Universidad de las Ciencias Informáticas

Facultad 4

Título: Modelo y diseño de un Data Warehouse utilizando vistas materializadas.

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autora:

Heidy Carmona Lora

Tutores:

Lic. Eddy Manuel Infante Alonso

Ing. Julio Cesar Díaz Vera

Ciudad de la Habana

Julio del 2007

Declaración de Autoría

Declaro que soy el único autor de este trabajo y autorizo al <nombre área> de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

_____

Heidy Carmona Lora
Autor

_____ _____

Lic. Eddy Manuel Infante Alonso

Tutor

Ing. Julio Cesar Diaz Vera

Tutor

Agradecimientos

A la **Revolución**, por darme la posibilidad de realizar mis sueños, por permitir mi formación profesional, por hacerme una persona triunfadora.

A la **UCI**, por ser mi hogar, por brindarme tantos buenos momentos, por darme la posibilidad de desarrollarme como persona.

A **Eddy** y a **Julio**, por su asesoramiento tanto en la investigación como en el estudio, por su apoyo, por ser los mejores tutores que pude tener.

A **mami** por creer en mi, por su apoyo, confianza y sacrificio, por ese amor brindado cada día de mi vida, por ser no solo madre sino también amiga, gracias.

A mi **hermana**, por esa forma tan especial de quererme y brindarme su apoyo, por siempre estar ahí para mí, te quiero mucho.

A mi **abuela**, por su amor desmedido, por su fe en mi, por demostrarme que siempre hay que seguir hacia adelante, gracias por todo.

A mi tío el **Dr. Guzmán**, por ser un ejemplo a seguir, por su experiencia, por ayudarme y enseñarme, por ser sobre todas las cosas un padre para mí.

A mi **tía Mirna** por siempre estar ahí en las buenas y en las malas, ¿cómo no agradecerte todo lo que has hecho por mí?

A mi **familia** en general por todo el apoyo brindado en cada momento de mi vida.

A **Mabel**, **Daymara**, **Dayami** y **Ana**, por haberme dado un grandioso quinto año, por su amistad.

A **Hayron**, a **Michel y Nolber**, gracias por la ayuda brindada.

A **Elvio**, **Yunior** y **Osniel** por esos momentos brindados, gracias por su amistad.

A **todos** los que de una forma u otra hicieron mi estancia más placentera en esta universidad.

Dedicatoria

A mami, a mi hermana, a mi abuela, al Dr. Guzmán y a todos,

los que siempre están.

Resumen

Un Data Warehouse facilita la compresión de los datos, transformándolos en información útil y sirviendo de apoyo a la toma de decisiones. Sin embargo se encuentra el problema del costo excesivamente alto de los proyectos Data Warehouse. Para reducir el costo de los proyectos de Data Warehouse es que surge como una alternativa factible el proceso de virtualización del mismo mediante, vistas materializadas, dicho proceso trae consigo el problema del consumo excesivo de espacio de almacenamiento.

La reducción del tiempo de la consulta mediante la selección de un sistema apropiado de vistas materializadas con un bajo costo de almacenamiento es crucial para un eficiente Data Warehouse.

Este trabajo, propone la utilización de un algoritmo eficiente para seleccionar un conjunto apropiado de vistas materializadas, que contemple las consideraciones de almacenaje y de costo de almacenamiento, y que explote con eficacia la ganancia y pérdida métrica.

Palabras claves

Data Warehouse, vistas materializadas, vistas.

Índice de contenido

Índice de figuras

Índice de tablas

Introducción

Desde el inicio de la era de la computadora, su evolución a través del tiempo se hace cada vez mas evidente y algunas organizaciones aun manejan una data no limpia e inconsistente, sobre las cuales se toman decisiones sumamente importantes, por lo que se reconoce que una manera de elevar su eficiencia es hacer un mejor uso de los recursos de información.

Las empresas enfrentan en este momento un creciente enfrentamiento potenciado por la gran competencia de los mercados cada vez más insaciables. Este hecho fomenta su competitividad llevándolas a reestructurar sus modelos de negocio.
Para obtener sistemas de información capaces de tener variantes de producción y estimar indicadores más precisos en tiempo real es necesario la regeneración de procesos y la automatización. De este modo, se colocaran a disposición de los agentes de decisión un conjunto de herramientas y mecanismos de control, incluidos en sistemas de soporte de decisiones. Este nuevo tipo de sistemas va evolucionando de forma clara todas las etapas de un proceso de toma de decisión, ofreciendo un acceso privilegiado a la información de mayor importancia para las actividades del negocio de una empresa.

Los sistemas de soporte de decisiones tienen el papel de integrar toda la información interna y externa en una única plataforma de datos. El objetivo primordial es tener disponible la información necesaria a los agentes de decisiones durante el proceso de toma de decisiones, en las diversas vertientes del negocio. Actualmente se tiende a un aumento de la inversión en investigaciones y al desenvolvimiento de técnicas de optimización. Estas técnicas son utilizadas para el mejoramiento del desempeño de los sistemas de información.

Al tener en cuenta la naturaleza cambiante de los datos, el mundo tiende al análisis integrado de ellos para gestionar sus enormes cantidades. Muchas empresas utilizan Data Warehouse porque integra datos en un solo depósito desde el cual los usuarios terminales pueden fácilmente ejecutar consultas, confeccionar reportes y realizar análisis, además de ser un ambiente para la toma de decisiones, que refuerza los datos almacenados en diferentes fuentes, organizándolos y poniéndolos a disposición de los

encargados de esta responsabilidad en la empresa. En esta tecnología no existen alternativas libres de implementación y los gestores Oracle y SQLServer, comercializan las herramientas de Data Warehouse como módulos de sus gestores que deben ser adquiridos de manera independiente lo cual incrementa el costo de estos productos que ya tienen un precio excesivamente alto en el mercado, por lo que es bastante difícil para empresas pequeñas la utilización de esta tecnología.

Esta técnica es utilizada para la recuperación y la integración de los datos a partir de fuentes distribuidas, autónomas y posiblemente heterogéneas. Los datos son almacenados en un gran deposito llamado Data Warehouse, que resume los datos que son organizados en dimensiones, disponiéndolos para consultas y análisis a través de aplicaciones OLAP y sistemas de soporte de decisiones.

Es de uso frecuente para apoyar el proceso analítico en línea (OLAP), y aumentar estas consultas, almacenar algunos resultados intermedios de la consulta que procesa el Data Warehouse. A estos resultados intermedios almacenados en un Data Warehouse se le llaman vistas materializadas. Es decir, que puede verse un Data Warehouse como un conjunto de vistas materializadas obtenida a través de una o más fuentes de datos.

Puesto que una consulta puede producir muchos resultados intermedios, una de las decisiones importantes en diseñar un almacén de los datos es seleccionar las vistas más convenientes. Si el espacio disponible es suficiente, se quisiera materializar todas las vistas posibles por adelantado para acelerar el proceso de la consulta de un Data Warehouse.

Este trabajo se centrará en el proceso de modelado de un Data Warehouse a partir de la utilización de vistas materializadas. Con el, se le dará solución a la interrogativa de ¿como disminuir el costo de los proyectos Data Warehouse a partir de un modelo de implementación virtual del mismo usando las vistas materializadas?

El objetivo principal es la propuesta de un modelo para la implementación de un Data Warehouse a partir de vistas materializadas, para lo cual se realizará el diseño del Data Warehouse a partir de la utilización de vistas materializadas.
Si se logra modelar un Data Warehouse para implementarlo utilizando vistas materializadas entonces se podrá reducir el costo de los proyectos de implementación.

Hipotéticamente si se cuenta con un Data Warehouse estructurado a partir de vistas materializadas, entonces se minimizará el tiempo de respuesta de las consultas y se reducirá el costo de los proyectos por contar con una herramienta para implementar de forma más eficiente las consultas de sistemas de soporte de decisiones.

Variable Independiente:

Modelo de implementación de un Data Warehouse a partir de vistas materializadas.

Variable Dependiente:

Reducción del costo de los proyectos de implementación.

Se propone en el desarrollo de este trabajo la selección exhaustiva de la bibliografía. Se investigará acerca de las características y facilidades que brinda la utilización de un Data Warehouse, además se estudiará e investigará sobre diferentes algoritmos para la selección de las vistas materializadas. También, se investigará sobre las facilidades que brinda la utilización de las vistas materializadas para crear un Data Warehouse.

CAPITULO 1: Fundamentación teórica

Este capítulo brindará un acercamiento a la descripción de un Data Warehouse, enfatizando en sus características principales y la importancia de su arquitectura, además de comparar los diferentes criterios que existen acerca de la creación de un Data Warehouse. Se ofrecerá una breve descripción de los gestores de bases de datos y se establecerá una breve comparación entre ellos, además de la relación que existe entre un Data Warehouse y las vistas materializadas.

Las empresas de hoy en día se caracterizan por sus estructuras de conducción dinámicas, donde los individuos que las componen deben tomar decisiones en forma rápida y efectiva basados en la última información disponible, para poder así mantener la ventaja competitiva. Por otro lado, las compañías están acumulando grandes volúmenes de datos en sus bases de datos operativas a un ritmo que, en promedio, se duplica cada año. Aún así, sólo el 7% de estos datos es aprovechado para obtener una ventaja en las decisiones de negocios. Recién ahora las organizaciones se están dando cuenta de que existe una significativa cantidad de información que puede ser extraída de sus bases de datos, necesaria para soportar las decisiones que deben ser tomadas por sus ejecutivos, llegando así al concepto de Data Warehouse.

1.1 Data Warehouse y sus características.

El Data Warehouse, es actualmente, el centro de atención de las grandes instituciones, porque provee un ambiente para que las organizaciones hagan un mejor uso de la información que está siendo administrada por diversas aplicaciones operacionales.

Es una colección de datos que se encuentra formada por la información de la empresa y se usa como soporte para la toma de decisiones. Para acelerar el proceso de análisis, consultas y acceder a la información en el menor tiempo posible, es necesario reunir los datos apropiados desde las diversas bases de datos.

Las aplicaciones para soporte de decisiones basadas en un Data Warehouse, pueden hacer más práctica y fácil la explotación de datos para una mayor eficacia del negocio, que no se logra cuando se usan sólo los datos que provienen de las aplicaciones operacionales (que ayudan en la operación de la empresa en sus operaciones

cotidianas), en los que la información se obtiene realizando procesos independientes y muchas veces complejos.

En sí, un Data Warehouse es una colección de datos integrada, orientada a sujetos, variante en el tiempo y no volátil, utilizada como apoyo para los procesos de toma de decisión.

Para comprender lo que expresa esta definición es necesario detenernos en cada uno de los calificadores.

- **Integrada:** Contiene una base de datos centralizada y consolidada que integra datos derivados de toda la organización. Los datos se almacenan en un formato consistente y existe un único esquema de representación.
- **Orientada a sujetos:** Los datos se organizan y se resumen por temas, tales como ventas, finanzas y transportación, para cada uno de los cuales el Data Warehouse contiene sujetos, tales como productos, compradores y regiones. Por tanto, un Data Warehouse se enfoca a las entidades del negocio, lo cual contrasta con los sistemas operacionales que se orientan a los procesos.
- **Variante en el tiempo:** Los datos se asocian con un punto en el tiempo o con un período. La toma de decisiones se apoya en diferentes modelos (estadísticos o de otro tipo) que necesitan información histórica. Esta característica básica de los datos en un Data Warehouse difiere del comportamiento en el ambiente operacional donde los datos reflejan exactamente el momento actual.
- **No volátil:** Los datos no se modifican una vez introducidos (solo lectura). Ello permite la optimización del acceso a los datos, puesto que el sistema no tiene que efectuar frecuentemente los chequeos de integridad requeridos por las operaciones de modificación. Además, se garantiza la disponibilidad de datos históricos.

El almacenamiento de los datos es un acercamiento para integrar la información de múltiples bases de datos operacionales, muy grandes, distribuidas, y heterogéneas. Un Data Warehouse es un depósito de datos que proporciona un ambiente integrado para las preguntas y el análisis de la toma de decisiones que requiere agregaciones complejas de cantidades enormes de los datos históricos.

Los Data Warehouses son muy utilizados por muchas empresas de todo el mundo, ya que proporcionan una fundación sólida de integración de los datos corporativos e

históricos para la realización de análisis gerenciales. Su construcción e implementación se realiza paso a paso, organizando y almacenando los datos sobre una perspectiva a largo plazo.

Así partiendo de los datos históricos básicos se puede realizar el análisis de tendencias. Por sus características básicas, integran los datos provenientes de varias fuentes diferentes, la etapa mas compleja de implementación de un Data Warehouse es el proceso de carga. En este proceso, los datos distribuidos por varios ambientes operacionales (bases de datos de producción que contienen los datos utilizados por varios sistemas de una empresa) deben ser seleccionados, trabajados con el objetivo de estandarizarlos y limpiarlos, transferirlos para un nuevo ambiente y finalmente sean cargados, siempre atendiendo al estándar de modelación utilizado para el Data Warehouse. Este proceso se hace periódicamente, siendo su frecuencia dependiente de varios factores relacionados al modelo de negocio utilizado por la empresa. De esta forma se puede decir que los datos almacenados en el Data Warehouse son para todos los propósitos prácticos. Una vez que los datos son almacenados en el Data Warehouse no sufren actualizaciones, siendo, por tanto un ambiente de carga y acceso.

Un Data Warehouse se crea, al extraer datos desde una o más bases de datos de aplicaciones operacionales, tanto internas como externas. La data extraída es transformada para eliminar inconsistencias y resumir si es necesario y luego, cargarlas en el Data Warehouse. El proceso de transformar, crea el detalle de tiempo variante, resume y combina los extractos de datos, ayudando a crear el ambiente para el acceso a la información Institucional. Luego en el proceso de carga se organiza y actualiza los datos obtenidos de las bases de datos y por último, y no menos importante el proceso de explotación que es en el cual se extrae y se analiza la información almacenada en el Data Warehouse. Este nuevo enfoque ayuda a las personas individualmente, en todos los niveles de la empresa, a efectuar su toma de decisiones con más responsabilidad.

Desde el punto de vista del usuario, el único proceso visible es la explotación, aunque el éxito del Data Warehouse radica fundamentalmente en los tres primeros procesos que alimentan la información del mismo y proporcionan el mayor porcentaje de esfuerzo (en torno a un 80%) a la hora de desarrollar el Data Warehouse.

Después de su creación y la primera carga, el Data Warehouse pasa a sufrir cargas incrementales que debe reflejar el ambiente operacional a lo largo del tiempo tornándose en una inmensa bases de datos para los sistemas de apoyo y decisiones.
Las empresas para mantenerlo, dejan el Data Warehouse disponible para lectura, a través de consultas durante el día, mientras durante la noche permanece indisponible, de forma tal que se pueda realizar el proceso de mantenimiento. Se torna claro, que un Data Warehouse en la gran mayoría de los casos tiene características de un ambiente estático.

En varias áreas del negocio los análisis son basados en resúmenes mensuales, ejemplo de ello es que la alta complejidad en el proceso de carga se transforma en un punto muy susceptible para la introducción de errores y puede llevar al colapso de todo un proceso de toma de decisiones. También la empresa al pasar por proceso de globalización necesitan de un Data Warehouse disponible el mayor tiempo posible, por lo que es importante la implementación de alternativas que reduzcan al máximo el tiempo necesario para la manutención o que se permita que el proceso de manutención sea ejecutado de forma concurrente a las consultas de usuarios, garantizando, sin embargo la consistencia de acceso al Data Warehouse.

Al Data Warehouse sufrir nuevas cargas constantemente, va aumentando su tamaño y debido a esto al utilizarlo aumenta la dificultad de responder las consultas de los usuarios de forma rápida y eficiente.
La alta complejidad del proceso de carga, la prolongada disponibilidad de un Data Warehouse, aliadas a sus características estáticas y las necesidades de mejorar el modelo de datos implementado, busca el análisis de algunas de las alternativas disponibles para una implementación más eficiente y dinámica de un Data Warehouse.

1.2 Importancia de la arquitectura de un Data Warehouse.

El concepto Data Warehouse es sustancial, porque es entendido como la plataforma que concentra toda la información de interés para la organización, sus fuentes de información son tanto las bases de datos corporativas, como otras fuentes externas.

El uso de esta herramienta de suma importancia para el mundo financiero, ya que es mucho más sencillo decidir cuándo invertir, en dónde invertir, en qué productos invertir. Sin duda es una herramienta que hará ahorrar varios miles de pesos a la organización.

Para el área de operaciones, el tema de los inventarios es de vital importancia. Con el uso de un Data Warehouse, se estaría asegurando en cierta manera, que los inventarios nunca se inflarán de más, y que tampoco faltará mercancía.

El contar con una herramienta de soporte a las decisiones es de gran utilidad para las empresas, en especial en el sector minorista, en donde la competencia es cada vez más agresiva, sobre todo con la entrada de las grandes compañías transnacionales.

En nuestros días, la información es un arma estratégica en los ambientes de negocio, pero acceder fácilmente a los datos, analizarlos y convertirlos en información útil no es una tarea sencilla. Un Data Warehouse se diseña para resolver los problemas que los usuarios terminales enfrentan continuamente al trabajar con los datos.

Las herramientas tradicionales de acceso y análisis de los datos no son adecuadas para manipular los grandes volúmenes de información actuales, por lo que se pueden presentar algunas de las dificultades siguientes cuando se necesita información:

Una vista no única de los datos:
Con frecuencia existe la necesidad de saber qué datos están accesibles, dónde y cómo obtenerlos. En ocasiones, los datos se definen por las áreas que los manejan sin tomar en consideración los requerimientos de otras dependencias o las interrelaciones existentes con otros datos de la organización. Así mismo, los datos también pueden duplicarse sin el debido control para aplicaciones específicas. Estas múltiples representaciones expresan un alto grado de redundancia y, por ende, la posibilidad de la existencia de las anomalías de inconsistencia relacionadas con ella.

Diferentes herramientas de usuarios:
Los datos pueden estar ubicados o no en almacenes diferentes y se pueden acceder o no utilizando herramientas de usuario distintas. Incluso, algunas de estas herramientas pueden responder a departamentos locales y, otras, a la organización. En cualquier caso, el empleo de diversas herramientas obliga a que el usuario terminal tenga que asimilarlas.

Problemas de consistencia:

Una vez que el usuario accede a los datos, necesita entenderlos. Usualmente el mismo dato se diferencia de una fuente a otra, quizás en su definición, formato, significado, u otros. Esto contribuye a que se dificulte su combinación y su comparación.

Carencia de datos históricos:
Muchos usuarios necesitan mantener los datos a través del tiempo, registrar los eventos que causan los cambios en los datos y otras informaciones útiles para la toma de decisiones. Generalmente, los sistemas operacionales no manejan información histórica aunque, en ocasiones, archivan datos en varios medios externos como copias de seguridad. Si no existe un tratamiento consecuente de los datos históricos, esta diversidad de soportes puede provocar problemas con el acceso a la información histórica, tales como incongruencia en las versiones, ausencia de datos, desorganización, entre otras.

Conflicto entre tipos de aplicaciones:
Es común que los sistemas operacionales y los sistemas informacionales utilicen una base de datos de forma compartida. Ahora bien, los objetivos de ambos tipos de sistemas son diferentes y no es recomendable que trabajen simultáneamente sobre los mismos datos. Los sistemas informacionales están dirigidos a la consulta de los datos, mientras que los operacionales realizan con frecuencia modificaciones de los datos. Esto puede provocar un conflicto en los resultados que obtienen las aplicaciones.

Problemas en la administración de los datos:
Existen múltiples y complejos tipos de datos en los sistemas operacionales, cada uno soportado por herramientas que, a su vez, también son múltiples y complejas. En este sentido se destaca la carencia de una estructura común, así como la falta de un punto único de control administrativo.

El Data Warehouse se ha convertido en la herramienta idónea para ayudar a los ejecutivos a tomar las decisiones apropiadas, que les permitan seguir compitiendo en el mercado. La innovación de la tecnología de información dentro de este, puede permitir a cualquier organización hacer un uso mas optimo de los datos, como principal elemento para una la toma de decisión de forma más efectiva.

Las organizaciones tienen que aprovechar sus recursos de informáticos para crear la información de la operación del negocio, pero deben considerarse las estrategias

tecnológicas necesarias para la implementación de una arquitectura completa de Data Warehouse.

1.3 Diferentes criterios sobre el Data Warehouse

El término Data Warehouse fue acuñado por Bill Inmon a principios de la década de los ´90 y lo definió de la siguiente manera:

"Un Data Warehouse es una colección de datos ***orientado a sujeto, integrado, variante en el tiempo y no volátil*** *para ayudar al proceso de toma de decisiones gerenciales".*

Obviamente que esta definición, ya clásica, debe tomarse como la definición *"pura"* sobre Data Warehouse. Después de diez años, sin embargo, algunos términos han sido manejados según las necesidades y capacidades del mercado, dando origen a conceptos como el de Data Mart o Data Warehouse volátiles, que ante la imposibilidad de almacenar toda la información histórica, almacenan una foto sobre determinado período.

Ralph Kimball define Data Warehouse de una forma más sencilla y práctica pero igual de importante, un Data Warehouse es:

"Una copia de los datos transaccionales específicamente estructurados para consultas y análisis".

Se puede decir que un Data Warehouse es una base de datos, orientada al análisis de la información histórica contenida en ella. Dependiendo las necesidades de análisis de la organización, puede almacenarse desde unos meses hasta varios años la información. El modelo que soporta la información que contiene, se encuentra diseñado, estructurado e implementado con la finalidad y propósito del análisis y navegación de los datos.

Se entiende por navegación, la posibilidad de ver información correspondiente a diferentes contextos o entornos, por ejemplo, analizar las ventas anuales y poder "abrirlas" por sucursal, después, analizar en más en detalle una sucursal para ver cómo se discriminan las ventas por cada producto.

La "lucha" Kimball contra Inmon es bien conocida. Kimball aseguro en 1997 su modelo multidimensional era "la única manera viable de diseñar bases de datos destinadas a su uso directo por parte de un usuario final".

Casi todos le siguieron la corriente, pero obviamente algunos valientes se le tiraron a la yugular entre ellos Inmon en el año 2000 cuando dijo que… “si diseñas un Data Warehouse desde el punto de vista de análisis de un solo individuo, condenas al resto a su mismo punto de vista y que difícilmente en el modelo dimensional puedes incluir información no incluida en el foco original del análisis”.

En este ámbito la creación de los Data Warehouse tienen dos grandes gurús, por un lado el archiconocido Kimball con su modelo multidimensional, y por el otro el quizás menos conocido, pero no menos importante Inmon.

Estilo Kimball:

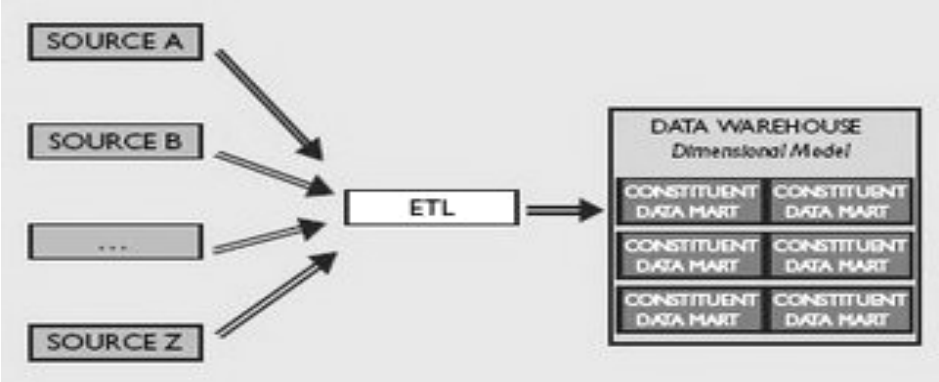

Figura 1 Modelo de Kimball

El modelo de Kimball es mas corporativo en el que un proceso ETL (Extracción, Transformación, y Carga), nutre un espacio Data Warehouse en el que se comparten las dimensiones entre diferentes puntos de vista y en el que los Data Marts de cada departamento se forman utilizando los hechos y las dimensiones ya establecidas para toda la compañía.

Estilo Inmon:

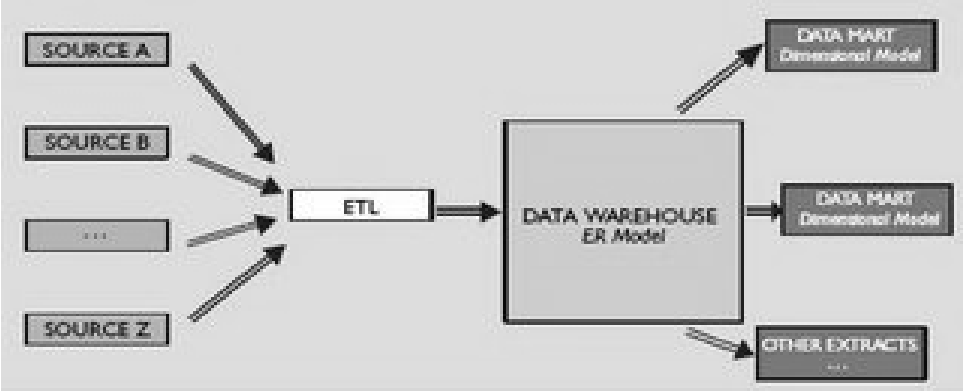

Figura 2 Modelo de Inmon

Coincide con Kimball en un único proceso ETL que nutra un Data Warehouse corporativo, pero el que el nutre no es dimensional es un Data Warehouse basado en el Modelo Entidad-Relación.

La idea de Inmon es que el Modelo Entidad-Relación es mucho más rico y adaptable que el multidimencional.

Una vez que se tiene el Data Warehouse Entidad-Relación corporativo se genera los Data Marts dimensionales que se quiera, y no solo eso, nos puede servir para crear cualquier otra extracción para cualquier otro sistema de toma de decisiones, como puede ser para minería de datos o para sistemas expertos.

Inmon no se cierra a un solo modelo y no solo eso, además su arquitectura mejora la trazabilidad decisional. Con ella se puede desgranar un valor hasta una serie de análisis y reportes que lo expliquen en detalle, tan en detalle como permiten los modelos Entidad-Relación que se tienen en nuestros sistemas operacionales.

Parece maravilloso, pero el problema es que es más costoso de mantener y de implementar. El de Inmon es un modelo que mira a largo plazo y para una metodología ágil el largo plazo es secundario.

El modelo de Kimball es más eficiente en ese aspecto porque es metodología con la rapidez necesaria para realizar un Data Warehouse a corto plazo. Es mas fácil de entender el proceso para llevar a cabo la creación del Data Warehouse y garantiza un mayor rendimiento debido a que el Data Warehouse esta compuesto por Data Mart que

solo esta orientado a una sola actividad y no satisfacen las necesidades de toda la empresa, solo se centra en su objetivo.

1.4 Diferentes gestores de bases de datos

En informática existen los sistemas gestores de bases de datos (SGBD), que permiten almacenar y posteriormente acceder a los datos de forma rápida y estructurada. Los sistemas de gestión de base de datos son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan.

Los SGBD manejan de grandes volúmenes de información con una gran velocidad, es decir en poco tiempo. Trata la información de manera independiente y no hay duplicidad de esta, además de asegurar la protección de la información. Aunque existen varios inconvenientes como el costo de actualización del hardware y software que hoy en día son de muy elevados precios en el mercado, además del elevado pago a los administradores de bases de datos, el mal diseño de las bases de datos que pueden producir en un futuro problemas y el mal adiestramiento de los usuarios.

Existen muchas formas de manejar informáticamente las bases de datos: con Access, Oracle, SQL, PostgreSQL o MySQL, entre otros. Cada sistema tiene sus características, sus ventajas y sus inconvenientes, la elección de uno u otro sistema para gestionar nuestra base de datos vendrá definida por nuestras necesidades.

Un sistema de gestión de bases de datos constituye el núcleo de la base de datos, contiene todas las rutinas necesarias para la gestión de los datos. Siendo una base de datos como un sistema de captación y mantenimiento de registros de forma computarizada, en este sistema se van a poder realizar las operaciones de inserción, borrado y modificación de un dato y modificaciones, borrados e inserciones de información de la estructura de la base de datos.

1.4.1 Algunos gestores de bases de datos:

PostgreSQL es un motor de base de datos, es servidor de base de datos relacionales, conjunto de dos o mas tablas estructuradas en registros (líneas) y campos (columnas), que se vinculan entre sí por un campo en común, en ambos casos posee las mismas características como por ejemplo el nombre de campo, tipo y longitud; a este campo generalmente se le denomina ID, identificador o clave.

PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Cada usuario obtiene una vista consistente de lo último a lo que se le hizo *commit*. Esta estrategia es superior al uso de bloqueos por tabla o por filas común en otras bases, eliminando la necesidad del uso de bloqueos explícitos.

También adicionalmente los usuarios pueden crear sus propios tipos de datos, los que pueden ser por completo indexables gracias a la infraestructura GiST de PostgreSQL.

MySQL es un sistema de gestión de base de datos, multihilo y multiusuario, es un bibliotecario computarizado que administra, gestiona, y opera con nuestros ficheros de datos. Si se le habla en un idioma que entienda los devolverá ordenados, clasificados, seleccionados o ambos inclusive.

Inicialmente, MySQL carecía de elementos considerados esenciales en las bases de datos relacionales, tales como integridad referencial y transacciones. A pesar de ello, atrajo a los desarrolladores de páginas Web con contenido dinámico, justamente por su simplicidad; aquellos elementos faltantes fueron llenados por la vía de las aplicaciones que la utilizan.

MySQL es un sistema de administración relacional de bases de datos. Una base de datos relacional archiva datos en tablas separadas en vez de colocar todos los datos en un gran archivo. Esto permite velocidad y flexibilidad. Las tablas están conectadas por relaciones definidas que hacen posible combinar datos de diferentes tablas sobre pedido.

MySQL es software de fuente abierta. Fuente abierta significa que es posible para cualquier persona usarlo y modificarlo. Cualquier persona puede bajar el código fuente de MySQL y usarlo sin pagar.

Solo MySQL implementa:

- Múltiples motores de almacenamiento (MyISAM, Merge, InnoDB, BDB, Memory/heap, MySQL Cluster, Federated, Archive, CSV, Blackhole y Example en 5.x), permitiendo al usuario escoger la que sea más adecuada para cada tabla de la base de datos.
- Agrupación de transacciones, reuniendo múltiples transacciones de varias conexiones para incrementar el número de transacciones por segundo.

Oracle es un sistema de gestión de base de datos relacional.

Se considera a Oracle como uno de los sistemas de bases de datos más completos, destacando su:

- Soporte de transacciones.
- Estabilidad.
- Escalabilidad.
- Es multiplataforma.

Su mayor defecto es su enorme precio, que es de varios miles de euros (según versiones y licencias). Otro aspecto que ha sido criticado por algunos especialistas es la seguridad de la plataforma, y las políticas de suministro de parches de seguridad, modificadas a comienzos de 2005 y que incrementan el nivel de exposición de los usuarios. En los parches de actualización provistos durante el primer semestre de 2005 fueron corregidas 22 vulnerabilidades públicamente conocidas, algunas de ellas con una antigüedad de más de 2 años.

Aunque su dominio en el mercado de servidores empresariales ha sido casi total hasta hace poco, recientemente sufre la competencia del Microsoft SQL Server de Microsoft y de la oferta de otros RDBMS con licencia libre como PostgreSQL, MySQL o Firebird. Las últimas versiones de Oracle han sido certificadas para poder trabajar bajo Linux.

1.4.2 ¿Qué diferencias hay entre los diferentes gestores?

Diferencias a la hora de funcionar en nuestra Web:

Se puede decir que el “Rolls Royce” de los gestores de bases de datos es Oracle: por su integridad referencial, rapidez en las consultas dado por el número de acceso concurrentes que soportan una gran cantidad de información, además de que se puede realizar una copia de seguridad sin necesidad de paralizar la Web, y es multiplataforma.
Si se decide trabajar con SQL se tendrá que alojar nuestra Web en un servidor con entorno Windows.
El gestor MySQL puede trabajar en entornos tanto Windows como Linux.

Para administrar una base de datos de una pequeña empresa o para gestionar bases de datos personales: libros, discos, contactos, entre otras. Access puede resultar ser una

excelente herramienta, una base de datos en la cual se puede crear tablas relacionadas, personalizar formularios, realizar informes, establecer módulos y acceder a páginas Web. El problema resulta ser cuando se necesita incluir una cantidad importante de información, en ese momento Access resulta ser excesivamente lento para las consultas.

Existen diferencias económicas:

Mientras que MySQL es un software gratuito, con SQL se tendrá que adquirir la versión que se utilice y Oracle para Web puede costar mas de 5 000 euros.

Diferencias Operativas:

El precio excesivamente alto de Oracle determina que casi ninguna empresa tenga este servicio, para usarlo se tendrá que disponer de un servidor exclusivo con las implicaciones que supone a nivel técnico y objetivos a largo plazo.
La Web de una gran empresa, que se puede permitir tener técnicos para controlar la seguridad de un servidor, que necesita soportar múltiples conexiones a su base de datos, cuyos usuarios interactúan con una gran cantidad de información, podría elegir a Oracle como su gestor de base de datos.

MySQL se encuentra en el otro extremo de la oferta. Es la opción que nos plantean todos los servidores de hosting gratuito que soportan bases de datos y es casi imposible encontrar una empresa de hosting de pago que no lo soporte

1.4.3 Comparación entre diferentes gestores:

Tabla 1 Información general de los diferentes gestores.

	Creador	**Fecha de la primera versión pública**	**Ultima versión estable**	**Licencia de software**
MySQL	MySQL AB	Noviembre de 1996	5.0	GPL o propietario
Oracle	Oracle Corporation	1977	10g Release 2	Propietario
PostgreSQL	PostgreSQL Global	Junio de 1989	8.2.3	Licencia BSD

	Development Group			
Microsoft SQL Server	Microsoft	1989	9.00.2047 (2005 SP1)	Propietario

Tabla 2 Soporte del sistema operativo.

	Windows	Mac OS X	Linux	BSD	Unix	z/OS
Microsoft SQL Server	Sí	No	No	No	No	No
MySQL	Sí	Sí	Sí	Sí	Sí	Quizá
Oracle	Sí	Sí	Sí	No	Sí	Sí
PostgreSQL	Sí	Sí	Sí	Sí	Sí	No

Tabla 3 Características fundamentales de los gestores.

	ACID	Integridad referencial	Transacciones	Unicode
Microsoft SQL Server	Sí	Sí	Sí	Sí
MySQL	Depende 1	Depende 1	Depende 1	Sí
Oracle	Sí	Sí	Sí	Sí
PostgreSQL	Sí	Sí	Sí	Sí

Nota (1): Para las transacciones y la integridad referencial, el tipo de tabla InnoDB debe ser usado; el tipo de tabla por defecto, MyISAM, no soporta estas características. Sin embargo, inclusive el tipo de tabla InnoDB permite el almacenamiento de valores que excedan el rango de datos; algunas vistas violan la limitación de ACID.

Tabla 4 Tablas y vistas.

	Tabla temporal	Vista materializada
Microsoft SQL Server	Sí	Similar 3
MySQL	Sí	No
Oracle	Sí	Sí
PostgreSQL	Sí	No 2

Nota (2): La vista materializada puede ser emulada con PL/PgSQL .

Nota (3): El servidor MS SQL provee vistas indexadas.

Tabla 5 Indices.

	Árbol R-/R+	Hash	Expresión	Parcial	Reversa	Mapa de bits
Microsoft SQL Server	?	?	No	No	No	No
MySQL	Tablas MyISAM solamente	Tablas HEAP solamente	No	No	No	No
Oracle	Edición EE solamente	?	Sí	No	Sí	Sí
PostgreSQL	Sí	Sí	Sí	Sí	No	No

Table 6 Otros objetos

	Dominio	Cursor	Trigger	Función 5	Procedimiento 5	Rutina externa 5
Microsoft SQL Server	No	Sí	Sí	Sí	Sí	Sí
MySQL	No	Sí 4	Sí 4	Sí 4	Sí 4	Sí
Oracle	Sí	Sí	Sí	Sí	Sí	Sí
PostgreSQL	Sí	Sí	Sí	Sí	Sí	Sí

Nota (4): Estos objetos de base de datos son disponibles a partir de MySQL 5.0 disponible desde 24/12/2005.

Nota (5): Función y procedimiento se refieren a las rutinas internas escritas en SQL o lenguajes procedurales como PL/SQL. Rutina externa se refiere a la escritura en los lenguajes anfitriones como C, Java, Cobol, etc. "Procedimiento almacenado" es un término comúnmente usado para ese tipo de rutinas. Sin embargo, su definición varía entre diferentes vendedores de bases de datos.

1.5 Data Warehouse y vistas materializadas.

Otra forma de implementar un Data Warehouse es definirlo como un conjunto de vistas materializadas que integran los datos a partir de múltiples fuentes de datos heterogéneas, y eventualmente distribuidas de información.

Una vista es una relación derivada, definida en términos de relaciones bases que es computada todas las veces donde se hace referencia. Una vista esta materializada cuando esta realmente almacenada en una base de datos en vez de ser computada a partir de las relaciones bases en respuesta a consultas. Una vista materializada puede ser vista como una caché, una copia de datos que puede ser accedida rápidamente.

Mas allá de definir el propio Data Warehouse como un conjunto de vistas materializadas basadas en datos de ambientes operacionales, estas vistas también pueden ser utilizadas como objetivo de optimizar consultas complejas, siendo que, para esto, se debe definir un conjunto compartido de vistas, cuidadosamente escogidas a partir de un análisis de las consultas mas frecuentes ejecutadas en un Data Warehouse. Estas vistas, al ser materializadas, apresurarían el acceso de datos necesarios para la realización de consultas en el Data Warehouse.

Para aumentar la eficacia de las consultas, un acercamiento de uso general, es almacenar algunos resultados intermedios de la consulta que procesa en el Data Warehouse. Estos resultados intermedios, almacenados en un Data Warehouse se llaman, las vistas materializadas.

Las vistas materializadas se pueden utilizar por una amplia variedad de consultas de los datos de las bases de datos. Ellas son extremadamente útiles en la optimización de la consulta. La más reciente investigación, es que se pretende integrar la información distribuida, y se ha concentrado en desarrollar un eficiente sistema de intermediarios que no sólo proporcionan un alto grado de independencia para los usuarios locales, también apoyan la integración flexible de las funciones requeridas para los usuarios globales. Sin embargo, ha habido poca atención prestada a la puesta en práctica de las consultas globales definidas en el intermediario. Es posible acelerar la ejecución de una consulta global, si el resultado anterior de una vista materializada es un intermediario. Puesto que el esquema de integración de un intermediario puede ser modificado incrementalmente y su uso puede también ser variado continuamente, el uso de la vista se debe supervisar cuidadosamente, para determinar el sistema optimizado de vistas materializadas. Además, como el número de vistas aumenta, el proceso de la optimización puede ser demasiado largo, de modo que el sistema optimizado identificado por un proceso largo, se convierta en obsoleto, debido al cambio reciente del uso de la vista. Se propone la adaptación de la selección de un método práctico para cada vista global, tal que el

almacenaje disponible en un intermediario se puede utilizar en cualquier momento (S. Agrawal 2000).

Reescribir una consulta basada en el uso de un sistema de vistas materializadas puede rendir un plan mucho más eficiente de la ejecución de la consulta para la consulta de OLAP en el ambiente de Data Warehouse (S. Flesca 2001). Aunque las vistas materializadas proporcionan una gran ventaja para las consultas de OLAP, la selección de un sistema de vistas concerniente a los diversos niveles de la agregación que se materializará en el Data Warehouse se debe hacer cuidadosamente, considerando el compromiso del funcionamiento de la consulta con restricciones del mantenimiento de la vista. Sin embargo, seleccionando el sistema apropiado de vistas materializadas es una tarea no trivial y es un problema completo (Gupta 1997).

Desde 1995, los investigadores han estado interesados en el estudio de vistas materializadas, generalmente basado en un todo o nada, modo para las vistas materializadas. Harinarayan y otros. (V. Harinarayan 1996) propusieron el algoritmo codicioso, basado en el concepto de la derivación de un grafico en el cual una vista materializada superior depende de otras vistas. En el primer paso, el algoritmo elige siempre la materialización de la vista que se puede utilizar para derivar cualquier otra vista. En el segundo paso, para cada vista en el gráfico que todavía no pertenece al sistema de vistas materializadas, el algoritmo determina la ventaja total de materializar esta vista. Entre estas vistas analizadas, el algoritmo selecciona la vista que maximiza la ventaja y la agrega a un sistema materializado. Este segundo paso se repite hasta que un número resuelto de vistas materializadas se selecciona o el límite del espacio de almacenaje disponible se alcanza. Este acercamiento particular no considera el costo de mantenimiento.

Ross y otros, (K.A. Ross 1996) consideraban el uso de vistas adicionales para reducir costo de mantenimiento. Liang y otros, (W. Liang 2001) propusieron un algoritmo de dos etapas que optimiza el tiempo de reacción total de la consulta en la primera etapa, y elige las vistas para la materialización bajo restricciones, dado el tiempo del mantenimiento en la segunda etapa. Desafortunadamente, si el espacio de almacenaje es inadecuado para contener todas las vistas que sean dependientes en otras vistas, entonces el algoritmo codicioso tiene el peor funcionamiento. Chen y Liu (Y.H. Chen 1997) consideraban que las vistas materializadas, es un método para aumentar la eficacia de la consulta de un

Data Warehouse. Sin embargo, uno encuentra el problema de la escasez del espacio, si todas las vistas posibles se materializan por adelantado. La reducción de tiempo de la consulta por medio de seleccionar un sistema apropiado de vistas materializadas con un costo más bajo es crucial para el almacenamiento eficiente de los datos. El propósito es, seleccionar un sistema apropiado de vistas materializadas bajo apremios del almacenaje y de costo y ayudar al incrementar la velocidad los datos enteros que almacenan proceso. Utilizando el algoritmo bifásico para seleccionar vistas materializadas. El propósito de la primera fase es, encontrar el espacio de almacenaje mínimo para que las vistas materializadas contesten a todas las preguntas posibles.

Yang y otros, (J. Yang 1997) propusieron un marco del análisis para las vistas materializadas, usando el plan de proceso de múltiples vistas (MVPP). El método de MVPP considera el costo como factor de la materialización de la vista que no toma el espacio requerido en consideración. Lin y Kuo (W.Y. Lin 2004) propusieron una solución mejorada del funcionamiento usando algoritmos genéticos. Gupta y Mumick (H. Gupta 1999) concluyeron que el almacenaje de datos era tan barato, que desarrollaron un método para seleccionar vistas materializadas con eficacia, dado el bajo costo de mantenimiento, sin la preocupación por apremios del espacio.

Theodoratos y Sellis (D. Theodoratos 1999) propusieron vistas simples y vistas auxiliares. Las vistas simples son para las consultas de usuarios; las vistas auxiliares se utilizan para mantener las vistas simples. Utilizaron el algoritmo exhaustivo incremental para analizar vistas materializadas. Puesto que el algoritmo recurrente toma tanto tiempo, los autores utilizaron el algoritmo re-codicioso para restringir el espacio de la búsqueda y los últimos algoritmos más usados de la heurística son para suprimir el espacio innecesario de la búsqueda.

Liang y otros, (W. Liang 1999) idearon un algoritmo para encontrar un sistema tan auxiliar de la vista explotando la información compartida entre las vistas auxiliares y materializó las vistas, de tal modo que redujo el número total de vistas auxiliares.

Park y otros (C.S. Park 2002) propusieron un algoritmo heurístico para encontrar un sistema de vistas materializadas y de sus regiones que darían lugar a un plan eficiente de la consulta. El algoritmo consiste en varios pasos principales. En el primer paso, el

algoritmo heurístico selecciona las vistas materializadas que serán utilizadas para reescribir, determinando regiones de la consulta. En el segundo paso, el algoritmo genera los bloques de la consultas para las vistas materializadas seleccionadas usando regiones de la consulta. El paso pasado integra los bloques de la consulta en una consulta rescrita final. Puede haber muchos rescritos equivalentes que contienen un diverso sistema de las vistas materializadas candidatas que se diferencian grandemente en el funcionamiento de la ejecución.

Zhang y otros. (C. Zhang 2001) aplicaron un algoritmo evolutivo híbrido para la selección de vistas materializadas, consistiendo en dos niveles de proceso del algoritmo. El algoritmo de alto nivel busca buenos planes de proceso globales, de los planes de proceso local basados en consultas. El algoritmo de nivel inferior selecciona el mejor sistema de vistas materializadas con el costo mínimo total para un plan de proceso global particular. Este algoritmo híbrido se realiza mejor que el algoritmo heurístico en términos de ahorros de costo, pero requiere un tiempo más largo de cómputo. Yu y otros. (J.X. Yu 2003) extendieron el trabajo de Zhang, proponiendo un algoritmo evolutivo, obligado con un procedimiento estocástico de la graduación para solucionar el problema de la selección de vistas reduciendo el costo de mantenimiento. Se presentó un modelo del costo basado en un trabajo previo (D.L. Yang 2002) para determinar las vistas materializadas deseables en las cuales el costo total de proceso de la consulta y el mantenimiento se reducen al mínimo.

Yang y otros, (D.L. Yang 2002) utilizaron la posición del punto-corte del espacio de almacenaje real, dado entre el espacio del candidato y el espacio mínimo para tomar decisiones mejores en la determinación de, si suprimir algunas vistas materializadas candidatas, o agregar más vistas materializadas al espacio mínimo.

Existen diversas maneras para desarrollar de forma más eficiente y efectiva, la utilización de vistas materializadas para la creación de un Data Warehouse.

En la siguiente sección, se analizarán diferentes algoritmos de materialización de vistas y se escogerá el más efectivo para determinar las mejores vistas materializadas en las cuales, el costo total del proceso de la consulta y el mantenimiento se reduzcan al mínimo.

CAPITULO 2: Proceso de virtualización de un DWH.

Un Data Warehouse (DWH), se puede ver como un conjunto de vistas materializadas, donde guardan la información extraída de una o mas bases de datos que la organización utiliza en sus sistemas operacionales.

Una vista materializada, es una consulta cuyo resultado está almacenado en una base de datos. Las consultas que pudieran utilizar vistas materializadas ya almacenadas, pueden ser ejecutadas de forma mucho más rápida, siendo que, para consultas complejas que envuelven grandes volúmenes de datos, esta alternativa favorece dramáticamente los resultados: de horas o días para minutos o segundos. Las vistas materializadas son vistas como una de las principales opciones para el control de desempeño de un Data Warehouse.
La desventaja de estas vistas materializadas, son las alteraciones hechas a los datos bases, a partir de los cuales una vista materializada es definida, se convierte entonces en una vista desactualizada. Para que una vista pueda ser nuevamente sincronizada, será necesario recrear una vista a partir de los datos origen, y actualizarla incrementalmente.

El uso de vistas materializadas en los Data Warehouse, requiere del análisis de la selección de las vistas más adecuadas llevando en cuenta los costos de mantención, los aspectos de materialización y utilización de las vistas para responder las consultas, además de los mecanismos que permiten la propagación correcta, cuando hay actualizaciones de las fuentes de datos bases, utilizando el concepto de las mismas vistas materializadas.

Para los servicios informativos, las posibles vistas necesitan ser restauradas siempre que las fuentes de datos cambien para asegurar validez, exactitud, y modernidad. Esto aumentará el costo de mantenimiento de las vistas materializadas. La selección de las vistas materializadas, implica una compensación difícil entre el funcionamiento de la consulta y el mantenimiento del costo, además de una utilización mejor del espacio de almacenaje. Sin embargo, la limitación del almacenaje prohíbe la materialización de todas las vistas posibles.

En este contexto, surge uno de los desafíos que tiene la utilización de vistas materializadas. Se trata de la selección de un conjunto de vistas que minimice el tiempo de procesamiento de las consultas y el tiempo de procesamiento de las vistas. Esta cuestión es concebida como el problema de selección de vistas a materializar.

2.1 Selección de las vistas a materializar

Una vista es una función que parte de un conjunto de tablas bases para una tabla derivada, siendo recalculada todas las veces que es referenciada.

La materialización de un conjunto de vistas, es una técnica muy utilizada en base de datos. Materializar una vista en una base de datos, consiste en almacenar las tuplas resultantes del procesamiento en una tabla.

Con el almacenamiento de las tuplas de las vistas en la base de datos, puede resultar entonces, la materialización, con la cual el acceso a estas vistas materializadas será más rápido que el recálculo, que es ejecutado todas las veces que las vistas son referenciadas.

Un Data Warehouse contiene múltiples vistas y esto hace que puedan estar relacionadas unas con otras, lo que resulta, a veces, ser más eficiente tan solo materializar cierta parte de estas vistas, por lo que se requiere seleccionar las vistas que serán materializadas.

Esta selección, se basa en determinaciones de un conjunto de vistas, donde el objetivo principal es seleccionar un conjunto apropiado de vistas que minimice el tiempo de respuesta de las consultas o el costo de manutención de las vistas elegidas, dado cierto número de recursos, como son: el tiempo de materialización y espacio para el almacenamiento, entre otras.

En sí, la materialización de vistas consiste en la anticipación del procesamiento y almacenamiento de las tuplas resultantes en una tabla. En efecto, el tiempo de respuesta de una consulta es menor, si las operaciones intermedias como selecciones, proyecciones, uniones y agregaciones, se encuentran ya almacenadas en una tabla.

2.1.1 Descripción resumida de los algoritmos de (J. Yang 1996):

Para la selección de las vistas a ser materializadas, existen diversos algoritmos, algunos definen aspectos de un modelo para el diseño de las vistas materializadas, seleccionando un conjunto de resultados intermedios de las consultas que serán materializadas de forma que el costo total de estas sea el menor. También se lleva en cuenta la frecuencia de las consultas y la frecuencia de las actualizaciones en los datos base.

Todo este proceso de selección de las vistas a ser materializadas también puede estar basado en el plan de procesamiento de múltiples vistas (Multiple View Processing Plan, MVPP), generado por un algoritmo que parte de planos individuales de cada consulta.

El trabajo realizado por (J. Yang 1996) esta basado en una base de datos ejemplo, que contiene las siguientes relaciones:

Product (Pid, name, Did)
División (Did, name, city)
Order (Pid, Cid, quantity, date)
Customers (Cid, name, city)
Part (Tid, name, Pid, supplier)

Para el acceso al Data Warehouse se utilizan diferentes consultas, suponiendo que se tienen las siguientes consultas:

Consulta 1: ***select*** Product. name
from Product, División
where División. city = “LA” and Product. Did= División. Did

Consulta 2: ***select*** Part. name
from Product, Part, División
where División. city = “LA” and Product. Did= División. Did
and Part. Pid=Product. Pid

Para cada una de estas consultas será generado un grafo de procesamiento, que representa el plano de acceso individual:

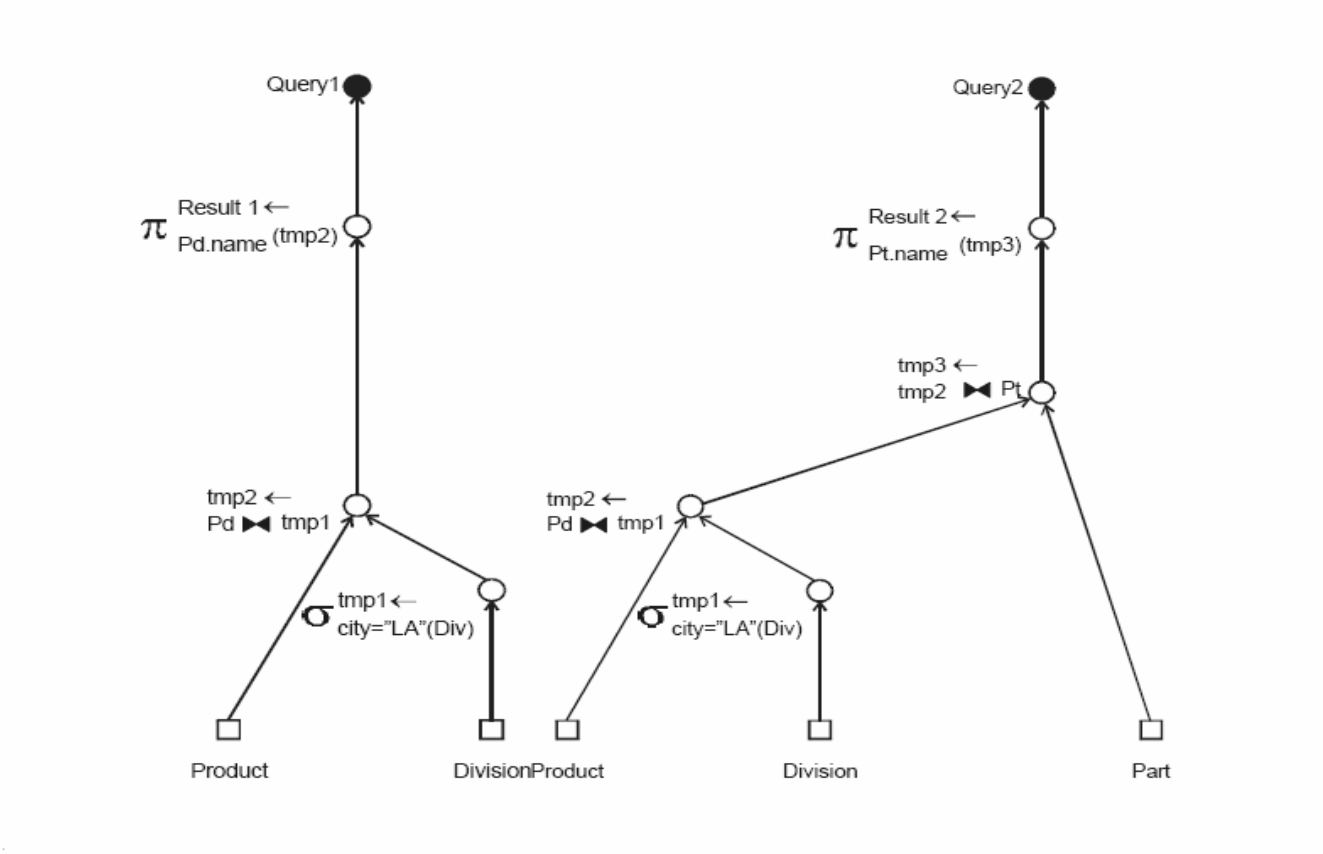

Figura 3 Plano de acceso individual para las consultas 1 y 2.

En los diagramas para una mejor representación se simplifican Pd de Product, Div de Division, Ord de Orden, Cust de Customer, Pt de Part.

Para la obtención de un rápido tiempo de respuesta de las consultas, una de las alternativas a utilizar seria la materialización de los intermedios de cada plano de acceso individual, de esta forma, entonces en el intermedio temp2 la consulta 1 es equivalente a la consulta 2, las cuales son llamadas sub-expresiones comunes. Por consiguiente queda formado un plano de los planos de la consulta 1 y 2:

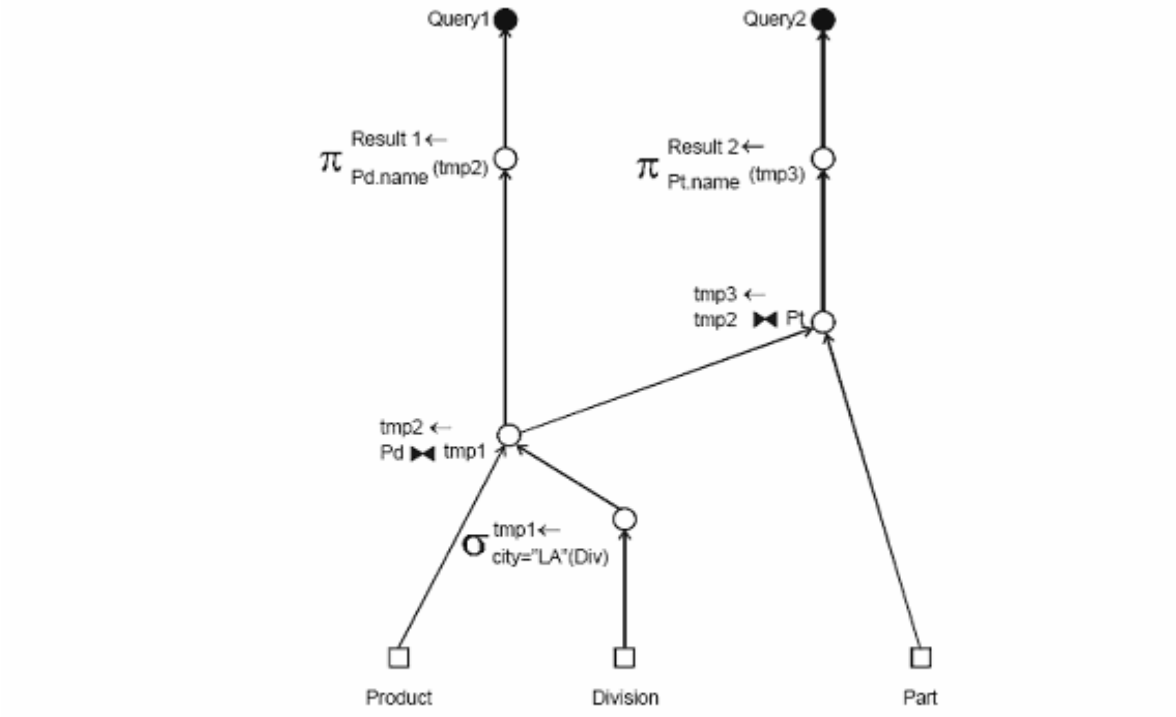

Figura 4 Plano de acceso combinado para las consultas 1 y 2.

Si se materializa temp1 entonces puede ser utilizado para las consultas, en vez de acceso a las relaciones bases, disminuyendo de esta forma el costo de ejecución. Además de que seria de gran beneficio por ser el costo de manutención de las vistas mucho menor.

Suponiendo que otras dos consultas de uso frecuente en el Data Warehouse son:

Consulta 3: ***select*** Customers. name, Product. name, quantity
from Product, División, Order, Customers
where División. city = "LA" ***and*** Product. Did= División. Did ***and***
Product. Pid= Order. Pid ***and*** Order. Cid= Customers. Cid ***and***
date > 7/ 1/ 96

Consulta 4: ***select*** Customers. city, date
from Order, Customers
where quantity > 100 and Order. Cid = Customers. Cid

En la Fig. 5 se representa un plan de acceso global llamando Multiple View Processing Plan (MVPP), donde se combinan los planos individuales de las cuatro consultas, después se puede decidir cuales serán materializadas, de forma tal que el costo de la consulta de la vista materializada sea el mínimo.

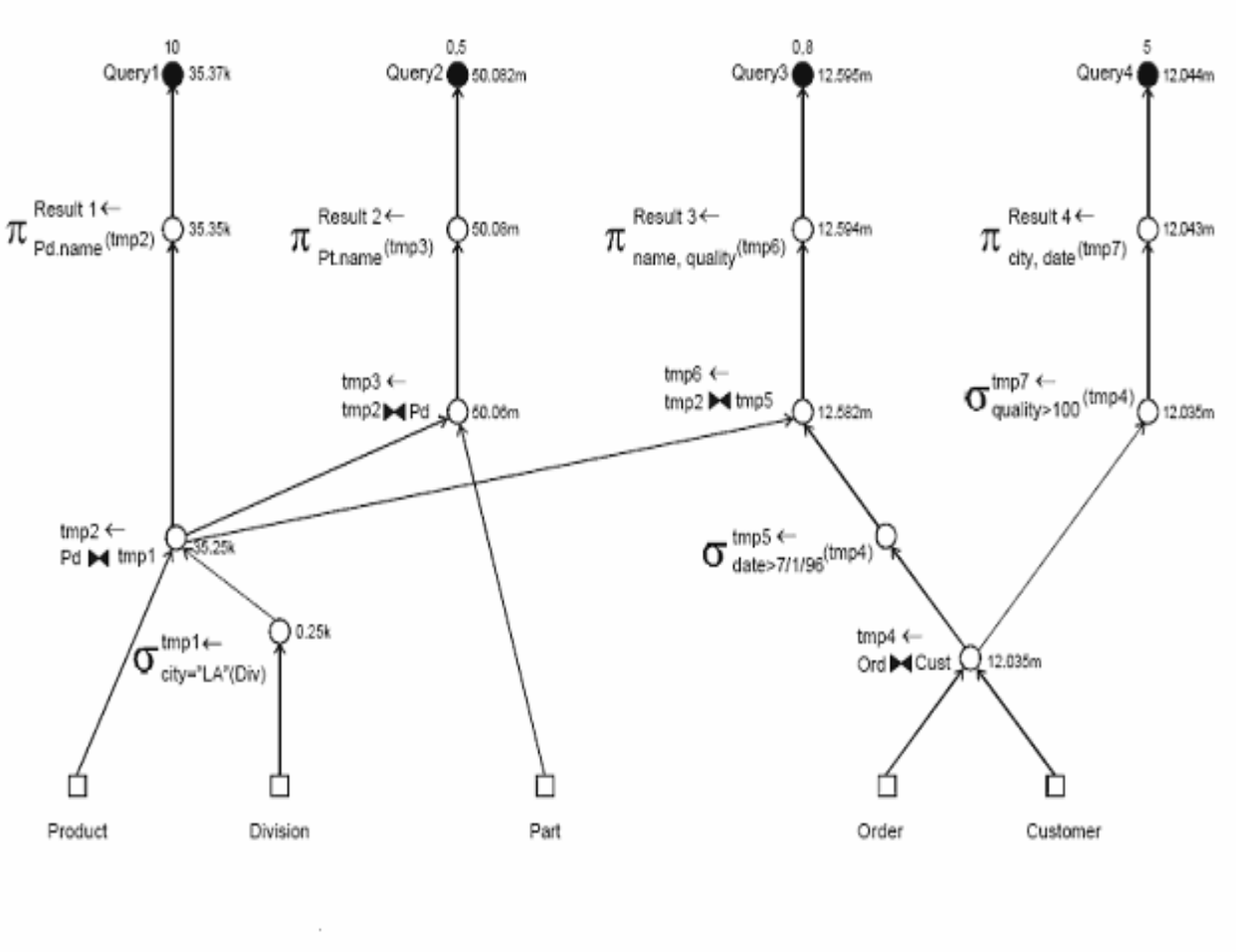

Figura 5 Un MVPP para el ejemplo.

Existen varias opciones para materializar el conjunto de vistas a seleccionar, una sería materializar todas las consultas, y otra materializar solo algunos de los intermedios, por ejemplo: temp1, temp2, entre otros.
Para tomar la mejor decisión es necesario calcular el costo de cada alternativa en términos de manutención de la vista.

En conclusión el algoritmo para la definición de MVPP normalmente obtiene uno o varios planos globales basados en las diferentes combinaciones de los planos individuales. Este algoritmo parte de los planos individuales óptimos y los ordena, basándose en la

frecuencia de las consultas y en los costos, una vez ordenado, combina las sub-expresiones comunes de los planos individuales, llegando a un conjunto de planos globales o un conjunto de MVPP. Cada MVPP generado, tiene su costo total y será analizado por un segundo algoritmo que selecciona los intermedios que serán materializados. Este algoritmo trabaja dado un MVPP, con el objetivo de definir un conjunto de vistas materializadas tal que, el costo de manutención de estas vistas sea el mínimo, por lo que compara los costos de cada combinación posible. Finalmente este algoritmo compara el costo total de cada MVPP generado, seleccionando el de menor costo.

2.1.2 Algoritmo A* y BUPS:

Estos algoritmos son los llamados algoritmos determinísticos, porque devuelven una sola solución por búsqueda exhaustiva, o sea, devuelven soluciones próximas a la más óptima. Usualmente las soluciones dadas por este tipo de algoritmo, son utilizadas como punto de partida para otros algoritmos.

2.1.2.1 Definición del problema.

Sea *L* (un grafo acíclico por ejemplo) el espacio de soluciones y $M\,(M \subseteq L)$ un conjunto de vistas a materializar. Cada vista $v \in L$ tiene tres valores asociados: a la frecuencia de la consulta f_v, la frecuencia de actualización g_v, y el costo de lectura r_v. Según (Panos Kalnis 2002) el costo de responder a una consulta *q* (que corresponde a una vista *v*) corresponde al tamaño de *v*, o sea, al número de tuplas (r_v).

El problema de la determinación de un espacio *L* de soluciones es abordado en (Dimitri Theodoratos 2004). En esta fase apenas se atiende al problema de la selección del conjunto de vistas *M* a materializar suponiendo la existencia de *L*. El costo de almacenamiento de *M* esta dado por:

$$S(M) = \sum_{v \in M} r_v \ . \qquad (1)$$

Sea *u* (*v*, M) el costo de actualización de la vista *v* cuando *M* es el conjunto de vistas materializadas. Entonces el costo de actualización de *M* es dado por:

$$U(M) = \sum_{v \in M} g_v \times u(v, M) \ . \qquad (2)$$

Sea *q* (*v*, M) el costo de si responder a una consulta *v* cuando *M* es el conjunto de vistas materializadas. El costo total de responder las consultas es dado por:

$$Q(M) = \sum_{v \in L} f_v \times q(v, M)\,. \qquad (3)$$

El problema de la selección de vistas a materializar consiste en escoger un conjunto *M* tal que *Q (M)* es mínimo y que se respeten las restricciones S (M) ≤ Smax y U (M) ≤ Umax. Siendo *Smax* el espacio disponible para materializar y *Umax* la ventana temporal disponible para la manutención del conjunto de vistas materializadas.

2.1.2.2 Descripción de los algoritmos A* y BUPS.

El algoritmo BUPS presentado por *Himanshu Gupta* en 1997 es de los más utilizados (Himanshu Gupta 2005), (Himanshu Gupta 1997), (Gupta 1997), (Gupta 1999). Utiliza grafos acíclicos para representar las vistas y las consultas, siendo que para cada nodo existe una operación de selección, agregación, proyección o unión. A cada nodo, esta asociado un número de tuplas determinado y un número de frecuencia de la consulta (número de veces que el nodo fue accedido).

Este algoritmo en cada iteración selecciona la vista con mayor beneficio por unidad de espacio, la noción del beneficio de una vista, es en relación al conjunto de vistas materializadas, y esta dado por la siguiente ecuación:

$$\tau(G, M) = \sum_{i=1 \wedge vi \in G}^{k} f_{vi} q(vi, M) \qquad (4)$$

Este cálculo del beneficio consiste en la diferencia de los costos de procesamiento de las vistas materializadas con o sin la vista.

El costo total del conjunto de vistas materializadas esta dado por:

$$B(v, M) = \tau(G, M) - \tau(G, M \cup \{v\})\,. \qquad (5)$$

El beneficio por unidad de espacio es calculado a través de la división entre el beneficio presentado por la vista y su espacio:

$$BPUS(v, M) = \frac{B(v, M)}{S(v)} \qquad (6)$$

En el algoritmo BUPS el tiempo de procesamiento es *O (kn2),* donde *n* es el numero de nodos en el grafo y *k* es el numero de iteraciones dado por este algoritmo.

Datos: G es un grafo de vista ANDOR, S una restricción de espacio

INICIO
M = {};

En cuanto (S (M) <S)
Sea V la vista que tiene un máximo beneficio
por unidad de espacio en relación a M.
M = M U {V};
Fin del En cuanto

Devuelve M

FIN

Algoritmo 1 – El algoritmo BPUS

El algoritmo A* presentado por *Himanshu Gupta.y M. Inderpal Singh* y extendido por (Gang Gou 2004), para la selección sobre una restricción de tiempo de manutención, sobre una restricción de espacio y utiliza igualmente los grafos acíclicos orientados de tal forma que representa el problema.
Este algoritmo se especializa en la representación de un cuboide, que no es mas que un grafo acíclico orientado al modelo multidimencional, con un nodo como raíz y otro como hoja. El nodo raíz es la tabla de hechos y el nodo hoja consiste en la agregación universal de los datos a la tabla de hechos, o sea, el total sobre todas las dimensiones.

El algoritmo A* expande un árbol binario de búsqueda *TG*. Cada nodo en *TG* es del tipo (*Nx*, *Mx*), denotado por *x* = (*Nx*, *Mx*), donde *Nx* es el conjunto de vistas visitadas y *Mx* es el conjunto de vistas seleccionadas para la materialización $(Mx \subseteq Nx)$

Si se sigue la orden de inserción predeterminada (v1, v2,..., vn), cada vista *vi* es considerada para la materialización. El descendiente izquierdo de x es definido por $l(x) = (N_x \cup \{v_{i+1}\}, M_x)$ que significa que la recién insertada vista *vi*+1 no será materializada y el descendiente derecho *x* definido por $r(x) = (N_x \cup \{v_{i+1}\}, M_x \cup \{v_{i+1}\})$ significa que la recién insertada vista *vi*+1 será materializada.

Finalmente, el nivel mas bajo del árbol *Nx = V,* significa que todas las vistas del cuboide *G* fueron consideradas o no para la materialización (*V* denota el conjunto de vistas o nodos en *G*).

El beneficio de expandir el nodo *x* es la suma de dos funciones: *g(x)* y *h(x)*. La primera calcula el beneficio adquirido y la segunda el beneficio estimado al materializar *x* (Gang Gou 2004).
El proceso de construcción del árbol binario de búsqueda A* cobra todo el espacio de 2n soluciones posibles. Esto asegura que el algoritmo contempla la solución óptima. La complejidad del algoritmo A* se sitúa entre *O(n)* y *O (2n)* dependiendo de la cantidad de la función estimadora *h(x)*. O sea, en el peor de los casos tiene un comportamiento exponencial, mucho peor que el BPUS.

Datos: Un cuboide G y una restricción de espacio S.

INICIO

Crear una árbol de pesquisa A* inicial TG,
teniendo como nodo de raíz ({}, {});
Determinar una orden de inserción de vistas:
(v1, v2,..., vn);
Crear una lista de prioridad L = {raíz};
Repite
Retirar un nodo x = (Nx, Mx) de L, donde x
tiene el menor g(x) + h(x) valor en L;
i = |Nx|;
Si (i = n) **Entonces**
Devuelve Mx;
Fin del Si
Insertar l(x) = (Nx U {vi+1}, Mx) en L;
Si (U (Mx U {vi+1}) <= S) **Entonces**
Insertar r(x) = (Nx U {vi+1}, Mx U {vi+1})
en L;
Fin del Si

Escribir (L estar vacía);

Devuelve {};

FIN

Escribir (L estar vacía);

Devuelve {};

FIN

Algoritmo 2 – El algoritmo A*

2.1.2.3 Análisis y comparación entre los algoritmos A* y BPUS:

En un Data Warehouse los datos son organizados según un modelo multidimensional, de forma tal, que mejora el desempeño de procesamiento de los datos y su visualización (Gang Gou 2004). En este ejemplo, se tiene un esquema con tres dimensiones y una tabla de hechos con cuatro atributos: VENTAS (Tiempo, Producto, Hoja, Precio). Los atributos Tiempo (T), Producto (P) y Hoja (L) forman el espacio dimensional. El último atributo, Precio, es la medida sobre la cual se pretenden analizar los datos. Los análisis realizados en OLAP consisten en agregaciones de algunas dimensiones. Las funciones de agregación usualmente disponibles en la base de datos, juntamente con el operador GROUP-BY, son: la suma; promedio; máximo y mínimo. Las ventas de producto por hoja se traducen en la siguiente consulta SQL:

select Producto, Hoja, SUM (Precio)
from VENTAS ***group by*** Producto, Hoja.
En esta consulta recorre al tipo de agregación producto por hoja (PL).

En el esquema multidimensional presentado se pueden efectuar 23 tipos de agregaciones diferentes (PLT, PL, PT, LT, P, L, T).
Generalmente se recurre al cuboide para representar las relaciones entre las posibles agregaciones a efectuar sobre un esquema multidimensional dado. Por ejemplo, a partir de la agregación producto por hoja (PL) aun es posible efectuar dos agregaciones (por producto, P, y por hoja, L). De estas dos, se puede llegar a la agregación universal (ALL), o agregación de las medidas en un único total.

En la figura 6 se ilustra el cuboide del ejemplo utilizado, donde TF simboliza la tabla de hechos. Para cada nodo del cuboide existen dos parámetros, T y F. El primero indica el

número de tuplas que resultan de la agregación especificada. El segundo indica la frecuencia de interrogaciones que utilizan la agregación en causa. Cualquier consulta puede ser respondida a partir de la tabla de hecho, motivo por el cual, esta no tiene una frecuencia de consulta asociada.

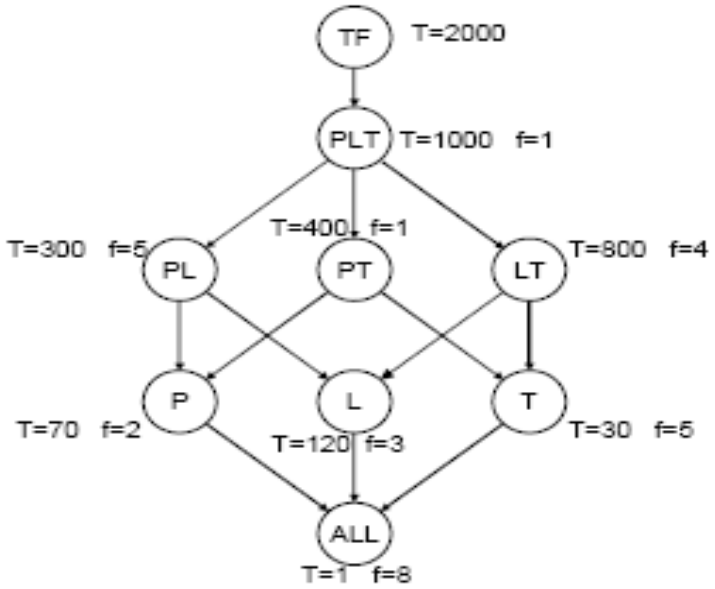

Figura 6 Cuboide

Ambos algoritmos utilizados fueron ejecutados en un ciclo donde varía la restricción del espacio. El espacio total para materializar todas las vistas de un cuboide es 2721 que no es más que la sumatoria del número de tuplas de todas las agregaciones.

La relación del costo total de procesamiento de las soluciones obtenidas por los algoritmos y la evolución de la restricción del espacio se ven en la figura 7.
Cuando el espacio de restricción es nulo, o sea, que el conjunto de vistas a materializar esta vacío, los costos de procesamiento son de 58000. Ya cuando se llega a una restricción de espacio de 2721, el costo de procesamiento se minimiza.

Figura 7 Costo contra restricción (primera prueba)

Como el algoritmo A* prefiere las vistas con mayor frecuencia de la consulta y tamaño, entonces una anomalía en este algoritmo, confirma una degradación en el costo de procesamiento cuando la restricción de espacio alcanza el valor 301. Este algoritmo selecciona el conjunto M= {ALL, PL}, con un costo de procesamiento de 25008. Es entonces necesario implementar una función estimadora *h(x).*

Si se observa la figura 7 se podría afirmar que el algoritmo BPUS tiene mejor comportamiento que el A*, para este ejemplo.

La figura 8, evidencia que el BPUS no respeta la restricción de espacio impuesta. Es cierto que el algoritmo implementado materializa siempre una vista extra, y como tal ultrapasa el límite de espacio impuesto por la restricción. Esto se refleja en el comportamiento del algoritmo, que acaba por presentar mejor desempeño que el concurrente, una vez que toma el espacio que en realidad no posee.

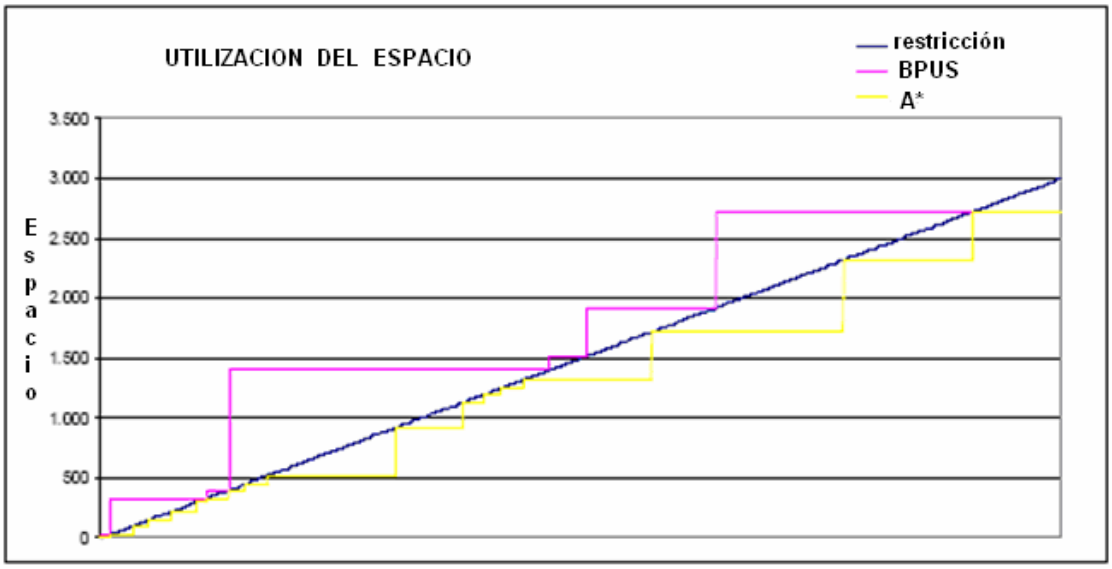

Figura 8 Utilización del espacio (primera prueba)

Si se procediera a algunos ajustes en el funcionamiento del algoritmo BPUS, de forma que se conservara integra la restricción de espacio. De esta forma, muestra que después de nuevas pruebas se verifica que el BPUS alterado respeta la restricción de espacio. (Figura 10).

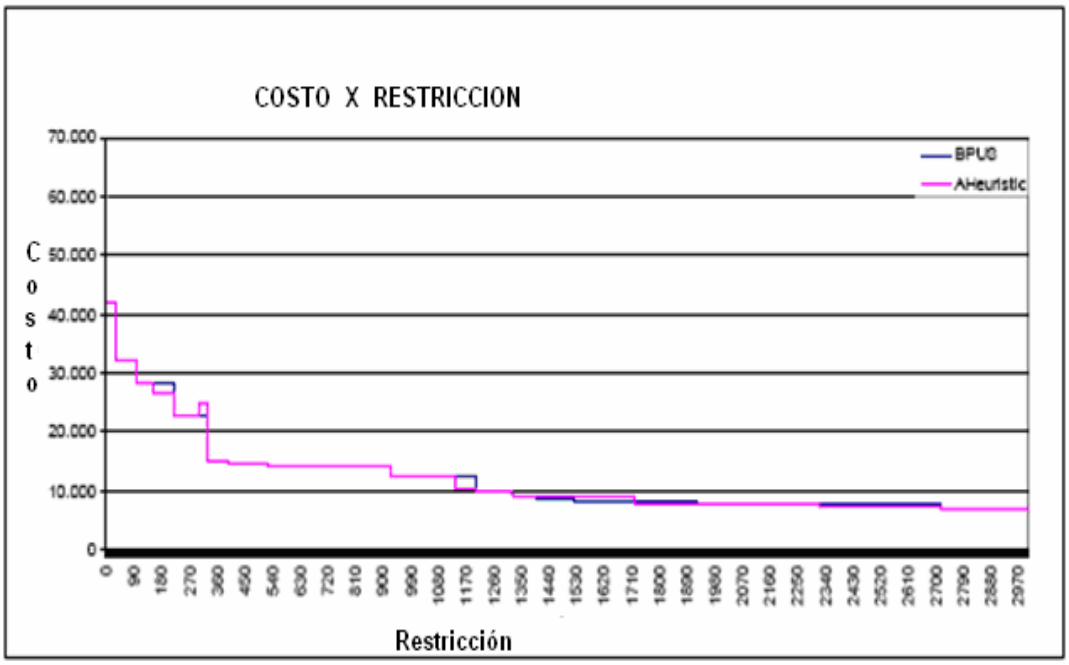

Figura 9 Costo contra restricción (segunda prueba).

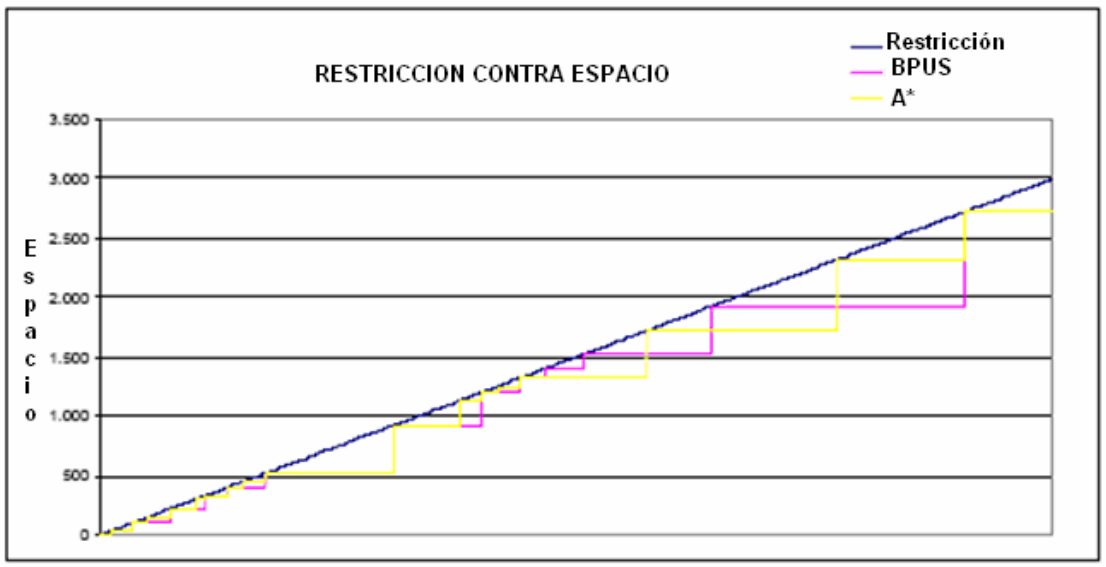

Figura 10 Restricción contra espacio (segunda prueba).

Los resultados de la segunda prueba, muestran un comportamiento semejante de los dos algoritmos para el ejemplo utilizado en la figura 9. Esto muestra que en la primera prueba, el BPUS toma claramente partido de la vista extra que podía materializar, para obtener beneficios en términos de costo de procesamiento.

La figura 11 muestra un análisis en términos de tiempo gastado por cada algoritmo en la búsqueda de la solución, haciendo igualmente variar la restricción de espacio, concluyendo que el algoritmo A* tiene mejor comportamiento, cuando lo se compara con el algoritmo BPUS.

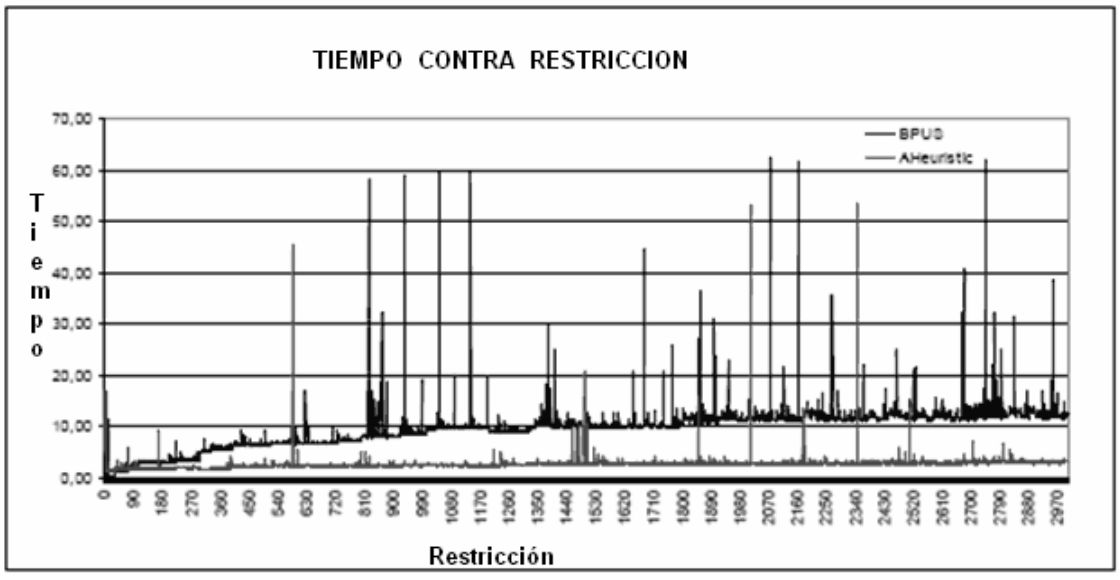

Figura 11 Tiempo contra restricción

Entonces se puede concluir que los resultados obtenidos muestran que el BPUS tiene un mejor comportamiento que el A* debido a la forma como fue concebido. Su concepción lógica, le permite implementar una vista extra, aunque no respeta la restricción de espacio. Como todo, después de modificado de forma tal que restringe el parámetro de espacio disponible, se verifica su comportamiento, en términos de costo de procesamiento, y es semejante al del otro algoritmo.

A su vez se muestra que el algoritmo A* puede ser mas rápido al resolver los mismos problemas. Como todo tiene una discrepancia en la evolución del costo de procesamiento con la restricción de espacio. Esto se debe al hecho de no tener una función estimadora *h(x)* apropiada.

La segunda idea es que en un último análisis ambos algoritmos tienen un comportamiento parecido, el que probablemente tendrá que ver con su tipo, ambos determinísticos.

2.1.3 Algoritmo *Counting* y algoritmo DRed:

Estos algoritmos describen la manutención incremental de las vistas materializadas. Ambos se aplican en las transacciones donde se refleja las alteraciones en la base de datos (inserciones, eliminaciones, y actualizaciones); por lo que las vistas pueden utilizar uniones, negaciones, agregaciones. Estos algoritmos generan reglas que permiten implementar alteraciones en las vistas, es decir, a partir de las alteraciones ocurridas en las relaciones base y ante las modificaciones de las vistas materializadas.

El trabajo de manutención de las vistas depende fuertemente de la cantidad de información manoseada (CRISTINA 2006). Innúmeros de algoritmos se ejecutan con objetivo de la manutención de las vistas, dependiendo de la cantidad de información y del tipo de la vista, donde todas las relaciones base y vistas materializadas están libres para la manutención. Lo cual lleva en consideración, características en el proceso de formación de las vistas. En este acápite se observará una breve descripción de algunos algoritmos, organizados por los tipos de vistas.

1. Vistas no recursivas: En este caso, las técnicas más apropiadas incluyen el algoritmo de counting y demás algoritmos que también se utilizan de contadores en la estructura principal. Se puede citar mecanismos que poseen punteros, los cuales parten de la tupla de origen para las tuplas derivadas y también estructuras que definen reglas de producción para mantener las vistas SQL definidas con el operador distinct,

agregaciones y vistas donde los atributos de las mismas define el funcionamiento de la relación base, la cual está siendo actualizada.

2. Vistas con outer-join: En esta clasificación se tiene el algoritmo, el cual redefine toda la vista, obteniendo dos únicas instrucciones (una instrucción con left-outer-join y otra con right-outer-join) para ejecutar las alteraciones futuramente implementadas.

3. Vistas recursivas: Se puede citar el algoritmo DRed (Deletion and Rederivation); el algoritmo PF (Propagation/Filtration), similar al DRed; el algoritmo Kuchenhoff, el cual genera reglas para ejecutar un cálculo de la diferencia entre estados consecutivos de la base de datos, y el algoritmo Urpi-Olive, que ejecuta reglas de transición reflejando las modificaciones en cada relación derivada de cada alteración en la relación base.

2.1.3.1 Descripción del algoritmo *Counting*:

Este algoritmo es utilizado en vistas que no poseen líneas duplicadas. La principal idea de este algoritmo es mantener un contador central del número de derivaciones para cada línea de registro de la vista.
En el ejemplo 1 se comprenderá mejor el algoritmo:

Ejemplo 1:
Dada la relación link = {(a, b), (b, c), (b, e), (a, d), (d, c)} y, como resultado, la vista hop = {(a, c), (a, e)}. La tupla hop (a, e) tiene apenas una sola ocurrencia, pues es derivada una vez de link, mientras que la tupla hop (a, c) posee dos derivaciones.
Si la vista no fue definida con el operador ***distinct***, o sea, que posea duplicidad, entonces se tendrá a ***count*** igual a 1 para hop (a, e) y ***count*** igual a 2 para hop (a, c). Como se muestra el algoritmo de counting describe la duplicidad en la vista y almacena la misma en contadores.
Suponiendo que la tupla link (a, b) sea excluida. La vista hop seria ejecutada nuevamente, y se tendría como resultado hop = {(a, c)}. De acuerdo con el algoritmo de counting una derivación de cada una de las tuplas hop (a, c) y hop (a, e) necesita ser excluida. El algoritmo se basa en los contadores que, la tupla hop (a, c) aun posee una derivación, por tanto hop (a, e) debe ser eliminada, por no poseer más derivaciones.

2.1.3.2 Descripción del algoritmo *DRed*:

El algoritmo DRed (*Deletion and Rederivation*), descrito en (CRISTINA 2006), no puede ser usado en vistas que no posean el operador ***distinct,*** es decir que las vistas no pueden obtener duplicidad de tuplas. Este algoritmo hace un cálculo en las alteraciones de las vistas, primero distingue las tuplas derivadas, las cuales serán eliminadas después de una estimación, es decir, que si una tupla *t* se encuentra en la estimación cualquier alteración que se le realice en la relación base entonces se anula la ocurrencia de *t*. Luego se remueven las tuplas que poseen derivación. Finalmente las nuevas tuplas que serán almacenadas, son calculadas a partir de las nuevas inserciones ejecutadas en la relación base y en la construcción de una vista materializada actualizada.
En el ejemplo 2 se comprenderá mejor este algoritmo:

Ejemplo 2:
Dada la relación link = {(a, b), (b, c), (b, e), (a, d), (d, c)} y, como resultado, la vista hop = {(a, c), (a, e)}. Suponga que se eliminara la tupla link (a, b). El algoritmo suprimirá las tuplas hop (a, c) y hop (a, e), pues ambas dependen de la tupla link (a, b), la cual fue eliminada. Después, el algoritmo ejecutara una búsqueda por derivaciones alternativas de cada una de las tuplas eliminadas. Luego se tendrá una nueva derivación de la tupla hop (a, c) y la inserción de la misma en la vista materializada. La tercera etapa del algoritmo es vacía para este ejemplo, ya que no se tiene tuplas a ser insertadas en la relación link.

2.1.3.3 Análisis del algoritmo counting:

Se observa a partir de los algoritmos presentados la relevancia de los mismos si se comparara la solución actual con los ejemplos citados, se tendría como relación link: el sistema SEER y las pruebas y disertaciones encontradas en VIRTUA. Y como vista hop: los artículos exportados de SEER y de DSPACE (repositorio de pruebas y disertaciones) (CRISTINA 2006).

El algoritmo de counting es el mas usado, debido a que posee más flexibilidad, ya que puede ser utilizado en vistas que poseen o no líneas duplicadas; además de utilizar vistas no recursivas.
Como este algoritmo permite la ejecución de líneas duplicadas, entonces se muestra que presenta duplicidad de tuplas generadas por el mismo. Esto puede ser notado a partir del contenido de las variables contadoras.

Abajo se muestra el pedazo de código responsable por la creación de la vista y la cuenta regresiva de cada tupla derivada y duplicada de la vista.

// **Creación de la vista SEER**

```
$query="CREATE VIEW seer AS SELECT
nArticleID,chMetaTitle,chMetaSubject,chMetaAbstract,dtDatePublished,fkIssueID FROM
tblarticles";
$result=mysql_query($query,$link);
$query2="SELECT
nArticleID,chMetaTitle,chMetaSubject,chMetaAbstract,dtDatePublished,fkIssueID
FROM tblarticles";
$result2=mysql_query($query2,$link);
if(mysql_num_rows($result)>0)
{
$aux=0;
while($campos=mysql_fetch_array($result))
{
```

// **Creación de la variable counting**

```
$counting[$aux]=0;
while($campos2=mysql_fetch_array($result2))
{
if($campos['chMetaTitle']==$campos2['chMetaTitle'])
$counting[$aux]++;
}
$aux++;
```

La variable $counting[$aux] almacena para cada registro verificado la cantidad de veces que el mismo fue duplicado. Para lo cual se verifica las variables $campos['chMetaTitle'] y $campos2['chMetaTitle'], que representan los títulos de los trabajos contenidos en los sistemas; pues con la función while se busca cada título de trabajo y se ejecuta una barredura por todos los otros títulos, en busca de algún otro título igual. Una vez ejecutada esta pesquisa la variable $counting[$aux] almacena la cantidad de derivaciones iguales para cada título de trabajo. A partir del contenido de la variable $counting[$aux] se sabe la duplicidad de cada registro de la vista generada.

Con la ejecución del algoritmo una derivación de cada una de las tuplas tiene que ser eliminada. Esto se observa en el código de abajo, donde se excluye una derivación duplicada de cada registro generado de acuerdo con el contenido de los contadores.

// **Eliminación de tuplas derivadas**

```
for($i=0;$campos=mysql_fetch_array($result);$i++)
{
if($counting[$i]>0)
{
$query="DELETE FROM seer WHERE chMetaTitle=".$campos['chMetaTitle']."AND
nArticleID=".$campos['nArticleID'];
$result2=mysql_query($query,$link);
}
}
```

Si la variable $counting[$i] es mayor que cero, entonces se sabe que el registro identificado por el índice $i tiene mas de una derivación, por lo que será necesario excluir una duplicidad de ese registro.
Es importante resaltar que se debe eliminar todas las derivaciones de cada tupla antes de finalizar el algoritmo, ya que no se puede tener registros duplicados en el resultado final de la vista.
El algoritmo debe ser ejecutado todas las veces que la relación base sea alterada, de esa forma la vista generada estará siempre en conformidad con los datos oriundos de la base de datos.

Este algoritmo fue ejecutado para la creación de la vista materializada de las pruebas y disertaciones del VIRTUA, formando el DSPACE. De esta forma se puede concluir, que con el uso del algoritmo de counting las vistas son generadas sin replicación de datos, construyendo un modelo formal de integración de sistemas de información. Por lo que se puede observar el modelo de integración propuesto como alternativa para los sistemas siempre que permanezca ínter-ligado, a través de la ejecución continua del algoritmo, garantizando de ese modo la fidelidad precisa de los datos generados por las vistas en conformidad con la relación base.

En la figura 12 se puede observar el nuevo modelo propuesto:

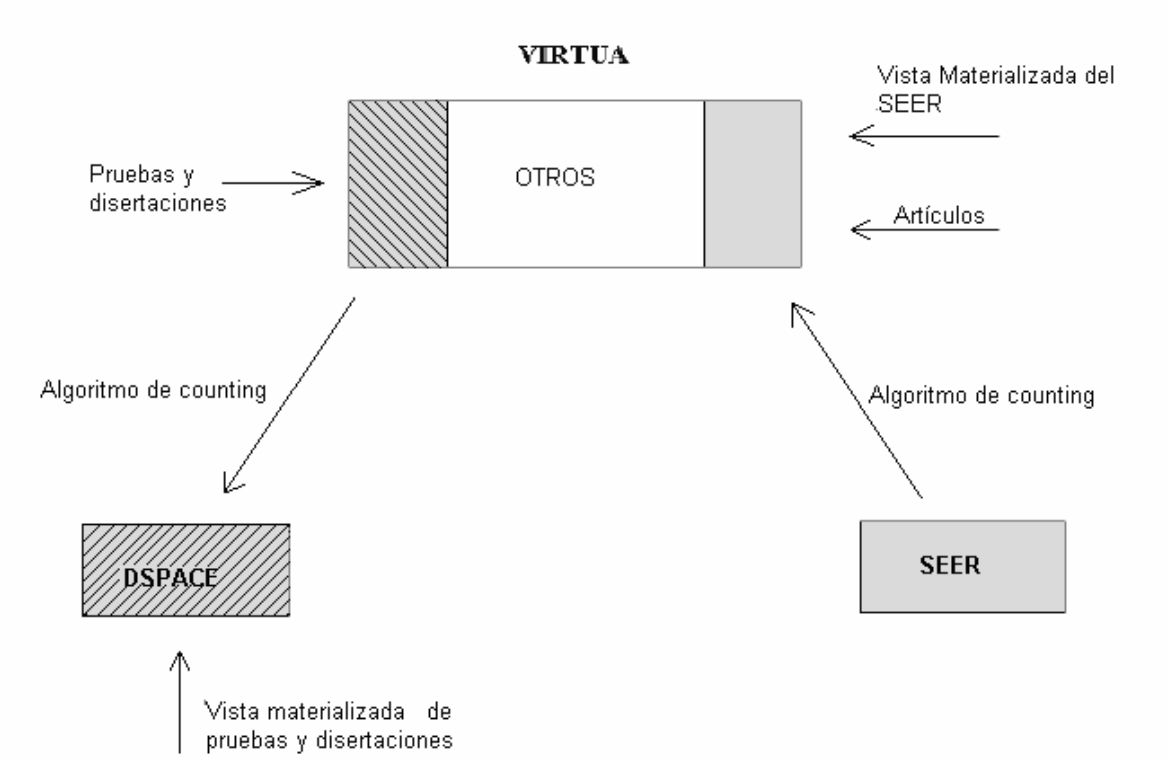

Figura 12 Modelo del VIRTUA.

El SEER ejecuta el algoritmo de counting, creando una vista materializada del SEER en VIRTUA, luego el VIRTUA ejecuta el algoritmo, construyendo una vista materializada de las pruebas y disertaciones encontradas en VIRTUA para el DSPACE. De esta forma se estandariza todo el envío de datos a través de la ejecución del algoritmo, garantizando la fidelidad de las informaciones extraídas de las relaciones base, o sea, las vistas generadas siempre estarán en conformidad con las informaciones almacenadas en la base de datos original. Por lo cual se puede afirmar que el algoritmo proporciona la exactitud de los datos replicados.

La falta de duplicidad de las informaciones generadas por las vistas materializadas, facilitan el acceso directo a los datos requeridos, la constante garantía de fidelidad y la integridad de las informaciones en relación a los datos base, permiten de esa manera al usuario tener la certeza de que la información que es accedida está siempre actualizada de acuerdo con la base de datos original; el uso de vistas materializadas como una copia de seguridad, de forma que los datos están siempre duplicados garantizan la

continua existencia de las informaciones almacenadas en la relación base y en las vistas materializadas; la utilización de un algoritmo consolidando vistas materializadas como prueba matemática en la modelación formal de un proyecto de integración de sistemas de información heterogéneos, posibilita la construcción de un estándar de integración de diferentes sistemas de información, a través del uso de vistas materializadas.

2.1.4 Lattice framework.

El lattice framework (V. Harinarayan 1996), es una herramienta en el análisis de la selección de vistas materializadas debido a que está adaptado a las vistas con dependencia. Cada cima del gráfico del lattice representa una vista que agrega medidas numéricas sobre las dimensiones presentes en esa cima, mientras que cada borde del gráfico representa las dependencias entre vistas.

Para las consultas Q1 y Q2, Q1 es el subconjunto propio de Q2, si solo si la consulta Q1 puede ser contestada usando el resultado de la consulta Q2, es decir que son dependientes. Por ejemplo, en el ejemplo 1, la consulta del PS puede también ser contestada usando la vista PSC. Es decir, PS es el subconjunto propio de PS y PS es el subconjunto propio de PSC.
En un cubo de los datos (CD) (Y. Liu 2006), (J. Gray 1997), una dependencia expresada en un lattice framework debe satisfacer las condiciones siguientes:
Primero, el subconjunto está en orden parcial. En segundo lugar, debe ser un elemento tope o base de otras vistas dependientes en un lattice (D. Theodoratos 2000).

Ejemplo 1: Se asume que se tienen transacciones de ventas en un sistema Data Warehouse. Un cubo de datos consiste en tres dimensiones importantes, piezas (p), surtidores (s), y clientes (c); y representa la venta de piezas de los surtidores a los clientes. Cada célula de un cubo de datos contiene la venta total de una pieza proveída por un surtidor a un cliente. La relación de piezas, de surtidores, y de clientes se ilustra en la figura 13.

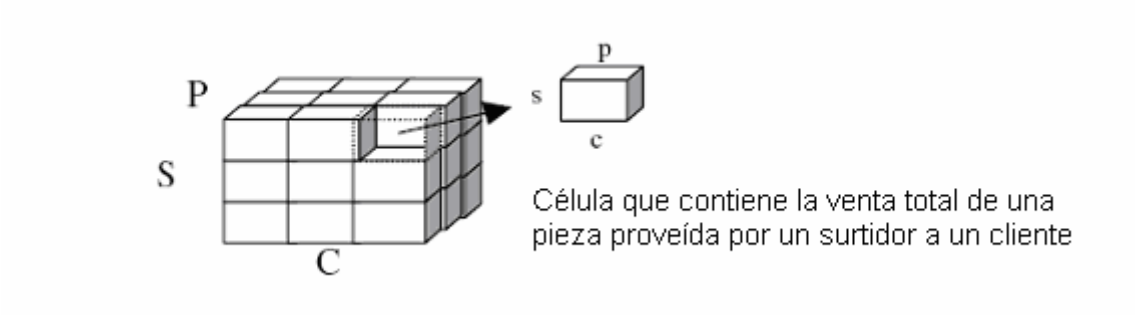

Figura 13 Relación entre pieza, surtidor y cliente.

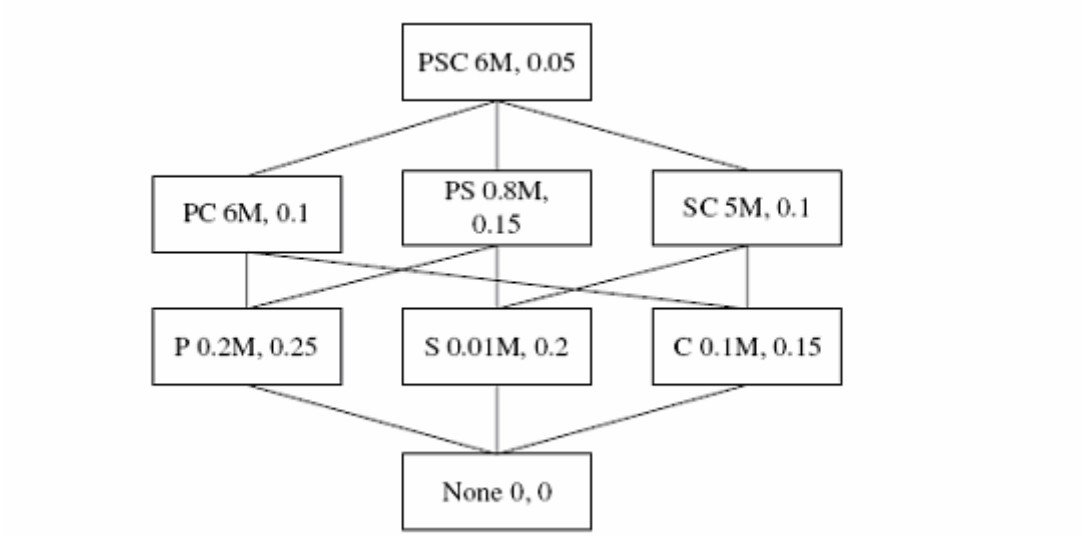

Figura 14 Lattice framework del cubo de datos

El lattice framework de un cubo de datos en el ejemplo 1 se ilustra en la figura 14. Las dependencias de vistas se expresan con las líneas de conexión. Por ejemplo, la línea de conexión de PSC al PS significa la dependencia que PS es subconjunto propio de PSC. Además PSC es una vista tope, de la cual otras vistas dependen. Los números al lado de una vista indican la cantidad de los datos y la frecuencia de las consultas. Por ejemplo, se muestra una cantidad de datos resultantes de 6M y una frecuencia de consultas de 0.05 en la consulta de la dimensión de PSC.

2.1.4.1 El vector de estructura de datos

El lattice framework se utiliza para expresar un cubo de datos. La representación de datos del lattice se expresa en vectores. Por ejemplo, un lattice consiste en tres dimensiones que tiene ocho vistas posibles en bit binarios: 111 para PSC, 101 para la PC, y así sucesivamente. Esto se demuestra en la figura 15.

Aquí se puede alcanzar los niveles superiores de vistas dependientes mientras que un bit 0 se gira en 1 y viceversa para alcanzar los niveles inferiores. Por ejemplo, las vistas tope de P (1 00) son el PS (110) y PC (101). Usando esta representación, los costos del proceso de la consulta y el mantenimiento de vistas dependientes pueden ser obtenidos fácilmente remontando sus jerarquías y bits correspondientes de 1 y 0.

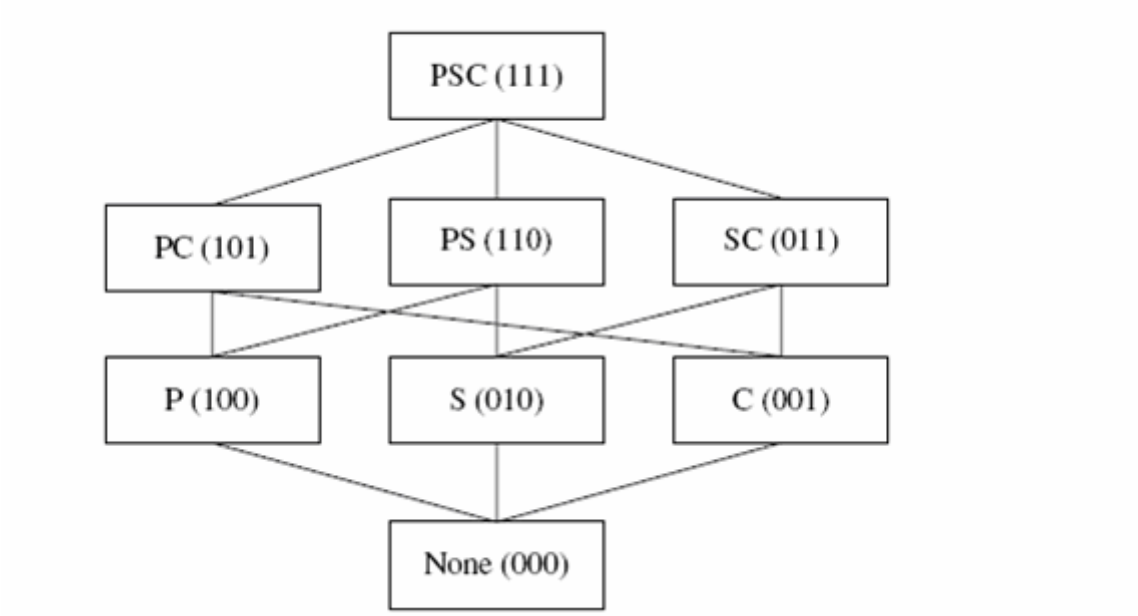

Figura 15 Representación del lattice para un cubo de datos.

2.1.4.2 Aplicación del vector de estructura de datos

La materialización de vistas tiene que ser capaz de responder cada consulta, utilizan el vector de estructura de datos descrito en la sección pasada para encontrar vistas materializadas dependientes. Por ejemplo, las vistas materializadas que se obtienen de la fig. 15 son el PS (110), PC (10 1), y C (001). Si se asume que una vista de la consulta es S (010), sus vistas dependientes son PS (11 0), SC (01 1), y PSC (111). En este caso, se puede utilizar la vista materializada PS (1 10) para contestar S (010). Para mejorar la eficacia de una consulta, se selecciona vistas de una capacidad más pequeña para procesar si el número de vistas materializadas dependientes es mayor que uno.

2.1.4.3 Reescritura de la consulta

¿Cómo puede una vista de la consulta determinar su vista dependiente inmediatamente y tomar buena ventaja de la materialización de vistas? La materialización parcial de los cubos de los datos (vistas) en un Data Warehouse realzará el funcionamiento de la consulta permitiendo la reescritura de la consulta basada en vistas de la consulta y en vistas materializadas.

La estructura del vector que se propone se puede aplicar a la reescritura de la consulta. Se busca el sistema de vistas materializadas usando la estructura del vector. Por medio de la estructura del vector, una consulta de usuario se puede correlacionar fácilmente a un subconjunto de las vistas materializadas fijadas, y computadas con el operador “and".

Cuando la vista de una consulta de usuario es la solución, la vista materializada es la vista dependiente de la consulta de usuario. Si hay más de una vista dependiente, se tiene que tomar tamaño de la vista en consideración. Por ejemplo, el PS (110), la PC (101), y C (00 1) son un sistema de vistas materializadas, mientras que P (100), S (0 10), la PC (101), el PS (110), y C (001) son consultas de usuario. Se puede determinar la vista dependiente de un sistema de vistas materializadas de la vista de la consulta S (010) de esta manera:

Aquí se utiliza el $\oplus$ operador ¨and¨ a nivel de bit.

$$\mathrm{S}(010) \oplus \mathrm{PS}(110) = (010)$$
$$\mathrm{S}(010) \oplus \mathrm{PC}(101) = (000)$$
$$\mathrm{S}(010) \oplus \mathrm{C}(001) = (000)$$

2.1.4.4 El modelo propuesto de costo

Los costos del proceso de la consulta y del mantenimiento son diferentes cuando varias vistas se utilizan en consultas. La selección de una vista puede influenciar varios resultados de la consulta en un cubo de datos. Por lo tanto, se necesita seleccionar las vistas óptimas para la materialización. En general, se generan vistas con una contribución mejor para reducir tiempo de la consulta, y se eliminan vistas de menos contribución para ahorrar el espacio de almacenaje. Sin embargo, el costo total y el

ahorro del espacio deben ser analizados más a fondo cuando dos o más vistas tienen las mismas contribuciones en la selección de las vistas.

2.1.4.4.1 Fórmulas del costo

Refiriéndose (J. Yang 1997), se computa el costo total usando un cociente de peso de la frecuencia del mantenimiento a la frecuencia de la consulta (w = frecuencia de mantenimiento/ frecuencia de la consulta) dado un periodo de tiempo. El peso w excede valor de 1 si ocurre el mantenimiento más a menudo que la consulta. El peso w es menor que el valor de 1 si la frecuencia del mantenimiento es menor que la frecuencia de la consulta. El peso w es igual a un valor de 1 si ambas frecuencias son idénticas. El valor de w es generalmente menor que 1 porque la frecuencia de la consulta es a menudo mayor que la frecuencia de mantenimiento.

Se asume que el SC, C, y P son vistas de la consulta de los usuarios. Hay una variedad de vistas materializadas incluyendo PSC, SC y PS, PSC y PS, o todas las vistas. ¿Como la vista materializada es más robusta si C se cambia muy a menudo? La respuesta depende del espacio de almacenaje disponible y de los costos totales de la consulta y del mantenimiento. Aquí se propone reducir al mínimo el costo total debido a la disponibilidad limitada del almacenaje. Es decir, se intenta determinar las vistas materializadas deseables en las cuales la suma de costo de elaboración de la consulta y el costo de mantenimiento sea el mínimo. El algoritmo óptimo propuesto se ilustra con los símbolos siguientes en la tabla 7.

Tabla 7 Definición de los símbolos del modelo de costo propuesto.

Símbolos	Definición
I	Subconjunto de fuentes de datos
r	Subconjunto de vistas
S_i	Espacio de la vista i
fu_i	Frecuencia de actualización de la vista i
fq_i	Frecuencia de las consultas de la vista i
CQ	Costo de la consulta
CM	Costo de mantenimiento
w	Peso de mantenimiento y consultas
mv	Conjunto de vistas materializadas
$\mathrm{mv} \rightarrow \mathrm{r}_i$	La vista i afectada por la vista materializada
$\mathrm{I} \rightarrow \mathrm{mv}_j$	La vista materializada j afectada por la actualización de datos
S_s	Espacio actual del Data Warehouse
S_{mv}	Espacio de las vistas materializadas
n	Número de vistas de la consulta
m	Número de vistas materializadas
$\mathrm{C}_{(\mathrm{mv}\rightarrow \mathrm{r}_i)}$	Costo de consultar la vista de la consulta i afectada por la vista materializada
$\mathrm{C}_{(\mathrm{I}\rightarrow \mathrm{mv}_j)}$	Costo de la vista materializada j afectada por la actualización de datos

La suma de todos los subconjuntos de las vistas materializadas (Si) igual al espacio total de las vistas materializadas (Smv) según muestra la fórmula (1).

$$\mathrm{S}_{\mathrm{mv}} = \sum_{i \in \mathrm{mv}} \mathrm{S}_i \qquad (1)$$

El costo de la consulta de cada vista es el producto de la frecuencia de la consulta y del costo de la materialización de la vista o de su vista dependiente. Por lo tanto, el costo total de la consulta (CQ) de la fórmula (2) es la suma de costos de la vistas de la consulta.

$$\mathrm{CQ} = \sum_{i=1}^{n} f_{\mathrm{q}_i} \mathrm{C}_{(\mathrm{mv}\rightarrow \mathrm{r}_i)} \qquad (2)$$

.

El costo de mantenimiento de una vista materializada es el producto de la frecuencia del mantenimiento y del costo de mantenimiento afectado por la actualización de los datos de la vista. Por lo tanto, el costo de mantenimiento total (CM) es la suma de costos de

mantenimiento de todas las vistas materializadas según las indicaciones del fórmula (3).

$$\mathrm{CM} = \sum_{j=1}^{m} f_{\mathrm{u}_j} \mathrm{C}_{(\mathrm{l} \to \mathrm{mv}_j)} \qquad (3)$$

¿Porque la frecuencia de la consulta es diferente de la frecuencia del mantenimiento en un Data Warehouse?, el peso w desempeña un papel importante calculando el costo total. Según lo indicado antes, excede el valor de 1 cuando la frecuencia del mantenimiento es mayor que la frecuencia de la consulta. De otra manera, w es menor que 1. Si dos frecuencias son similares, w es igual a 1. Por lo tanto, el costo total se demuestra con el peso, en la fórmula (4).

$$\mathrm{Cost}(\mathrm{mv}) = \mathrm{CQ} + w^* \mathrm{CM} = \sum_{i=1}^{n} f_{\mathrm{q}_i} \mathrm{C}_{(\mathrm{mv} \to r_i)} + w \sum_{j=1}^{m} f_{\mathrm{u}_j} \mathrm{C}_{(\mathrm{l} \to \mathrm{mv}_j)} \qquad (4)$$

Se puede calcular el espacio total de vistas materializadas por medio de la fórmula (1) para decidir si la adición o la eliminación de vistas es posible. Es decir, se puede agregar vistas si más espacio está disponible para la materialización de las vistas; de otra forma, se suprimirán algunas vistas para aumentar el espacio real si las vistas materializadas exceden el espacio disponible.

La fórmula (4) se toma como indicador para determinar si la materialización de una vista es necesaria o no. Cuanto más bajo es el valor de un indicador, más pequeño es el costo total que él representa. Nuestro foco es bajar el costo total seleccionando un sistema óptimo de vistas materializadas. El método de adición o la eliminación se utiliza para seleccionar vistas materializadas. Para la adición, el costo total de materializar una nueva vista se calcula acorde se convierten en un candidato. Entonces, se selecciona el costo más bajo de los candidatos convirtiéndose en una vista materializada. En el proceso de eliminar una vista candidata, el costo total de materializar las vistas restantes se calcula para que cada vista candidata sea suprimida. Finalmente se selecciona al candidato que conduce al costo total más bajo.

En muchos casos, se puede seleccionar simplemente la vista con un gran espacio en el cual el costo total es el más bajo. Sin embargo, ninguna vista adicional se puede materializar para conseguir una solución mejor. Puesto que una solución óptima es un problema, se necesita más ayuda para mejorar nuestro resultado. A tal efecto, se propone la ganancia métrica y la pérdida métrica en las dos secciones siguientes que toman el espacio de la vista en consideración. Con la medida de la unidad de espacio, se alcanza un proceso mucho más rentable de vistas materializadas.

2.1.4.4.2 Ganancia métrica

La ganancia métrica es un indicador que permite la adición de una vista para la materialización. En el método de la adición para la selección de la vista, se estima el costo total de materialización para un nodo materializado (vistas) en cada lattice. Usando la ganancia métrica, se determina la rentabilidad de cada nodo y las vistas selectas con una ganancia métrica más alta como candidata para la materialización. La definición de ganancia métrica se demuestra como sigue:

Baja limitación del espacio real (Smv+ {v} ≤ Ss), si el costo total se reduce en respuesta al número creciente de las vistas materializadas (v), entonces se refiere al nivel del aumento como ganancia métrica. Aquí está su definición matemática:

{Cost (mv) – Cost (mv + {v})} / S (v) (5)

Es decir, para computar la diferencia entre el costo inicial y el costo total después de la adición de una vista, se divide la diferencia de costos por el espacio requerido para las vistas adicionales. Es decir, denota la ventaja por unidad de espacio de una vista a materializar. En el algoritmo propuesto descrito más adelante, se prefiere seleccionar las vistas materializadas que tienen un valor de ganancia métrica más alto para alcanzar un funcionamiento mejor en la reducción de costos. Las vistas de una ganancia métrica más alta que el valor no son seleccionadas si el espacio requerido excede el espacio real.

Ejemplo 2. Se asume que se tiene una transacción de las ventas en el Data Warehouse; el lattice framework del cubo de datos se demuestra en la fig 16. En este ejemplo, el cubo de datos consiste en cuatro dimensiones importantes: clientes (c), piezas (p),

surtidores (s), y tiempo (t). Hay nueve vistas {CPT, PST, CP, CS, PT, ST, C, P, y S} que serán consultadas. El requisito de espacio y la frecuencia de las consultas se demuestran en la tabla 8. Ahora, dado que se han materializado las cuatro vistas (es decir, mv = {CPT, PST, CS, y PT}), el espacio disponible permite que se aumente una vista para la materialización. En este caso particular, se selecciona una de dos vistas candidatas {C y P} para materializar. Primero, se calcula el costo total Cost(mv), Cost(mv + {C}) y Cost(mv + {P}) que usa la fórmula (4). Los resultados son 35.55, 34.8 y 31.2, respectivamente. En segundo lugar, se calcula la ganancia métrica para las vistas materializadas candidatas {C} y {P} usando la fórmula (5). El valor de la ganancia métrica es igual (35.55-34.8) /5 = 0.15 para la vista candidata {C}.

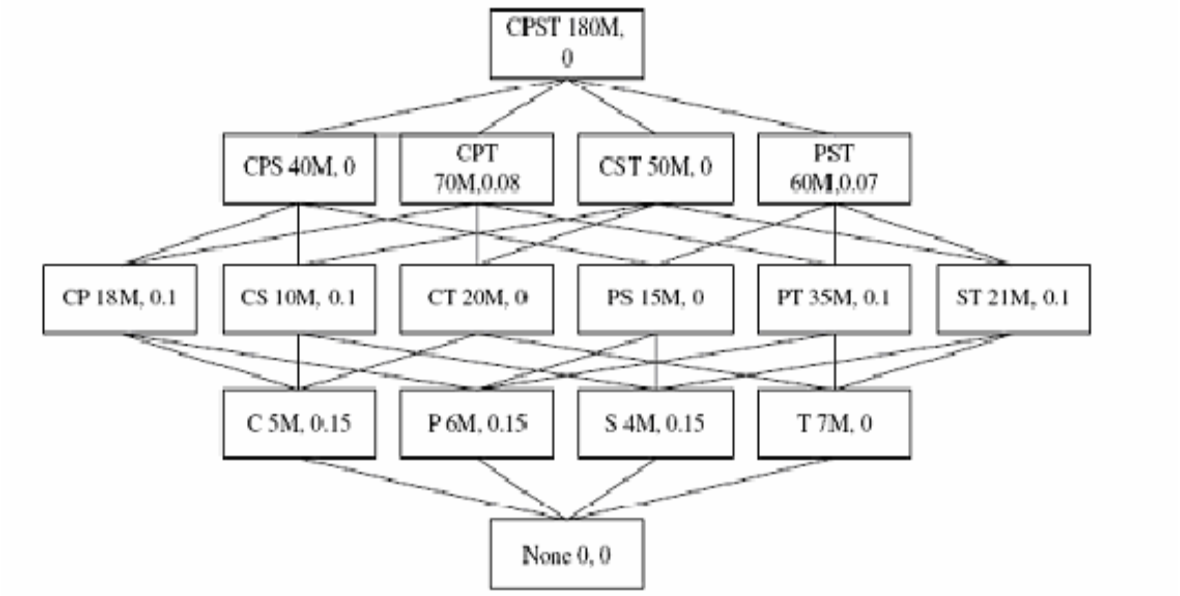

Figura 16 Lattice framework más complejo.

Tabla 8 Prueba de los datos para el experimento con w = 0.

Nombre de la vista	Cantidad de Datos	Frecuencia de la consulta
CPST	180M	0
CPS	40M	0
CPT	70M	0.08
CST	50M	0
PST	60M	0.07

CP	18M	0.1
CS	10M	0.1
CT	20M	0
PS	15M	0
PT	35M	0.1
ST	21M	0.1
C	5M	0.15
P	6M	0.15
S	4M	0.15
T	7M	0
Ninguno	0	0

La otra vista candidata {P} tiene una ganancia métrica de (35.55 - 31.2) /6 = 0.725. En este caso, se selecciona la vista candidata {P} para la materialización porque tiene una ganancia métrica mayor.

2.1.4.4.3 Pérdida métrica

La pérdida métrica es un indicador para quitar las vistas materializadas candidatas como método de la eliminación para la selección de vistas. En el algoritmo propuesto mas adelante, se estima el costo total de materialización, después de suprimir los nodos materializados (vistas) de un lattice. Usando el indicador de la perdida métrica, se determina la rentabilidad de cada nodo y se selecciona las vistas con el valor más bajo de perdida métrica como candidatas para la eliminación. La definición de una pérdida métrica se demuestra como sigue:

Baja limitación del espacio disponible (Smv+ {v} ≤ Ss), si el costo total se aumenta en respuesta al número disminuido de las vistas materializadas (v), se le llama al nivel de la pérdida como pérdida métrica. Aquí está su definición matemática:

{Cost (mv – {v}) – Cost (mv)} / S (v) (6)

Es decir, para calcular la diferencia del costo total después de la eliminación y del costo total inicial, se divide el resultado por el espacio de una vista eliminada del sistema de vistas materializadas. En el algoritmo propuesto, se prefiere vistas de un valor mínimo de pérdida métrica para la eliminación, debido a su costo total que es el menor. Sin embargo, las vistas de la pérdida métrica más baja no se seleccionan si el espacio requerido después de la eliminación todavía excede el espacio real.

Ejemplo 3. Como en el ejemplo anterior, se asume que todas las vistas de la consulta están materializadas, pero su espacio requerido total excede el espacio disponible. Se utiliza el método de la eliminación para quitar algunas de las vistas materializadas del sistema materializado para emparejar el espacio disponible y para realzar funcionamiento total. En este caso particular, se elimina una de las dos vistas candidatas {C, P} del sistema de vistas materializadas. Primero, se calcula el costo total Cost (mv), Cost (mv - {C}), y Cost (mv - {P}) usando la fórmula (4). Los resultados son 20.45, 21.2 y 22.25, respectivamente. En segundo lugar, se calcula la pérdida métrica para las vistas materializadas candidatas {C} y {P} usando la fórmula (6). El valor de la pérdida métrica es igual (21.2-20.45) /5 = 0.15 para la vista candidata {C}. La otra vista candidata {P} tiene un valor de pérdida métrica de (22.25-20.45) /6 = 0.3. Finalmente, se selecciona la vista candidata {C} con un valor de pérdida métrica menor para la eliminación. En este caso, se quita la vista candidata {C}.

2.1.4.5 Los algoritmos propuestos de la selección

Se propone el uso de dos algoritmos, el algoritmo que localiza el punto mediano (MPL) y el algoritmo que localiza el punto mediano con las vistas candidatas (MPL-CV) para seleccionar el sistema de vistas materializadas.

El algoritmo del MPL se basa en la posición del corte-punto de cualquier espacio real dado (Ss) que miente entre los valores del espacio máximo y del espacio mínimo. De acuerdo con la posición del corte-punto, uno puede decidir si suprimir o agregar vistas materializadas. Para mejorar el funcionamiento del algoritmo del MPL, se propone el algoritmo de MPL-CV. La vista materializada que excede el espacio disponible se toma en consideración en el algoritmo de MPL-CV. Usando este algoritmo realzado, se puede eliminar el problema de un juicio erróneo de un punto mediano. El propósito de los algoritmos propuestos es reducir al mínimo el costo total seleccionando un sistema

óptimo de vistas materializadas obligadas por el espacio real. Los cálculos del costo son métricas críticas en el modelo propuesto en el cuál el costo total considera los costos del proceso y de mantenimiento de la consulta. Las Fig. 17 y 18 constituyen una representación visual simplificada de los algoritmos del MPL y de MPL-CV.

2.1.4.5.1 Algoritmo que localiza el punto mediano (MPL)

Primero, se determina el espacio mínimo (Sss) y sus vistas (Mss) para la materialización. El espacio mínimo (Y.H. Chen 1997) representa el menor número de las vistas materializadas para contestar todas las consultas posibles. El espacio máximo (Sls) representa el espacio en el cual se materializan todas las vistas (Mls). Entonces, se define el espacio disponible (Sas), que es igual a Ss- Sss. Es decir, el espacio disponible es la diferencia entre el espacio real dado (Ss) y el espacio mínimo (Sss). Se define k como el valor que es igual al espacio disponible dividido por la diferencia entre el espacio máximo y el espacio mínimo. Se utiliza para indicar si el espacio real está cerca del espacio mínimo o del espacio máximo. Cuando k no es menor que 0.5 (es decir, el espacio real está más cercano al espacio máximo), todas las vistas de la consulta serán materializadas. Entonces, las vistas de la pérdida métrica más baja se quitan una por una hasta que el límite del espacio sea satisfecho. Inversamente, cuando k es igual o menor que 0.5, las vistas mínimas son materializadas. Entonces, las vistas de una ganancia métrica más alta son agregadas una a la vez hasta que no hay más espacio disponible.

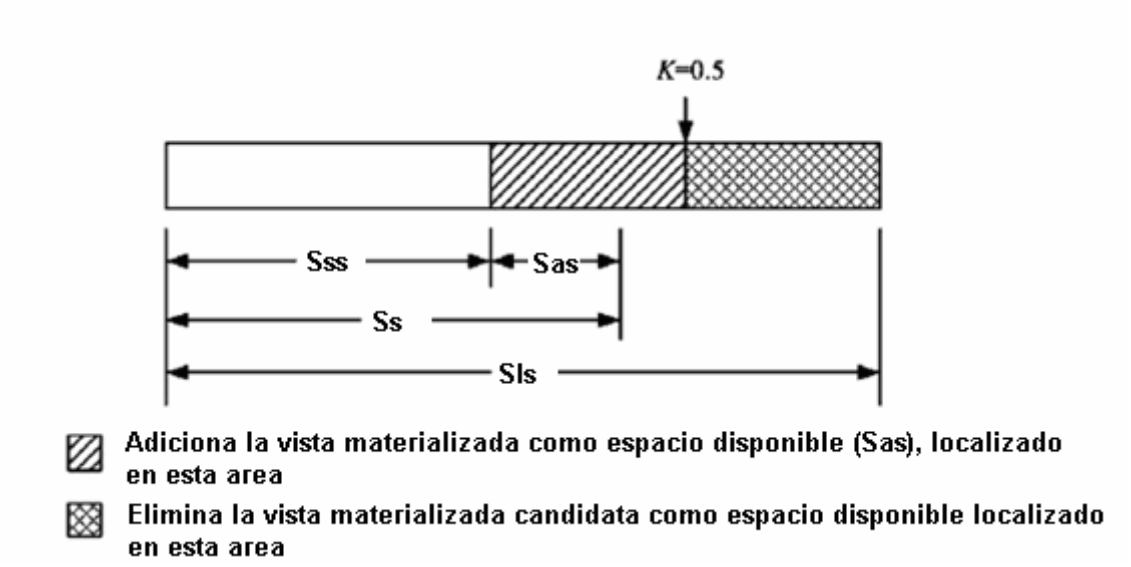

Figura 17 Representación visual del algoritmo MPL.

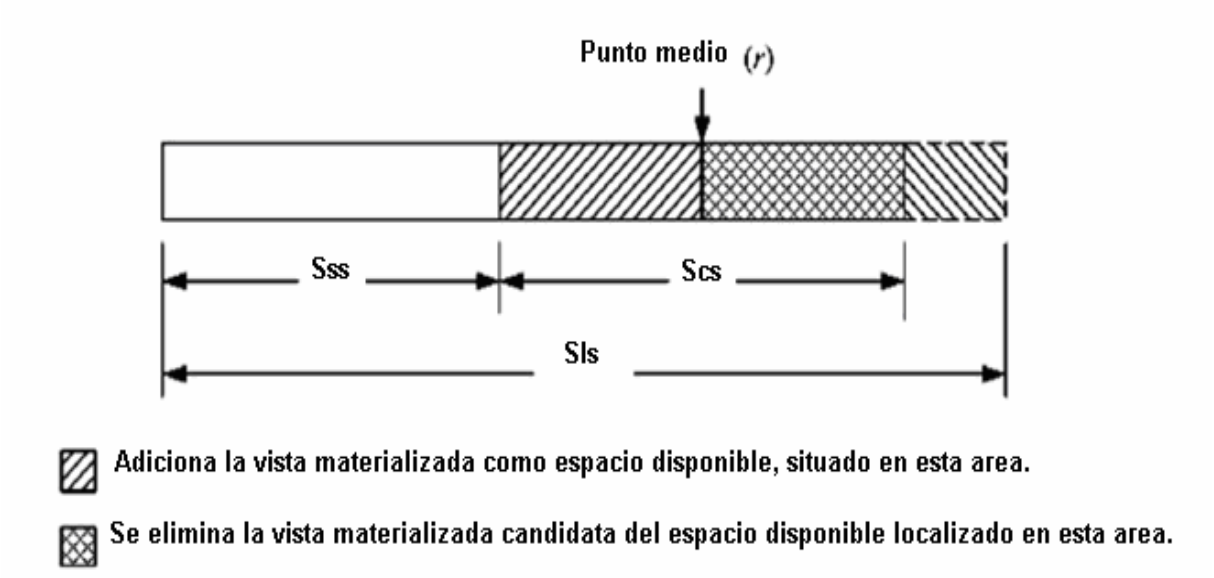

Figura 18 Representación visual del algoritmo MPL-CV.

El algoritmo del MPL incluye 10 pasos, descritos más abajo.

Algoritmo: Algoritmo MPL:

Paso 1: Determinar todas las vistas materializadas (Mls), y el espacio máximo (Sls).

Paso 2: Determinar las vistas materializadas mínimas (Mss), y el espacio mínimo (Sss).

Paso 3: Según el espacio real dado (Ss), se computa el espacio disponible (Sas = Ss - Sss).

Si Ss ≥ Sls entonces
mv = Mls /* Materializar todas */
Ir al paso 10
Fin del si
Si Ss < Sss entonces
mv = Ø /* El mínimo Ss puede ser > Sss */
Ir al paso 10
Fin del si

Paso 4: Determinar el valor de k (k = Sas/Sls _ Sss)

Paso 5: **Si k < 0.5 entonces**
mv = Mss

$$S_{mv} = \sum_{i \in mv} S_i$$

sino
mv = Mls

$$S_{mv} = \sum_{i \in mv} S_i$$

Ir al paso 8
Fin del si.

Paso 6: Utilizar el vector de estructura de datos para computar el Cost (mv) de los resultados del lattice basados en cada vista candidata, y determinar las vistas de mayor ganancia métrica (v).

Paso 7: **Si Ss > S mv+{v} entonces**
mv = mv + {v}
Ir al paso 6
Sino
Ir al paso 10
Fin del si.

Paso 8: Utilizar el vector de estructura de datos para computar el Cost (mv) de los resultados del lattice basados en cada vista candidata, y determinar las vistas de la pérdida métrica más baja (v).

Paso 9: **mv = mv – {v}**

Si Ss < Smv entonces
Ir al paso 8
Fin del si.

Paso 10: Resultados del sistema de vistas materializadas (mv).

2.1.4.5.2 Algoritmo que localiza el punto mediano con vistas candidatas.

Primero, se determina el espacio mínimo (Sss), el espacio máximo (Sls) y las vistas (Mss, Mls) para materializar en el algoritmo del MPL. También, se define el espacio disponible (Sas), que es igual al Ss - Sss.

Hay algunas diferencias entre los algoritmos MPL y MPL-CV. Se tiene como meta la optimización, que es la utilización completa del espacio disponible; las vistas que no pertenecen al sistema Mss se pueden materializar en el procedimiento de optimización. Para acelerar el procedimiento de la optimización, una vista más grande que el espacio disponible no necesita ser considerada. De acuerdo con la discusión antedicha, se define una vista que se deba considerar en el procedimiento de la optimización como una vista candidata que requiere menos espacio que el espacio disponible restante. Eso es, el espacio candidato (Scs) es el tamaño total de todas las posibles vistas candidatas.

Entonces se define un cociente r como el valor de (espacio candidato) /2. Cuando r es menor que Sas, se materializan todas las vistas candidatas; las vistas de una pérdida métrica más baja se suprimen gradualmente hasta que se alcanza el límite de espacio disponible. Esta estrategia aumenta costo de la consulta y reduce costo de mantenimiento. Cuando r no es menor que Sas, se reducen al mínimo las vistas materializadas; las vistas de una ganancia métrica más alta se agregan progresivamente hasta que no hay espacio disponible, de tal modo se reduce el costo de la consulta y aumenta el costo de mantenimiento. El algoritmo de MPL-CV incluye 10 pasos, descrito más abajo.

Algoritmo. Algoritmo MPL-CV:

Paso 1: Determinar las vistas materializadas mínimas (Mss), y el espacio mínimo (Sss).

Paso 2: Determinar todas las vistas materializadas (Mls), y el espacio máximo (Sls).

Paso 3: Según el espacio dado (Ss), se computa el espacio disponible (Sas = Ss -Sss).

Si Ss ≥ Sls entonces
mv = Mls /* Materializar todas las vistas */
Ir al paso 10
Si Ss < Sss entonces
mv = Ø /* El minimo Ss puede ser > Sss */
Ir al paso 10
Fin del si
Si Ss = Sss entonces
Mv = Mss /* Vistas minimas */
Ir al paso 10
Fin del si.

Paso 4: Determinar todas las vistas materializadas candidatas (MCS), el espacio candidato (Scs), y la localización del punto medio (r = Scs/2).

Paso 5: **Si r ≥ Sas entonces**
mv = Mss

$$S_{mv} = \sum_{i \in mv} S_i$$

Sino
mv = Mss + Mcs

$$S_{mv} = \sum_{i \in mv} S_i$$

Si Smv ≤ Ss entonces
Ir al paso 10
Fin del si
Ir al paso 8
Fin del si.

Paso 6: Determinar el espacio disponible restante (Srs = Ss - Smv), y las nuevas vistas candidatas (MCS) que usan el vector de estructura de datos para computar el Cost (mv) del resultado del lattice basado en cada vista candidata, y seleccionar la vista con la ganancia métrica más alta (v).

Si $M_{cs} = \varnothing$ or $S_{rs} < \text{Min}(S_i)_{i \in Mcs}$ **Entonces**
Ir al paso 10
Fin del si.

Paso 7: **mv = mv + {v}**

$$S_{mv} = \sum_{i \in mv} S_i$$

Mcs = Mcs – {v}
Ir al paso 6

Paso 8: Utilizar el vector de estructura de datos para computar el Cost (mv) del resultado del lattice basado en cada vista candidata (MCS), y determinar la vista con menor pérdida métrica (v).

Paso 9: **mv = mv – {v}**

$$S_{mv} = \sum_{i \in mv} S_i$$

Mcs = Mcs – {v}
Si Ss < Smv entonces
Ir al paso 8
Fin del si.

Paso 10: Obtener los resultados del sistema de vistas materializadas (mv).

Finalmente, se decidió que para la virtualización del Data Warehouse se utilizara el modelo propuesto donde a partir de la obtención de un cubo de datos, se crea el lattice framework para, de esta manera, materializar las vistas finalmente seleccionadas, las de mayor eficiencia, mediante el algoritmo propuesto. Este modelo fue, definitivamente, seleccionado, porque tiene una mayor precisión, debido a que está basado en una métrica, y calcula el espacio almacenado buscando su disminución. En fin se reduce el costo de la consulta y se disminuye el espacio de almacenamiento, de forma que se acelera el proceso de un Data Warehouse.

CAPITULO 3: Caso de estudio.

El proyecto SIGEP (Sistema de Gestión Penitenciaria) se encarga de llevar el control de todas las prisiones de Venezuela. Este proyecto esta fraccionado en varios módulos que se encargan de las diversas informaciones necesarias sobre el trabajo realizado en las dichas prisiones. Uno de los principales objetivos de SIGEP consiste en la automatización del trabajo de las prisiones venezolanas. Entre sus módulos se encuentra el desarrollo de una Sala Situacional que brinda servicios a todos aquellos usuarios que interactúen con la aplicación y necesiten la información, entre los que pueden encontrarse los directores de las prisiones, delegados de prueba, custodios de los reclusos, entre otros. Las necesidades de información de los usuarios en este modulo son:

Internos por régimen intramuros: Internos que están pasando su condena dentro de la prisión.

Composición de la población
- Centro penitenciario
- Raza
- Nacionalidad
- Sexo
- Estado civil
- Categoría por cada delito

Característica sobre la situación Jurídica
- Situación legal (procesado y penado)
- Régimen penitenciario
- Progresividad
- Defunciones
- Evadidos
- Fugados
- Con solicitudes de revocatoria

De los internos en régimen extramuros: Internos que dado su comportamiento tienen varios estados.

Total de internos por cada uno de los estados en que se encuentran los internos.

Estados:

- Régimen ordinario: Internos que tienen trabajos en la calle y duermen en la prisión.
- Evadidos
- Sin presentarse

- Libertad condicional
- Confinamiento
- Defunciones
- Reposo medico
- Permisos judiciales
- Permisos extraordinarios
- Niveles de supervisión

Luego del estudio de los datos se evidenció la conveniencia de diseñar un Data Warehouse a partir de vistas materializadas para dar respuesta a las necesidades informacionales. A partir de las demandas que se pretenden satisfacer se utiliza el modelo mencionado en la sección anterior, el lattice framework.

Para esto es necesario la creación de un cubo de datos, por lo cual se definieron cuatro dimensiones importantes, Individuos (I), Datos de estancia (D), Expediente (E), y Ubicación geográfica (U).

Para responder las necesidades informacionales se realizaron las siguientes consultas:

Tabla de hechos

select tbl-individuo. id-individuo,
tbl-expediente. id-expediente,
tbl-datos-estancia.id-estacia,
tbl-ubicación-geografica.id-estado, tbl-ubicación-geografica.id-municipio,
tbl-ubicación-geografica.id-parroquia
from tbl-individuo
inner join tbl-expediente ***on*** tbl-individuo.id-individuo = tbl-expediente.id-individuo
inner join tbl-datos-estancia ***on*** tbl-datos-estancia. id-expediente = tbl-expediente. id-expediente
inner join tbl-ubicación-geografica ***on*** tbl-ubicación-geografica.id-municipio = tbl-individuo. id-municipio ***and*** tbl-ubicación-geografica.id-parroquia = tbl-individuo. id-parroquia;

Centro penitenciario:

select tbl-individuo. nombre-individuo,
tbl-datos-estancia.id-estacia,
from tbl-tabla-hechos
inner join tbl-individuo ***on*** tbl-individuo.id-individuo = tbl-tabla-hechos. id-individuo
inner join tbl-expediente ***on*** tbl-expediente.id-expediente = tbl-tabla-hechos. id-expediente
inner join tbl-datos-estancia ***on*** tbl-datos-estancia. id-estancia = tbl-tabla-hechos. id-estancia

inner join tbl-ubicación-geografica ***on*** tbl-ubicación-geografica.id-municipio = tbl-tabla-hechos. id-municipio ***and*** tbl-ubicación-geografica.id-parroquia = tbl-tabla-hechos. id-parroquia;
where
tbl-individuo.id-individuo = ¨? ¨ ***and*** tbl-datos-estancia. nombre-centro like ¨ ¨
group by tbl-individuo. nombre-individuo, tbl-datos-estancia. id-estancia;

Raza:

select tbl-individuo. . nombre-individuo,
tbl-individuo. color-piel,
from tbl-tabla-hechos
inner join tbl-individuo ***on*** tbl-individuo.id-individuo = tbl-tabla-hechos. id-individuo
inner join tbl-expediente ***on*** tbl-expediente.id-expediente = tbl-tabla-hechos. id-expediente
inner join tbl-datos-estancia ***on*** tbl-datos-estancia. id-estancia = tbl-tabla-hechos. id-estancia
inner join tbl-ubicación-geografica ***on*** tbl-ubicación-geografica.id-municipio = tbl-tabla-hechos. id-municipio ***and*** tbl-ubicación-geografica.id-parroquia = tbl-tabla-hechos. id-parroquia;
where
tbl-individuo.id-individuo = ¨ ?¨ ***and*** tbl-individuo. color-piel = ¨negra¨ ***or***
tbl-individuo. color-piel =¨mestiza¨ ***or*** tbl-individuo. color-piel = ¨blanca¨
group by tbl-individuo. nombre-individuo, tbl-individuo. color-piel;

Nacionalidad

select tbl-individuo. nombre-individuo,
tbl-individuo. nombre-nacionalidad,
from tbl-tabla-hechos
inner join tbl-individuo on tbl-individuo.id-individuo = tbl-tabla-hechos. id-individuo
inner join tbl-individuo ***on*** tbl-individuo.id-individuo = tbl-tabla-hechos. id-individuo
inner join tbl-expediente ***on*** tbl-expediente.id-expediente = tbl-tabla-hechos. id-expediente
inner join tbl-datos-estancia ***on*** tbl-datos-estancia. id-estancia = tbl-tabla-hechos. id-estancia
inner join tbl-ubicación-geografica ***on*** tbl-ubicación-geografica.id-municipio = tbl-tabla-hechos. id-municipio ***and*** tbl-ubicación-geografica.id-parroquia = tbl-tabla-hechos. id-parroquia;
where
tbl-individuo.id-individuo = ¨? ¨ ***and*** tbl-individuo. nacionalidad = ¨ ¨
group by tbl-individuo. nombre-individuo, tbl-individuo. nacionalidad;

Sexo

select tbl-individuo. nombre-individuo,
tbl-individuo. sexo,
from tbl-tabla-hechos
inner join tbl-individuo ***on*** tbl-individuo.id-individuo = tbl-tabla-hechos. id-individuo
inner join tbl-expediente ***on*** tbl-expediente.id-expediente = tbl-tabla-hechos. id-expediente
inner join tbl-datos-estancia ***on*** tbl-datos-estancia. id-estancia = tbl-tabla-hechos. id-estancia

inner join tbl-ubicación-geografica ***on*** tbl-ubicación-geografica.id-municipio = tbl-tabla-hechos. id-municipio ***and*** tbl-ubicación-geografica.id-parroquia = tbl-tabla-hechos. id-parroquia;
where
tbl-individuo.id-individuo = ¨ ?¨ ***and*** tbl-individuo. sexo = ¨ masculino ¨ ***or***
tbl-individuo. sexo = ¨ femenino ¨
group by tbl-individuo. nombre-individuo, tbl-individuo. sexo;

Estado civil

select tbl-individuo. nombre-individuo,
tbl-individuo. nombre-estado-civil,
from tbl-tabla-hechos
inner join tbl-individuo ***on*** tbl-individuo.id-individuo = tbl-tabla-hechos. id-individuo
inner join tbl-expediente ***on*** tbl-expediente.id-expediente = tbl-tabla-hechos. id-expediente
inner join tbl-datos-estancia ***on*** tbl-datos-estancia. id-estancia = tbl-tabla-hechos. id-estancia
inner join tbl-ubicación-geografica ***on*** tbl-ubicación-geografica.id-municipio = tbl-tabla-hechos. id-municipio ***and*** tbl-ubicación-geografica.id-parroquia = tbl-tabla-hechos. id-parroquia;
where
tbl-individuo.id-individuo = ¨? ¨ ***and*** tbl-individuo. nombre-estado-civil = ¨ soltero ¨ ***or***
tbl-individuo. nombre-estado-civil = ¨ casado¨ ***or***
tbl-individuo. nombre-estado-civil = ¨ viudo ¨
or tbl-individuo. nombre-estado-civil = ¨ divorciado ¨ ***or***
tbl-individuo. nombre-estado-civil = ¨ concubino ¨
group by tbl-individuo. nombre-individuo, tbl-individuo. nombre-estado-civil;

Categoría por delitos

select tbl-individuo. nombre-individuo,
tbl-expediente.nombre-delito,
tbl-expediente.activo,
tbl-datos-estancia.fecha-ingreso-estancia,
tbl-datos-estancia.fecha-egreso-estancia,
tbl-individuo-sexo,
tbl-individuo.nombre-nacionalidad,
tbl-datos-estancia.nombre-centro,
tbl-datos-estancia.nombre-tipo-centro,
from tbl-tabla-hechos
inner join tbl-individuo ***on*** tbl-individuo.id-individuo = tbl-tabla-hechos. id-individuo
inner join tbl-expediente ***on*** tbl-expediente.id-expediente = tbl-tabla-hechos. id-expediente
inner join tbl-datos-estancia ***on*** tbl-datos-estancia. id-estancia = tbl-tabla-hechos. id-estancia
inner join tbl-ubicación-geografica ***on*** tbl-ubicación-geografica.id-municipio = tbl-tabla-hechos. id-municipio ***and*** tbl-ubicación-geografica.id-parroquia = tbl-tabla-hechos. id-parroquia;
where
tbl-expediente.probado = 1 ***and*** tbl-datos-estancia.nombre-tipo-centro like "
group by tbl-individuo. nombre-individuo, tbl-expediente. nombre-delito,
tbl-expediente.activo, tbl-datos-estancia.fecha-ingreso-estancia,

tbl-datos-estancia.fecha-egreso-estancia, tbl-individuo.sexo,
tbl-individuo.nombre-nacionalidad, tbl-datos-estancia.nombre-centro,
tbl-datos-estancia.nombre-tipo-centro;

Situación legal (procesados)

select tbl-individuo. nombre-individuo,
tbl-expediente. id-expediente,
tbl-expediente. fecha-ingreso-sistema,
tbl-expediente. fecha-egreso-sistema,
from tbl-tabla-hechos
inner join tbl-individuo ***on*** tbl-individuo.id-individuo = tbl-tabla-hechos. id-individuo
inner join tbl-expediente ***on*** tbl-expediente.id-expediente = tbl-tabla-hechos. id-expediente
inner join tbl-datos-estancia ***on*** tbl-datos-estancia. id-estancia = tbl-tabla-hechos. id-estancia
inner join tbl-ubicación-geografica ***on*** tbl-ubicación-geografica.id-municipio = tbl-tabla-hechos. id-municipio ***and*** tbl-ubicación-geografica.id-parroquia = tbl-tabla-hechos. id-parroquia;
where
tbl-expediente. activo = ¨1¨ ***and*** tbl-expediente. id-pena = null
group by tbl-individuo. nombre-individuo, tbl-expdiente.id-expediente,
tbl-expediente. fecha-ingreso-sistema, tbl-expediente. fecha-egreso-sistema;

Situación legal (penados)

select tbl-individuo. nombre-individuo,
tbl-expediente. id-expediente,
tbl-expediente. fecha-ingreso-sistema,
tbl-expediente. fecha-egreso-sistema,
from tbl-tabla-hechos
inner join tbl-individuo ***on*** tbl-individuo.id-individuo = tbl-tabla-hechos. id-individuo
inner join tbl-expediente ***on*** tbl-expediente.id-expediente = tbl-tabla-hechos. id-expediente
inner join tbl-datos-estancia ***on*** tbl-datos-estancia. id-estancia = tbl-tabla-hechos. id-estancia
inner join tbl-ubicación-geografica ***on*** tbl-ubicación-geografica.id-municipio = tbl-tabla-hechos. id-municipio ***and*** tbl-ubicación-geografica.id-parroquia = tbl-tabla-hechos. id-parroquia;
where
tbl-expediente. activo = 1 ***and*** tbl-expediente. id-pena is not null
group by tbl-individuo. nombre-individuo, tbl-expdiente.id-expediente,
tbl-expediente. fecha-ingreso-sistema, tbl-expediente. fecha-egreso-sistema;

Régimen penitenciario

select tbl-individuo. nombre-individuo,
tbl-datos-estancia.id-estacia,
from tbl-tabla-hechos
inner join tbl-individuo ***on*** tbl-individuo.id-individuo = tbl-tabla-hechos. id-individuo
inner join tbl-expediente ***on*** tbl-expediente.id-expediente = tbl-tabla-hechos. id-expediente

inner join tbl-datos-estancia ***on*** tbl-datos-estancia. id-estancia = tbl-tabla-hechos. id-estancia
inner join tbl-ubicación-geografica ***on*** tbl-ubicación-geografica.id-municipio = tbl-tabla-hechos. id-municipio ***and*** tbl-ubicación-geografica.id-parroquia = tbl-tabla-hechos. id-parroquia;
where
tbl-datos-estancia. nombre-centro like ¨ ¨
group by tbl-individuo. nombre-individuo, tbl-datos-estancia. id-estancia;

Progresividad

select tbl-individuo. nombre-individuo,
tbl-expediente.fecha,
tbl-expediente. nombre-estado-progresividad,
from tbl-tabla-hechos
inner join tbl-individuo ***on*** tbl-individuo.id-individuo = tbl-tabla-hechos. id-individuo
inner join tbl-expediente ***on*** tbl-expediente.id-expediente = tbl-tabla-hechos. id-expediente
inner join tbl-datos-estancia ***on*** tbl-datos-estancia. id-estancia = tbl-tabla-hechos. id-estancia
inner join tbl-ubicación-geografica ***on*** tbl-ubicación-geografica.id-municipio = tbl-tabla-hechos. id-municipio ***and*** tbl-ubicación-geografica.id-parroquia = tbl-tabla-hechos. id-parroquia;
where
tbl-expediente.activo= 1
group by tbl-individuo. nombre-individuo, tbl-expediente.fecha,
tbl-expediente. nombre-estado-progresividad;

Motivos de egreso

select tbl-individuo. nombre-individuo,
tbl-datos-estancia. nombre-motivo-egreso,
from tbl-tabla-hechos
inner join tbl-individuo ***on*** tbl-individuo.id-individuo = tbl-tabla-hechos. id-individuo
inner join tbl-expediente ***on*** tbl-expediente.id-expediente = tbl-tabla-hechos. id-expediente
inner join tbl-datos-estancia ***on*** tbl-datos-estancia. id-estancia = tbl-tabla-hechos. id-estancia
inner join tbl-ubicación-geografica ***on*** tbl-ubicación-geografica.id-municipio = tbl-tabla-hechos. id-municipio ***and*** tbl-ubicación-geografica.id-parroquia = tbl-tabla-hechos. id-parroquia;
where
tbl-expediente.activo= 1 ***and***
tbl-datos-estancia. nombre-motivo-egreso = ¨defuncion¨ ***or***
tbl-datos-estancia. nombre-motivo-egreso = ¨fuga¨ ***or***
tbl-datos-estancia. nombre-motivo-egreso = ¨evadidos¨
group by tbl-individuo. nombre-individuo, tbl-datos-estancia. nombre-motivo-egreso;

Solicitudes revocatorias

select tbl-individuo. nombre-individuo,
tbl-datos-estancia. nombre-tipo-documento,

from tbl-tabla-hechos
inner join tbl-individuo ***on*** tbl-individuo.id-individuo = tbl-tabla-hechos. id-individuo
inner join tbl-expediente ***on*** tbl-expediente.id-expediente = tbl-tabla-hechos. id-expediente
inner join tbl-datos-estancia ***on*** tbl-datos-estancia. id-estancia = tbl-tabla-hechos. id-estancia
inner join tbl-ubicación-geografica ***on*** tbl-ubicación-geografica.id-municipio = tbl-tabla-hechos. id-municipio ***and*** tbl-ubicación-geografica.id-parroquia = tbl-tabla-hechos. id-parroquia;
where
tbl-expediente.activo= 1
group by tbl-individuo. nombre-individuo, tbl-datos-estancia. nombre-tipo-documento;

Régimen extramuros

select tbl-individuo. nombre-individuo,
tbl-datos-estancia.id-estacia,
from tbl-tabla-hechos
inner join tbl-individuo ***on*** tbl-individuo.id-individuo = tbl-tabla-hechos. id-individuo
inner join tbl-expediente ***on*** tbl-expediente.id-expediente = tbl-tabla-hechos. id-expediente
inner join tbl-datos-estancia ***on*** tbl-datos-estancia. id-estancia = tbl-tabla-hechos. id-estancia
inner join tbl-ubicación-geografica ***on*** tbl-ubicación-geografica.id-municipio = tbl-tabla-hechos. id-municipio ***and*** tbl-ubicación-geografica.id-parroquia = tbl-tabla-hechos. id-parroquia;
where
tbl-datos-estancia. nombre-centro like ¨ ¨ ***or*** tbl-expediente.tipo-estado=regimen-ordinario ***or*** tbl-expediente.tipo-estado=evadidos ***or*** tbl-expediente.tipo-estado=sin-presentarse ***or*** tbl-expediente.tipo-estado=libertad-condicional ***or*** tbl-expediente.tipo-estado=confinamiento ***or*** tbl-expediente.tipo-estado=defunción ***or*** tbl-expediente.tipo-estado=reposo ***or*** tbl-expediente.tipo-estado=permisos-judiciales ***or*** tbl-expediente.tipo-estado=permisos-extraordinarios ***or*** tbl-expediente.tipo-estado=nivel-supervision
group by tbl-individuo. nombre-individuo, tbl-datos-estancia. id-estancia;

Luego se realiza el cubo de datos, donde cada célula contiene la relación entre las dimensiones: Individuos (I), Datos de estancia (D), Ubicación Geográfica (U) y Expediente (E), ilustrado en la fig. 19:

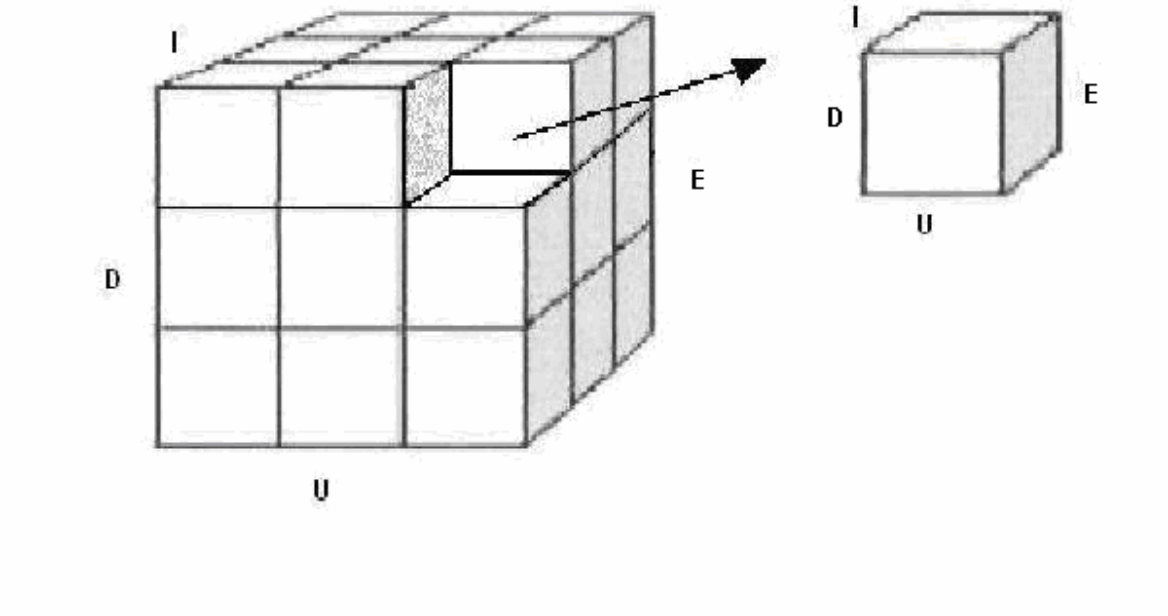

Figura 19 Relación entre individuos, datos de estancia, ubicación geográfica y expediente.

Posteriormente se construye el Lattice framework de este cubo de datos, donde la dependencia entre las vistas se representan a través de líneas de conexión. Para este lattice se tiene cuatro dimensiones por lo que presenta 16 posibles vistas. Esto se muestra en la figura 20:

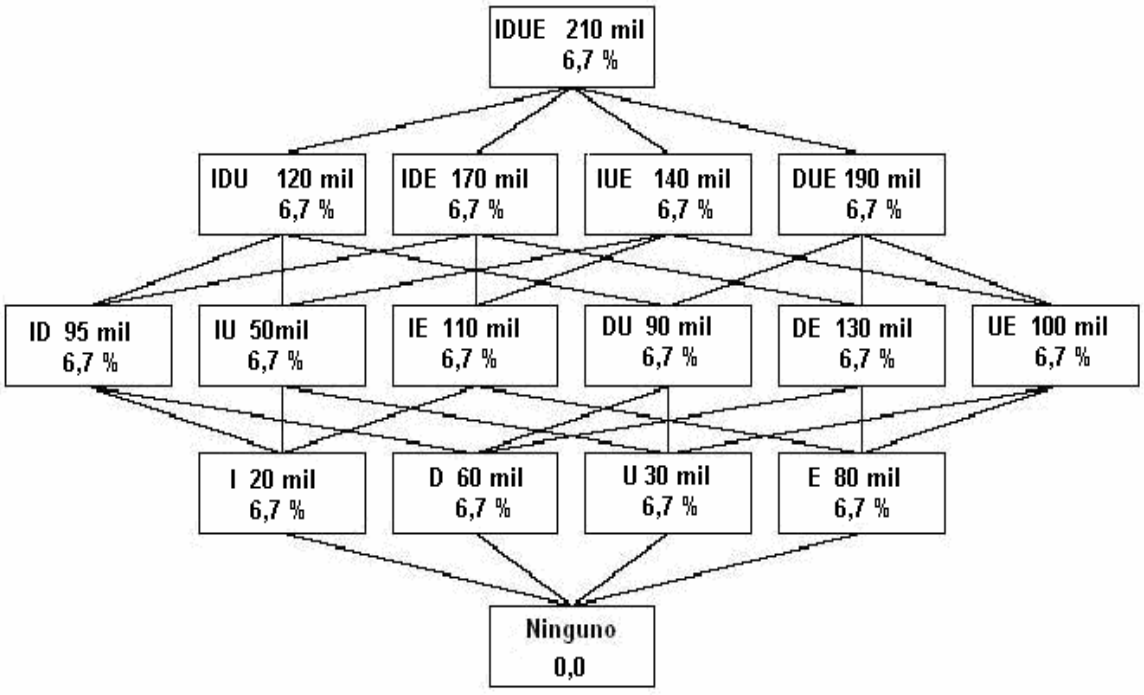

Figura 20 Lattice framework del cubo de datos

En este lattice se tiene como vista tope la vista IDUE de la cual otras vistas dependen. Las dependencias entres las vistas se busca por la relación que existe entre ellas, en este lattice se tiene cuatro dimensiones por lo cual se debe buscar la relación entre cada una de ellas, por ejemplo, la conexión entre la vista IDUE con la vista IDE significa que entre ellas existe dependencia, es decir que la vista IDE es un subconjunto propio de la vista IDUE, y la vista IDUE es su vista tope. En la figura 20 se muestra el lattice y los número al lado de la vistas representan la cantidad de datos y la frecuencia de las consultas.

Suponiendo que las cantidades de datos y la frecuencia de las consultas son las que muestran la tabla 9. Se tiene por ejemplo que la vista IDE tiene como cantidad de datos 170 mil y una frecuencia de 6,7 %. En este caso de estudio se toma como frecuencia de las consultas 6,7 % para todas las consultas, debido a que todavía no esta definido cual es la frecuencia de las consultas por lo que se asumió que todas las consultas se realizaran con la misma frecuencia.

Vistas	Cantidad de datos	Frecuencia consultas
IDUE	210 mil	6,7 %
IDU	120 mil	6,7 %
IDE	170 mil	6,7 %
IUE	140 mil	6,7 %
DUE	190 mil	6,7 %
ID	95 mil	6,7 %
IU	50 mil	6,7 %
IE	110 mil	6,7 %
DU	90 mil	6,7 %
DE	130 mil	6,7 %
UE	100 mil	6,7 %
I	20 mil	6,7 %
D	60 mil	6,7 %
U	30 mil	6,7 %
E	80 mil	6,7 %

Tabla 9: Las vistas y sus datos.

El lattice se utiliza para expresar el cubo de datos, cuya representación se expresa en vectores. Por lo que se le dio a cada vista un numero binario: para IDUE se le dio 1111, 1011 para IDU, 1011 para IDE, 1110 para IUE, para DUE se le asigno 0111, así sucesivamente, teniendo en cuenta que para alcanzar los niveles superiores de las vistas dependientes es necesario que un bit 0 se gire en un bit 1. por ejemplo: se observa que las vistas topes de ID son la vista IDU, la vista IDE, y la vista IUE, para ir de la vista ID con un numero binario (1100) a la vista IDE con un numero binario (1110), que se encuentra en un nivel superior se cambia el ultimo bit de la vista ID para el bit 1 y así se llega a la vista IDE.

También para alcanzar niveles inferiores es necesario girar un bit 1 en un bit 0, por ejemplo se tiene la vista ID con un numero binario (1100), como se vio anteriormente, ahora para ir a un nivel inferior en este caso puede ir a la vista I con un numero binario (1000) o la vista D con un numero binario (0100), para ir a la vista I se gira tercer bit de mirando se derecha a izquierda y para llegar de ID a la vista D es necesario girar el cuarto bit de la vista ID. Si se usa esta representación, los costos de la consulta y el mantenimiento de las vistas dependientes pueden ser obtenidas fácilmente remontando los bit 0 y 1. Esto se demuestra en la fig. 21.

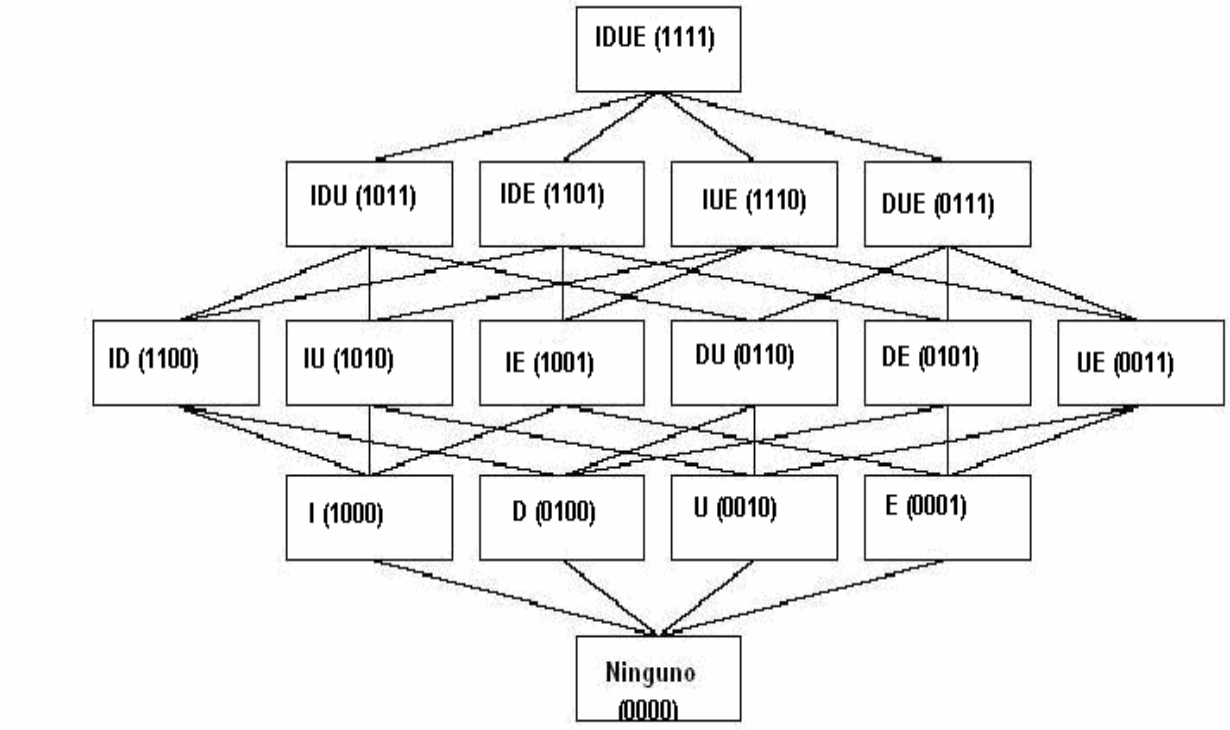

Figura 21 Representación del lattice para el cubo de datos.

Esta representación en vectores se utiliza para encontrar vistas materializadas dependientes. Esto es que una vez que se tienen las vistas materializadas, para mejorar la eficiencia de la consulta se selecciona la vista con capacidad más pequeña para procesar, si el número de sus vistas dependientes es mayor que uno.
Para este caso de estudio las vistas materializadas que se obtienen son la vista IDU, la vista IUE, la vista IU y la vista DE. Esta selección esta basada en la capacidad de las vistas donde se busca la vista con menor capacidad, además de que con ella se pueda responder otras vistas.
Asumiendo que se materializa la vista IDU a partir de ella se pueden responder la vista ID, la vista IU, la vista DU, y la vista IDUE. Si se materializa para este lattice la vista IUE, entonces respondería la vista IU, la vista IE, la vista UE, la vista IDUE. La vista materializada IU responde las vistas: I, U, IDU y la vista IUE, y finalmente si se materializa la vista DE la cual responde la vista D, la vista E, la vista DUE y la vista IDE.

En conclusión después del proceso realizado con las vistas se obtuvo un sistema de vistas las cuales se materializaran para un mayor rendimiento del proceso, y para disminuir el espacio de almacenaje.

Conclusiones

En la actualidad, la presencia en información útil y disponible es el motivo del surgimiento de los sistemas de soporte de decisiones.

El aparecimiento de técnicas de optimización aumenta el desempeño de esos sistemas. La materialización de vistas es una técnica de optimización del procesamiento de las consultas, en ambientes Data Warehouse.

En esta investigación se definió un algoritmo para la selección de vistas materializadas para de esta forma disminuir el espacio de almacenaje y el costo de procesamiento. A través del lattice framework se selecciona las vistas que con una menor capacidad conlleve a responder las consultas. De esta forma se obtiene un sistema de vistas materializadas que finalmente serán cargadas al Data Warehouse, disminuyendo el costo de estos procesos.

En el futuro, los Data Warehouse y los sistemas de toma de decisión serán imprescindibles aplicaciones para cualquier empresa. Para incitar a la calidad de la eficiencia de un sistema de toma de decisión, los Data Warehouse el soportan la optimización de las consultas, es una tendencia inevitable. Por lo tanto, las vistas materializadas juegan un papel importante y su funcionamiento merecen un estudio adicional.

Recomendaciones

El uso de vistas materializadas tanto para la virtualización como para el mejoramiento del rendimiento de los data Warehouse es una línea abierta de investigación a nivel mundial en la que no esta escrita la ultima palabra, este trabajo propone la utilización de uno de los algoritmo de mayor éxito practico para la selección de vistas materializadas en el proceso de desarrollo de Data Warehouse que se recomienda sea llevado a cabo de manera mas eficiente y desarrollado como una línea independiente del grupo de nuevas tecnologías de bases de datos con vistas a obtener resultados que puedan ser avalados desde la producción que muestren el incremento en rendimiento de las aplicaciones y por tanto contribuyan al avance de la universidad.

Referencias bibliográficas

C. Zhang, X. Y., J. Yang (2001) An evolutionary approach to materialized views selection in a data warehouse environment. Volume, DOI:

C.S. Park, M. H. K., Y.J. Lee (2002) Finding an efficient rewriting of OLAP queries using materialized views in Data Warehouses. Volume, DOI:

CRISTINA, I. I. E., J. F (2006) O Uso de Visões Materializadas em Data Warehouses. Volume, DOI:

D. Theodoratos, T. S. (1999) Designing Data Warehouses. Volume, DOI:

D. Theodoratos, T. S. (2000) Answering multidimensional queries in cubes using other cubes. Volume, DOI:

D.L. Yang, M. L. H., M.C. Hung (2002) Efficient utilization of materialized views in a Data Warehouse. Volume, DOI:

Dimitri Theodoratos, W. X. (2004) Constructing search spaces for materialized view selection. Volume, DOI:

Gang Gou, J. X. Y., Hongjun Lu (2004) A* Search: An Efficient and Flexible Approach to Materialized View Selection. Volume, DOI:

Gupta, H. (1997) Selection of Views to Materialize in a Data Warehouse. Volume, DOI:

Gupta, H. (1999) Selection and Maintenance of Views in a Data Warehouse. Volume, DOI:

H. Gupta, I. S. M. (1999) Selection of views to materialize under a maintenance cost constraint. Volume, DOI:

Himanshu Gupta, I. S. M. (2005) Selection of Views to Materialize in a Data Warehouse. Volume, DOI:
Himanshu Gupta, V. H., Anand Rajaraman, Jeffrey D. Ullman (1997) Index Selection for OLAP. Volume, DOI:

J. Gray, S. C., A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, H. Pirahesh (1997) Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals. Volume, DOI:

J. Yang, K. K., Q. Li (1996) A framework for designing materialized views in data warehousing environment. Volume, DOI:

J. Yang, K. K., Q. Li (1997) A framework for designing materialized views in data warehousing environment. Volume, DOI:

J.X. Yu, X. Y., C.H. Choi, G. Gou (2003) Materialized view selection as constrained evolutionary optimization. Volume, DOI:

K.A. Ross, D. S., S. Sudarshan (1996) Materialized view maintenance and integrity constraint checking: trading space for time. Volume, DOI:

Panos Kalnis, N. M., Dimitris Papadias (2002) View selection using randomized search. Volume, DOI:

S. Agrawal, S. C., V. Narasayya (2000) Automated selection of materialized views and indexes for SQL databases. Volume, DOI:

S. Flesca, S. G. (2001) Rewriting queries using views. Volume, DOI:

V. Harinarayan, A. R., J.D. Ullman (1996) Implementing data cubes efficiently. Volume, DOI:

W. Liang, H. L., M.E. Orlowska (1999) Making multiple views self-maintainable in a Data Warehouse. Volume, DOI:

W. Liang, H. W., M.E. Orlowska (2001) Materialized view selection under the maintenance time constraint. Volume, DOI:

W.Y. Lin, I. C. K. (2004) A genetic selection algorithm for OLAP data cubes. Volume, DOI:

Y. Liu, S. Y. S., H. Xiong (2006) A cubic-wise balance approach for privacy preservation in data cubes. Volume, DOI:

Y.H. Chen, Y. C. L. (1997) Selecting materialized views in Data Warehouses. Volume, DOI:

VIDAL, L. V. and M. V. MONTEAGUDO. Estudio teorico conceptual sobre Data Warehouse, 2000. p.

Glosario de Términos

Sistema de soporte de decisiones (DSS): Es una herramienta de utilizada en el campo de la inteligencia del negocio que permite realizar el análisis de las diferentes variables de negocio para apoyar una decisión:

- Permite extraer y manipular información de una manera flexible
- Ayuda en decisiones no estructuradas
- Permite al usuario definir, interactivamente, que información necesita, y como combinarla.

OLAP: (Procesamiento analítico en línea).Sustenta el estudio del comportamiento del negocio y su proyección. Se caracteriza por el análisis dimensional y dinámico de los datos consolidados de la empresa apoyando las actividades de análisis y navegación del usuario terminal. Ayuda al usuario a sintetizar la información de la empresa a través de vistas personalizadas, análisis históricos y pronósticos.

Sistemas operacionales: Son aquellos que permiten realizar las funciones de una empresa, se utilizan en el funcionamiento de los negocio en tiempo real, basándose en datos actuales.

Sistemas informacionales: Son aquellos que permiten estudiar el comportamiento de la empresa, se utilizan para administrar y controlar la empresa. Se basan en datos históricos.

Data Mart: Consta de las mismas de las mismas características de un data warehouse y brinda las mismas facilidades, pero esta orientado a una sola actividad y a no satisfacer las necesidades de toda la empresa.

Datos históricos: Son datos estables que se utilizan en un momento en el tiempo, es decir que pueden trascender con el tiempo.

Transacción: consiste en una interacción con una estructura de datos que, aún siendo compleja y estar compuesta por varios procesos que se han de aplicar uno después del

otro. Es decir, que se realice de una sola vez y que la estructura a medio manipular no sea jamás alcanzable por el resto del sistema.

SGBD: Los Sistemas de gestión de base de datos son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta.

RDBMS: El sistema de manejo de bases de datos realcionales esta basado en el modelo relacional. Las bases de datos relacionales es el tipo más común de bases de datos usadas en la actualidad.

ACID: Se usa para describir cuatro propiedades:

- Automaticidad: una transacción debe ser realizada o desecha completamente. En el vento de un fallo, todas las operaciones y procedimientos deben ser desechos y todos los datos deben ser devueltos a su estado previo.
- Consistencia: Una transacción debe transformar un sistema de un estado consistente a otro estado consistente.
- Aislamiento: Cada transición debe pasar independiente de otras transacciones ocurriendo al mismo tiempo.
- Durabilidad: Transacciones completadas deben mantenerse permanentes aún durante fallos del sistema.

Grafo acíclico: Es un grafo que no contiene ciclos. Los ciclos son caminos tal que u_1, u2,..., up_{-1} son diferentes, y $u_p=u_1$, es decir, es un camino cerrado.

Subconjunto propio: Se dice que B es un subconjunto propio de C si todo elemento x que pertenece a B pertenece a C, pero existe al menos un elemento que pertenece a C y que no pertenece a B.

Orden parcial: Es una relación binaria *R* sobre un conjunto *X* que es reflexiva, antisimétrica, y transitiva, es decir, para cualesquiera *a*, *b*, y *c* en *X* se tiene que:

- aRa (reflexividad).
- Si aRb y bRa, entonces a = b (antisimetría).
- Si aRb y bRc, entonces aRc (transitividad).

Sea A un conjunto cualquiera; se dice que R es una relación binaria A×A, es decir, si R es un subconjunto del producto cartesiano citado.

Una relación binaria R sobre un conjunto X es reflexiva si se cumple que para todo x perteneciente a X, x está relacionado consigo mismo.

Una relación binaria R sobre un conjunto X es antisimétrica si se cumple que para todo a y b pertenecientes a X si a está relacionado con b y b está relacionado con a entonces a = b.

R sobre un conjunto X es transitiva si se cumple que para todo a, b, y c pertenecientes a X, que si a está relacionado con b y b está relacionado con c, entonces a está relacionado con c.