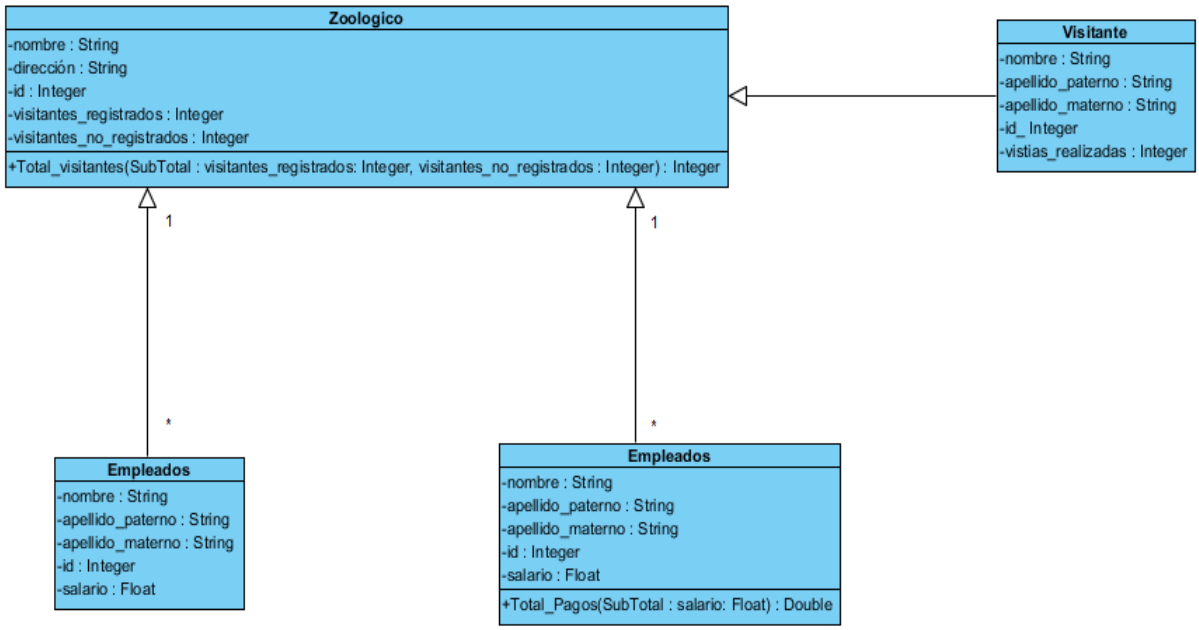
Para las relaciones muchos a muchos se pueden desarrollar de varias maneras dependiendo el caso de la relación.

Por ejemplo si en la relación no existen atributos entre las dos relaciones, entonces solo se necesita que ambos objetos tengan un atributo que contendrá el conjunto de valores del otro objeto.

En el caso que la relación tenga datos específicos se requiere un tercer objeto, como en la entidad intermedia de las Bases de Datos Relacionales.

La otra forma de relacionar objetos en las BDOO es mediante la herencia, se puede definir esta relación como *"es un"* donde las subclases son casos específicos de la superclase. También se definen como *"extiende"* en la que la subclase tiene atributos idénticos a la superclase además de poseer atributos propios. La diferencia que podemos observar es que en el tipo *"es un"* se limita a utilizar los atributos de la

superclase y en el tipo *"extiende"* se puede utilizar los atributos de la superclase y además atributos propios (Marqués, Merche, 2012).



Img 19. Ejemplo relación con herencia en el modelo orientado a objetos

2.1.4.2. CONSULTA DE DATOS

Las BDOO utilizan un lenguaje llamado Object Query Language (OQL), el cual es un lenguaje declarativo basado en SQL-92 que permite realizar consultas sobre los objetos de la base de datos. Como está basado en SQL permite el manejo de conjuntos, estructuras y listas a un alto nivel (Besemal, Isabel, s.f).

Este lenguaje no soporta operaciones de actualización, para realizar estas operaciones se tiene que utilizar los métodos definidos en los objetos.

Para poder realizar consultas, OQL utiliza la estructura de cada objeto para localizar los atributos requeridos y el campo de búsqueda necesario.

OQL puede traer como resultado de la operación un objeto que se puede utilizar en otra consulta, además de poder realizar operaciones matemáticas en las consultas.

Un ejemplo del lenguaje OQL es como el siguiente:

SELECT a.name FROM a in student WHERE a.key = '567342';

Aunque este lenguaje es utilizado como base en varios manejadores, no todos lo han implementado debido a su complejidad y algunos utilizan un propio lenguaje, como es el caso de Versant.

2.1.4.3. Región de Control

Versant es un manejador de bases de datos orientado a objetos (OODBMS) que maneja varios módulos de software para administrar los procesos (Versant, 2005).

Versant Manager: realiza la validación de los objetos, manejo de las consultas, administración de grandes consultas, creación de esquemas, relaciones, versiones y administración de checkouts y check in.

Versant Server: recuperación de objetos, realizar operaciones de actualización, recuperar las páginas de disco, soporte de consultas, manejo de las clases de almacenamiento, índices, transacciones, manejo de login y candados.

Virtual System Layer: es una capa de software que comunica el módulo de Versant Server con el sistema operativo, específicamente con el hardware. Este módulo traduce los mensajes para que sea entendido por el sistema operativo y realizar la operación.

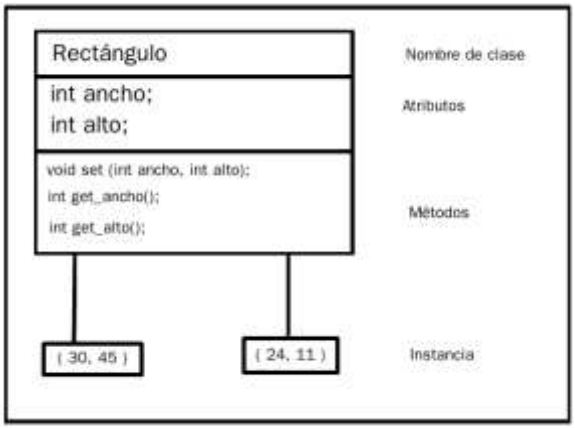
2.1.4.4. Almacenamiento

En Versant cada base de datos consiste en volúmenes de archivos o dispositivos raw almacenados en lugares de disco. Estos se puede dividir en dos volúmenes (Versant, 2005):

System Volume: este módulo se encarga de almacenar las clases e instancias de objetos en disco y son creadas en el momento en que se crea la base de datos.

Data Volume: volúmenes extra de espacio en disco para incrementar la capacidad de la base de datos.

Physical Log Volume y Logical Log Volume: estos módulos son utilizados para guardar las transacciones activas y proporcionar información para realizar operaciones de recuperación de estado y de rollback. Estos módulos también son creados cuando la base de datos es creada.



Img 20. Almacenamiento en bases de datos orientado a objetos

2.1.4.5. OBSERVACIONES

- Este modelo proporciona un mejor rendimiento en transacciones, pero se limita a realizar búsquedas a las relaciones del objeto, y no una general.
- Las operaciones se realizan a un bajo nivel.
- No tienen un lenguaje de consulta de datos estandarizado.

2.2. MANEJADORES DE BASES DE DATOS COLUMNARES

Las manejadores de bases de datos columnares son un tipo de base de datos que almacenan los datos en el disco de manera diferente a como lo hacen las bases de datos de almacenamiento por registro, este almacenamiento se hace, como dice su nombre, por columnas.

A pesar de las diferencias en el almacenamiento entre las bases de datos columnares y las bases de datos relacionales, ambos se pueden modelar de la misma manera, es decir que una base de datos columnar es independiente del modelado lógico que se emplee, su diferencia recae en la forma en que almacena los datos en disco. Así mismo, los ambientes transaccionales y los ambientes data warehouse pueden usar

ambos tipos de bases de datos indistintamente, sin embargo, pueden tener diferencias en el desempeño. Esas diferencias serán analizadas más adelante cuando se hagan pruebas de desempeño para comparar el almacenamiento columnar y el almacenamiento por registro.

Estas características nos permiten realizar operaciones analíticas sobre esta cantidad de datos, por ejemplo, cuando se requiere la suma de todas las ventas realizadas, solo se utiliza una sola columna de cada registro para realizar la suma, todas las demás columnas no son necesarias para que se obtenga esta información.

Como las bases de datos columnares son utilizadas para ambientes OLAP (On-Line Analytical Processing), en los que se obtiene información de una considerable cantidad de datos. Para que la información se procese de manera rápida y correcta las bases de datos columnares proporcionan las siguientes propiedades:

- Índices relacionados a columnas, lo que permite tener un acceso rápido a los datos, pero complicado para los registros.
- Compresión de los datos para no desperdiciar espacio en disco y guardar la mayor cantidad de datos en disco.
- Operaciones sobre columnas de datos mucho más rápido ya que no se tiene que realizar demasiadas operaciones de I/O en disco.
- Esta tecnología es independiente del modelado que se utilice para generar una base de datos.

Otras de las características de las bases de datos columnares son las siguientes:

- Tiempo de carga**:** Se refiere a cuánto tiempo se tarda en cargar los datos en una base de datos columnar. El tiempo de carga varía dependiendo de cómo se organizan los datos en la base de datos. Los tiempos de carga son menores en comparación de las bases de datos relacionales, esto se debe a que las bases de datos columnares pueden asignar hilos de ejecución concurrentes para atender la carga de cada columna, es decir, las cargas se paralelizan por columna.

- Carga incremental**:** Una vez que un conjunto inicial de datos se ha cargado, se puede ir haciendo cargas incrementales sin tener que reconstruir o hacer recargas completas de los datos.
- Comprensión de datos**:** Algunos sistemas columnares pueden comprimir la fuente de datos y archivos resultantes a fin de tomar una fracción de espacio en el disco original. La compresión y descompresión de los datos tiene un impacto mínimo en las operaciones de escritura y lectura ya que no se utilizan algoritmos de compresión sino que los datos solo se mapean a representaciones más sencillas que el dato original a través de mapas de bits.
- Limitaciones estructurales**:** Las bases de datos columnares son independientes del diseño lógico que se utilice, es decir el mismo diseño aplica tanto para una base relacional como para una base de datos columnar.
- Técnicas de acceso**:** Prácticamente la mayoría de las bases de datos columnares utilizan SQL para la administración, creación y consulta de datos.
- Rendimiento**:** Por lo general estos sistemas superan a los sistemas de relaciones en casi todas las circunstancias.
- Escalabilidad**:** El punto de las bases de datos columnares es obtener buenos resultados en bases de datos muy grandes, con una escalabilidad posible hasta en petabytes.

3. Características de una Base de Datos Columnar

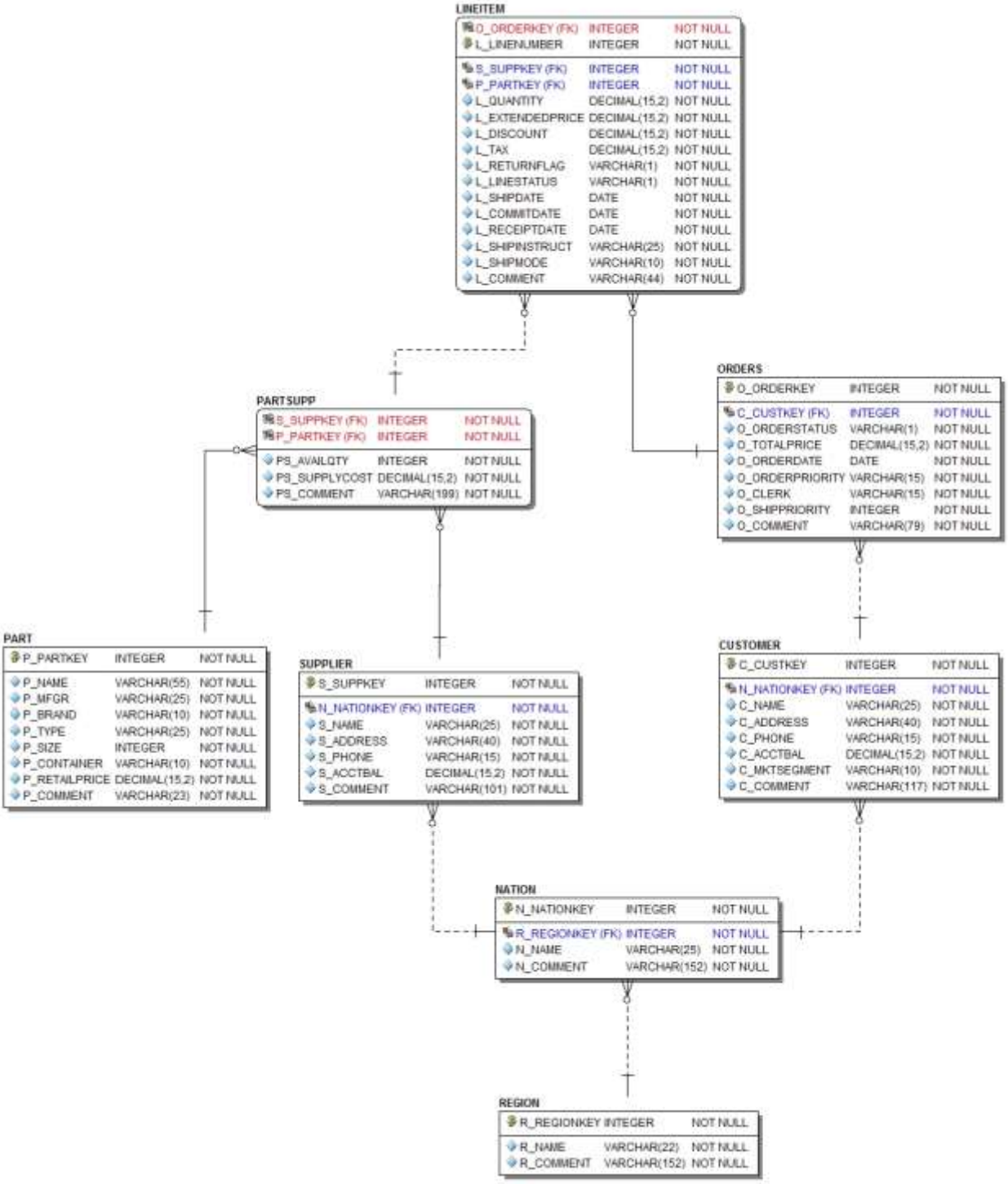
En el presente capítulo analizaremos los algoritmos que utilizan las BDC para realizar las operaciones de búsqueda, inserción, actualización y eliminación de datos. Así mismo, realizaremos la comparación de desempeño entre las BDC y las bases de datos de almacenamiento por renglón en las operaciones antes mencionadas.

No mencionaremos el nombre de los productos que utilizaremos, sin embargo, nombraremos al manejador de bases de datos columnares como CDBMS y al manejador de bases de datos de almacenamiento por renglón como RDBMS.

3.1. Consideraciones para las pruebas de desempeño

Para realizar las pruebas de desempeño utilizaremos un ambiente OLAP. Obtendremos el ambiente de la organización Transaction Processing Performance Council (TPC). Para el ambiente OLAP utilizaremos el estándar TPC-H obtenido de la siguiente página: http://www.tpc.org/. Para simular el ambiente OLTP utilizaremos el ambiente anterior con la diferencia de que haremos inserciones, eliminaciones y actualizaciones de tal manera que podamos hacer que se comporte de una manera transaccional.

El diagrama entidad/relación del ambiente OLAP es el siguiente:



Img 21. Imagen con el modelo de la base de datos

Estas pruebas se realizaron en un Sistema Operativo Ubuntu 16.04 LTS de 64-bit, memoria de 7.8 GB, procesador Intel Core™ i3-2310M CPU @ 2.10GHz × 4, con un disco duro de 978.1 GB.

3.2. ALMACENAMIENTO DE DATOS

Como se mencionó anteriormente, las BDC almacenan los datos en disco de una manera diferente a como lo hacen las bases de datos de almacenamiento por registro. En este apartado detallaremos más ese almacenamiento para que más adelante podamos comprender la diferencia de los tiempos al realizar las pruebas entre los dos motores de bases de datos.

El almacenamiento columnar y el almacenamiento por registro son dos paradigmas muy grandes que, aunque se utilizan para almacenar datos en discos, tienen muchas diferencias. Para empezar a compararlos primero tenemos que hablar de sus fundamentos.

- ➢ Almacenamiento por registro

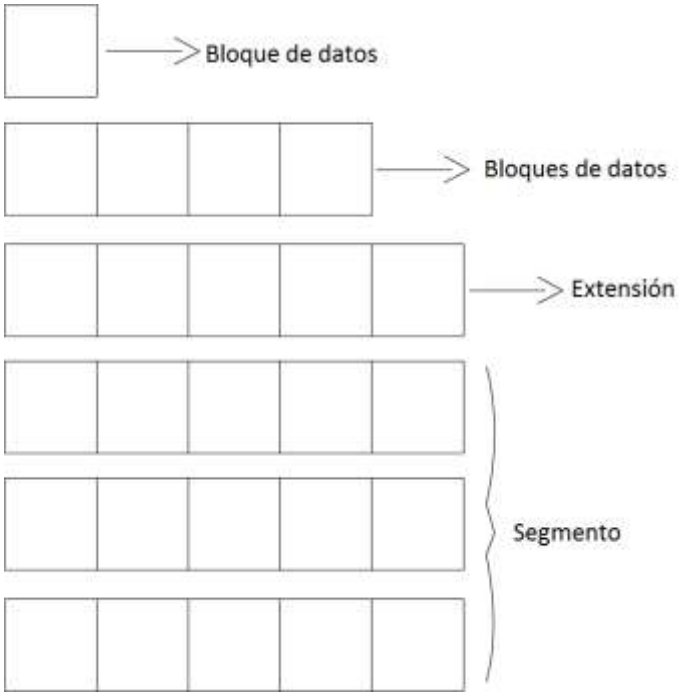
El almacenamiento por registro es el que más se ha utilizado a lo largo de los años debido a su facilidad de implementación y que la mayoría de las empresas manejan productos con esas características. Se fundamenta en la necesidad de soportar grandes cantidades de actualizaciones sobre pocos datos (MacNicol, Roger y French, B., s/f). Para poder cumplir con esa necesidad, los motores de bases de datos que utilizan el almacenamiento por registro siguen algunas reglas para cumplir su objetivo. Por ejemplo, las páginas de disco donde se almacenan los datos deben ser de un tamaño pequeño para poder realizar las lecturas y escrituras de datos de una manera más rápida, esto teniendo en cuenta que son pocos los datos que sufren modificaciones en una misma transacción (MacNicol, Roger y French, B., s/f).

Para poder entender mejor esto analizaremos la estructura del RDBMS que utilizaremos en el proyecto.

El RDBMS posee la siguiente arquitectura para poder almacenar la información, la unidad más pequeña de almacenamiento se llama bloque de datos. Los bloques de datos pueden tener 5 tamaños diferentes dependiendo de las necesidades del sistema. Para sistemas de Procesamiento de Transacciones en Línea (OLTP por sus siglas en inglés) se recomienda usar 2 KB o 4 KB, mientras que para Sistemas de Soporte a la Decisión (DSS por sus siglas en inglés) se recomiendan 8 KB, 16 KB o 32 KB. Varios bloques de datos almacenados de manera contigua y que cumplen un propósito específico se llaman extensiones.

El nivel más grande de almacenamiento se llama segmento, y es un conjunto de extensiones que también cumplen el mismo propósito. Cada segmento sirve para un único propósito. Por ejemplo, si se desea crear una tabla, esta se le guarda en un segmento específico y ese mismo segmento no podrá alojar a ningún otro objeto (Oracle, 2011).

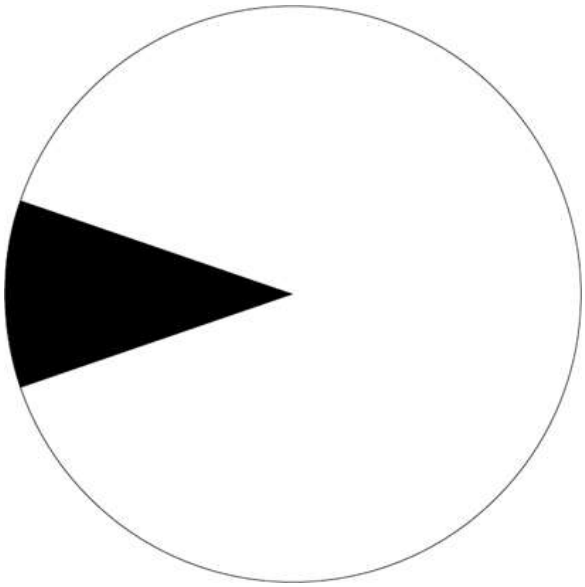
En la siguiente imagen podemos ver de una manera gráfica los niveles antes mencionados.



Img 22. Niveles de almacenamiento físico en el RDBMS

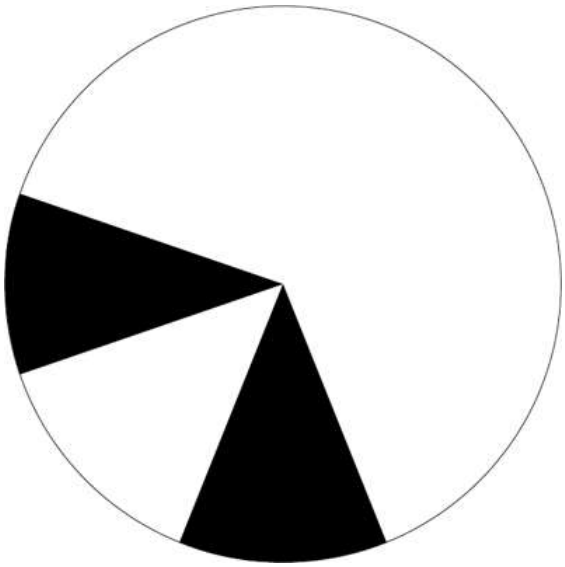
Cuando se ingresa información a la base de datos, cada segmento va recibiendo los extensiones para almacenar información de uno en uno, y cuando necesita más espacio para almacenar información, el RDBMS le asigna un extensión más. Debido a eso es posible que no todas las extensiones sean contiguas en el disco.

Tomemos por ejemplo una tabla de “empleados” para analizar su almacenamiento. Supongamos que la primera extensión del segmento dedicado a la tabla “empleados” se ha utilizado por completo. En la siguiente imagen podemos ver su representación en disco, siendo la parte sombreada el área donde se almacena la tabla.



Img 23. Imagen que representa el segmento con una extensión ocupada

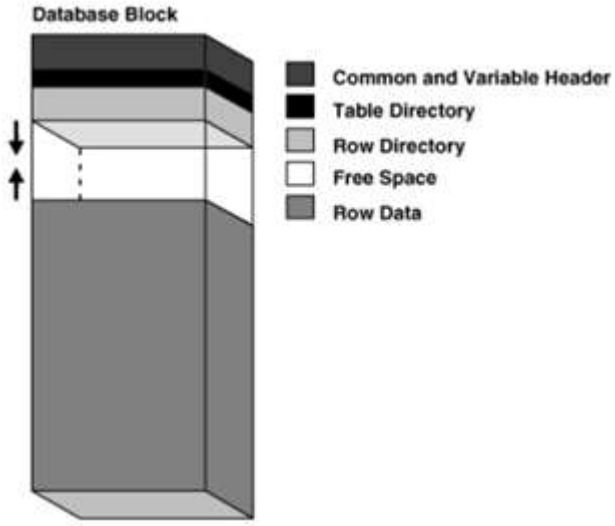
Posteriormente, se intentan almacenar nuevos registros. Lo que hace el RDBMS es asignarle un nuevo espacio de disco, es decir una extensión, al segmento donde se almacena la tabla, como se muestra a continuación.



Img 24. Imagen con nuevo segmento asignado

Como podemos observar no toda la información de la tabla se encuentra contigua en el segmento. Sin embargo existen varios métodos para desfragmentar el disco y poder tener la información de los segmentos juntos en el disco. Si la información no se encuentra contigua el manejador tendrá que realizar más lecturas para poder extraer o modificar los datos, de ahí surge la importancia de desfragmentar el disco periódicamente.

Analizaremos la estructura de los data block para saber cómo se almacena la información. Cada data block se divide en 5 partes:



Img 25. Imagen con la división de un data block

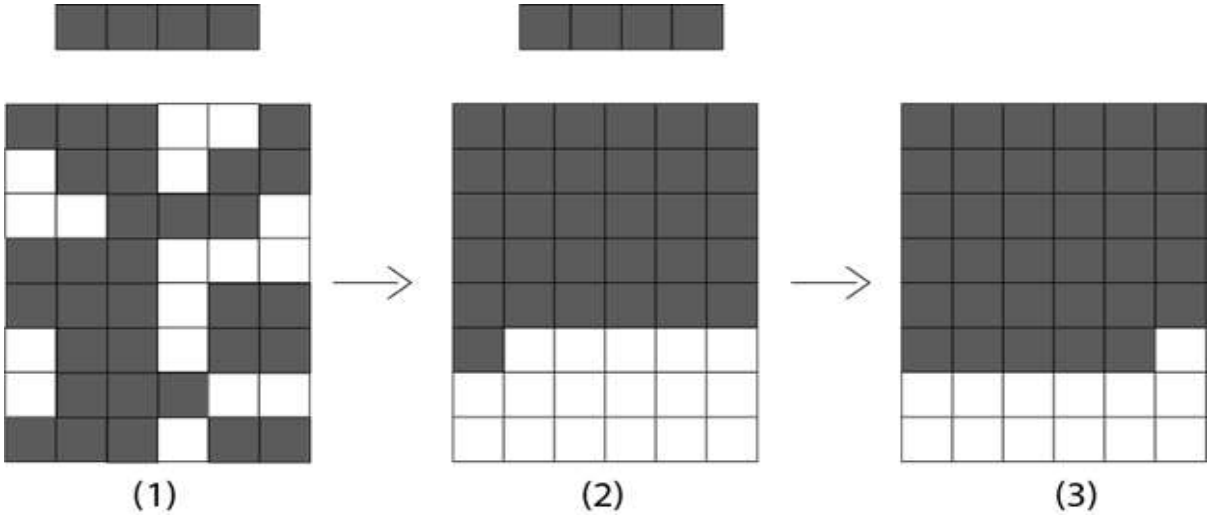
- *Header* (*Common and variable header*). Contiene la dirección del bloque, el tipo de información que se almacenará, por ejemplo tablas o índices, entre otros datos generales.
- *Table directory*. En esta sección se guarda la información de la tabla cuyos registros se encuentran almacenados en el data block.
- *Row directory*. Aquí se guarda la información de los registros que posee el bloque, por ejemplo, la dirección en la que se encuentra cada registro en el *row data*.
- *Overhead.* El *Header*, *Table directory* y *Row directory* juntos se conocen como Overhead del data block. El tamaño del *Overhead* es variable, aunque en promedio alcanza entre 84 y 107 bytes.
- *Row data*. En esta zona se almacenan los datos de la tabla o índice a la que el data block está destinado.
- *Free space*. Es un espacio sin información que se utiliza cuando se insertarán más registros al *row data* o cuando se realizan actualizaciones sobre registros que requieren más espacio del que ocupaba su versión anterior.

El manejo más importante que tiene un data block es el de su *free space*. La administración del f*ree space* se puede realizar de manera manual y automática

dependiendo de la configuración de la base de datos. El manual del RDBMS nos recomienda usar la administración automática debido a que proporciona algunos beneficios, por ejemplo, es más fácil de usar, se optimiza la utilización del espacio disponible, tiene un mejor ajuste de tiempo a las variaciones en accesos concurrentes, entre otros (Oracle, 2011).

Así como una actualización de datos en el *row data* puede ocupar parte del *free space*, también se presentan dos casos en los que se libera espacio de datos haciendo el *free space* más grande. Estos casos son en la sentencia DELETE, ya que se eliminan registros, y en la sentencia UPDATE, solo que en este caso la actualización ocupa menos espacio del que ocupaba su versión anterior.

Debido a la naturaleza de las sentencias de inserción, eliminación y actualización de datos, el espacio que se libera o se utiliza tiene como consecuencia que el área de *free space* esté esparcida en el área de la información. Para poder fusionar el *free space* y poder utilizarlo de mejor manera se requiere realizar varios procesos de desfragmentación que utilizan tiempo de procesamiento, haciendo que el desempeño de la base de datos descienda si se realiza constantemente. El RDBMS solo realiza ese proceso en dos situaciones específicas. La primera de ellas es cuando no se tiene espacio contiguo suficiente para poder realizar una inserción o actualización de datos. Veamos la siguiente imagen.



Img 26. Ejemplo de desfragmentación de disco

En el punto 1 se tiene un registro que ocupa 4 bloques esperando ser insertado, sin embargo, no hay suficiente espacio libre contiguo para realizar la operación. En el paso 2 se fusionan los espacios libres dando como resultado un área contigua de gran tamaño. Finalmente, en el paso 3, se inserta el registro.

Otro punto a considerar en el área de *free space* es que, en determinadas circunstancias, la información que se quiere agregar es demasiado grande como para caber en un simple data block. Esto se puede presentar en las siguientes dos circunstancias:

- El registro que se desea insertar es demasiado grande para poder ser insertado en un data block. En este caso la información se inserta en dos o más data blocks, dependiendo del tamaño de la información.
- El registro se actualiza y no cabe en un sólo data block. En este caso todo el registro es migrado a un nuevo data block. El RDBMS mantiene el registro original apuntando a la localidad del nuevo registro con la información migrada.

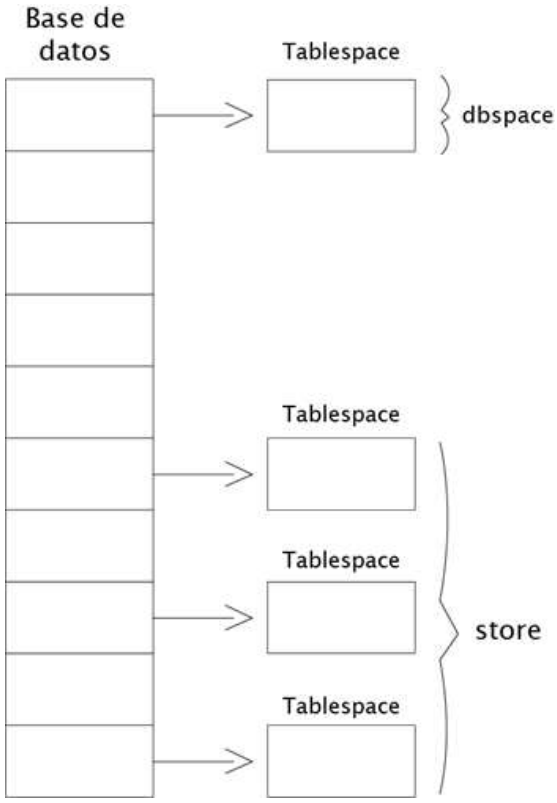
En ambos casos las operaciones de entrada/salida ven su desempeño afectado debido a que ahora se deben escanear dos o más data blocks para poder obtener la información que se desea. Por lo tanto, añadir o eliminar un índice o una columna en las bases de datos que utilizan el almacenamiento por renglón es muy costoso debido a que se deben actualizar todas las páginas de datos para poder manipular la información requerida.

➢ Almacenamiento columnar

Por otro lado, el almacenamiento columnar utiliza páginas de disco muy amplias para poder almacenar la mayor cantidad de celdas de una columna en una sola lectura (MacNicol, Roger y French, B., s/f).

Como se puede observar en la siguiente imagen, todas las bases de datos del CDBMS están preasignadas a un espacio de memoria permanente, ya sea un archivo de sistema o una partición. Estas bases de datos están formadas por tablespaces. Los

tablespaces son unidades de almacenamiento que permiten administrar un subconjunto de toda la información de manera lógica.



Img 27. Imagen con las unidades de almacenamiento lógico en base de datos columnares

Un tablespace está compuesto de uno o más archivos de sistema operativo es llamado dbspace.

Un store es uno o más dbspaces que almacenan información temporal o persistente para un propósito específico. En CDBMS que utilizaremos para realizar las pruebas cuenta con 5 stores, cada uno con un propósito diferente.

El Catalog Store, o simplemente el catálogo, contiene metadatos para las bases de datos. Esta metadata describe la estructura que deben tener las tablas, columnas e índices. El catálogo es similar al diccionario de datos que se utiliza en otras bases de datos.

IQ_SYSTEM_MAIN es un dbspace especial que contiene todas las estructuras necesarias para poder iniciar las bases de datos. Estas estructuras son el log de control, los datos de rollforward y rollback para cada transacción a la que se le haya realizado un commit y cada punto de control para las transacciones.

Cuando se crea una base de datos se crea un dbspace para poder alojar toda la información temporal que se va a crear. Este dbspace es llamado IQ_SYSTEM_TEMP.

Otro dbspace que se utiliza para datos temporales es el IQ_SHARED_TEMP. En esta sección se puede realizar el procesamiento de consultas hechas por diferentes servidores secundarios sin la necesidad de asignarle un espacio fijo a cada uno.

El quinto store es un almacenamiento in-memory que se utiliza para poder realizar actualizaciones a nivel de registro con un gran rendimiento. Este store es conocido como row-level versioning (RLV).

El último dbspace se llama IQ_SYSTEM_MSG y guarda una referencia al archivo de log de la base de datos. No es considerado como un store porque no guarda datos.

Por último existen los user dbspaces en los cuales se almacenan los objetos como datos del usuario final que se asocian a una tabla, índices, join índices y metadata de las tablas, lo más recomendable es crear todos los objetos necesarios en este tipo de dbspace y evitar realizarlo en el IQ_SYSTEM_MAIN.

Se pueden realizar varias acciones sobre los dbspaces. El primer dbspace de cada store se crea automáticamente cuando se crea la base de datos. Después de ese momento se pueden crear más dbspaces de ser necesario. Estos dbspaces pueden ser cambiados de tamaño dependiendo de las necesidades que se tengan.

Se recomienda crear todos los dbspaces que se necesitarán desde un principio en lugar de crearlos gradualmente (Sybase, s/f). Cuando se crea o carga una tabla, los datos se distribuyen entre todos los dbspaces existentes que tengan stores con espacio disponible. Así mismo, cuando se crea un dbspace se puede reservar espacio para que después se pueda expandir si es necesario.

De la misma manera que en el RDBMS, el CDBMS nos permite definir un tamaño para las páginas donde se almacenarán los datos. Las opciones recomendadas por el manual (SAP, s.f) son 64KB, 128KB, 256KB y 512 KB. En la siguiente tabla se explican las diferencias entre las opciones.

Tamaño de Página	Características	Número de registros en la tabla de mayor tamaño	Espacio en Disco
64 KB	Es el tamaño mínimo para una nueva base de datos. Tiene el mejor rendimiento en plataformas de 32-bit	Máximo mil millones de registros	Máximo 8TB de tamaño
128 KB	Default para plataformas de 64/bit	De 1 a 4 mil millones de registros	Mayor a 8TB
256 KB		Más de 4 mil millones de registros	Mayor a 8TB

Img 28. Tabla con las características de los tamaños de página

Para determinar el tamaño total que debe tener una base de datos del CDBMS se deben tomar en cuenta estimación de la cantidad de registros final generada por los usuarios y el número de índices que se crearán para poder manejar esos datos. En caso de que el espacio disponible en la base de datos se agote cuando se está realizando una carga o una inserción de datos, el CDBMS realizará un rollback de toda la transacción o

un rollback hasta un punto de salvado para garantizar la integridad de los datos (SAP, 2013).

Hay tres factores que nos ayudan a determinar el tamaño que deberá tener IQ_SYSTEM_MAIN:

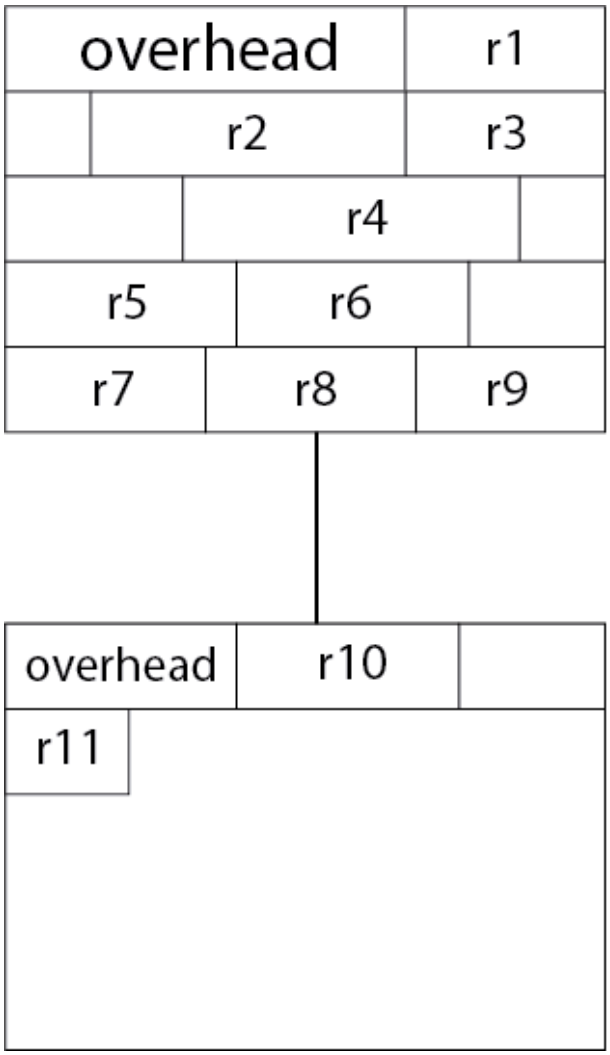
- El número de versiones que se mantendrán almacenadas. Este tema se analizará más adelante.
- La naturaleza de la información y los índices.
- La capacidad que tendrá la base de datos para poder realizar cargas de datos en cualquier momento. (SAP, 2013)

Para poder explicar estas diferencias supongamos que tenemos la siguiente tabla Empleado:

Nombre	Apellido Paterno	Apellido Materno	Salario	Comisión
Emiliano	Bravo	Dominguez	15000	10
Juan	Gúzman	Gómez	17000	20
María	Hernández	Silvia	12000	5

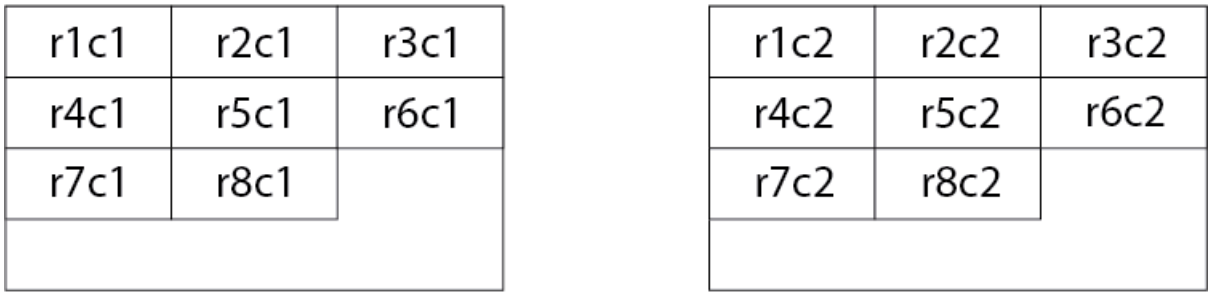
Img 29. Tabla empleados

Una base de datos de almacenamiento por renglón almacenaría la información de la siguiente manera.



Img 30. Páginas en un almacenamiento por renglón

Mientras que una base de datos de almacenamiento columnar lo haría como se indica en la siguiente imagen.



Img 31. Páginas en un almacenamiento columnar

Observando los ejemplos anteriores, suponiendo que se realizará una consulta sobre una sola columna, en el almacenamiento columnar obtendríamos todos los datos haciendo menos acceso a disco que en el almacenamiento por renglón. Sin embargo, si se necesitara recuperar la información de un registro completo tendríamos que hacer más accesos a disco en un almacenamiento por renglón.

➢ Comparación de operaciones de almacenamiento

Comparemos los resultados que se obtienen al realizar las inserciones de datos en el RDBMS y el CDBMS.

En el RDBMS el método de inserción que se utilizó es sqlldr, en el cual utilizamos un archivo .ctl para ingresar los datos. En el CDBMS insertamos los datos mediante el comando LOAD TABLE. Ambas operaciones son los métodos de carga de datos más rápidos para los dos manejadores.

En la siguiente tabla comparamos los tiempos obtenidos.

Tabla	Número de registros	Tiempo en RDBMS [s]	Tiempo en CDBMS usando LOAD TABLE [s]	Veces más rápido CDBMS vs. RDBMS
REGION	5	0.19	0.04	4

NATION	25	0.48	0.038	13
SUPPLIER	10,000	7.81	0.125	61
PARTSUPP	800,000	512	62.057	7
PART	200,000	132	1.094	131
CUSTOMER	150,000	85	1.091	84
ORDERS	1,500,000	939	115.64	7
LINEITEM	6,001,215	4140	21.59	196
TOTAL	**7,502,405**	**5,816.48**	**201.675**	**29**

Img 32. Tabla con la comparación de tiempos de llenado

Haciendo la comparación de tiempos de carga totales se puede observar lo siguiente:

Tiempos de Carga Totales
RDBMS: 1 hora 36 minutos 56 segundos
CDBMS: 3 minutos 21 segundos
El CDBMS es **29 veces más rápido en la carga de datos** que el RDBMS

Img 33. Tabla con la comparación de tiempos totales de llenado

Ahora analicemos otro ejemplo, la inserción con el método IMPORT en dos tablas, este método del CDBMS nos permite importar datos desde archivos planos a las tablas de la base de datos, cuando se insertan los datos a las tablas lo que realmente se realiza es construir nuevos índices en las columnas correspondiente por cada nuevo registro (Referencia). Debido a que en la mayoría de las tablas el tiempo de inserción supera más de un día solo se realizaron dos operaciones de IMPORT (SAP, s.f).

El tiempo de aplicar el método IMPORT sobre la tabla CUSTOMER con un total de 150000 registros fue aproximadamente de más de 7 horas.

Img 34. Imagen con el resultado del import sobre la tabla Customer

El tiempo de aplicar el método IMPORT sobre la tabla SUPPLIER con un total de 10000 registros fue de 10 minutos con 10 segundos.

Img 35. Imagen con el resultado del import sobre la tabla Supplier

Comparando los tiempos podemos observar que la instrucción que toma más tiempo en cargar los datos es IMPORT en el CDBMS. Después está sqlldr en el RDBMS y, finalmente, el mejor desempeño lo tiene LOAD TABLE del CDBMS.

LOAD TABLE tiene el mejor desempeño debido a que solo inserta los datos sin generar ninguna transacción, a diferencia de IMPORT que realiza una transacción por cada inserción que hace. Más adelante detallaremos el tema de la administración de transacciones y observaremos el proceso que se realiza durante una transacción. Aunado a eso, en el almacenamiento por columnas, una página solo guarda datos de una sola columna de diferentes registros. Para almacenar un solo registro se debe de ocupar varias páginas para ir guardando cada columna del registro. Mientras que en el

RDBMS, insertar un registro es más rápido debido a que solo se realiza una inserción en una página.

Por último, el CDBMS con LOAD TABLE tiene un mejor desempeño en la carga masiva de datos comparado con sqlldr del RDBMS.

➢ Comparación del espacio de almacenamiento

En las bases de datos columnares se comprimen los datos por default a diferencia de los modelos relacionales en los cuales se requiere de una configuración especial para poder realizar esto.

En la siguiente tabla comparamos el espacio en KB y el número de bloques utilizadas obtenidos y el porcentaje de diferencia obtenido.

Tabla	Número de registros	Espacio utilizado en CDBMS [KB]	Espacio utilizado en RDBMS [KB]	Número de bloques utilizados CDBMS	Número de bloques utilizados RDBMS	Ahorro de Espacio [KB]
REGION	5	1,072	40	134	5	1,032
NATION	25	1,688	40	211	5	1,648
SUPPLIER	10000	3,616	1,952	452	244	1,664
PARTSUPP	800000	65,336	137,896	8,167	17,237	72,560
PART	200000	13,192	27,152	1,649	3,394	13,960
CUSTOMER	150000	22,104	27,152	2,763	3,394	5,048
ORDERS	1500000	74,792	186,856	9,349	23,357	112,064
LINEITEM	6001215	188,416	807,016	23,552	100,877	618,600
TOTAL	**7,502,405**	**370,216**	**1,188,104**	**46,277**	**148,513**	

Img 36. Tabla con la comparación de espacio ocupado

Espacio Ocupado
RDBMS: 1,160.25 MB
CDBMS: 361.53 MB
El CDBMS ocupa el **32% de espacio** de lo que ocupa el RDBMS El **ahorro de espacio** en el CDBMS es de **798.72 MB**

Img 37. Tabla con la comparación de espacio ocupado total

Podemos observar varias características de esta tabla comparativa, las tablas con menor número de registros ocupan más espacio en disco y un mayor número de bloques, esto se debe a que el tamaño de página tiene un mayor peso. Por otro lado cuando se almacenan un mayor número de datos la compresión es más eficiente y se ocupa menos espacio y un menor número de bloques.

Podemos concluir que las bases de datos columnares funcionan mucho mejor cuando se tienen grandes volúmenes de datos, que cuando se tienen catálogos con pocos registros.

3.3. ALGORITMOS DE BÚSQUEDA

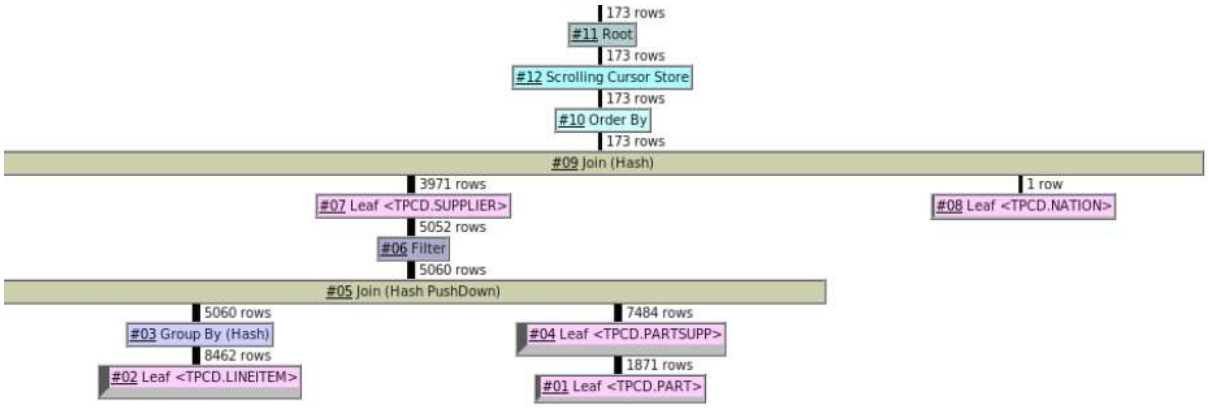
Recordemos que las bases de datos columnares se utilizan en ambientes de análisis por lo cual las operaciones no requieren que se recupere todas las columnas de los registros y además se requieren realizar operaciones sobre los datos recuperados.

Ya que se tienen los datos almacenados en forma columnar al realizar las operaciones analíticas sólo se necesitan leer las páginas de disco que contienen esas columnas, a diferencia de las bases de datos relacionales que se requeriría la lectura de muchas más páginas. Además podemos aprovechar los tamaños de página y la compresión de datos que nos proporcionan estas bases de datos para obtener muchos

más datos sin realizar tantas lecturas de disco. Al realizar una menor cantidad de lecturas al disco ya implica una mejoría en el rendimiento de las consultas de tipo analíticas.

3.3.1. LECTURA DE DATOS

Para realizar una consulta eficientemente el CDBMS construye el query plan con las operaciones que conforman a la consulta, como son los joins, group by, subqueries y condiciones.



Img 38. Query plan en el CDBMS.

En la parte más baja lo conforman las hojas, estas son las tablas que se requieren utilizar y a las que se les aplican las operaciones condicionales '*WHERE'*, una ventaja que presenta el CDBMS es que estos datos se pueden procesar de manera vertical, es decir que las operaciones de proyección sobre las tablas son accedidas mucho más rápido debido al número de índices que se tienen en las columnas. Luego se empieza a reducir el número de registros con las siguientes operaciones (SAP, 2012).

La diferencia más importante entre los modelos relacionales y columnares es la forma en la que se seleccionan los índices que se utilizarán para obtener la información. Una base de datos columnar manejan varios índices en casi todas las columnas, esto permite seleccionar el mejor índice para realizar la consulta. En el caso del manejador que estamos utilizando se crea un índice automáticamente por cada columna, es decir,

el mínimo de índices que puede tener una columna es 1. El índice creado se puede cambiar por otro dependiendo de las necesidades del sistema.

En las bases de datos relacionales, una columna solo puede tener un índice asociado, lo cual limita los distintos tipos de accesos que se le podría dar a las tablas. Sin embargo, al momento de generar varias transacciones que alteren los datos no perderá tanto tiempo actualizando todos los índices.

Para saber qué camino debe tomar el manejador para obtener el mejor rendimiento, se basa en las estadísticas que obtienen de las tablas de sistema, esta información es por ejemplo, el número de registros que tiene la tabla que se requiere consultar, el número de registros que se esperan obtener después de completar la consulta, el espacio que se ocupa en memoria para realizar la consulta, la concurrencia del dato que se requiere o el número de usuarios que requieren el dato, los tiempos y números de procesadores disponibles y el tiempo que se toma en recuperar la información (SAP, 2012).

- ➢ Selectividad

La selectividad es la porción de registros que cumplen con una condición específica.

Por ejemplo, si al realizar una consulta de una tabla con 10,000 registros y en la condición de la búsqueda se conoce que 2,500 cumplen con la condición, entonces la selectividad sería la siguiente:

$$selectividad = \frac{número\ de\ registros\ encontrados}{número\ total\ de\ registros}; \qquad \frac{2500}{10000} = 0.25$$

En las bases de datos columnares conocer el número de registros que cumplen con la restricción se hace mediante los índices que se tienen en cada columna. En el CDBMS se utilizan los índices High Group (HG), Low Fast (LF) y Fast Projection (FP) para obtener la cuenta de los registros que cumplen con la condición. Cuando no se tienen la información exacta por parte de los índices, se utilizan estimados dependiendo de la operación que se va a realizar.

Tipo de operación	Selectividad
Igualdad (=)	0.2
Rango (>,>=,<,<=)	0.4
BETWEEN	0.4
Like (%)	0.2
Igualdad entre columnas	0.3
Rango entre columnas	0.5

Img 39. Tabla con los valores de selectividad

En el RDBMS se utiliza las estadísticas para calcular la selectividad, o hace uso de los histogramas, el cual cuenta con la información de la distribución de los datos. Pero algunas veces esta información no es exacta por la falta de actualización del histograma. En caso de no contar con estadísticas se utilizan las llamadas estadísticas dinámicas, las cuales son sentencias que escanea una pequeña parte de la tabla para obtener un estimado de los números, o en otros casos se utilizan valores por default (Oracle, s.f).

La ventaja en las bases columnares son los diferentes índices que tienen, ya que así se puede recuperar los datos estadísticos de manera más rápida, además realizar los conteos para las estadísticas son más rápidas por la compresión y una recuperación de valores más rápida por cómo se almacenan los valores

- Agrupamiento

Las funciones de agrupamiento se encargan de agrupar los resultados obtenidos en una sola columna y en un solo registro, para realizar una consulta eficiente se consideran dos elementos: el número de registros que se va a agrupar y el algoritmo que se utilizará para realizar la agrupación.

Ambos manejadores cuentan con dos tipos de algoritmos para realizar el agrupamiento:

Group By(Hash): en este algoritmo se crea una tabla hash en la memoria principal de los grupos que se generan. Esto se realiza al momento en que se va leyendo cada registro y se van creando los valores hash con el parámetro indicado, después se actualiza la tabla con el valor hash creado.

Este tipo de agrupamiento es más eficiente si son pocos registros a agrupar, ya que si existen más registros que se puedan guardar en una tabla, entonces se particiona la tabla en pedazos más pequeños y se procesan de manera recursiva, lo que genera más tiempo de procesamiento.

Group By(Sort): en este algoritmo se va ordenando la tabla de resultados a partir del parámetro por el que se va a agrupar. Después se van haciendo operaciones sobre la tabla ya ordenada.

Este método es más utilizado si se tienen muchos registros como resultado, ya que comparado con el método hash, el ordenamiento se convierte en un método más rápido.

Aunque se tengan los mismos algoritmos se tienen algunas ventajas en el agrupamiento en las bases de datos columnares, una de ellas es debido a que como todos los valores que son parte de la agrupación están en el mismo bloque de datos, por lo que se realizan menos llamadas al disco. Otra ventaja que se puede obtener es por la compresión de los datos, ya que se tienen los datos representados por un valor delta de tipo entero, realizar operaciones sobre estos valores es computacionalmente menos costoso, es decir utiliza menos recursos de procesador al operar con valores enteros que con el valor original. Además generalmente estos valores ya se encuentran ordenados por lo que se ahorra tiempo de ordenamiento.

➢ Proyección

Esta operación es la selección de una o varias columnas en particular, en las base de datos relacionales, para realizar esto se genera un subconjunto de los registros

obtenidos en los que solo se agregan los atributos o campos deseados. Como podemos observar el recuperar primero todos los campos puede resultar menos eficiente si se solicitan pocos campos. En las bases de datos columnares la recuperación por proyección es más eficiente porque se recuperan muchos menos bloques de página (Harrison, Guy, 2016).

En el caso de que se requieran recuperar numerosos campos las bases de datos columnares tratan de acomodar los bloques de datos de manera contigua tomando en cuenta la frecuencia en la que se recuperan las combinaciones de los campos, así las proyecciones tienen un mejor rendimiento ya que no se tiene que realizar muchas búsquedas en disco.

Una ventaja que proporcionan algunos manejadores es que obtiene datos ordenados previamente. Otra es pre cargar los join, en la cual se crean columnas con los criterios de joins de múltiples tablas, esto asemeja a crear vistas materializadas para ciertas operaciones de joins específicos.

3.3.2. TIPOS DE JOIN

Las operaciones de join que se realizan entre dos tablas se determinan a partir de la información que se tiene de las dos tablas a las que se realizará la operación, los parámetros que se obtienen para determinarlo son por ejemplo, el número de registros que se encuentran tanto en la tabla izquierda como en la derecha, la estimación del número registros como resultado de la operación, el tipo de operador de comparación entre una tabla y otra, los tipos de datos de las columnas que se están comparando y el tipo de relación que cuenta con la otra tabla ya sea Uno a Muchos, Uno a Uno o Muchos a Muchos y validar los posibles algoritmos que puede utilizar.

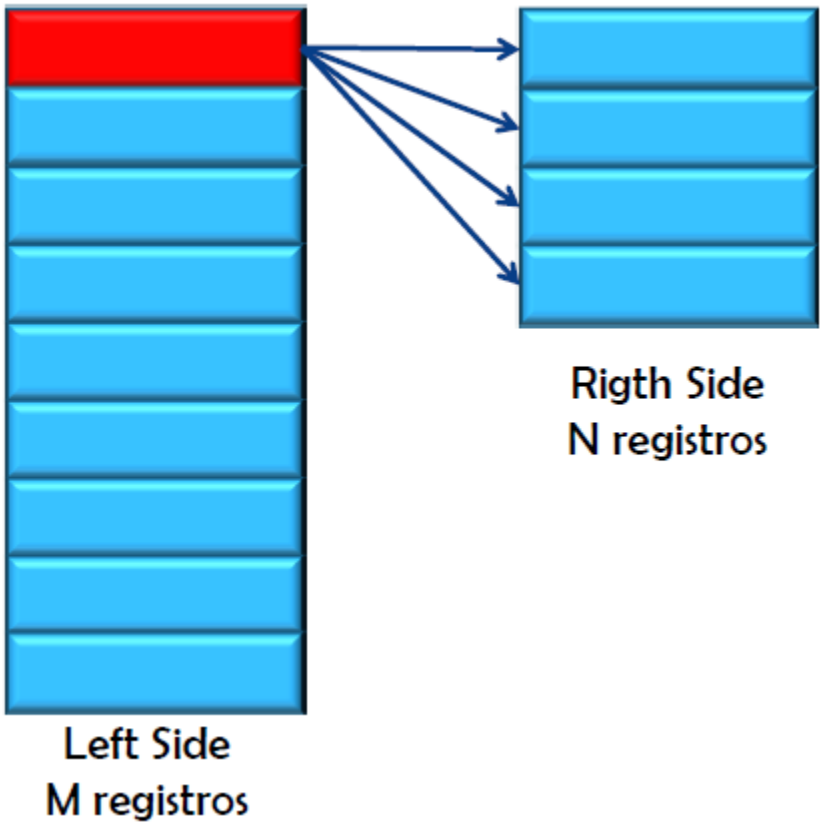
El CDBMS maneja tres tipos de Join el Nested Loop Join, Hash Join y Sort Merge Join (SAP Sybase Blo, s.f).

- Nested Loop Join

Este método realiza una búsqueda completa o barrido sobre la tabla derecha por cada registro de la tabla izquierda. En este tipo de join no importa cómo se encuentren ordenadas las dos tablas.

Generalmente se escoge utilizar este tipo de join cuando no se tienen condiciones de igualdad en la búsqueda o para obtener el primer resultado.

Debido al número de comparaciones que se realizan y el número de lecturas a disco que se realizan, este método se considera como el menos eficiente.



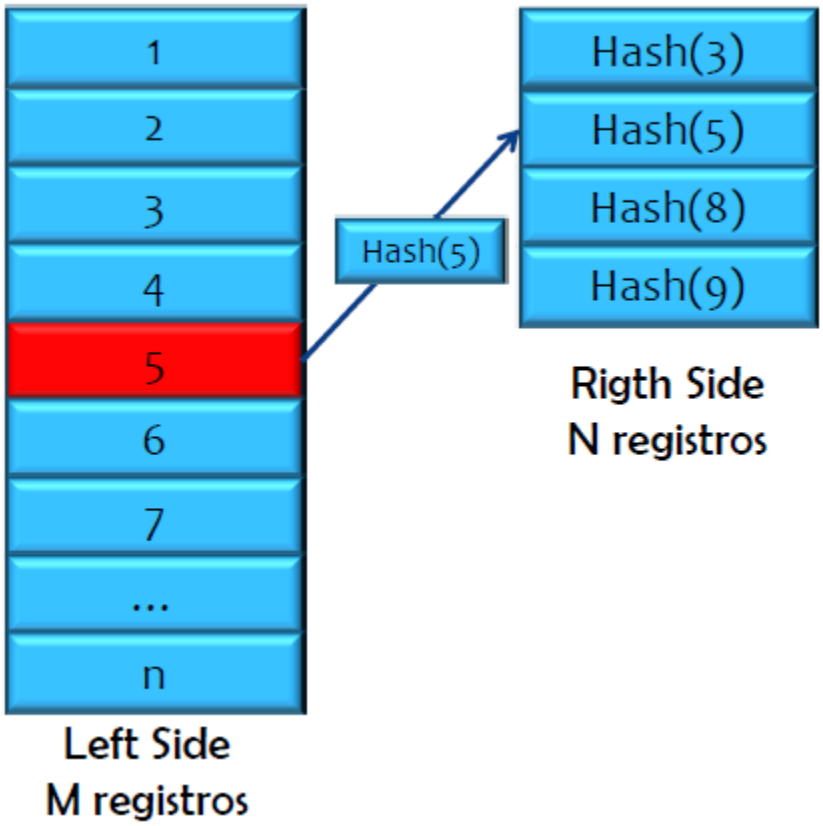
Img 40.Imagen recuperada de la clase Bases de Datos Avanzadas del profesor Bautista, Héctor

- Hash Join

El Hash Join construye y almacena una hash table de la tabla más pequeña que se tiene de entrada. Y luego lee la tabla más grande y realiza las operaciones hash en la otra tabla para encontrar las coincidencias. Si la tabla más pequeña es muy grande para

almacenarla en memoria principal se particionan ambas tablas hasta que las particiones tienen el espacio adecuado.

Este algoritmo resulta con el mejor rendimiento, siempre que la tabla más pequeña tenga el espacio adecuado en memoria. Y generalmente se utiliza este método cuando la diferencia entre las tablas es muy grande, para poder almacenar la tabla más pequeña en memoria principal y evitar hacer la partición, el máximo número de operaciones que se realizan son de M+N.



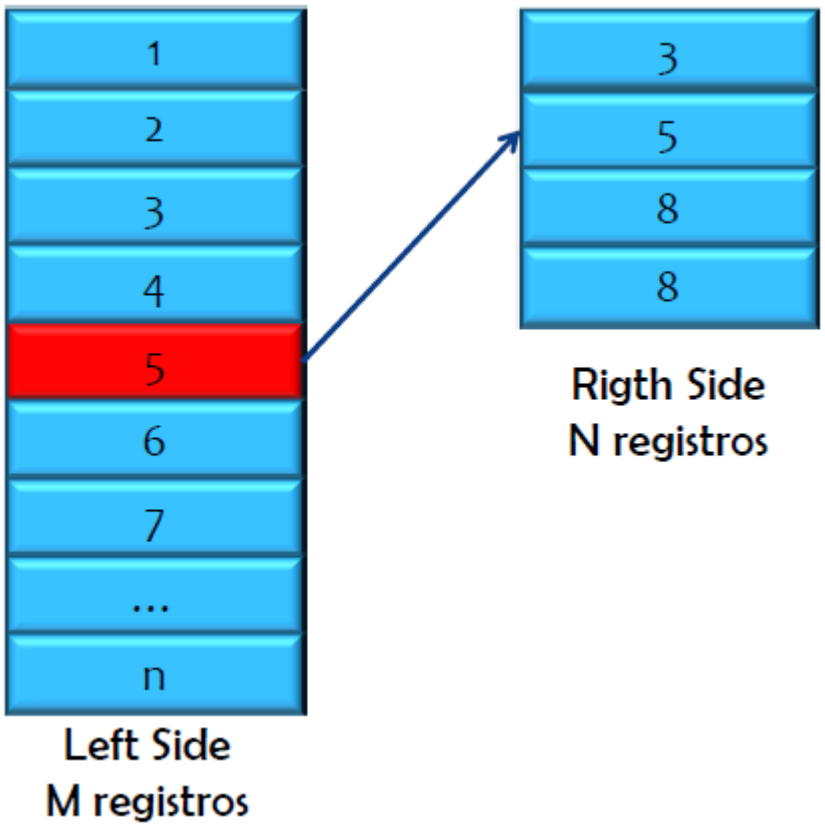
Img 41.Imagen recuperada de la clase Bases de Datos Avanzadas del profesor Bautista, Héctor

- Sort Merge Join

Cuando se utiliza el algoritmo Sort Merge Join, se lee las dos tablas de entrada y se ordenan por los campos de join que se especificaron, una vez que se tienen la lista ordenada se empieza a comparar los registros que coinciden. El manejador decide

utilizar este método si dentro de la operación de la consulta se realiza una operación de ordenado, ya que utiliza el ordenamiento previo y solo compara los datos.

La ventaja de usar este método es que puede ser más rápida si se utiliza varias veces y los manejadores generalmente deciden aplicarlo si las dos tablas contienen cantidades parecidas de registros.



Img 42.Imagen recuperada de la clase Bases de Datos Avanzadas del profesor Bautista, Héctor

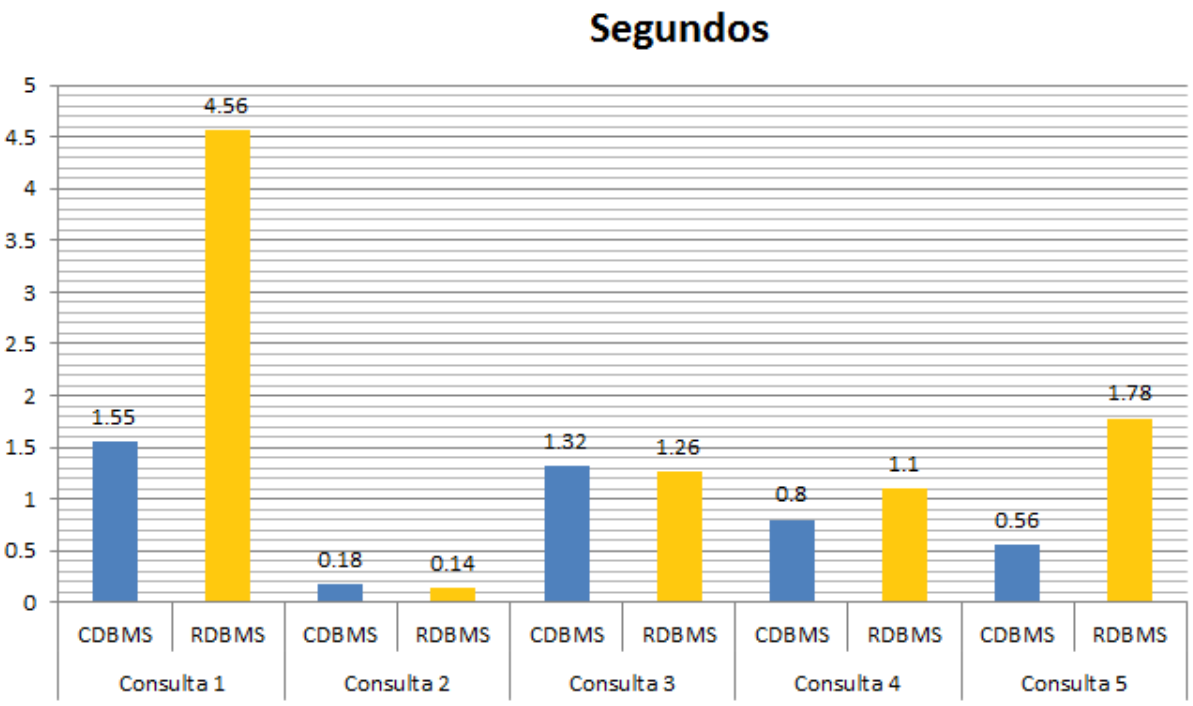
3.3.3. COMPARACIÓN DE RESULTADOS

Las consultas que se realizan aquí son parte del Benchmark de TPC-H, son un total de 22 consultas, este conjunto de consultas están diseñadas para observar el rendimiento de las bases de datos. Estas consultas se les ha dado un contexto realista del análisis empresarial de una compañía de ventas en mayoreo.

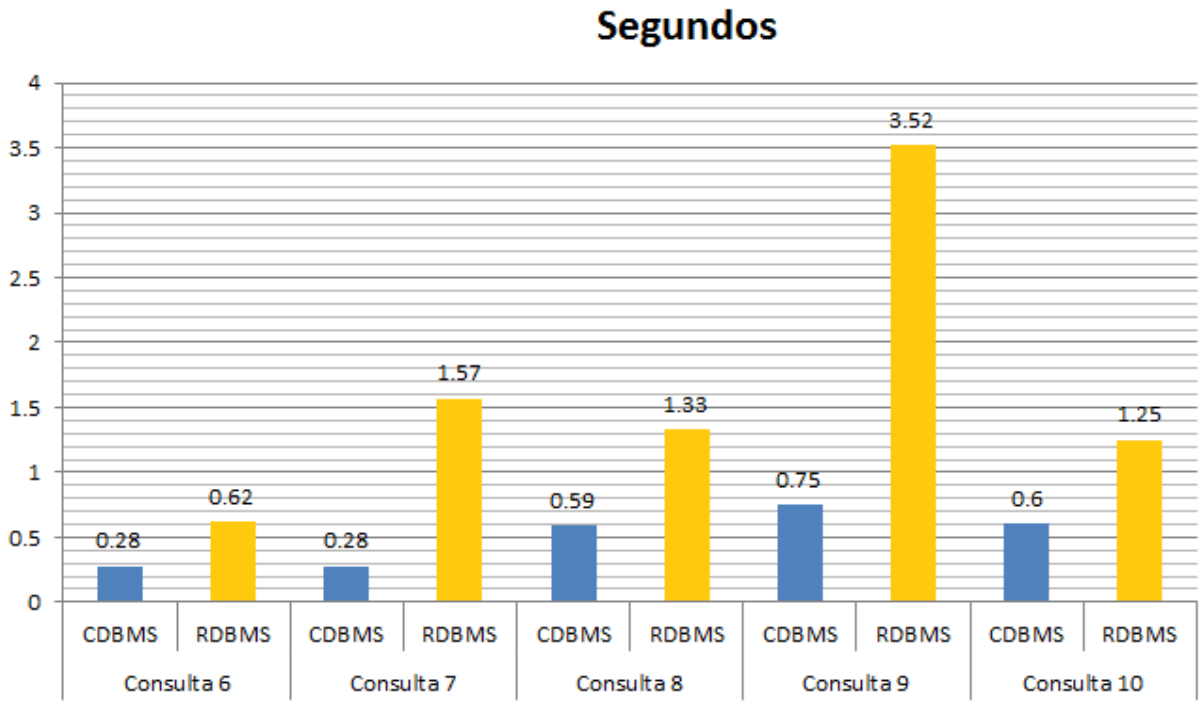
Consulta	Tiempo en CDBMS [s]	Tiempo en RDBMS [s]	Veces Más Rápido [s]
1	1.55	4.56	2
2	0.18	0.14	0.28
3	1.32	1.26	0.04
4	0.8	1.10	0.38
5	0.56	1.78	2.17
6	0.28	0.62	1.21
7	0.38	1.57	2,13
8	0.59	1.33	1.25
9	0.75	3.52	3.69
10	0.60	1.25	1.08
11	0.12	0.67	4.58
12	0.25	1.34	4.36
13	1.27	1.49	0.17
14	0..20	0.84	3.2
15	0.50	0.9	0.8
16	0.39	0.29	0.34
17	0.29	0.94	2.24

18	1.78	3.44	0.93
19	0.11	0.95	7.63
20	0.18	0.91	4.05
21	1.62	10.88	5.72
22	0.33	0.32	0.03
TOTAL	**13.85**	**40.1**	

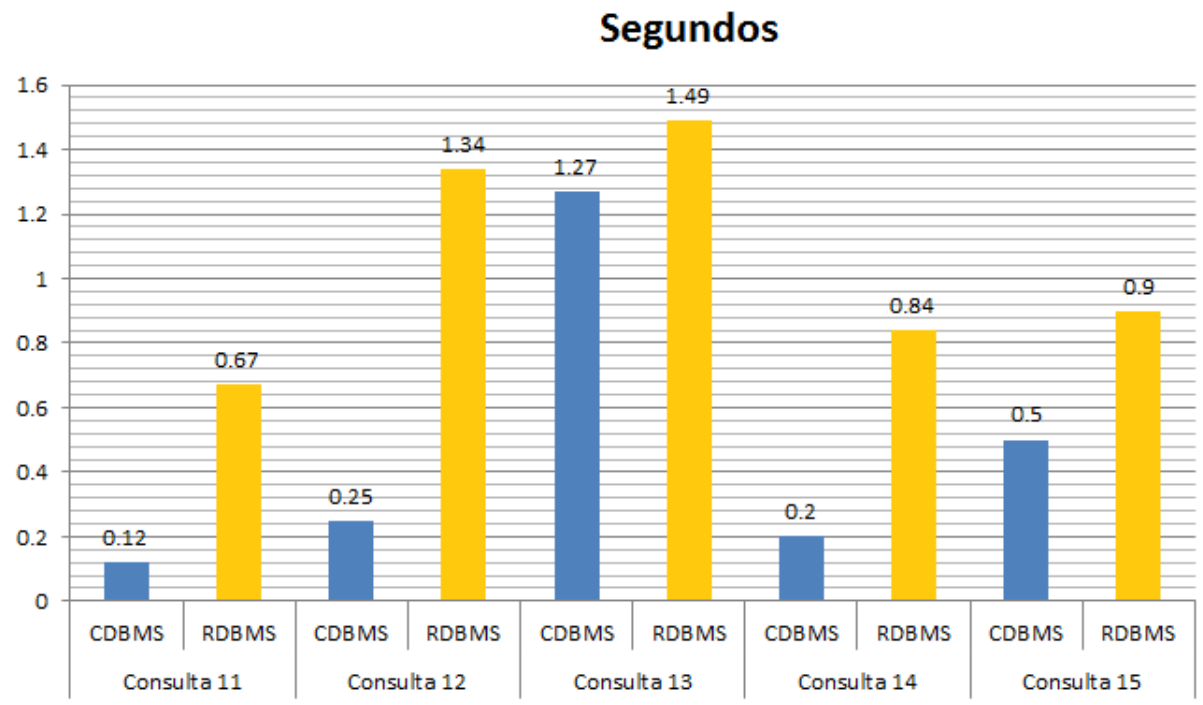
Img 43. Tabla comparativa de los tiempos de ejecución



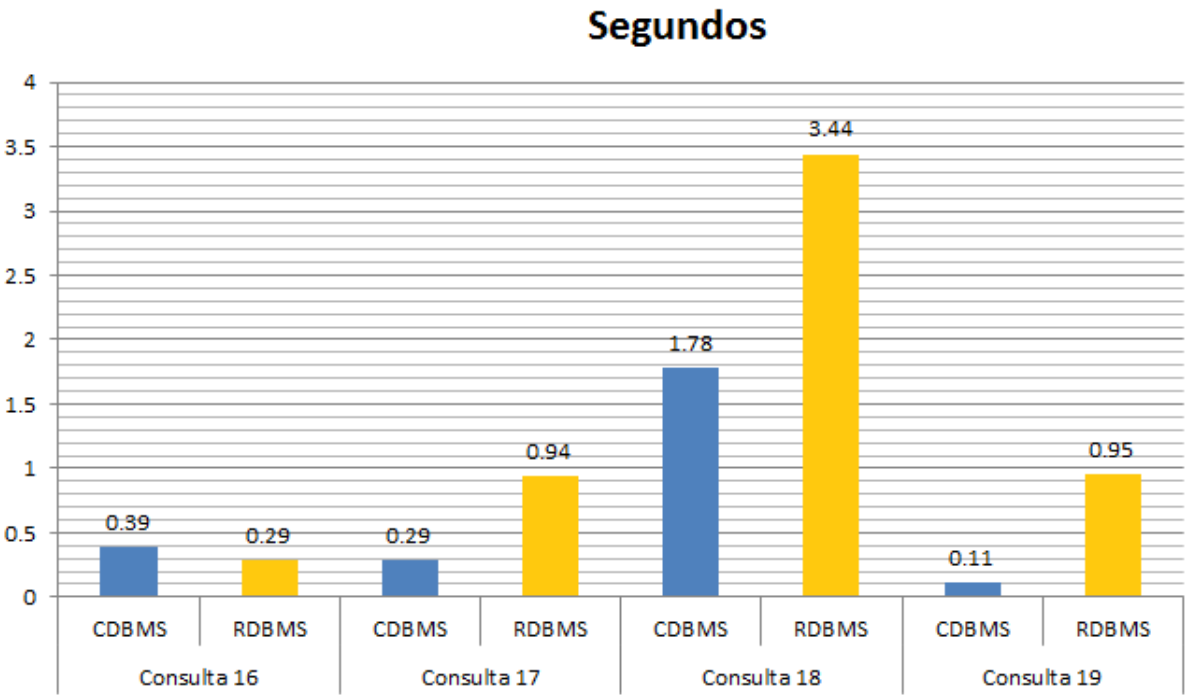
Img 44. Gráfica comparativa de la consulta 1 a la consulta 5



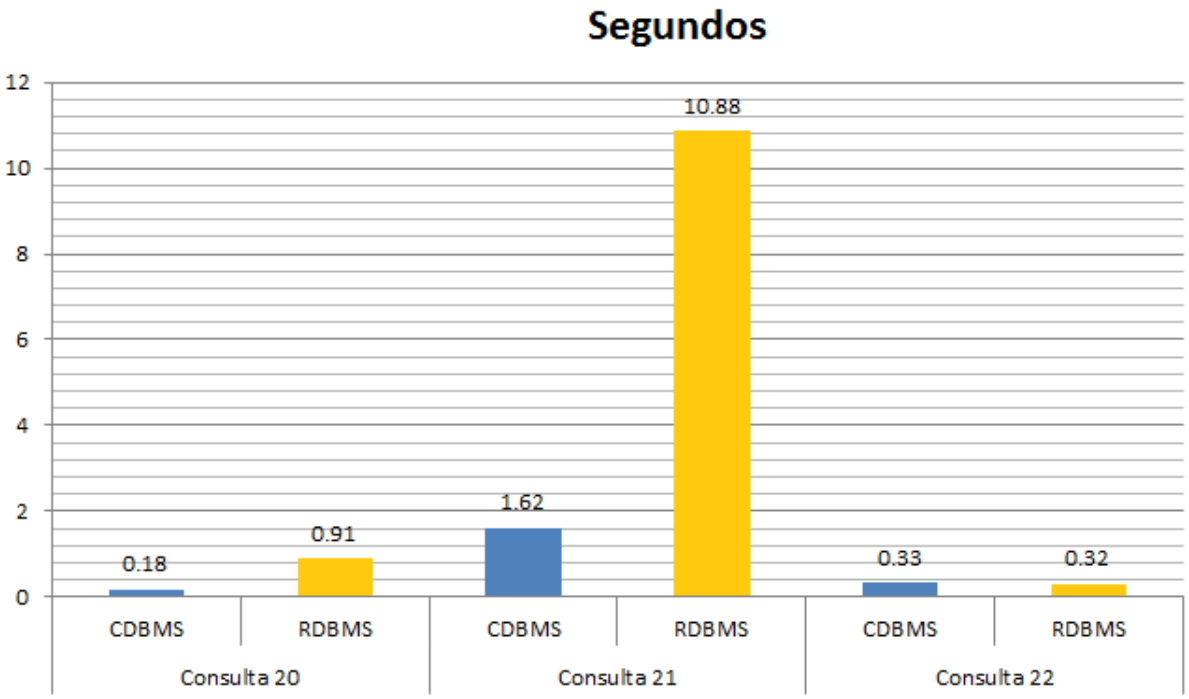
Img 45. Gráfica comparativa de la consulta 6 a la consulta 10



Img 46. Gráfica comparativa de la consulta 11 a la consulta 15



Img 47. Gráfica comparativa de la consulta 16 a la consulta 19



Img 48. Gráfica comparativa de la consulta 20 a la consulta 22

Tiempo Total de Ejecución de Consultas
RDBMS: 40.1 s
CDBMS: 13.85 s
El CDBMS ejecutó **1.78 veces más rápido** las consultas que el RDBMS

Img 49. Gráfica comparativa de los tiempos totales de ejecución de consultas

Comparando los resultados podemos observar que el CDBMS tiene un mejor desempeño al momento de ejecutar las consultas. Aunque hay algunas consultas donde la diferencia de tiempos es mínima, en la consulta 21 podemos observar una diferencia de casi 9 segundos. Esto se debe a la cantidad de datos que regresó la base de datos. Entre más datos abarque la consulta más marcada es la diferencia.

3.4. TRANSACCIONES

Los dos manejadores administran las transacciones de manera diferente, sin embargo, se basan en la misma teoría. Lo primero es hablar sobre el manejo de los bloqueos de las transacciones.

Los bloqueos son mecanismos de control de concurrencia, que aseguran la consistencia de datos dentro de las transacciones. En ambientes multiusuario los bloqueos son necesarios para evitar que varios usuarios trabajen en los mismos datos al mismo tiempo.

Un punto importante de los bloqueos es su granularidad, es decir, que tantos datos se bloquean al mismo tiempo. El CDBMS maneja una granularidad alta en la información

La granularidad del RDBMS es la siguiente:

- A nivel de registro. Bloquea los registros que se están utilizando. Este es el nivel de granularidad más grande.
- A nivel de página. Bloquea las páginas que están utilizando
- A nivel de extensión. Bloquea la extensión completo de los registros que se están utilizando.
- A nivel de tabla. Bloquea toda la tabla sin importar cuántos registros se están utilizando en la transacción.
- A nivel de base de datos. Es el nivel más grande de bloqueo y el nivel más bajo de granularidad. Bloquea la base de datos por completo.

La granularidad puede ser elegida dependiendo de las necesidades del sistema. Entre menor sea el nivel de granularidad menor será la concurrencia que se tenga en las transacciones, por lo tanto habrá un menor desempeño. Por otro lado, a mayor nivel de granularidad mayor concurrencia.

Así mismo, los manejadores tienen diferentes modos de bloqueo predefinidos dependiendo de las operaciones que se realicen. Los modos de bloqueo del RDBMS son los siguientes:

- “Compartido: para operaciones sólo de lectura. Se permiten lecturas concurrentes, pero ninguna actualización.
- Actualización: para operaciones que pueden escribir. Sólo se permite que una transacción adquiera este bloqueo. Si la transacción modifica datos, se convierte en exclusivo, en caso contrario en compartido.
- Exclusivo. para operaciones que escriben datos. Sólo se permite que una transacción adquiera este bloqueo.
- Intención: se usan para establecer una jerarquía de bloqueo. Por ejemplo, si una transacción necesita bloqueo exclusivo y varias transacciones tienen bloqueo de intención, no se concede el exclusivo.
    - Intención compartida. Bloqueo compartido.
    - Intención exclusiva. Bloqueo exclusivo.

- Compartido con intención exclusivo. Algunos bloqueos compartidos y otros exclusivos.” (Pérez, Fernando, 2010)

El CDBMS maneja los bloqueos de la siguiente manera:

- Las lecturas no bloquean las escrituras. (SAP, 2013, p.233)
- Las escrituras no bloquean las lecturas. (SAP, 2013, p.233) Sin embargo, las lecturas verán una versión anterior de los datos.
- Un usuario que realice una actualización y varios usuarios que realicen lecturas pueden acceder a la misma tabla concurrentemente. (SAP, 2013, p.233)
- Únicamente un usuario puede actualizar una tabla al mismo tiempo, evitando que se realicen actualizaciones concurrentes sobre la misma tabla. (SAP, 2013, p.233)

Después de analizar los bloqueos de transacción hablaremos sobre los niveles de aislamiento, que especifican hasta qué punto una transacción está aislada de otras transacciones.

ANSI define 4 niveles de aislamiento (SAP, 2013).

- Nivel 0. Lecturas no confirmadas (Read Uncommited)

Se utiliza en ambientes que manejan muchas transacciones al mismo tiempo debido a que tiene un muy buen desempeño. Se basa en poder leer datos que han sido modificados pero aún no han sido confirmados por otras transacciones, por lo que puede traer problemas de inconsistencia en los datos si se ejecuta algún rollback. A lo anterior se le llama “Lecturas sucias”.

Tiene una mayor concurrencia a cambio de una menor consistencia.

- Nivel 1. Lecturas confirmadas (Read Commited)

Este nivel garantiza que cualquier dato que se consulte sea un dato confirmado por las otras transacciones, por lo que no permite “Lecturas sucias”. Sin embargo, no garantiza que si se realizan dos lecturas del mismo dato se obtenga el mismo resultado

debido a que permite que se realicen lecturas entre dos transacciones. A estas lecturas se les llama “Lecturas no repetibles”.

- Nivel 2. Lecturas estables (Repeatable Reads)

Con este bloqueo se garantiza que entre dos lecturas de la misma transacción siempre se obtenga el mismo resultado, por lo que no permite “Lecturas no repetibles”. Esto lo logra aplicando bloqueos compartidos para que otras transacciones no puedan modificar los datos que está utilizando una transacción.

La desventaja es que el bloqueo solo se aplica sobre los datos que se están utilizando. Para explicar esto supongamos una columna con valores numéricos y una consulta que busca los registros que tengan un valor menor a 100 pero mayor a 50. Si la consulta regresa 50 registros, se aplicará un bloqueo a estos 50 registros para que en posteriores consultas estos datos no sufran modificaciones. Pero el bloqueo no nos garantiza que se inserten nuevos datos que cumplan con esa condición, o que se modifiquen los datos que no entraban en ese rango logrando que en la siguiente consulta se regresen más de 50 registros. Estas consultas se llaman “Phantom Reads”.

- Nivel 3. Lecturas repetibles (Serializable)

Es el nivel más alto de aislamiento. Genera bloqueos a toda la tabla para garantizar que en una transacción siempre se regresen los mismos datos. Es el que tiene la mayor consistencia, aunque sacrifica el rendimiento debido a que otros usuarios no pueden realizar modificaciones hasta que las otras transacciones terminen de utilizar sus tablas.

Para empezar la comparación entre los manejadores hablaremos sobre qué niveles de aislamiento utiliza cada uno de ellos, cuál es su nivel de aislamiento por defecto y qué buscan utilizando dicho nivel.

El RDBMS que utilizamos maneja los siguientes niveles de aislamiento.

- Read commited

Encuentra su equivalencia con el nivel 1 antes mencionado.

- Serializable

Encuentra su equivalencia con el nivel 3 antes mencionado.

- Read only

Este nivel no se encuentra dentro del estándar de ANSI. Es similar a Serializable con la diferencia de que tampoco permite que la transacción que realiza la consulta modifique los datos.

Su nivel por defecto es Read commited aunque se puede elegir cualquier otro dependiendo de las necesidades del sistema. Otros manejadores pueden tener un nivel 0 de aislamiento y pueden tener otros niveles por defecto.

El CDBMS que utilizamos, por su parte, utiliza los niveles que define ANSI aunque por defecto utiliza el nivel 3 de aislamiento debido a que busca tener la mayor consistencia de datos posible. En la versión 16, que es la que estamos utilizando para las pruebas, el manejador permite cambiar el nivel de aislamiento, sin embargo, siempre se ejecutará el nivel 3 (SAP, 2011).

Como podemos observar, ambos manejadores buscan diferentes resultados utilizando diferentes niveles de aislamiento. Mientras que el RDMBS utiliza un nivel 1, el CDBMS utiliza un nivel 3. Eso nos deja que el CDBMS busca tener una buena integridad en los datos que se leen aunque las transacciones vean su desempeño afectado, mientras que el RDBMS le da prioridad al desempeño de las transacciones descuidando un poco la integridad de los datos leídos.

Lo siguiente es hablar acerca de la administración de las transacciones. EL CDBMS utiliza un método llamado *snapshot versioning*. Se basa en darle a cada usuario una copia de sus cambios realizados en los datos, sin que los demás usuarios lo vean, hasta que se realiza un commit y la copia sustituye a la versión anterior de estos datos en la base de datos.

El versionamiento por omisión es un snapshot a nivel de tabla. Este permite a varios usuarios modificar una tabla, con la condición de que no lo pueden hacer simultáneamente, es decir, tienen que hacerlo de uno en uno, recordando el nivel de

aislamiento que maneja el CDBMS. La ventaja de utilizar este mecanismo es que los usuarios no tienen que preocuparse por las sentencias que ejecutan y cuando las ejecutan ya que el CDBMS asegura que ninguna transacción va a interferir con otra transacción (SAP, 2013).

Una de las desventajas de utilizar el snapshot versioning es que utiliza más espacio en disco que otras técnicas. Cada que se realiza una transacción se crea una nueva versión de los datos insertados o modificados. Si la transacción utiliza 5 páginas de datos debido a la distribución de los mismos, entonces se crearán 1 copia de cada página, que deberán ser almacenadas en memoria mientras se ejecuta la transacción.

Al finalizar cada transacción, el CDBMS realiza un commit para confirmar que los cambios se guarden permanentemente en la base de datos. Si por alguna razón una de las instrucciones falla, el CDBMS realizará un rollback hasta algún punto especificado por el usuario o desechará todos los cambios realizados. Esas opciones pueden ser seleccionadas por el usuario para poder tener un control completo sobre las transacciones (SAP, 2013).

En los DBMS se pueden generar marcadores que nos ayudan a dividir grandes transacciones en partes más pequeñas. Estos marcadores se conocen como savepoints y pueden ser declarados en las transacciones. Si un savepoint es declarado se garantiza que, una vez alcanzado dicho savepoint en la transacción, cuando se ejecute un rollback la información editada antes de ese punto se guardará satisfactoriamente en la base de datos. El CDBMS genera una copia de las páginas de datos utilizadas por cada savepoint encontrado, de manera que al realizar un rollback solo es necesario buscar la copia asociada al último savepoint para ejecutar un commit sobre esa copia. Cuando se realiza un commit las copias son liberadas.

El RDBMS utiliza un mecanismo diferente para realizar las transacciones. Un espacio de disco asignada a la base de datos, conocido como el log de transacciones, se encarga de guardar todos los cambios que han sido realizados a la base de datos durante las transacciones. Cuando se ejecuta una modificación, inserción o eliminación

de un dato se crea un registro en el log de transacciones especificando la operación realizada, de tal manera que, cuando se realiza un rollback sobre la transacción, se pueden revisar los movimientos realizados y realizar su opuesto para llegar a una versión estable del sistema.

Cuando se realiza un commit los bloqueos generados en la tabla, recordando el nivel de aislamiento, se liberan para que se puedan realizar más transacciones al mismo segmento de datos.

EL RDBMS también maneja savepoints, solo que sigue la misma estructura usando el log de transacciones. Cuando se realiza un rollback se revisará el log de transacciones ejecutando el inverso de las instrucciones hasta encontrar un savepoint. Todas las instrucciones realizadas antes del savepoint se mantendrán en la base de datos (Oracle, 2011).

Comparando la administración de las transacciones entre ambos manejadores podemos llegar a una conclusión similar a lo visto en los niveles de aislamiento. El CDBMS sacrifica el desempeño en las transacciones al realizar copias enteras de la información para trabajar con ellas y, posteriormente, liberar esa copia al resto de los usuarios, mientras que el RDBMS tiene un mejor desempeño en las transacciones debido a que realiza las operaciones directamente en la información que los demás usuarios pueden ver. Sin embargo, el CDBMS mantiene una mejor consistencia de los datos leídos porque no permite editar una copia que está utilizando otro usuario con otra transacción, a diferencia del RDBMS que maneja la misma versión de datos para todos los usuarios y en, su nivel de aislamiento por defecto, permite ver diferente datos entre sucesivas lecturas. Una solución al problema de consistencia que pueden llegar a tener los RDBMS es usar la sentencia *SELECT… FOR UPDATE* que bloquea los registros leídos para que, al momento de actualizarlos, no puedan ser utilizados en otras lecturas.

Otra ventaja que tiene el CDBMS contra el RDBMS es al momento de realizar un rollback. Como se explicó, el CDBMS crea versiones de los datos al momento de realizar una transacción, por lo que al momento de realizar un rollback solo tiene que desechar la

versión que está utilizando para mantener una versión anterior estable. Por otro lado, el RDBMS tendría que revisar su log de transacciones para realizar el inverso de las operaciones que fueron realizadas durante la transacción, lo que es más tardado que tan solo eliminar una versión.

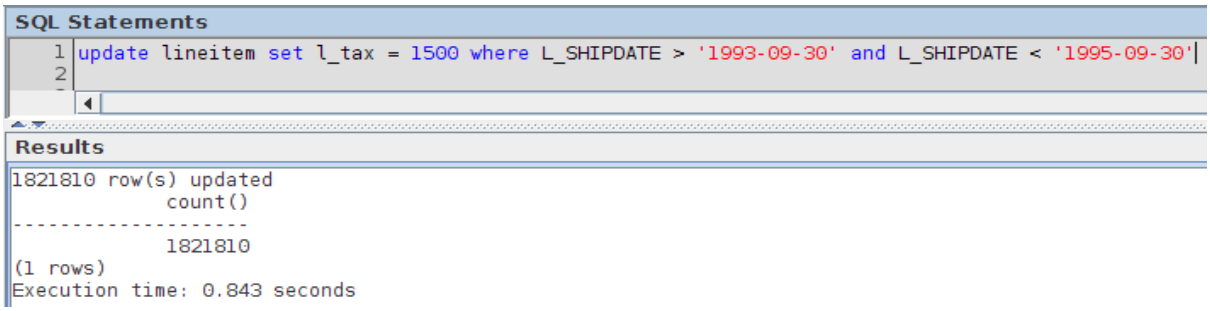
3.4.1. ALGORITMOS DE ACTUALIZACIÓN

Se realizarán dos pruebas utilizando la operación de UPDATE para conocer el comportamiento de ambos manejadores al momento de realizar transacciones con dicha operación.

En la primera prueba se realizarán varios UPDATE, donde cada UPDATE afectará a un conjunto de registros dentro de la misma transacción. En la segunda prueba se realizarán varios UPDATE, donde cada uno afectará a un solo registro, generando múltiples transacciones en una misma prueba.

Resultado de realizar operaciones UPDATE en la tabla LINEITEM, y modificar el valor de L_TAX a un valor de 1500.

Resultado en el CDBMS



Img 50.Primer update CDBMS

Resultado en el RDBMS

```
SQL> update lineitem set l_tax = 1500 where L_SHIPDATE > '1993-09-30' and L_SHIP
DATE < '1995-09-30';

1821810 filas actualizadas.

Transcurrido: 00:17:06.49
SQL>
```

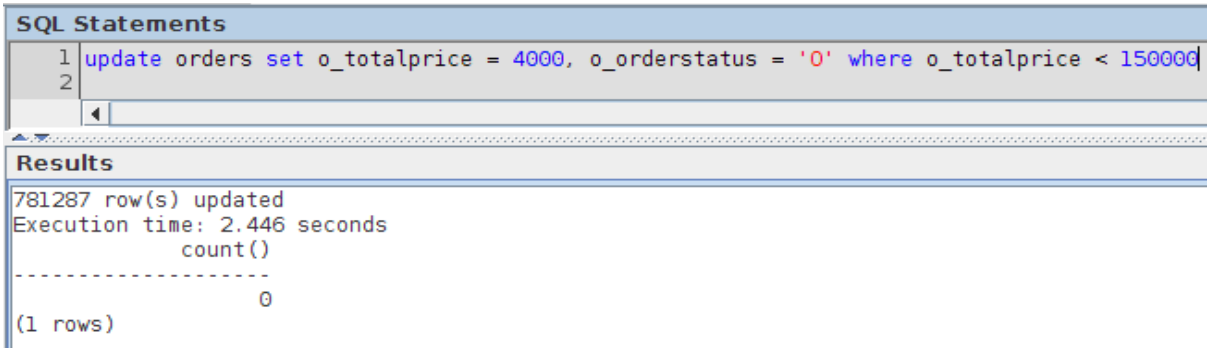
Img 51.Primer update RDBM

En el primer update realizado con una columna podemos observar que el tiempo de ejecución fue menor en el CDBMS que en el RDBMS. Un factor que puede dar pie a estos resultados es que, en el CDBMS, todos los datos de una columna se encuentran juntos por lo que la actualización de una sola columna es muy rápida. Mientras que en el RDBMS se tienen que recorrer varias páginas para encontrar los registros que cumplen con la condición.

Para ilustrar mejor el desempeño de las operaciones de update haremos dos experimentos. Primero actualizaremos dos columnas y después le agregaremos otras dos columnas. Ambas operaciones tendrán la misma condición.

Resultado de realizar operaciones UPDATE en la tabla ORDERS en los campos O_TOTALPRICE, O_ORDERSTATUS donde O_TOTALPRICE sea menor a 150,000.

Resultado en el CDBMS



Img 52.Segundo update CDBMS

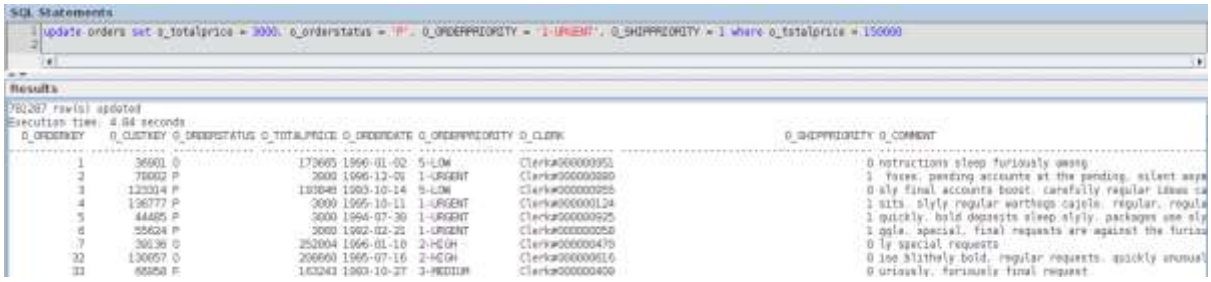
Resultado en el RDBMS

```
SQL> update orders set o_totalprice = 4000, o_orderstatus = 'O' where o_totalpri
ce < 150000;

781287 filas actualizadas.

Transcurrido: 00:00:46.36
SQL>
```

Img 53.Segundo update RDBMS

Resultado de realizar operaciones UPDATE en la tabla ORDERS en los campos O_TOTALPRICE, O_ORDERSTATUS, O_ORDERPRIORITY y O_SHIPPRIORITY donde O_TOTALPRICE sea menor a 150,000.

Resultado en el CDBMS

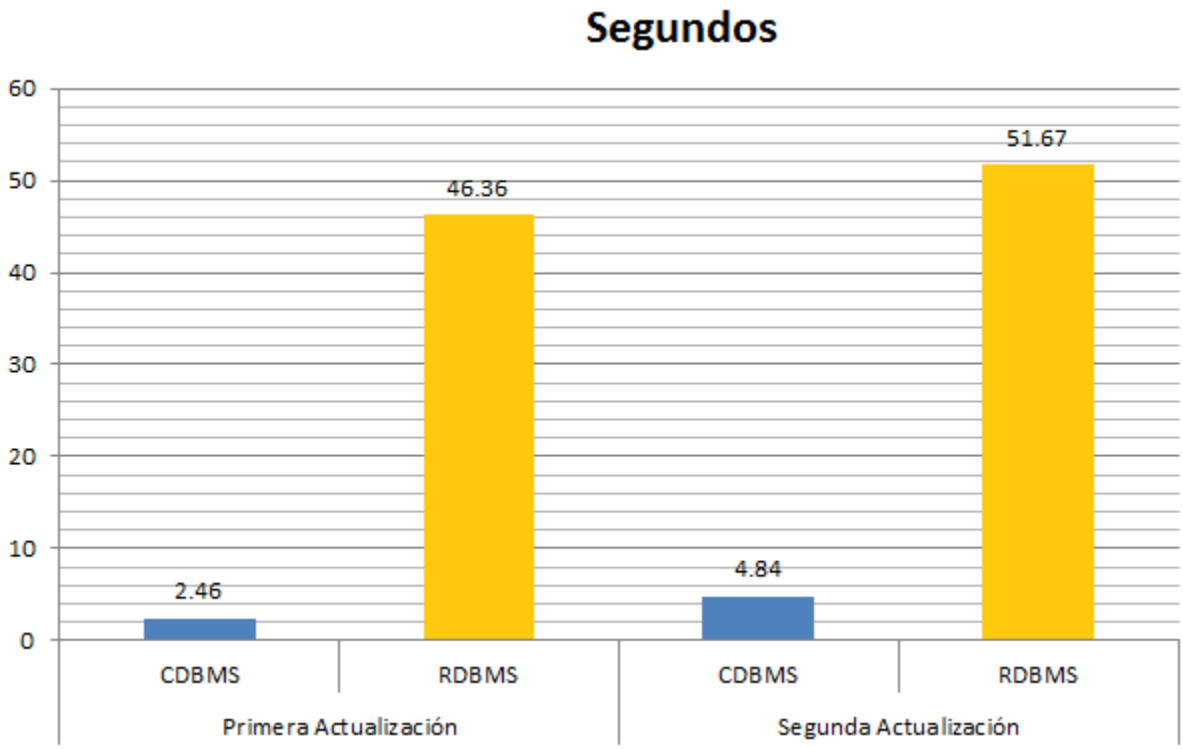


Img 54.Tercer update CDBMS

Resultado en el RDBMS

```
SQL> update orders set o_totalprice = 3000, o_orderstatus = 'P', O_ORDERPRIORITY
 = '1-URGENT', O_SHIPPRIORITY = 1 where o_totalprice < 150000;

781287 filas actualizadas.

Transcurrido: 00:00:51.67
SQL>
```

Img 55.Tercer update RDBMS



Img 56.Gráfica de comparación de tiempos del primer y segundo update

Tabla comparativa de los resultados update

Operación	Número de registros afectados	Tiempo en CDBMS [s]	Tiempo en RDBMS [s]	Veces más rápido
Primer update	1,821,810	0.84	1,026.49	1,221.01
Segundo update	781,287	2.45	46.36	17.92
Tercer update	781,287	4.84	51.67	9.68

Img 57.Tabla de comparación de tiempos de la operación UPDATE

Con dos columnas los tiempos obtenidos fueron 2.45 segundos en el CDBMS y 46.36 segundos en el RDBMS. Mientras que con cuatro columnas los tiempos fueron 4.84 segundos en el CDBMS y 51.67 segundos en RDBMS.

Podemos observar que el CDBMS sigue teniendo un mejor desempeño que el RDBMS, sin embargo, al analizar los tiempos notamos que de dos a cuatro columnas el CDBMS tuvo un incremento del 97.87% mientras que en el RDBMS el incremento del tiempo fue del 11.45%.

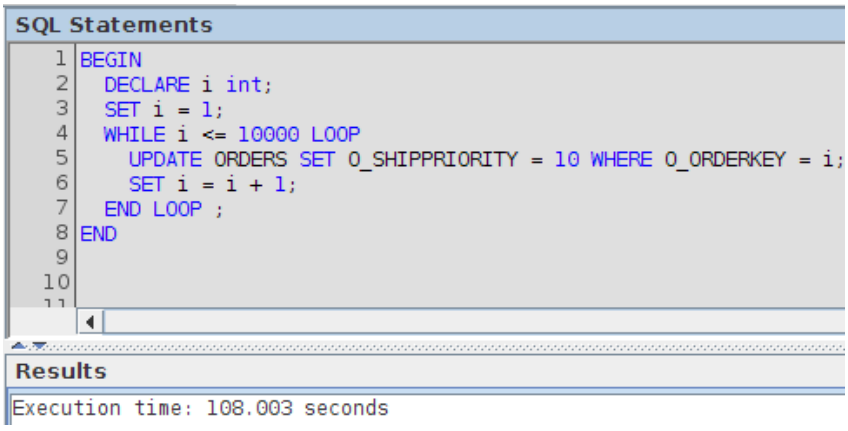
Como en el primer caso, el CDBMS tuvo un mejor desempeño porque la información de las columnas se encuentra junta, sin embargo, al momento de agregar más columnas a la transacción el CDBMS aumenta bastante su tiempo de ejecución debido a que tiene que recorrer todas las páginas de una columna, luego hacer lo mismo con la próxima columna y así sucesivamente. Por otro lado, el RDBMS no aumenta tanto su tiempo debido a que la información de diferentes columnas se encuentra contigua y en la misma lectura puede realizar diferentes actualizaciones. Hay que tomar en cuenta que solo realizamos una transacción sobre un conjunto de datos, por lo que el CDBMS solo realiza una nueva versión de los datos mientras que el RDBMS solo hace una inserción en el log de transacciones.

En las siguientes pruebas de actualización de datos realizaremos varias transacciones en lugar de una sola para poder comparar el desempeño entre ambos manejadores.

- ➢ Prueba de actualización con un conjunto de instrucciones

Resultados de realizar 2,503 operaciones UPDATE sobre la tabla ORDERS

Resultados en CDBMS



Img 58.Resultado de la ejecución de varias operaciones UPDATE en CDBMS

Resultados en el RDBMS

```
SQL> set serveroutput on
BEGIN
    FOR i IN 1..10000 LOOP
    UPDATE ORDERS SET O_SHIPPRIORITY = 10 WHERE O_ORDERKEY = i;
  END LOOP;
END;
/SQL>   2    3    4    5    6

Procedimiento PL/SQL terminado correctamente.

Transcurrido: 00:00:00.35
SQL>
```

Img 59.Resultado de la ejecución de varias operaciones UPDATE en el RDBMS

- Comparación de resultados

Operación	Número de registros modificados	Tiempo en CDBMS [s]	Tiempo en RDBMS [s]	Veces más rápido
UPDATE	2,503	108.00	0.35	307.57

Img 60.Tabla comparativa de tiempos de la operación UPDATE

En esta última operación de update el CDBMS tuvo un desempeño muy lento comparado con el RDBMS. El RDBMS corrió 307 veces más rápido el mismo número transacciones que el CDBMS. Con esto podemos concluir que el CDBMS es bueno en la modificación masiva de datos en una sola transacción debido a que todos los datos de una columna se encuentran contiguos en el disco, sin embargo, cuando se realizan varias transacciones el CDBMS tiene que realizar un versionamiento de los datos en cada transacción, lo que provoca que consuma mucho tiempo y recursos.

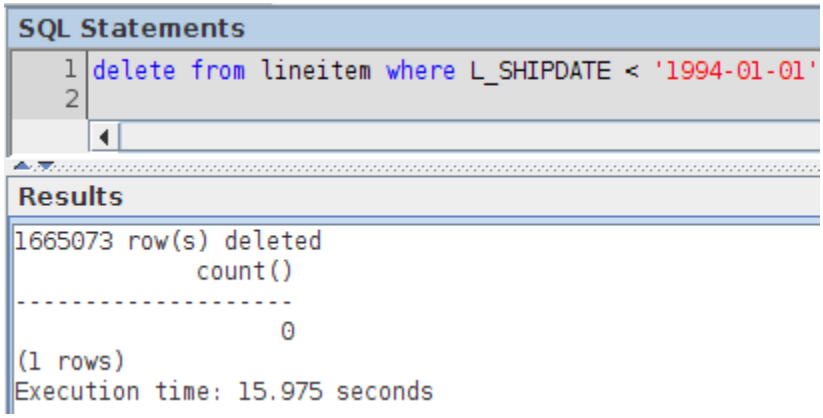
3.4.2. ALGORITMOS DE ELIMINACIÓN

De manera similar a las operaciones UPDATE, también se realizará una prueba sobre la operación de DELETE para conocer el comportamiento de los manejadores. En la prueba un solo delete afectará a un conjunto de registros, es decir, solo se realizará una transacción.

Se presentan los resultados de los tiempos de realizar operaciones de DELETE sobre las tablas.

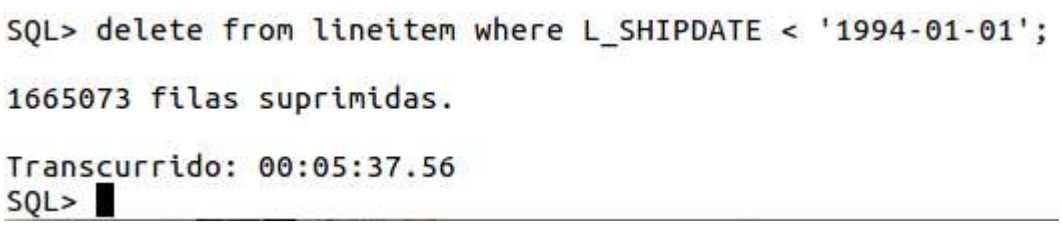
Eliminar los registros de la tabla LINEITEM dónde cumplen con L_SHIPDATE < '1994-01-01'.

Resultado CDBMS



Img 61.Primer delete CDBMS

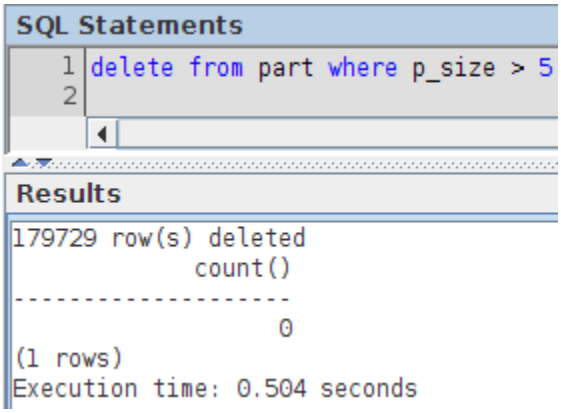
Resultado RDBMS



Img 62.Primer delete RDBMS

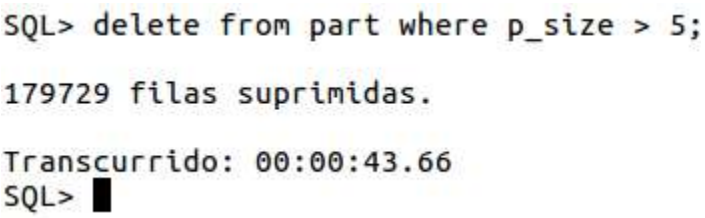
Eliminar los registros de la tabla PART dónde cumplen con P_SIZE > 5.

Resultados en CDBMS



Img 63.Segundo delete CDBMS

Resultado en RDBMS

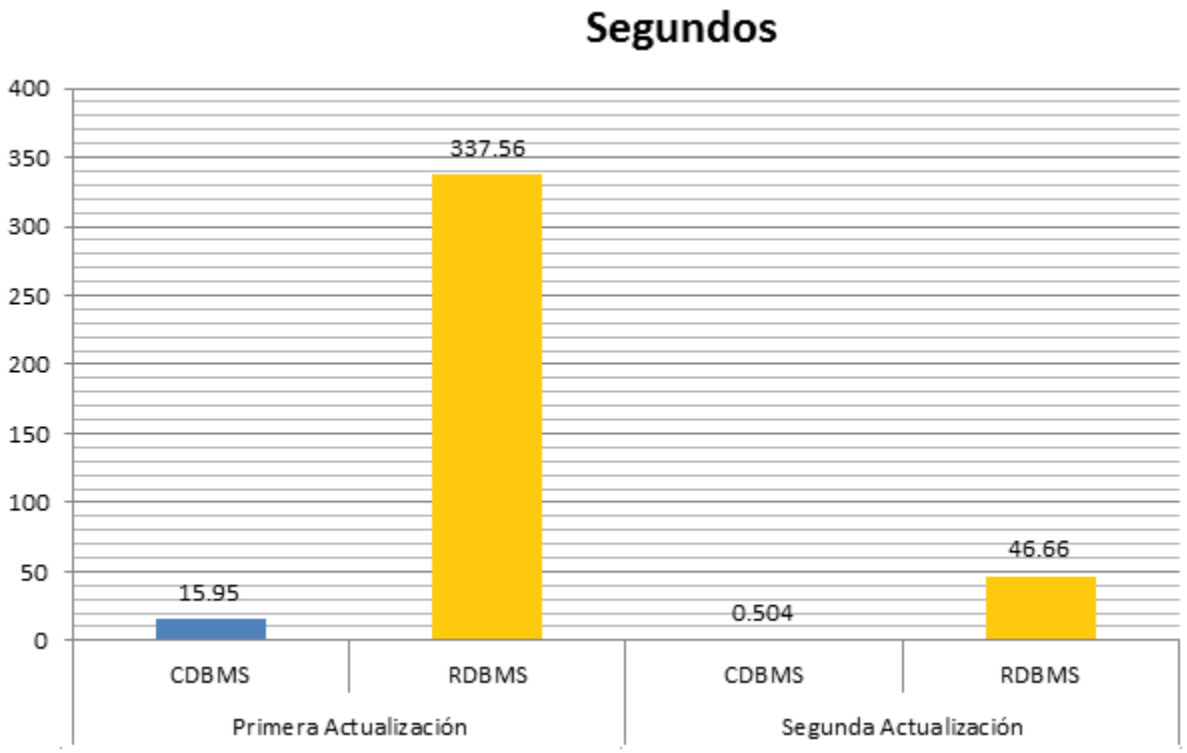


Img 64.Segundo delete RDBMS

- Comparación de resultados

Operación	Registros afectados	Tiempo en CDBMS [s]	Tiempo en RDBMS [s]	Veces más rápido
Primer delete	1,665,073	15.98	337.56	21.12
Segundo delete	179,729	0.50	43.66	87.32

Img 65.Tabla comparativa de las operaciones DELETE



Img 66.Gráfica de los deletes

Como vemos en el primer ejemplo que se realizó se eliminaron 1,665,073 registros, en donde el CDBMS lo realizó en un tiempo de 15.96 segundos, y el RDBMS en un tiempo de 5 minutos con 37.56 segundos. Podemos observar que hay una gran diferencia de tiempos, esto se debe a que el CDBMS maneja índices para el borrado de tablas llamados HG y WD, los cuales eligen el algoritmo indicado dependiendo de la cantidad de registros que se eliminaran. Estos índices se analizarán más adelante.

En este momento solo hablaremos de los algoritmos que utilizan los índices antes mencionados. Los algoritmos dependen de la cantidad de datos que se van a borrar:

- Small delete. Es el algoritmo utilizado cuando se va a eliminar solo un registro o cuando el delete tiene un predicado de igualdad con alguna columna con un índice HG.
- Mid delete. Se utiliza cuando se desea eliminar un grupo grande de datos. Ordena el índice HG para recorrerlo mientras busca los registros a borrar.

- Large delete. Es similar al anterior pero se aplica con un número mayor de registros por borrar. Otra diferencia es que se corre en paralelo para dividir el procesamiento entre varios nodos.

3.5. ALGORITMOS DE COMPRESIÓN DE DATOS

Para poder mejorar el almacenamiento y tener un mejor desempeño al momento de realizar lecturas de datos, las BDC utilizan algoritmos de compresión de datos, por otro lado aplicar estos algoritmos, es decir al comprimir y descomprimir los datos, implica un mayor uso de recursos de cómputo y tiempo de procesamiento. Utilizaremos un ejemplo para poder explicar el concepto de la compresión de datos:

En los sistemas OLAP los registros regularmente tienen fechas para saber cuándo ocurrió el movimiento, Con esta fecha posteriormente se puede hacer análisis y reportes con los historiales de los movimientos. Comúnmente se guardan todas las fechas asociadas a los registros, aún cuando muchas de estas fechas sean iguales debido a que varios movimientos se registraron el mismo día.

Fecha
10/08/1993
10/08/1993
10/08/1993
11/08/1993
11/08/1993
11/08/1993
12/08/1993
12/08/1993

Img 67.Tabla con las diferentes fechas

Como se puede observar en la figura anterior se repiten algunas veces varias fechas. Para evitar eso la compresión de datos asocia un mapa de bits a cada fecha distinta, guardando un diccionario de datos con las relaciones.

Valor	Llave
10/08/1993	1
11/08/1993	2
12/08/1993	3

Img 68.Tabla con la compresión de los datos

Utilizando el diccionario creado por la compresión el almacenamiento de las fechas quedaría de la siguiente manera:

Fecha
1
1
1
2
2
2
3
3

Img 69.Tabla con el almacenamiento de las fechas

Como mencionamos anteriormente, con la compresión de datos ahorramos espacio debido a que es menos costoso almacenar un diccionario de datos con las fechas y después solamente almacenar enteros para poder hacer la relación; además de que mejoramos el desempeño en la lectura de datos porque la lectura de un entero es menos costosa que la lectura de una fecha y al final solo tendríamos que utilizar el diccionario de datos para poder obtener la fecha.

Las bases de datos de almacenamiento por renglón también pueden comprimir datos para optimizar los procesos que ejecutan. En el caso del RDBMS, que se utilizó para ejecutar las pruebas, se puede realizar la compresión de tablas. En este manejador la compresión se realiza de manera similar a como lo hacen los CDBMS en el ejemplo anterior.

El RDBMS pone una tabla de símbolos al inicio del bloque de datos donde se encuentra alojada la información que contienen las tablas. Esta tabla de símbolos es la encargada de guardar los datos no comprimidos para, posteriormente, poder hacer referencia a ellos.

En ambos tipos de bases de datos la eliminación e inserción de registros a las tablas comprimidas es más rápido que en tablas no comprimidas. En el caso de la CDBMS solo se tiene que eliminar o insertar la llave dependiendo del valor que deba procesar. En el RDBMS solo se realizan los procesos con las referencias a la tabla de símbolos (Oracle, 2011).

Sin embargo, a pesar de que la compresión de datos se puede utilizar en ambos tipos de bases de datos, las BDC tienen un mejor desempeño al realizar el proceso anterior al almacenar todos los datos de una misma columna adyacentemente.

En resumen, las ventajas de la compresión de datos o tablas son las siguientes:

- Ahorro de espacio.
- Se mejora la lectura de datos.
- Se mejora la eliminación e inserción de registros a la tabla.

A pesar de que la compresión de datos está disponible en ambos tipos de bases de datos, los manuales del RDBMS recomiendan utilizarla solo en datos que sean de solo lectura o porciones de datos que cambian con poca frecuencia. Basándonos en lo anterior, las tablas que más podrían aprovechar la compresión de datos son las Dimensiones de Cambio Lento (SDCD Slow Data Change Dimension), como pueden ser localizaciones geográficas, tablas de productos o clientes, en ambientes Data Warehouse, debido a que almacenan históricos de información que no cambian regularmente (Oracle, 2011).

3.6. MECANISMOS DE PARALELIZACIÓN

Una de las características de la evolución de los manejadores de bases de datos es que, progresivamente, han buscado la manera de optimizar los procesos. Uno de los

resultados de este avance es la paralelización de procesos. Cada manejador tiene su propia manera de realizar esta paralelización, obteniendo diferentes resultados, ventajas y desventajas.

El CDBMS maneja dos tipos de paralelización de procesos. El primero del que hablaremos es la arquitectura Multiplex. Multiplex es un sistema multinodo, de almacenamiento compartido, que fue diseñado para ambientes Data Warehouse (MacNicol, Roger y French, B., s/f).

La arquitectura Multiplex maneja un conjunto de múltiples instancias del CDBMS repartidas en un conjunto de múltiples nodos conectados a una misma fuente de datos. Cada nodo tiene la capacidad de acceder directamente a la base de datos. Los nodos se encuentran clasificados en tres tipos, nodo coordinador, nodo escritor y nodo lector.

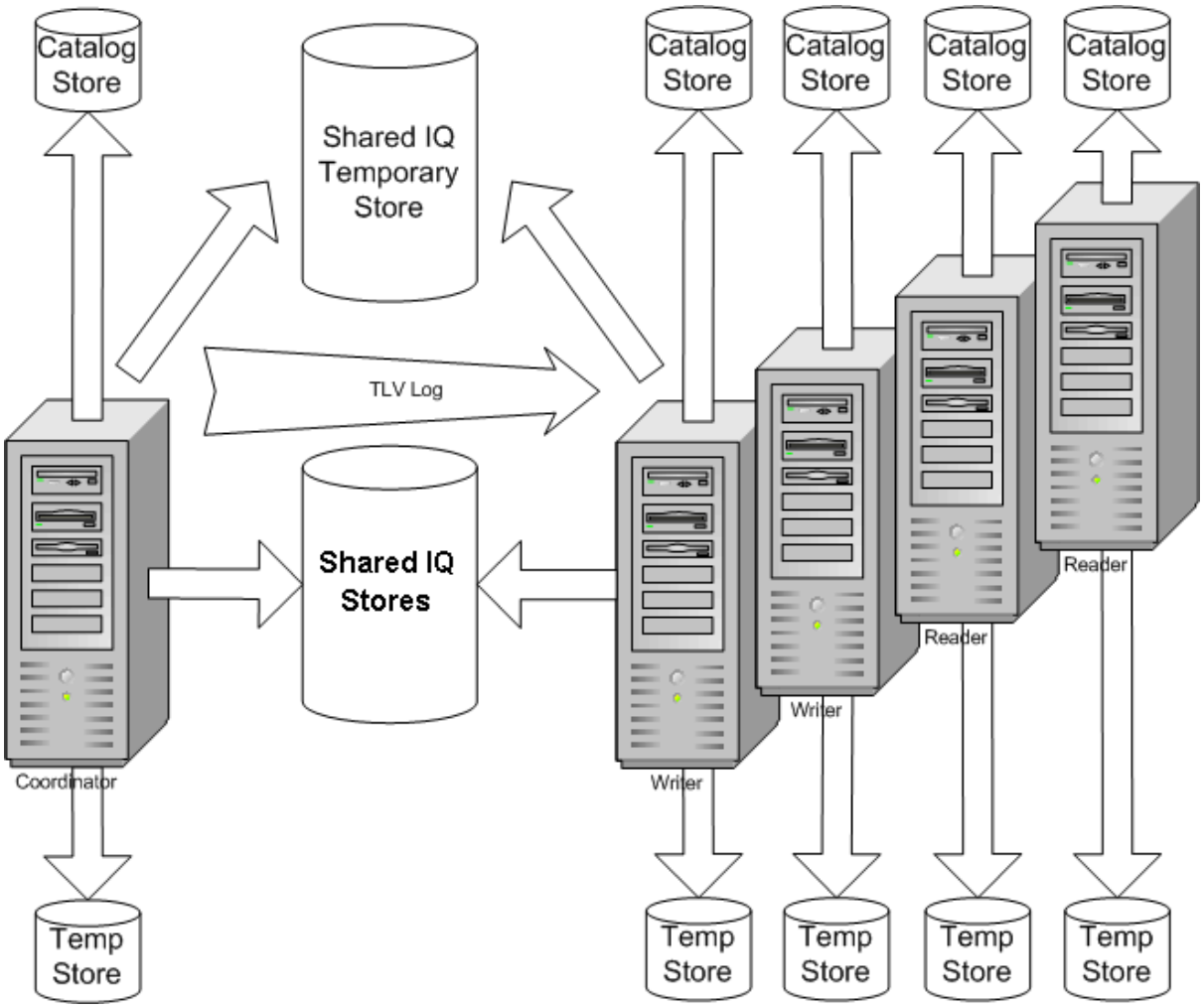
El nodo coordinador, como su nombre lo indica es el encargado de coordinar a los demás nodos para que puedan realizar sus tareas adecuadamente.

Por otro lado, el objetivo del rol escritor es poder manejar las transacciones que se realicen sobre la base de datos, de manera que ese nodo es el único con permisos de escritura sobre los datos.

Finalmente, los nodos lectores sólo tienen permiso de lectura sobre los datos y se encargan de realizar todas las consultas que los usuarios ejecuten sobre la base de datos.

Como se mencionó anteriormente, los nodos están conectados a una misma fuente de datos, conocido como almacenamiento compartido (shared storage en inglés). Sin embargo, cada nodo tiene su propio almacenamiento local (local storage en inglés). En el almacenamiento local se encuentran el catálogo de metadata, los logs de transacciones y los datos temporales.

En la siguiente imagen se puede apreciar la arquitectura Multiplex.



Img 70.Arquitectura Multiplex IQ. Imagen recuperada de SAP.

Físicamente, la comunicación entre los nodos es mínimo debido a que cada uno tiene sus propias funciones, por lo que el costo de la comunicación es bajo. La red soporta una gran cantidad de nodos conectados, por lo que es difícil que se llegue a saturar aún cuando la demanda de los usuarios crezca.

Esta arquitectura nos permite realizar una gran cantidad de consultas sobre la base de datos al mismo tiempo, obteniendo un buen desempeño en cada una de ellas. La desventaja es que, al solo haber un nodo encargado de las transacciones, la arquitectura puede ver su desempeño afectado si se realizan muchas inserciones o actualizaciones a la base de datos.

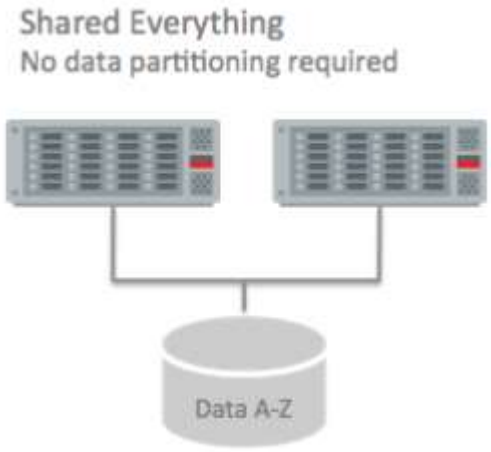
El segundo mecanismo de paralelización es el denominado multi-procesamiento simétrico (SMP por sus siglas en inglés, symmetrical multi-procesing). En el SMP se utiliza un solo nodo que puede realizar tareas simultáneas sobre el mismo conjunto de datos utilizando memoria compartida, así como estructuras de memoria.

El nodo puede ejecutar procesos simétricamente. Dentro de estos procesos se encuentran I/O a disco, bloqueos, entre otros.

El RDBMS también tiene la capacidad de poder paralelizar procesos. La paralelización que realiza RDBMS usa el siguiente proceso:

- Basándose en la información contenida en el overhead el manejador decide si es conveniente paralelizar los procesos. Se puede dar el caso de que sean tan pocos los datos que el mismo proceso de paralelización sea más tardado que simplemente procesar la información.
- El siguiente paso es dividir la información en partes iguales para ser repartido entre los nodos que trabajarán al mismo tiempo, de manera que se pueda garantizar que todos los nodos cumplirán su tarea en el mismo tiempo.

El RDBMS utiliza una arquitectura conocida como *Shared Everything* (Compartir Todo) que utiliza varios nodos, entre los que se dividirá el proceso, y un solo conjunto de datos al que todos pueden acceder. La ventaja que tiene esta arquitectura es que se ahorra el tener que dividir el conjunto de datos en varios subconjuntos. Todos los nodos pueden acceder a toda la información sin limitaciones (Oracle, 2014).



Img 71.Arquitectura Shared Everything. Imagen recuperada de Oracle, 2014.

Las operaciones que puede realizar el RDBMS en paralelo son cargas de datos, sentencias Select, sentencias DML, creación de objetos, etc.

La diferencia principal entre los manejadores se encuentra en la arquitectura, específicamente en las tareas que pueden realizar los nodos. El RDBMS le permite a sus nodos realizar operaciones de escritura y de lectura, mientras que el CDBMS asigna tareas específicas a cada nodo.

3.7. ÍNDICES

Los diferentes manejadores de bases de datos utilizan índices para optimizar la búsqueda de datos. Los más comunes son los árboles en sus diferentes versiones, por ejemplo los árboles balanceados o los árboles B. La mayoría de los índices son propios del manejador de bases de datos, es decir, que las compañías diseñan sus propios índices para poder explotar al máximo sus motores de bases de datos.

No podemos comparar directamente los índices de las bases de datos columnares y de las bases de almacenamiento por renglón por lo mencionado anteriormente, pero podemos analizar varios tipos de índices para entender su fundamento y así poder determinar en que se tienen que basar los diferentes índices dependiendo del problema que intenten resolver, en este caso el almacenamiento columnar y el almacenamiento por renglón.

Para el análisis de las bases de datos columnares revisaremos los índices del CDBMS que estamos utilizando. A diferencia de los índices de las bases de datos tradicionales, los índices usados por el CDBMS ocupan el mismo espacio que la información que se desea indexar.

Así mismo, las consultas se pueden realizar de una manera más eficiente debido a que el CDBMS puede utilizar varios índices al mismo tiempo dependiendo de la información solicitada (SAP, 2013). Para analizar esta característica en particular, tenemos que recordar cómo se almacena la información en disco. Al almacenar de manera contigua cada columna en las páginas de datos es posible que se aplique un índice distinto a cada columna por separado sin que afecte a las otras columnas, de manera similar a lo que hace el RDBMS. Sin embargo, la diferencia principal radica en que el CDBMS permite crear varios índices sobre una misma columna, otorgándole la capacidad de poder decidir qué índice utilizará dependiendo de la consulta que se realice. A diferencia del RDBMS que solo permite generar un índice por columna.

En el manejador que utilizamos, al momento de crear una nueva tabla se crea por defecto un índice llamado, Fast Projection (FP) por cada columna. Este índice se puede complementar con otros índices de los que explicaremos más adelante en este capítulo. El índice FP permite al motor de bases de datos evaluar ciertos tipos de condiciones (SAP, 2013).

Hay dos tipos de índices FP, flat FP o NBit. Para poder definir qué tipo de índice se utiliza la directiva IQ UNIQUE(n). Esta directiva define la cardinalidad, valores distintos, que se espera tener en determinada columna. Cuando se le asigna un valor a esta cláusula se crea un índice flat FP en la columna. Por el contrario, cuando no se declara un valor para esta cláusula se crea un índice NBit dependiendo de las variables FP_NBIT_AUTOSIZE_LIMIT y FP_NBIT_LOOKUP_MB (SAP, 2013).

- FP_NBIT_AUTOSIZE_LIMIT limita el número de valores distintos que puede cargar un NBit.
- FP_NBIT_LOOKUP_MB fija un límite para el tamaño total del diccionario NBit.

- FP_NBIT_ROLLOVER_MAX_MB fija el tamaño del diccionario para cuando se tiene que regresar de un NBit a un flat FP.
- FP_NBIT_ENFORCE_LIMITS hace cumplir los límites puestos anteriormente. Por defecto su valor es OFF.

Como mencionamos anteriormente, IQ UNIQUE se utiliza para definir si el índice será un flat FP, sin embargo, cuando el número de valores distintos sobrepasa el FP_NBIT_AUTOSIZE_LIMIT se cargará un índice NBit.

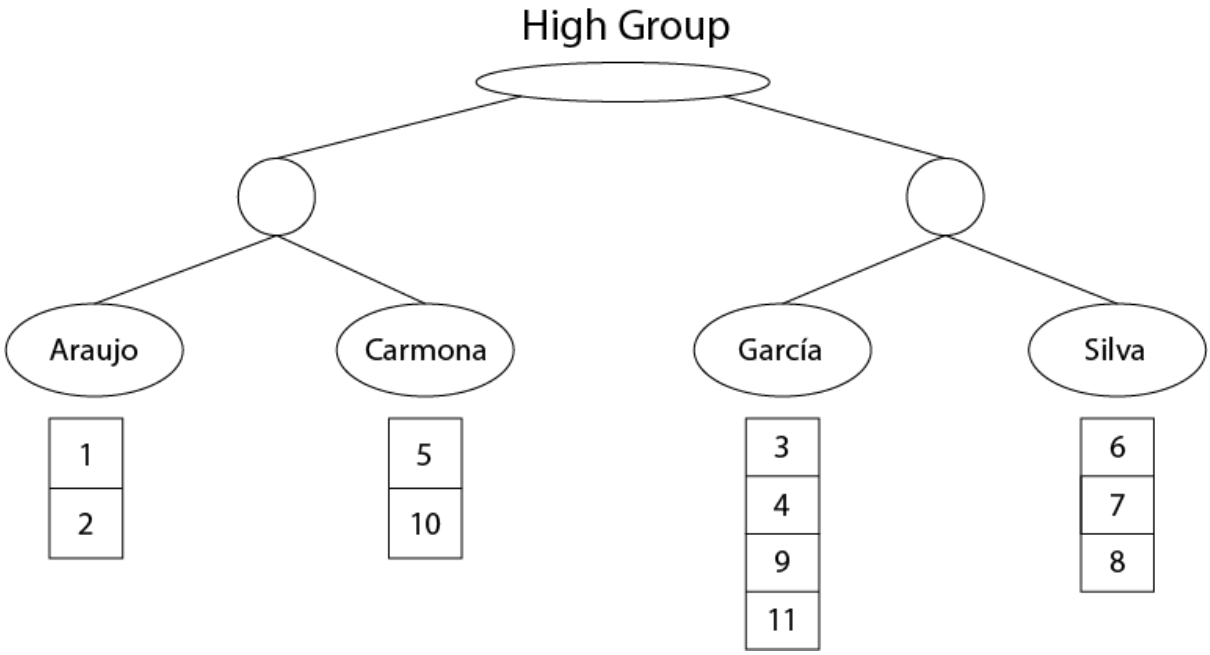
- Índice High Group (HG)

Este tipo de índice es recomendado para usarse en columnas que realizarán joins con datos de tipo entero (SAP, 2013). Los índices HG cuentan con dos variantes, una para utilizarse cuando se quieren indexar varias columnas al mismo tiempo (multicolumn HG) y una sencilla para una sola columna.

Se recomienda utilizar los índices HG cuando se realizará un predicado de join en la consulta y cuando la columna tiene más de 1,000 valores distintos. El multicolumn HG se usa para optimizar el rendimiento de las consultas que utilizan la cláusula ORDER BY con referencias a varias columnas (SAP, 2013). Cabe resaltar que las llaves foráneas deben llevar su propio índice HG.

Los índices HG funcionan utilizando un árbol B y un arreglo dinámico a nivel de columna. Se construye un árbol B de todos los valores de tal manera que los valores que se pueden encontrar en la columna se almacenan en las hojas del árbol. En esas hojas hay un apuntador a un arreglo dinámico que guarda todos los identificadores de los registros que tienen ese valor.

Por ejemplo, tomemos una columna que almacena los apellidos de los habitantes de una colonia. En la siguiente imagen podemos observar cómo se almacenan los valores en el árbol y en cada hoja se encuentran los identificadores de los registros que corresponden con ese apellido.



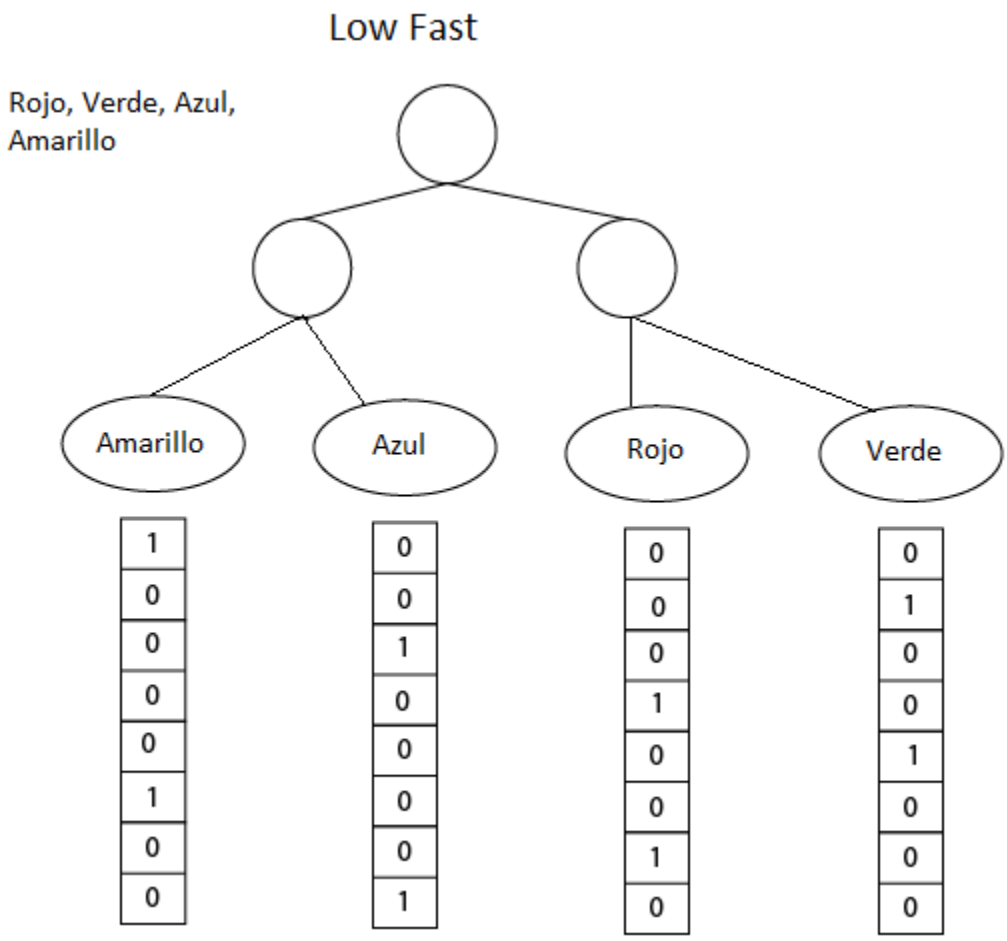
Img 72.Representación de los índices high group

- Índice Low Fast (LF)

Este tipo de índice funciona para columnas con bajas cardinalidades, específicamente con columnas que tienen una cardinalidad menor a 1,000 valores distintos. Cuando se usa correctamente es el índice más rápido que tiene el CDBMS (SAP, 2013). Sin embargo, cuando su cardinalidad empieza a aumentar su desempeño se ve mermado, además de que, entre más valores tenga, mayor será el espacio en disco que utilizará para poder realizar todas sus operaciones internas.

Tiene un árbol B con los valores que se pueden encontrar en la columna, de manera similar al HG. La diferencia es que cada hoja apunta a un índice bitmap.

Si solo utilizáramos un índice bitmap tendríamos que leer todas las columnas para poder saber a cual pertenece. Con el índice LF solo tenemos que recorrer el árbol para poder llegar a un único campo representado por un índice bitmap, con lo que no tenemos que recorrer las otras columnas.

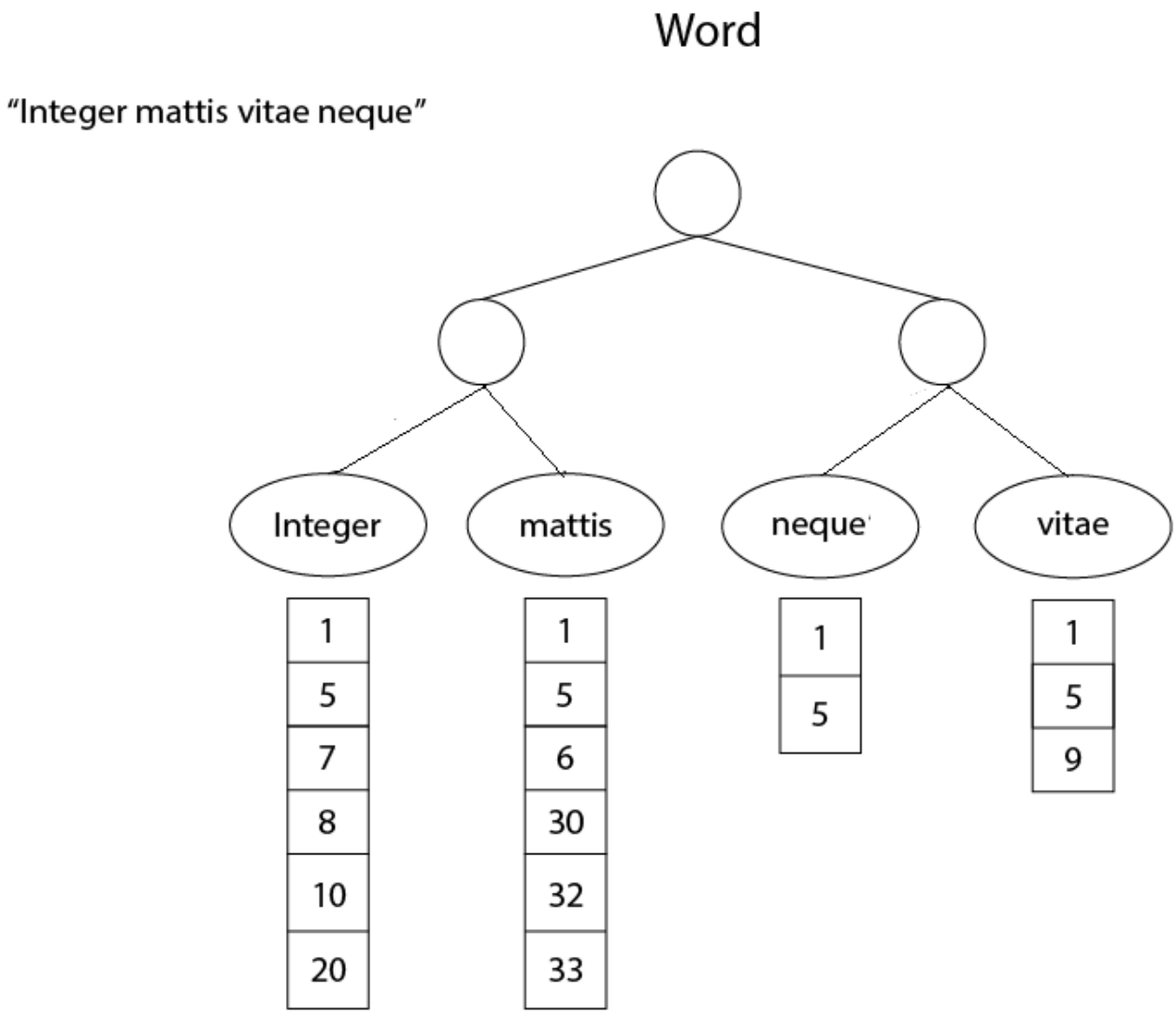


Img 73.Representación de los índices low fast

- Índice Word (WD)

Este tipo de índice funciona de manera similar a los índices HG, utilizando un árbol B y un arreglo dinámico. La diferencia es que los índices Word están optimizados para utilizar texto y comparaciones con palabras como LIKE.

Lo que se almacena en este índice son las palabras (tokens) que conforman los textos almacenados en los campos. Por ejemplo, el texto “Integer mattis vitae neque quis aliquam”, los valores que se almacenarán en el árbol B son “Integer”, “mattis”, “vitae”, “neque”, “quis” y “aliquam”. De esta manera cada hoja guardará los registros donde se presenta esa palabra.



Img 74.Representación de los índices low fast

Una de las restricciones de este índice es que solo funciona con columnas que tengan el tipo de dato CHAR, VARCHAR y LONG VARCHAR. Tampoco se puede utilizar en columnas que tengan el atributo UNIQUE.

Los índices WD solo se pueden utilizar en búsquedas que contengan los predicados LIKE o CONTAINS. Así mismo, se debe tener bastante cuidado con los campos que se declaren como CHAR. Cuando se inserta un registro, el tipo de dato CHAR rellena de espacios la cadena para obtener una cadena con el tamaño especificado en la

declaración, esto puede llegar a entregar resultados erróneos en las consultas que utilicen los índices WD.

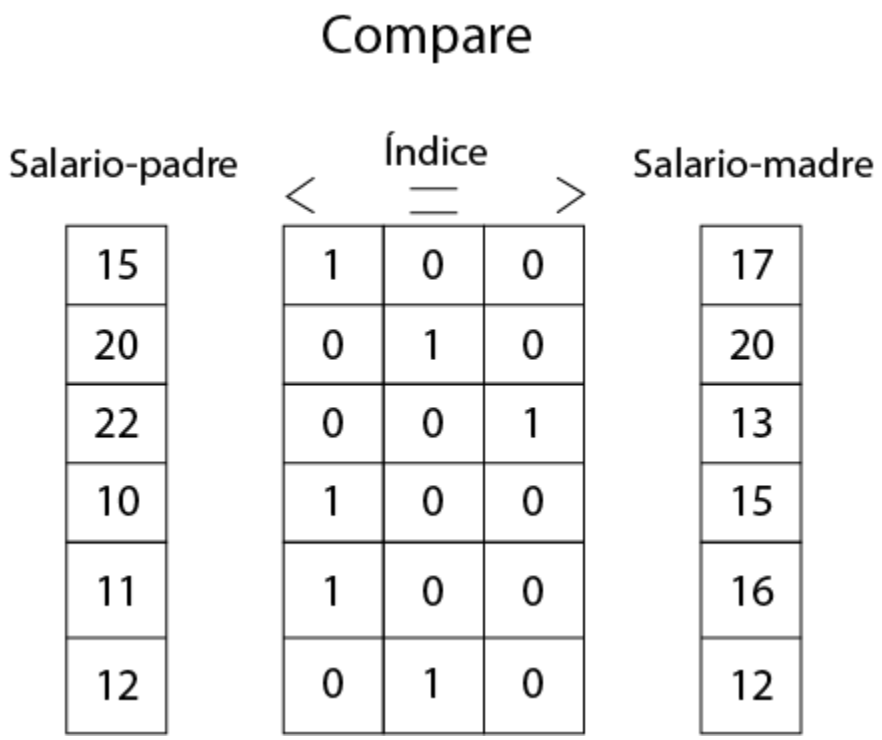
- Índice Compare (CMP)

Este índice se utiliza para poder comparar fácilmente los valores de dos columnas que tengan el mismo tipo de dato, la misma precisión y la misma escala en la misma tabla. El índice CMP guarda la comparación entre las dos columnas, ya sea “<”, “>” o “=” (Sybase, 2013).

En el caso de los tipos de datos CHAR, VARCHAR, BINARY y VARBINARY la precisión se refiere a que las columnas tengan la misma longitud. El índice CMP no se puede utilizar en columnas que tengan el tipo de dato BIT, FLOAT, DOUBLE y REAL.

Las columnas que tengan este tipo de índice no se pueden eliminar o modificar, si se intenta hacerlo se lanzará una excepción en el manejador.

Para dar un ejemplo utilicemos la siguiente imagen que compara dos columnas. Al momento de realizar una consulta que utilice alguna de las comparaciones entre esas dos columnas solo se tiene que recorrer el arreglo de bits de dicha comparación para obtener los registros que cumplen con la condición. De esta manera solo se realiza la comparación una vez, al momento de crear el índice, y posteriormente solo solo se realizan lecturas.



Img 75.Comparación entre columnas

- Índices Date (DATE) y Datetime (DTTM)

Estos índices, como su nombre lo indica, se utilizan cuando se realizan consultas sobre columnas que trabajan con fechas. DATE es usado en columnas con el tipo de dato DATE mientras que DTTM es usado con los tipos de dato DATETIME y TIMESTAMP.

Existen algunas consideraciones que se tienen que tomar para poder utilizar estos índices. Por ejemplo, si un DATE, DATETIME o TIMESTAMP es usado en la cláusula GROUP BY o para realizar operaciones de join, el mejor desempeño será logrado con los índices HG y LF (SAP, 2013).

Para obtener el mejor rendimiento de los índices DATE y DTTM se tiene que utilizar la función DATEPART y una constante en las comparaciones. Por ejemplo, para las comparaciones = y != se utiliza de esta manera:

$$SELECT\ *\ FROM\ table\ WHERE\ DATEPART(YEAR, fecha)\ =\ 1993;$$

$$SELECT\ *\ FROM\ table\ WHERE\ DATEPART(YEAR, fecha)\ !=\ 1993;$$

Para las condiciones de comparación <, >, <=, >=, !>, !< la forma sería la siguiente:

$$SELECT\ *\ FROM\ table\ WHERE\ DATEPART(HOUR, fecha)\ >=\ 10;$$

En el caso de BETWEEN… AND utilizaremos la función DATEPART antes del BETWEEN y constantes numéricas a los lados del AND:

$$SELECT\ *\ FROM\ table\ WHERE\ DATEPART(MONTH, fecha)\ BETWEEN\ 5\ AND\ 9;$$

Finalmente, para IN utilizaremos la función DATEPART a su izquierda y utilizaremos constantes como los valores de comparación:

$$SELECT\ *\ FROM\ table\ WHERE\ DATEPART(MONTH, fecha)\ IN\ (1,3,5,8);$$

Otras restricciones de los índices DATE y DTTM es que solo se pueden utilizar en una sola columna, así mismo, no se pueden utilizar en una columna UNIQUE.

- Índice High_Non_Group (HNG)

Este es una variante de los índices HG. Es usado cuando no se realizarán operaciones de agrupación debido a que utiliza 3 veces menos espacio que los HG. Sin embargo, en la mayoría de los casos los índices HG muestran un mejor desempeño que los índices HNG, por ejemplo, cuando se utilizarán joins entre varias tablas.

El índice HNG es una buena opción cuando la cardinalidad de la columna supera más de 1,000 valores distintos, el límite de los LF, y que no se utilizarán funciones de agrupación, es decir, GROUP BY.

- Índice TEXT

Este tipo de índice se utiliza para el análisis de datos sin estructura. A diferencia del índice WORD, el índice TEXT almacena información de texto basándose en la posición de los términos. Por ejemplo, nos permite buscar en una columna los registros que contengan cierta palabra en cierta posición, esto puede ser antes o después de alguna palabra u oración.

- Índice Time (TIME)

Este índice funciona igual a los índices DATE Y DTTM con la diferencia de que es usado con el tipo de dato TIME.

Al igual que DATE y DTTM, TIME es mejor aprovechado si se usa DATEPART como parte de las comparaciones entre tiempos. Si se realizan comparaciones sin DATEPART es mejor utilizar los índices HG y LF debido a que tienen un mejor rendimiento a la hora de realizar las consultas (SAP, 2013).

Una desventaja es que no soporta Milliseconds en la función DATEPART, además de que solo se puede crear en una columna y no se puede utilizar junto al constraint UNIQUE.

A continuación se muestran algunas tablas para poder comparar de una manera más fácil los índices antes descritos.

En la siguiente tabla podemos ver los índices recomendados dependiendo del tipo de consulta que se utilice.

Tipo de consulta	**Tipo de índice recomendado**
En una proyección select	Índice por defecto (FP)
En el cálculo de expresiones como SUM(A+B)	Índice por defecto (FP)
En el cálculo de expresiones como AVG/SUM	Índice LF, HG, por defecto
En el cálculo de expresiones como	Índice LF, HG

MIN/MAX	
En el cálculo de expresiones como COUNT	Índice por defecto
En el cálculo de expresiones como COUNT DISTINCT, SELECT DISTINCT o GROUP BY	Índice LF, HG, por defecto
Cuando se usa un argumento LIKE en la cláusula WHERE	Índice por defecto
Cuando se usa un argumento IN	Índice HG, LF
Cuando se usa un predicado CONTAINS	Índice WD, TEXT
Cuando se usa DATEPART	Índice DATE, TIME, DTTM
En una igualdad o desigualdad (!=, =)	Índice HG, LF, CMP
Columnas usadas en un join ad hoc	LF, HG, por defecto
Si el campo no permite valores duplicados	Índice HG

Img 76.Tabla con los diferentes índices

Cuando analizamos los índices se mencionó que ciertos índices funcionaban solo con ciertos tipos de datos. En la siguiente tabla expondremos mejor esa característica.

Tipo de dato	**Índices soportados**	**Índices no soportados**

tinyint	CMP, HG, HNG, LF	WD, DATE, TIME, DTTM, TEXT
smallint	CMP, HG, HNG, LF	WD, DATE, TIME, DTTM, TEXT
int	CMP, HG, HNG, LF	WD, DATE, TIME, DTTM, TEXT
unsigned int	CMP, HG, HNG, LF	WD, DATE, TIME, DTTM, TEXT
bigint	CMP, HG, HNG, LF	WD, DATE, TIME, DTTM, TEXT
unsigned bigint	CMP, HG, HNG, LF	WD, DATE, TIME, DTTM, TEXT
numeric, decimal	CMP, HG, HNG, LF	WD, DATE, TIME, DTTM, TEXT
double	LF (HG permitted but not recommended)	CMP, HNG, WD, DATE, TIME, DTTM, TEXT
float	LF (HG permitted but not recommended)	CMP, HNG, WD, DATE, TIME, DTTM, TEXT
real	LF (HG permitted but not recommended)	CMP, HNG, WD, DATE, TIME, DTTM, TEXT

bit	(Default index only)	CMP, HG, HNG, LF, WD, DATE, TIME, DTTM, TEXT
date	CMP, HG, HNG, LF, DATE	WD, TIME, DTTM, TEXT
time	CMP, HG, HNG, LF, TIME	WD, TIME, DTTM, TEXT
datetime, timestamp	CMP, HG, HNG, LF, DTTM	WD, TIME, DATE, TEXT
char <= 255 bytes, character	CMP, HG, HNG, LF, WD, TEXT	DATE, TIME, DTTM
char > 255 bytes	CMP, WD, TEXT	HG, HNG, LF, DATE, TIME, DTTM
varchar <= 255 bytes	CMP, HG, HNG, LF, WD, TEXT	DATE, TIME, DTTM
varchar > 255 bytes	CMP, WD, TEXT	HG, HNG, LF, DATE, TIME, DTTM
long varchar	WD, TEXT	CMP, HG, HNG, LF,

		DATE, TIME, DTTM
binary <= 255 bytes	CMP, HG, LF, TEXT	HNG, WD, DATE, TIME, DTTM
binary > 255 bytes	CMP, TEXT	HG, HNG, LF, WD, DATE, TIME, DTTM
varbinary <= 255 bytes	CMP, HG, LF, TEXT	HNG, WD, DATE, TIME, DTTM
varbinary > 255 bytes	CMP, TEXT	HG, HNG, LF, WD, DATE, TIME, DTTM

Img 77.Tabla con los tipos de datos y sus índices

Una característica del CDBMS es que se puede asignar más de un índice a una columna. La ventaja que nos da sobre otros manejadores de bases de datos es que, dependiendo del tipo de consulta que se utilizará sobre la columna, el manejador utilizará el mejor índice para poder obtener el mejor desempeño en la lectura de datos. La siguiente tabla nos muestra las combinaciones de índices recomendadas y las no recomendadas.

	Índice por añadir					
Índice existente	HG	HNG	LF	CMP	WD	DATE, TIME o DTTM
HG	-	1	2	1	1	1

HNG	1	-	1	1	2	2
LF	2	1	-	1	2	1

Img 78.Tabla con los la combinación de los diferentes índices

Dónde:

- 1 = Combinación recomendada
- 2 = Combinación no recomendada

El índice CMP es un caso especial debido a que se aplica a dos columnas que ya tienen un índice aplicado.

Los índices que utiliza el CDBMS van enfocados a las columnas, es decir, se aplica el índice a la columna seleccionada sin importarle las demás columnas de la tabla. Algunas características de este tipo de índices son las siguientes:

- En el momento que se inserten nuevos datos los índices tienen que ser modificados. Debido a lo anterior, al tener varios índices creados en una tabla las inserciones de nuevos datos pueden volverse lentas al tener que actualizar todos los índices.
- Debido a que la información está almacenada por columnas, los índices pueden comprimir los datos de una determinada columna logrando un ahorro de espacio.
- Tomando el punto anterior, la compresión de los datos también mejora la lectura de los mismos al tener que realizar menores lecturas de páginas de disco para obtener todos los valores que se están buscando.

Hablaremos poco sobre los índices que utiliza el RDBMS debido a que son comunes y muy utilizados por varios manejadores de bases de datos, sin embargo mencionaremos algunas características de cómo administra el RDBMS la información a través de estos índices.

El RDBMS permite crear varios índices en la misma tabla utilizando diferentes columnas o diferentes combinaciones de columnas. Los índices son independientes, tanto lógica como físicamente, de los datos a los que están asociados en una tabla. Debido a esto se pueden crear o eliminar índices de manera independiente a la aplicación, sin embargo, las consultas pueden volverse más lentas sin los índices. Otro punto es que los índices ocupan su propio espacio en el disco (Oracle, 2011).

De manera similar al CDBMS, cuando se actualizan, agregan o eliminan datos los índices también se actualizan (Oracle, 2001), provocando que esas transacciones se alenten si son muchos los índices que se tienen que modificar.

Los RDBMS utilizan índices B-tree por defecto. Las ventajas de estos índices es que son balanceados, es decir, que el tiempo de acceso a cualquier registro es aproximadamente el mismo. Así mismo, tienen un buen desempeño en columnas que tienen una pequeña o una gran cantidad de registros. Una desventaja de este tipo de índices es que ocupan un espacio en disco muy grande, similar o incluso más grande que el espacio utilizado por los mismos datos.

Otro tipo de índices que utiliza el RDBMS son los índices Bitmap. Estos índices tienen su mejor rendimiento cuando son utilizados en columnas que tienen una baja cardinalidad. Otra característica es que ocupan menos espacio que otros tipos de índices al utilizar solo dos tipos de valores para crear el índice, 0 y 1.

A pesar de que los índices de ambos manejadores tienen la misma funcionalidad, mejorar la lectura de los datos, hay varias diferencias entre ellos. Para poder realizar una mejor comparación analizaremos el fundamento de cada conjunto de índices, en lugar de compararlos uno a uno.

Definiremos a los índices que utiliza el CDBMS como índices orientados a columnas y los índices que utiliza el RDBMS como índices orientados a renglones. En esta comparación retomaremos varios conceptos vistos anteriormente, como el mecanismo de almacenamiento.

Para realizar la comparación analizaremos dos casos, cuando se requiere leer una sola columna y cuando se requieren leer todas las columnas de una tabla utilizando solo una de ellas como condición.

En la lectura de una sola columna los índices orientados a columnas tienen un buen desempeño debido a que, por defecto, todas las columnas tienen un índice asociado que puede ser cambiado dependiendo de las características de cada columna. Eso nos asegura que, sin importar la columna que quiera ser leída, un índice nos ayudará a realizar una búsqueda más rápida. Por otro lado, los índices orientados a renglones no nos aseguran que la columna que estemos buscando o la que estemos usando para la comparación tenga un índice, por lo que dependemos de la creación de nuestros índices para obtener un mejor desempeño.

En el segundo caso, cuando se quieren realizar la lectura de varias columnas de una tabla, los índices orientados a columnas no tienen tan buen desempeño debido a que, aunque se utilicen los índices en las columnas que tienen las comparaciones, se realizarán varios accesos a disco para obtener todas las páginas que contienen las columnas. En los índices orientados a registros, nuevamente dependerá de si tenemos índices asignados a las columnas que se utilizan en las comparaciones para obtener un mejor desempeño, teniendo en cuenta que todos los campos de un registro se encuentran juntos en la misma página, por lo que, al usar el índice, las lecturas en disco se reducirían.

De lo anterior podemos concluir que los índices orientados a renglones tienen un desempeño promedio en ambos casos, mientras que los índices orientados a columnas tienen un mejor desempeño cuando solo se lee una o pocas columnas pero su desempeño se ve muy mermado cuando se tienen que leer varias columnas.

4. Conclusiones de la Aplicación de una Base de Datos Columnar

4.1. Aplicaciones de las BD Columnares

Tomando en cuenta a los índices, añadir o eliminar un índice o una columna en ambos tipos de bases de datos es muy costoso debido a que se deben actualizar todas las páginas de datos para poder manipular la información requerida. Las bases de datos columnares tienen la capacidad de crear varios índices sobre una misma columna. Esto nos puede encaminar un poco más a la utilización de este tipo de bases de datos. Las bases de datos columnares nos sirven en ambientes que tengan pocas transacciones atómicas pero en los que se realicen una gran cantidad de lecturas sobre los datos o que modifiquen una gran cantidad de registros para una sola operación, para aprovechar al máximo sus capacidades.

Tomando en cuenta lo analizado en los capítulos anteriores podemos concluir que la aplicación por excelencia de las BD Columnares son los ambientes Data Warehouse y los DSS. Los Data Warehouse son sistemas donde se almacenan grandes cantidades de datos para poder realizar análisis de la información que ayuden a la toma de decisiones de las empresas. Esa información se construye a partir de las transacciones realizadas en un ambiente OLTP. El llenado de un Data Warehouse se puede llevar a cabo por un proceso de extracción, transformación y carga (ETL) o mediante una replicación de las transacciones del ambiente OLTP.

Los Data Warehouse tienen como objetivo el “consolidar información proveniente de diferentes bases de datos operacionales y hacerla disponible para la realización de análisis de datos de tipo gerencial” (Peralta, Veronika, 2001). Podemos decir que las operaciones que se realizan mayormente en un Data Warehouse son las lecturas de grandes cantidades de datos para poder realizar reportes de todas las transacciones realizadas en diferentes periodos de tiempo.

En ambientes OLTP, en los que se realizan una gran cantidad de transacciones en periodos cortos de tiempo, las BD Columnares tienen un desempeño muy pobre comparado con las bases de datos de almacenamiento por renglón.

4.2. Beneficios de las BD Columnares

Como se ha visto en todo el proyecto, los beneficios de las BD Columnares son las rápidas lecturas que puede realizar sobre una columna en particular o un conjunto limitado de columnas, así como la compresión de los datos que logra un ahorro importante de disco.

La capacidad de indexar varias columnas de una tabla e inclusive el poder crear varios índices en una misma columna representa una gran capacidad para poder realizar búsquedas de grandes cantidades de datos de una manera eficiente.

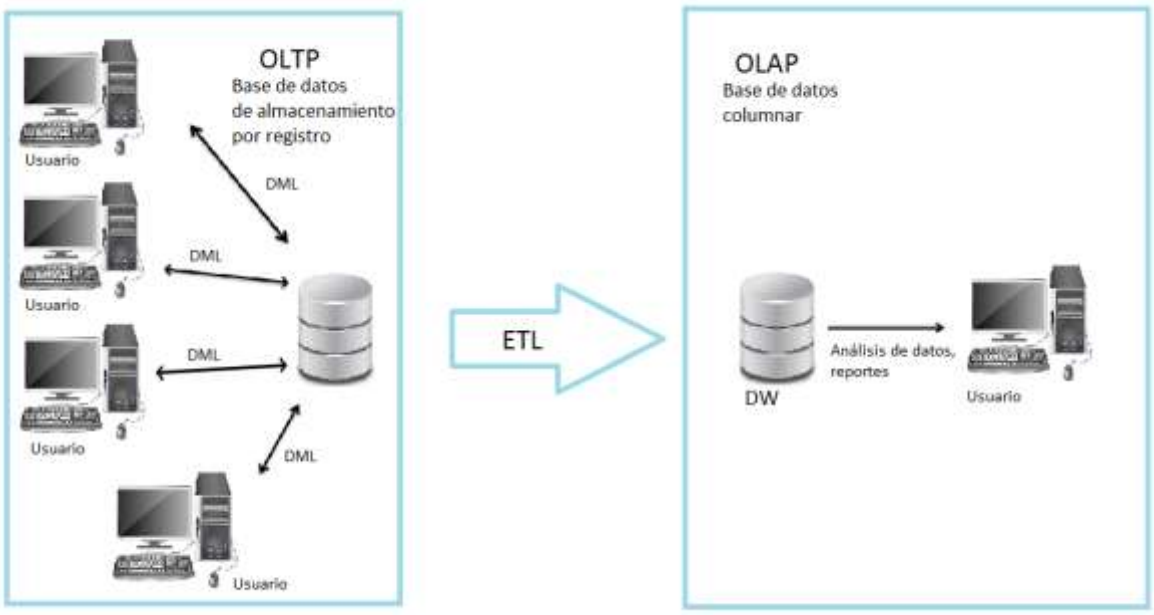
Sin embargo, todo debe permanecer en equilibrio. La misma capacidad de realizar búsquedas de datos puede costar mucho tiempo al momento de generar inserciones, eliminaciones o actualizaciones atómicas de los datos.

Llegamos a la conclusión de que se puede utilizar una base de datos de almacenamiento por renglón y una base de datos columnar en la misma empresa, de manera que se aproveche lo mejor de cada paradigma.

Supongamos que tenemos un supermercado que realiza cientos de ventas de miles de productos al día. Una base de datos de almacenamiento por renglón podría soportar todas las transacciones que se realizan día a día debido a que están optimizadas para poder acceder fácilmente a las páginas que almacenan un renglón. Así mismo, realizar las inserciones solo es cuestión de acceder a la última página de datos creada y guardar la información. Sin embargo, al momento de analizar las grandes cantidades de datos vería mermado su desempeño, ya que, como se analizó anteriormente, acceder a una o dos columnas para poder obtener reportes derivaría en tener que leer prácticamente todas las páginas de datos.

Una solución a esa problemática sería el diseñar un ambiente Data Warehouse para el mismo supermercado, donde, periódicamente, se almacene toda la información importante para poder realizar un análisis de los movimientos. Este ambiente de Data Warehouse debería ser administrado por una base de datos columnar. Como se observó en las pruebas realizadas anteriormente, cargar los datos utilizando un mecanismo similar a LOAD TABLE nos reduce el tiempo de inserción. En este caso podríamos utilizar LOAD TABLE debido a que no es necesario guardar los registros en ningún log de transacciones ya que la información estaría consolidada en la parte transaccional del sistema. Sin embargo, para poder hacer uso de este segundo sistema es necesario realizar un análisis de la información para poder generar un Data Mart donde se pueda almacenar la información, lo que llevaría a un costo extra.

En la siguiente imagen se muestra un ejemplo de la arquitectura.



Img 79.Imagen de una arquitectura OLTP a OLAP

Para concluir, utilizando este esquema podríamos tener los beneficios de ambos tipos de bases de datos sin tener ninguna de sus desventajas técnicas. La única

desventaja que podría tener el utilizar los dos tipos de bases de datos sería el costo de mantener las dos bases de datos, así como sus licencias si es que se opta por una versión comercial.

Otros ejemplos de este ambiente mixto los podemos encontrar, por ejemplo, en una compañía de telefonía o en un banco. En la compañía de telefonía diariamente se registran miles o millones de llamadas que pueden ser manejadas por una base de datos de almacenamiento por renglón. La base de datos columnar puede entrar al momento de realizar reportes sobre las llamadas realizadas en un periodo de tiempo dado.

De igual manera, en un banco se realizan miles de transacciones en un corto periodo de tiempo, así mismo, los bancos emiten reportes de esas transacciones al finalizar el día o la semana, por lo que tener un ambiente mixto con una base de datos de almacenamiento por renglón y una base de datos columnar sería un esquema de gran ayuda.

Cabe recalcar que, actualmente, la mayoría de los manejadores de bases de datos de almacenamiento por renglón están complementando su oferta tecnológica con versiones columnares debido al auge que están teniendo los ambientes data warehouse en la actualidad. Podemos mencionar los casos de Sybase IQ, Oracle en su ambiente columnar, entre otros.

4.3. Principales proveedores de BD Columnares

Finalmente, describiremos a los principales proveedores de bases de datos columnares, así como sus características especiales, es decir, lo que los diferencía de los otros manejadores de bases de datos.

- Infobright

Es una base de datos columnar que comprime los datos en paquetes. Al mismo tiempo almacena información sobre esos datos, metadata, para poder realizar consultas de una manera optimizada. Esto lo logra analizando la información que se pide en la

consulta para saber exactamente qué paquetes de datos debe descomprimir para regresarle al usuario.

Debido a su arquitectura elimina la necesidad de crear índices o particionar tablas, lo que reduce el costo de la administración de la misma base. Así mismo, guarda gran cantidad de estadísticas que le permiten “aprender” de las consultas recibidas para entregar mejores tiempos de respuesta

Infobrigth no tiene productos de licencia gratuita pero cuenta con una versión de prueba de 30 días con todas sus funcionalidades, después de este periodo se requiere de adquirir la licencia pagada.

- Vertica

Es un producto adquirido por HP. Gracias a su arquitectura puede ser escalado fácilmente con solo agregar servidores a la red. Otra característica es que puede comprimir grandes cantidades de datos ahorrando espacio en disco. También permite paralelizar las consultas entre varios nodos a un almacenamiento redundante de la información.

Una de sus características principales es que permite crear proyecciones de los datos, de tal manera que los datos se almacenen tal como se van a leer, ahorrando tiempo de lectura.

Vertica maneja dos espacios de almacenamiento. En el primero, llamado zona de escritura, se guardan los datos que se han insertado recientemente. Estos datos tienen la característica de que son susceptibles a sufrir modificaciones, por lo tanto, la zona de escritura está optimizada para modificar los datos. La segunda zona, o zona de lectura, se utiliza para guardar los datos que han sido consolidados y que es poco probable sufran modificaciones.

Cuenta con una versión Community Edition la cual tiene un soporte de hasta 1TB por tiempo ilimitado. Además cuenta con versiones en la nube y otra para SQL en Hadoop, esta última cuenta con 30 días de prueba. (HP, 2012).

- Greenplum

Greenplum es una base de datos basada en Postgres que utiliza una arquitectura *Shared nothing* diseñada para ambientes Data Warehouse (Wass, Florian). Gracias a su arquitectura puede realizar una paralelización de procesos, como pueden ser transacciones, sentencias DDL, sentencias Select, etc.

En esencia, Greenplum funciona como un conjunto de varias instancias de PostgreSQL funcionando como un solo DBMS (GoPivotal, 2013). Las instancias de PostgreSQL son modificadas de tal manera que pueda soportar la arquitectura que utiliza Greenplum.

Algunas de las características de Greenplum son las siguientes:

- Carga de datos en paralelo.
- Administración de recursos.
- Optimización de consultas.
- Mejoras en el almacenamiento de datos.

Algunas de las características que se implementaron en Greenplum, posteriormente fueron implementadas en PostgreSQL, tal es el caso de el particionamiento de tablas. Greenplum es un manejador de tipo Open Source por lo que cuenta con una licencia completamente gratuita (GoPivotal, 2013).

CONCLUSIONES

Al realizar este trabajo pudimos obtener los resultados esperados, debido a que las bases de datos columnares tuvieron un mejor desempeño en casi todos los rubros que son compatibles con los ambientes Data Warehouse, estos rubros son el tiempo de carga de datos, el tiempo de ejecución de consultas y operaciones de UPDATE y DELETE en donde se realizaba una sola transacción. Por otra parte el almacenamiento de datos por columnas tuvo un bajo desempeño al momento de realizar un conjunto grande de transacciones.

Mediante estos resultados se puede diseñar una arquitectura que obtenga los beneficios del almacenamiento columnar y por renglón, dependiendo de las necesidades del sistema y los recursos con los que se cuente.

Profesionalmente, durante toda la carrera no tuvimos la experiencia de hacer una investigación de este tipo, por lo que nos llevó más tiempo de lo planeado el realizar el trabajo. Sin embargo, esta experiencia nos ayudó a desarrollar prácticas y habilidades que son necesarias para realizar este tipo de trabajo y que podríamos utilizar en caso de realizar otras investigaciones.

También conocimos sobre más tecnologías emergentes y las que ya estaban establecidas como son los manejadores de bases de datos que fueron mencionados, así mismo, afianzamos varios conceptos que vimos a lo largo de la carrera, como son los diferentes algoritmos de ordenamiento e indexado, la escritura sobre páginas de disco y el diseño, manejo y administración de bases de datos. Esto nos brinda una ventaja porque no muchos profesionistas conocen y manejan este tipo de bases de datos.

Finalmente, nos pareció un tema muy interesante ya que al investigar sobre las bases de datos columnares aprendimos sobre otros temas de computación en general.

ANEXO

INSTALACIÓN Y PRUEBAS DE DESEMPEÑO DE AMBOS MANEJADORES.

Estas pruebas se están realizando en un Sistema Operativo Ubuntu 16.04 LTS de 64-bit, memoria de 7.8 GB, procesador Intel Core™ i3-2310M CPU @ 2.10GHz × 4, con un disco duro de 978.1 GB.

TPCH

Instalación y configuración de la herramienta

Para poder realizar las pruebas se realizará un poblado de datos, se utilizará el generador de datos TPC-H con el programa DBGEN. Primero descargamos el programa tpch dbgen de la página oficial, para este ejemplo utilizaremos la versión 2.17.1.

Liga de descarga tpch:

http://www.tpc.org/tpc_documents_current_versions/current_specifications.asp

Nos registramos y recibiremos un correo en el cual nos vendrá la liga para la descarga. Una vez que tengamos el archivo descargado proseguiremos con la generación de los datos.

Para ello seguiremos la referencia de la siguiente página https://husnusensoy.wordpress.com/2010/10/22/create-your-own-oracle-tpc-h-playground-on-linux/, en la cual explica cómo generar tus datos para un ambiente RDBMS con un sistema operativo Linux utilizando la herramienta TPC-H.

Primero debemos asegurarnos que nuestro sistema operativo tiene las herramientas necesarias para compilar el lenguaje C, puede ser gcc o make en caso de Linux.

Para revisar si tenemos gcc o make en nuestro SO podemos utilizar los siguientes comandos:

Para gcc

```
oscar@oscar-Satellite-C645:~$ gcc --version
gcc (Ubuntu 4.8.4-2ubuntu1~14.04.1) 4.8.4
Copyright (C) 2013 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Para make

```
oscar@oscar-Satellite-C645:~$ make --version
GNU Make 3.81
Copyright (C) 2006  Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
```

En caso de no contar con estos compiladores se puede utilizar los siguientes comandos para su instalación:

Para gcc:

sudo apt-get install gcc

Para make:

sudo apt-get install make

Una vez confirmado el compilador para C se actualizan paquetes y dependencias de los programas, es decir obtener las versiones más actuales de los paquetes ya instalados y evitar problemas de compatibilidad:

sudo apt-get update

Después debemos crear una carpeta en la cual vamos a tener nuestro generador de datos, para ello seguiremos los siguientes pasos:

- Crear la carpeta.

sudo mkdir Documents/BD_COLUMNAR/tpch

- Mover el archivo tpch que descargamos al directorio que acabamos de crear.

sudo mv Downloads/06bd4ba2-8a59-45cb-9522-1912dfc0dcd6-tpc-h-tool.zip Documents/BD_COLUMNAR/tpch

- Acceder a la carpeta tpch y descomprimimos el archivo que descargamos.

cd Doc uments/BD_COLUMNAR/tpch

sudo unzip 06bd4ba2-8a59-45cb-9522-1912dfc0dcd6-tpc-h-tool.zip

- Si se desea, se elimina el archivo .zip.

sudo rm -R 06bd4ba2-8a59-45cb-9522-1912dfc0dcd6-tpc-h-tool.zip

- Acceder a la dirección donde se encuentra los archivos de configuración del generador de datos.

cd tpch_2_17_0/dbgen/

- Copiar el archivo makefile.suite y lo nombraremos makefile

sudo cp makefile.suite makefile

- Modificar el archivo makefile con los siguientes parámetros, en este ejemplo se utiliza el editor *nano* de Linux:

sudo nano makefile

```
################
## CHANGE NAME OF ANSI COMPILER HERE
################
CC      = gcc
# Current values for DATABASE are: INFORMIX, DB2, TDAT (Teradata)
#                                  SQLSERVER, SYBASE, ORACLE, VECTORWISE
# Current values for MACHINE are:  ATT, DOS, HP, IBM, ICL, MVS,
#                                  SGI, SUN, U2200, VMS, LINUX, WIN32
# Current values for WORKLOAD are:  TPCH
DATABASE= ORACLE
MACHINE =  LINUX
WORKLOAD = TPCH
#
```

CC: tipo de compilador C que vamos a utilizar en este caso utilizaremos gcc.

DATABASE: nombre del manejador de base de datos que vamos a utilizar en este caso ORACLE.

MACHINE: tipo de Sistema Operativo que se está utilizando, en esta ocación es LINUX.

WORKLOAD: tipo de herramienta que se está utilizando, en este caso TPCH.

- Guardar los cambios y salir del editor
- El siguiente paso es ejecutar el siguiente comando para poder ejecutar el archivo update_release.sh

make

chmod 755 update_release.sh

./update_release.sh 2 12 0

El resultado debe ser parecido al siguiente:

```
print.c:138:3: warning: format '%ld' expects argument of type 'long int', but argument 4 has type 'int' [-Wformat=]
gcc -g -DDBNAME=\"dss\" -DLINUX -DORACLE -DTPCH -DRNG_TEST -D_FILE_OFFSET_BITS=64    -c -o load_stub.o load_stub.c
gcc -g -DDBNAME=\"dss\" -DLINUX -DORACLE -DTPCH -DRNG_TEST -D_FILE_OFFSET_BITS=64    -c -o bcd2.o bcd2.c
gcc -g -DDBNAME=\"dss\" -DLINUX -DORACLE -DTPCH -DRNG_TEST -D_FILE_OFFSET_BITS=64    -c -o speed_seed.o speed_seed.c
gcc -g -DDBNAME=\"dss\" -DLINUX -DORACLE -DTPCH -DRNG_TEST -D_FILE_OFFSET_BITS=64    -c -o text.o text.c
gcc -g -DDBNAME=\"dss\" -DLINUX -DORACLE -DTPCH -DRNG_TEST -D_FILE_OFFSET_BITS=64    -c -o permute.o permute.c
gcc -g -DDBNAME=\"dss\" -DLINUX -DORACLE -DTPCH -DRNG_TEST -D_FILE_OFFSET_BITS=64    -c -o rng64.o rng64.c
gcc -g -DDBNAME=\"dss\" -DLINUX -DORACLE -DTPCH -DRNG_TEST -D_FILE_OFFSET_BITS=64  -O -o dbgen build.o driver.o bm_utils.o rnd.o
rint.o load_stub.o bcd2.o speed_seed.o text.o permute.o rng64.o -lm
gcc -g -DDBNAME=\"dss\" -DLINUX -DORACLE -DTPCH -DRNG_TEST -D_FILE_OFFSET_BITS=64    -c -o qgen.o qgen.c
qgen.c: In function 'qsub':
qgen.c:197:25: warning: zero-length gnu_printf format string [-Wformat-zero-length]
                       fprintf(ofp, SET_DBASE, db_name);
                       ^
qgen.c: In function 'main':
```

Nota: puede indicar unos warnings a la hora de compilar los archivos pero no tiene conflicto con el programa.

Si todo resulto bien entonces ya tenemos nuestro dbgen configurado, lo podemos corroborar ejecutando el siguiente comando en la carpeta de generación de datos:

./dbgen -h

```
root@oscar-Satellite-C645:/home/oscar/Documents/BD_COLUMNAR/tpch/tpch_2_17_0/dbgen# ./dbgen -h
TPC-H Population Generator (Version 2.17.0 build 0)
Copyright Transaction Processing Performance Council 1994 - 2010
USAGE:
dbgen [-{vf}][-T {pcsoPSOL}]
        [-s <scale>][-C <procs>][-S <step>]
dbgen [-v] [-O m] [-s <scale>] [-U <updates>]

Basic Options
===========================
-C <n> -- separate data set into <n> chunks (requires -S, default: 1)
-f     -- force. Overwrite existing files
-h     -- display this message
-q     -- enable QUIET mode
-s <n> -- set Scale Factor (SF) to  <n> (default: 1)
-S <n> -- build the <n>th step of the data/update set (used with -C or -U)
-U <n> -- generate <n> update sets
-v     -- enable VERBOSE mode

Advanced Options
===========================
-b <s> -- load distributions for <s> (default: dists.dss)
-d <n> -- split deletes between <n> files (requires -U)
-i <n> -- split inserts between <n> files (requires -U)
-T c   -- generate cutomers ONLY
-T l   -- generate nation/region ONLY
-T L   -- generate lineitem ONLY
-T n   -- generate nation ONLY
-T o   -- generate orders/lineitem ONLY
-T O   -- generate orders ONLY
-T p   -- generate parts/partsupp ONLY
-T P   -- generate parts ONLY
-T r   -- generate region ONLY
-T s   -- generate suppliers ONLY
-T S   -- generate partsupp ONLY

To generate the SF=1 (1GB), validation database population, use:
        dbgen -vf -s 1
```

Generar la carga de datos

Para generar nuestros datos ejecutaremos el archivo dbgen que se encuentra en el directorio donde se encuentra el programa. Utilizaremos el siguiente comando para ejecutar nuestro archivo dbgen:

./dbgen -s 1 -v

```
root@oscar-Satellite-C645:/home/oscar/Documents/BD_COLUMNAR/tpch/tpch_2_17_0/dbgen# ./dbgen -s 1 -v
TPC-H Population Generator (Version 2.17.0)
Copyright Transaction Processing Performance Council 1994 - 2010
Generating data for suppliers table/
Preloading text ... 100%
done.
Generating data for customers tabledone.
Generating data for orders/lineitem tablesdone.
Generating data for part/partsupplier tablesdone.
Generating data for nation tabledone.
Generating data for region tabledone.
root@oscar-Satellite-C645:/home/oscar/Documents/BD_COLUMNAR/tpch/tpch_2_17_0/dbgen# _
```

Los parámetros que estamos utilizando son:

- -s : Indica el factor de escala, es decir la cantidad de datos que se va a generar, en este caso si la escala es 1, se genera 1GB de datos, si fuera la escala de 3 se generan 3 GB de datos.
- -v: habilita la opción de verbosidad, es decir, que nos muestre comentarios de la operación mientras lo está generando, así podemos ver el estado en tiempo real de la ejecución.

Nota: este proceso puede ser tardado dependiendo de la capacidad de la computadora que se está utilizando, para agilizarlo se puede paralelizar el proceso utilizando el parámetro -C el cual indicamos el número de pedazos en los que se paralizarán, si se utiliza este parámetro también es necesario utilizar el comando -S para indicar cada uno de los pasos para generar los datos

Una vez generado los datos revisamos los archivos con el siguiente comando:

*du -ch *.tbl* | tail -1*

```
oscar@oscar-Satellite-C645:~/Documents/BD_COLUMNAR/tpch/tpch_2_17_0/dbgen$ du -ch *.tbl* | tail -1
1.1G    total
```

El comando du (Disk usage) nos permite conocer el espacio que ocupan los archivos, en este caso solo los que tengan la extensión *.tbl* y además como se utiliza el comando tail -1 indicamos que muestre los resultados en una sola línea por lo que sumará todos los resultados y los desplegara.

Ahora ya tenemos preparada nuestros datos de carga

Ambiente de desarrollo para el RDBMS, creación de tablespace y usuario.

Primero se crea el tablespace para la prueba, borramos la tablespace si existe:

drop tablespace BD_COLUMNAR including contents cascade constraints;
create tablespace BD_COLUMNAR;

Luego creamos el usuario que utilizaremos, borramos el usuario si existe:

drop user BD_COLUMNAR cascade;

create user BD_COLUMNAR identified by BD_COLUMNAR;

Asignamos la tablespace:

alter user BD_COLUMNAR default tablespace BD_COLUMNAR quota unlimited on BD_COLUMNAR;

Asignamos permisos al usuario:

grant CREATE SESSION, CREATE TABLE, DROP ANY TABLE, ALTER ANY TABLE, SELECT ANY TABLE, INSERT ANY TABLE, DELETE ANY TABLE, CREATE ANY VIEW, UPDATE ANY TABLE to BD_COLUMNAR;

Antes de crear las tablas nos conectamos con el usuario BD_COLUMNAR que creamos.

conn BD_COLUMNAR/BD_COLUMNAR

Definición de las tablas en el RDBMS

drop table region;

drop table nation;

drop table supplier;

drop table customer;

drop table orders;

```
drop table part;

drop table partsupp;

drop table lineitem;

//TABLA PART
CREATE TABLE PART(
    P_PARTKEY NUMBER(10) NOT NULL,
    P_NAME VARCHAR2(55) NOT NULL,
    P_MFGR VARCHAR2(25) NOT NULL,
    P_BRAND VARCHAR2(10) NOT NULL,
    P_TYPE VARCHAR2(25) NOT NULL,
    P_SIZE NUMBER(38) NOT NULL,
    P_CONTAINER VARCHAR2(10) NOT NULL,
    P_RETAILPRICE NUMBER NOT NULL,
    P_COMMENT VARCHAR2(23) NOT NULL
);
//TABLA REGION
CREATE TABLE REGION(
    R_REGIONKEY NUMBER(10) NOT NULL,
    R_NAME VARCHAR2(25) NOT NULL,
    R_COMMENT VARCHAR2(152) NOT NULL
);
//TABLA NATION
CREATE TABLE NATION(
    N_NATIONKEY NUMBER NOT NULL,
```

```
    N_NAME VARCHAR2(25) NOT NULL,
    N_REGIONKEY NUMBER NOT NULL,
    N_COMMENT VARCHAR2(152) NOT NULL
);
//TABLA SUPPLIER
CREATE TABLE SUPPLIER(
    S_SUPPKEY NUMBER NOT NULL,
    S_NAME VARCHAR2(25) NOT NULL,
    S_ADDRESS VARCHAR2(40) NOT NULL,
    S_NATIONKEY NUMBER NOT NULL,
    S_PHONE VARCHAR2(15) NOT NULL,
    S_ACCTBAL NUMBER NOT NULL,
    S_COMMENT VARCHAR2(101) NOT NULL
);
//TABLA PARTSUPP
CREATE TABLE PARTSUPP(
    PS_PARTKEY NUMBER(10) NOT NULL,
    PS_SUPPKEY NUMBER(10) NOT NULL,
    PS_AVAILQTY NUMBER(38) NOT NULL,
    PS_SUPPLYCOST NUMBER NOT NULL,
    PS_COMMENT VARCHAR2(199) NOT NULL
);
//TABLA CUSTOMER
CREATE TABLE CUSTOMER(
```

```
    C_CUSTKEY NUMBER(10) NOT NULL,
    C_NAME VARCHAR2(25) NOT NULL,
    C_ADDRESS VARCHAR2(40) NOT NULL,
    C_NATIONKEY NUMBER(10) NOT NULL,
    C_PHONE VARCHAR2(15) NOT NULL,
    C_ACCTBAL NUMBER NOT NULL,
    C_MKTSEGMENT VARCHAR2(10) NOT NULL,
    C_COMMENT VARCHAR2(117) NOT NULL
);
//TABLA ORDERS
CREATE TABLE ORDERS(
    O_ORDERKEY NUMBER(10) NOT NULL,
    O_CUSTKEY NUMBER(10) NOT NULL,
    O_ORDERSTATUS CHAR(1) NOT NULL,
    O_TOTALPRICE NUMBER NOT NULL,
    O_ORDERDATE VARCHAR2(10) NOT NULL,
    O_ORDERPRIORITY VARCHAR2(15) NOT NULL,
    O_CLERK VARCHAR2(15) NOT NULL,
    O_SHIPPRIORITY NUMBER(38) NOT NULL,
    O_COMMENT VARCHAR2(79) NOT NULL
);
//TABLA LINEITEM
CREATE TABLE LINEITEM(
    L_ORDERKEY NUMBER(10) NOT NULL,
```

```
    L_PARTKEY NUMBER(10) NOT NULL,
    L_SUPPKEY NUMBER(10) NOT NULL,
    L_LINENUMBER NUMBER(38) NOT NULL,
    L_QUANTITY NUMBER NOT NULL,
    L_EXTENDEDPRICE NUMBER NOT NULL,
    L_DISCOUNT NUMBER NOT NULL,
    L_TAX NUMBER NOT NULL,
    L_RETURNFLAG CHAR(1) NOT NULL,
    L_LINESTATUS CHAR(1) NOT NULL,
    L_SHIPDATE VARCHAR2(10) NOT NULL,
    L_COMMITDATE VARCHAR2(10) NOT NULL,
    L_RECEIPTDATE VARCHAR2(10) NOT NULL,
    L_SHIPINSTRUCT VARCHAR2(25) NOT NULL,
    L_SHIPMODE VARCHAR2(10) NOT NULL,
    L_COMMENT VARCHAR2(44) NOT NULL
);
```

No se han agregado los constraints a las tablas para no perder rendimiento a la hora de cargar los datos, una vez que se carguen los datos se procede a agregarle los constraints a las tablas.

Carga de datos

Para cargar los datos en el RDBMS utilizaremos un archivo con extensión .ctl por cada tabla que creamos para indicar cómo se va a realizar la inserción de los datos. Estos archivos deben estar en la misma dirección (o carpeta) en la que se tienen los archivos .tbl generadas a partir del dbgen.

```
//CTL PART

load data

INFILE 'part.tbl'

INTO TABLE  PART

FIELDS TERMINATED BY '|'

(P_PARTKEY,  P_NAME,  P_MFGR,  P_BRAND,  P_TYPE,  P_SIZE,  P_CONTAINER,
P_RETAILPRICE, P_COMMENT)


//CTL NATION

load data

INFILE 'nation.tbl'

INTO TABLE  NATION

FIELDS TERMINATED BY '|'

(N_NATIONKEY, N_NAME, N_REGIONKEY, N_COMMENT)


//CTL ORDERS

load data

INFILE 'orders.tbl'

INTO TABLE  ORDERS

FIELDS TERMINATED BY '|'

(O_ORDERKEY,  O_CUSTKEY,  O_ORDERSTATUS,  O_TOTALPRICE,
O_ORDERDATE, O_ORDERPRIORITY, O_CLERK, O_SHIPPRIORITY, O_COMMENT)


//CTL REGION

load data

INFILE 'region.tbl'

INTO TABLE  REGION
```

FIELDS TERMINATED BY '|'

(R_REGIONKEY, R_NAME, R_COMMENT)

//CTL CUSTOMER

load data

INFILE 'customer.tbl'

INTO TABLE CUSTOMER

FIELDS TERMINATED BY '|'

(C_CUSTKEY, C_NAME, C_ADDRESS, C_NATIONKEY, C_PHONE, C_ACCTBAL, C_MKTSEGMENT, C_COMMENT)

//CTL LINEITEM

load data

INFILE 'lineitem.tbl'

INTO TABLE LINEITEM

FIELDS TERMINATED BY '|'

(L_ORDERKEY, L_PARTKEY, L_SUPPKEY, L_LINENUMBER, L_QUANTITY, L_EXTENDEDPRICE, L_DISCOUNT, L_TAX, L_RETURNFLAG, L_LINESTATUS, L_SHIPDATE, L_COMMITDATE, L_RECEIPTDATE, L_SHIPINSTRUCT, L_SHIPMODE, L_COMMENT)

//CTL PARTSUPP

load data

INFILE 'partsupp.tbl'

INTO TABLE PARTSUPP

FIELDS TERMINATED BY '|'

(PS_PARTKEY, PS_SUPPKEY, PS_AVAILQTY, PS_SUPPLYCOST, PS_COMMENT)

//CTL SUPPLIER

load data

INFILE 'supplier.tbl'

INTO TABLE SUPPLIER

FIELDS TERMINATED BY '|'

(S_SUPPKEY, S_NAME, S_ADDRESS, S_NATIONKEY, S_PHONE, S_ACCTBAL, S_COMMENT)

Una vez que se tienen cada uno de los archivos, se pasa a cargar los datos, para ello tenemos que utilizar los siguientes comandos, dentro de la carpeta donde se encuentran los archivos .ctl:

sqlldr <username>/<password> control = <filename>

//Para PART

sqlldr BD_COLUMNAR/BD_COLUMNAR control = part.ctl

//Para NATION

sqlldr BD_COLUMNAR/BD_COLUMNAR control = nation.ctl

//Para ORDERS

sqlldr BD_COLUMNAR/BD_COLUMNAR control = orders.ctl

//Para REGION

sqlldr BD_COLUMNAR/BD_COLUMNAR control = region.ctl

//Para CUSTOMER

sqlldr BD_COLUMNAR/BD_COLUMNAR control = customer.ctl

//Para LINEITEM

sqlldr BD_COLUMNAR/BD_COLUMNAR control = lineitem.ctl

//Para PARTSUPP

sqlldr BD_COLUMNAR/BD_COLUMNAR control = partsupp.ctl

//Para SUPPLIER

sqlldr BD_COLUMNAR/BD_COLUMNAR control = supplier.ctl

Aplicar constrains a las tablas cargadas

Cuando las tablas están cargadas, aplicamos los constrains a las tablas:

Llaves primarias

//Para REGION

ALTER TABLE REGION ADD CONSTRAINT REGION_PK PRIMARY KEY (r_regionkey);

//Para NATION

ALTER TABLE NATION ADD CONSTRAINT NATION_PK PRIMARY KEY (n_nationkey);

//Para SUPPLIER

ALTER TABLE SUPPLIER ADD CONSTRAINT SUPPLIER_PK PRIMARY KEY (s_suppkey);

//Para PARTSUPP

CREATE UNIQUE INDEX partsupp_pk ON partsupp(ps_partkey,ps_suppkey) parallel 2;

ALTER TABLE PARTSUPP ADD CONSTRAINT PARTSUPP_PK PRIMARY KEY(ps_partkey,ps_suppkey) using index PARTSUPP_PK;

//Para PART

CREATE UNIQUE INDEX PART_PK ON PART(p_partkey) parallel 2;

ALTER TABLE PART ADD CONSTRAINT PART_PK PRIMARY KEY (p_partkey) using index PART_PK;

//Para ORDERS

CREATE UNIQUE INDEX ORDERS_PK ON ORDERS(o_orderkey) parallel 2;

ALTER TABLE ORDERS ADD CONSTRAINT ORDERS_PK PRIMARY KEY (o_orderkey) using index ORDERS_PK;

//Para LINEITEM

CREATE UNIQUE INDEX LINEITEM_PK ON LINEITEM(l_linenumber, l_orderkey) parallel 2;

ALTER TABLE LINEITEM ADD CONSTRAINT LINEITEM_PK PRIMARY KEY (l_linenumber, l_orderkey) using index LINEITEM_PK;

//Para CUSTOMER

CREATE UNIQUE INDEX CUSTOMER_PK ON CUSTOMER(c_custkey) parallel 2;

ALTER TABLE CUSTOMER ADD CONSTRAINT CUSTOMER_PK PRIMARY KEY (c_custkey) using index CUSTOMER_PK;

Llaves foráneas

//Llave foránea de LINEITEM que referencia PARTSUPP

ALTER TABLE LINEITEM ADD CONSTRAINT LINEITEM_PARTSUPP_FK FOREIGN KEY (l_partkey, l_suppkey) REFERENCES PARTSUPP(ps_partkey, ps_suppkey) NOT DEFERRABLE;

//Llave foránea de ORDERS que referencia CUSTOMER

ALTER TABLE ORDERS ADD CONSTRAINT ORDER_CUSTOMER_FK FOREIGN KEY (o_custkey) REFERENCES CUSTOMER (c_custkey) NOT DEFERRABLE;

//Llave foránea de PARTSUPP que referencia PART

ALTER TABLE PARTSUPP ADD CONSTRAINT PARTSUPP_PART_FK FOREIGN KEY (ps_partkey) REFERENCES PART (p_partkey) NOT DEFERRABLE;

//Llave foránea de PARTSUPP que referencia a SUPPLIER

ALTER TABLE PARTSUPP ADD CONSTRAINT PARTSUPP_SUPPLIER_FK FOREIGN KEY (ps_suppkey) REFERENCES SUPPLIER (s_suppkey) NOT DEFERRABLE;

//Llave foránea de SUPPLIER que referencia a NATION

ALTER TABLE SUPPLIER ADD CONSTRAINT SUPPLIER_NATION_FK FOREIGN KEY (s_nationkey) REFERENCES NATION (n_nationkey) NOT DEFERRABLE;

//Llave foránea de CUSTOMER que referencia a NATION

ALTER TABLE CUSTOMER ADD CONSTRAINT CUSTOMER_NATION_FK FOREIGN KEY (c_nationkey) REFERENCES NATION (n_nationkey) NOT DEFERRABLE;

//Llave foránea de NATION que referencia a REGION

ALTER TABLE NATION ADD CONSTRAINT NATION_REGION_FK FOREIGN KEY (n_regionkey) REFERENCES REGION (r_regionkey) NOT DEFERRABLE;

//Llave foránea de LINEITEM que referencia a ORDERS

ALTER TABLE LINEITEM ADD CONSTRAINT LINEITEM_ORDER_FK FOREIGN KEY (l_orderkey) REFERENCES ORDERS (o_orderkey) NOT DEFERRABLE;

Una vez terminado se tiene listo nuestro ambiente para realizar el trabajo de pruebas.

Instalar el CDBMS

Descargaremos el CDBMS de la siguiente dirección: http://scn.sap.com/community/developer-center/analytic-server, elegimos alguna de las dos versiones, en este caso utilizamos la versión de 30 días de prueba. Nos registramos para poder descargar el producto.

Para la instalación seguiremos la referencia de la siguiente página http://www.petersap.nl/SybaseWiki/index.php?title=Installation_guidelines_IQ_16, en la cual explica como instalar el CDBMS en Ubuntu 14.04, como se nos explica en la página, el CDBMS no es soportado de manera normal para Ubuntu 14.04, ya que en la documentación no se menciona, pero sí se puede trabajar en Ubuntu 16.04.

Primero se crean las carpetas y usuarios que se necesita para poder instalar el CDBMS

sudo mkdir /opt/sybase

sudo mkdir /var/sybase

En la carpeta */opt* es donde se instalará el DBMS, y en la carpeta */var* es donde se guardarán los archivos en dispositivos de tipo raw. Después se genera un grupo con un usuario para poder utilizar el manejador, para ello utilizaremos los siguientes comandos:

sudo groupadd BD_COLUMNAR

sudo useradd -g BD_COLUMNAR -d /opt/sybase sybaseIQ16

Una vez generados nuestros usuarios, les damos permiso sobre las carpetas creadas:

sudo chown sybaseIQ16:BD_COLUMNAR /opt/sybase

sudo chown sybaseIQ16:BD_COLUMNAR /var/sybase

Ahora tenemos que instalar estos paquetes para que pueda trabajar:

sudo apt-get install libaio1

sudo apt-get install csh

sudo apt-get install libxrender1

sudo apt-get install libxtst6

sudo apt-get install lib32z1

sudo apt-get install lib32ncurses5

sudo apt-get install lib32bz2-1.0

Luego se genera una contraseña al usuario *sybaseIQ16*, mediante los siguientes comandos, en este ejemplo le asignaremos la contraseña: *sybase*.

```
oscar@oscar-Satellite-C645:~$ sudo passwd sybaseIQ16
[sudo] password for oscar:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

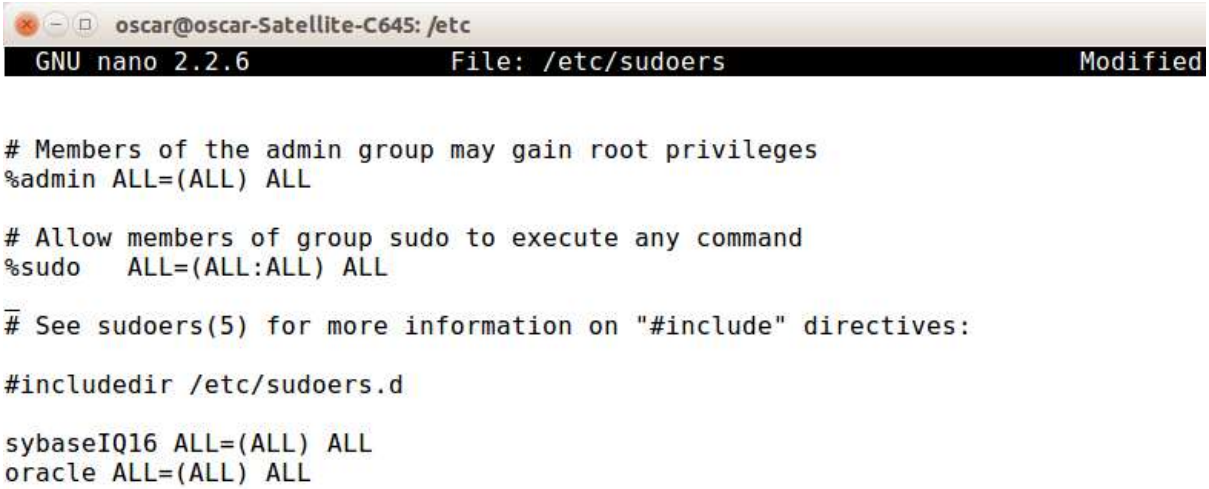
Ahora como configuración extra le damos el permiso de ejecutar el comando sudo modificando el archivo sudoers con el siguiente comando:

sudo nano /etc/sudoers

Y en el cual agregamos el siguiente renglón:

sybaseIQ16 ALL=(ALL:ALL) ALL

Como se muestra en el siguiente ejemplo:



Ahora cambiamos de usuario al usuario *sybaseIQ16* mediante el siguiente comando:

sudo su - sybaseIQ16

Se crean los siguientes directorios para instalar el manejador y accedemos a la carpeta *work* en la cual pondremos el archivo descargado de la página para la instalación del CDBMS:

mkdir work

mkdir iq16

cd work

sudo mv .../Downloads/iq160_LinuxAMD64.tgz /opt/sybase/work

Después se descomprime el archivo en la misma carpeta con el siguiente comando:

sudo tar -xf iq160_LinuxAMD64.tgz

Una vez que se tenga el archivo ejecutamos el archivo setup.bin con el siguiente comando:

sudo ./setup.bin

Con lo que se muestra la siguiente pantalla:

```
===============================================================================
Sybase IQ Server Suite                          (created with InstallAnywhere)
-------------------------------------------------------------------------------

Preparing CONSOLE Mode Installation...




===============================================================================
Title
-----

CONSOLE

PRESS <ENTER> TO CONTINUE: _
```

Seguimos los pasos de instalación, presionamos ENTER donde nos pregunte, cuando nos indique el lugar de instalación le indicamos el folder default, que es el que ya creamos previamente:

```
Where would you like to install?

  Default Install Folder: /opt/sybase/iq16

ENTER AN ABSOLUTE PATH, OR PRESS <ENTER> TO ACCEPT THE DEFAULT
      : _
```

Después elegimos la instalación típica:

```
Please choose the Install Set to be installed by this installer.

  ->1- Typical

    2- Customize...

ENTER THE NUMBER FOR THE INSTALL SET, OR PRESS <ENTER> TO ACCEPT THE DEFAULT
   : 1_
```

Luego se muestra la opción de elegir el tipo de licencia que queremos instalar la versión de prueba o la versión completa de sybase IQ 16, en este ejemplo elegiremos la versión de prueba que es de 30 días, los suficientes para realizar las pruebas.

```
What would you like to do?

    1- Install licensed copy of Sybase IQ Server Suite 16.0 sp01 (64-bit)

  ->2- Evaluate Sybase IQ Server Suite 16.0 sp01 (64-bit)

Enter one of the options above: 2_
```

Al término de este paso, elegimos la región en la que nos localizamos, en este ejemplo será México.

```
1)  Americas (Mid/So.) and Asia Pacif  2)  Argentina
3)  Australia                          4)  Belgium(English)
5)  Brazil                             6)  Canada
7)  Denmark                            8)  Europe,Middle East, and Africa - G
9)  France(English)                    10) France(French)
11) Germany(English)                   12) Hong Kong
13) India                              14) Italy(English)
15) Italy(Italy)                       16) Japan
17) Korea                              18) Malaysia
19) Mexico                             20) Netherlands
21) New Zealand                        22) Norway
23) People's Republic of China(PRC)    24) Singapore
25) Spain(English)                     26) Spain(Spanish)
27) Sweden                             28) Switzerland(English)
29) Taiwan                             30) United Kingdom
31) United States of America           32) Any Other Locations

Please enter the number of the location you are installing. (1-32) (DEFAULT:
   1): 19_
```

Después leemos los términos de condiciones de uso del software y aceptamos las condiciones si queremos instalarlo.

```
I agree to the terms of the Sybase license for the install location
   specified. (Y/N): Y_
```

Continuando con la instalación nos indicará la información de la instalación, el espacio que va a ocupar y características del software, después empezará el proceso de instalación del software.

```
================================================================================
Installing...
-------------

 [=================|=================|=================|=================]
 [---_
```

Una vez terminada la instalación aceptamos los puertos por default que el sistema va a utilizar:

```
Sybase Control Center Service
-----------------------------

HTTP Port: [integer (1025-65535)] (DEFAULT: 8282):
HTTPS Port: [integer (1025-65535)] (DEFAULT: 8283):
```

Y también agregamos las contraseñas para el administrador y al agente, en este caso será sybase para todas ellas:

```
Sybase Control Center - Security Login Modules
----------------------------------------------

Enter SCC administrator and SCC agent administrator users password.

SCC administrator password:
Confirm SCC administrator password:
SCC agent administrator password:
Confirm SCC agent administrator password:
```

Por último nos preguntará si queremos iniciar el CDBMS, pero no lo iniciamos todavía:

```
Do you want to start Sybase Control Center Server?

  ->1- Yes
    2- No

Enter one of the options above: 2_
```

Una vez terminada la instalación borramos la carpeta work, para liberar espacio, para ello utilizamos el siguiente comando:

sudo rm -R /opt/sybase/work

Configurar el ambiente del sistema

Para ello se utiliza el siguiente comando, para cambiar la fuente al archivo IQ.sh, en el caso de Ubuntu cambiamos la fuente utilizando “.”

. /opt/sybase/iq16/IQ.sh

Iniciar el servidor IQ

Para empezar se crea la carpeta donde guardaremos los archivos de la base de datos, se pueden utilizar raw devices o archivos normales.

mkdir /var/sybase/IQ1

cd /var/sybase/IQ1

mkdir iq_files

Después se asigna la variable *IQLOGDIR16* la carpeta donde se encuentran los logfiles, en el caso de que no se asigne se apunta a /opt/sybase/iq16/IQ-16_0/logfiles.

export IQLOGDIR16=/var/sybase/IQ1

Por último se indica el servidor IQ, usando el siguiente comando y especificamos el nombre del servidor usando el parámetro -n.

start_iq -n IQ1

Una vez terminada la ejecución de la operación veremos un resultado como este:

```
I. 03/31 21:29:17. Database server started at Thu Mar 31 2016 21:29
I. 03/31 21:29:17. Trying to start SharedMemory link ...
I. 03/31 21:29:17.     SharedMemory link started successfully
I. 03/31 21:29:17. Trying to start TCPIP link ...
I. 03/31 21:29:17. Starting on port 2638
I. 03/31 21:29:22.     TCPIP link started successfully
I. 03/31 21:29:22. Now accepting requests
New process id is 7080
-e
Server started successfully
```

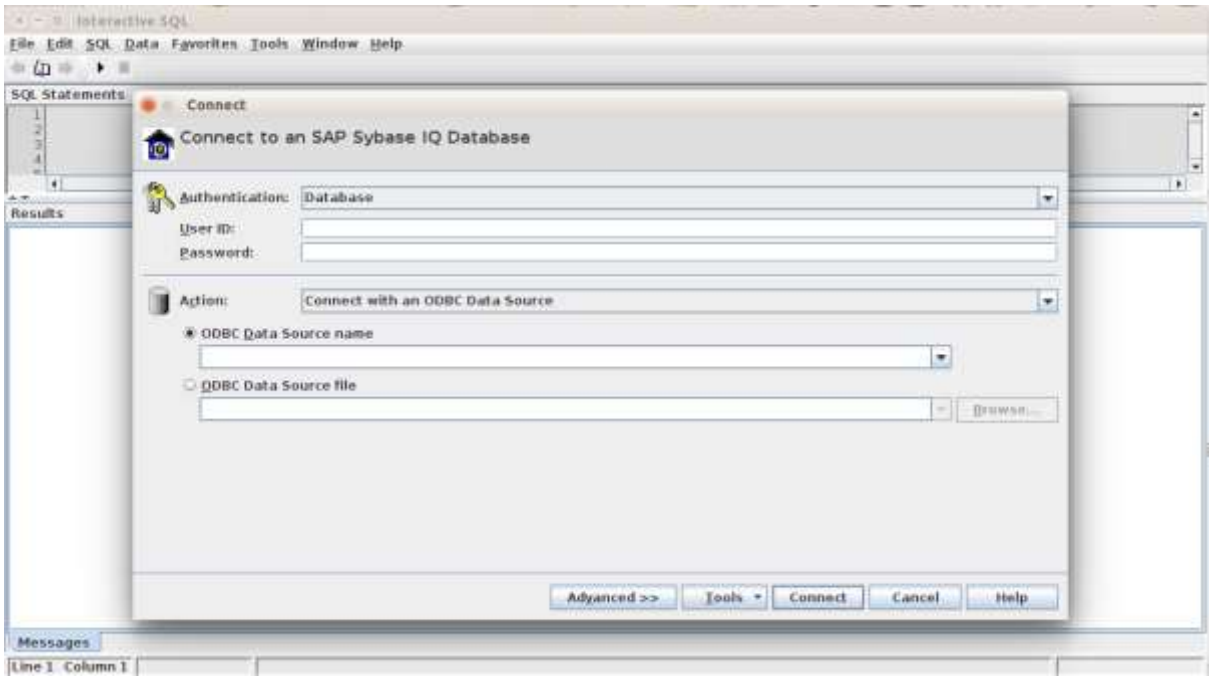
Conectarse al servidor

Primero habilitamos a todos los usuarios el uso de la interfaz gráfica con el siguiente comando en una nueva terminal:

xhost +

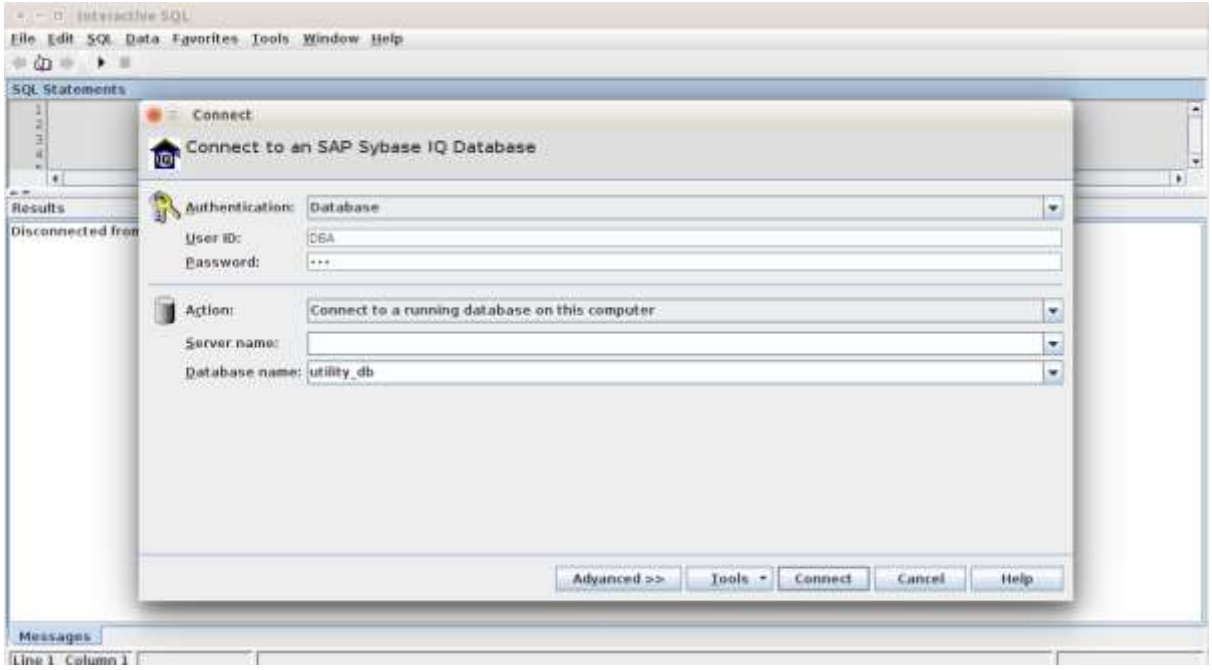
Para poder ver el asistente gráfico de sybase IQ utilizaremos el siguiente comando para conectarnos al servidor IQ con el usuario sybase, después de ejecutar el comando se muestra la siguiente interfaz:

dbisql



Creación de la base de datos

Utilizamos el usuario *"DBA*" para UserID, *"sql"* para el password, escogemos la opción de Action: *"Connect to a running database on this computer"* y en Database name utilizamos el nombre *"utility_db"*, seleccionamos las opciones como se muestra en pantalla.



Se crea la base de datos con el nombre IQ1, en este ejemplo se da las siguientes configuraciones de tamaño por razones de práctica pero estas se pueden configurar según sea el caso.

SQL Statements

```
1  create database 'IQ1.db'
2  transaction log on 'IQ1.log'
3  mirror 'IQ1.mirror'
4  message path 'IQ1.iqmsg'
5  iq path '/var/sybase/IQ1/iq_files/IQ1_01.iq'
6  iq size 5000
7  iq reserve 2000
8  temporary path '/var/sybase/IQ1/iq_files/IQ1_01.iqtmp'
9  temporary size 3000
10 temporary reserve 1000
11 dba user 'BDC'
12 dba password 'BDC'
13
```

create database 'IQ1'

transaction log on 'IQ1.log'

mirror 'IQ1.mirror'

message path 'IQ1.iqmsg'

iq path '/var/sybase/IQ1/iq_files/IQ1_01.iq'

iq size 5000

iq reserve 2000

temporary path '/var/sybase/IQ1/iq_files/IQ1_01.iqtmp'

temporary size 3000

temporary reserve 1000

dba user 'BDC'

dba password 'BDC'

Por último detenemos el servidor y salimos de dbisql

stop engine

exit

Una vez que se crea la base de datos procedemos a crear las tablas, llenarlas y aplicar al final las restricciones de llaves para un llenado más rápido.

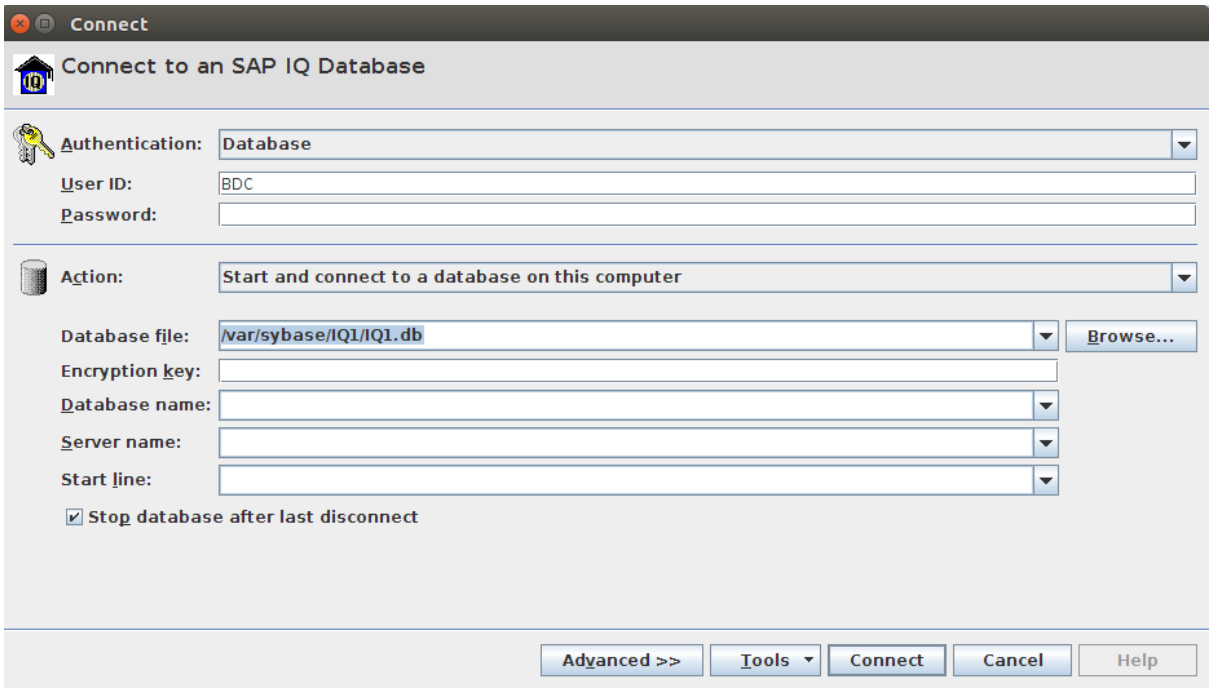
Primero se inicia nuevamente la base de datos pero ahora con el comando *start_iq*, pero con parámetros para incrementar el espacio de memoria que va a utilizar:

start_iq -n IQ1 -iqmc 4000 -iqtc 6000

Luego iniciamos de nuevo el ambiente gráfico:

dbisql

Para acceder a la base de datos creada se indica el nombre del usuario y la contraseña que se utilizó al crear la base de datos, luego se elige la opción *“Start and connect to a database on this computer”* en la opción *Action*, luego en la opción *Database file*, se escoge la base de datos creada, en este caso */var/sybase/IQ1/IQ1.db* y por último nos conectamos.



Ahora se realiza la construcción de las tablas de la base de datos, tienen la misma estructura que se utilizó para el RDBMS, solo se tienen cambios en los nombres de los tipos de datos.

//Eliminar tablas

drop table region;

drop table nation;

drop table supplier;

drop table customer;

drop table orders;

drop table part;

drop table partsupp;

drop table lineitem;

```
//TABLA PART
create table PART(
    P_PARTKEY NUMERIC(10) NOT NULL,
    P_NAME VARCHAR(55) NOT NULL,
    P_MFGR VARCHAR(25) NOT NULL,
    P_BRAND VARCHAR(10) NOT NULL,
    P_TYPE VARCHAR(25) NOT NULL,
    P_SIZE NUMERIC(38) NOT NULL,
    P_CONTAINER VARCHAR(10) NOT NULL,
    P_RETAILPRICE INT NOT NULL,
    P_COMMENT VARCHAR(23) NOT NULL
);
//TABLA REGION
create table REGION(
    R_REGIONKEY NUMERIC(10) NOT NULL,
    R_NAME VARCHAR(25) NOT NULL,
    R_COMMENT VARCHAR(152) NOT NULL
);
//TABLA NATION
CREATE TABLE NATION(
    N_NATIONKEY NUMERIC(10) NOT NULL,
    N_NAME VARCHAR(25) NOT NULL,
    N_REGIONKEY NUMERIC(10) NOT NULL,
    N_COMMENT VARCHAR(152) NOT NULL
);
//TABLA SUPPLIER
```

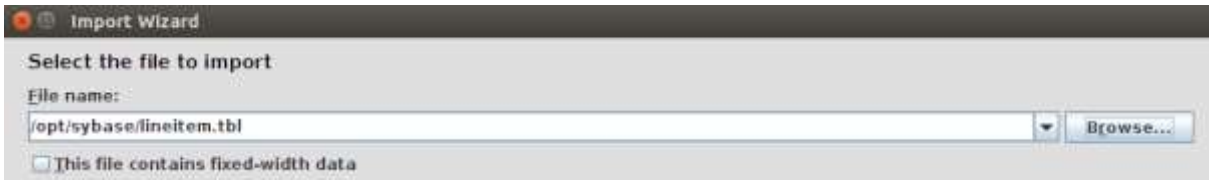
```
CREATE TABLE SUPPLIER(
    S_SUPPKEY NUMERIC (10) NOT NULL,
    S_NAME VARCHAR(25) NOT NULL,
    S_ADDRESS VARCHAR(40) NOT NULL,
    S_NATIONKEY NUMERIC(10) NOT NULL,
    S_PHONE VARCHAR(15) NOT NULL,
    S_ACCTBAL INT NOT NULL,
    S_COMMENT VARCHAR(101) NOT NULL
);
//TABLA PARTSUPP
CREATE TABLE PARTSUPP(
    PS_PARTKEY NUMERIC(10) NOT NULL,
    PS_SUPPKEY NUMERIC(10) NOT NULL,
    PS_AVAILQTY NUMERIC(38) NOT NULL,
    PS_SUPPLYCOST INT NOT NULL,
    PS_COMMENT VARCHAR(199) NOT NULL
);
//TABLA CUSTOMER
CREATE TABLE CUSTOMER(
    C_CUSTKEY NUMERIC(10) NOT NULL,
    C_NAME VARCHAR(25) NOT NULL,
    C_ADDRESS VARCHAR(40) NOT NULL,
    C_NATIONKEY NUMERIC(10) NOT NULL,
    C_PHONE VARCHAR(15) NOT NULL,
    C_ACCTBAL INT NOT NULL,
    C_MKTSEGMENT VARCHAR(10) NOT NULL,
```

```
    C_COMMENT VARCHAR(117) NOT NULL
);
//TABLA ORDERS
CREATE TABLE ORDERS(
    O_ORDERKEY NUMERIC(10) NOT NULL,
    O_CUSTKEY NUMERIC(10) NOT NULL,
    O_ORDERSTATUS CHAR(1) NOT NULL,
    O_TOTALPRICE INT NOT NULL,
    O_ORDERDATE VARCHAR(10) NOT NULL,
    O_ORDERPRIORITY VARCHAR(15) NOT NULL,
    O_CLERK VARCHAR(15) NOT NULL,
    O_SHIPPRIORITY NUMERIC(38) NOT NULL,
    O_COMMENT VARCHAR(79) NOT NULL
);
//TABLA LINEITEM
CREATE TABLE LINEITEM(
    L_ORDERKEY NUMERIC(10) NOT NULL,
    L_PARTKEY NUMERIC(10) NOT NULL,
    L_SUPPKEY NUMERIC(10) NOT NULL,
    L_LINENUMBER NUMERIC(38) NOT NULL,
    L_QUANTITY NUMERIC(15,2) NOT NULL,
    L_EXTENDEDPRICE NUMERIC(15,2) NOT NULL,
    L_DISCOUNT NUMERIC(15,2) NOT NULL,
    L_TAX NUMERIC(15,2) NOT NULL,
    L_RETURNFLAG CHAR(1) NOT NULL,
    L_LINESTATUS CHAR(1) NOT NULL,
```

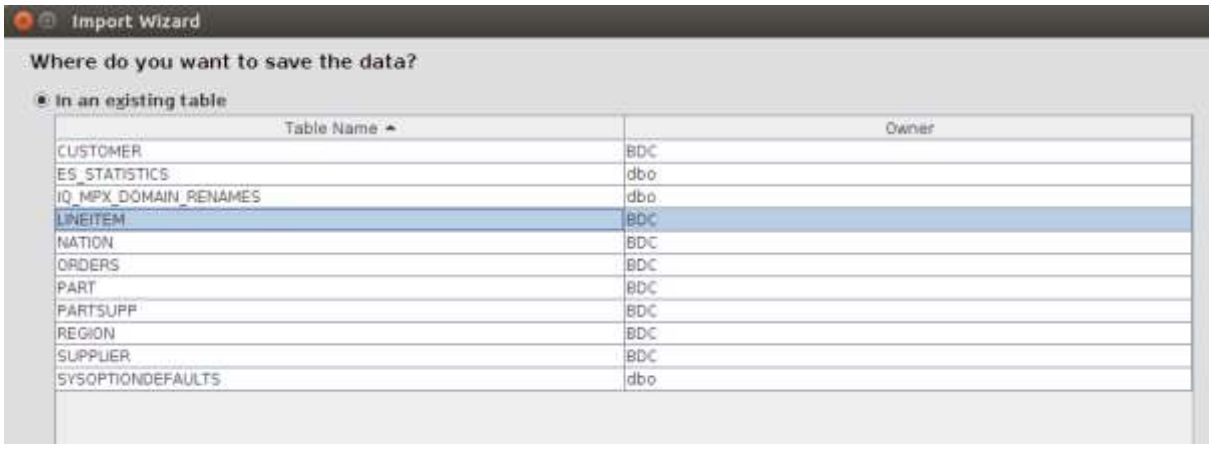
```
    L_SHIPDATE VARCHAR(10) NOT NULL,
    L_COMMITDATE VARCHAR(10) NOT NULL,
    L_RECEIPTDATE VARCHAR(10) NOT NULL,
    L_SHIPINSTRUCT VARCHAR(25) NOT NULL,
    L_SHIPMODE VARCHAR(10) NOT NULL,
    L_COMMENT VARCHAR(44) NOT NULL
);
```

Para agregar los datos a la base de datos primero se muestra cómo utilizar el método **Import** que nos proporciona el CDBMS, para ello en nuestra ventana nos vamos a la opción Data y elegimos la opción **Import**. Luego elegimos la opción “In a text file” para que podamos elegir la tabla que queramos importar.

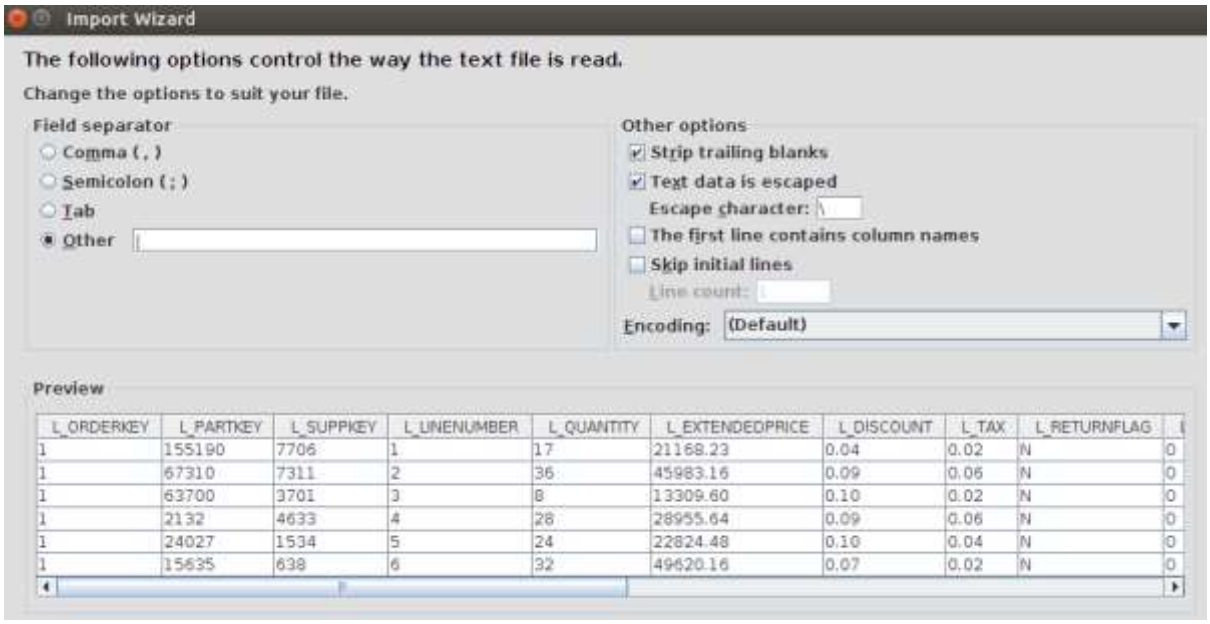
Luego se selecciona el archivo .tbl correspondiente.



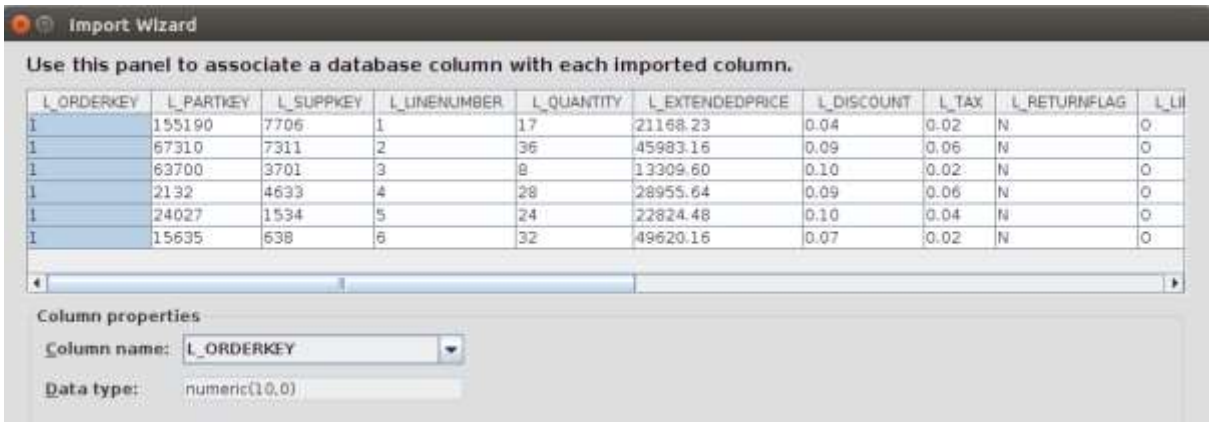
Se selecciona la tabla en la que se van a agregar los datos



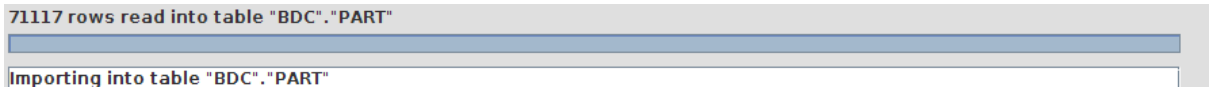
Luego se indica el separador para que encuentre los datos, en este caso seleccionamos la opción *"Other"* e introducimos el símbolo de separador en este ejercicio le agregamos el carácter | para separar los datos.



Verificamos que las columnas coincidan con los datos, y seleccionamos la opción *"Importar"*.



Por último esperamos a que termine a realizar la importación de los datos.



Cuando terminemos todos los datos estarán almacenados en la base de datos.

Un segundo método es usar la función LOAD TABLE, el cual es una forma de cargar datos masiva de manera rápida y sin generar log de transacción, la operación LOAD TABLE del CDBMS, es una operación equivalente a realizar un Bulk Copy. Cabe mencionar que este método tiene un mejor rendimiento si no se tiene restricciones en las tablas y cada vez que se utiliza esta operación se agregan nuevos registros.

//Sentencia LOAD TABLE la tabla PART

```
LOAD TABLE PART (
        P_PARTKEY,
        P_NAME,
        P_MFGR,
        P_BRAND,
        P_TYPE,
        P_SIZE,
        P_CONTAINER,
```

```
        P_RETAILPRICE,
        P_COMMENT)
FROM '/opt/sybase/part.tbl'
QUOTES OFF
ESCAPES OFF
DELIMITED BY '|'
//Sentencia LOAD TABLE para la tabla REGION
LOAD TABLE REGION (
        R_REGIONKEY,
        R_NAME,
        R_COMMENT)
FROM '/opt/sybase/region.tbl'
QUOTES OFF
ESCAPES OFF
DELIMITED BY '|'
//Sentencia LOAD TABLE para la tabla NATION
LOAD TABLE NATION (
   N_NATIONKEY,
   N_NAME,
   N_REGIONKEY,
   N_COMMENT)
FROM '/opt/sybase/nation.tbl'
QUOTES OFF
ESCAPES OFF
DELIMITED BY '|'
//Sentencia LOAD TABLE para la tabla SUPPLIER
```

```
LOAD TABLE SUPPLIER (
   S_SUPPKEY,
   S_NAME,
   S_ADDRESS,
   S_NATIONKEY,
   S_PHONE,
   S_ACCTBAL,
   S_COMMENT)
FROM '/opt/sybase/supplier.tbl'
QUOTES OFF
ESCAPES OFF
DELIMITED BY '|'
//Sentencia LOAD TABLE para la tabla PARTSUPP
LOAD TABLE PARTSUPP (
   PS_PARTKEY,
   PS_SUPPKEY,
   PS_AVAILQTY,
   PS_SUPPLYCOST,
   PS_COMMENT)
FROM '/opt/sybase/partsupp.tbl'
QUOTES OFF
ESCAPES OFF
DELIMITED BY '|'
//Sentencia LOAD TABLE para la tabla CUSTOMERS
LOAD TABLE CUSTOMER (
   C_CUSTKEY,
```

```
    C_NAME,
    C_ADDRESS,
    C_NATIONKEY ,
    C_PHONE,
     C_ACCTBAL,
    C_MKTSEGMENT,
    C_COMMENT)
FROM '/opt/sybase/customer.tbl'
QUOTES OFF
ESCAPES OFF
DELIMITED BY '|'
//Sentencia LOAD TABLE para la tabla ORDERS
LOAD TABLE ORDERS (
    O_ORDERKEY,
    O_CUSTKEY,
    O_ORDERSTATUS,
    O_TOTALPRICE,
    O_ORDERDATE,
    O_ORDERPRIORITY,
    O_CLERK,
    O_SHIPPRIORITY,
    O_COMMENT)
FROM '/opt/sybase/orders.tbl'
QUOTES OFF
ESCAPES OFF
DELIMITED BY '|'
```

```
//Sentencia LOAD TABLE para la tabla LINEITEM
LOAD TABLE LINEITEM (
   L_ORDERKEY,
   L_PARTKEY,
   L_SUPPKEY,
   L_LINENUMBER,
   L_QUANTITY,
   L_EXTENDEDPRICE,
   L_DISCOUNT,
   L_TAX,
   L_RETURNFLAG,
   L_LINESTATUS,
   L_SHIPDATE,
   L_COMMITDATE,
   L_RECEIPTDATE,
   L_SHIPINSTRUCT,
   L_SHIPMODE,
    L_COMMENT)
FROM '/opt/sybase/lineitem.tbl'
QUOTES OFF
ESCAPES OFF
DELIMITED BY '|'
```

Una vez realizando el llenado de datos aplicamos las restricciones para las tablas.

Restricciones de llaves primarias y foráneas en el CDBMS.

Llaves primarias

//Para REGION

ALTER TABLE REGION ADD CONSTRAINT REGION_PK PRIMARY KEY (r_regionkey);

//Para NATION

ALTER TABLE NATION ADD CONSTRAINT NATION_PK PRIMARY KEY (n_nationkey);

//Para SUPPLIER

ALTER TABLE SUPPLIER ADD CONSTRAINT SUPPLIER_PK PRIMARY KEY (s_suppkey);

//Para PARTSUPP

ALTER TABLE PARTSUPP ADD CONSTRAINT PARTSUPP_PK PRIMARY KEY(ps_partkey,ps_suppkey);

//Para PART

ALTER TABLE PART ADD CONSTRAINT PART_PK PRIMARY KEY (p_partkey);

//Para ORDERS

ALTER TABLE ORDERS ADD CONSTRAINT ORDERS_PK PRIMARY KEY (o_orderkey);

//Para LINEITEM

ALTER TABLE LINEITEM ADD CONSTRAINT LINEITEM_PK PRIMARY KEY (l_linenumber, l_orderkey);

//Para CUSTOMER

ALTER TABLE CUSTOMER ADD CONSTRAINT CUSTOMER_PK PRIMARY KEY (c_custkey);

Llaves foráneas

//Llave foránea de ORDERS que referencia CUSTOMER

ALTER TABLE ORDERS ADD CONSTRAINT ORDER_CUSTOMER_FK FOREIGN KEY (o_custkey) REFERENCES CUSTOMER (c_custkey);

//Llave foránea de PARTSUPP que referencia PART

ALTER TABLE PARTSUPP ADD CONSTRAINT PARTSUPP_PART_FK FOREIGN KEY (ps_partkey) REFERENCES PART (p_partkey);

//Llave foránea de PARTSUPP que referencia a SUPPLIER

ALTER TABLE PARTSUPP ADD CONSTRAINT PARTSUPP_SUPPLIER_FK FOREIGN KEY (ps_suppkey) REFERENCES SUPPLIER (s_suppkey);

//Llave foránea de SUPPLIER que referencia a NATION

ALTER TABLE SUPPLIER ADD CONSTRAINT SUPPLIER_NATION_FK FOREIGN KEY (s_nationkey) REFERENCES NATION (n_nationkey);

//Llave foránea de CUSTOMER que referencia a NATION

ALTER TABLE CUSTOMER ADD CONSTRAINT CUSTOMER_NATION_FK FOREIGN KEY (c_nationkey) REFERENCES NATION (n_nationkey);

//Llave foránea de NATION que referencia a REGION

ALTER TABLE NATION ADD CONSTRAINT NATION_REGION_FK FOREIGN KEY (n_regionkey) REFERENCES REGION (r_regionkey);

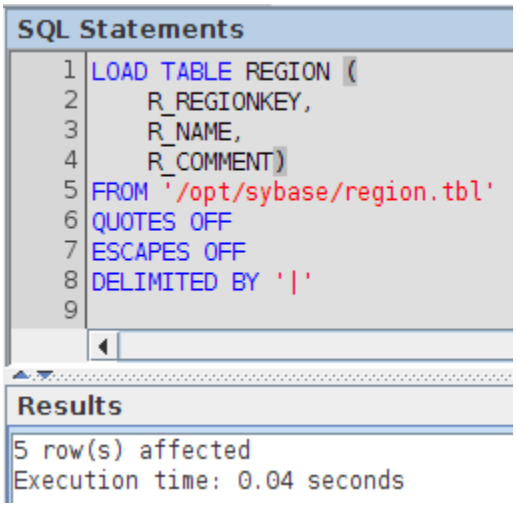
//Llave foránea de LINEITEM que referencia a ORDERS

ALTER TABLE LINEITEM ADD CONSTRAINT LINEITEM_ORDER_FK FOREIGN KEY (l_orderkey) REFERENCES ORDERS (o_orderkey);

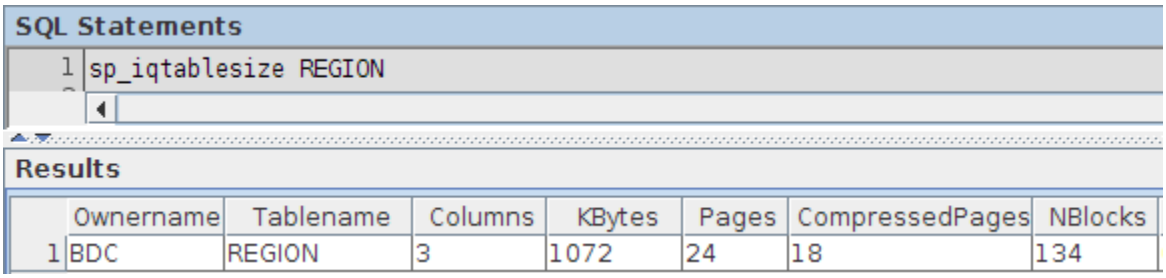
Una vez terminado se tiene listo nuestro ambiente para realizar el trabajo de pruebas.

REGISTRO DE TIEMPOS DE LLENADO:

//LLENADO LOAD TABLE CDBMS TABLA REGION



//ESPACIO UTILIZADO TABLA REGION CDBMS



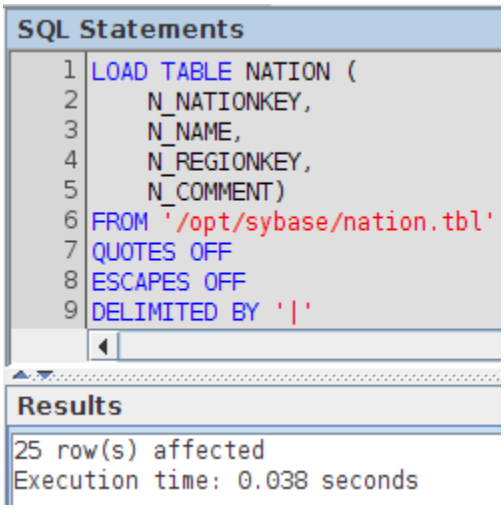
//LLENADO SQLLDR RDBMS TABLA REGION

```
Space allocated for bind array:                  49536 bytes(64 rows)
Read   buffer bytes: 1048576

Total logical records skipped:          0
Total logical records read:             5
Total logical records rejected:         0
Total logical records discarded:        0

Run began on Mon Mar 07 21:21:16 2016
Run ended on Mon Mar 07 21:21:16 2016

Elapsed time was:     00:00:00.19
CPU time was:         00:00:00.05
```

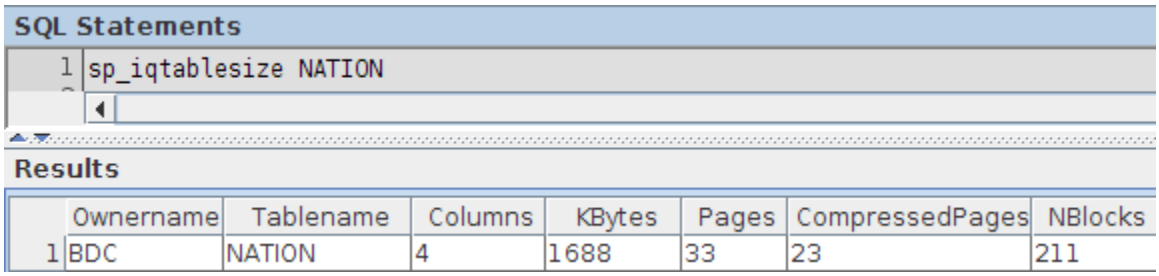
//ESPACIO UTILIZADO TABLA REGION RDBMS

```
SQL> SELECT
    lower(table_name)  AS table_name
    ,num_rows
    ,blocks*8      AS size_kb
    ,blocks
    ,compression
FROM
        all_tables
WHERE
        owner = 'BD_COLUMNAR'
        and table_name = 'REGION';  2    3    4    5    6    7    8    9
 10   11

TABLE_NAME
--------------------------------------------------------------------------------
------
  NUM_ROWS    SIZE_KB     BLOCKS COMPRESS
---------- ---------- ---------- --------
region
         5         40          5 DISABLED


SQL>
```

//LLENADO LOAD TABLE CDBMS TABLA NATION



//ESPACIO UTILIZADO TABLA NATION CDBMS



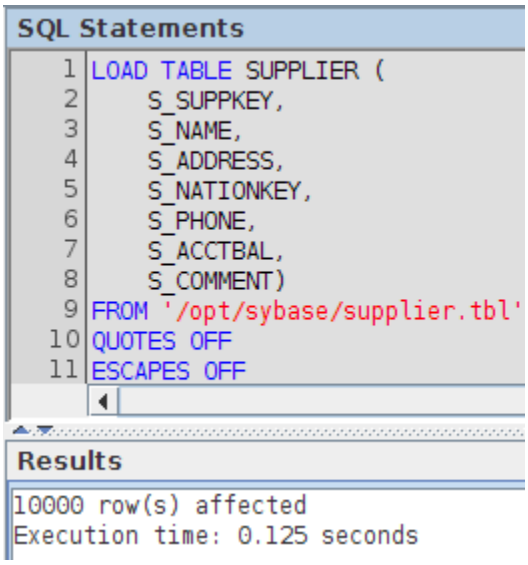
//LLENADO SQLLDR RDBMS TABLA NATION

```
Space allocated for bind array:                  66048 bytes(64 rows)
Read   buffer bytes: 1048576

Total logical records skipped:          0
Total logical records read:            25
Total logical records rejected:         0
Total logical records discarded:        0

Run began on Mon Mar 07 20:50:57 2016
Run ended on Mon Mar 07 20:50:57 2016

Elapsed time was:     00:00:00.48
CPU time was:         00:00:00.04
```

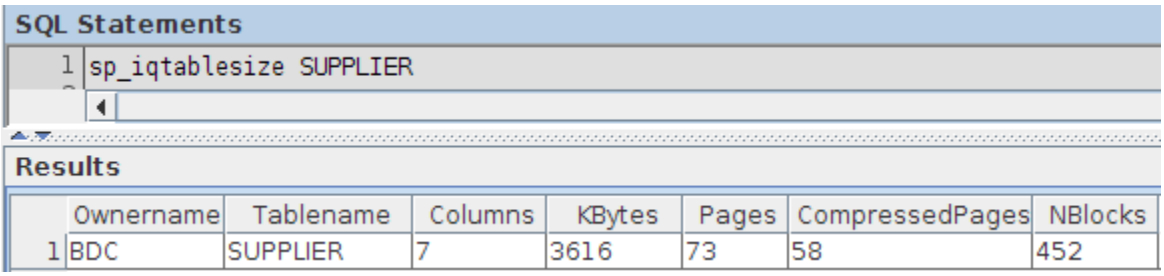
//ESPACIO UTILIZADO TABLA NATION RDBMS

```
SQL> SELECT
    lower(table_name)  AS table_name
    ,num_rows
    ,blocks*8      AS size_kb
    ,blocks
    ,compression
FROM
        all_tables
WHERE
        owner = 'BD_COLUMNAR'
        and table_name = 'NATION';  2    3    4    5    6    7    8    9
 10   11

TABLE_NAME
--------------------------------------------------------------------------------
------
  NUM_ROWS    SIZE_KB     BLOCKS COMPRESS
---------- ---------- ---------- --------
nation
        25         40          5 DISABLED


SQL>
```

//LLENADO LOAD TABLE CDBMS TABLA SUPPLIER



//ESPACIO UTILIZADO TABLA SUPPLIER CDBMS



SQL Statements

```
1 sp_iqtablesize SUPPLIER
```

Results

	Ownername	Tablename	Columns	KBytes	Pages	CompressedPages	NBlocks
1	BDC	SUPPLIER	7	3616	73	58	452

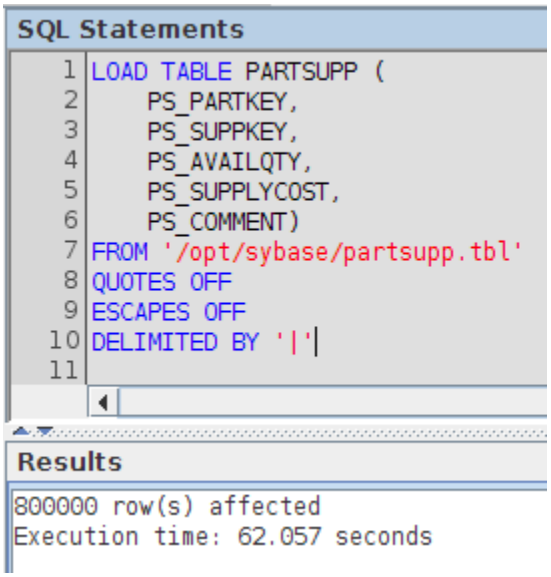
//LLENADO SQLLDR RDBMS TABLA SUPPLIER

```
Space allocated for bind array:                 115584 bytes(64 rows)
Read   buffer bytes: 1048576

Total logical records skipped:          0
Total logical records read:         10000
Total logical records rejected:         0
Total logical records discarded:        0

Run began on Tue Mar 08 20:46:51 2016
Run ended on Tue Mar 08 20:46:59 2016

Elapsed time was:     00:00:07.81
CPU time was:         00:00:00.16
```

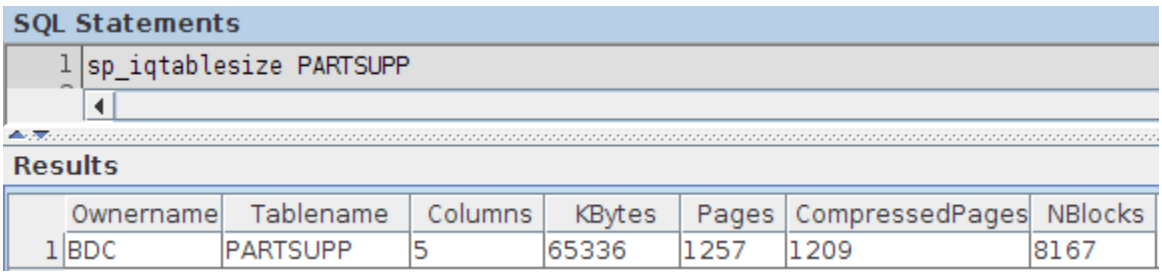
//ESPACIO UTILIZADO TABLA SUPPLIER RDBMS

```
SQL> SELECT
    lower(table_name)  AS table_name
    ,num_rows
    ,blocks*8      AS size_kb
    ,blocks
    ,compression
FROM
        all_tables
WHERE
        owner = 'BD_COLUMNAR'
        and table_name = 'SUPPLIER';  2    3    4    5    6    7    8    9
   10   11

TABLE_NAME
--------------------------------------------------------------------------------
------
  NUM_ROWS    SIZE_KB     BLOCKS COMPRESS
---------- ---------- ---------- --------
supplier
     10000       1952        244 DISABLED


SQL>
```

//LLENADO LOAD TABLE CDBMS TABLA PARTSUPP



//ESPACIO UTILIZADO TABLA PARTSUPP CDBMS



	Ownername	Tablename	Columns	KBytes	Pages	CompressedPages	NBlocks
1	BDC	PARTSUPP	5	65336	1257	1209	8167

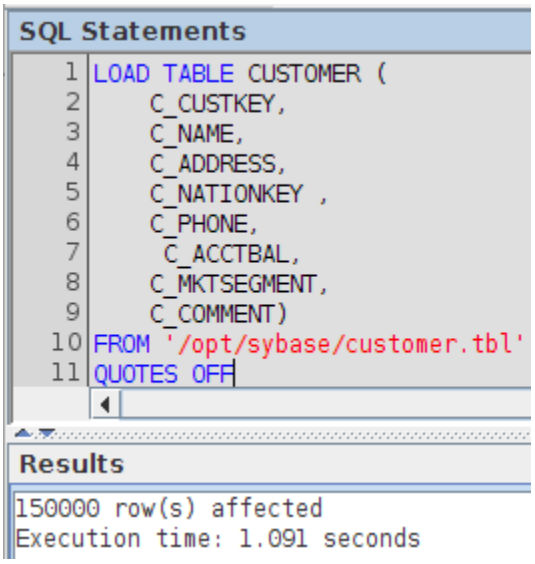
//LLENADO SQLLDR RDBMS TABLA PARTSUPP

```
Space allocated for bind array:                  82560 bytes(64 rows)
Read   buffer bytes: 1048576

Total logical records skipped:          0
Total logical records read:        800000
Total logical records rejected:         0
Total logical records discarded:        0

Run began on Sat Jun 18 21:11:52 2016
Run ended on Sat Jun 18 21:20:23 2016

Elapsed time was:     00:08:31.03
CPU time was:         00:00:09.44
```

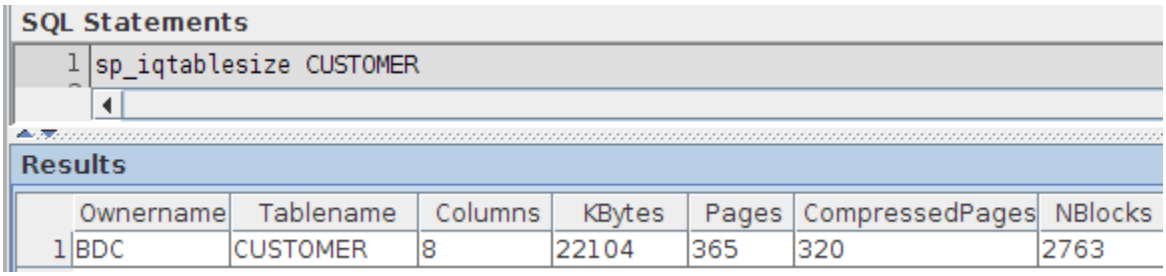
//ESPACIO UTILIZADO TABLA PARTSUPP RDBMS

```
SQL> SELECT
    lower(table_name)  AS table_name
    ,num_rows
    ,blocks*8      AS size_Kb
    ,blocks
    ,compression
FROM
        all_tables
WHERE
        owner = 'BD_COLUMNAR'
        and table_name = 'PARTSUPP';  2    3    4    5    6    7    8    9
   10   11

TABLE_NAME
--------------------------------------------------------------------------------
------
  NUM_ROWS    SIZE_KB     BLOCKS COMPRESS
---------- ---------- ---------- --------
partsupp
    800000     137896      17237 DISABLED


SQL>
```

//LLENADO LOAD TABLE CDBMS TABLA CUSTOMER



//ESPACIO UTILIZADO TABLA CUSTOMER CDBMS



SQL Statements

```
1 sp_iqtablesize CUSTOMER
```

Results

	Ownername	Tablename	Columns	KBytes	Pages	CompressedPages	NBlocks
1	BDC	CUSTOMER	8	22104	365	320	2763

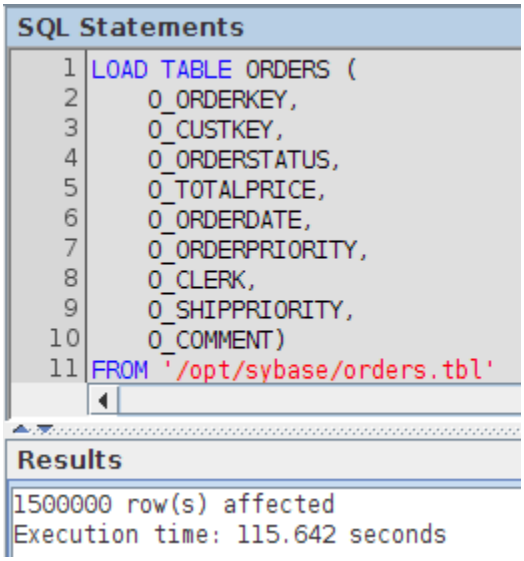
//LLENADO SQLLDR RDBMS TABLA CUSTOMER

```
Space allocated for bind array:                 132096 bytes(64 rows)
Read   buffer bytes: 1048576

Total logical records skipped:          0
Total logical records read:        150000
Total logical records rejected:         0
Total logical records discarded:        0

Run began on Mon Mar 07 21:25:09 2016
Run ended on Mon Mar 07 21:26:35 2016

Elapsed time was:     00:01:25.94
CPU time was:         00:00:02.20
```

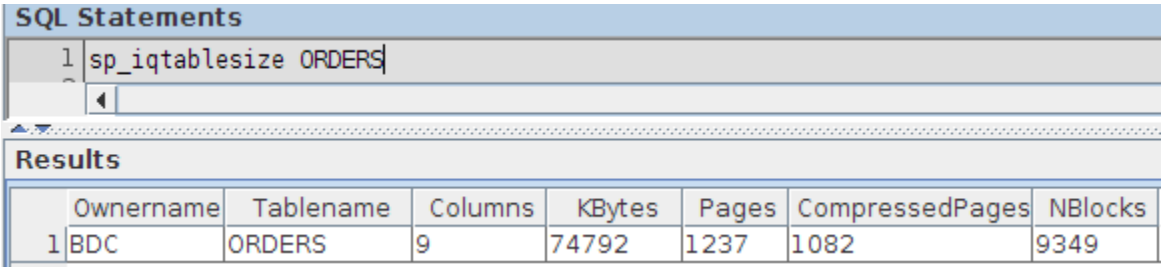
//ESPACIO UTILIZADO TABLA CUSTOMER RDBMS

```
SQL> SELECT
    lower(table_name)  AS table_name
    ,num_rows
    ,blocks*8      AS size_kb
    ,blocks
    ,compression
FROM
        all_tables
WHERE
        owner = 'BD_COLUMNAR'
        and table_name = 'CUSTOMER';  2    3    4    5    6    7    8    9
   10   11

TABLE_NAME
--------------------------------------------------------------------------------
------
  NUM_ROWS    SIZE_KB     BLOCKS COMPRESS
---------- ---------- ---------- --------
customer
    150000      27152       3394 DISABLED


SQL>
```

//LLENADO LOAD TABLE CDBMS TABLA ORDERS



//ESPACIO UTILIZADO TABLA ORDERS CDBMS



	Ownername	Tablename	Columns	KBytes	Pages	CompressedPages	NBlocks
1	BDC	ORDERS	9	74792	1237	1082	9349

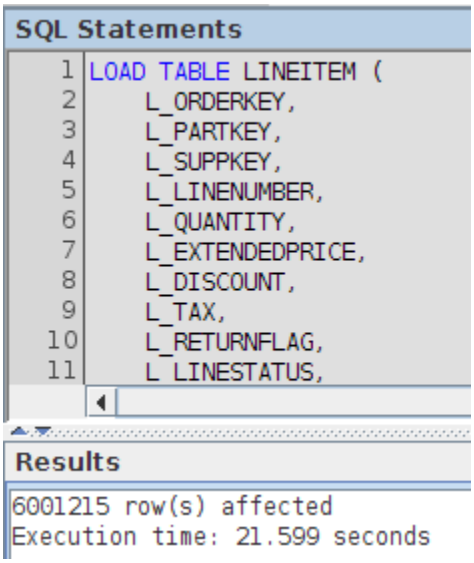
//LLENADO SQLLDR RDBMS TABLA ORDERS

```
Space allocated for bind array:                148608 bytes(64 rows)
Read   buffer bytes: 1048576

Total logical records skipped:          0
Total logical records read:       1500000
Total logical records rejected:         0
Total logical records discarded:        0

Run began on Mon Mar 07 21:02:25 2016
Run ended on Mon Mar 07 21:18:04 2016

Elapsed time was:     00:15:39.74
CPU time was:         00:00:21.05
```

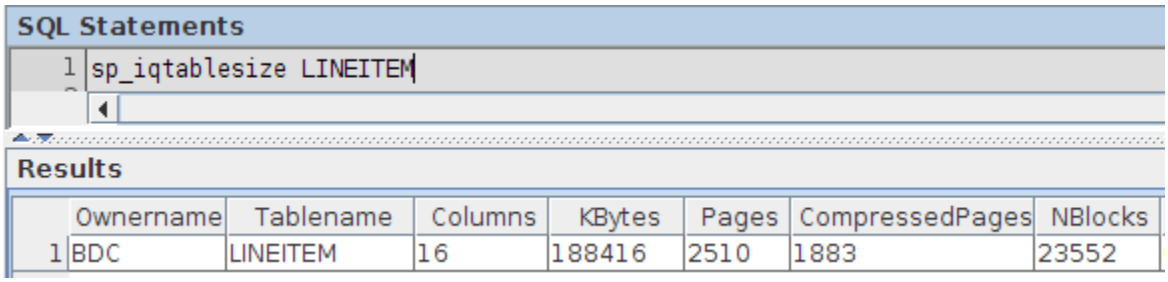
//ESPACIO UTILIZADO TABLA ORDERS RDBMS

```
SQL> SELECT
    lower(table_name)  AS table_name
    ,num_rows
    ,blocks*8      AS size_kb
    ,blocks
    ,compression
FROM
        all_tables
WHERE
        owner = 'BD_COLUMNAR'
        and table_name = 'ORDERS';  2    3    4    5    6    7    8    9
 10   11

TABLE_NAME
--------------------------------------------------------------------------------
------
  NUM_ROWS    SIZE_KB     BLOCKS COMPRESS
---------- ---------- ---------- --------
orders
   1500000     186856      23357 DISABLED


SQL>
```

//LLENADO LOAD TABLE CDBMS TABLA LINEITEM



//ESPACIO UTILIZADO TABLA LINEITEM CDBMS



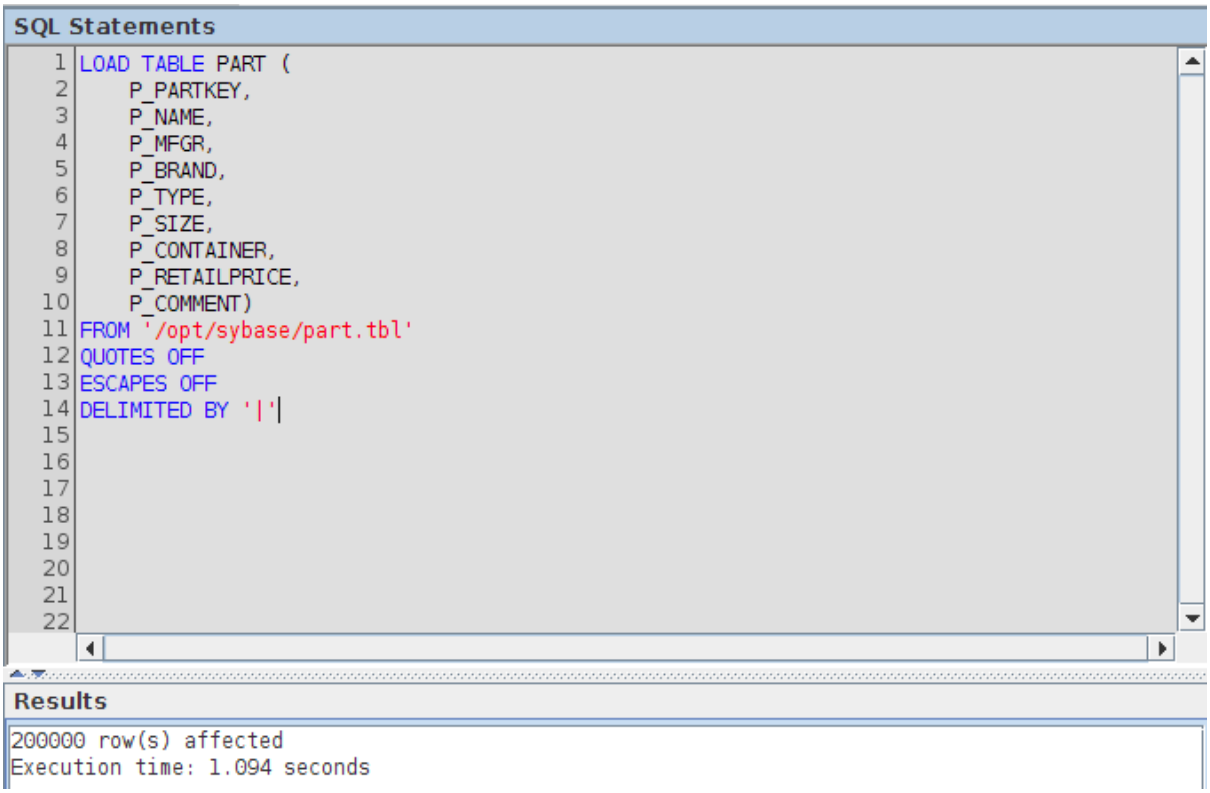
//LLENADO SQLLDR RDBMS TABLA LINEITEM

```
Space allocated for bind array:                  255936 bytes(62 rows)
Read   buffer bytes: 1048576

Total logical records skipped:          0
Total logical records read:       6001215
Total logical records rejected:         0
Total logical records discarded:        0

Run began on Mon Mar 07 22:40:57 2016
Run ended on Mon Mar 07 23:50:55 2016

Elapsed time was:     01:09:57.97
CPU time was:         00:01:52.93
```

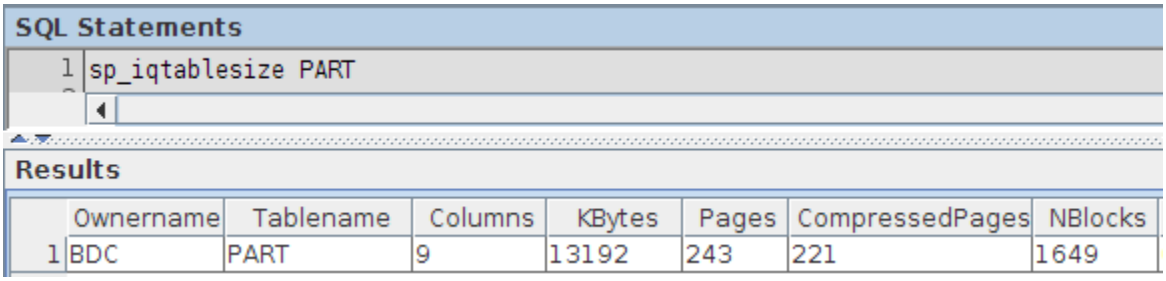
//ESPACIO UTILIZADO TABLA LINEITEM RDBMS

```
SQL> SELECT
    lower(table_name)  AS table_name
    ,num_rows
    ,blocks*8      AS size_kb
    ,blocks
    ,compression
FROM
        all_tables
WHERE
        owner = 'BD_COLUMNAR'
        and table_name = 'LINEITEM';  2    3    4    5    6    7    8    9
   10   11

TABLE_NAME
--------------------------------------------------------------------------------
------
  NUM_ROWS    SIZE_KB     BLOCKS COMPRESS
---------- ---------- ---------- --------
lineitem
   6001215     807016     100877 DISABLED


SQL>
```

//LLENADO LOAD TABLE CDBMS 16 TABLA PART



//ESPACIO UTILIZADO TABLA PART CDBMS



//LLENADO SQLLDR RDBMS TABLA PART

```
Space allocated for bind array:                 148608 bytes(64 rows)
Read   buffer bytes: 1048576

Total logical records skipped:          0
Total logical records read:        200000
Total logical records rejected:         0
Total logical records discarded:        0

Run began on Mon Mar 07 20:39:29 2016
Run ended on Mon Mar 07 20:41:41 2016

Elapsed time was:     00:02:12.21
CPU time was:         00:00:02.94
```

//ESPACIO UTILIZADO TABLA PART RDBMS

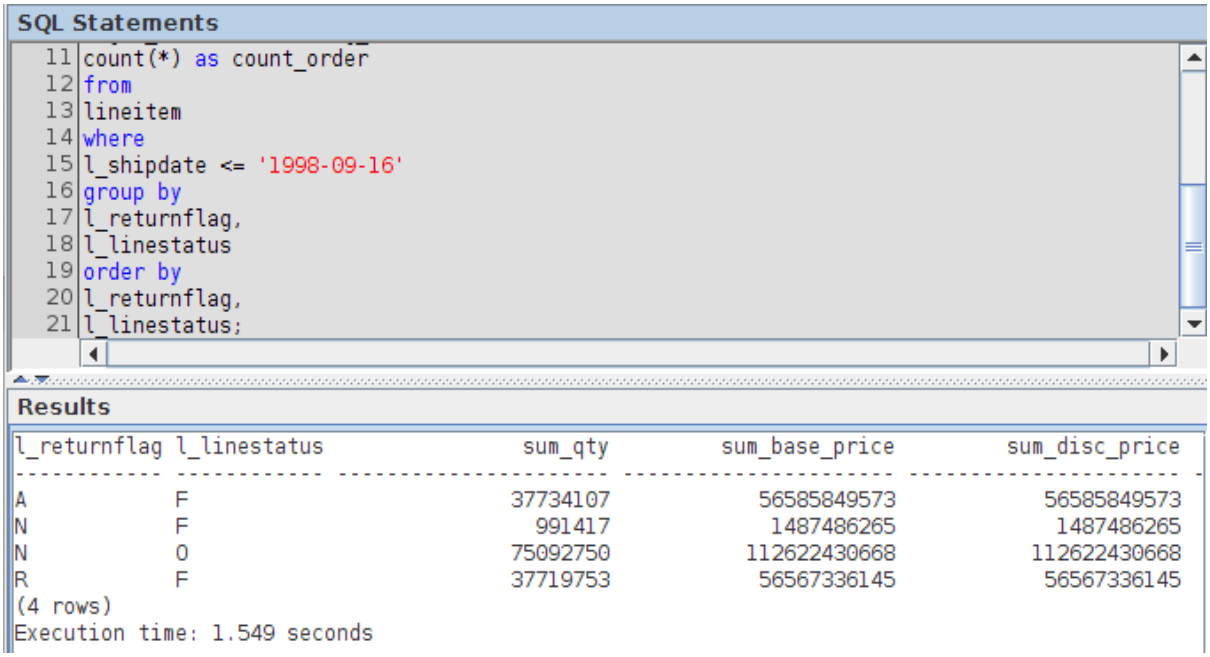
```
SQL> SELECT
    lower(table_name)  AS table_name
    ,num_rows
    ,blocks*8      AS size_kb
    ,blocks
    ,compression
FROM
        all_tables
WHERE
        owner = 'BD_COLUMNAR'
        and table_name = 'PART';  2    3    4    5    6    7    8    9   1
0   11

TABLE_NAME
--------------------------------------------------------------------------------
------
  NUM_ROWS    SIZE_KB     BLOCKS COMPRESS
---------- ---------- ---------- --------
part
    200000      27152       3394 DISABLED


SQL>
```

Consulta 1:

```
select
   l_returnflag,
   l_linestatus,
   sum(l_quantity) as sum_qty,
   sum(l_extendedprice) as sum_base_price,
   sum(l_extendedprice*(1-l_discount)) as sum_disc_price,
```

```
    sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count(*) as count_order
from
    lineitem
where
    l_shipdate <= '1998-09-16'
group by
    l_returnflag,
    l_linestatus
order by
    l_returnflag,
    l_linestatus;
```

Resultado en CDBMS



Resultado en RDBMS

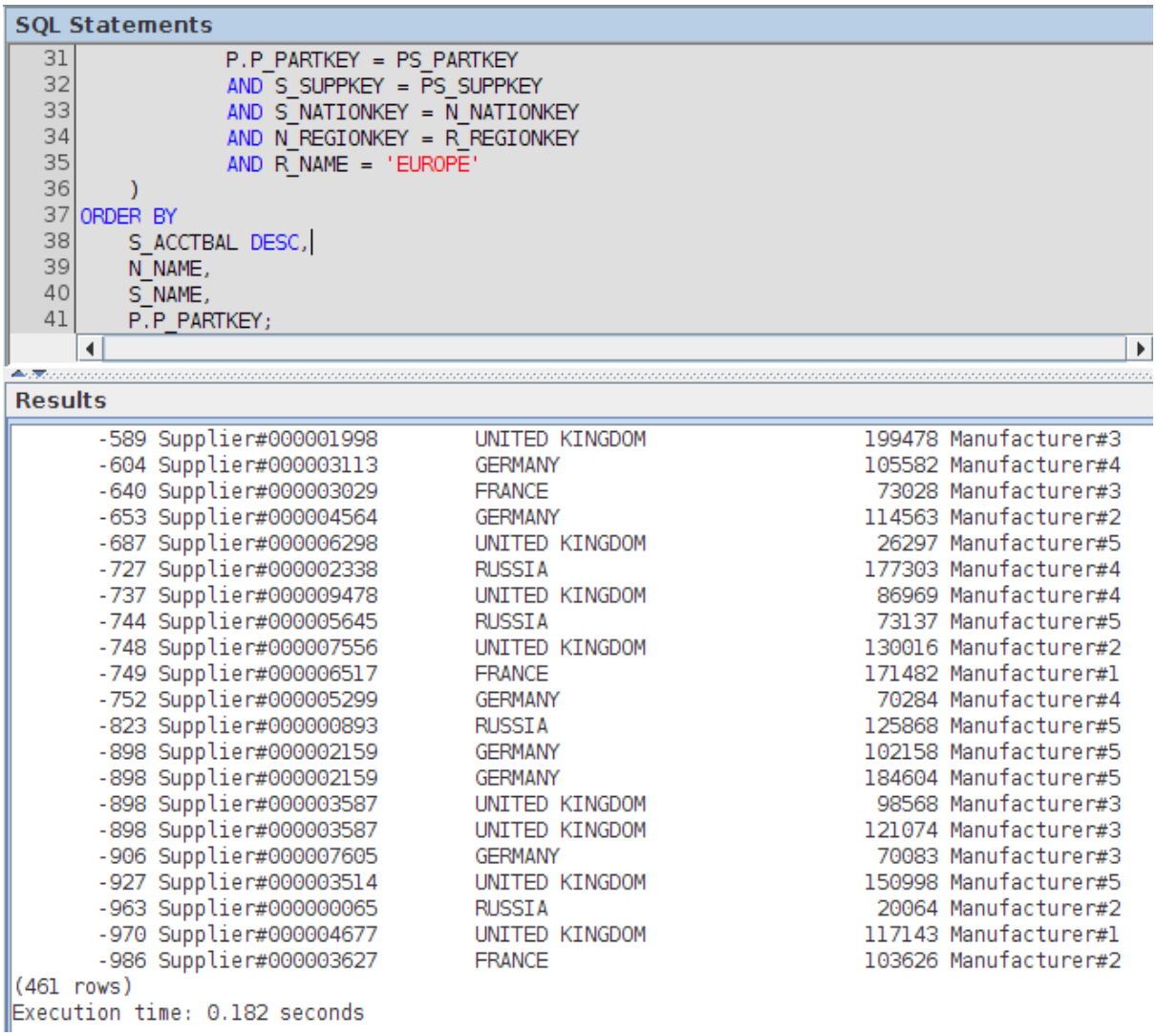
```
L L    SUM_QTY SUM_BASE_PRICE SUM_DISC_PRICE SUM_CHARGE    AVG_QTY  AVG_PRICE
- - ---------- -------------- -------------- ---------- ---------- ----------
  AVG_DISC COUNT_ORDER
---------- -----------
A F   37734107     5.6587E+10     5.3758E+10 5.5909E+10 25.5220059 38273.1297
.049985296     1478493

N F     991417     1487504710     1413082168 1469649223 25.5164719 38284.4678
.050093427       38854

N O   75092750     1.1262E+11     1.0699E+11 1.1128E+11 25.5010801 38246.4275
.049996397     2944689


L L    SUM_QTY SUM_BASE_PRICE SUM_DISC_PRICE SUM_CHARGE    AVG_QTY  AVG_PRICE
- - ---------- -------------- -------------- ---------- ---------- ----------
  AVG_DISC COUNT_ORDER
---------- -----------
R F   37719753     5.6568E+10     5.3741E+10 5.5890E+10 25.5057936 38250.8546
.050009406     1478870


Transcurrido: 00:00:04.56
SQL>
```

Consulta 2:

```
SELECT
   S_ACCTBAL,
   S_NAME,
   N_NAME,
   P.P_PARTKEY,
   P.P_MFGR,
   S_ADDRESS,
   S_PHONE,
   S_COMMENT
FROM
   PART P,
   SUPPLIER,
   PARTSUPP,
   NATION,
   REGION
WHERE
   P.P_PARTKEY = PS_PARTKEY
```

```
    AND S_SUPPKEY = PS_SUPPKEY
    AND P.P_SIZE = 15
    AND P.P_TYPE LIKE '%BRASS'
    AND S_NATIONKEY = N_NATIONKEY
    AND N_REGIONKEY = R_REGIONKEY
    AND R_NAME = 'EUROPE'
    AND PS_SUPPLYCOST = (
            SELECT
                    MIN(PS_SUPPLYCOST)
            FROM
                    PARTSUPP, SUPPLIER,
                    NATION, REGION
            WHERE
                    P.P_PARTKEY = PS_PARTKEY
                    AND S_SUPPKEY = PS_SUPPKEY
                    AND S_NATIONKEY = N_NATIONKEY
                    AND N_REGIONKEY = R_REGIONKEY
                    AND R_NAME = 'EUROPE'
    )
ORDER BY
    S_ACCTBAL DESC,
    N_NAME,
    S_NAME,
    P.P_PARTKEY;
```

Resultado CDBMS



Resultado RDBMS

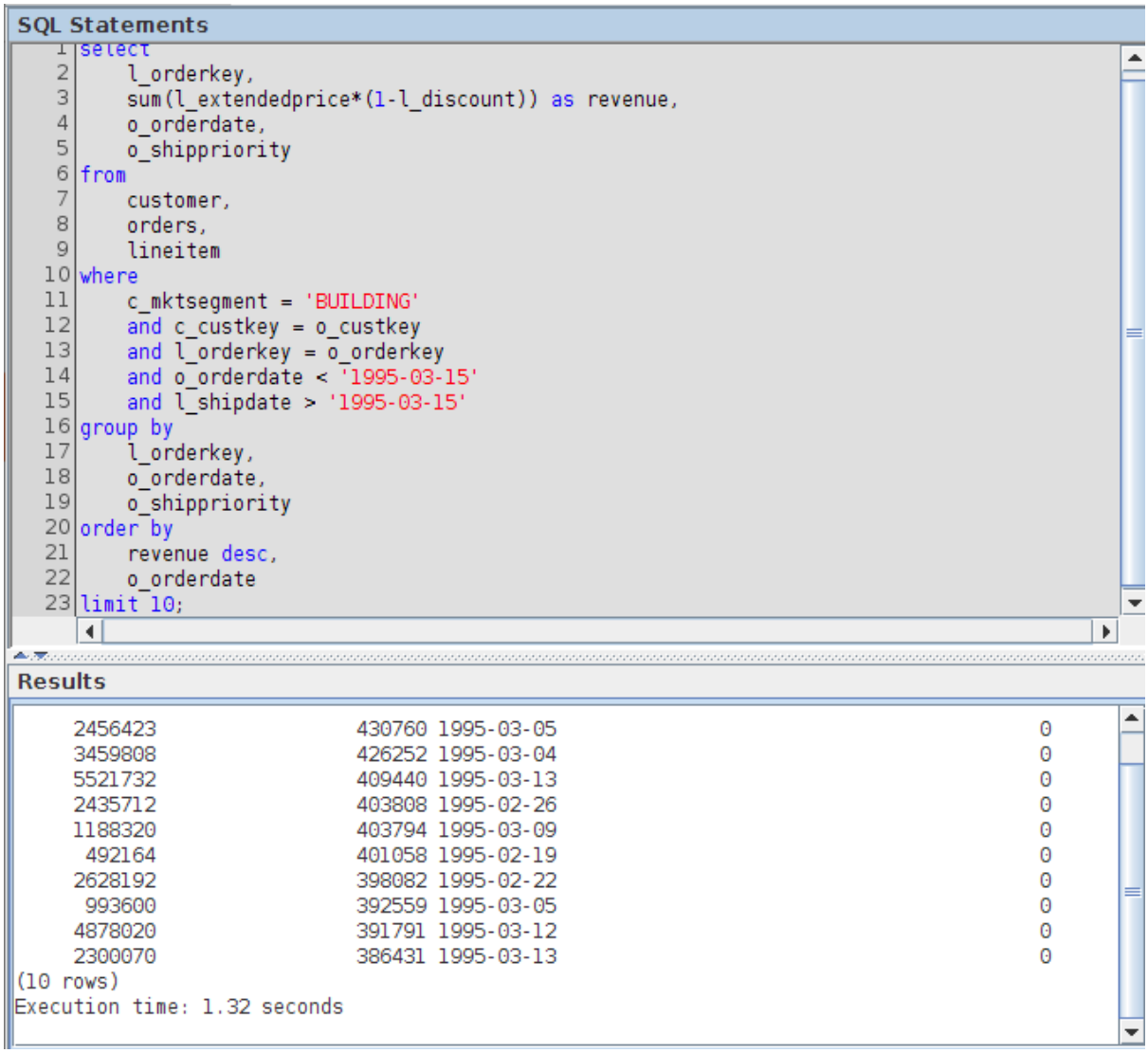
```
 S_ACCTBAL S_NAME                    N_NAME                    P_PARTKEY
---------- ------------------------- ------------------------- ----------
P_MFGR                    S_ADDRESS
------------------------- ----------------------------------------
S_PHONE
---------------
S_COMMENT
-------------------------------------------------------------------------------
es haggle blithe

   -986.14 Supplier#000003627        FRANCE                        103626
Manufacturer#2            77,1uiRw rXybJh
16-568-745-4062

 S_ACCTBAL S_NAME                    N_NAME                    P_PARTKEY
---------- ------------------------- ------------------------- ----------
P_MFGR                    S_ADDRESS
------------------------- ----------------------------------------
S_PHONE
---------------
S_COMMENT
-------------------------------------------------------------------------------
closely blithely regular dolphins. fluffi


460 filas seleccionadas.

Transcurrido: 00:00:00.14
SQL>
```

Consulta 3 CDBMS

```
select TOP 10
    l_orderkey,
    sum(l_extendedprice*(1-l_discount)) as revenue,
    o_orderdate,
    o_shippriority
from
    customer,
    orders,
    lineitem
where
    c_mktsegment = 'BUILDING'
    and c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate < '1995-03-15'
```

```
    and l_shipdate > '1995-03-15'
group by
    l_orderkey,
    o_orderdate,
    o_shippriority
order by
    revenue desc,
    o_orderdate
```

Resultado CDBMS



Consulta 3 RDBMS

```
select
    l_orderkey,
    sum(l_extendedprice*(1-l_discount)) as revenue,
    o_orderdate,
    o_shippriority
from
    customer,
    orders,
    lineitem
where
    c_mktsegment = 'BUILDING'
    and c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate < '1995-03-15'
    and l_shipdate > '1995-03-15'
group by
    l_orderkey,
    o_orderdate,
    o_shippriority
order by
    revenue desc,
    o_orderdate
FETCH FIRST 10 ROWS ONLY;
```

Resultado RDBMS

```
SQL> select
        l_orderkey,
        sum(l_extendedprice*(1-l_discount)) as revenue,
        o_orderdate,
        o_shippriority
from
        customer,
        orders,
        lineitem
where
        c_mktsegment = 'BUILDING'
        and c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate < '1995-03-15'
        and l_shipdate > '1995-03-15'
group by
        l_orderkey,
        o_orderdate,
        o_shippriority
order by
        revenue desc,
        o_orderdate
FETCH FIRST 10 ROWS ONLY;  2    3    4    5    6    7    8    9   10   11
  12   13   14   15   16   17   18   19   20   21   22   23

L_ORDERKEY    REVENUE O_ORDERDAT O_SHIPPRIORITY
---------- ---------- ---------- --------------
   2456423 406181.011 1995-03-05              0
   3459808 405838.699 1995-03-04              0
    492164 390324.061 1995-02-19              0
   1188320 384537.936 1995-03-09              0
   2435712 378673.056 1995-02-26              0
   4878020 378376.795 1995-03-12              0
   5521732 375153.922 1995-03-13              0
   2628192 373133.309 1995-02-22              0
    993600  371407.46 1995-03-05              0
   2300070 367371.145 1995-03-13              0

10 filas seleccionadas.

Transcurrido: 00:00:01.26
SQL>
```

Consulta 4:

```
SELECT
   O_ORDERPRIORITY,
   COUNT(*) AS ORDER_COUNT
FROM
   ORDERS O
```

```
WHERE
   O_ORDERDATE  BETWEEN '1993-07-01' AND '1993-09-30'
   AND EXISTS (
            SELECT
                     *
            FROM
                     LINEITEM
            WHERE
                     L_ORDERKEY = O.O_ORDERKEY
                     AND L_COMMITDATE < L_RECEIPTDATE
   )
GROUP BY
   O_ORDERPRIORITY
ORDER BY
   O_ORDERPRIORITY;
```

Resultado CDBMS

SQL Statements

```
1  SELECT
2  O_ORDERPRIORITY,
3  COUNT(*) AS ORDER_COUNT
4  FROM
5  ORDERS O
6  WHERE
7  O_ORDERDATE  BETWEEN '1993-07-01' AND '1993-09-30'
8  AND EXISTS (
9  SELECT
10 *
11 FROM
12 LINEITEM
13 WHERE
14 L_ORDERKEY = O.O_ORDERKEY
15 AND L_COMMITDATE < L_RECEIPTDATE
16 )
17 GROUP BY
18 O_ORDERPRIORITY
19 ORDER BY
20 O_ORDERPRIORITY;
21
22
```

Results

```
O_ORDERPRIORITY          ORDER_COUNT
-------------- --------------------
1-URGENT                       10594
2-HIGH                         10476
3-MEDIUM                       10410
4-NOT SPECIFIED                10556
5-LOW                          10487
(5 rows)
Execution time: 0.8 seconds
```

Resultado RDBMS

```
SQL> SELECT
O_ORDERPRIORITY,
COUNT(*) AS ORDER_COUNT
FROM
ORDERS O
WHERE
O_ORDERDATE  BETWEEN '1993-07-01' AND '1993-09-30'
AND EXISTS (
SELECT
*
FROM
LINEITEM
WHERE
L_ORDERKEY = O.O_ORDERKEY
AND L_COMMITDATE < L_RECEIPTDATE
)
GROUP BY
O_ORDERPRIORITY
ORDER BY
O_ORDERPRIORITY;  2    3    4    5    6    7    8    9   10   11   12   13
   14   15   16   17   18   19   20

O_ORDERPRIORITY ORDER_COUNT
--------------- -----------
1-URGENT              10594
2-HIGH                10476
3-MEDIUM              10410
4-NOT SPECIFIED       10556
5-LOW                 10487

Transcurrido: 00:00:01.10
SQL>
```

Consulta 5:

```
select
    n_name,
    sum(l_extendedprice * (1 -l_discount)) as revenue
from
    customer,
    orders,
    lineitem,
    supplier,
    nation,
    region
where
```

```
    c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and l_suppkey = s_suppkey
    and c_nationkey = s_nationkey
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = 'ASIA'
    and o_orderdate >= '1994-01-01'
    and o_orderdate < '1995-01-01'
group by
    n_name
order by
    revenue desc;
```

Resultado CDBMS

SQL Statements

```
    n_name,
    sum(l_extendedprice * (1 -l_discount)) as revenue
from
    customer,
    orders,
    lineitem,
    supplier,
    nation,
    region
where
    c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and l_suppkey = s_suppkey
    and c_nationkey = s_nationkey
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = 'ASIA'
    and o_orderdate >= '1994-01-01'
    and o_orderdate < '1995-01-01'
group by
    n_name
order by
    revenue desc;
```

Results

```
n_name                                    revenue
------------------------- ----------------------
INDONESIA                                58355388
VIETNAM                                  58272781
CHINA                                    56572386
INDIA                                    54836573
JAPAN                                    47801690
(5 rows)
Execution time: 0.564 seconds
```

Resultado RDBMS

```
SQL> select
        n_name,
        sum(l_extendedprice * (1 -l_discount)) as revenue
from
        customer,
        orders,
        lineitem,
        supplier,
        nation,
        region
where
        c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and l_suppkey = s_suppkey
        and c_nationkey = s_nationkey
        and s_nationkey = n_nationkey
        and n_regionkey = r_regionkey
        and r_name = 'ASIA'
        and o_orderdate >= '1994-01-01'
        and o_orderdate < '1995-01-01'
group by
        n_name
order by
        revenue desc;  2    3    4    5    6    7    8    9   10   11   12
   13   14   15   16   17   18   19   20   21   22   23   24

N_NAME                      REVENUE
------------------------- ----------
INDONESIA                 55502041.2
VIETNAM                     55295087
CHINA                     53724494.3
INDIA                       52035512
JAPAN                     45410175.7

Transcurrido: 00:00:01.78
SQL>
```

Consulta 6:

SELECT

SUM(L_EXTENDEDPRICE*L_DISCOUNT) AS REVENUE

FROM

LINEITEM

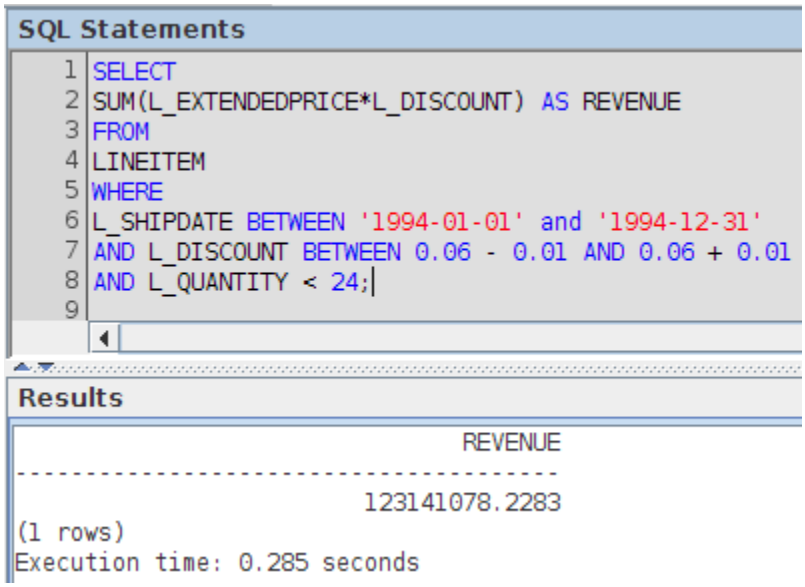
WHERE

L_SHIPDATE BETWEEN '1994-01-01' and '1994-12-31'

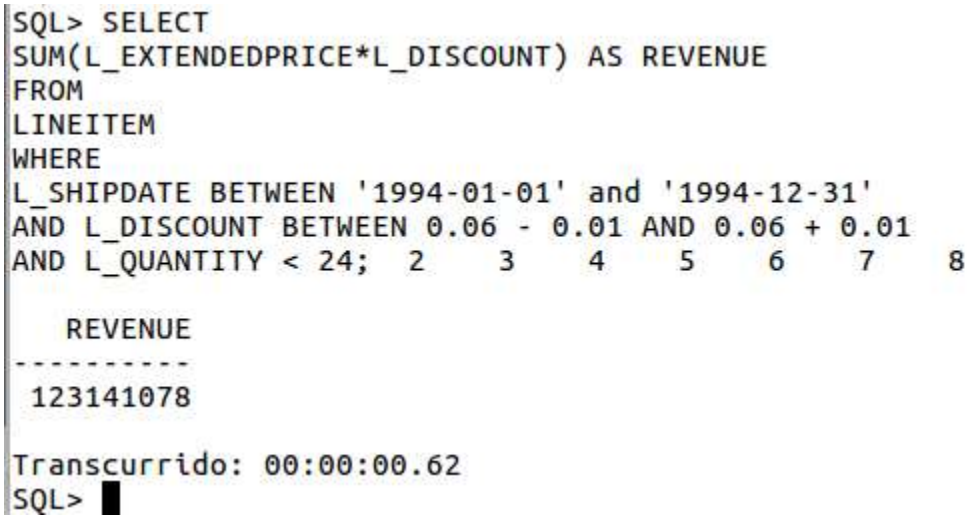
AND L_DISCOUNT BETWEEN 0.06 - 0.01 AND 0.06 + 0.01

AND L_QUANTITY < 24;

Resultado CDBMS



Resultado RDBMS



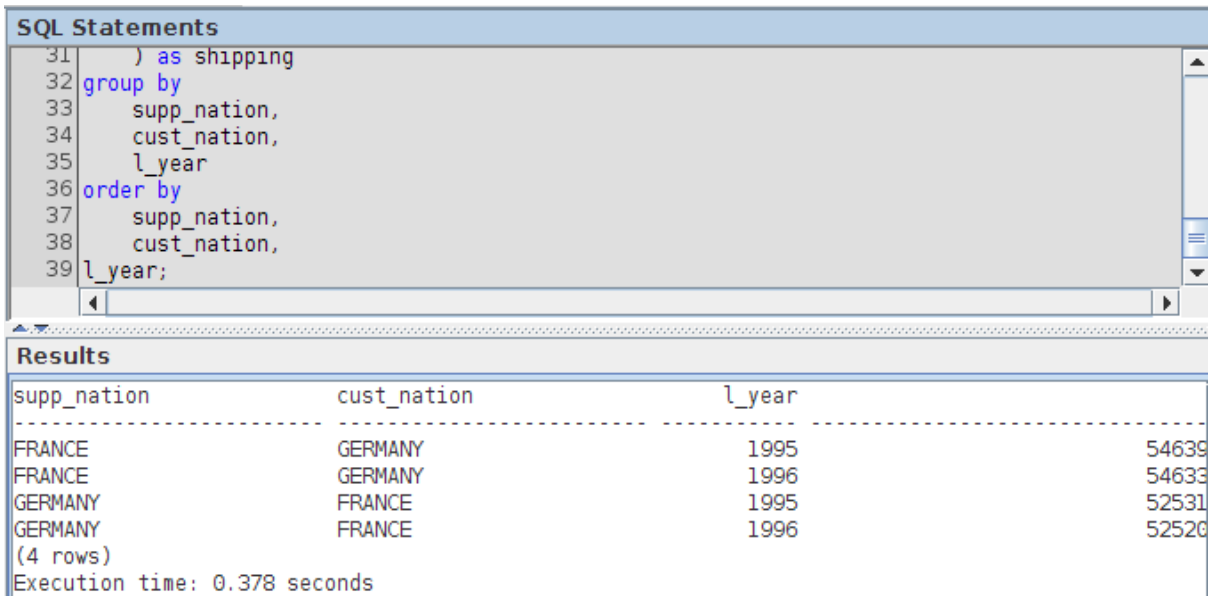
Consulta 7 CDBMS

```
select
    supp_nation,
    cust_nation,
    l_year,
```

```
    sum(volume) as revenue
from
    (
        select
            n1.n_name as supp_nation,
            n2.n_name as cust_nation,
            year(l_shipdate) as l_year,
            l_extendedprice * (1 - l_discount) as volume
        from
            supplier,
            lineitem,
            orders,
            customer,
            nation n1,
            nation n2
        where
            s_suppkey = l_suppkey
            and o_orderkey = l_orderkey
            and c_custkey = o_custkey
            and s_nationkey = n1.n_nationkey
            and c_nationkey = n2.n_nationkey
            and (
                (n1.n_name = 'FRANCE' and n2.n_name = 'GERMANY')
                or (n1.n_name = 'GERMANY' and n2.n_name = 'FRANCE')
            )
            and l_shipdate between '1995-01-01' and '1996-12-31'
    ) as shipping
group by
    supp_nation,
    cust_nation,
    l_year
order by
    supp_nation,
```

```
    cust_nation,
l_year;
```

Resultado CDBMS



Consulta 7 RDBMS

```
select
    supp_nation,
    cust_nation,
    l_year,
    sum(volume) as revenue
from
    (
        select
            n1.n_name as supp_nation,
            n2.n_name as cust_nation,
            EXTRACT(YEAR FROM TO_DATE(l_shipdate, 'YY-MM-DD')) as l_year,
            l_extendedprice * (1 - l_discount) as volume
        from
            supplier,
            lineitem,
```

```
                orders,
                customer,
                nation n1,
                nation n2
        where
                s_suppkey = l_suppkey
                and o_orderkey = l_orderkey
                and c_custkey = o_custkey
                and s_nationkey = n1.n_nationkey
                and c_nationkey = n2.n_nationkey
                and (
                        (n1.n_name = 'FRANCE' and n2.n_name = 'GERMANY')
                        or (n1.n_name = 'GERMANY' and n2.n_name = 'FRANCE')
                )
                and l_shipdate between '1995-01-01' and '1996-12-31'
    ) shipping
group by
    supp_nation,
    cust_nation,
    l_year
order by
    supp_nation,
    cust_nation,
l_year;
```

Resultado RDBMS

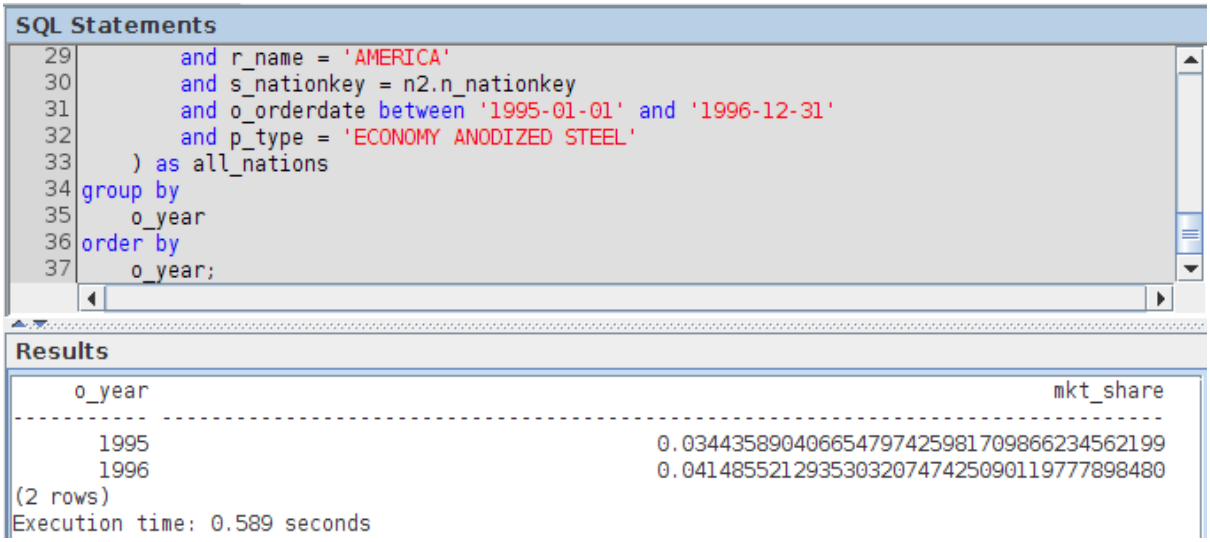
```
                from
                        supplier,
                        lineitem,
                        orders,
                        customer,
                        nation n1,
                        nation n2
                where
                        s_suppkey = l_suppkey
                        and o_orderkey = l_orderkey
                        and c_custkey = o_custkey
                        and s_nationkey = n1.n_nationkey
                        and c_nationkey = n2.n_nationkey
                        and (
                                (n1.n_name = 'FRANCE' and n2.n_name = 'GER
MANY')
                                or (n1.n_name = 'GERMANY' and n2.n_name =
'FRANCE')
                        )
                        and l_shipdate between '1995-01-01' and '1996-12-3
1'
        ) shipping
group by
        supp_nation,
        cust_nation,
        l_year
order by
        supp_nation,
        cust_nation,
l_year;  2    3    4    5    6    7    8    9   10   11   12   13   14   1
5   16   17   18   19   20   21   22   23   24   25   26   27   28   29
30   31   32   33   34   35   36   37   38   39

SUPP_NATION               CUST_NATION                   L_YEAR    REVENUE
------------------------- ------------------------- ---------- ----------
FRANCE                    GERMANY                         1995 54639732.7
FRANCE                    GERMANY                         1996 54633083.3
GERMANY                   FRANCE                          1995 52531746.7
GERMANY                   FRANCE                          1996   52520549

Transcurrido: 00:00:01.57
SQL>
```

Consulta 8 CDBMS

```
select
   o_year,
   sum(case
           when nation = 'BRAZIL'
           then volume
           else 0
```

```
    end)/ sum(volume) as mkt_share
from (
    select
            year(o_orderdate) as o_year,
            l_extendedprice * (1-l_discount) as volume,
            n2.n_name as nation
    from
            part,
            supplier,
            lineitem,
            orders,
            customer,
            nation n1,
            nation n2,
            region
    where
            p_partkey = l_partkey
            and s_suppkey = l_suppkey
            and l_orderkey = o_orderkey
            and o_custkey = c_custkey
            and c_nationkey = n1.n_nationkey
            and n1.n_regionkey = r_regionkey
            and r_name = 'AMERICA'
            and s_nationkey = n2.n_nationkey
            and o_orderdate between '1995-01-01' and '1996-12-31'
            and p_type = 'ECONOMY ANODIZED STEEL'
    ) as all_nations
group by
    o_year
order by
    o_year;
```

Resultado CDBMS

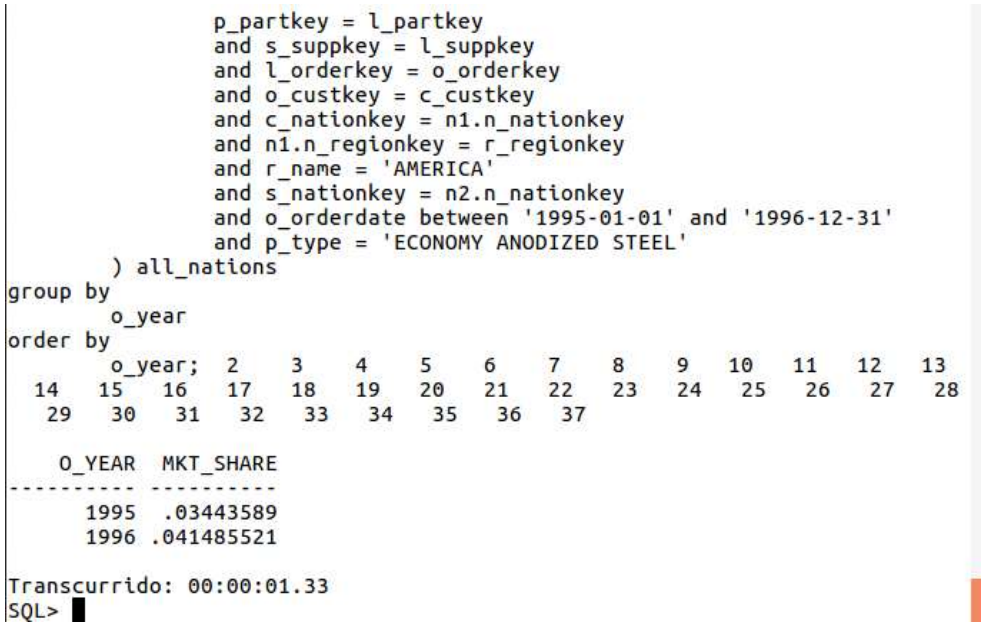


Consulta 8 el RDBMS

```
select
    o_year,
    sum(case
            when nation = 'BRAZIL'
            then volume
            else 0
    end)/ sum(volume) as mkt_share
from (
    select
            EXTRACT(YEAR FROM TO_DATE(o_orderdate, 'YY-MM-DD')) as o_year,
            l_extendedprice * (1-l_discount) as volume,
            n2.n_name as nation
    from
            part,
            supplier,
            lineitem,
            orders,
            customer,
            nation n1,
            nation n2,
```

```
        region
    where
        p_partkey = l_partkey
        and s_suppkey = l_suppkey
        and l_orderkey = o_orderkey
        and o_custkey = c_custkey
        and c_nationkey = n1.n_nationkey
        and n1.n_regionkey = r_regionkey
        and r_name = 'AMERICA'
        and s_nationkey = n2.n_nationkey
        and o_orderdate between '1995-01-01' and '1996-12-31'
        and p_type = 'ECONOMY ANODIZED STEEL'
    ) all_nations
group by
    o_year
order by
    o_year;
```

Resultado RDBMS

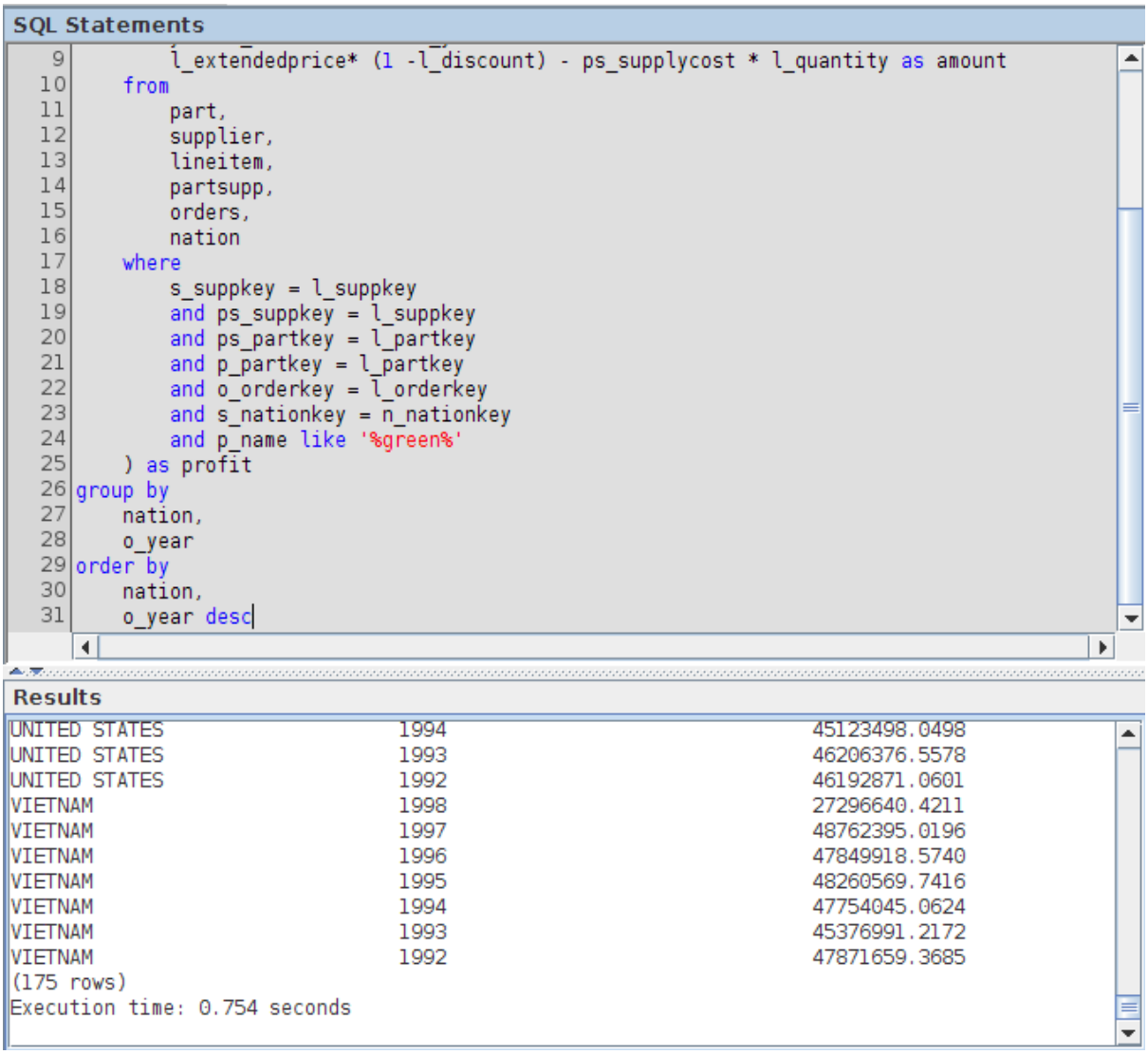


```
                p_partkey = l_partkey
                and s_suppkey = l_suppkey
                and l_orderkey = o_orderkey
                and o_custkey = c_custkey
                and c_nationkey = n1.n_nationkey
                and n1.n_regionkey = r_regionkey
                and r_name = 'AMERICA'
                and s_nationkey = n2.n_nationkey
                and o_orderdate between '1995-01-01' and '1996-12-31'
                and p_type = 'ECONOMY ANODIZED STEEL'
        ) all_nations
group by
        o_year
order by
        o_year;  2    3    4    5    6    7    8    9   10   11   12   13
  14   15   16   17   18   19   20   21   22   23   24   25   26   27   28
   29   30   31   32   33   34   35   36   37

    O_YEAR  MKT_SHARE
---------- ----------
      1995  .03443589
      1996 .041485521

Transcurrido: 00:00:01.33
SQL>
```

Consulta 9 CDBMS

```
select
    nation,
    o_year,
    sum(amount) as sum_profit
from (
    select
            n_name as nation,
            year(o_orderdate) as o_year,
            l_extendedprice* (1 -l_discount) - ps_supplycost * l_quantity as amount
    from
            part,
            supplier,
            lineitem,
            partsupp,
            orders,
            nation
    where
            s_suppkey = l_suppkey
            and ps_suppkey = l_suppkey
            and ps_partkey = l_partkey
            and p_partkey = l_partkey
            and o_orderkey = l_orderkey
            and s_nationkey = n_nationkey
            and p_name like '%green%'
    ) as profit
group by
    nation,
    o_year
order by
    nation,
    o_year desc
```

Resultado CDBMS



Consulta 9 RDBMS

```
select
    nation,
    o_year,
    sum(amount) as sum_profit
from (
    select
            n_name as nation,
```

```
        EXTRACT(YEAR FROM TO_DATE(o_orderdate, 'YY-MM-DD')) as o_year,
        l_extendedprice* (1 -l_discount) - ps_supplycost * l_quantity as amount
    from
        part,
        supplier,
        lineitem,
        partsupp,
        orders,
        nation
    where
        s_suppkey = l_suppkey
        and ps_suppkey = l_suppkey
        and ps_partkey = l_partkey
        and p_partkey = l_partkey
        and o_orderkey = l_orderkey
        and s_nationkey = n_nationkey
        and p_name like '%green%'
    ) profit
group by
    nation,
    o_year
order by
    nation,
    o_year desc;
```

Resultado RDBMS

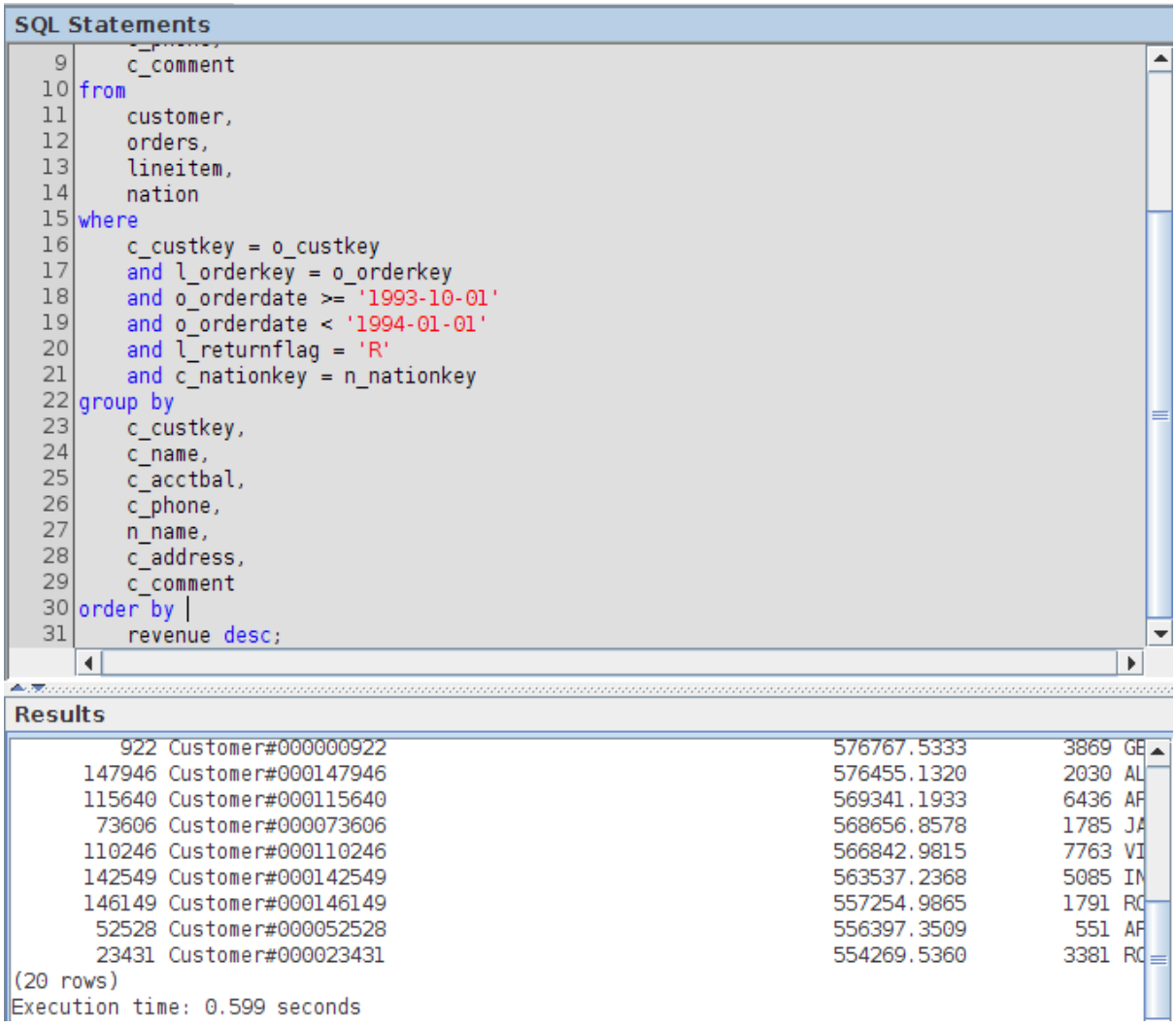
```
NATION                    O_YEAR SUM_PROFIT
------------------------- ---------- ----------
UNITED STATES                 1994 45099092.1
UNITED STATES                 1993 46181600.5
UNITED STATES                 1992 46168214.1
VIETNAM                       1998   27281931
VIETNAM                       1997 48735914.2
VIETNAM                       1996 47824595.9
VIETNAM                       1995 48235135.8
VIETNAM                       1994 47729256.3
VIETNAM                       1993 45352676.9
VIETNAM                       1992 47846355.6

175 filas seleccionadas.

Transcurrido: 00:00:03.52
SQL>
```

Consulta 10 CDBMS

```
select TOP 20
    c_custkey,
    c_name,
    sum(l_extendedprice * (1 -l_discount)) as revenue,
    c_acctbal,
    n_name,
    c_address,
    c_phone,
    c_comment
from
    customer,
    orders,
    lineitem,
    nation
where
    c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate >= '1993-10-01'
    and o_orderdate < '1994-01-01'
    and l_returnflag = 'R'
    and c_nationkey = n_nationkey
```

```
group by
    c_custkey,
    c_name,
    c_acctbal,
    c_phone,
    n_name,
    c_address,
    c_comment
order by
    revenue desc;
```

Resultado CDBMS

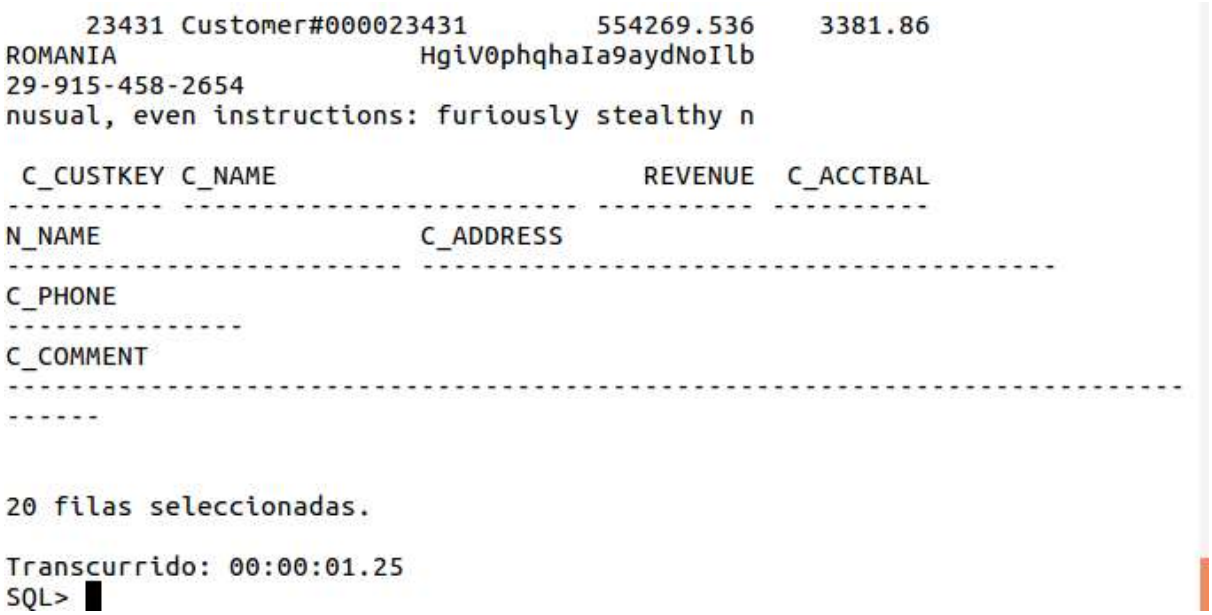


Consulta 10 RDBMS

```
select
    c_custkey,
    c_name,
    sum(l_extendedprice * (1 -l_discount)) as revenue,
    c_acctbal,
    n_name,
    c_address,
    c_phone,
    c_comment
from
    customer,
    orders,
    lineitem,
    nation
where
    c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate >= '1993-10-01'
    and o_orderdate < '1994-01-01'
    and l_returnflag = 'R'
    and c_nationkey = n_nationkey
group by
    c_custkey,
    c_name,
    c_acctbal,
    c_phone,
    n_name,
    c_address,
    c_comment
order by
    revenue desc
```

```
FETCH FIRST 20 ROWS ONLY;
```

Resultado RDBMS



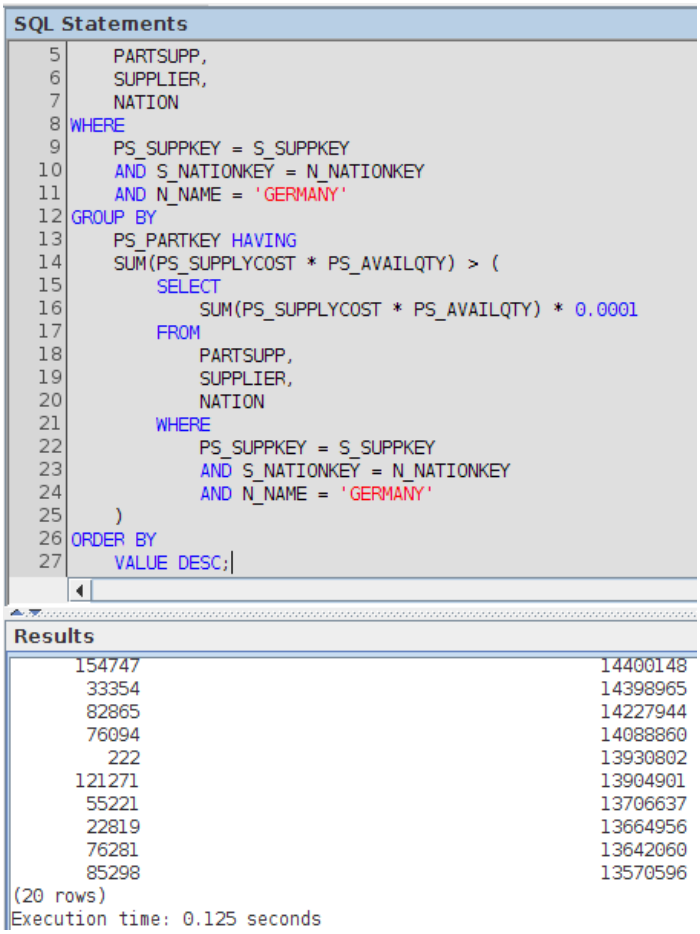
```
     23431 Customer#000023431        554269.536    3381.86
ROMANIA                   HgiV0phqhaIa9aydNoIlb
29-915-458-2654
nusual, even instructions: furiously stealthy n

 C_CUSTKEY C_NAME                       REVENUE  C_ACCTBAL
---------- ------------------------- ---------- ----------
N_NAME                    C_ADDRESS
------------------------- ----------------------------------------
C_PHONE
---------------
C_COMMENT
--------------------------------------------------------------------------------
------


20 filas seleccionadas.

Transcurrido: 00:00:01.25
SQL>
```

Consulta 11 CDBMS

```
SELECT TOP 20
   PS_PARTKEY,
   SUM(PS_SUPPLYCOST * PS_AVAILQTY) AS VALUE
FROM
   PARTSUPP,
   SUPPLIER,
   NATION
WHERE
   PS_SUPPKEY = S_SUPPKEY
   AND S_NATIONKEY = N_NATIONKEY
   AND N_NAME = 'GERMANY'
GROUP BY
   PS_PARTKEY HAVING
   SUM(PS_SUPPLYCOST * PS_AVAILQTY) > (
            SELECT
                     SUM(PS_SUPPLYCOST * PS_AVAILQTY) * 0.0001
```

```
        FROM
                PARTSUPP,
                SUPPLIER,
                NATION
        WHERE
                PS_SUPPKEY = S_SUPPKEY
                AND S_NATIONKEY = N_NATIONKEY
                AND N_NAME = 'GERMANY'
  )
ORDER BY
  VALUE DESC;
```

Resultado CDBMS



Consulta 11 RDBMS

```
SELECT
    PS_PARTKEY,
    SUM(PS_SUPPLYCOST * PS_AVAILQTY) AS VALUE
FROM
    PARTSUPP,
    SUPPLIER,
    NATION
WHERE
    PS_SUPPKEY = S_SUPPKEY
    AND S_NATIONKEY = N_NATIONKEY
    AND N_NAME = 'GERMANY'
GROUP BY
    PS_PARTKEY HAVING
    SUM(PS_SUPPLYCOST * PS_AVAILQTY) > (
            SELECT
                    SUM(PS_SUPPLYCOST * PS_AVAILQTY) * 0.0001
            FROM
                    PARTSUPP,
                    SUPPLIER,
                    NATION
            WHERE
                    PS_SUPPKEY = S_SUPPKEY
                    AND S_NATIONKEY = N_NATIONKEY
                    AND N_NAME = 'GERMANY'
    )
ORDER BY
    VALUE DESC
FETCH FIRST 20 ROWS ONLY;
```

Resultado RDBMS

```
PS_PARTKEY      VALUE
---------- ----------
    129760 17538456.9
    166726 16503353.9
    191287   16474802
    161758 16101755.5
     34452 15983844.7
    139035 15907078.3
      9403 15451755.6
    154358 15212937.9
     38823 15064802.9
     85606 15053957.2
     33354 14408297.4

PS_PARTKEY      VALUE
---------- ----------
    154747 14407580.7
     82865 14235489.8
     76094   14094247
       222 13937777.7
    121271   13908336
     55221 13716120.5
     22819 13666434.3
     76281 13646853.7
     85298 13581154.9

20 filas seleccionadas.

Transcurrido: 00:00:00.67
SQL>
```

Consulta 12:

```
SELECT
  L_SHIPMODE,
  SUM(CASE
        WHEN O_ORDERPRIORITY ='1-URGENT'
        OR O_ORDERPRIORITY ='2-HIGH'
        THEN 1
        ELSE 0
  END) AS HIGH_LINE_COUNT,
  SUM(CASE
        WHEN O_ORDERPRIORITY <> '1-URGENT'
        AND O_ORDERPRIORITY <> '2-HIGH'
        THEN 1
```

```
        ELSE 0
        END) AS LOW_LINE_COUNT
FROM
   ORDERS,
   LINEITEM
WHERE
   O_ORDERKEY = L_ORDERKEY
   AND L_SHIPMODE IN ('MAIL', 'SHIP')
   AND L_COMMITDATE < L_RECEIPTDATE
   AND L_SHIPDATE < L_COMMITDATE
   AND L_RECEIPTDATE BETWEEN '1994-01-01' and '1994-12-31'
GROUP BY
   L_SHIPMODE
ORDER BY
   L_SHIPMODE;
```

Resultado CDBMS

SQL Statements

```
        OR O_ORDERPRIORITY ='2-HIGH'
        THEN 1
        ELSE 0
    END) AS HIGH_LINE_COUNT,
    SUM(CASE
        WHEN O_ORDERPRIORITY <> '1-URGENT'
        AND O_ORDERPRIORITY <> '2-HIGH'
        THEN 1
        ELSE 0
        END) AS LOW_LINE_COUNT
FROM
    ORDERS,
    LINEITEM
WHERE
    O_ORDERKEY = L_ORDERKEY
    AND L_SHIPMODE IN ('MAIL', 'SHIP')
    AND L_COMMITDATE < L_RECEIPTDATE
    AND L_SHIPDATE < L_COMMITDATE
    AND L_RECEIPTDATE BETWEEN '1994-01-01' and '1994-12-31'
GROUP BY
    L_SHIPMODE
ORDER BY
    L_SHIPMODE;
```

Results

```
L_SHIPMODE      HIGH_LINE_COUNT       LOW_LINE_COUNT
---------- -------------------- --------------------
MAIL                      6202                 9324
SHIP                      6200                 9262
(2 rows)
Execution time: 0.254 seconds
```

Resultado RDBMS

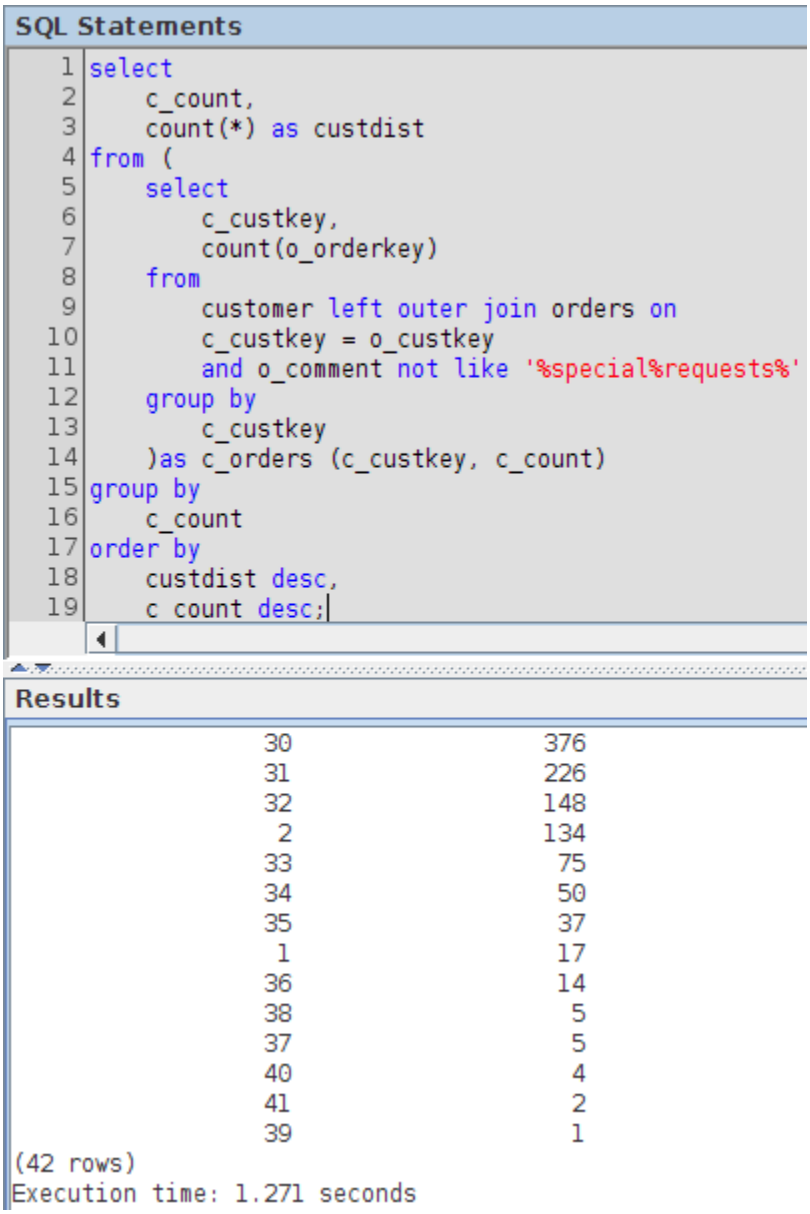
```
SQL> SELECT
        L_SHIPMODE,
        SUM(CASE
                WHEN O_ORDERPRIORITY ='1-URGENT'
                OR O_ORDERPRIORITY ='2-HIGH'
                THEN 1
                ELSE 0
        END) AS HIGH_LINE_COUNT,
        SUM(CASE
                WHEN O_ORDERPRIORITY <> '1-URGENT'
                AND O_ORDERPRIORITY <> '2-HIGH'
                THEN 1
                ELSE 0
                END) AS LOW_LINE_COUNT
FROM
        ORDERS,
        LINEITEM
WHERE
        O_ORDERKEY = L_ORDERKEY
        AND L_SHIPMODE IN ('MAIL', 'SHIP')
        AND L_COMMITDATE < L_RECEIPTDATE
        AND L_SHIPDATE < L_COMMITDATE
        AND L_RECEIPTDATE BETWEEN '1994-01-01' and '1994-12-31'
GROUP BY
        L_SHIPMODE
ORDER BY
        L_SHIPMODE;  2    3    4    5    6    7    8    9   10   11   12
 13   14   15   16   17   18   19   20   21   22   23   24   25   26   27


L_SHIPMODE HIGH_LINE_COUNT LOW_LINE_COUNT
---------- --------------- --------------
MAIL                  6202           9324
SHIP                  6200           9262

Transcurrido: 00:00:01.34
SQL>
```

Consulta 13 CDBMS

```
select
   c_count,
   count(*) as custdist
from (
   select
           c_custkey,
           count(o_orderkey)
   from
```

```
            customer left outer join orders on
            c_custkey = o_custkey
            and o_comment not like '%special%requests%'
    group by
            c_custkey
    )as c_orders (c_custkey, c_count)
group by
    c_count
order by
    custdist desc,
    c_count desc;
```

Resultado CDBMS



Consulta 13 RDBMS

```
select
    c_count,
    count(*) as custdist
from (
    select
            c_custkey,
            count(o_orderkey) as c_count
```

```
    from
            customer left outer join orders on
            c_custkey = o_custkey
            and o_comment not like '%special%requests%'
    group by
            c_custkey
    ) c_orders
group by
    c_count
order by
    custdist desc,
    c_count desc;
```

Resultado Consulta RDBMS

```
   C_COUNT   CUSTDIST
---------- ----------
        26       1612
        27       1179
         4       1007
        28        893
        29        593
         3        415
        30        376
        31        226
        32        148
         2        134
        33         75

   C_COUNT   CUSTDIST
---------- ----------
        34         50
        35         37
         1         17
        36         14
        38          5
        37          5
        40          4
        41          2
        39          1

42 filas seleccionadas.

Transcurrido: 00:00:01.49
SQL>
```

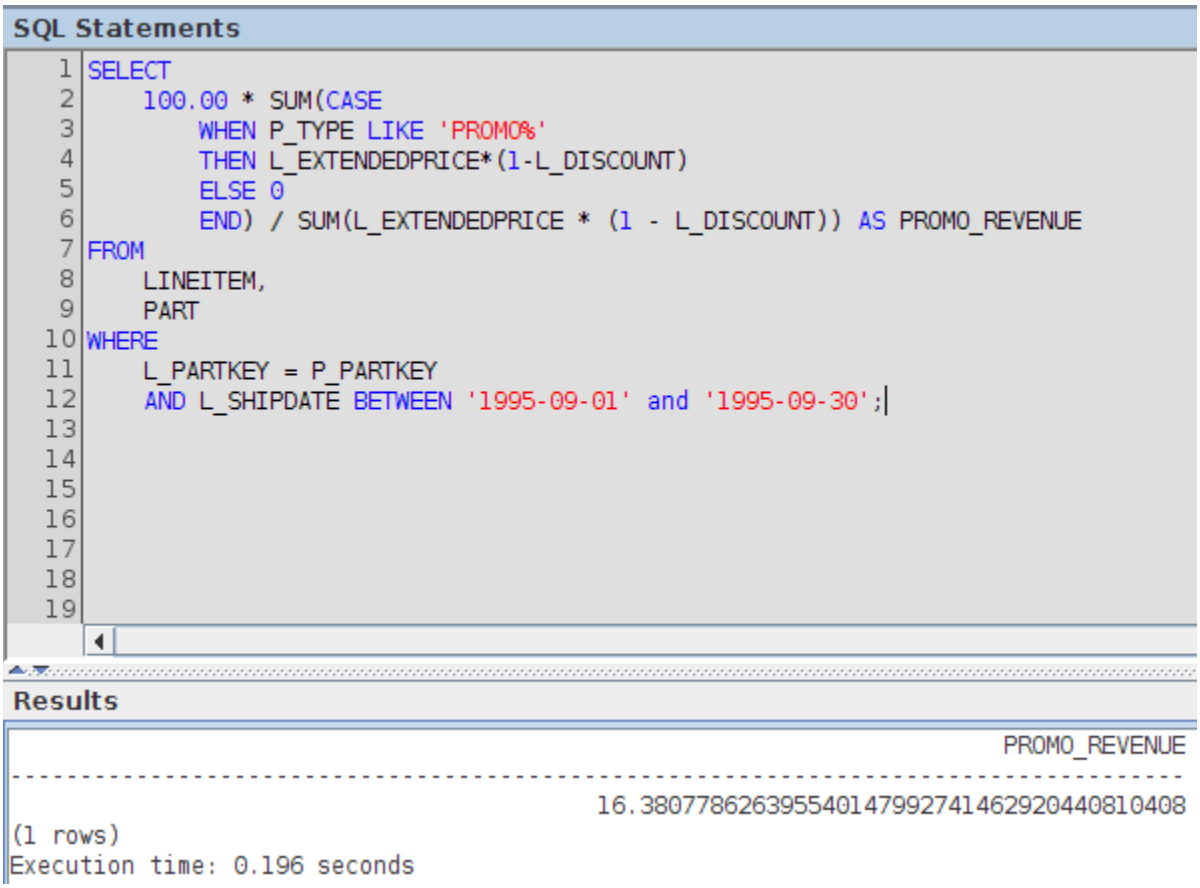
Consulta 14:

```
SELECT
    100.00 * SUM(CASE
            WHEN P_TYPE LIKE 'PROMO%'
            THEN L_EXTENDEDPRICE*(1-L_DISCOUNT)
            ELSE 0
            END) / SUM(L_EXTENDEDPRICE * (1 - L_DISCOUNT)) AS PROMO_REVENUE
FROM
    LINEITEM,
    PART
WHERE
```

L_PARTKEY = P_PARTKEY

AND L_SHIPDATE BETWEEN '1995-09-01' and '1995-09-30';

Resultado CDBMS



Resultado RDBMS

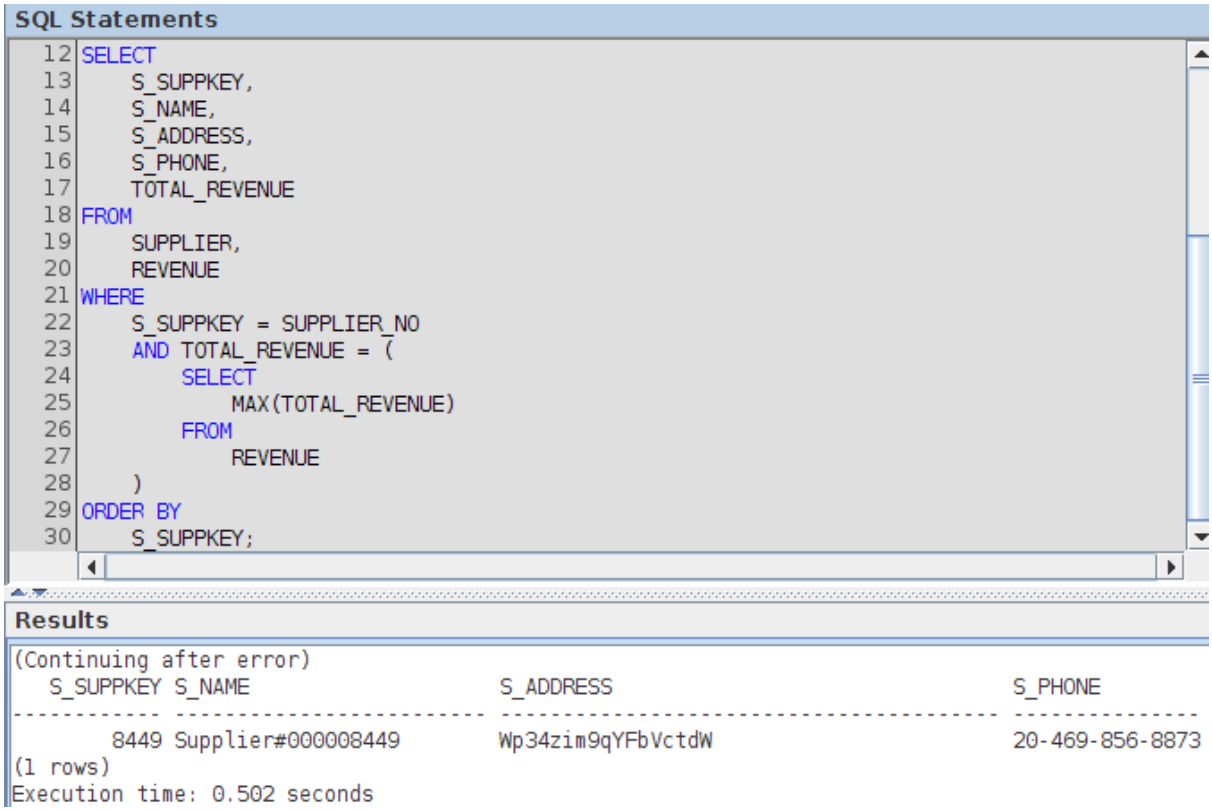
```
SQL> SELECT
        100.00 * SUM(CASE
                WHEN P_TYPE LIKE 'PROMO%'
                THEN L_EXTENDEDPRICE*(1-L_DISCOUNT)
                ELSE 0
                END) / SUM(L_EXTENDEDPRICE * (1 - L_DISCOUNT)) AS PROMO_RE
VENUE
FROM
        LINEITEM,
        PART
WHERE
        L_PARTKEY = P_PARTKEY
        AND L_SHIPDATE BETWEEN '1995-09-01' and '1995-09-30';  2    3    4
    5    6    7    8    9   10   11   12

PROMO_REVENUE
-------------
   16.3807786

Transcurrido: 00:00:00.84
SQL>
```

Consulta 15:

```
CREATE VIEW REVENUE (SUPPLIER_NO, TOTAL_REVENUE) AS
  SELECT
        L_SUPPKEY,
        SUM(L_EXTENDEDPRICE * (1 - L_DISCOUNT))
  FROM
        LINEITEM
  WHERE
        L_SHIPDATE BETWEEN '1996-01-01' and '1996-03-31'
  GROUP BY
        L_SUPPKEY;
SELECT
  S_SUPPKEY,
  S_NAME,
  S_ADDRESS,
  S_PHONE,
  TOTAL_REVENUE
FROM
  SUPPLIER,
```

```
    REVENUE
WHERE
    S_SUPPKEY = SUPPLIER_NO
    AND TOTAL_REVENUE = (
            SELECT
                    MAX(TOTAL_REVENUE)
            FROM
                    REVENUE
    )
ORDER BY
    S_SUPPKEY;
```

Resultado CDBMS



Resultado RDBMS

```
SQL> SELECT
        S_SUPPKEY,
        S_NAME,
        S_ADDRESS,
        S_PHONE,
        TOTAL_REVENUE
FROM
        SUPPLIER,
        REVENUE
WHERE
        S_SUPPKEY = SUPPLIER_NO
        AND TOTAL_REVENUE = (
                SELECT
                        MAX(TOTAL_REVENUE)
                FROM
                        REVENUE
        )
ORDER BY
        S_SUPPKEY;  2    3    4    5    6    7    8    9   10   11   12
13   14   15   16   17   18   19

 S_SUPPKEY S_NAME                    S_ADDRESS
---------- ------------------------- ------------------------------------
---
S_PHONE          TOTAL_REVENUE
--------------- -------------
      8449 Supplier#000008449        Wp34zim9qYFbVctdW
20-469-856-8873    1772627.21


Transcurrido: 00:00:00.90
SQL>
```

Consulta 16 CDBMS

```
SELECT
   P_BRAND,
   P_TYPE,
   P_SIZE,
   COUNT(DISTINCT PS_SUPPKEY) AS SUPPLIER_CNT
FROM
   PARTSUPP,
   PART
WHERE
   P_PARTKEY = PS_PARTKEY
   AND P_BRAND <> 'brand#45'
   AND P_TYPE NOT LIKE 'MEDIUM POLISHED%'
   AND P_SIZE IN (49, 14, 23, 45, 19, 3, 36, 9)
   AND PS_SUPPKEY NOT IN (
          SELECT
                 S_SUPPKEY
```

```
        FROM
                SUPPLIER
        WHERE
                S_COMMENT LIKE '%CUSTOMER%COMPLAINTS%'
  )
GROUP BY
  P_BRAND,
  P_TYPE,
  P_SIZE
ORDER BY
  SUPPLIER_CNT DESC,
  P_BRAND,
  P_TYPE,
  P_SIZE;
```

Resultado CDBMS

SQL Statements

```
    PARTSUPP,
    PART
WHERE
    P_PARTKEY = PS_PARTKEY
    AND P_BRAND <> 'brand#45'
    AND P_TYPE NOT LIKE 'MEDIUM POLISHED%'
    AND P_SIZE IN (49, 14, 23, 45, 19, 3, 36, 9)
    AND PS_SUPPKEY NOT IN (
        SELECT
            S_SUPPKEY
        FROM
            SUPPLIER
        WHERE
            S_COMMENT LIKE '%CUSTOMER%COMPLAINTS%'
    )
GROUP BY
    P_BRAND,
    P_TYPE,
    P_SIZE
ORDER BY
    SUPPLIER_CNT DESC,
    P_BRAND,
    P_TYPE,
    P_SIZE;
```

Results

```
Brand#42    LARGE PLATED STEEL                  45
Brand#42    LARGE POLISHED STEEL                14
Brand#42    MEDIUM ANODIZED STEEL               14
Brand#42    MEDIUM ANODIZED TIN                 19
Brand#42    MEDIUM BRUSHED COPPER                9
Brand#42    MEDIUM BRUSHED STEEL                14
Brand#42    MEDIUM BURNISHED COPPER             49
Brand#42    MEDIUM BURNISHED NICKEL             23
(First 500 rows)
Execution time: 0.388 seconds
```

Consulta 16 RDBMS

```
SELECT
    P_BRAND,
    P_TYPE,
    P_SIZE,
    COUNT(DISTINCT PS_SUPPKEY) AS SUPPLIER_CNT
FROM
    PARTSUPP,
```

```
    PART
WHERE
    P_PARTKEY = PS_PARTKEY
    AND P_BRAND <> 'brand#45'
    AND P_TYPE NOT LIKE 'MEDIUM POLISHED%'
    AND P_SIZE IN (49, 14, 23, 45, 19, 3, 36, 9)
    AND PS_SUPPKEY NOT IN (
            SELECT
                    S_SUPPKEY
            FROM
                    SUPPLIER
            WHERE
                    S_COMMENT LIKE '%CUSTOMER%COMPLAINTS%'
    )
GROUP BY
    P_BRAND,
    P_TYPE,
    P_SIZE
ORDER BY
    SUPPLIER_CNT DESC,
    P_BRAND,
    P_TYPE,
    P_SIZE
FETCH FIRST 500 ROWS ONLY;
```

Resultado RDBMS

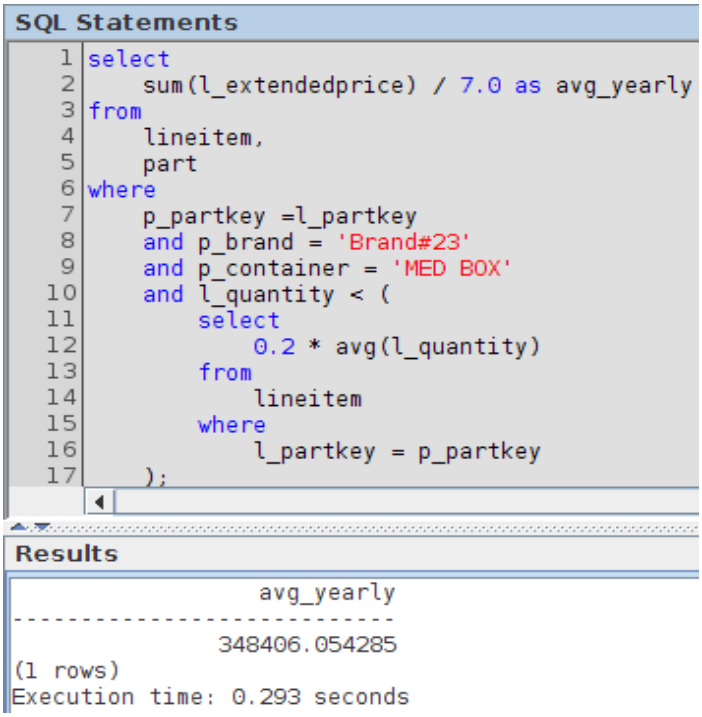
```
P_BRAND    P_TYPE                      P_SIZE SUPPLIER_CNT
---------- ------------------------- ---------- ------------
Brand#41   SMALL POLISHED TIN                14           16
Brand#41   STANDARD BRUSHED NICKEL           45           16
Brand#42   ECONOMY BRUSHED STEEL             14           16
Brand#42   ECONOMY BURNISHED STEEL            9           16
Brand#42   ECONOMY BURNISHED STEEL           45           16
Brand#42   LARGE ANODIZED TIN                23           16
Brand#42   LARGE BRUSHED STEEL              14           16
Brand#42   LARGE BURNISHED NICKEL           19           16
Brand#42   LARGE PLATED STEEL               45           16
Brand#42   LARGE POLISHED STEEL             14           16
Brand#42   MEDIUM ANODIZED STEEL            14           16

P_BRAND    P_TYPE                      P_SIZE SUPPLIER_CNT
---------- ------------------------- ---------- ------------
Brand#42   MEDIUM ANODIZED TIN              19           16
Brand#42   MEDIUM BRUSHED COPPER             9           16
Brand#42   MEDIUM BRUSHED STEEL             14           16
Brand#42   MEDIUM BURNISHED COPPER           49           16
Brand#42   MEDIUM BURNISHED NICKEL           23           16

500 filas seleccionadas.

Transcurrido: 00:00:00.29
SQL>
```

Consulta 17:

```
select
    sum(l_extendedprice) / 7.0 as avg_yearly
from
    lineitem,
    part
where
    p_partkey =l_partkey
    and p_brand = 'Brand#23'
    and p_container = 'MED BOX'
    and l_quantity < (
            select
                    0.2 * avg(l_quantity)
            from
```

```
                lineitem
        where
                l_partkey = p_partkey
);
```

Resultado CDBMS



Resultado RDBMS

```
SQL> select
        sum(l_extendedprice) / 7.0 as avg_yearly
from
        lineitem,
        part
where
        p_partkey =l_partkey
        and p_brand = 'Brand#23'
        and p_container = 'MED BOX'
        and l_quantity < (
                select
                        0.2 * avg(l_quantity)
                from
                        lineitem
                where
                        l_partkey = p_partkey
        );  2    3    4    5    6    7    8    9   10   11   12   13   14
  15   16   17

AVG_YEARLY
----------
348406.054

Transcurrido: 00:00:00.94
SQL>
```

Consulta 18 CDBMS

```
SELECT TOP 100
   C_NAME,
   C_CUSTKEY,
   O_ORDERKEY,
   O_ORDERDATE,
   O_TOTALPRICE,
   SUM(L_QUANTITY)
FROM
   CUSTOMER,
   ORDERS,
   LINEITEM
WHERE
   O_ORDERKEY IN (
          SELECT
                 L_ORDERKEY
```

```
            FROM
                LINEITEM
            GROUP BY
                L_ORDERKEY HAVING
                SUM(L_QUANTITY) > 300
    )
    AND C_CUSTKEY = O_CUSTKEY
    AND O_ORDERKEY = L_ORDERKEY
GROUP BY
    C_NAME,
    C_CUSTKEY,
    O_ORDERKEY,
    O_ORDERDATE,
    O_TOTALPRICE
ORDER BY
    O_TOTALPRICE DESC,
    O_ORDERDATE;
```

Resultado CDBMS

SQL Statements

```
        FROM
            LINEITEM
        GROUP BY
            L_ORDERKEY HAVING
            SUM(L_QUANTITY) > 300
    )
    AND C_CUSTKEY = O_CUSTKEY
    AND O_ORDERKEY = L_ORDERKEY
GROUP BY
    C_NAME,
    C_CUSTKEY,
    O_ORDERKEY,
    O_ORDERDATE,
    O_TOTALPRICE
ORDER BY
    O_TOTALPRICE DESC,
    O_ORDERDATE;
```

Results

```
Customer#000003680              3680     3861123 1998-07-03        433525
Customer#000113131            113131      967334 1995-12-15        432957
Customer#000141098            141098      565574 1995-09-24        430986
Customer#000093392             93392     5200102 1997-01-22        425487
Customer#000015631             15631     1845057 1994-05-12        419879
Customer#000112987            112987     4439686 1996-09-17        418161
Customer#000012599             12599     4259524 1998-02-12        415200
Customer#000105410            105410     4478371 1996-03-05        412754
Customer#000149842            149842     5156581 1994-05-30        411329
Customer#000010129             10129     5849444 1994-03-21        409129
Customer#000069904             69904     1742403 1996-10-19        408513
Customer#000017746             17746        6882 1997-04-09        408446
Customer#000013072             13072     1481925 1998-03-15        399195
Customer#000082441             82441      857959 1994-02-07        382579
Customer#000088703             88703     2995076 1994-01-30        363812
(57 rows)
Execution time: 1.781 seconds
```

Consulta 18 RDBMS

```
SELECT
    C_NAME,
    C_CUSTKEY,
    O_ORDERKEY,
    O_ORDERDATE,
    O_TOTALPRICE,
    SUM(L_QUANTITY)
```

```
FROM
    CUSTOMER,
    ORDERS,
    LINEITEM
WHERE
    O_ORDERKEY IN (
            SELECT
                    L_ORDERKEY
            FROM
                    LINEITEM
            GROUP BY
                    L_ORDERKEY HAVING
                    SUM(L_QUANTITY) > 300
    )
    AND C_CUSTKEY = O_CUSTKEY
    AND O_ORDERKEY = L_ORDERKEY
GROUP BY
    C_NAME,
    C_CUSTKEY,
    O_ORDERKEY,
    O_ORDERDATE,
    O_TOTALPRICE
ORDER BY
    O_TOTALPRICE DESC,
    O_ORDERDATE
FETCH FIRST 100 ROWS ONLY;
```

Resultado RDBMS

```
C_NAME                    C_CUSTKEY O_ORDERKEY O_ORDERDAT O_TOTALPRICE
------------------------- ---------- ---------- ---------- ------------
SUM(L_QUANTITY)
---------------
Customer#000010129             10129    5849444 1994-03-21    409129.85
            309

Customer#000069904             69904    1742403 1996-10-19       408513
            305

Customer#000017746             17746       6882 1997-04-09    408446.93
            303


C_NAME                    C_CUSTKEY O_ORDERKEY O_ORDERDAT O_TOTALPRICE
------------------------- ---------- ---------- ---------- ------------
SUM(L_QUANTITY)
---------------
Customer#000013072             13072    1481925 1998-03-15    399195.47
            301

Customer#000082441             82441     857959 1994-02-07    382579.74
            305

Customer#000088703             88703    2995076 1994-01-30    363812.12
            302


57 filas seleccionadas.

Transcurrido: 00:00:03.44
SQL>
```

Consulta 19:

```
SELECT
   SUM(L_EXTENDEDPRICE * (1 - L_DISCOUNT) ) AS REVENUE
FROM
   LINEITEM,
   PART
WHERE
   (
          P_PARTKEY = L_PARTKEY
          AND P_BRAND = 'Brand#12'
          AND P_CONTAINER IN ( 'SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
```

```
        AND L_QUANTITY >= 1 AND L_QUANTITY <= 1 + 10
        AND P_SIZE BETWEEN 1 AND 5
        AND L_SHIPMODE IN ('AIR', 'AIR REG')
        AND L_SHIPINSTRUCT = 'DELIVER IN PERSON'
)
OR
(
        P_PARTKEY = L_PARTKEY
        AND P_BRAND = 'Brand#23'
        AND P_CONTAINER IN ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
        AND L_QUANTITY >= 10 AND L_QUANTITY <= 10 + 10
        AND P_SIZE BETWEEN 1 AND 10
        AND L_SHIPMODE IN ('AIR', 'AIR REG')
        AND L_SHIPINSTRUCT = 'DELIVER IN PERSON'
)
OR
(
        P_PARTKEY = L_PARTKEY
        AND P_BRAND = 'Brand#34'
        AND P_CONTAINER IN ( 'LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
        AND L_QUANTITY >= 20 AND L_QUANTITY <= 20 + 10
        AND P_SIZE BETWEEN 1 AND 15
        AND L_SHIPMODE IN ('AIR', 'AIR REG')
        AND L_SHIPINSTRUCT = 'DELIVER IN PERSON'
);
```

Resultado CDBMS

SQL Statements

```
19          AND P_BRAND = 'Brand#23'
20          AND P_CONTAINER IN ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
21          AND L_QUANTITY >= 10 AND L_QUANTITY <= 10 + 10
22          AND P_SIZE BETWEEN 1 AND 10
23          AND L_SHIPMODE IN ('AIR', 'AIR REG')
24          AND L_SHIPINSTRUCT = 'DELIVER IN PERSON'
25      )
26      OR
27      (
28          P_PARTKEY = L_PARTKEY
29          AND P_BRAND = 'Brand#34'
30          AND P_CONTAINER IN ( 'LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
31          AND L_QUANTITY >= 20 AND L_QUANTITY <= 20 + 10
32          AND P_SIZE BETWEEN 1 AND 15
33          AND L_SHIPMODE IN ('AIR', 'AIR REG')
34          AND L_SHIPINSTRUCT = 'DELIVER IN PERSON'
35      );
```

Results

```
                                REVENUE
----------------------------------------
                          3083843.0578
(1 rows)
Execution time: 0.112 seconds
```

Resultado RDBMS

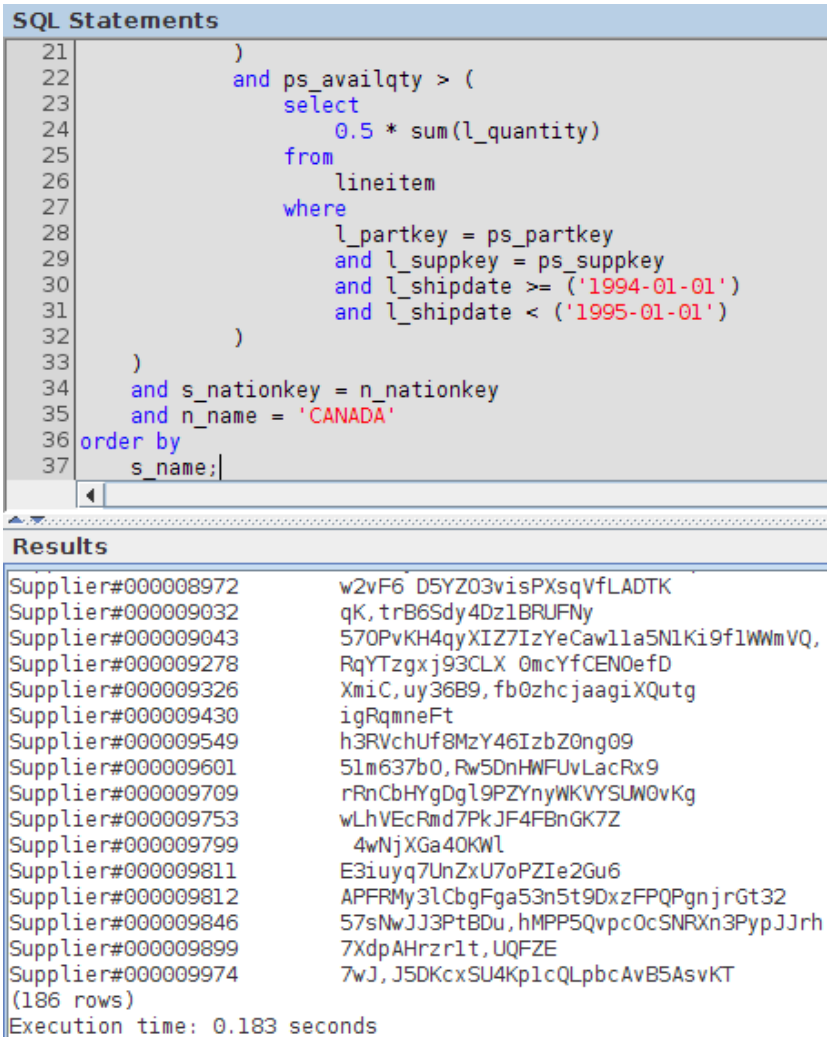
```
WHERE
        (
                P_PARTKEY = L_PARTKEY
                AND P_BRAND = 'Brand#12'
                AND P_CONTAINER IN ( 'SM CASE', 'SM BOX', 'SM PACK', 'SM P
KG')
                AND L_QUANTITY >= 1 AND L_QUANTITY <= 1 + 10
                AND P_SIZE BETWEEN 1 AND 5
                AND L_SHIPMODE IN ('AIR', 'AIR REG')
                AND L_SHIPINSTRUCT = 'DELIVER IN PERSON'
        )
        OR
        (
                P_PARTKEY = L_PARTKEY
                AND P_BRAND = 'Brand#23'
                AND P_CONTAINER IN ('MED BAG', 'MED BOX', 'MED PKG', 'MED
PACK')
                AND L_QUANTITY >= 10 AND L_QUANTITY <= 10 + 10
                AND P_SIZE BETWEEN 1 AND 10
                AND L_SHIPMODE IN ('AIR', 'AIR REG')
                AND L_SHIPINSTRUCT = 'DELIVER IN PERSON'
        )
        OR
        (
                P_PARTKEY = L_PARTKEY
                AND P_BRAND = 'Brand#34'
                AND P_CONTAINER IN ( 'LG CASE', 'LG BOX', 'LG PACK', 'LG P
KG')
                AND L_QUANTITY >= 20 AND L_QUANTITY <= 20 + 10
                AND P_SIZE BETWEEN 1 AND 15
                AND L_SHIPMODE IN ('AIR', 'AIR REG')
                AND L_SHIPINSTRUCT = 'DELIVER IN PERSON'
        );  2    3    4    5    6    7    8    9   10   11   12   13   14
  15   16   17   18   19   20   21   22   23   24   25   26   27   28   29
   30   31   32   33   34   35

   REVENUE
----------
3083843.06

Transcurrido: 00:00:00.95
SQL>
```

Consulta 20:

```
select
   s_name,
   s_address
from
   supplier,
```

```
    nation
where
    s_suppkey in (
            select
                    ps_suppkey
            from
                    partsupp
            where
                    ps_partkey in (
                            select
                                    p_partkey
                            from
                                    part
                            where
                                    p_name like 'forest%'
                    )
                    and ps_availqty > (
                            select
                                    0.5 * sum(l_quantity)
                            from
                                    lineitem
                            where
                                    l_partkey = ps_partkey
                                    and l_suppkey = ps_suppkey
                                    and l_shipdate >= ('1994-01-01')
                                    and l_shipdate < ('1995-01-01')
                    )
    )
    and s_nationkey = n_nationkey
    and n_name = 'CANADA'
order by
    s_name;
```

Resultado CDBMS



Resultado RDBMS

```
S_NAME                    S_ADDRESS
------------------------- ----------------------------------------
Supplier#000008742        HmPlQEzKCPEcTUL14,kKq
Supplier#000008841        I 85Lu1sekbg2xrSIzm0
Supplier#000008872        8D 45GgxJO2OwwYP9S4AaXJKvDwPfLM
Supplier#000008879        rDSA,D9oPM,65NMWEFrmGKAu
Supplier#000008967        2kwEHyMG 7FwozNImAUE6mH0hYtqYculJM
Supplier#000008972        w2vF6 D5YZO3visPXsqVfLADTK
Supplier#000009032        qK,trB6Sdy4Dz1BRUFNy
Supplier#000009043        57OPvKH4qyXIZ7IzYeCaw11a5N1Ki9f1WWmVQ,
Supplier#000009278        RqYTzgxj93CLX 0mcYfCENOefD
Supplier#000009326        XmiC,uy36B9,fb0zhcjaagiXQutg
Supplier#000009430        igRqmneFt

S_NAME                    S_ADDRESS
------------------------- ----------------------------------------
Supplier#000009549        h3RVchUf8MzY46IzbZ0ng09
Supplier#000009601        51m637bO,Rw5DnHWFUvLacRx9
Supplier#000009709        rRnCbHYgDgl9PZYnyWKVYSUW0vKg
Supplier#000009753        wLhVEcRmd7PkJF4FBnGK7Z
Supplier#000009799         4wNjXGa4OKWl
Supplier#000009811        E3iuyq7UnZxU7oPZIe2Gu6
Supplier#000009812        APFRMy3lCbgFga53n5t9DxzFPQPgnjrGt32
Supplier#000009846        57sNwJJ3PtBDu,hMPP5QvpcOcSNRXn3PypJJrh
Supplier#000009899        7XdpAHrzr1t,UQFZE
Supplier#000009974        7wJ,J5DKcxSU4Kp1cQLpbcAvB5AsvKT

186 filas seleccionadas.

Transcurrido: 00:00:00.91
SQL>
```

Consulta 21:

```
SELECT
   S_NAME,
   COUNT(*) AS NUMWAIT
FROM
   SUPPLIER,
   LINEITEM L1,
   ORDERS,
   NATION
WHERE
   S_SUPPKEY = L1.L_SUPPKEY
```

```
    AND O_ORDERKEY = L1.L_ORDERKEY
    AND O_ORDERSTATUS = 'F'
    AND L1.L_RECEIPTDATE > L1.L_COMMITDATE
    AND EXISTS (
        SELECT
            *
        FROM
            LINEITEM L2
        WHERE
            L2.L_ORDERKEY = L1.L_ORDERKEY
            AND L2.L_SUPPKEY <> L1.L_SUPPKEY
        )
    AND NOT EXISTS (
        SELECT
            *
        FROM
            LINEITEM L3
        WHERE
            L3.L_ORDERKEY = L1.L_ORDERKEY
            AND L3.L_SUPPKEY <> L1.L_SUPPKEY
            AND L3.L_RECEIPTDATE > L3.L_COMMITDATE
        )
    AND S_NATIONKEY = N_NATIONKEY
    AND N_NAME = 'SAUDI ARABIA'
GROUP BY
    S_NAME
ORDER BY
    NUMWAIT DESC,
    S_NAME;
```

Resultado CDBMS

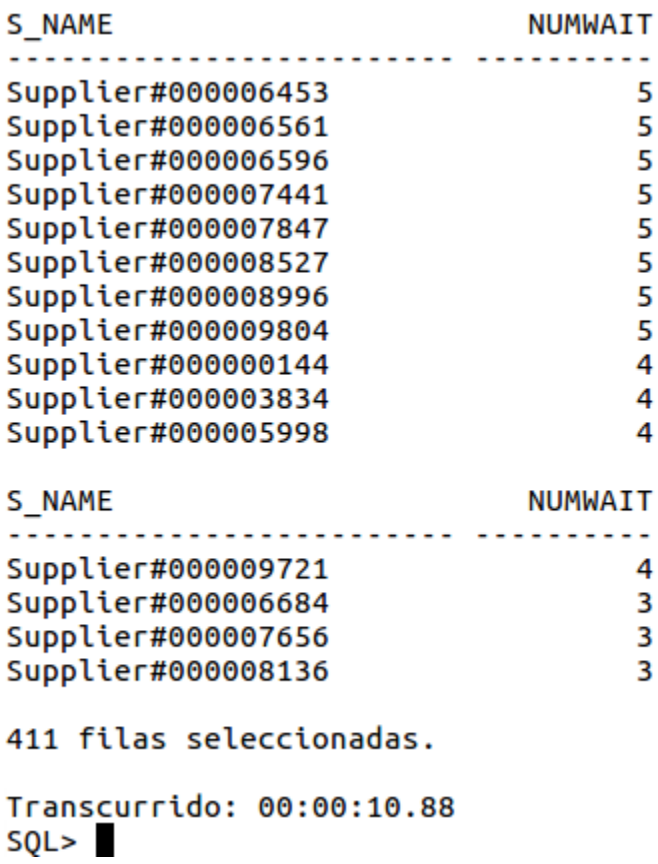
SQL Statements

```
23      AND NOT EXISTS (
24          SELECT
25              *
26          FROM
27              LINEITEM L3
28          WHERE
29              L3.L_ORDERKEY = L1.L_ORDERKEY
30              AND L3.L_SUPPKEY <> L1.L_SUPPKEY
31              AND L3.L_RECEIPTDATE > L3.L_COMMITDATE
32          )
33      AND S_NATIONKEY = N_NATIONKEY
34      AND N_NAME = 'SAUDI ARABIA'
35  GROUP BY
36      S_NAME
37  ORDER BY
38      NUMWAIT DESC,
39      S_NAME;
```

Results

```
Supplier#000006087                          5
Supplier#000006453                          5
Supplier#000006561                          5
Supplier#000006596                          5
Supplier#000007441                          5
Supplier#000007847                          5
Supplier#000008527                          5
Supplier#000008996                          5
Supplier#000009804                          5
Supplier#000000144                          4
Supplier#000003834                          4
Supplier#000005998                          4
Supplier#000009721                          4
Supplier#000006684                          3
Supplier#000007656                          3
Supplier#000008136                          3
(411 rows)
Execution time: 1.622 seconds
```

Resultado RDBMS



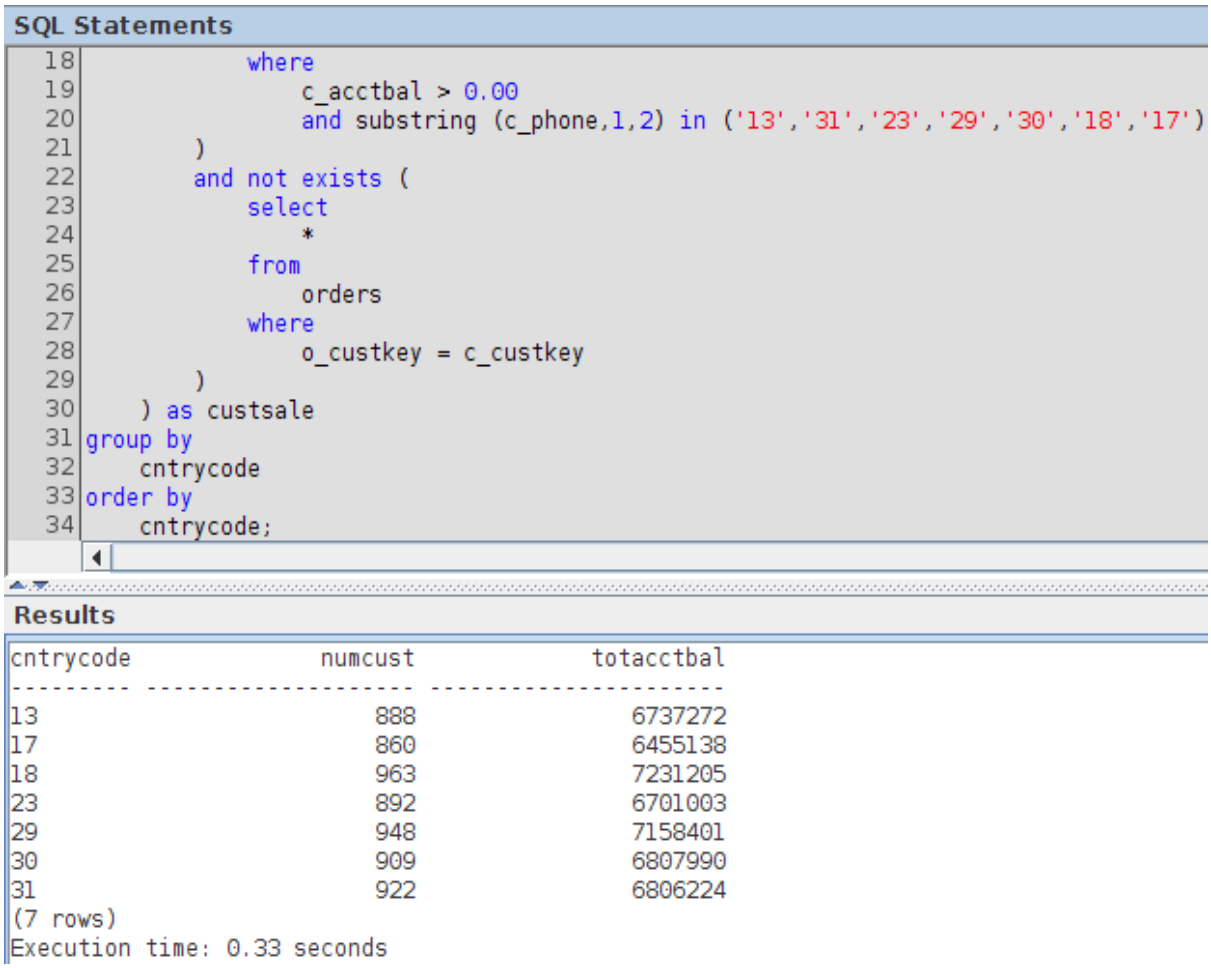
```
S_NAME                         NUMWAIT
-------------------------- ----------
Supplier#000006453                  5
Supplier#000006561                  5
Supplier#000006596                  5
Supplier#000007441                  5
Supplier#000007847                  5
Supplier#000008527                  5
Supplier#000008996                  5
Supplier#000009804                  5
Supplier#000000144                  4
Supplier#000003834                  4
Supplier#000005998                  4

S_NAME                         NUMWAIT
-------------------------- ----------
Supplier#000009721                  4
Supplier#000006684                  3
Supplier#000007656                  3
Supplier#000008136                  3

411 filas seleccionadas.

Transcurrido: 00:00:10.88
SQL>
```

Consulta 22 CDBMS

```
select
    cntrycode,
    count(*) as numcust,
    sum(c_acctbal) as totacctbal
from (
    select
            substring(c_phone,1,2) as cntrycode,
            c_acctbal
    from
            customer
    where
            substring(c_phone,1,2) in ('13','31','23','29','30','18','17')
            and c_acctbal > (
                    select
```

```
                avg(c_acctbal)
        from
                customer
        where
                c_acctbal > 0.00
                and substring (c_phone,1,2) in ('13','31','23','29','30','18','17')
    )
    and not exists (
        select
                *
        from
                orders
        where
                o_custkey = c_custkey
    )
  ) as custsale
group by
  cntrycode
order by
  cntrycode;
```

Resultado CDBMS



Consulta 22 RDBMS

```
select
    cntrycode,
    count(*) as numcust,
    sum(c_acctbal) as totacctbal
from (
    select
            substr(c_phone, 1, 2) as cntrycode,
            c_acctbal
    from
            customer
    where
            substr(c_phone,1,2) in ('13','31','23','29','30','18','17')
```

```
        and c_acctbal > (
                select
                        avg(c_acctbal)
                from
                        customer
                where
                        c_acctbal > 0.00
                        and substr(c_phone,1,2) in ('13','31','23','29','30','18','17')
        )
        and not exists (
                select
                        *
                from
                        orders
                where
                        o_custkey = c_custkey
        )
    ) custsale
group by
    cntrycode
order by
    cntrycode;
```

Resultado RDBMS

```
                and not exists (
                        select
                                *
                        from
                                orders
                        where
                                o_custkey = c_custkey
                )
        ) custsale
group by
        cntrycode
order by
        cntrycode;  2    3    4    5    6    7    8    9   10   11   12
13   14   15   16   17   18   19   20   21   22   23   24   25   26   27
 28   29   30   31   32   33   34

CNTRYCOD    NUMCUST TOTACCTBAL
-------- ---------- ----------
13              888 6737713.99
17              861 6460573.72
18              964  7236687.4
23              892 6701457.95
29              948 7158866.63
30              909 6808436.13
31              922 6806670.18

7 filas seleccionadas.

Transcurrido: 00:00:00.32
SQL>
```

BIBLIOGRAFÍA

- François, J.L.
  URL:
  http://profesores.fi-b.unam.mx/jlfl/Seminario_IEE/Metodologia_de_la_Inv.pdf
- IBM, s.f
  URL:
  https://www.ibm.com/support/knowledgecenter/SSEPH2_13.1.0/com.ibm.ims13.doc.dag/ims_mainstoragedbs.htm
- Mainframe Tutorials, s.f
  URL:
  http://mftuto.blogspot.mx/2009/05/idms.html
- Gálvez, s.f
  URL:
  http://www.lcc.uma.es/~galvez/ftp/bdst/ Tema2.pdf
- Microsoft, s.f
  URL:
  https://technet.microsoft.com/es-es/library/ms190623(v=sql.105).aspx
- Oracle, 2012
  URL:
  http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-explain-the-explain-plan-052011-393674.pdf
- Oracle, 2002
  URL:
  https://docs.oracle.com/cd/B10501_01/server.920/a96533/ex_plan.htm#838
- Sánchez, Jorge, 2004
  URL:
  http://www.jorgesanchez.net/bd/arquOracle.pdf
- Fernández, Jesualdo, s.f
  URL:

http://dis.um.es/~jfernand/0405/dbd/DBD04T06-oracle.pdf

- Vegas, Jesús, 1998
  URL:
  http://www.infor.uva.es/~jvegas/cursos/bd/orarq/orarq.html#3.3.2
- Bertina, Eliso y Martino, Lorenzo 1993
  URL:
  https://books.google.com.mx/books?id=XohLQySVNMC&pg=PA13&dq=bases+de+datos+Orientada+a+Objetos&hl=es&sa=X&ved=0ahUKEwih9N20nZbNAhVKxYMKHbcMBToQ6AEIKDAA#v=onepage&q&f=false
- Besembel, Isable, s.f
  URL:
  http://www.webdelprofesor.ula.ve/ingenieria/ibc/bda/s6ldmc.html
- Marqués, Merche, 2002
  URL:
  http://www3.uji.es/~mmarques/e16/teoria/cap2.pdf
- Versant, 2005
  URL:
  http://web.archive.org/web/20070928121526/http://www.versant.com/developer/resources/objectdatabase/database_fund_man.pdf
- SAP Sybase Blog, s.f
  URL:
  http://sybaseblog.com/2011/01/28/joins-algorithms