Universidad del Bío-Bío, Chile

Facultad de Ciencias Empresariales

Departamento de Sistemas de Información

Estructuras de Datos Compactas para Representar Data Warehouses en Memoria Principal

Tesis presentada por Cristian Manuel Vallejos Vega
para obtener el grado de Magíster en Ciencias de la Computación
Dirigida por Dra. Mónica Caniupán Marileo
Dr. Gilberto Gutiérrez Retamal

2017

Agradecimientos

En primer lugar, agradezco a Dios por darme la salud y la fortaleza de seguir adelante con este trabajo a pesar de las dificultades.

A mis directores de tesis, por todo el apoyo durante el proceso de desarrollo de esta tesis. A la Dra. Mónica Caniupán , por su paciencia, correcciones, consejos y acompañamiento durante todo este tiempo. Al Dr. Gilberto Gutiérrez por aquellas extensas reuniones de trabajo y discusiones académicas que fueron de mucha ayuda.

A los académicos Dr. Gonzalo Navarro (U. de Chile), Dra. Nieves Brisaboa (U. da Coruña), Dr. Diego Seco (U. de Concepción), Dr. Miguel Romero (U. del Bío-Bío), Mg. Raúl Soto (U. de Los Lagos), por contribuir a esta tesis con su amplio conocimiento en sus respectivas áreas, mediante alguna reunión o respuestas vía correo electrónico.

A Carlos Aburto, quien gracias a su trabajo de titulación hizo muy buenos aportes a la discusión de este trabajo de tesis.

A mis compañeros y amigos del Magíster en Ciencias de la Computación, Jorge, Joel, Luis, Raúl y Juan José, quienes hicieron que la permanencia en el programa fuera agradable y de mucho provecho.

A mis padres, hermanas y abuelos por la constante preocupación y palabras de ánimo en todo momento durante mi paso por el Magister.

A mis amigos más cercanos, a mis colegas, a mis estudiantes y ex estudiantes, por todas sus muestras de apoyo y acompañamiento.

A todos los académicos y al personal administrativo del Magíster en Ciencias de la Computación, por cada uno de sus aportes valiosos a mi formación.

Y a todo aquel que de una u otra forma contribuyo a que este trabajo fuera finalizado.

Esta tesis se encuentra enmarcada en el Proyecto Exploratorio “Estructuras de Datos Compactas para procesar eficientemente datos espaciales y espacio-temporales en el contexto de Big-Data”, Programa Ingeniería 2030 código 1638 y el proyecto de investigación DIUBB “Estructuras de Datos Compactas para representar y consultar grandes conjuntos de datos espaciales” código 171319 4/. Además, la tesis fue apoyada por el grupo de investigación ALBA código GI 160119/EF.

Abstract

In this thesis we propose the use of the compact data structure k^2-treap to process data cubes of Data Warehouses (DWs) into main memory. Compact data structures are data structures that allow compacting the data without losing the capacity of querying them in their compact form.

A DW is a data repository to store historical data for decision support, and consists of *dimensions* and *facts*. The former are an abstract concept that groups data with a similar meaning, and are modelled as hierarchies of levels, which contain elements. The latter are quantitative data associated to dimensions. A data cube is a typical way to retrieve facts at different levels of granularity (through navigation on dimension hierarchies). A DW can store terabytes of data, thus the efficient processing of data cubes is key in OLAP (On-line Analytical Processing).

We show that by using a compact representation of a data cube and bitmaps to represent dimensions we are able to improve the use of space in main memory, and achieve better performance for query processing. We consider aggregate queries with aggregate functions `SUM` and `MAX`, but the work can be easily extended to the aggregate functions `AVG`, and `COUNT`. However, the k^2-treap compact data structure was not designed to answer efficiently queries with `MIN` function.

Keywords — Databases, data warehousing, compact data structures.

Resumen

En esta tesis proponemos el uso de la estructura de datos compacta k^2-treap para procesar cubos de datos de Data Warehouses (DWs) en memoria principal. Las estructuras de datos compactas son estructuras de datos que permiten compactar los datos sin perder la capacidad de consultarlos en su forma compacta.

Un DW es un repositorio de datos que almacena datos históricos para apoyar la toma de decisiones, y se componen de *dimensiones* y *hechos*. Las primeras son un concepto abstracto que agrupa datos con un significado similar, son modeladas como jerarquías de niveles, los cuales contienen elementos. Los últimos son datos cuantitativos asociados a las dimensiones. Un cubo de datos es una forma típica de recuperar hechos a diferentes niveles de granularidad (a través de la navegación por las jerarquías de las dimensiones). Un DW puede almacenar terabytes de datos, por lo que el procesamiento eficiente de los cubos de datos es clave en OLAP (On-line Analytical Processing).

Mediante experimentación sobre DWs con datos sintéticos mostramos que utilizando una representación compacta de un cubo de datos y bitmaps para representar las dimensiones de un DW podemos mejorar el uso de almacenamiento en memoria principal, y lograr un mejor rendimiento en el procesamiento de consultas de agregación. En esta tesis consideramos consultas con funciones de agregación `SUM` y `MAX`, pero el trabajo puede ser fácilmente extendido a otras funciones de agregación, tales como, `AVG` (promedio), y `COUNT`. Sin embargo, la estructura compacta k^2-treap no fue diseñada para responder de manera eficiente consultas de agregación con la función `MIN`.

Palabras Clave — Bases de datos, data warehousing, estructuras de datos compactas.

Índice General

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

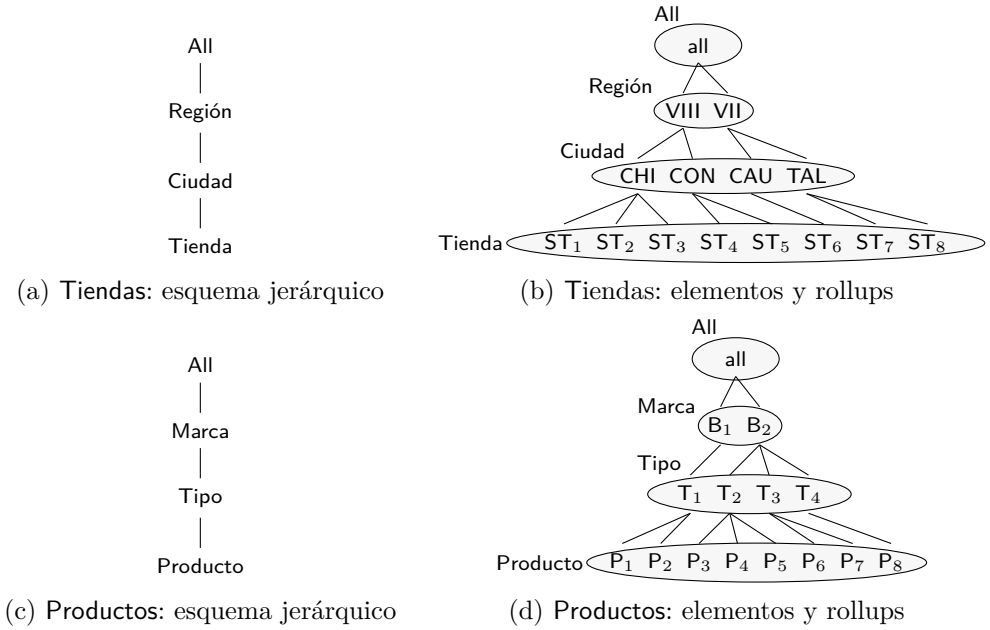
Capítulo 1

Introducción

La forma tradicional de consultar grandes conjuntos de datos es traer los datos desde memoria secundaria (por ejemplo, desde discos) hasta memoria principal y luego procesar las consultas en la memoria principal. Esto puede ser apoyado con el uso de índices para acceder a los datos adecuados desde el disco de manera rápida. Sin embargo, en los últimos años se ha impulsado la tendencia de utilizar la memoria principal, tanto, para consultar los datos, como para almacenarlos (Garcia-Molina y Salem, 1992; Plattner y Zeier, 2011; Ross y Zaman, 2000). Al tener los datos en memoria principal podemos lograr mayor eficiencia en el procesamiento dada la rapidez de esta memoria. Por ejemplo, la referencia a memoria principal toma 100 nanosegundos, pero las operaciones de lectura y escritura desde disco pueden tomar 5 milisegundos (Plattner y Zeier, 2011). Sin embargo, la memoria principal es más cara que la memoria secundaria, por lo cual es necesario usarla eficientemente. Para lograr este objetivo, podemos utilizar *estructuras de datos compactas*. Las estructuras de datos compactas son estructuras de datos que usan poco espacio pero conservan la capacidad de acceso directo a los datos, sin la necesidad de volver al estado original de los datos (al contrario de lo que sucede si los datos se comprimen), lo que permite mantener su funcionalidad y realizar operaciones de consulta de forma eficiente (Raman et al., 2001). Además, las estructuras de datos compactas permiten procesar grandes conjuntos de datos en memoria principal, evitando parcial o completamente el acceso a memorias externas como los discos, ya que pueden ser almacenadas en los niveles más altos de la jerarquía de memoria (cerca de la CPU), donde el tiempo de acceso disminuye con respecto a los niveles más bajos de la jerarquía de memoria.

Las estructuras de datos compactas se han utilizado en diversas áreas. Por ejemplo, en (Brisaboa et al., 2009, 2014b; Claude y Navarro, 2007) son usadas para representar grafos de la Web. En (Brisaboa et al., 2013a; Claude y Navarro, 2009; Navarro, 2014; Navarro y Sadakane, 2014) permiten la representación de documentos en el contexto de recuperación de información. También, se han utilizado para mejorar la eficiencia de las consultas en Sistemas de Información Geográfica (Brisaboa et al., 2013b; De Bernardo et al., 2013; Ladra et al., 2017) (ver Capítulo 3 para mayores detalles).

En esta tesis proponemos el uso de la estructura de datos compacta k^2-treap para computar consultas de agregación con operadores `SUM` y `MAX` sobre cubos de datos de un



(a) Tiendas: esquema jerárquico

(b) Tiendas: elementos y rollups

(c) Productos: esquema jerárquico

(d) Productos: elementos y rollups

Figura 1.1: Dimensiones Tiendas y Productos

Data Warehouse (DW). Inicialmente, esta estructura de datos compacta fue presentada para computar consultas *top-k* sobre matrices de valores (Brisaboa et al., 2014a), tales como: "obtener los 5 vendedores que realizaron más ventas".

Un Data Warehouse es un repositorio histórico de datos que sirve de apoyo a la toma de decisiones (Chaudhuri y Dayal, 1997). Los DWs se organizan de acuerdo a *dimensiones* y *hechos*. Las *dimensiones* reflejan las perspectivas desde las cuales los datos pueden ser visualizados. Son modeladas como jerarquías de elementos (también llamados miembros) donde cada elemento pertenece a un nivel (o categoría) en una jerarquía (una red de niveles, llamada *esquema jerárquico*). Los hechos corresponden a datos cuantitativos (también conocidos como *medidas*) asociados a las dimensiones. Los hechos pueden ser agregados (una operación llamada *rollup*), filtrados, usualmente a través de operaciones *slice* y *dice* y referenciados usando las dimensiones, a través del proceso OLAP (On-line Analytical Processing). Un cubo de datos es una estructura multidimensional para capturar y analizar los hechos de acuerdo a las dimensiones. El Ejemplo 1.1 ilustra estos conceptos.

Ejemplo 1.1 Consideremos un DW para una empresa que tiene locales de ventas para sus productos. La Figura 1.1(a) y la Figura 1.1(c), muestran respectivamente, los esquemas jerárquicos de las dimensiones Tiendas y Productos. La dimensión Tiendas está compuesta por los niveles Tienda, Ciudad, Región y All. El nivel Tienda alcanza (o hace rollup, en terminología DW) a Ciudad, a su vez Ciudad rollup a Región, que a su vez se relaciona con el nivel All. Este último nivel es alcanzado por cada nivel inferior en la jerarquía. La Figura 1.1(b) muestra los elementos y rollups para la dimensión Tiendas. Los elementos del nivel Tienda son: {ST_1,

ST_2, ST_3, ST_4, ST_5, ST_6, ST_7, ST_8}. El nivel Ciudad tiene los elementos CHI (Chillán), CON (Concepción), CAU (Cauquenes) y TAL (Talca). Los elementos del nivel Región son: {VIII, VII}. El único elemento en All es: {all} (por definición). Como ilustración, la relación rollup entre los niveles Tienda y Ciudad de la dimensión Tiendas, contiene los pares de elementos: {(ST_1,CHI), (ST_2,CHI), (ST_3,CHI), (ST_4,CON), (ST_5,CON), (ST_6,CAU), (ST_7,TAL), (ST_8,TAL)} (ver Figura 1.1(b)).

La dimensión Productos tiene las categorías Producto, Tipo, Marca y All. La categoría Producto rollup a Tipo, que alcanza a Marca, la que a su vez rollup a All. Los elementos del nivel Producto son: {P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8}. El nivel Tipo contiene los elementos: {T_1, T_2, T_3, T_4}. Los elementos del nivel Marca son: {B_1, B_2}. A modo de ilustración, el elemento P_1 en el nivel Producto alcanza al elemento T_1 en el nivel Tipo, el cual rollup a B_1 en el nivel Marca. □

En esta tesis solo consideramos dimensiones con esquemas jerárquicos lineales, como los mostrados en la Figura 1.1. Es decir, esquemas donde existe un único camino desde el nivel inferior al nivel superior All. Sin embargo, en la literatura existen dimensiones más complejas donde el nivel inferior puede alcanzar el nivel superior All a través de más de un camino desde el nivel inferior, incluso, pueden existir varios niveles inferiores. Estos esquemas se denominan esquemas heterogéneos (Hurtado y Gutierrez, 2007). El Ejemplo 1.2 muestra una dimensión con esquema heterogéneo.

Ejemplo 1.2 La Figura 1.2 muestra el esquema jerárquico para la dimensión Artículos, la cual es heterogénea. Esta dimensión corresponde a artículos científicos y está compuesta por los niveles Artículo, Revista, Área, Tema y All. Por un camino, el nivel Artículo alcanza al nivel Revista, el cual hace rollup al nivel Área, que finalmente se relaciona con el nivel All. Por otro lado, el nivel Artículo alcanza el nivel Tema, el cual hace rollup al nivel All. □

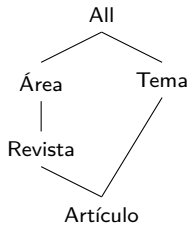


Figura 1.2: Dimensión heterogénea Artículos

En DWs las consultas más comunes son aquellas que permiten la agregación de los hechos (datos numéricos), por ejemplo, considerando el DW del Ejemplo 1.1, se puede computar la consulta *obtener el total de ventas de productos agrupados por región y año*. Este tipo de consultas se visualizan por medio de *cubos de datos* (Chaudhuri y Dayal, 1997). Un cubo de datos representa los hechos como puntos en un espacio multidimensional determinado por las dimensiones del DW. Son operaciones comunes sobre cubos de datos,

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
ST_1	1	1	1	1	2	1	3	1
ST_2	1	2	2	2	1	2	2	1
ST_3	1	0	2	2	1	2	2	1
ST_4	2	2	0	1	0	1	0	4
ST_5	1	2	1	2	1	2	2	3
ST_6	1	2	3	2	1	2	1	0
ST_7	0	3	0	4	1	2	0	1
ST_8	1	2	3	1	0	0	1	2

(a) Unidades vendidas por Tienda y Producto

	T_1	T_2	T_3	T_4
CHI	6	14	12	3
CON	7	5	5	7
CAU	3	6	3	0
TAL	6	9	3	3

(b) Unidades vendidas por Ciudad y Tipo

Figura 1.3: Cubos de datos para el DW de la Figura 1.1

el realizar *rollup*, es decir, agrupar los hechos por un nivel superior en la jerarquía de una dimensión, *drill down* que corresponde a lo opuesto al rollup, y también las operaciones de *slice* y *dice*, que permiten filtrar información de los cubos de datos. El Ejemplo 1.3 ilustra el concepto de cubo de datos.

Ejemplo 1.3 La Figura 1.3(a) muestra un cubo de datos de ventas de productos por tienda para el DW de la Figura 1.1. Por ejemplo, para la tienda ST_1 se vendió una unidad de los productos P_1, P_2, P_3, P_4, P_6 y P_8, dos unidades del producto P_5 y tres unidades del producto P_7. A partir de este cubo de datos se puede computar la consulta de agregación "*obtener las ventas agrupadas por ciudad y tipo de productos*" (entre otras), para ello se deben utilizar las relaciones rollup entre los niveles Tienda y Ciudad de la dimensión Tiendas y la relación rollup entre los niveles Producto y Tipo de la dimensión Productos. La respuesta a la consulta de agregación es el nuevo cubo de datos ilustrado en la Figura 1.3(b). □

Las consultas de agregación requieren la navegación mediante las jerarquías de las dimensiones que componen el data warehouse. Dado que los DWs pueden alcanzar grandes volúmenes de datos, debido a que almacenan datos para grandes horizontes de tiempo, obtener consultas de manera eficiente es clave en data warehousing. El enfoque más común en DWs es utilizar resultados pre-computados, denominadas tablas de resúmenes, para responder a una nueva consulta. También se pueden construir índices sobre estas tablas de resúmenes para acceder a los datos de manera más eficiente (Gupta et al., 1997; Harinarayan et al., 1996; Mumick et al., 1997). Otros trabajos han explorado métodos para comprimir los cubos y de esta manera disminuir sus volúmenes de datos (Furtado y Madeira, 2000b; Golfarelli y Rizzi, 2013; Li et al., 1999; Li y Srivastava, 2002; Wu et al., 2006; Yu y Wang, 2002). Sin embargo, al comprimir los cubos se pierde precisión al responder consultas de agregación. Un enfoque diferente es usar estructuras de datos compactas para disminuir el espacio de almacenamiento y acelerar el cómputo de consultas. En (Brisaboa et al., 2014a) se presenta la estructura de datos compacta k^2-treap para computar consultas *top-k* sobre matrices de datos numéricos y en (Brisaboa et al., 2016) se presenta la

estructura compacta CMHD (Compact representation of Multidimensional data on Hierarchical Domains) para calcular consultas de agregación con la función de agregación `SUM` también sobre matrices de datos numéricos.

1.0.1. Contribuciones

Las contribuciones de esta tesis son las siguientes:

- Implementar cubos de datos de DWs en la estructura de datos compacta k^2-treap en memoria principal.
- Implementar dimensiones de DWs en estructuras compactas bitmaps.
- Generar algoritmos para simular la navegación sobre las dimensiones usando las nuevas representaciones compactas de las dimensiones (bitmaps).
- Generar algoritmos para computar consultas de agregación con funciones `SUM` y `MAX` sobre las estructuras compactas definidas.
- Generar cubos de datos sintéticos de diferentes tamaños para realizar experimentación y demostrar la eficiencia en términos de ahorro de almacenamiento en memoria y tiempo de ejecución de consultas.

1.0.2. Organización del documento

El resto del documento se organiza de la siguiente manera. En el Capítulo 2 se presentan las hipótesis de la tesis, el objetivo general, los objetivos específicos, los alcances de la tesis y la metodología de trabajo. En el Capítulo 3 se presentan trabajos relacionados. En el Capítulo 4 se presenta el concepto de data warehouse y la estructura de datos compacta k^2-treap que será utilizada para representar cubos de datos de DWs. En el Capítulo 5 se presenta la representación de cubos de datos y dimensiones en estructuras de datos compactas. En el Capítulo 6 se presentan los algoritmos desarrollados en la tesis para computar consultas de agregación sobre DWs representados de manera compacta en memoria principal. En el Capítulo 7 se presentan los resultados de la experimentación realizada. Finalmente, en el Capítulo 8 se presentan las conclusiones de la tesis y trabajo futuro.

Capítulo 2

Propuesta de Tesis

En este capítulo se presentan las hipótesis de la tesis, junto con el objetivo general, los objetivos específicos, los alcances de la tesis y la metodología de trabajo.

2.1. Hipótesis

Las hipótesis de esta tesis son:

1. Es posible utilizar estructuras de datos compactas para representar cubos de datos y dimensiones de DWs en memoria principal y lograr ahorro en espacio de almacenamiento.

2. Es posible implementar algoritmos para computar de manera eficiente consultas de agregación sobre cubos de datos y dimensiones de DWs representados de manera compacta.

2.2. Objetivos

2.2.1. Objetivo General

El objetivo de esta tesis es utilizar estructuras de datos compactas para representar cubos de datos y dimensiones de DWs en memoria principal, permitiendo ahorro de espacio de almacenamiento y cómputo eficiente de consultas de agregación sobre los datos compactos.

2.2.2. Objetivos Específicos

1. Definir el tipo de dimensiones a considerar y el número de dimensiones en los cubos de datos.

2. Seleccionar estructuras de datos compactas ad-hoc para representar dimensiones y cubos de datos de DWs en memoria principal.

3. Definir e implementar algoritmos que permitan realizar consultas de agregación sobre cubos de datos en su forma compacta, considerando la funciones de agregación `SUM` y `MAX`.

4. Realizar experimentos que demuestren el ahorro de espacio de almacenamiento y eficiencia en el cómputo de consultas de agregación.

5. Comparar la ejecución de consultas con el método tradicional de cómputo de consultas vía sistemas de gestión de bases de datos.

2.3. Alcance de la investigación

Las representaciones y algoritmos de consulta implementados en esta tesis consideran dimensiones lineales (como las mostradas en la Figura 1.1) y no complejas en su estructura (como la mostrada en la Figura 1.2). Además, todas las dimensiones contienen un único nivel inferior y se consideran cubos de datos con dos dimensiones. Esta última restricción se debe a que la estructura de datos escogida (k^2-treap soporta matrices de dos dimensiones, ver Capítulo 4).

Para la experimentación se considerará el DW del Ejemplo 1.1. Los cubos de datos se generarán aleatoriamente a través de algoritmos que utilizan distribución normal y uniforme. Se considerarán cubos de datos con diferentes cantidades de elementos, siendo el de menor tamaño 2,500 elementos y el de mayor tamaño de 1,000,000 de elementos.

2.4. Metodología

La metodología de trabajo de esta investigación está basada en función de los objetivos específicos planteados anteriormente y considera las siguientes etapas:

1. Seleccionar estructuras de datos compactas ad-hoc para representar cubos de datos y dimensiones de DWs en memoria principal.

2. Generar un DW en las estructuras de datos compactas seleccionadas.

3. Implementar algoritmos para realizar consultas de agregación con funciones `SUM` y `MAX` sobre DWs almacenados en estructuras de datos compactas en memoria principal.

4. Diseñar los conjuntos de datos para la experimentación, lo que implica la utilización de algoritmos aleatorios para generar cubos de datos con distribución normal y uniforme.

5. Ejecutar experimentos para demostrar la eficiencia en términos de ahorro de espacio de almacenamiento y tiempos de ejecución de consultas de los algoritmos sobre la representación compacta de los DWs versus la computación de consultas (y almacenamiento de datos) en un sistema de gestión de bases de datos tradicional.

Capítulo 3

Estado del Arte

En los últimos años se está viviendo un diluvio de datos, es decir, la cantidad de datos que se están generando es enorme, por ejemplo, las tecnologías actuales, tales como teléfonos inteligentes, GPS (Global Positioning Systems), satélites, sensores, etc., hacen posible recoger y transmitir gran cantidad de datos que necesitan ser procesados de manera eficiente. Sin embargo, los procesadores comunes no tienen la capacidad necesaria para procesar terabytes de datos de manera eficiente.

Además, hay casos en que el verdadero problema no es el tamaño original de los datos, sino que las estructuras de datos que son usadas para responder eficientemente a consultas sobre estos datos, muchas veces superan considerablemente el tamaño de los éstos (Navarro, 2016). Por ejemplo, el ADN del genoma humano, el cual consta de al rededor de 3300 millones de bases, donde cada una de ellas es representada por la inicial de su nombre (A, C, G, T), tiene un tamaño original de unos 800 MB. Si no se trabaja con el índice apropiado, responder a consultas como "encontrar la auto repetición más larga" podría tomar tiempo cuadrático. Sin embargo, al utilizar una estructura de datos como el árbol de sufijos, esta consulta podría responderse en tiempo lineal. Sin embargo, el árbol de sufijos requiere al menos 10 bytes para representar cada base, requiriendo más de 30 GB de almacenamiento para representar el genoma humano. Ante esto, una alternativa para poder procesar eficientemente grandes cantidades de datos es usar *estructuras de datos compactas*, que, como ya hemos introducido, son estructuras de datos que ocupan poco espacio de almacenamiento y permiten consultar los datos en su forma compacta. Volviendo al ejemplo del genoma humano, si se utilizaran estructuras de datos compactas, éste se podría representar utilizando 2 GB de almacenamiento, lo cual permite trabajar los datos en memoria principal, mejorando la eficiencia de las consultas.

En esta línea, a continuación se presentan algunos trabajos relacionados con estructuras de datos compactas y trabajos que pretenden mejorar el procesamiento de consultas en Data Warehouses.

3.1. Estructuras de Datos Compactas

Las estructuras de datos compactas se han utilizado en diversas áreas. En (Brisaboa et al., 2009; Claude y Navarro, 2007; Navarro, 2016) se presenta la estructura de datos compacta k^2-tree que se utiliza para representar el grafo de la Web, el cual al año 2004 contenía unos 11.5 millones de nodos y 150 mil millones de links, solo considerando la web estática indexada. El grafo de la Web necesitaría unos 600 GB de RAM para ser almacenado. Utilizando estructuras de datos compactas, se podría almacenar en unos 100 GB, permitiendo realizar consultas sobre el grafo de manera eficiente. En (Brisaboa et al., 2013a; Claude y Navarro, 2009; Navarro, 2014; Navarro y Sadakane, 2014) se reportan estructuras de datos compactas para representar documentos en el contexto de recuperación de la información. En (Brisaboa et al., 2013b) se presentan estructuras de datos compactas para mejorar el procesamiento de consultas en GIS (Geographical Information Systems). En (De Bernardo et al., 2013; Ladra et al., 2017) se presentan estructuras de datos compactas para representar y consultar datos espaciales de tipo ráster (por ejemplo, datos de temperaturas).

En (Brisaboa et al., 2014a) se presenta la estructura de datos compacta k^2-treap para el cómputo de consultas *top-k* sobre matrices de dos dimensiones con datos numéricos. Una consulta *top-k* es aquella que pregunta por los k valores máximos dentro de un rango de valores numéricos. Por lo tanto, la estructura compacta k^2-treap fue diseñada para responder eficientemente este tipo de consultas y no para responder otras consultas de agregación que usen el operador `MIN`, como obtener los k menores valores dentro de un rango de valores numéricos.

En esta tesis utilizamos com base la estructura compacta k^2-treap para implementar consultas de agregación sobre DWs, restringiendo su uso a consultas con funciones de agregación `MAX` y `SUM`. Es importante mencionar, que hasta donde hemos analizado, no existen trabajos que utilicen estructuras de datos compactas para procesar consultas sobre Data Warehouses, y en este sentido, esta tesis es el primer aporte del uso de estructuras compactas sobre bases de datos. Dado nuestros resultados experimentales, que se presentan en el Capítulo 7, es altamente probable pasar de un enfoque de base de datos tradicional, es decir, datos almacenados en un Sistemas de Gestión de Bases de Datos (SGBD) a bases de datos almacenadas en memoria principal.

Recientemente en (Brisaboa et al., 2016) se presenta una nueva estructura de datos compacta llamada CMHD (Representación compacta de datos multidimensionales en dominios jerárquicos) para calcular consultas de agregación con la función de agregación `SUM`. Esta estructura, se basa en la estructura de datos compacta k^n-treap (variante del k^2-treap), que admite múltiples dimensiones.

3.2. Procesamiento Eficiente de Consultas en Data Warehouses

La eficiencia en las consultas en DWs ha sido ampliamente analizada en la literatura. En DWs es común reducir el tiempo de ejecución de las consultas mediante el uso de resultados pre-computados para responder consultas y construir índices sobre estas tablas de resumen. En (Harinarayan et al., 1996) se analiza el problema de cómo seleccionar adecuadamente los cubos de datos a materializar para mejorar la eficiencia en el cómputo de consultas. En la misma dirección, en (Gupta et al., 1997) se presentan algoritmos para seleccionar automáticamente tablas de resumen para ser materializadas. Además, los autores describen un método basado en mantener deltas de tablas de resumen para actualizar los resultados previamente materializados de manera eficiente. Por otro lado, en (Mumick et al., 1997) se analiza el problema de cómo mantener vistas agregadas de datos.

En otra dirección de investigación se ha analizado la generación compacta de cubos en sistemas OLAP. En (Golfarelli y Rizzi, 2013) los autores proponen el operador *shrink*, que fusiona rebanadas (*slices* en terminología de DW) de cubos de datos y las reemplaza con una única vista representativa. La idea detrás de esto es generar cubos de menor tamaño para la visualización de tablas pivote (modo usual de visualización de los cubos de datos en herramientas OLAP). El operador *shrink* es aplicado sobre cubos para reducir el tamaño de éstos, pero controlando la pérdida de precisión. La nueva representación es una aproximación de las rebanadas de cubos de datos. Por ejemplo, el operador *shrink* sobre un cubo de datos C trabaja de la siguiente manera: (i) las rebanadas del cubo C se particionan dentro de grupos similares (clusters). Por ejemplo, las ventas de productos se agrupan por ciudades que pertenecen (rollup) a una misma región. (ii) Se fusionan rebanadas dentro de una única rebanada representativa. Ejemplo: se fusionan todas las ventas de productos de las ciudades que pertenecen a la región del Bío-Bío. (iii) Se computa la medida representativa para esta nueva rebanada, utilizando el promedio de las medidas de las rebanadas en el grupo. Por ejemplo: el nuevo valor de la medida ventas es el promedio de ventas de productos en cada una de las rebanadas del grupo. La aplicación de este operador permite al usuario obtener un cubo de datos de menor tamaño, pero se pierde precisión en el cómputo de las medidas. Una idea similar se presenta en (Wang et al., 2002).

En (Furtado y Madeira, 2000a,b), el autor propone una estrategia de compresión que sustituye los valores del cubo de datos por otros valores aproximados que son calculados a través del método de cuantificación descrito en (Lloyd, 1982), con un grado de error permitido que es dado por el usuario. El objetivo es generar cubos de datos de menor tamaño de almacenamiento, evitando generar medidas (como las ventas) demasiado grandes. Una idea similar se propone en (Yu y Wang, 2002).

En (Li et al., 1999; Li y Srivastava, 2002; Wu et al., 2006), se presentan algoritmos que permiten realizar agregación sobre cubos de datos comprimidos por algunas de las técnicas de compresión tradicionales, tales como, eliminación de valores nulos, sustitución de patrones, entre otras. En (Ross y Srivastava, 1997), los autores proponen un algoritmo de particionado de cubo de datos, que mediante la estrategia *divide y vencerás*, divide un cubo de datos en pequeños cubos de datos (slice y dice) para poder llevar las porciones de

cubo de datos que se necesitan para responder consultas a memoria principal.

Hasta lo que hemos investigado, los únicos trabajos sobre estructuras de datos compactas para el cómputo eficiente de consultas de agregación son los trabajos presentados en (Brisaboa et al., 2014a) y (Brisaboa et al., 2016). Sin embargo, el primero no está presentado en el escenario de DWs, donde se requiere la navegación a través de las jerarquías de dimensiones. En este sentido, necesitamos además definir alguna forma compacta para los esquemas jerárquicos de las dimensiones de un DW (ver Capítulo 5). El segundo trabajo propone la estructura de datos compacta denominada CMHD (Compact representation of Multidimensional data on Hierarchical Domains) creada para trabajar sobre matrices multidimensionales y responder consultas con función `SUM`. Esta estructura de datos compacta divide recursivamente la matriz de hechos de acuerdo a los agrupamientos definidos en las jerarquías de cada dimensión. En general las submatrices son de diferente tamaño en cada nivel de partición. Esta división recursiva constituye un árbol ordenado, cuya raíz representa la matriz completa; a su vez el segundo nivel del árbol representa cada una de las submatrices obtenidas usando los agrupamientos establecidos en el primer nivel de las jerarquías de cada dimensión. Cada una de estas submatrices es considerada como un nodo de la raíz. El proceso de división se repite por cada submatriz y teniendo en cuenta los siguientes niveles de las jerarquías de las dimensiones. Si para una submatriz no existen valores, dicha submatriz no se vuelve a dividir. Paralelamente a la partición, en cada nodo se almacena el valor pre-calculado de la función de agregación SUM de la submatriz. El árbol definido previamente para CMHD es conceptual, y es representado de manera compacta usando diferentes estructuras de datos, tanto para las dimensiones como para la matriz. Para cada dimensión (usando LOUDS (Navarro, 2016)) se utilizan dos bits por cada nodo del árbol permitiendo navegar de manera eficiente a través de los diferentes niveles de la dimensión mediante las operaciones de *rank* y *select*(ver Capítulo 4, Sección 4.2.1). La topología del árbol y límites de las submatrices se representan mediante dos bitmaps que permiten navegar el árbol. Finalmente, tanto los valores de la matriz original como los valores pre-calculados se generan por medio de un recorrido por nivel del árbol representado con DACs (Brisaboa et al., 2013a). Con CMHD es posible responder consultas de agregación con la función `SUM` sobre cualquiera de las submatrices generadas por las jerarquías de las dimensiones. Mediante una serie de experimentos se muestra que CMHD con un poco más de almacenamiento procesa en menor tiempo las consultas de agregación sobre submatrices de tamaño irregular (escenario realista) que la opción que considera una partición regular de la matriz. Los experimentos no consideran el desempeño de CMHD con respecto a DWs almacenados en memoria secundaria. Tampoco se mide el desempeño considerando otras funciones de agregación, cuya inclusión puede incrementar el almacenamiento requerido por la estructura y el tiempo de las consultas.

Capítulo 4

Preliminares

En este capítulo se presentan conceptos preliminares necesarios para comprender qué es un Data Warehouse y cómo está organizado. Además, se presenta la estructura de datos compacta k^2-treap, considerando su construcción y navegación para la realización de consultas sobre ésta.

4.1. Data Warehouses

Los DWs son repositorios de datos históricos, resumidos, que se utilizan para obtener informes de gestión para apoyar la toma de decisiones. Los DWs se organizan mediante dimensiones y hechos (Chaudhuri y Dayal, 1997). Una dimensión es un concepto abstracto que permite dar contexto a los hechos o datos numéricos. Éstas se componen de niveles que se organizan de manera jerárquica (Hurtado y Gutierrez, 2007; Jensen et al., 2010) y se utilizan para seleccionar y agrupar datos de acuerdo a un determinado nivel de detalle o granularidad. Dado que existe una jerarquía entre niveles, existe una relación hijo/padre entre dos niveles que se denomina relación *rollup*. Toda dimensión tiene un nivel superior denominado All que representa el agrupamiento total y el nivel de granularidad más alto. A su vez, toda dimensión tiene uno o más niveles inferiores, que representan el nivel de agregación más bajo. En esta tesis, consideramos dimensiones con un único nivel inferior. Las instancias de una dimensión son típicamente llamadas *valores de dimensión* o *miembros de dimensión*, donde cada valor pertenece a un nivel de la dimensión (Jensen et al., 2010). Dos elementos de dos niveles de una dimensión se relacionan sí y solo sí los correspondientes niveles se relacionan en el esquema jerárquico de la dimensión.

Ejemplo 4.1 La Figura 1.1(a) muestra la dimensión Tiendas compuesta por los niveles Tienda, Ciudad, Región, y el nivel superior All. Las relaciones rollup para la dimensión Tiendas son: {(Tienda, Ciudad), (Ciudad, Región), (Región, All)}.

La Figura 1.1(b) corresponde a una instancia para la dimensión Tiendas con los siguientes elementos por niveles y relaciones rollup:

- All = {all},

- Región = {VIII, VII},
- Ciudad = {CHI, CON, CAU, TAL},
- Tienda = {ST_1, ST_2, ST_3, ST_4, ST_5, ST_6, ST_7, ST_8},
- $\mathcal{R}$(Tienda, Ciudad) = {(ST_1, CHI), (ST_2, CHI), (ST_3, CHI), (ST_4, CON), (ST_5, CON), (ST_6, CAU), (ST_7, TAL), (ST_8, TAL)},
- $\mathcal{R}$(Ciudad, Region) = {(CHI, VIII), (CON, VIII), (CAU, VII), (TAL, VII)},
- $\mathcal{R}$(Region, All) = {(VIII, all), (VII, all)}. □

Los usuarios de DWs necesitan analizar los hechos agregados a múltiples niveles de abstracción (Hurtado y Gutierrez, 2007; Jensen et al., 2010). Un *cubo de datos* es una estructura multidimensional para capturar y analizar datos. Un cubo de datos generaliza la representación tabular de dos dimensiones, dando lugar a la posibilidad de analizar múltiples dimensiones. Un cubo de datos consiste de celdas, las celdas no vacías se denominan hechos. Un hecho puede tener *medidas* asociadas, que corresponden a valores numéricos. La Figura 1.3 muestra dos cubos de datos a diferentes niveles de granularidad. En ambos cubos de datos la medida es la cantidad de ventas realizadas. Generalmente, solo se materializan los *cubos de datos base*, es decir, aquellos que asocian hechos a niveles inferiores de dimensiones (como el cubo de ventas en la Figura 1.3(a)), el resto de los cubos se obtienen a partir de cubos base y la navegación a través de las jerarquías de las dimensiones (como el cubo de ventas en la Figura 1.3(b).

4.2. k^2-treaps

La estructura de datos compacta k^2-treap es presentada en (Brisaboa et al., 2014a) y está basada en dos estructuras de datos compactas, el k^2-tree (Brisaboa et al., 2009) y el treap (Seidel y Aragon, 1996). El k^2-treap fue creado para representar matrices bidimensionales que almacenan valores numéricos, y diseñado para resolver, de manera eficiente, consultas de tipo *top-k* sobre él, las cuales obtienen los k mayores resultados contenidos en la matriz. Las consultas *top-k* más simples son aquellas que involucran a la totalidad de los datos en la matriz, sin embargo, existen consultas que requieren ser aplicadas sobre un subconjunto de esos datos. Estas últimas consultas se denominan consultas de rango.

4.2.1. Construcción del k^2-treap

Sea $M[n \times n]$ una matriz donde una celda puede contener un valor o estar vacía. Esta matriz puede ser particionada en k^2 submatrices, pudiéndose obtener un árbol como se indica a continuación: (i) la raíz del árbol almacena el máximo valor de la matriz y las coordenadas de la celda que lo contiene. (ii) Luego, el valor es eliminado de la matriz. Si varias celdas comparten el mismo valor, una de ellas es elegida. (iii) Luego, la matriz es descompuesta en k^2 submatrices de igual tamaño, y k^2 nodos hijos son añadidos a la

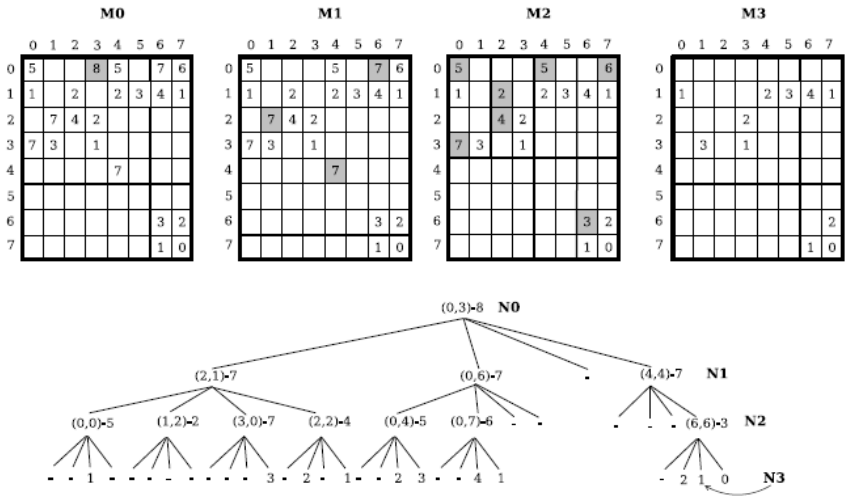


Figura 4.1: Construcción de un k^2-treap a partir de una matriz $M[8 \times 8]$(Brisaboa et al., 2014a)

raíz del árbol, cada uno de los cuales representa una de las submatrices. Este proceso es repetido recursivamente para cada hijo (submatriz), hasta que la matriz esté vacía. El siguiente ejemplo ilustra este proceso.

Ejemplo 4.2 Considerar la matriz $M0[8 \times 8]$ en la Figura 4.1. El valor máximo en la matriz está ubicado en la coordenada $(0, 3)$ y corresponde al valor 8. Esta coordenada y este valor componen la raíz del árbol. Luego, el valor es borrado de $M0$. Después, $M0$ es dividida en k^2 submatrices. Considerar $k = 2$, entonces se obtienen 4 submatrices en $M1$ (ver Figura 4.1). Este proceso es aplicado recursivamente sobre las submatrices, cada cuadrante es un hijo de la raíz. Para $M0$ hay otras dos subdivisiones llamadas $M2$ y $M3$, respectivamente. Los nodos vacíos en el árbol son representados con el símbolo "$-$". □

El árbol luego es modificado para ahorrar espacio de almacenamiento. Esto es posible al cambiar las coordenadas por las posiciones en la correspondiente submatriz, de izquierda a derecha (comenzando cada submatriz en la posición $(0, 0)$), y siendo cada valor re-codificado diferencialmente con respecto a su nodo padre. La Figura 4.2 muestra el nuevo árbol para el árbol de la Figura 4.1.

Luego, la estructura del árbol del k^2-treap es almacena en un k^2-tree (Brisaboa et al., 2009), y los valores y coordenadas del árbol son almacenados en arreglos específicos. Básicamente, un k^2-tree almacena 1s y 0s, un 1 representa la presencia de un valor, y un 0 representa la ausencia de éste. En términos computacionales, no es necesario representar la estructura del árbol, ya que se utiliza un bitmap llamado T que almacena 1s y 0s. Este arreglo puede ser explorado usando las operaciones *rank* y *select*. Sea $B[1, n]$ una secuencia de bits (también llamada bitmaps). La operación $rank_1(B, i)$ retorna el número de ocurrencias de 1s en $B[1, i]$. La operación $select_1(B, j)$ retorna la posición de la j-ésima ocurrencia

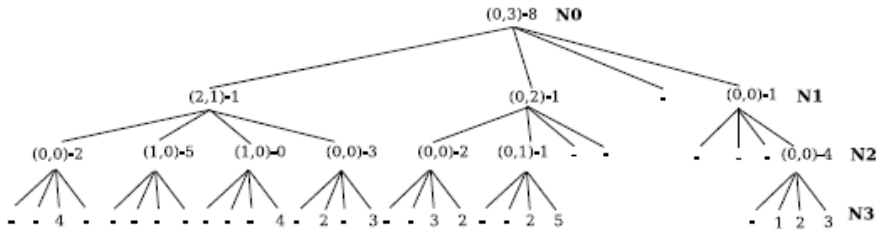


Figura 4.2: Nuevo árbol re-codificado para el árbol de la Figura 4.1

de 1 en B. Ambas operaciones pueden ser resueltas en tiempo constante. Para encontrar los hijos de un nodo x en un k^2-tree se utiliza la función $child_i(x) = (rank_1(T, x) \times k^2) + i$, donde i es el i-ésimo hijo del nodo x en T (donde la primera posición en T es 0). Por ejemplo, considere el k^2-tree de la Figura 4.3, para encontrar la posición de los hijos del segundo hijo de la raíz, se aplica la función $child_0(2) = (rank_1(T, 1) \times k^2) + 1 = (2 \times 4) + 0 = 8$, es decir, los hijos del segundo nodo parten en la posición 8 del arreglo T.

Los otros arreglos utilizados para representar la estructura de datos compacta k^2-treap son: (i) arreglos para almacenar las coordenadas del árbol modificado, un arreglo llamado `values` para almacenar los valores del árbol, y un arreglo llamado `first` que indica la posición inicial de un nuevo nivel en el arreglo `values`. La Figura 4.3 muestra la representación del k^2-treap de la Figura 4.2.

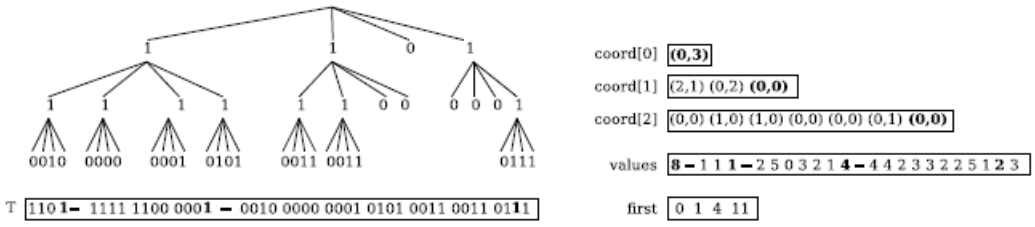


Figura 4.3: Representación del k^2-treap para la matriz $M0$ de la Figura 4.1

4.2.2. Navegación a través del árbol

Supongamos que se desea obtener la coordenada $C(x, y)$ en el k^2-treap de la Figura 4.3. La navegación comienza por la raíz del k^2-tree, que corresponde al nodo con coordenadas (x_0, y_0) y valor $v = values[0]$. Si $C = (x_0, y_0)$, el valor v es retornado, de lo contrario es necesario encontrar el cuadrante donde la celda estaría ubicada en el k^2-tree. Sea p una posición de un nodo en el array T (que almacena el k^2-tree), si $T[p] = 0$ la correspondiente submatriz está vacía y el algoritmo retorna. Por otro lado, es necesario encontrar las coordenadas y el valor del nodo. Así, se computa $r = rank_1(T, p)$ el cual retorna el número de ocurrencias de 1 en T hasta la posición p, el valor del nodo es $v = values[r]$ y

sus coordenadas con $coord[l][r - first[l]]$, donde l es el nivel actual en el árbol. El valor v debe ser reconstruido desde el arreglo `values` donde los valores son almacenados diferencialmente con respecto al valor de sus nodos padre. Así, $v_1 = v - values[r]$. Asumiendo que las coordenadas en $coord[l][r - first[l]]$ son (x_1, y_1), si $C = (x_1, y_1)$, se retorna v_1, o en caso contrario nuevamente, el cuadrante correcto donde C está localizado debe ser encontrado, recursivamente.

4.2.3. Cómputo de consultas top-k a partir de un k^2-treap

El proceso para obtener respuestas a consultas *top-k* sobre un rango de la matriz $Q[x_1, x_2] \times [y_1, y_2]$ comienza desde la raíz del árbol mediante la implementación de una cola de prioridad que almacena los valores de la matriz. El primer elemento de esta cola es el valor raíz, y luego, iterativamente, el proceso extrae el primer elemento de la cola, si la coordenada del valor está dentro de Q, una nueva respuesta es recuperada. Todos los hijos del nodo extraído, cuyas coordenadas intersectan Q son insertados en la cola. El proceso continúa hasta que las k respuestas son obtenidas. Este proceso se ilustra en el siguiente ejemplo.

Ejemplo 4.3 Considere la matriz $M0[8 \times 8]$ de la Figura 4.1, y la consulta "obtener los tres valores más altos en el rango $[4, 4], [7, 7]$", esto es, el cuarto cuadrante de $M0$. El primer elemento de la cola de prioridad es la raíz del árbol, esto es, la coordenada $(0, 3)$ con valor 8. Ya que las coordenadas de este nodo no están en el rango Q, este valor es descartado como respuesta. Así, todos los hijos de la raíz cuyas coordenadas intersectan el rango de consulta son insertados en la cola, esto es, el cuarto hijo en el nivel $N1$ de la Figura 4.1, con coordenada $(4, 4)$ y valor 7, el cual corresponde a la primera respuesta. Luego, los hijos no vacíos de este nodo son añadidos a la cola, esto es, el nodo con coordenada $(6, 6)$ y valor 3, el cual corresponde a la segunda respuesta. Finalmente, ya que aun se necesita una tercera respuesta, los hijos no vacíos de este nodo son agregados a la cola de prioridad, esto es, los nodos con coordenadas $(6, 7)$, $(7, 6)$ y $(7, 7)$, los cuales son todos hojas del árbol, con valores $2, 1$ y 0 respectivamente. Así, la tercera respuesta corresponde al valor 2, entonces, las respuestas a esta consulta *top-3* son $\{7, 3, 2\}$. □

Capítulo 5

Estructuras de Datos Compactas para Data Warehouses

En este capítulo se presenta el modo de representar de forma compacta dimensiones y cubos de datos de DWs, para ahorrar espacio de almacenamiento y poder trabajar en memoria principal.

5.1. Representación de Dimensiones de un DW

Para representar las dimensiones de un DW de manera compacta, trabajamos con los niveles y las relaciones rollup por separado. Los elementos de cada nivel de las dimensiones se almacenan en arreglos. Esto es necesario para todos los niveles de una dimensión, excepto para el nivel inferior, cuyos valores están en el cubo de datos, y el nivel superior, ya que en este nivel sólo existe el elemento all.

Ejemplo 5.1 Considere las dimensiones Tiendas y Productos de la Figura 1.1. Las tablas de la Figura 5.1 almacenan los elementos de los niveles correspondientes, en donde Ciudad y Región pertenecen a la dimensión Tiendas y Tipo y Marca pertenecen a la dimensión Productos. □

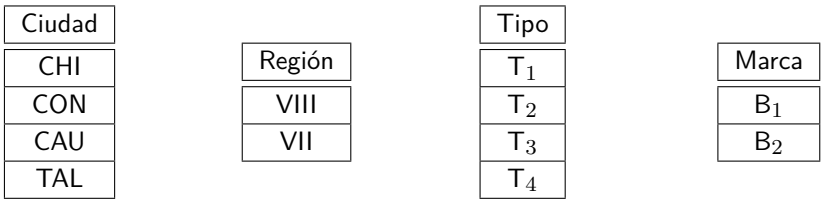


Figura 5.1: Tablas de una columna para almacenar los elementos de cada nivel de las dimensiones

Para representar las relaciones rollup que se dan entre los elementos de una dimensión, se utilizan secuencias de bits, también llamados bitmaps, de tal manera que se permita saber cuántos elementos de un nivel alcanzan a un determinado elemento del nivel superior. Partiendo del nivel inferior, se construye un bitmap por cada nivel presente en la dimensión, con tantas posiciones como elementos existan en el respectivo nivel. En la primera posición del bitmap, correspondiente al primer elemento del nivel a representar, se le asigna un 1 para indicar que el elemento hace rollup con el primer elemento de la categoría del nivel superior. Luego, se asigna un 0 a todos los elementos siguientes que hagan rollup al mismo elemento del nivel superior, volviendo a poner un 1 cada vez que el elemento actual haga rollup a un nuevo elemento del nivel superior.

Para las dimensiones Tiendas y Productos de la Figura 1.1, se generan las tablas de la Figura 5.2.

R_1		
Posición	Tienda	Bitmap
0	ST_1	1
1	ST_2	0
2	ST_3	0
3	ST_4	1
4	ST_5	0
5	ST_6	1
6	ST_7	1
7	ST_8	0

R_2		
Posición	Ciudad	Bitmap
0	CHI	1
1	CON	0
2	CAU	1
3	TAL	0

(a) Bitmaps para dimensión Tiendas

R_3		
Posición	Producto	Bitmap
0	P_1	1
1	P_2	0
2	P_3	1
3	P_4	0
4	P_5	0
5	P_6	1
6	P_7	0
7	P_8	1

R_4		
Posición	Tipo	Bitmap
0	T_1	1
1	T_2	1
2	T_3	0
3	T_4	0

(b) Bitmaps para dimensión Productos

Figura 5.2: Bitmaps para almacenar las relaciones rollup de un DW

Estas tablas entregan información sobre las relaciones rollup entre elementos de un

nivel C_i con elementos del nivel C_j, tal que C_i rollup a C_j en el esquema jerárquico. Por ejemplo, el significado de la tabla R_1 es que los elementos ST_1, ST_2 y ST_3 rollup al primer elemento en la tabla R_2, que almacena los elementos del nivel Ciudad. Luego, el valor "1" en la columna *bitmap* que aparece en la fila correspondiente al elemento ST_4 nos indica que el elemento ST_4 rollup al segundo elemento en la tabla R_2, el cual corresponde al elemento CON; el valor '0" en ST_5 nos indica que el elemento ST_5 también hace rollup con el elemento CON. Luego, el elemento ST_6 rollup a CAU y los elementos ST_7 y ST_8 rollup al elemento TAL del nivel Ciudad.

5.2. Representación de Cubos de Datos

Para representar el cubo de datos de forma compacta, se utiliza el k^2-treap (descrito en el Capítulo 4.2). Dado que consideramos dos dimensiones por cubo de datos, el cubo de datos es una matriz de valores, que es la entrada que necesita la estructura compacta k^2-treap.

Sea N el número de elementos del nivel inferior de una dimensión A, y M el número de elementos del nivel inferior de una dimensión B, de tal manera que se forma un cubo de datos a partir de A y B; se tiene una matriz C de $N \times M$ elementos, los que corresponden a los hechos del cubo de datos. Dado esto, el tamaño de la matriz del cubo de datos depende únicamente del número de elementos existentes en los niveles inferiores de las dimensiones que la forman, sin importar cuántos elementos existan en los niveles superiores.

Los valores contenidos en una tabla de hechos de un DW, dado que las medidas representan cantidades, pueden tomar cualquier valor positivo o cero. Como el cero representa una no ocurrencia de un hecho entre un elemento de una dimensión y un elemento de otra, estos no serán considerados en la construcción del k^2-treap, con el objetivo de utilizar menos espacio de almacenamiento.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
ST_1	0	15	7	5	0	0	0	0
ST_2	3	9	10	1	0	0	0	0
ST_3	0	4	0	8	0	0	0	0
ST_4	13	0	0	3	0	0	0	0
ST_5	0	0	4	0	0	0	8	0
ST_6	0	0	7	12	0	0	0	0
ST_7	4	1	0	0	0	0	3	0
ST_8	5	6	0	0	0	0	7	10

(a) Ventas por Tienda y Producto

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
ST_1		15	7	5				
ST_2	3	9	10	1				
ST_3		4		8				
ST_4	13			3				
ST_5			4				8	
ST_6			7	12				
ST_7	4	1					3	
ST_8	5	6					7	10

(b) Ventas por Tienda y Producto, sin considerar ceros

Figura 5.3: Cubo de datos para el DW de la Figura 1.1

Ejemplo 5.2 La Figura 5.3(a) representa una matriz del cubo de datos para las dimensiones Tiendas y Productos de la Figura 1.1. La matriz debe entenderse como las ventas que

se realizaron de un producto P en una tienda ST. La Figura 5.3(b) corresponde al mismo cubo de datos, considerando solo los valores para los que hubo ventas. □

La matriz del cubo de datos se carga en el k^2-treap de acuerdo al procedimiento descrito en la Sección 4.2.1. El Ejemplo 5.3 muestra este procedimiento para el cubo de datos representado por la matriz de la Figura 5.3(b).

Ejemplo 5.3 Consideremos la matriz de la Figura 5.3(b), la cual pasa a ser la matriz M0 en la Figura 5.4, se elige el mayor valor contenido en ella y se ubica, junto a su coordenada, como nodo raíz del árbol. En este caso, el valor es 15 y su coordenada (0,1). Este valor se borra de la matriz M0 y da origen a la matriz M1, la cual está dividida en cuatro submatrices. El valor máximo de cada submatriz se almacena junto a su coordenada en nuevos nodos que pasan a ser hijos del nodo que representa a la matriz que los generó. Repitiendo estos pasos hasta que la subdivisión de matrices genere submatrices de uno o cero elementos, se obtienen las matrices M2 y M3 de la Figura 5.4. □

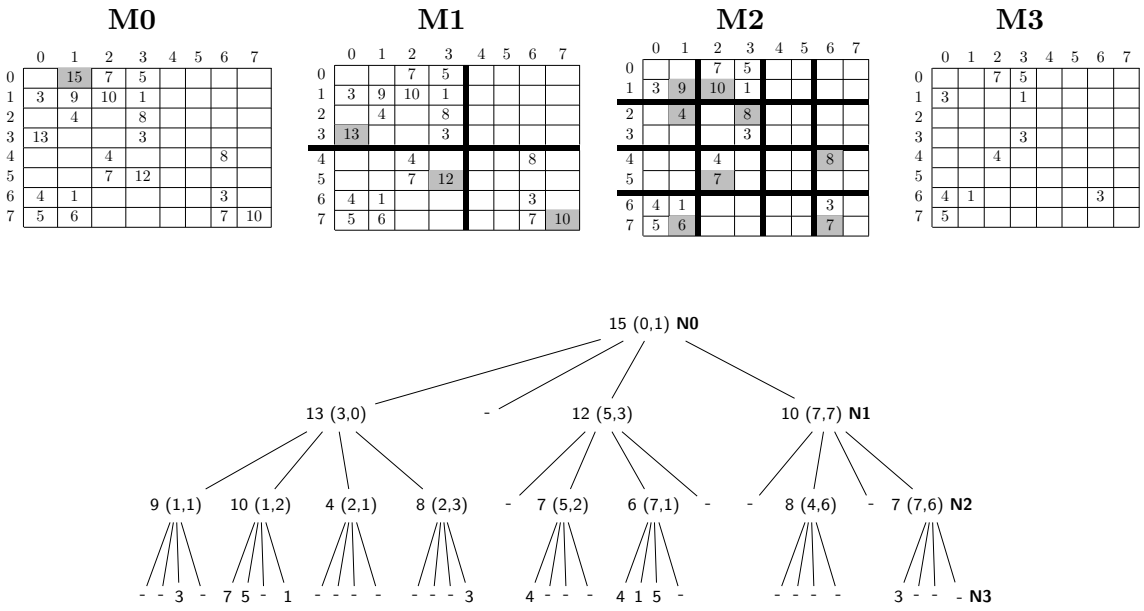


Figura 5.4: Construcción del k^2-treap para el cubo de datos de la matriz $M0$

Los valores de los nodos del árbol obtenido a partir de la matriz son re-codificados en valores más pequeños, representables con menos espacio de almacenamiento. El valor de cada nodo es codificado de forma diferencial con respecto a su nodo padre (el valor de cada nodo corresponde al valor de su padre, menos el valor propio, así, para el primer hijo del nodo raíz, se tiene 15-13=2, el que pasa a ser su nuevo valor). Del mismo modo, las coordenadas de cada nodo son re-codificadas con respecto a las coordenadas locales

dentro de la submatriz que el nodo representa. La Figura 5.5 muestra el resultado de esta operación para el árbol de la Figura 5.4.

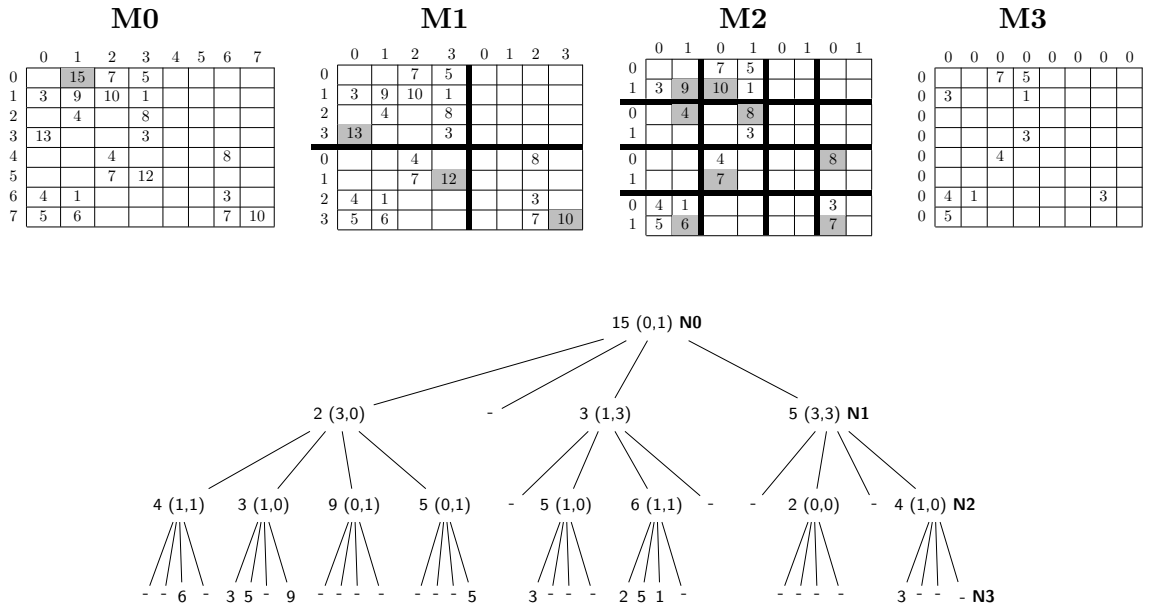


Figura 5.5: Re-codificación del árbol de la Figura 5.4

El árbol de la Figura 5.5 se almacena en un k^2-treap, según lo indicado en la Sección 4.2.1, dando como resultado las estructuras que se muestran en la Figura 5.6.

El bitmap T y los arreglos *coord*, *values* y $first$ corresponden a la estructura compacta k^2-treap, la cual representa al cubo de datos original y se almacena en memoria principal para poder procesar las consultas de agregación sobre este cubo de datos.

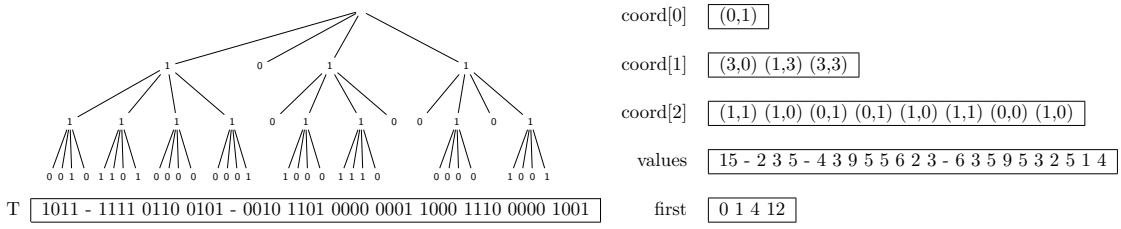


Figura 5.6: k^2-treap para el cubo de datos de la Figura 5.3

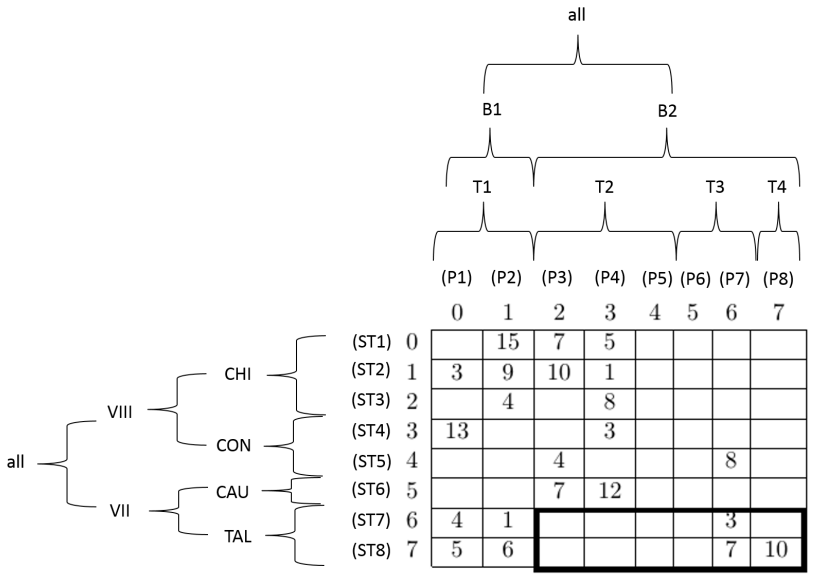
Capítulo 6

Algoritmos para Computar Consultas de Agregación en DWs

Para computar consultas de agregación sobre un DW representado mediante un k^2-treap, se emplean las consultas de rango disponibles para el k^2-treap, las que permiten reportar los k elementos mayores que se encuentran contenidos en un rectángulo de consulta. Las consultas más comunes en DWs son aquellas que utilizan la función de agregación `SUM`, agrupando por diferentes niveles de las dimensiones, por ejemplo, "obtener el total de ventas agrupadas por ciudad y por tipo de producto". Así, se puede usar el mismo algoritmo para computar las consultas *top-k* presentadas en (Brisaboa et al., 2014a) y explicadas en la Sección 4.2, pero en lugar de recuperar los *top-k* valores, se recuperan todos los valores contenidos en el rango de consulta (dado por los niveles en la consulta) y se suman. Para obtener los niveles, es necesario el uso de los bitmaps descritos en el Capítulo 5.1. El siguiente ejemplo ilustra el proceso de computar consultas de agregación.

Ejemplo 6.1 Considerar el cubo de datos de la Figura 6.1 con el nivel Tienda en las filas y Producto en las columnas, junto a los esquemas jerárquicos en la Figura 1.1. Para computar la consulta de agregación "obtener la cantidad de ventas por tiendas en la ciudad TAL y marca B_2", es necesario sumar los valores en el rango {(6,2), (7,7)} en el cubo. El resultado es 20. □

Para obtener el rango de una consulta de agregación es necesario identificar los niveles involucrados en la consulta, y así, ir bajando a través de las jerarquías correspondientes para obtener los niveles inferiores en el cubo. Esta operación es llamada drill-down en terminología de DW (Chaudhuri y Dayal, 1997). En el caso del Ejemplo 6.1 se sabe que TAL es el último elemento del nivel Ciudad, el cual es almacenado en la tabla Ciudad de la Figura 5.1. Así, es necesario conocer cuáles son las tiendas que alcanzan a TAL. Esta información puede ser obtenida buscando el cuarto 1 en el bitmap R_1 de la Figura 5.2, utilizando la operación $select_1(R_1, 4) = 6$, lo que significa que la primera tienda que alcanza a TAL es ST_7. Ya que TAL es el último elemento en el nivel Ciudad, el resto de los elementos en el bitmap R_1 hacen rollup a TAL, con lo cual se puede concluir que el rango de la



	0	1	2	3	4	5	6	7
(ST1) 0		15	7	5				
(ST2) 1	3	9	10	1				
(ST3) 2		4		8				
(ST4) 3	13			3				
(ST5) 4			4				8	
(ST6) 5			7	12				
(ST7) 6	4	1					3	
(ST8) 7	5	6					7	10

Figura 6.1: Rango para la consulta de agregación del Ejemplo 6.1

consulta considera las filas desde la 6 hasta la 7 en el cubo. Entonces, para encontrar las columnas que representan a los productos que hacen rollup a B_2, se busca en el bitmap R_4 utilizando $select_1(R_4, 2) = 1$ (ya que la marca B_2 es el segundo elemento en la tabla Marca). Entonces, el primer tipo que hace rollup a la marca B_2 es T_2 y el último es T_4. Ahora, es necesario saber qué productos hacen rollup a los tipos T_2, T_3 y T_4, mediante el uso de operaciones select sobre el bitmap R_3.Ya que el tipo T_2 es el segundo elemento en la tabla Tipo se utiliza $select_1(R_3, 2) = 2$, lo cual corresponde al producto P_3. Para encontrar el último elemento que hace rollup al tipo T_2 se usa $select_1(R_3, 3) - 1 = 4$, lo cual corresponde al producto P_5. Las mismas operaciones son realizadas para los tipos T_3 y T_4, obteniendo que se deben considerar las columnas desde la 2 hasta la 7 en el cubo. Lo anterior entrega el rango {(6,2), (7,7)} para la consulta de agregación (ver Figura 6.1). El rango de la consulta es obtenido por el Algoritmo 2, el cual es llamado por el Algoritmo 1 (Línea 3).

Los siguientes algoritmos permiten computar consultas de agregación utilizando las funciones `SUM` y `MAX`. Estos algoritmos utilizan la función *top-k* desarrollada en la librería C++ SDSL (Succinct Data Structure Library)[1] que incluye todas las funcionalidades de la estructura de datos compacta k^2-treap.

El Algoritmo 1 es el algoritmo general para computar consultas de agregación sobre un k^2-treap. Recibe como entrada la consulta de agregación Q, el conjunto de bitmaps B, el conjunto de arreglos T que almacenan los elementos de los niveles de cada dimensión, y el k^2-treap K. En primer lugar, el algoritmo obtiene los niveles para las agregaciones (Líneas

[1] https://github.com/simongog/sdsl-lite

Algoritmo 1: Cómputo de consultas de agregación sobre DWs compactos

Input: Una consulta Q, un conjunto de bitmaps B, un conjunto de arreglos T, un k^2-treap $K2T$
Output: $Res(Q)$

```
1 catDim1 ← ObtenreCategoríaDim1(Q);
2 catDim2 ← ObtenerCategoríaDim2(Q);
3 RQ ← ObtenerRangoConsulta(catDim1, catDim2, B, T);
4 Res(Q) ← ComputarAgregación(RQ, K2T);
5 return Res(Q)
```

1-2). Luego, obtiene el rango de la consulta (Línea 3), y finalmente computa la `SUM` de los valores retornados en el rango sobre el k^2-treap (Línea 4).

El Algoritmo 2 obtiene el rango de la consulta, para lo cual requiere de la consulta Q, el conjunto de B, y el conjunto de arreglos T.

Algoritmo 2: Obtener Rango Consulta

Input: Niveles $catDim1$ y $catDim2$, un conjunto de bitmaps B, un conjunto de arreglos T
Output: RQ

```
1 x1 ← ObtenerFilaInicial(catDim1, B, T);
2 x2 ← ObtenerFilaFinal(catDim1, B, T);
3 y1 ← ObtenerColumnaInicial(catDim2, B, T);
4 y2 ← ObtenerColumnaFinal(catDim2, B, T);
5 RQ ← {(x1, y1), (x2, y2)};
6 return RQ
```

Finalmente, el Algoritmo 3 computa consultas de agregación con función `SUM`. El procedimiento de cómputo comienza con la obtención de los valores contenidos en el rango de la consulta RQ (Línea 2), para lo cual usa la función *topk* implementada en la librería SDSL. Así, cuando los valores son obtenidos, el algoritmo los suma, generando la respuesta a la consulta (Línea 3).

Algoritmo 3: Computar Agregación (SUM)

Input: Rango RQ, un k^2-treap $K2T$
Output: $Res(Q)$

```
1 Res(Q) ← ∅;
2 Valores ← topk(K2T, RQ);
3 foreach i ∈ Valores do
4     Res(Q) ← Res(Q) + i.valor;
5 return Res(Q)
```

De igual manera, el Algoritmo 4 computa consultas de agregación con función `MAX`. El procedimiento de cómputo es similar al cómputo de consultas de agregación con función `SUM`, ya que se utiliza la función *topk* para obtener los valores contenidos en el rango RQ (Línea 1) y se retorna el primer valor de la lista resultante, que está ordenada de mayor a menor (Línea 2).

Algoritmo 4: Computar Agregación (MAX)

Input: Rango RQ, un k^2-treap $K2T$
Output: $Res(Q)$
1 $Valores \leftarrow$ **topk**(**K2T**, **RQ**);
2 $Res(Q) \leftarrow Valores[0]$;
3 **return** $Res(Q)$

Capítulo 7

Experimentación

En este capítulo se presentan los resultados experimentales de nuestros algoritmos en términos de uso de espacio y del tiempo de ejecución de las consultas.

7.1. Descripción del Escenario de Experimentación

Se consideró el DW presentado en el Ejemplo 1.1 con las dimensiones en la Figura 1.1 y el cubo de datos de la Figura 1.3. El DW fue implementado en el SGBD PostgreSQL versión 9.6, usando un esquema de copo de nieve (Chaudhuri y Dayal, 1997), el cual permite la representación de las jerarquías de las dimensiones. La Figura 7.1 muestra el esquema para este DW. Se consideraron diferentes conjuntos de datos sintéticos.

Los experimentos se ejecutaron en un computador con procesador Intel Core i7 de 2.40 GHz, con 12 GB de memoria RAM, sistema operativo Linux Debian 8.4.0 amd64. Todas las estructuras de datos fueron implementadas en C++ y compiladas con gcc 6.3.

Se construyó diferentes cubos de datos (matrices) con 2,500; 10,000; 250,000 y 1,000,000 elementos, siendo de 50×50, 100×100, 500×500 y 1000×1000, respectivamente. Los valores en las matrices fueron generados aleatoriamente, considerando dos distribuciones de probabilidad[1]:

- **Distribución Uniforme Discreta (DU):** *En C++ uniform_int_distribution<>*. Permite generar un número finito de valores con la misma probabilidad, es decir, se intentará que la frecuencia de aparición de todos los valores sea la misma. Para el caso particular de la experimentación, se crearon cubos de datos con valores entre 0 y 10,000,000 distribuidos de forma uniforme.
- **Distribución Normal (DN):** *En C++ normal_distribution<>*. También llamada *distribución gaussiana*, permite generar valores aleatorios cuya mayor frecuencia se concentre en la media (definida en el programa) y que se distancien de ella de acuerdo a la varianza (cuyo valor se define en el programa). A mayor varianza, más

[1]La implementación de ambas distribuciones de probabilidad está disponible en la librería *random.h* para C++

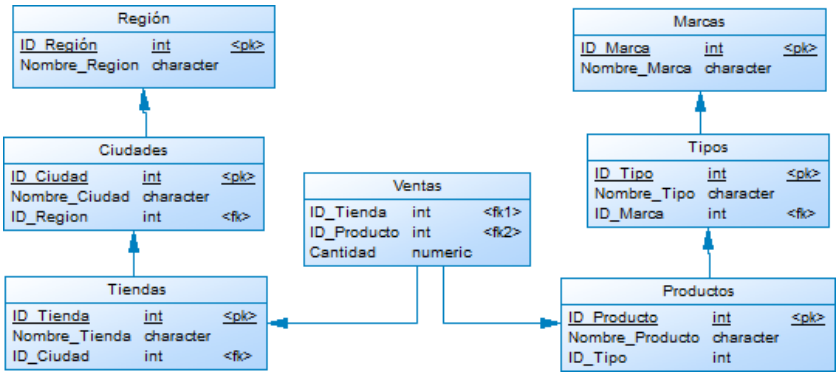


Figura 7.1: Esquema de copo de nieve para el DW del Ejemplo 1.1

separados se encontrarán los valores y habrá menos valores cercanos a la media. Para los experimentos se generaron cubos de datos, con distribución normal, con datos concentrados en dos valores, la mitad de los datos se generaron con $media = 0$ y la otra mitad con $media = 1,000,000$, ambos con $varianza = 10$.

7.2. Resultados en Almacenamiento Utilizado en Memoria Principal

La Tabla 7.1 muestra qué cubos de datos fueron generados, con distribución uniforme y normal, considerando diferentes cantidades de elementos. Además, la tabla muestra el espacio de almacenamiento ocupado por PostgreSQL y por la estructura de datos compacta k^2-treap. La Figura 7.2 muestra que la representación compacta del cubo de datos ahorra una cantidad considerable de espacio, en la mayoría de los casos el ahorro de espacio es al rededor del 80 % con respecto a PostgreSQL. Como puede verse en la Figura 7.2 con una distribución normal de los datos el ($k^2 - treap$ (n)) se ahorra más espacio que al usar distribución uniforme ($k^2 - treap$ (u)). Esto se debe, en primer lugar, a la cantidad de ceros que fueron generados en los cubos con distribución normal, los cuales no son considerados en la representación compacta, pero sí son almacenados en el SGBD. Además, los valores de los cubos generados con distribución normal tienen más alta probabilidad de ser similares.

	Distribución uniforme			Distribución normal		
#Elementos	#Ceros	SGBD (KB)	k^2-treap (KB)	#Ceros	SGBD (KB)	k^2-treap (KB)
2,500 (50×50)	0	6,954	9.5	99	6,954	5.37
10,000 (100×100)	0	7,130	36.9	426	7,130	20.31
250,000 (500×500)	0	17,000	910.92	9,563	17,000	499.99
1,000,000 (1000×1000)	0	49,000	3,637.93	38,348	49,000	1,995.35

Tabla 7.1: Tamaño de los cubos de datos generados con distribución uniforme y normal

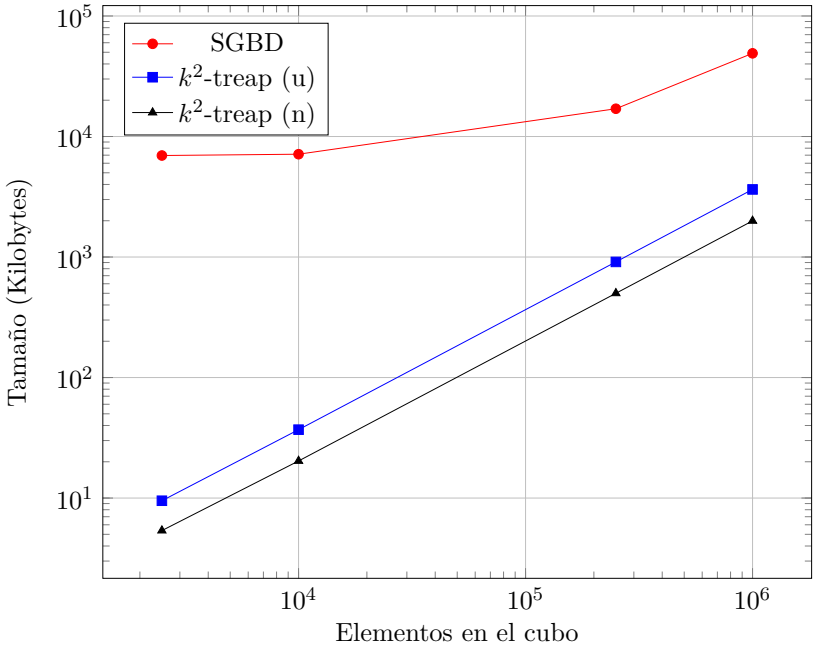


Figura 7.2: Espacio ahorrado en cubos generados con distribución normal y uniforme

7.3. Resultados en Tiempos de Ejecución de Consultas

La Tabla 7.2 y la Tabla 7.3 muestran, respectivamente, los tiempos de ejecución de todas las consultas de agregación `SUM` ejecutadas sobre cubos generados con distribución uniforme (normal) para el DW en el Ejemplo 1.1. Como puede observarse en la mayoría de los casos, se obtienen mejores tiempos de ejecución que PostgreSQL, salvo casos puntuales. Pero, cuando el k^2-treap gana en tiempo de ejecución, las diferencias son considerables.

Se muestran los mejores tiempos de ejecución sobre los cubos generados con distribución normal, ya que, en estos cubos hay ceros, los cuales son almacenados en el SGBD pero ignorados en la estructura de datos compacta. La Figura 7.3(a) muestra los tiempos de ejecución para consultas de agregación agrupadas por niveles Producto y Tienda, y la Figura 7.3(b) muestra los tiempos de ejecución para consultas de agregación agrupadas por niveles Producto y Ciudad. Las Figuras 7.4, 7.5, 7.6, y la Figura 7.7(a) muestran la misma tendencia. Sin embargo, la Figura 7.7(b) muestra que al ejecutar la consulta de agregación "obtener la cantidad total de ventas" (esto es agrupando los niveles All de ambas dimensiones) PostgreSQL tiene mejores tiempos de ejecución en cubos de datos con más de $10,000$ elementos. Esto puede explicarse por el uso de índices sobre las claves primarias y foráneas y los métodos de optimización implementados en PostgreSQL.

La Tabla 7.4 y la Tabla 7.5 muestran, respectivamente, los tiempos de ejecución de todas las consultas de agregación `MAX` ejecutadas sobre cubos generados con distribución uniforme y normal para el DW en el Ejemplo 1.1. Como puede observarse en todos los

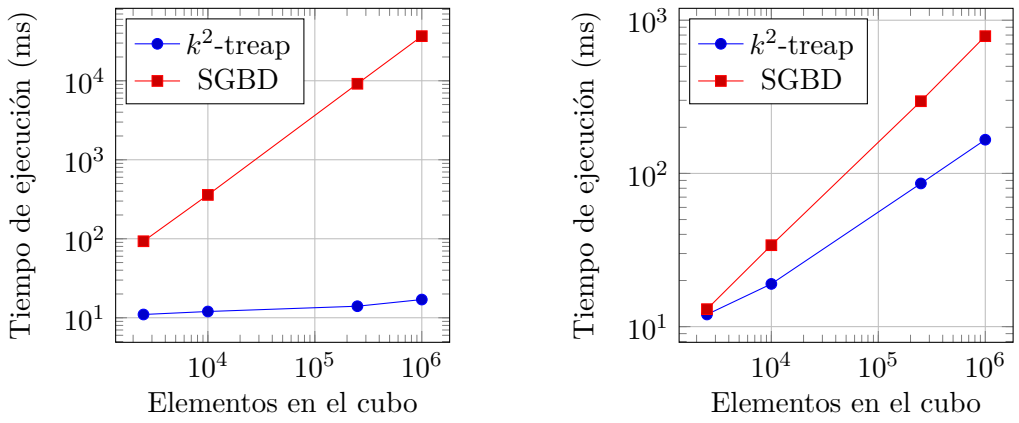
Nivel de la consulta	# Elementos en los cubos							
	2,500		10,000		250,000		1,000,000	
	k^2-treap	SGBD	k^2-treap	SGBD	k^2-treap	SGBD	k^2-treap	SGBD
Producto - Tienda	12	103	14	1,893	17	10,094	20	41,079
Producto - Ciudad	12	13	20	31	89	306	171	788
Producto - Región	9	11	18	92	85	2,750	169	16,636
Producto - All	8	12	17	22	85	223	171	646
Tipo - Tienda	12	13	18	33	84	395	173	1,160
Tipo - Ciudad	15	12	34	23	722	223	2,950	635
Tipo - Región	12	13	32	51	709	2,186	2,955	11,755
Tipo - All	10	13	29	22	721	222	3,112	605
Marca - Tienda	9	12	15	22	81	354	172	1,624
Marca - Ciudad	12	12	29	21	717	233	3,006	635
Marca - Región	9	13	28	41	705	1,451	3,012	7,858
Marca - All	8	12	25	23	724	214	3,159	585
All - Tienda	7	13	15	23	80	223	174	626
All - Ciudad	10	13	29	21	718	212	3,137	576
All - Región	7	12	28	23	708	223	3,119	615
All - All	6	13	26	21	728	182	3,240	474

Tabla 7.2: Tiempos de ejecución en milisegundos en consultas `SUM` para los cubos de datos generados con distribución uniforme

casos, se obtienen mejores tiempos de ejecución que PostgreSQL, una ventaja considerable con respecto a las consultas `SUM`. La ventaja obtenida en las consultas `MAX`, es gracias a la estructura del k^2-treap, que al ejecutar consultas *top-k* retorna los valores ordenados de mayor a menor, obteniendo acceso directo al mayor elemento de cada rango consultado. Las Figuras 7.8, 7.9, 7.10, 7.11 y la Figura 7.12 muestran los tiempos de ejecución de las consultas `MAX` en los cubos de datos generados con distribución normal.

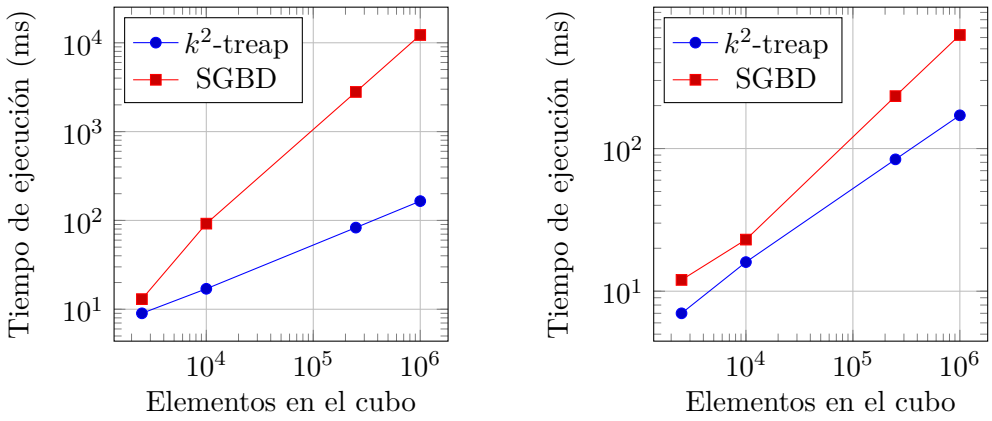
Nivel de la consulta	# Elementos en los cubos							
	2,500		10,000		250,000		1,000,000	
	k^2-treap	SGBD	k^2-treap	SGBD	k^2-treap	SGBD	k^2-treap	SGBD
Producto - Tienda	11	93	12	359	14	9,162	17	36,688
Producto - Ciudad	12	13	19	34	86	296	166	787
Producto - Región	9	13	17	92	83	2,795	165	12,234
Producto - All	7	12	16	23	84	233	171	625
Tipo - Tienda	10	12	17	33	79	394	171	1,170
Tipo - Ciudad	14	12	32	23	732	222	3,035	635
Tipo - Región	11	13	41	51	720	2,164	3,029	12,704
Tipo - All	10	12	37	22	734	213	3,180	595
Marca - Tienda	7	13	19	21	78	343	172	1,140
Marca - Ciudad	11	12	28	23	733	233	3,086	635
Marca - Región	8	12	27	41	720	3,076	3,098	7,736
Marca - All	7	13	25	22	740	202	3,235	585
All - Tienda	6	13	14	21	78	223	174	596
All - Ciudad	10	11	29	22	732	212	3,193	585
All - Región	7	12	28	22	725	213	3,192	606
All - All	6	12	26	23	753	182	3,316	474

Tabla 7.3: Tiempos de ejecución en milisegundos en consultas `SUM` para los cubos de datos generados con distribución normal



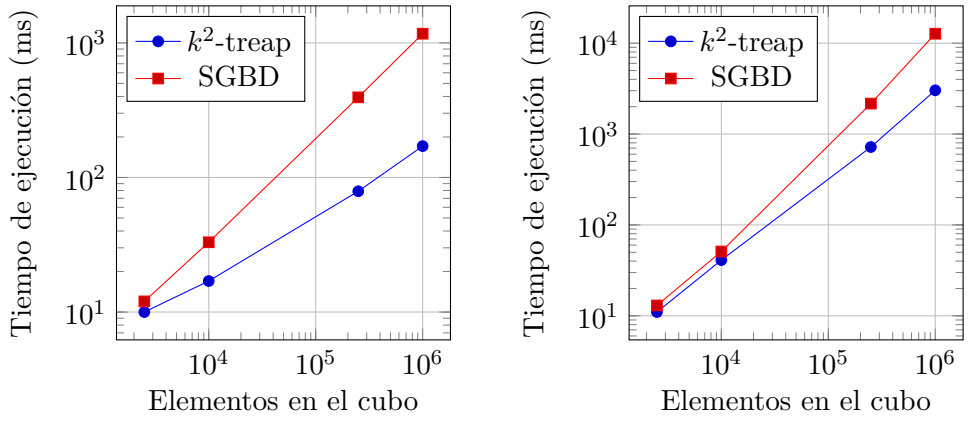
(a) Consulta agrupada por Producto y Tienda (b) Consulta agrupada por Producto y Ciudad

Figura 7.3: Tiempos de ejecución para consultas de agregación `SUM` agrupadas por los niveles Producto-Tienda y Producto-Ciudad



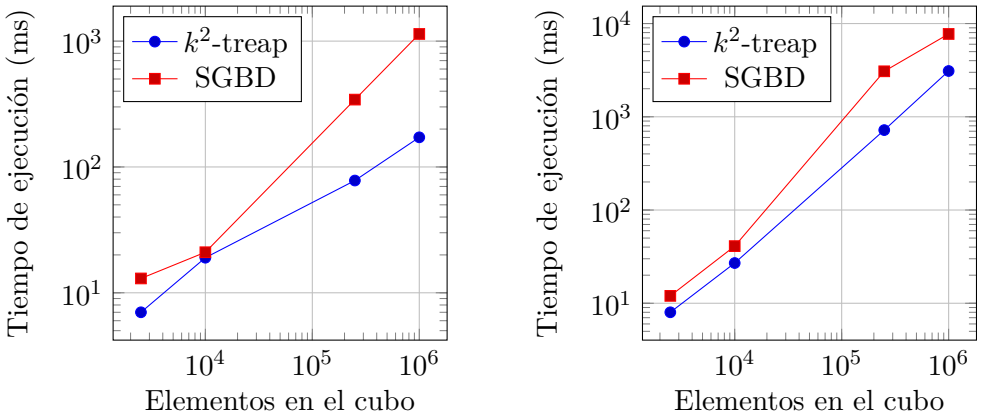
(a) Consulta agrupada por Producto y Región (b) Consulta agrupada por Producto y All

Figura 7.4: Tiempos de ejecución para consultas de agregación `SUM` agrupadas por los niveles Producto-Región y Producto-All



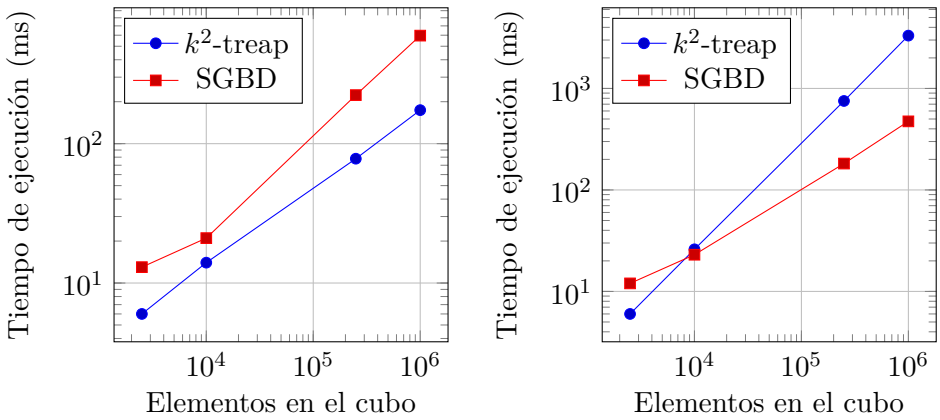
(a) Consulta agrupada por Tipo y Tienda (b) Consulta agrupada por Tipo y Región

Figura 7.5: Tiempos de ejecución para consultas de agregación `SUM` agrupadas por los niveles Tipo-Tienda y Tipo-Región



(a) Consulta agrupada por Marca y Tienda (b) Consulta agrupada por Marca y Región

Figura 7.6: Tiempos de ejecución para consultas de agregación `SUM` agrupadas por los niveles Marca-Tienda y Marca-Región



(a) Consulta agrupada por All y Tienda (b) Consulta agrupada por All y All

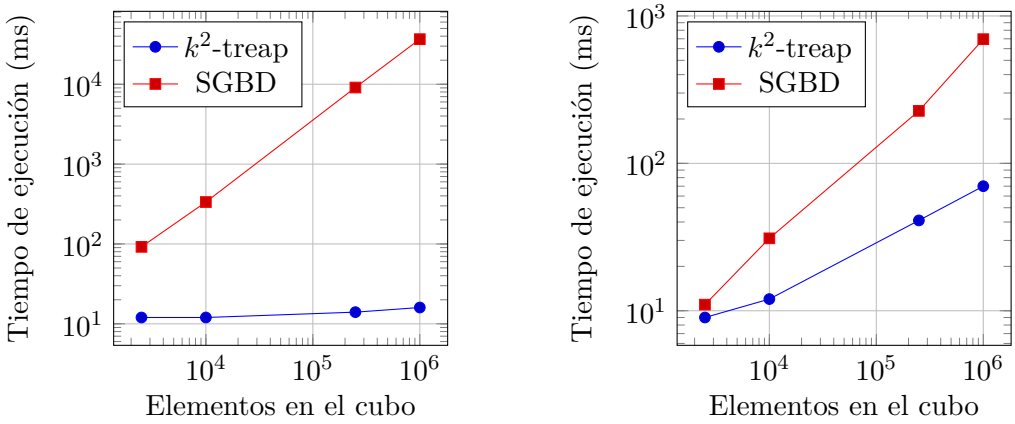
Figura 7.7: Tiempos de ejecución para consultas de agregación `SUM` agrupadas por los niveles All-Tienda y All-All

Nivel de la consulta	# Elementos en los cubos							
	2,500		10,000		250,000		1,000,000	
	k^2-treap	SGBD	k^2-treap	SGBD	k^2-treap	SGBD	k^2-treap	SGBD
Producto - Tienda	12	101	12	376	16	9988	19	40311
Producto - Ciudad	9	11	12	31	23	277	22	697
Producto - Región	6	11	10	11	16	2720	15	12416
Producto - All	5	12	8	24	9	141	9	625
Tipo - Tienda	7	11	8	31	18	394	26	1100
Tipo - Ciudad	3	11	3	11	5	154	7	555
Tipo - Región	2	11	1	12	2	1944	3	8942
Tipo - All	1	12	0	11	1	132	3	524
Marca - Tienda	4	11	5	11	10	304	17	1139
Marca - Ciudad	2	11	1	12	3	142	2	555
Marca - Región	1	11	0	11	1	1148	0	5697
Marca - All	0	11	0	11	0	213	0	514
All - Tienda	3	11	3	24	6	163	12	534
All - Ciudad	1	13	1	11	2	132	1	535
All - Región	0	12	0	11	0	131	0	565
All - All	0	12	0	11	0	102	0	393

Tabla 7.4: Tiempos de ejecución en milisegundos en consultas `MAX` para los cubos de datos generados con distribución uniforme

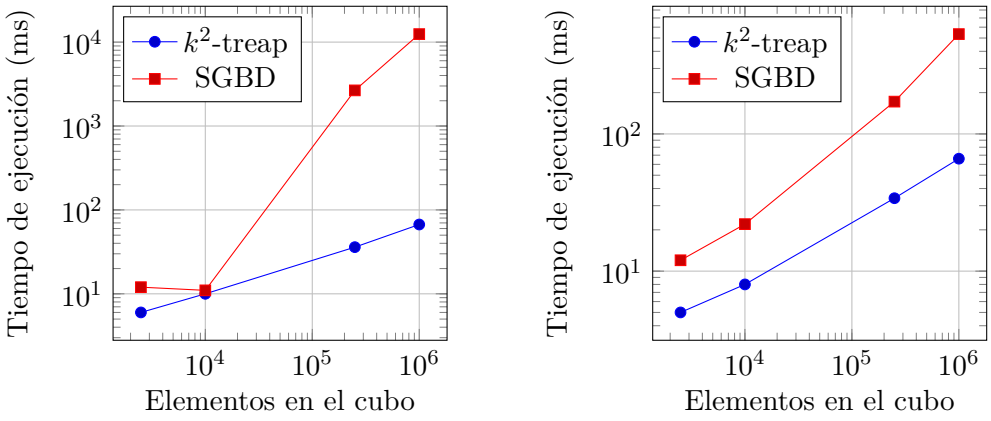
Nivel de la consulta	# Elementos en los cubos							
	2,500		10,000		250,000		1,000,000	
	k^2-treap	SGBD	k^2-treap	SGBD	k^2-treap	SGBD	k^2-treap	SGBD
Producto - Tienda	12	92	12	335	14	9100	16	36686
Producto - Ciudad	9	11	12	31	41	227	70	696
Producto - Región	6	12	10	11	36	2659	67	12456
Producto - All	5	12	8	22	34	172	66	534
Tipo - Tienda	7	12	8	21	18	314	20	1120
Tipo - Ciudad	3	11	3	12	4	213	4	545
Tipo - Región	2	13	1	11	2	1923	1	8801
Tipo - All	1	13	0	11	1	131	0	514
Marca - Tienda	4	11	5	12	11	364	14	1079
Marca - Ciudad	2	11	1	12	2	142	1	626
Marca - Región	1	11	0	22	0	1149	0	5771
Marca - All	0	14	0	12	0	132	0	585
All - Tienda	3	11	3	12	7	152	10	525
All - Ciudad	1	12	1	11	1	121	1	504
All - Región	0	12	0	24	0	131	0	576
All - All	0	12	0	12	0	102	0	444

Tabla 7.5: Tiempos de ejecución en milisegundos en consultas `MAX` para los cubos de datos generados con distribución normal



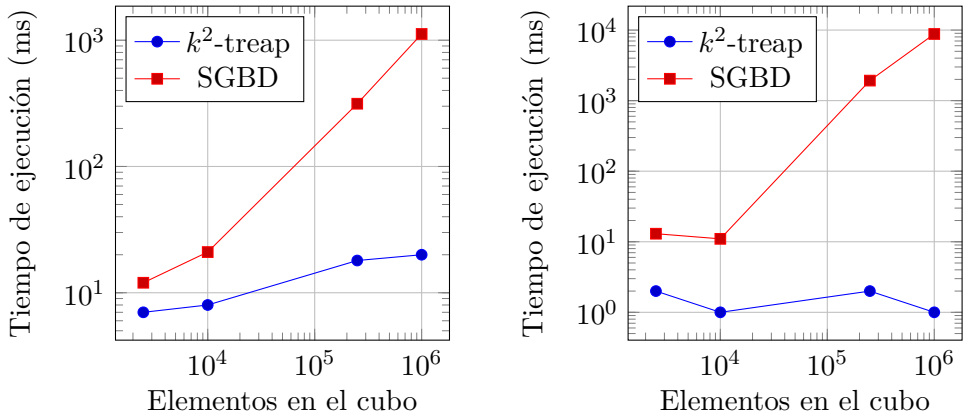
(a) Consulta agrupada por Producto y Tienda (b) Consulta agrupada por Producto y Ciudad

Figura 7.8: Tiempos de ejecución para consultas de agregación `MAX` agrupadas por los niveles Producto-Tienda y Producto-Ciudad



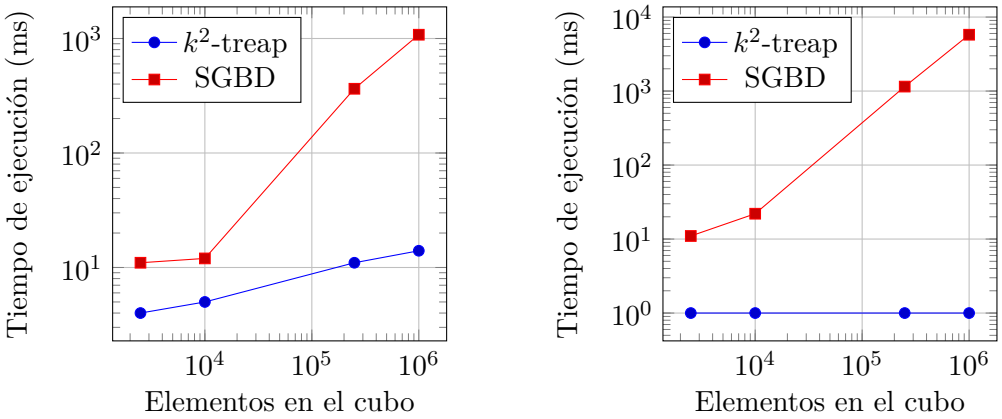
(a) Consulta agrupada por Producto y Región (b) Consulta agrupada por Producto y All

Figura 7.9: Tiempos de ejecución para consultas de agregación `MAX` agrupadas por los niveles Producto-Región y Producto-All



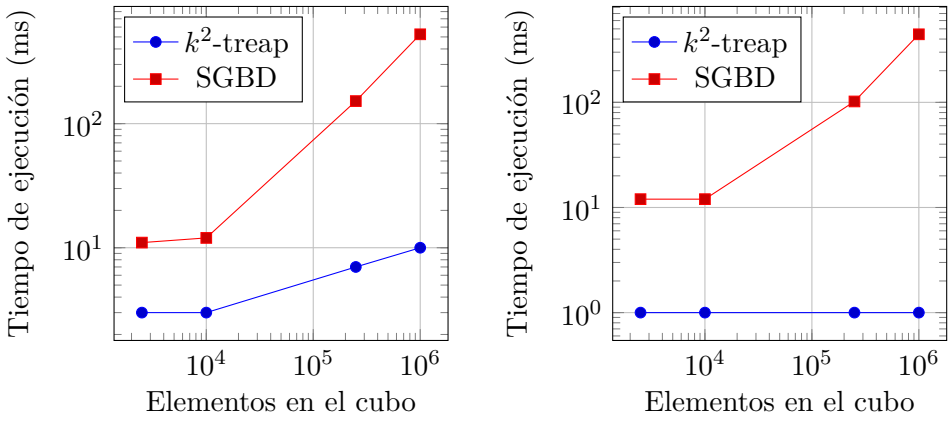
(a) Consulta agrupada por Tipo y Tienda (b) Consulta agrupada por Tipo y Región

Figura 7.10: Tiempos de ejecución para consultas de agregación `MAX` agrupadas por los niveles Tipo-Tienda y Tipo-Región



(a) Consultas agrupadas por Marca y Tienda (b) Consultas agrupadas por Marca y Región

Figura 7.11: Tiempos de ejecución para consultas de agregación `MAX` agrupadas por los niveles Marca-Tienda y Marca-Región



(a) Consultas agrupadas por All y Tienda (b) Consulta agrupada por All y All

Figura 7.12: Tiempos de ejecución para consultas de agregación `MAX` agrupadas por los niveles All-Tienda y All-All

Capítulo 8

Conclusiones y Trabajo Futuro

Mejorar la eficiencia en el procesamiento de consultas OLAP ha sido ampliamente estudiado en la literatura de los DWs. Un enfoque común es computar consultas utilizando tablas de resumen (resultados pre-computados) y construir índices sobre estas tablas. En esta línea, existen trabajos que atacan el problema de cómo seleccionar estas tablas y mantenerlas actualizadas. Otros trabajos se han centrado en implementar métodos de compresión para disminuir el volumen de los cubos de datos. Sin embargo, más que compresión propiamente tal, estos trabajos realizan agregación de datos, que a su vez generan imprecisiones en las respuestas a las consultas. En estos trabajos se asume que el cubo de datos se almacena en memoria secundaria, y no en memoria principal.

En esta tesis se propone un nuevo enfoque para DWs que permita generar cubos de datos en memoria principal a través de estructuras de datos compactas. Lo anterior, con el objetivo de reducir de manera efectiva el tamaño de los cubos de datos. Además, se propone la definición e implementación de métodos eficientes para procesar consultas sobre las estructuras de datos definidas, de modo que se puedan entregar respuestas exactas y no aproximadas a las consultas.

Se presenta la representación de DWs utilizando la estructura de datos compacta k^2-treap, arreglos y relaciones en bitmaps. Solo fueron considerados cubos de datos con dos dimensiones, pero la estructura de datos compacta k^2-treap puede ser modificada para considerar más dimensiones (Brisaboa et al., 2016).

Los experimentos presentados sobre datos sintéticos, muestran que, utilizando estructuras de datos compactas se puede ahorrar espacio de almacenamiento en memoria principal y que al realizar consultas de agregación con las funciones `SUM` y `MAX` se puede responder más eficientemente (en la mayoría de los casos), que usando un SGBD tradicional. Realizar consultas de agregación con la función de agregación `MAX` presentó mejores resultados que al utilizar la función de agregación con la función de agregación `SUM`, ya que la estructura de datos k^2-treap fue diseñada para responder consultas *top-k*, lo cual permite obtener el valor mayor contenido en un rango de consulta, correspondiente al primer elemento de la cola de prioridad. Como trabajo futuro se considerará implementar las demás funciones de agregación, tales como, `AVG` (promedio), y `COUNT`. Sin embargo, la estructura compacta k^2-treap no fue diseñada para responder de manera eficiente consultas de agregación con

la función `MIN`, por lo que podría no dar resultados muy favorables.

Un trabajo reciente, presentado en (Brisaboa et al., 2016) describe una nueva estructura de datos compacta llamada CMHD (Compact representation of Multidimensional data on Hierarchical Domains) para computar consultas de agregación con la función de agregación `SUM`. Esta estructura de datos compacta está basada en otra estructura de datos compacta llamada k^n-treap (variante del k^2-treap), que soporta multiples dimensiones. Como trabajo futuro planificamos comparar nuestros algoritmos con los algoritmos sobre la estructura de datos compacta CMHD considerando más dimensiones.

Referencias

N. Brisaboa, A. Cerdeira-Pena, N. López-López, G. Navarro, M. Penabad, y F. Silva-Coira. Efficient representation of multidimensional data over hierarchical domains. En *Proceedings of the 23rd International Symposium on String Processing and Information Retrieval*, págs. 191–203. 2016.

N. Brisaboa, G. De Bernardo, R. Konow, y G. Navarro. K2-treaps: Range top-k queries in compact space. En *Proceedings of the 21st International Symposium on String Processing and Information Retrieval*, tomo 8799 de *Lecture Notes of Compute Cience*, págs. 215–226. 2014a.

N. Brisaboa, S. Ladra, y G. Navarro. K2-trees for compact web graph representation. En *Proceedings of the 16th International Symposium on String Processing and Information Retrieval*, SPIRE'09, págs. 18–30. 2009.

N. Brisaboa, S. Ladra, y G. Navarro. Dacs: Bringing direct access to variable-length codes. *Information Processing and Management*, 49(1):392–404, 2013a.

N. Brisaboa, S. Ladra, y G. Navarro. Compact representation of web graphs with extended functionality. *Inf. Syst.*, 39:152–174, 2014b.

N. Brisaboa, M. Luaces, G. Navarro, y D. Seco. Space-efficient representations of rectangle datasets supporting orthogonal range querying. *Information Systems*, 38(5):635–655, 2013b.

S. Chaudhuri y U. Dayal. An overview of data warehousing and olap technology. *SIGMOD Record*, 26(1):65–74, 1997.

F. Claude y G. Navarro. A fast and compact web graph representation. En *Proceedings of the 14th International Symposium on String Processing and Information Retrieval*, tomo 4726 de *Lecture Notes in Computer Science*, págs. 105–116. 2007.

F. Claude y G. Navarro. Practical rank/select queries over arbitrary sequences. En *Proceedings of the 15th International Symposium on String Processing and Information Retrieval*, SPIRE'08, págs. 176–187. 2009.

G. De Bernardo, S. Álvarez-García, N. Brisaboa, G. Navarro, y O. Pedreira. Compact querieable representations of raster data. En *String Processing and Information Retrieval - 20th International Symposium*, págs. 96–108. 2013.

P. Furtado y H. Madeira. Data cube compression with quanticubes. En *Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery*, págs. 162–167. 2000a.

P. Furtado y H. Madeira. Fcompress: A new technique for queriable compression of facts and datacubes. En *Proceedings of the International Database Engineering and Applications Symposium*, págs. 197–206. 2000b.

H. Garcia-Molina y K. Salem. Main memory database systems: An overview. *IEEE Trans. Knowl. Data Eng.*, 4(6):509–516, 1992.

M. Golfarelli y S. Rizzi. Honey, I shrunk the cube. En *Proceedings of the Advances in Databases and Information Systems*, tomo 8133 de *Lecture Notes in Computer Science*, págs. 176–189. 2013.

H. Gupta, V. Harinarayan, A. Rajaraman, y J. Ullman. Index selection for olap. En *Proceedings of the Thirteenth International Conference on Data Engineering*, págs. 208–219. 1997.

V. Harinarayan, A. Rajaraman, y J. Ullman. Implementing data cubes efficiently. *SIGMOD Rec.*, 25(2):205–216, 1996.

C. Hurtado y C. Gutierrez. *Data Warehouses and OLAP: Concepts, Architectures and Solutions*, cap. Handling Structural Heterogeneity in OLAP. Idea Group, Inc, 2007.

Ch. Jensen, T. Pedersen, y Ch. Thomsen. *Multidimensional Databases and Data Warehousing*. Morgan and Claypool Publishers, 1era edición, 2010.

S. Ladra, J. Paramá, y F Silva-Coira. Scalable and queryable compressed storage structure for raster data. *Information Systems*, 72(Supplement C):179 – 204, 2017.

J. Li, D. Rotem, y J. Srivastava. Aggregation algorithms for very large compressed data warehouses. En *Proceedings of 25th International Conference on Very Large Data Bases*, págs. 651–662. 1999.

J. Li y J. Srivastava. Efficient aggregation algorithms for compressed data warehouses. *IEEE Trans. Knowl. Data Eng.*, 14(3):515–529, 2002.

S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–136, 1982.

I. Mumick, D. Quass, y B. Mumick. Maintenance of data cubes and summary tables in a warehouse. *SIGMOD Rec.*, 26(2):100–111, 1997.

G. Navarro. Spaces, trees, and colors: The algorithmic landscape of document retrieval on sequences. *ACM Computing Surveys*, 46(4):52:1–52:47, 2014.

G. Navarro. *Compact Data Structures: A Practical Approach*. Cambridge University Press, 1era edición, 2016.

G. Navarro y K. Sadakane. Fully functional static and dynamic succinct trees. *ACM Transactions on Algorithms*, 10(3):16:1–16:39, 2014.

H. Plattner y A. Zeier. *In-Memory Data Management: An Inflection Point for Enterprise Applications*. Springer Publishing Company, Incorporated, 1era edición, 2011.

R. Raman, V. Raman, y S. Rao. Succint dynamic data structures. En *Algorithms and Data Structures*, tomo 2125 de *Lecture Notes in Computer Science*, págs. 426–437. 2001.

K. Ross y D. Srivastava. Fast computation of sparse datacubes. En *Proceedings of 23rd International Conference on Very Large Data Bases*, págs. 116–125. 1997.

K. Ross y K. Zaman. Serving datacube tuples from main memory. En *Proceedings of the 12th International Conference on Scientific and Statistical Database Management*, págs. 182–195. 2000.

R. Seidel y C. Aragon. Randomized search trees. *Algorithmica*, 16(4/5):464–497, 1996.

W. Wang, H. Lu, J. Feng, y J. Xu Yu. Condensed cube: An efficient approach to reducing data cube size. En *Proceedings of the 18th International Conference on Data Engineering*, págs. 155–165. 2002.

W Wu, H. Gao, y J. Li. New algorithm for computing cube on very large compressed data sets. *IEEE Trans. Knowl. Data Eng.*, 18(12):1667–1680, 2006.

F. Yu y S. Wang. Compressed data cube for approximate OLAP query processing. *J. Comput. Sci. Technol.*, 17(5):625–635, 2002.