Universidad del Bío-Bío, Chile

Facultad de Ciencias Empresariales

Departamento de Sistemas de Información

Sistema Web y Móvil que implementa consultas de agregación sobre Data Warehouses almacenados en la estructura de datos compacta CMHD

Proyecto de título presentado por Fernando Matías Linco Poblete
Para obtener el título de Ingeniero Civil en Informática
Dirigida por Dra. Mónica Caniupán

2017

Resumen

En este proyecto se presenta el desarrollo de un sistema Web y un sistema para dispositivos móviles para implementar consultas de agregación con funciones SUM, MIN, MAX, COUNT y AVG sobre Data Warehouses (DWs) almacenados en la estructura de datos compacta CMHD (Compact representation of Multidimensional data on Hierarchical Domains). Las estructuras de datos compactas son estructuras que permiten compactar los datos sin perder la capacidad de consultarlos en su forma compacta, estas combinan en una única estructura de datos una representación compacta de los datos para acceder a dichos datos. La estructura de datos compacta CMHD permite consultar eficientemente información agregada. Inicialmente fue implementada para computar consultas de agregación con función SUM sobre matrices multidimensionales y en este proyecto se extiende su capacidad para computar consultas con funciones de agregación MAX, MIN, COUNT y AVG sobre DWs compactos. Un Data Warehouse es una colección de datos orientados a un tema, integrados, no volátiles e históricos, organizados para apoyar el proceso de toma de decisiones. La mejora en la eficiencia al momento de procesar consultas en DWs es un tema de gran importancia, por lo tanto, se han realizado diversos esfuerzos por mejorar el procesamiento de consultas tales como mantener vistas de datos materializados y actualizados. En este proyecto nos enfocamos en el uso de la EDC CMHD para responder a consultas sobre DWs compactados de manera eficiente. Mediante experimentación sobre DWs con datos sintéticos mostramos que utilizando una representación compacta de DWs, podemos lograr un mejor rendimiento en el procesamiento de consultas de agregación.

Keywords — Estructuras de datos compactas, CMHD, Data Warehouses.

Agradecimientos

En primer lugar, agradecer a mi profesora guía la Dra. Mónica Caniupán, por su paciencia y sus constantes correcciones que fueron de gran ayuda durante todo el proceso del desarrollo de este proyecto.

Al académico Fernando Silva, por facilitarme su código para implementar la estructura de datos compacta CMHD, indispensable para que fuera posible este proyecto. Además por contribuir a este proyecto con sus conocimientos y sugerencias principalmente vía correo electrónico.

Agradecer a mis padres y familia por confiar siempre en mí, darme apoyo y ánimos para continuar y no rendirme.

A todos aquellos amigos que me dieron ánimos y buenos consejos durante todo este periodo.

Por último, agradecer que este proyecto de título fue financiado por el grupo de investigación de Algoritmos y Bases de Datos (ALBA) código GI 160119/EF

Índice de General

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

Capítulo 1

Introducción

Un Data Warehouse (DW) es una colección de datos orientados a un tema, integrados, no volátiles e históricos, organizados para apoyar al proceso de toma de decisiones (Inmon, 1992). Los DWs se organizan de acuerdo a *dimensiones* y *hechos*. Una *dimensión* define un área de análisis de la información, por ejemplo, el tiempo, la geografía, etc. Donde dimensión se define como una jerarquía de niveles o categorias. El conjunto de elementos de una dimensión se denominan *miembros*. Los *hechos* corresponden a datos cuantitativos o medidas asociados a las dimensiones (Chaudhuri y Dayal, 1997), por ejemplo, las ventas o los costos de productos en una fecha determinada, etc. Se debe considerar la granularidad de la información, es decir, si tenemos una granularidad diaria, podemos analizar estos datos por día, sin embargo si se define una granularidad mensual, el acceso a la información debe ser mensual. Los hechos se presentan generalmente en *cubos de datos*, un *cubo de datos* es una estructura multidimensional para capturar y analizar datos, estos cubos contienen los *hechos* a distintos niveles de granularidad por cada dimensión.

Generalmente, los DWs no reciben operaciones de inserción y/o eliminación de datos, sino que recargas diarias o semanales de datos, por lo que es posible hacer consultas sobre vistas del DW o bien, sobre una versión no actualizada recientemente. La mejora en la eficiencia al momento de procesar consultas en un DWs es un tema de gran importancia, por lo tanto, se han realizado diversos esfuerzos por mejorar el procesamiento de consultas. En (Silva, 2017) se presentan estructuras de datos compactas para almacenar DWs en memoria principal y realizar consultas de agregación de manera eficiente. 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 (Gupta et al., 1997) se presentan algoritmos para seleccionar automáticamente tablas de resumen para ser materializadas. Por otro lado, en (Mumick et al., 1997) se analiza el problema de cómo mantener vistas agregadas de datos.

En muchos dominios de aplicaciones, los datos se organizan en matrices multidimensionales. En algunos casos, como el modelado 3D y el modelado en GIS (Geographic Information Systems) (Worboys y Duckham, 2004), los datos son puntos que se encuentran en un espacio discreto de dos o tres dimensiones. Sin embargo, existen otros dominios, como los sistemas OLAP (Online Analytical Processing) (Chaudhuri y Dayal, 1997), donde los

datos son conjuntos de tuplas en un cubo de datos multidimensional. En este tipo de entornos, es muy frecuente tener que consultar grandes conjuntos de datos, donde se involucran varias dimensiones. Además, es necesario responder a operaciones complejas para obtener los valores sobre regiones muy específicas del cubo multidimensional, lo que implica recuperar un gran número de celdas. Esto nos presenta dos grandes desafíos: debemos reducir los requisitos de espacio de esos conjuntos de datos y, al mismo tiempo, incluir algún tipo de índice para mejorar la eficiencia de las consultas de agregación (Navarro, 2016). Para lograr estos objetivos, podemos utilizar *estructuras de datos compactas*.

Las estructuras de datos compactas son estructuras de datos que permiten compactar los datos sin perder la capacidad de consultarlos en su forma compacta, lo que permite mantener su funcionalidad y realizar consulta sobre los datos almacenados de forma eficiente. Una estructura de datos compacta combina, en una única estructura, una representación comprimida de los datos y los métodos para acceder a dichos datos. Además, estas estructuras permiten procesar grandes conjuntos de datos en memoria principal, donde el tiempo de acceso disminuye con respecto a los niveles más bajos de la jerarquía de memoria (Raman et al., 2001). Las estructuras de datos compactas se han utilizado en diversas áreas. Por ejemplo, en (Brisaboa et al., 2013a) permiten la representación de documentos en el contexto de recuperación de información. En (Brisaboa et al., 2009) son usadas para representar grafos de la WEB. También, se han utilizado para mejorar la eficiencia de las consultas en Sistemas de Información Geográfica (Brisaboa et al., 2013b), entre otros. En este proyecto proponemos el uso de la estructura de datos compacta CMHD (Compact representation of Multidimensional data on Hierarchical Domains) para computar consultas de agregación con operadores SUM, MIN, MAX, COUNT y AVG sobre Data Warehouses. Esta estructura de datos compacta fue presentada en (Brisaboa et al., 2016), para computar consultas de agregación sobre matrices multidimensionales con la función SUM, tales como: "obtener la suma de ventas que se realizaron en una ciudad" o "obtener la suma de productos vendidos en un país".

El resto del documento se organiza de la siguiente manera. En el Capítulo 2 se presenta el objetivo general y los objetivos específicos de este proyecto, además de los alcances y límites de éste. En el Capítulo 3 se describen los conceptos preliminares como, Data Warehouse, estructuras de datos compactas y la estructura de datos compacta CMHD. En el Capítulo 4 se presentan los algoritmos desarrollados en este proyecto para computar consultas de agregación sobre DWs. En el Capítulo 5 se presentan las herramientas utilizadas para la creación de ambas aplicaciones, además de ejemplos de funcionamiento, y también se presenta la interfaz de los sistemas. En el Capítulo 6 se presentan diferentes escenarios de experimentación donde se demuestra la eficiencia de las aplicaciones desarrolladas en cuanto a ahorro de espacio de almacenamiento en memoria principal y tiempo de ejecución de las diferentes consultas de agregación. Finalmente, en el Capítulo 7 se presentan las conclusiones del proyecto y posibles trabajos futuros.

Capítulo 2

Objetivos

En este capítulo se presentan los objetivos generales y específicos asociados al proyecto, además de los alcances y límites del mismo.

2.1. Objetivo General

El objetivo principal de este proyecto es implementar un sistema WEB y un sistema para dispositivos móviles que permitan realizar consultas de agregación con funciones SUM, MIN, MAX, COUNT y AVG sobre Data Warehouses almacenados en la estructura de datos compacta CMHD.

2.2. Objetivos Específicos

Los objetivos específicos del proyecto son los siguientes:

1. Adaptar el algoritmo existente que implementa la estructura de datos compacta CMHD en lenguaje C++.

2. Implementar algoritmos para computar consultas de agregación con las funciones MIN, MAX, COUNT y AVG sobre la estructura de datos compacta CMHD.

3. Generar DWs ficticios con n-dimensiones y cubos de datos de diferentes tamaños.

4. Implementar un sistema WEB y un sistema para dispositivos móviles (Android) que interactúe con DWs almacenados en la estructura compacta CMHD.

5. Generar un escenario de experimentación que involucre DWs de prueba y consultas.

6. Realizar experimentación de las implementaciones de los sistemas sobre los diferentes DWs generados.

2.3. Alcances y Límites

Este proyecto contempla únicamente la estructura de datos compacta CMHD, no se consideran otras estructuras de datos compactas como k^2-treap (Brisaboa et al., 2009) que solo sirve para DW lineales y cubos de datos de dos dimensiones o k^n-treap (Brisaboa et al., 2009) ya que CMHD demuestra ser más eficiente que éste (Brisaboa et al., 2016). Los DWs utilizados en este proyecto solo se consideran dimensiones con esquemas jerárquicos lineales (como las mostradas en la Figura 3.2) y con un único nivel inferior, estas restricciones se deben a que la estructura de datos compacta CMHD solo soporta este tipo de dimensiones (Silva, 2017). Además, solo se considera el uso de un DW ficticio de tres dimensiones con diversos cubos de datos, pero CMHD puede trabaja con n dimensiones.

Solo considera el sistema operativo Android para aplicación de dispositivos móviles.

Capítulo 3

Conceptos Preliminares

Este capítulo presenta conceptos básicos, notaciones y la estructura de datos compacta CMHD que se utilizaran durante este proyecto. En la Sección 3.1 se presenta la definición de Data Warehouse. En la Sección 3.2 se define bitmap y las operaciones *Rank* y *Select*. En la Sección 3.3 se muestra la representacion de árboles comprimidos LOUDS. Finalmente, en la Sección 3.4 se presenta la estructura de datos compacta CMHD.

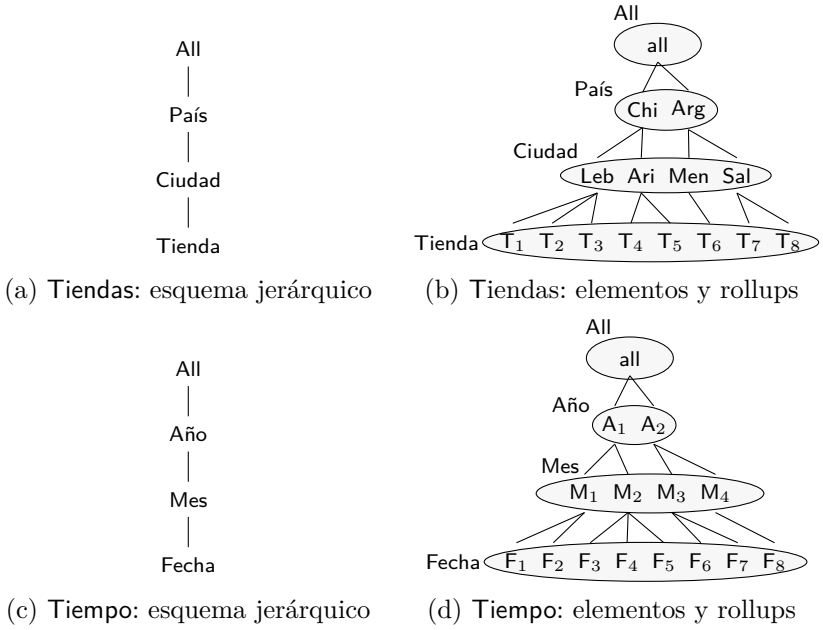
3.1. Data Warehouse(DW)

Un Data Warehouse (DW) es una colección de datos orientados a un tema, integrados, no volátiles e históricos, organizados para el apoyo del proceso de toma de decisiones (Inmon, 1992). Los DWs se organizan de acuerdo a *dimensiones* y *hechos*.

- *Dimensiones*: Una dimensión define un área de análisis de la información, por ejemplo, el tiempo, geografía, etc. Éstas se componen de niveles que se organizan de manera jerárquica 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*. *Rollup*, en terminología DW, hace referencia a la relación entre un nivel inferior y uno superior de una dimensión, estas relaciones son fundamentales para computar agregaciones de datos (Hurtado et al., 1999) (ver ejemplo 3.1). 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, por simplicidad en este proyecto consideramos dimensiones con un nivel inferior. El conjunto de elementos de una dimensión se denominan *miembros*.

  Por ejemplo la dimensión *Tiempo* tiene un esquema jerárquico de *Fecha-Mes-Año-All*. Estas jerarquías se representan generalmente mediante árboles. En la Figura 3.1(a) y la Figura 3.1(c), muestran respectivamente, los esquemas jerárquicos de las dimensiones *Tiendas* y *Tiempo*. Ambas dimensiones están compuestas por cuatro categorías, la dimensión *Tiendas* está compuesta por: *Tienda - Ciudad – País -*

All y la dimensión *Tiempo* está compuesta por: *Fecha* - *Mes* – *Año* – *All*. La operación de navegación sobre los niveles de las dimensiones se denomina *rollup*. En este proyecto solo consideramos dimensiones con esquemas jerárquicos lineales como se dijo anteriormente (ver Figura 3.1). Es decir, esquemas donde existe un único camino desde el nivel inferior al nivel superior *All*. Sin embargo, existen otras dimensiones más complejas donde el nivel inferior puede alcanzar el nivel superior *All* a través de más de un camino, incluso, pueden existir varios niveles inferiores. Estos esquemas se denominan esquemas heterogéneos (Hurtado-Larrain, 2002).



(a) Tiendas: esquema jerárquico (b) Tiendas: elementos y rollups

(c) Tiempo: esquema jerárquico (d) Tiempo: elementos y rollups

Figura 3.1: DW con dimensiones Tiendas y Tiempo

Ejemplo 3.1 Consideremos un DW para una empresa que tiene tiendas de ventas en diferentes países y ciudades, además poseen el registro de dichas ventas por fechas. La Figura 3.1(a) muestra el esquema jerárquico de la dimensión *Tiendas* y la Figura 3.1(c) muestra el esquema para la dimensión *Tiempo*. Estas dimensiones están compuestas por varios niveles, la dimensión *Tiendas* está compuesta por: $Tienda, Ciudad, País$ y *All*. Diremos que el nivel *Tienda* hace *rollup* a *Ciudad*, esto quiere decir que el nivel *Tienda* es hijo del nivel *Ciudad*, por lo tanto existe una relación *rollup*. *Ciudad* hace *rollup* a *País*, que a su vez hace *rollup* con *All*. La Figura 3.1(b) muestra estos elementos y las relaciones *rollups* para la dimensión *Tiendas*. Los elementos para el nivel *Tienda* son: $\{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8\}$. Para el nivel *Ciudad* los elementos son: $\{Leb(Lebu), Ari(Arica), Men(Mendoza), Sal(Salta)\}$. Los elementos del nivel *País* son: $\{Chi(Chile), Arg(Argentina)\}$. Finalmente el nivel *All* se compone de un elemento: $\{all\}$ (por definición). La dimensión *Tiempo* tiene los niveles de $Fecha, Mes, Año$ y*All*. El nivel *Fecha* hace

rollup a Mes, el cual hace *rollup* a *Año*, que a su vez hace *rollup* con *All*. El nivel $Fecha$ posee los elemente: $\{F_1, F_2, F_3, F_4\}$. Los elementos del nivel Mes son: $\{M_1, M_2, M_3, M_4, M_5, M_6, M_7, M_8\}$. Los elementos del nivel *Año* son: $\{A1, A2\}$, al igual que en la dimensión $Tiendas$, la dimensión $Tiempo$ tiene un nivel *All*: $\{all\}$. Ver la Figura 3.1(d).

- $Hechos$: Los *hechos* corresponden a datos cuantitativos o medidas asociados a las dimensiones, por ejemplo, las ventas de productos o los costos. Se debe considerar la granularidad de la información, es decir, si tenemos una granularidad diaria, podemos analizar estos datos por día. La granularidad es el nivel de detalle de los datos a almacenar. Se debe definir las formas de agregación de cada hecho por cada dimensión (suma, máximo, mínimo, cantidad, promedio). Para analizar los hechos existen cuatro operaciones básicas, las operaciones *drilldown* y *rollup* son utilizadas para mover la vista hacia y desde un mayor nivel de detalle a un menor nivel de detalle, las operaciones de *slice* y *dice* corresponden a operaciones para la visualización de los datos a través del cubo, estas son realizadas a través del proceso OLAP (On-line Analytical Processing) (Chaudhuri y Dayal, 1997). Un *cubo de datos* es una estructura multidimensional para capturar y analizar datos. Los *hechos* se presentan generalmente en cubos de datos, que contienen los *hechos* a distintos niveles de granularidad por las dimensiones. Luego para consultar los diferentes niveles de las dimensiones se usan las jerarquías de éstas por medio de operaciones *rollup* y *drilldown*. La Figura 3.2 y el Ejemplo 3.2 ilustran estos conceptos.

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
F_1	0	1	1	0	1	0	2	2
F_2	1	2	1	2	1	0	1	0
F_3	0	0	0	0	0	0	0	0
F_4	0	0	0	0	0	1	0	0
F_5	0	0	0	0	0	3	0	0
F_6	0	0	0	0	1	0	0	0
F_7	0	0	0	0	1	0	0	0
F_8	1	1	0	0	0	0	1	2

(a) *rollup*: Fecha y Tienda

	Leb	Ari	Men	Sal
M_1	6	4	0	5
M_2	0	0	4	0
M_3	0	2	0	0
M_4	2	0	0	3

(b) *rollup*: Mes y Ciudad

Figura 3.2: Cubos de datos para el DW de la Figura 3.1

Ejemplo 3.2 La Figura 3.2 muestra los cubos de datos de las ventas realizadas agrupadas por $Fecha - Tienda$ y $Mes - Ciudad$, esto para el DW de la Figura 3.1. Por ejemplo en la Figura 3.2(a), para la Fecha F_1 se registró una venta en las Tiendas $\{T_2, T_3, T_5\}$, para $\{T_7, T_8\}$ se registraron dos ventas y para $\{T_1, T_4, T_6\}$ cero ventas. En la tabla 3.2(b) se muestran las ventas agrupadas en un nivel superior, esta responde a consultas de agregación como: "obtener la cantidad de ventas en Arica (Ari) para el Mes (M_1), para responder a esta consulta se deben utilizar las relaciones *rollup* entre los niveles $Tienda$ y $Ciudad$ para la dimensión $Tiendas$ y para la dimensión $Tiempo$ los *rollup* de $Fecha$ y Mes.

Generalmente, los DWs no reciben operaciones de inserción y/o eliminación de datos, sino que recargas diarias o semanales de datos, por lo que es posible hacer consultas sobre vistas del DW o bien, sobre una versión no actualizada recientemente. La mejora en la eficiencia al momento de procesar consultas en un DWs es un tema de gran importancia, por lo tanto, se han realizado diversos esfuerzos por mejorar el procesamiento de consultas (Silva, 2017).

3.2. Bitmaps y operaciones sobre bitmaps

Un bitmap es una estructura de datos que se asemeja a un arreglo pero que sus valores son $0-1$. Es decir, cada celda es un bit (Navarro, 2016). Dado un bitmap $B[1,n]$, podemos definir tres operaciones básicas:

- $Rank_b(B,i)$ devuelve el número de ocurrencias del bit $b \in \{0,1\}$ en $B[1,i]$. Considere un mapa de bits $B = [10110101011]$. Por lo tanto, $rank_0(B,6) = 2$, es decir, se cuentan dos 0 hasta la posición 5, mientras que $rank_1(B,5) = 3$, desde la secuencia $B[1,3]$ tiene tres $1s$. Al omitir b, la operación de $rank$ devuelve el número de $1s$ hasta una posición dada, es decir, $rank(B,i) = rank_1(B,i)$.
- $Select_b(B,i)$ ubica la posición de la i-ésima ocurrencia de b en B. Siguiendo el ejemplo anterior, $select_0(B,2) = 5$ y $select_1(B,2) = 3$, lo que significa que el segundo 0 aparece en la posición 5 y el segundo 1 en la posición 3 en el bitmap B.
- $Access(B,i)$ devuelve el valor del bit en la posición i. Por ejemplo, $access(B,5) = 0$, es decir, el quinto valor del bitmaps B es un 0.

Estas operaciones básicas son utilizadas en la mayoría de las estructuras de datos compactas. La Figura 3.3 ilustra las tres operaciones previas.

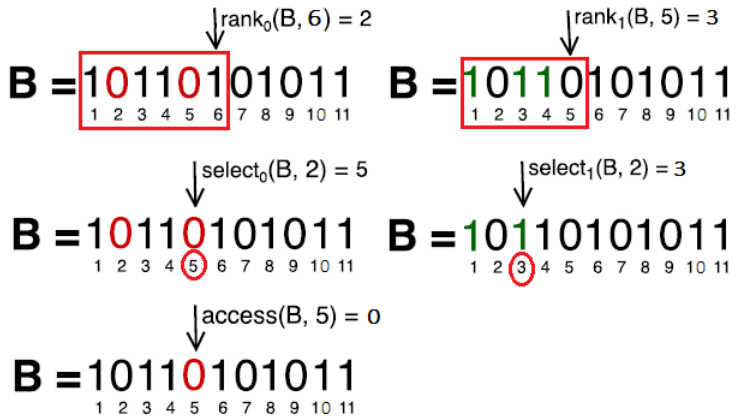


Figura 3.3: Funciones Rank, Select y Access sobre el bitmap B

3.3. Representación de árboles comprimidos LOUDS

Level-Ordered Unary Degree Sequence (LOUDS) (Jacobson, 1989), es una representación de árboles binarios, para árboles ordenados, que agrega el grado r de cada nodo (de izquierda a derecha) en orden de nivel, a un código unario representado por (1^r0), donde r representa la cantidad de hijos que tiene un nodo (cantidad de 1s en la secuencia). Donde cada nodo se representa con un bit 0 y cada hijo de ese nodo con un bit 1. Por ejemplo, si un nodo tiene 3 hijos, vale decir su grado es $r = 3$, la secuencia de bits (S) almacena el valor de $S = [1110]$. Esta secuencia almacena un árbol de n nodos utilizando $2n - 1$ bits. Al agregar un nodo raíz falso, podemos mantener la propiedad de que todos los nodos del árbol, incluida la raíz, corresponden a un nodo de bit 1 en la secuencia S.

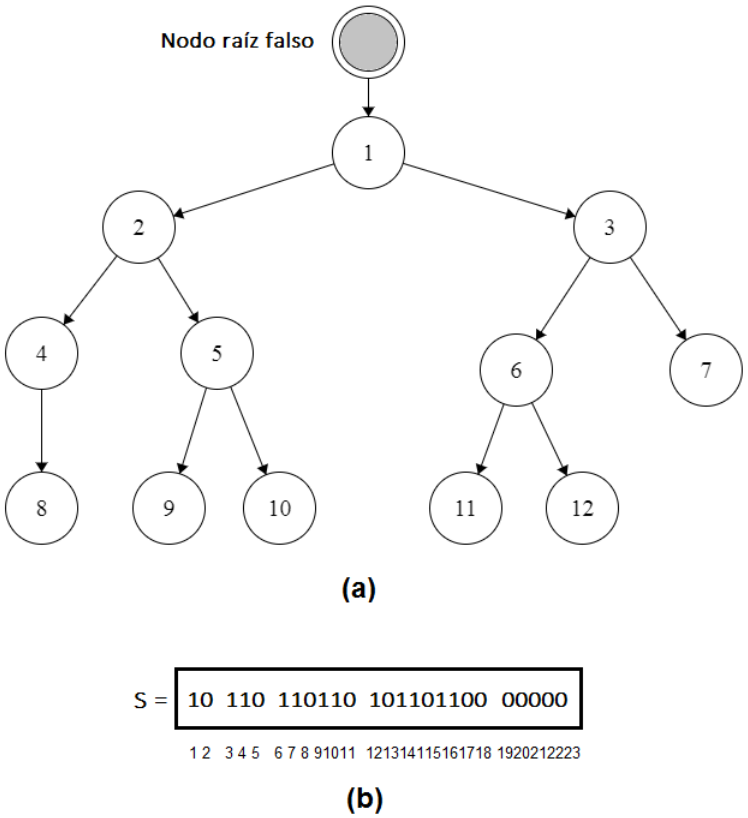


Figura 3.4: Representación LOUDS para un árbol de 12 nodos

Ejemplo 3.3 La Figura 3.4 (a) muestra un árbol ordenado y la Figura 3.4 (b) su representación LOUDS S. El nodo raíz falso (en color gris) no pertenece al árbol original. El proceso sigue un orden de nivel de izquierda a derecha, comenzando por el nodo raíz. Este nodo tiene un grado $r = 1$ (solo un hijo), por lo tanto, agregamos 1 en código unario $(1^r0 = 1^10 = 10)$ a la secuencia S. Luego, continuamos con el siguiente nodo etiquetado como 1 (el nodo raíz real). El nodo 1 tiene un grado r = 2, por lo que concatenamos la

secuencia de los bits 110 a S. A continuación, pasamos al siguiente nivel y procesamos el nodo etiquetado como 2, que tiene un grado $r = 2$, por lo que la secuencia 110 se agrega a S. El proceso continúa para todos los nodos restantes del árbol y se detiene cuando todos los grados sean igual a 0, por lo tanto se llego al final del árbol. La secuencia final es $S = [10110110110101101100000000]$ (25 bits de longitud).

Para navegar sobre esta representación y consultar por un nodo *Padre* o un nodo $Hijo$, se utilizan las operaciones de *Rank* y *Select*. Se utilizarán las siguientes fórmulas, donde x representa la posición del nodo a consultar.

$$\begin{aligned} \text{Padre}\,(x) &= \text{Select}_1(\text{Rank}_0(x)) \\ \text{Hijo}\,(x) &= \text{Select}_0(\text{Rank}_1(x)) + 1 \end{aligned}$$

Figura 3.5: Consultas para Padre e Hijo sobre LOUDS

Ejemplo 3.4 Si consideramos el ejemplo de la Figura 3.4, y queremos consultar por un nodo *Padre* o un nodo $Hijo$ utilizamos las consultas de la Figura 3.5. Por ejemplo, para saber el nodo *Padre* del nodo en el árbol (a) cuya etiqueta es 4 y cuyo bit 1 se encuentra en la posición 6 de la secuencia S, tenemos que $Padre(6) = Select_1(Rank_0(6)) = Select_1(2) = 3$. Por lo tanto el nodo *Padre* del sexto bit 1 de la secuencia S se encuentra en la posición 3 de S, que corresponde al nodo etiquetado como 2. Ahora si queremos consultar por el primer $Hijo$ del nodo ubicado en la posición 4, cuya etiqueta es 3, tenemos $Hijo(4) = Select_0(Rank_1(4)) + 1 = Select_0(3) + 1 = 8 + 1 = 9$, por lo tanto el primer nodo $Hijo$ del nodo etiquetado como 3 se encuentra en la posición 9, corresponde al nodo etiquetado como 6.

3.4. Estructura de datos compacta CMHD

La estructura de datos compacta CMHD (Compact representation of Multidimensional data on Hierarchical Domains) fue presentada en (Silva, 2017), ésta divide la matriz siguiendo la jerarquía natural de los elementos en cada dimensión. De esta forma, permite la respuesta eficiente de consultas que consideran la semántica de las dimensiones. Esta estructura es mucho más eficiente que una estructura multidimensional genérica, ya que las consultas se resuelven agregando muchos menos nodos del árbol que las conforma. A continuación se presenta una matriz de dos dimensiones creada a partir del DW de la Figura 3.1 y el cubo de datos de la Figura 3.2, esta será utilizada para ejemplificar la creación de un CMHD. La matriz registra el número de ventas para todas las tiendas en un tiempo determinado. En particular, las tiendas se agrupan en ciudades y estas en países, mientras que las fechas se agrupan por meses y estos en años.

			Chi					Arg		
			Leb			Ara		Men	Sal	
			T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
A1	M_1	F_1	0	1	1	0	1	0	2	2
		F_2	1	2	1	2	1	0	1	0
	M_2	F_3	0	0	0	0	0	0	0	0
		F_4	0	0	0	0	0	1	0	0
		F_5	0	0	0	0	0	3	0	0
A2	M_3	F_6	0	0	0	0	1	0	0	0
		F_7	0	0	0	0	1	0	0	0
	M_4	F_8	1	1	0	0	0	0	1	2

Figura 3.6: Matriz para generar la estructura CMHD para el DW de la Figura 3.1 y el cubo de datos de la Figura 3.2(a)

3.4.1. Construcción del CMHD

Si consideramos la matriz de la Figura 3.6, una matriz de dos dimensiones, donde cada celda puede contener un valor (en este caso ventas) o estar vacía, CMHD divide recursivamente la matriz en varias sub-matrices. CMHD particiona de acuerdo a las dimensiones que forman a la matriz, teniendo en cuenta los límites impuestos por los niveles jerárquicos de cada dimensión. La construcción del CMHD se divide en dos parte, la representación de las jerarquías de las dimensiones y la representación para los datos de la matriz en sí.

- Representación de las jerarquías de dimensión: La jerarquía de una dimensión es esencialmente un árbol de N nodos, donde N es el número de elementos de las dimensiones. Representamos este árbol usando LOUDS (Jacobson, 1989), que usa $2N - 1$ bits, y se puede navegar de manera eficiente (ver en la Sección 3.3). A continuación se presenta la construcción de los LOUDS de las dimensiones para la matriz de la Figura 3.6. Esta matriz posee dos dimensiones, *Tiendas* y *Tiempo* destacadas en la Figura 3.7. Ambas dimensiones poseen tres niveles jerárquicos.

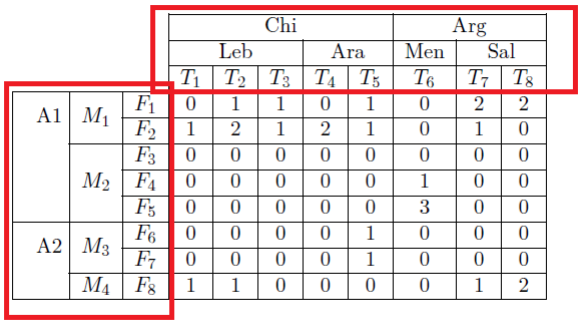


Figura 3.7: Dimensiones de la Figura 3.6

Ejemplo 3.5 El proceso de creación del árbol para la dimensión *Tiendas* comienza en el nodo raíz (por definición) el cual no se agrega a $d1$, luego se agrega el nivel 1 de la dimensión *Tiendas* al árbol, el cuál contiene dos elementos, $\{Chi, Arg\}$ y se almacena en d_1 la secuencia de bits 110. Como el nivel no tiene más elementos se baja al siguiente nivel. En el nivel 2 vemos que los elementos $\{Leb, Ara\}$ tiene relación con $\{Chi\}$ (elemento del nivel superior), por lo tanto pasarán a ser hijos de ese elemento en el árbol, y agregaremos a d_1 la secuencia de bits 110. Este proceso se repite para todos los elementos de la dimensión hasta agregarlos todos al árbol conceptual. El árbol y la secuencia de bits $d1$ para la dimensión *Tiendas* se presentan en la Figura 3.8 (a). Este proceso se repite para la dimensión *Tiempo*, la Figura 3.8 (b) muestra la representación LOUDS (árbol conceptual y secuencia de bits d_2) para esta dimensión.

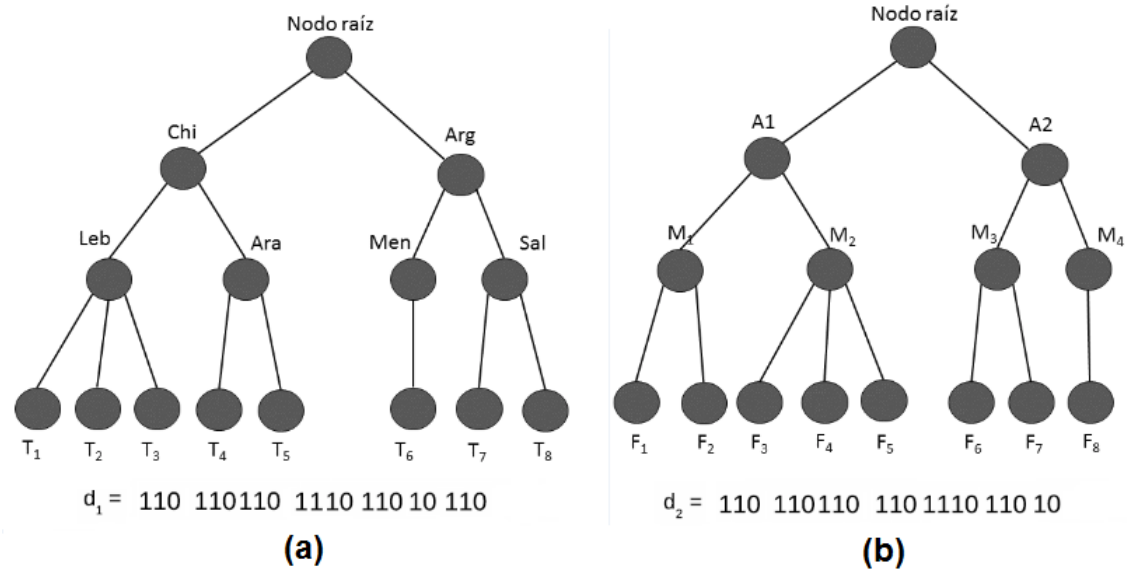


Figura 3.8: Representación LOUDS para las dimensiones *Tiempo* (a) y *Tiendas* (b) de la Figura 3.6

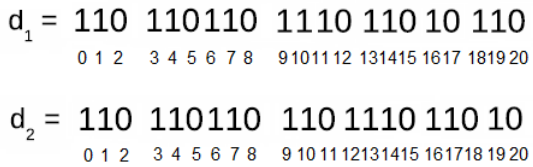


Figura 3.9: Tablas Hash para las secuencias d_1 y d_2

Terminado este proceso, se crea una tabla hash para cada secuencia (d_1 y d_2), donde cada elemento se asocia a una posición (Ver Figura 3.9). Estas se usarán para navegar por las dimensiones del CMHD usando las funciones *Rank* y *Select* (ver Sección 3.3).

- Representación de los datos:

  La Figura 3.10, presenta la matriz de la Figura 3.6 sub-dividida a partir de los niveles jerárquicos de sus dimensiones.

			Chi					Arg		
			Leb			Ara		Men	Sal	
			T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
A1	M_1	F_1	0	1	1	0	1	0	2	2
		F_2	1	2	1	2	1	0	1	0
	M_2	F_3	0	0	0	0	0	0	0	0
		F_4	0	0	0	0	0	1	0	0
		F_5	0	0	0	0	0	3	0	0
A2	M_3	F_6	0	0	0	0	1	0	0	0
		F_7	0	0	0	0	1	0	0	0
	M_4	F_8	1	1	0	0	0	0	1	2

Figura 3.10: Matriz sub-dividida por las jerárquias de las dimensiones

El nodo raíz del árbol almacena la suma total de ventas(datos) de la matriz. Luego, la matriz se sub-divide considerando la partición correspondiente al primer nivel jerárquico de las dimensiones (ver líneas rojas). Cada una de las sub-matrices se covertirá en un nodo hijo del nodo raíz, manteniendo la suma de los valores de los elementos en esa sub-matriz correspondiente. Este proceso se repite para cada nodo hijo, considerando los niveles sub siguientes de las jerarquías (ver líneas verdes), como se explicó, hasta llegar al último.

Este proceso concluye cuando todas las ramas del árbol alcanzan sub-matrices vacías (es decir, para este ejemplo, cuando no se registran mas ventas). El Ejemplo 3.6 y la Figura 3.11 ilustran este proceso. La Figura 3.13 muestra el árbol final para la Figura 3.10.

Ejemplo 3.6 Consideremos la matriz de la Figura 3.10, la cual nos presenta una matriz particionada de acuerdo a las jerarquías de sus dimensiones, el nodo raíz del árbol (ver Figura 3.12 (a)) almacenará la suma total de esta matriz (ver Figura 3.11 (a), sombreado amarillo), cuyo valor será 26. El siguiente valor almacenado corresponde a la primera sub-matriz (siempre de izquierda a derecha, desde arriba hacia abajo) de acuerdo a las particiones realizadas por las jerarquías (ver Figura 3.11 (b)), la suma total de esta sub-matriz es 10, este valor pasa a formar parte del

árbol (ver Figura 3.12 (b)), siendo un nodo hijo del nodo raíz, ya que, se encuentra en un nivel jerárquico menor. Este proceso se repite para las siguientes sub-matrices, Figura 3.11 (c), Figura 3.11 (d), Figura 3.11 (e) cuyos valores corresponden a 9, 4 y 3 respectivamente, todos estos valores son nodos hijos del nodo raíz (ver Figura 3.12 (c), Figura 3.12 (d) y Figura 3.12 (e)).

La Figura 3.11 (f), es una sub-matriz de una jerarquía menor a las anteriores y forma parte de la sub-matriz de la Figura 3.11 (b), por lo tanto esta corresponde a un nodo hijo para dicha matriz, la suma total de esta matriz es 6, este valor se almacena como un nodo en el árbol (ver Figura 3.12 (f)), este proceso se repite para las demas sub-matrices.

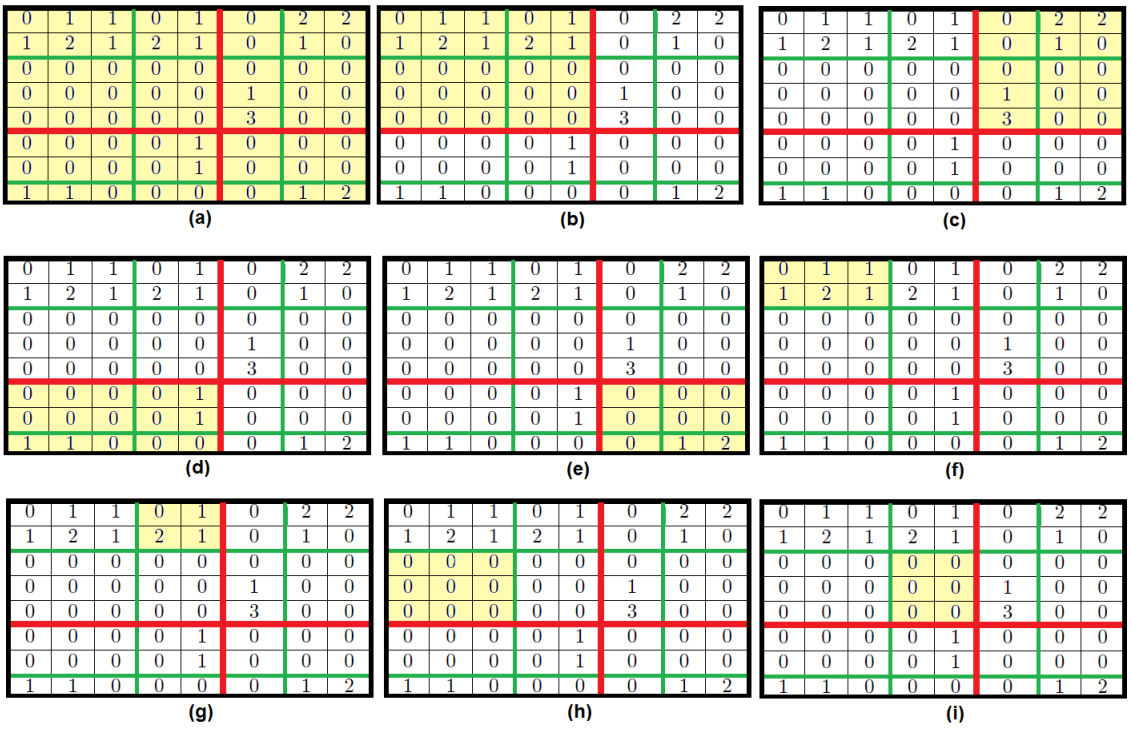


Figura 3.11: Matrices para el ejemplo 3.6

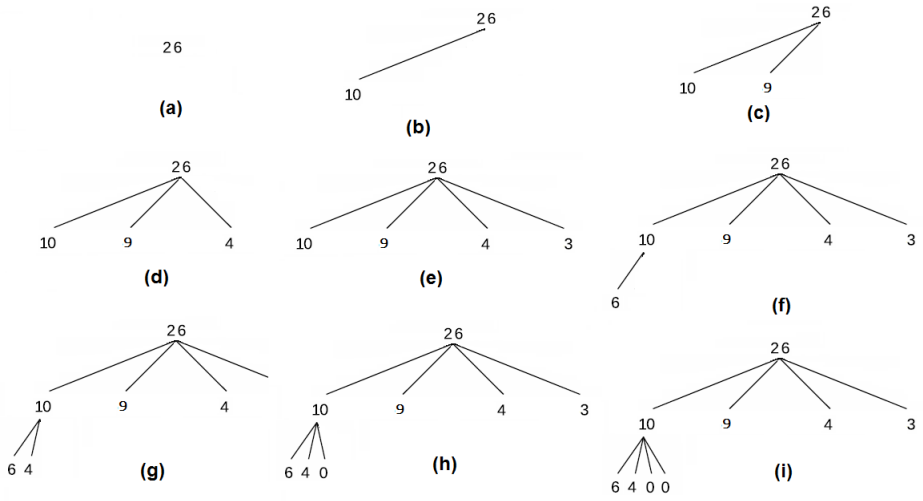


Figura 3.12: Árbol con función SUM resultante de la Figura 3.11

La Figura 3.13 presenta el árbol conceptual final para la matriz del CMHD, este contiene todas las agregaciones (sumas) realizadas en los nodos padres, mientras que en sus hojas almacena valores de las celdas correspondiente a la matriz.

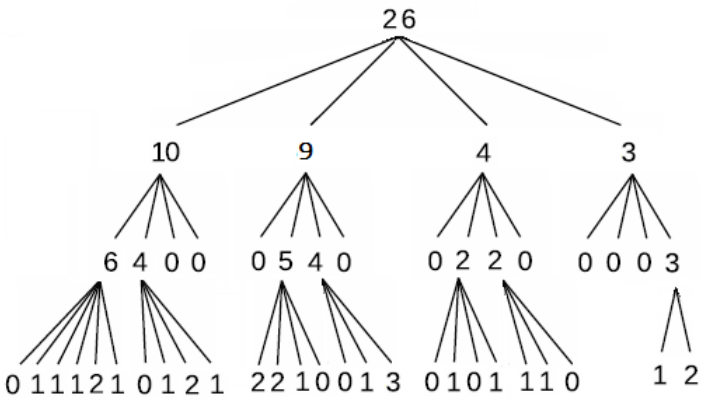


Figura 3.13: Árbol final para la Figura 3.10

Paralelamente con la creación del árbol conceptual para la matriz de la Figura 3.10 se crearon las siguientes estructuras de datos, las cuales se usan para representar dicha matriz de n dimensiones (en este caso 2 dimensiones).

- Estructuras de árbol (Ta y Tc): para navegar en el CMHD, necesitamos usar dos diferentes estructuras de datos en conjunto. Primero, Ta, un bitmap que proporciona una representación compacta del árbol conceptual independientemente de los valores de los nodos, para todos los niveles del árbol, excepto el último. Es decir, los nodos internos (padres) cuyo valor asociado es mayor que 0, se representarán con un 1. En otro caso, se etiquetarán con un 0. CMHD sigue diferentes particiones de jerarquía, lo que da como resultado sub-matrices irregulares. Por lo tanto, también se requiere una segunda estructura de datos, Tc, para navegar por el CMHD. Este es un bitmap alineado con Ta, que marca los límites de cada nodo del árbol en Ta (en este caso, se considera el último nivel de árbol). Si el siguiente nodo del árbol en Ta tiene Z hijos, agregamos $1^{Z-1}0$ a Tc. Por ejemplo, si Ta tiene 4 hijos ($Z = 4$), entonces $Tc = 1110$. El Ejemplo 3.7 y la Figura 3.14 ilustran la creación de estas estructuras. El Ejemplo 3.8 y la Figura 3.16 muestran como recorrer estas estructuras.
- $Valores(V)$: el CMHD se recorre por nivel guardando los valores asociados para cada nodo (ya sea un valor correspondiente a celdas de la matriz originales, o a agregaciones de datos(SUM) en una secuencia única V, que luego se representa con DAC (Brisaboa et al., 2013a). El Ejemplo 3.7 y la Figura 3.14 ilustran la creación de esta secuencia.

La Figura 3.15 muestra a Ta, Tc y V resultantes del árbol de la Figura 3.13. Estas se usarán para poder recorrer el árbol conceptual del CMHD al momento de generar una consulta.

Ejemplo 3.7 (Continuación Ejemplo 3.6), cuando se crea el árbol conceptual de la matriz para el CMHD, se crean 2 estructuras auxiliares para recorrer dicho árbol (Ta y Tc) y un vector (V) el cual almacena los valores del árbol creado. El primer valor del árbol (la raíz) se almacena directamente en V, por lo tanto, $V = 26$ que corresponde al valor de dicho nodo(ver Figura 3.14 (a)). Se recorre al siguiente nivel del árbol, el siguiente valor del nodo del árbol es 10, por lo tanto $Ta = 1$ ya que el valor es mayor que 0, $TC = 1$ por que no es el nodo final del nivel y $V = 10$ almacena el valor (ver Figura 3.14 (b)). El siguiente nodo del árbol tiene un valor de 9, por lo tanto $Ta = 1$, $Tc = 1$ y $V = 9$ (ver Figura 3.14 (c)). El siguiente valor del nodo es 4, por lo tanto, $Ta = 1$, $Tc = 1$ y $V = 4$ (ver Figura 3.14 (d)). Este proceso se repite para los demás nodos hasta recorrer todo el árbol.

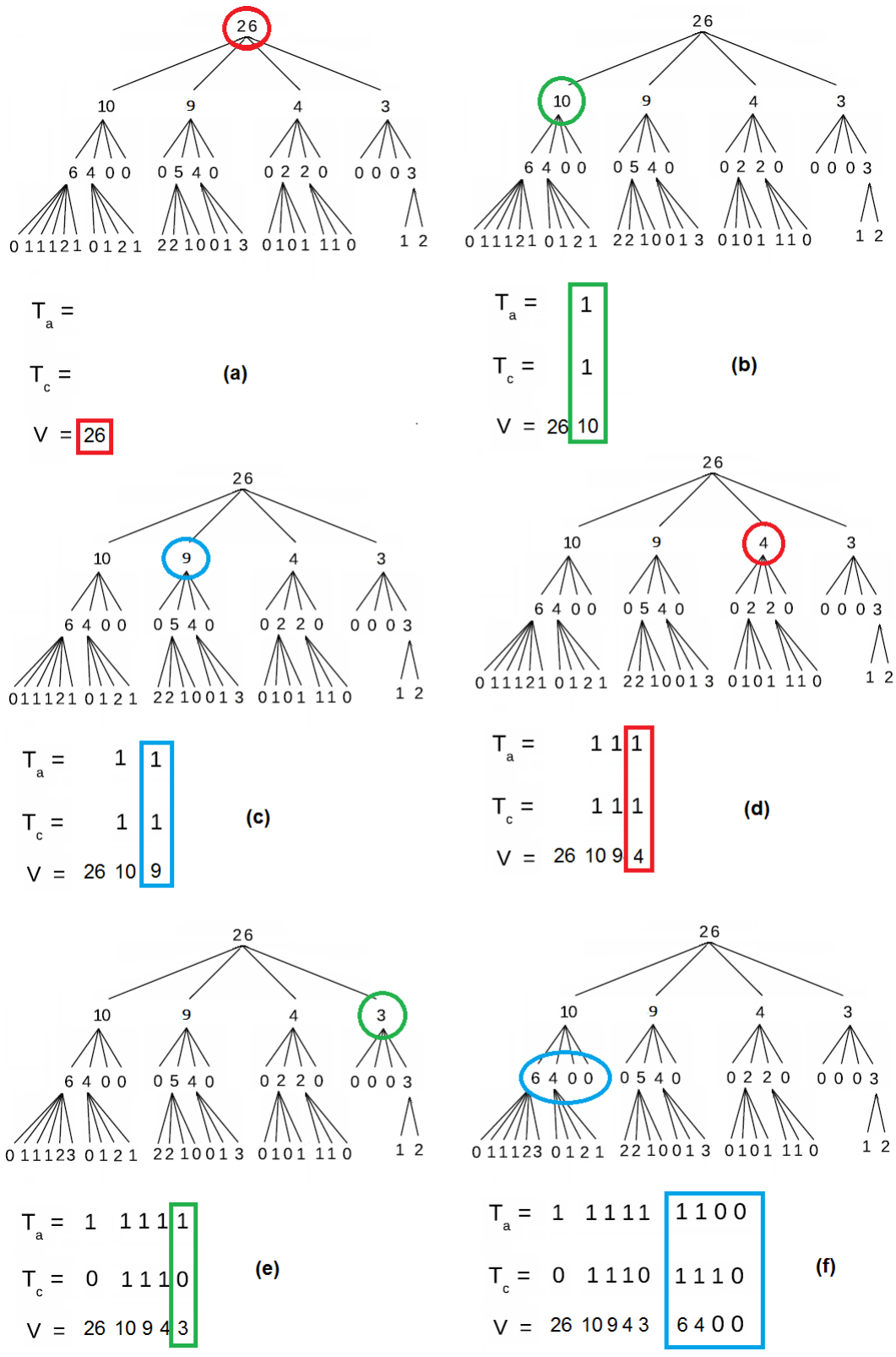


Figura 3.14: Proceso para almacenar Ta, Tc y V, a partir del árbol de la Figura 3.13

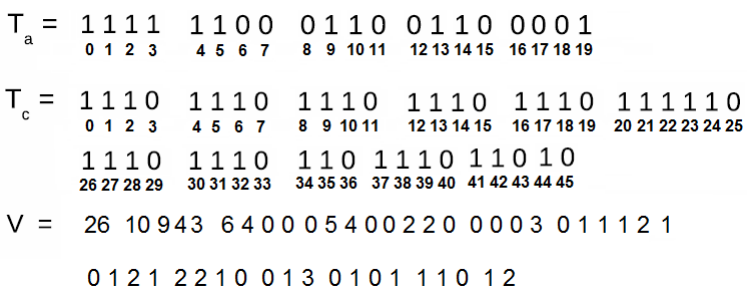


Figura 3.15: Ta, Tc y V resultantes para el árbol de la Figura 3.13

Ejemplo 3.8 Cada nodo de Ta está asociado con un 0 en Tc, lo que permite navegar por los árboles usando *Rank* y *Select* en Ta y Tc. Si decimos que estamos en un nodo en Ta que comienza en la posición i; entonces tiene un hijo $k-ésimo$ si $Ta[i+k-1] = 1$, y si es así, este hijo comienza en la posición $Select_0(Tc, Rank_1(Ta, i+k-1))+1$. Por ejemplo, si estamos en la posición 10 de Ta y consultamos por el primer hijo de este nodo ($k = 1$), entonces $Select_0(Tc, Rank_1(Ta, 10)) + 1 = Select_0(Tc, 8) + 1 = 33 + 1 = 34$, por lo tanto el primer hijo de $Ta[10]$ se encuentra en Tc[34], cuyo valor en V[34+1] = 0.

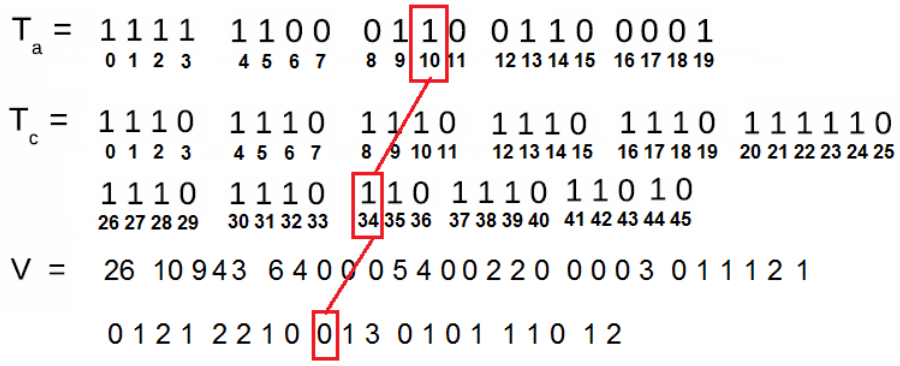
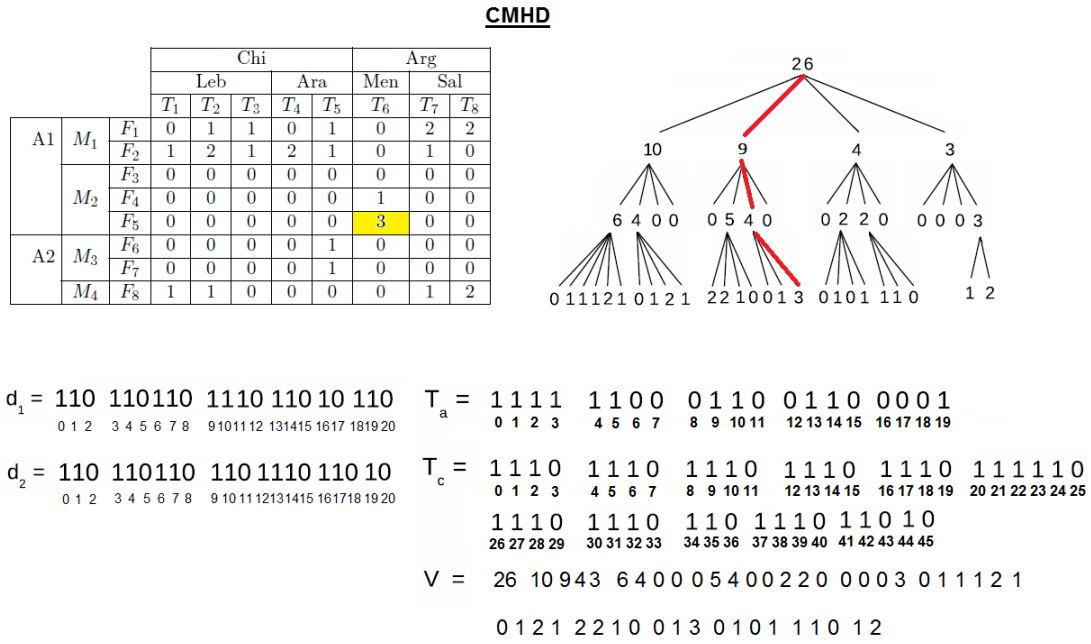


Figura 3.16: Ejemplo para navegar sobre Ta y Tc

Finalmente, la Figura 3.17 presenta el ejemplo de CMHD completo para una matriz de dos dimensiones, el cual se compone de una matriz, un árbol conceptual, y las deferentes estructuras para navegar por esta estructura de datos compacta.



			Chi					Arg		
			Leb			Ara		Men	Sal	
			T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
A1	M_1	F_1	0	1	1	0	1	0	2	2
		F_2	1	2	1	2	1	0	1	0
	M_2	F_3	0	0	0	0	0	0	0	0
		F_4	0	0	0	0	0	1	0	0
		F_5	0	0	0	0	0	3	0	0
A2	M_3	F_6	0	0	0	0	1	0	0	0
		F_7	0	0	0	0	1	0	0	0
	M_4	F_8	1	1	0	0	0	0	1	2

Figura 3.17: Ejemplo de CMHD para una matriz de dos dimensiones

3.4.2. Consultas sobre el CMHD

Inicialmente la estructura de datos compacta CMHD fue implementada para computar consultas de agregación con función SUM. La cual se obtiene de la siguiente manera. Las consultas sobre el CMHD reciben los nombres de los elementos de las diferentes dimensiones y pregunta por la suma de las celdas definidas para esos valores. Dependiendo de la consulta, podemos responderla simplemente informando un único valor agregado ya almacenado en V, o recuperando varios valores almacenados, y luego agregándolos (sumar). El primer escenario surge cuando los todos elementos de las diferentes dimensiones especificadas en la consulta se encuentran en el mismo nivel en sus respectivas jerarquías. Por ejemplo, si consultamos por las ventas realizadas en el país Chile *Chi* (Tiendas) y en el año $A1$ (Tiempo) (ver Figura 3.10). La segunda situación surge cuando la consulta realizada usa elementos de diferentes niveles de las dimensiones, por ejemplo si consultamos por las ventas realizadas en el país Chile *Chi* (Tiendas) y en la fecha $F1$ (Tiempo) (ver Figura 3.10).

En ambos contextos, se requieren recorridos descendentes a través del árbol conceptual del CMHD hasta llegar al nodo (o nodos) que se corresponden con el resultado. Entonces, CMHD tiene que saber por cuál rama tiene que descender en cada nivel del árbol, para eso, lo primero que hace CMHD es preguntar por los padres de cada uno de los elementos que conforman la consulta, por ejemplo si queremos obtener las ventas en la tiendas T_1 y en la fecha F_1, sabemos que T_1 es hijo de *Leb* y este es hijo de *Chile*, para F_1 sabemos que es hijo de M_1 y este a su vez es hijo de A_1. Ahora, con esta información podemos ir descendiendo por el árbol del CMHD. Siempre se comienza por la raíz del árbol, luego

para decidir qué rama se sigue en el proceso, se utilizan los elementos correspondientes a dicho nivel. Comenzamos en la raíz de Ta y descendemos al hijo número $k_i + a_i * k_j$, donde k_i es el hijo que sigue en la "i-ésima" dimensión, k_j es el hijo que sigue en la "j-ésima" dimensión y a_i es el número de nodos hijos de la raíz en la "i-ésima" dimensión para llegar al nodo consultado (a_i se calcula fácilmente con el árbol LOUDS de su dimensión). Por ejemplo para dos dimensiones tenmos que $k_1 + a_1 * k_2$. Continuamos de manera similar en el nodo del siguiente nivel, y así sucesivamente, hasta que lleguemos a uno de los nodos de la consulta en una dimensión, digamos que es primer nodo hallado. Para llegar al otro nodo en la segunda dimensión, debemos descender por cada hijo en la primera dimensión, en cada nivel, hasta llegar al segundo nodo consultado. Finalmente, cuando hemos alcanzado todos los nodos, recopilamos y sumamos los valores correspondientes de V. Si todos los nodos consultados están en el mismo nivel, realizamos un recorrido único en Ta. Tenga en cuenta también que, si encontramos un cero en un nodo de Ta a lo largo de este recorrido, inmediatamente terminamos el recorrido de esa rama, ya que su sub-matriz no contiene datos (esta vacía). El Ejemplo 3.9 presenta un ejemplo de consulta para el CMHD.

En el caso de consultas que combinan elementos de diferentes niveles jerárquicos, se aplicaría el mismo procedimiento, pero teniendo que obtener los valores correspondientes a todas las combinaciones posibles con el elemento del nivel de jerarquía más bajo (por ejemplo, si queremos obtener el número de ventas en Chile Chi para la fecha F_1, primero debemos recuperar todos los valores asociados con $\{F_1\}$: $\{F_1 - T_1\}$, $\{F_1 - T_2\}$, $\{F_1 - T_3\}$, $\{F_1 - T_4\}$ y $\{F_1 - T_5\}$, y luego agregar (sumar) todos los valores recuperados.

Ejemplo 3.9 Se quiere recuperar la cantidad total de ventas en la tienda T_6 para la fecha F_5 (ver Figura 3.17 celda amarilla). Como ambos elementos de la consulta pertenecen al mismo nivel, se debe recuperar un único valor almacenado en ese nivel. El camino para llegar a él se ha resaltado en el árbol conceptual (ver Figura 3.17 líneas rojas del árbol). Sabemos que la tienda T_6 es hijo de la ciudad Men (Mendoza) y éste es hijo del país Arg (Argentina), también sabemos que la fecha F_5 es hijo del mes M_2 y éste es hijo del año A_1. Por lo tanto, $\{T_6 - Men - Arg\}$ y $\{F_5 - M_2 - A_1\}$.

Para iniciar la búsqueda, debemos comenzar en la raíz del árbol Ta y tenemos que descender por el nodo que corresponde a los elementos Arg y A_1. Para saber a que nodo hijo de la raíz corresponde usamos la fórmula $k_1 + a_1 * k_2$, donde k_1 es el número de hijo del elemento de la primera dimensión (Arg), k_2 es el número de hijo del elemento de la segunda dimensión (A_1) y a_1 es el número de hijos de la raíz (padre) de la jerarquía de la primera dimensión. El elemento Arg es el hijo número 1 ($k_1 = 1$) (Chi sería el hijo 0), el elemento A_1 es el hijo número 0 ($k_2 = 0$) (A_2 es el hijo 1) y la raíz (padre) de la primera dimensión tiene un total de 2 hijos ($\{Chi, Arg\}$) ($a_1 = 2$). Por lo tanto, tenemos que descender por la rama $k_1 + a_1 * k_2 = 1 + 2 * 0 = 1$, donde $Ta[1] = 1$. Por lo tanto, accedemos a la posición 1 en Ta. Como $Ta[1] = 1$ (ver Figura 3.18 (b)), debemos continuar descendiendo al siguiente nivel.

Ahora, debemos calcular la posición de los hijos de $Ta[1]$, el hijo comienza en la posición $Select_0(Tc, Rank_1(Ta, 1))+1 = Select_0(Tc, 2)+1 = 8$ en Ta. Por lo tanto los hijos de $Ta[1]$ comienzan en la posición $Ta[8]$ (ver Figura 3.18 (c)). Utilizamos la fórmula $k_1 + a_1 * k_2$ para

saber por cual de sus hijos se tiene que descender. En este caso, Men es el hijo 0 ($k_1 = 0$) de Arg y M_2 es el hijo 1 ($k_2 = 1$) de $A1$, Arg tiene 2 hijos ($a_1 = 2$), por lo tanto $0 + 2 * 1 = 2$, el hijo que necesitamos se encuentra en la posición $Ta[8 + 2] = Ta[10]$, así que verificamos $Ta[10] = 1$ (ver Figura 3.19 (d)). De nuevo, como estamos en un nodo interno, sabemos que sus hijos están ubicados en la posición $Select_0(Tc, Rank_1(Ta, 10))+1 = Select_0(Tc, 8)+1 = 33+1 = 34$. Por lo tanto los hijos de $Ta[10]$ comienzan en la posición $Ta[34]$ (ver Figura 3.19 (e)). Finalmente, llegamos al tercer y último nivel del árbol, volvemos a utilizar $k_1 + a_1 * k_2$ para saber por cual de sus hijos se tiene que descender. T_6 es el hijo 0 ($k1 = 0$) de Men y F_5 es el hijo 2 ($k2 = 2$) de M_2, Men solo tiene 1 hijo ($a1 = 1$), por lo tanto $0 + 1 * 2 = 2$; $Ta[34 + 2] = Ta[36]$ (ver Figura 3.19 (f)), recordar que el ultimo nivel del árbol no se representa en Ta. Para realizar este último paso, examinamos directamente el arreglo $V[i + 1]$: $V[36 + 1] = V[37] = 3$.

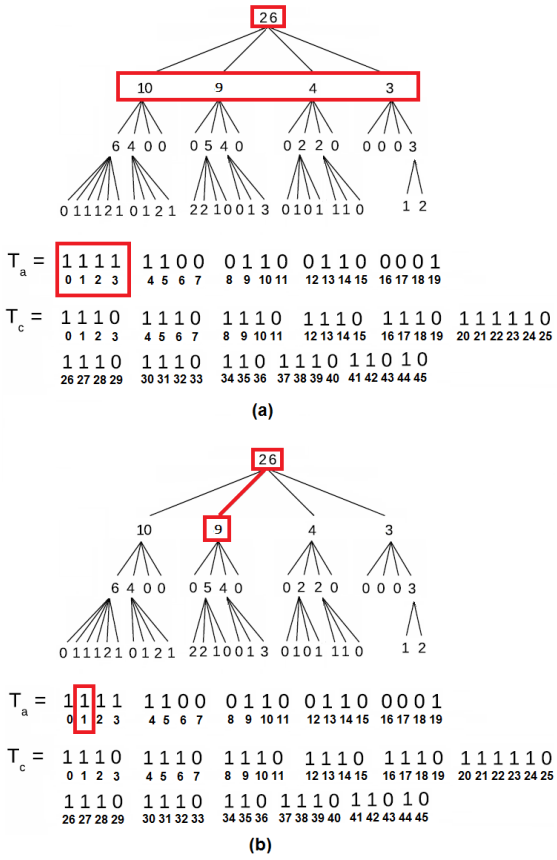


Figura 3.18: Ejemplo de consulta para CMHD (parte 1)

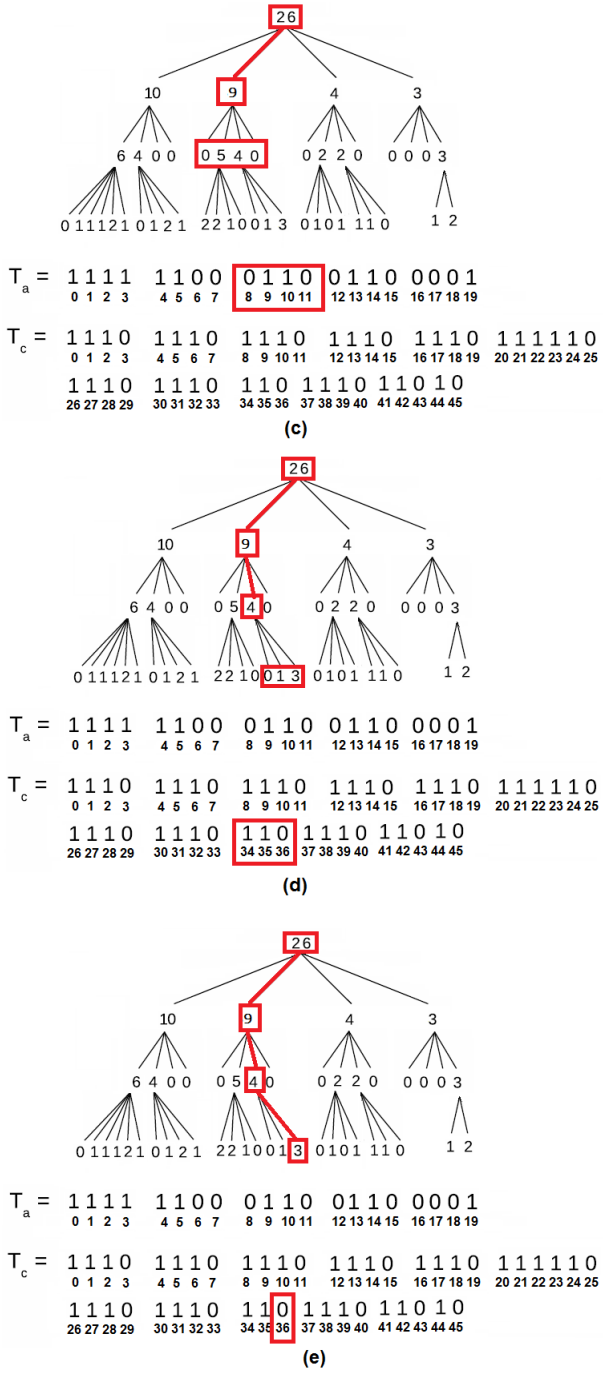


Figura 3.19: Ejemplo de consulta para CMHD (parte 2)

Capítulo 4

Algoritmos para almacenar y consultar en CMHD

Como se explicó anteriormente, CMHD inicialmente fue creado y probado solo para consultas de agregación con función SUM (Brisaboa et al., 2016). En este proyecto proponemos ampliar este tipo de consultas, y dar mayor cobertura a las diferentes consultas de agregación sobre DWs que usan funciones MAX, MIN, COUNT y AVG. Se presenta el algoritmo original con el cual CMHD almacena los valores agregados (valores sumados) del árbol conceptual en el arreglo V (ver Algoritmo 1), este algoritmo será modificado para satisfacer las diferentes consultas de agregación. Dependiendo de la consulta, podemos responderla simplemente informando un único valor agregado ya guardado en V, o recuperando varios valores almacenados, y luego agregándolos (ver Sección 3.4). El primer escenario surge cuando los elementos de las diferentes dimensiones especificadas en la consulta están todos en el mismo nivel en sus respectivas jerarquías. La segunda situación surge de consultas usando elementos de diferentes niveles. En ambos contextos, se requieren recorridos descendentes del CMHD conceptual para obtener los valores.

4.1. Algoritmos de almacenamiento para CMHD

El Algoritmo 1 presentado, corresponde al algoritmo original creado para CMHD, el cual guarda todos los valores sumados calculados en el arreglo V y sirve para ejecutar consultas con función de agregación SUM. Este algoritmo recibe como entrada una matriz, la cual puede ser una parte (celda) o la matriz entera que compone al cubo de datos que se quiere almacenar, esta matriz contiene un número de entradas y valores asociados para cada entrada, el algoritmo procesa esta matriz a traves de un ciclo for (ver linea 4 del algoritmo), donde se suman todos los valores de la matriz en un arreglo auxiliar ($values$) para luego pasar dicha suma total al vector V, el cual formará parte del CMHD final. La duración de este ciclo va a depender del número de entradas que posee la matriz. Este

proceso se repite hasta llegar a los nodos hojas que conforman el árbol final.

Algoritmo 1: Calcular la suma de los valores dentro del CMHD

```
   Input: Nueva entrada matriz tipo *Matriz y un arreglo V → position
   Output: Arreglo V → position con el valor de la suma total de la matriz
 1 /*Las variables se inician como vacías*/
 2 values[0, n] ← 0; /*arreglo auxiliar para calcular la suma*/
 3 /*Recibe la entrada tipo *Matriz y comienza a recorrerla */
 4 for i = 0; i < matriz → numEntries; i++ do
 5 |   /*Calcula la suma de todos los valores entrantes*/
 6 |   values[i] += matriz → value;
 7 |   matriz → nextEntry ;
 8 /*Guardar la suma total de los valores en el arreglo V*/
 9 /*Pregunta si V no está en la primera posición*/
10 if V → position != 0 then
11 |   V → nextposition;
12 V → position = values[i − 1];
13 return V;
```

Este algoritmo cálcula los valores para el árbol conceptual del CMHD, y almacena dichos valores sumados en el arreglo V, el cuál forma junto a Ta y Tc el árbol final del CMHD, en ningún momento modifica las demas estructuras de datos que conforman al CMHD. Un ejemplo para este algoritmo sería el Ejemplo 3.6 realizado anteriormente, donde se explica cómo generar el árbol con los valores sumados para el CMHD, la Figura 3.11 muestra cómo recorrer la matriz y la Figura 3.12 muestra cómo se va formando el árbol a medida que se recorre la matriz y se calculan las sumas correspondientes (cálculo que se realiza en este algoritmo). Finalmente la Figura 3.13 presenta el árbol final para el CMHD que almacena todas las sumas calculadas.

Los siguientes algoritmos permiten cambiar la forma en que el CMHD guarda la información de la matriz, de tal manera que permita realizar consultas de agregación para todos los operadores nombrados anteriormente, estos algoritmos se basan en el algoritmo original de CMHD, pero sufren una modificación a la hora de calcular los valores finales que se almacenarán en el vector V. Se genera un algoritmo por función de agregación y un nuevo arreglo V para cada uno. A continuación se presentan los algoritmos para los cálculos MAX, MIN, COUNT y AVG. El Algoritmo 2 sirve para calcular y almacenar los máximos en el vector V_{max} y el cual genera un nuevo árbol con nuevos valores para el CMHD. El Algoritmo 3, calcula todos los mínimos para el nuevo árbol del CMHD y los almacena en V_{min}. El Algoritmo 4 cuenta la cantidad de celdas no nulas de la matriz, estos nuevos valores formarán un nuevo árbol y se almacenarán en V_{count}. Finalmente, el Algoritmo 5 calcula el promedio de los valores dentro de la matriz, estos valores formarán un nuevo árbol y se almacenarán en V_{avg}. Todos los algoritmos trabajan de la misma manera, como se explicó para el algoritmo 1 de SUM, solo cambia el cálculo realizado. El ejemplo 4.1 ilustra cómo estos algoritmos cálculan los nuevos valores para cada uno de los nuevos

vectores Vs y formando así un nuevo árbol para cada función de agregación.

Algoritmo 2: Calcular los valores máximos dentro del CMHD

Input: Nueva entrada *matriz* tipo $*Matriz$ y arreglo $V_{max} \rightarrow position$
Output: Arreglo $V_{max} \rightarrow position$ con el valor máximo calculado de la *matriz*

```
1  /*Las variables se inician como vacías*/
2  values[0, n] ← 0; /*arreglo auxiliar para calcular el valor máximo de la matriz*/
3  /*Recibe la entrada tipo *Matriz y comienza a recorrerla */
4  for i = 0; i < matriz → numEntries; i++ do
5      /*Pregunta si es el primer valor y lo guarda en el arreglo*/
6      if i == 0 then
7          values[i] = matriz → value;
8      matriz → nextEntry ;
9      /*Calcular el valor máximo entre el valor previo y la nueva entrada*/
10     values[i] = MAX(values[i], matriz → value);
11 /*Guardar el valor máximo en el arreglo V_max*/
12 /*Pregunta si V_max no está en la primera posición*/
13 if V_max → position != 0 then
14     V_max → nextposition;
15 V_max → position = values[i − 1];
16 return V_max → position;
```

Algoritmo 3: Calcular los valores mínimos dentro del CMHD

Input: Nueva entrada *matriz* tipo $*Matriz$y arreglo $V_{min} \rightarrow position$
Output: Arreglo $V_{min} \rightarrow position$ con el valor mínimo calculado de la *matriz*

```
1  /*Las variables se inician como vacías*/
2  values[0, n] ← 0; /*arreglo auxiliar para calcular el valor mínimo de la matriz*/
3  /*Recibe la entrada tipo *Matriz y comienza a recorrerla */
4  for i = 0; i < matriz → numEntries; i++ do
5      /*Pregunta si es el primer valor y lo guarda en el arreglo*/
6      if i == 0 then
7          values[i] = matriz → value;
8      matriz → nextEntry ;
9      /*Calcular el valor mínimo entre el valor previo y la nueva entrada*/
10     values[i] = MIN(values[i], matriz → value);
11 /*Guardar el valor mínimo en el arreglo V_min*/
12 /*Pregunta si V_min no está en la primera posición*/
13 if V_min → position != 0 then
14     V_min → nextposition;
15 V_min → position = values[i − 1];
16 return V_min → position;
```

Algoritmo 4: Calcular la cantidad de elementos dentro del CMHD

Input: Nueva entrada *matriz* tipo $*Matriz$ y un arreglo $V_{count} \rightarrow position$
Output: Arreglo $V_{count} \rightarrow position$ con la cantidad de elementos de la *matriz*

```
1  /*Las variables se inician como vacías*/
2  numEntry ← 0 ;
3  /*Iniciamos la función */
4  for i = 0; i < matriz → numEntries; i++ do
5      /*Calcula la cantidad de valores entrantes*/
6      if matriz → value != 0 then
7          numEntry++;
8      matriz → nextEntry ;
9  /*Guarda la cantidad de elementos en el arreglo V_count*/
10 /*Pregunta si V_count no esta en la primera posición*/
11 if V_count → position != 0 then
12     V_count → nextposition;
13 V_count → position = numEntry;
14 return V_count → position;
```

Algoritmo 5: Calcular los promedios dentro del CMHD

Input: Nueva entrada *matriz* tipo $*Matriz$ y un arreglo $V_{avg}[position]$
Output: Arreglo $V_{avg}[position]$ con el valor promedio calculado de la *matriz*

```
1  /*Las variables se inician como vacías*/
2  numEntry ← 0 ;
3  values[0, n] ← 0; /*arreglo auxiliar para calcular el valor promedio de la matriz*/
4  /*Iniciamos la función */
5  for i = 0; i < matriz → numEntries; i++ do
6      /*Calcula la suma de todos los valores entrantes*/
7      values[i] += matriz → value;
8      /*Calcula la cantidad de valores entrantes*/
9      if matriz → value != 0 then
10         numEntry++;
11     matriz → nextEntry ;
12 /*Calculamos el promedio entre la suma previa y el número de entradas calculado*/
13 if numEntry != 0 and i != 0 then
14     values[i] = values[i − 1]/numEntry ;
15 /*Guarda el valor promedio calculado en el arreglo V_avg*/
16 /*Pregunta si V_avg no está en la primera posición*/
17 if V_avg → position != 0 then
18     V_avg → nextposition;
19 V_avg → position = values[i];
20 return V_avg → position;
```

Como se dijo anteriormente estos algoritmos siguen el mismo patrón que el algoritmo original del CMHD (ver Algoritmo 1), solo cambia el cálculo realizado para los valores, por lo que podemos usar el Ejemplo 3.6 para explicar el mismo proceso para cada algoritmo.

Ejemplo 4.1 Usando la Figura 3.11 podemos recorrer la matriz que se usó para crear el CMHD anterior y generar los nuevos árboles y vectores Vs para cada función de agregación respectivamente. La Figura 4.1 (a) muestra el nuevo árbol para el Algoritmo 2 (MAX), la Figura 4.1 (b) muestra el nuevo árbol para el Algoritmo 3 (MIN), la Figura 4.1 (c) muestra el nuevo árbol para el Algoritmo 4 (COUNT) y la Figura 4.1 (d) muestra el nuevo árbol

para el Algoritmo 5 (AVG). La Figura 4.2 muestra los vectores Vs resultante para cada algoritmo, estos vectores se utilizarán para procesar las consultas de agregación MAX, MIN, COUNT y AVG. Por lo tanto, el nuevo CMHD tendrá 5 vectores Vs, uno para cada agregación.

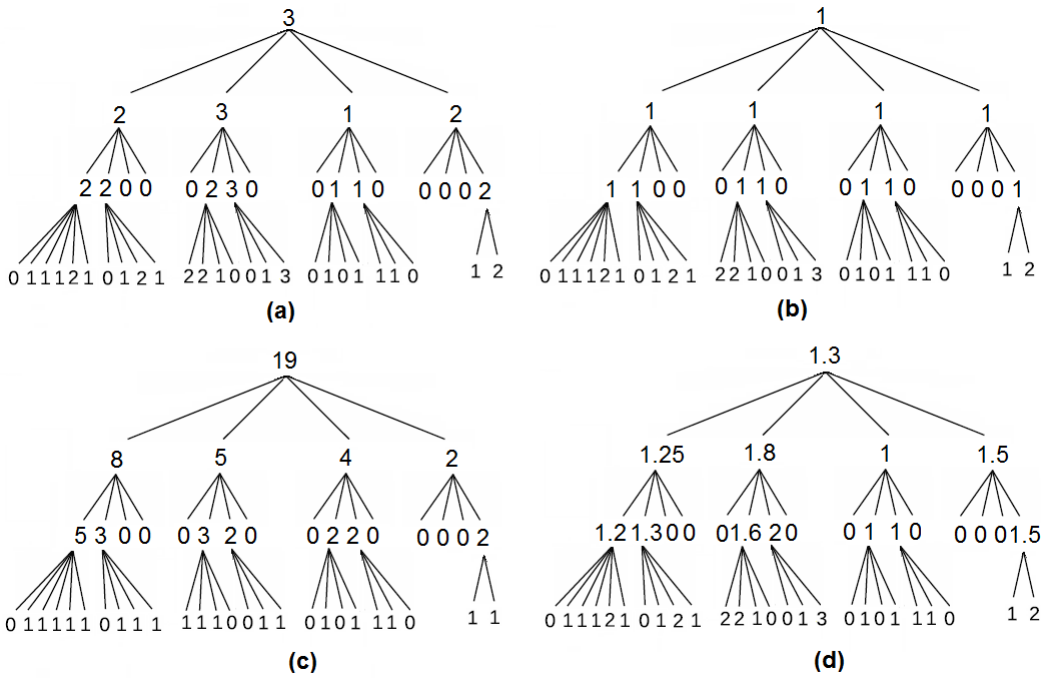


Figura 4.1: Matrices para el ejemplo 3.6

V_{max} = 3 2 3 1 2 2 2 0 0 0 2 3 0 0 1 1 0 0 0 0 2 0 1 1 1 2 1 0 1 2 1 2 2 2 1 0 0 1 3 0 1 0 1 1 1 0 1 2

V_{min} = 1 1 1 1 1 1 1 0 0 0 1 1 0 0 1 1 0 0 0 0 1 0 1 1 1 2 1 0 1 2 1 2 2 2 1 0 0 1 3 0 1 0 1 1 1 0 1 2

V_{count} = 19 8 5 4 2 5 3 0 0 0 3 2 0 0 2 2 0 0 0 0 2 0 1 1 1 1 1 0 1 1 1 1 1 0 0 1 1 0 1 0 1 1 1 0 1 1

V_{avg} = 1.3 1.25 1.8 1 1.5 1.2 1.3 0 0 0 1.6 2 0 0 1 1 0 0 0 0 1.5 0 1 1 1 2 1 0 1 2 1 2 2 2 1 0 0 1 3 0 1 0 1 1 1 0 1 2

Figura 4.2: Vectores Vs para los árboles de la Figura 4.1

4.2. Algoritmo para consultar sobre el CMHD

Existen dos escenario para las consultas en el CMHD, como se explicó anteriormente, si los elementos de la consulta se encuentran en un mismo nivel jerárquico para cada dimensión, solo se devolverá un valor agregado, en cambio si estas se encuentran en diferentes niveles jerárquicos, se devolverá un conjunto de valores, los cuales finalmente serán agregados para formar el valor final consultado. El algoritmo siempre comienza a buscar en las tablas hash las etiquetas (elementos) proporcionadas por la consulta para las diferentes dimensiones, para localizar los nodos LOUDS correspondientes. Desde los nodos LOUDS, recorremos cada jerarquía hacia arriba para conocer su profundidad y el nodo hijo que debe seguirse en cada nivel para alcanzarlo. Esta información luego se usa para encontrar los nodos deseados en Ta. Ta se recorre con la ayuda de Tc, cuando encontramos el nodo hijo cuyos elementos conforman la consulta, devolvemos esa posición y recuperamos el valor asociado en el vector V correspondiente a la agregación que elegimos.

Algoritmo 6: Computar consultas sobre el CMHD

Input: Un DW compactado por CMHD $CMHD$, un número de dimensiones N, una agregación $Agre$ y una consulta Q
Output: $V[Ta+1]$ con el valor obtenido

```
1  Aux ← 0;
2  Ta[0, n] ← 0;
3  V[Ta + 1] ← 0;
4  i = 0;
5  /*Iniciamos la función */
6  while Ta → value ! = 0 do
7  |   Aux[i] = HASH(N, CMHD → D_N, Q → D_N) ;
8  |   Aux[i + 1] = CMHD → Ta[Aux[i]] → [CMHD → Tc[Aux[i]]];
9  |   if Ta[Aux[i + 1] == 0 then
10 |   |   return V[1] = 0;
11 |   end
12 |   Ta[Aux[i + 1]] = HASH_Agre(CMHD → Ta, Ta[Aux[i]]) ;
13 |   if Ta[Aux] ! = CMHD → Ta → position then
14 |   |   V[Ta + 1] = CMHD → V_{Q_Agre}[Ta + 1] ;
15 |   |   return V[Ta + 1];
16 |   end
17 |   i++;
18 end
```

Ejemplo 4.2 Este algoritmo fue el que utilizamos en el Ejemplo 3.9, en el cual consultamos por elementos en un mismo nivel jerárquico. Para poder consultar por las diferentes agregaciones basta con poner una etiqueta dentro de la consulta, la cual nos indique qué tipo de consulta se desea realizar, Por ejemplo, si queremos responder al Ejemplo 3.9 nuevamente, pero esta vez queremos los máximos o mínimos, entonces usaremos la posición ya calculada en Ta pero ahora buscaremos el valor deseado en el V que corresponda para cada agregación. Si queremos recuperar las ventas máximas para la consulta T_6 y la fecha F_5, como ya se cálculo la posición $Ta[36]$, podemos consultar en $V_{max}[36+1] = 3$.

Capítulo 5

Desarrollo e Interfaz de los sistemas Web y Móvil

Este capítulo describe las herramientas que se utilizaron para la creación del Sistema Web y el Sistema para dispositivos móviles, se presentarán además capturas de pantalla para ambos sistemas, con la finalidad de mostrar su funcionalidad.

Si bien, este proyecto consta tanto de un sistema WEB como de un sistema para dispositivos móviles, la finalidad de estas es la misma: Permitir diversas consultas sobre DWs almacenados en la estructura de datos compacta CMHD, para realizar consultas de agregación con funciones SUM, MAX, MIN, COUNT y AVG.

5.1. Herramientas utilizadas

A continuación se presentan las herramientas utilizadas para desarrollar el sistema Web y el sistema para dispositivos móviles.

5.1.1. PHP

PHP es un lenguaje de programación de código abierto diseñado por Rasmus Lerdorf en el año 1994, el cual es utilizado principalmente para desarrollo Web y puede ser incrustado en HTML. PHP representa un acrónimo recursivo para *PHP Hypertext Pre-Processor*[1]. Este lenguaje se suele procesar directamente en un servidor, el cual genera código HTML que puede ser enviado a una aplicación cliente.

Shellexec de PHP permite ejecutar un comando de consola mediante el intérprete de comandos y devolver la salida completa como una cadena, esto nos permite mandar comandos a la consola de Linux por medio del sistema Web desarrollado. En este caso *shellexec* será la principal herramienta para llamar a los programas alojados en el servidor web. De tal forma que el usuario no se de cuenta que haciendo un simple click en un

[1] http://php.net/manual/es/intro-whatis.php

botón puede ejecutar diversos comandos de Linux de una manera practica y sencilla. A continuación se presenta un ejemplo de *shellexec*:

```php
<?php
$salida = shell_exec('ls -lart');
echo "<pre>$salida</pre>";
?>
```

Figura 5.1: Ejemplo de la función shell

La Figura 5.1 muestra cómo *shellexec* ejecuta el comando $ls-lart$, que sirve para listar el contenido de un directorio y ver toda su información correspondiente. De esta misma forma podemos usar esta función de PHP para llamar y ejecutar los programas alojados en el servidor web que ejecutaran las consultas sobre DWs almacenados en la estructura de datos compacta CMHD.

5.1.2. HTML

HTML por su parte, del inglés *HyperText Markup Language*[2], es decir, Lenguaje de Marcas de Hipertexto, es un lenguaje de marcado utilizado para la elaboración de páginas Web. HTML es un lenguaje que sirve para describir la estructura y organización de una página, y la forma en que se muestra su contenido (ya sea texto, imágenes, videos, entre otros), además de incluir enlaces (*links*) hacia otras páginas o documentos. Su estructura básica la podemos visualizar en la Figura 5.2.

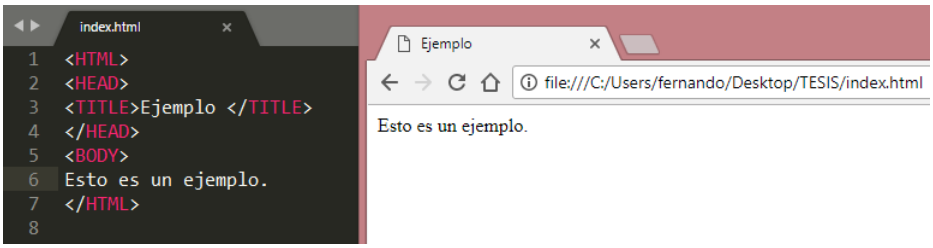


Figura 5.2: Ejemplo de una estructura básica HTML

5.1.3. Sistema operativo Android

Si bien en la actualidad existe una diversa cantidad de sistemas operativos móviles, se escogió Android para almacenar el sistema móvil de este proyecto. Android es un sistema operativo para dispositivos móviles que está basado en Linux, este ofrece edición de códigos de primer nivel, depuración, herramientas de rendimiento, un sistema de compilación flexible y un sistema instantáneo de compilación e implementación, además de poseer un lenguaje de código abierto. En cuanto al entorno de desarrollo (IDE), utilizamos el software oficial para la plataforma Android llamado Android Studio (En su versión 3.0.1.), el

[2]https://developer.mozilla.org/es/docs/Web/HTML

cual trabaja en lenguaje Java. Para el desarrollo de la aplicación móvil, se contempló su funcionabilidad para versiones de Android 3.5.2 en adelante. La interfaz del sistema creado se visualiza en la Figura 5.5.

5.2. Interfaz de las aplicaciones

El sistema Web para el CMHD cuenta con una interfaz desarrollada en HTML y PHP, amigable y fácil de usar para el usuario. Este sistema contiene un DW de tres dimensiones pre-cargado, el cual se encuentra almacenado dentro de la estructura de datos compacta CMHD, por lo tanto está disponible para cualquier tipo de consulta de agregación.

La Figura 5.3 muestra la interfaz del sistema WEB, en el cual se debe seleccionar entre los DWs almacenados en el sistema, en este caso, solo se almaceno un DW de tres dimensiones. La Figura 5.4 muestra la interfaz para el DW de tres dimensiones, el sistema muestra todos los cubos de datos almacenados y todas las dimensiones disponibles para el DW. El sistema lista los cubos de datos y todos los elementos para cada dimensión del DW, estos elementos se deben elegir manualmente al igual que la función de agregación que se quiera utilizar para formar una consulta. El sistema permite elegir entre cinco tipos de funciones de agregación: SUM, MAX, MIN, COUNT y AVG, la cuál junto al cubo de datos y a los elementos seleccionados para cada dimensión formarán una consulta. La Figura 5.5 muestra la interfaz del sistema Móvil, la cual posee la misma funcionalidad que el sistema Web.

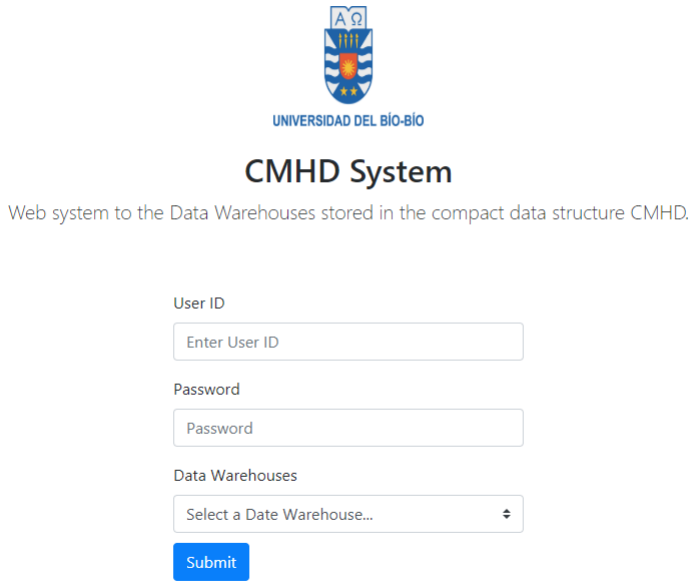


Figura 5.3: Interfaz del sistema Web

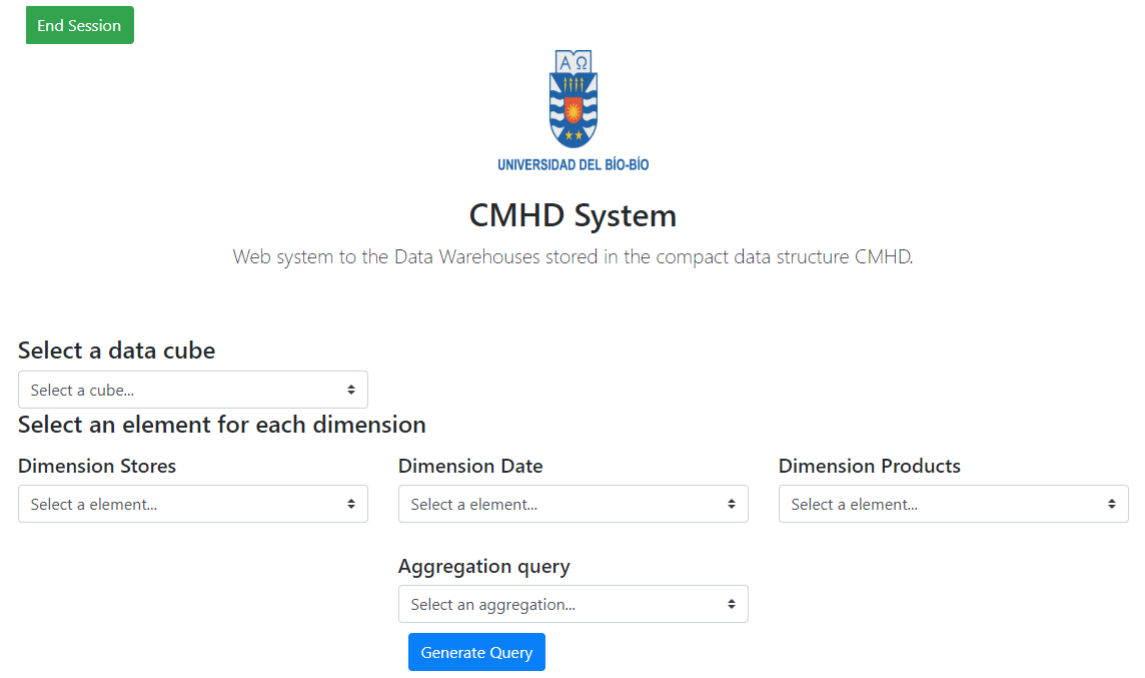


Figura 5.4: Interfaz del sistema WEB para DW de 3 dimensiones

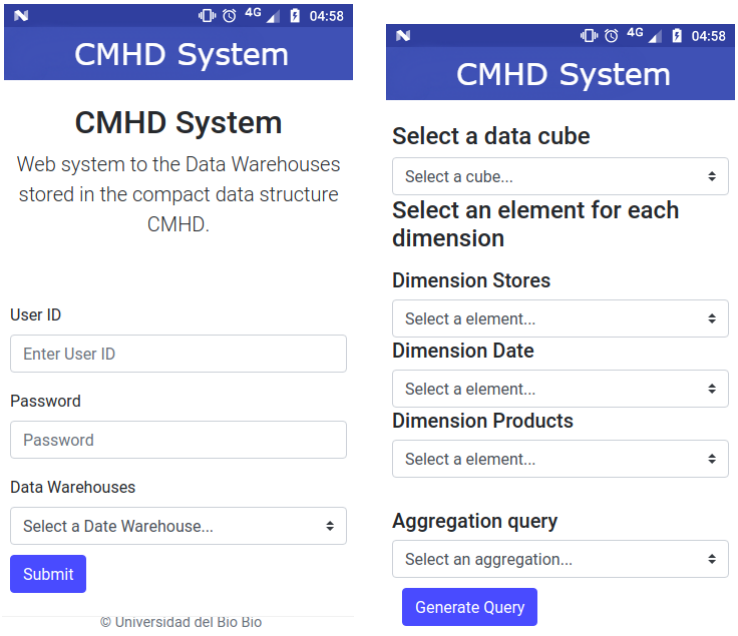


Figura 5.5: Interfaz del sistema Móvil

Ejemplo 5.1 La Figura 5.6, muestra un ejemplo de cómo ejecutar una consulta en el sistema Web. El sistema despliega una lista de los cubos de datos y de todos los elementos para cada dimensión, primero debemos seleccionar un cubo de datos y a continuación un elemento por dimensión, además de una función de agregación para realizar la consulta correspondiente. En este caso, la consulta corresponde a el maximo (MAX) valor de ventas para el cubo de datos *Sales*16, para todas las *Ciudades*, *Años* y *Marcas*. Una vez seleccionado todos los elementos para la consulta, debemos dar click al botón *GenerateQuery*, el cual enviará la consulta al sistema, este la procesa y nos muestra una tabla con el resultado obtenido. La Figura 5.7 muestra el resultado obtenido para esta consulta.

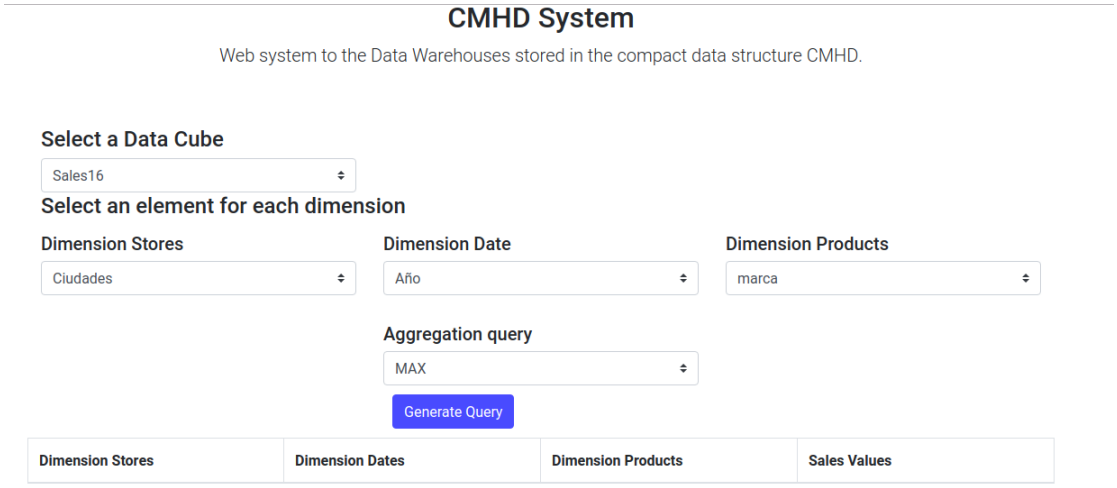
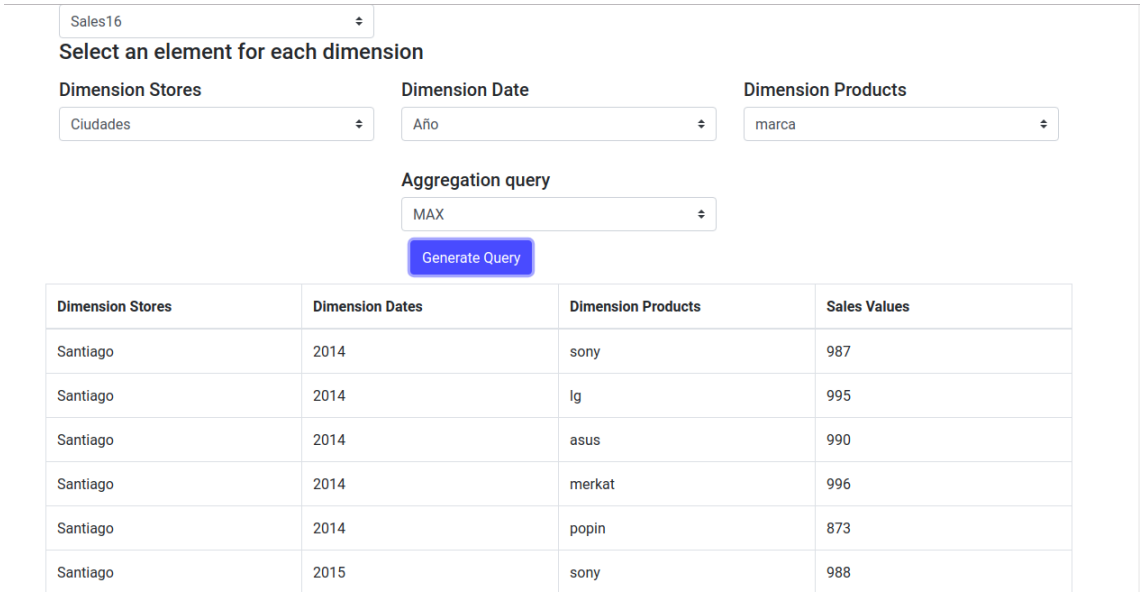


Figura 5.6: Consulta sobre el sistemas Web



Dimension Stores	Dimension Dates	Dimension Products	Sales Values
Santiago	2014	sony	987
Santiago	2014	lg	995
Santiago	2014	asus	990
Santiago	2014	merkat	996
Santiago	2014	popin	873
Santiago	2015	sony	988

Figura 5.7: Respuesta para la consulta de la Figura 5.6 en el sistema Web

Capítulo 6

Experimentación

En este capítulo se presenta la experimentación realizada a partir de las distintas pruebas sobre los cubos de datos de los DWs, se presentan pruebas para mostrar la reducción de espacio de almacenamiento y la eficiencia de tiempos de respuestas para las diferentes consultas de agregación realizadas. Esto para comparar la eficiencia de la estructura de datos compacta CMHD frente al SGBD tradicional MySQL. En la Sección 6.1 se presenta el hardware utilizado para desarrollar los experimentos, en la Sección 6.2 se describe el escenario de experimentación, finalmente en las Secciones 6.3 y 6.4 se presentan los experimentos y pruebas realizadas en este proyecto.

6.1. Hardware

Para especificar el hardware a utilizar, se tomaron en cuanta diferentes elementos: El procesador o CPU del cual dispone el dispositivo, la cantidad de memoria RAM, el espacio de almacenamiento disponible, y el sistema operativo utilizado. Los experimentos de ejecutaron en un ordenador con procesador Intel Core i3-6000u - 2.0GHz (2 núcleos físicos), con 8 GB de memoria RAM, 1 TB de disco duro y sistema operativo Ubuntu 16.04 de 64 Bits. Todos los algoritmos para el CMHD fueron creados en C++ y compilados con g++/gcc versión 4.7, el sistema Web fue desarollado en HTML y PHP. El dispositivo móvil, modelo Samsung Galaxy J5, posee un procesador Cortex A53 - 1.20GHz (4 núcleos), memoria RAM de 1.5 GB, sistema operativo Android 5.1.

6.2. Escenario de experimentación

Se consideraron el DW presentado en la Figura 6.1 de tres dimensiones. Este fue implementado en el SGBD MySQL versión 4.5.1, usando el esquema copo de nieve (Chaudhuri y Dayal, 1997), el cual permite la representación de las jerarquías de las dimensiones. La Figura 6.1 muestra el esquema de tres dimensiones.

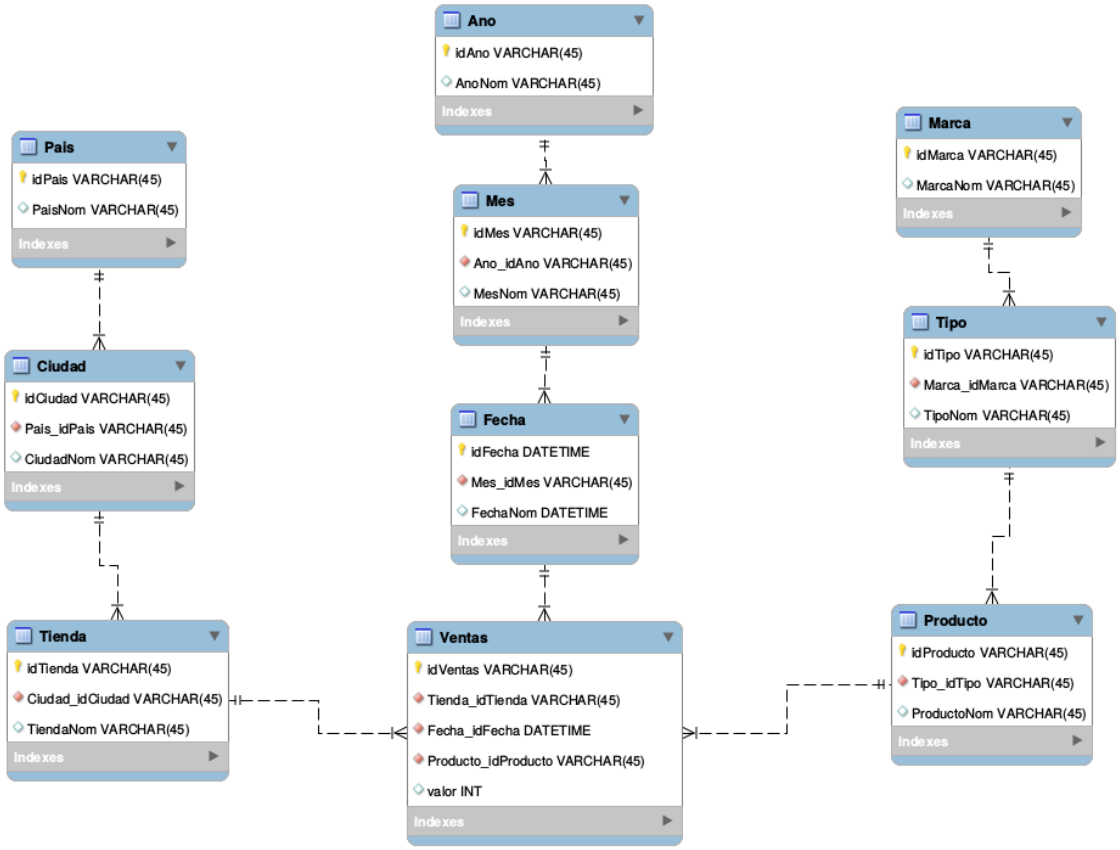


Figura 6.1: Esquema copo de nieve para un DW de tres dimensiones

Todos los datos generados para estas pruebas son datos sintéticos y fueron generados con una distribución uniforme. Esta distribución permite generar un número finito de valores con la misma probabilidad, los valores usados están en el rango de 0 a 1000. Se generaron 4 cubos de datos: los cubos de datos de 3 dimensiones que poseen 16, 32, 64 y 96 elementos en su nivel más bajo para cada dimensión. Por ejemplo para el cubo de 3 dimensiones con 16 elementos, quiere decir que el tamaño total del cubo es de $16x16x16 = 4096$ elementos. La Tabla 6.1, muestra los cubos de datos generados y sus respectivos tamaños.

6.3. Pruebas de almacenamiento

Para medir el poder real de compactación por parte del CMHD y la eficiencia en espacio de almacenamiento ocupado en kilobytes (Kb), se realizaron pruebas con cubos de datos de diferentes tamaños, comparando el tamaño de estos almacenados en el SGBD MySQL y su tamaño final una vez compactados por la estructura de datos compacta CMHD. La Tabla 6.1 muestra los cubos que fueron generados, considerando la cantidad de elementos que

tienen y su peso final en Kb. La Figura 6.2 muestra que que la representación compacta de los cubos de datos ahorra una cantidad considerable de espacio, para el primer caso CMHD logra compactar hasta un 97 % del tamaño del cubo de datos en comparación a MySQL. Además vemos que esta tendencia se mantiene a mayor tamaño de los cubos de datos, para los últimos casos el CMHD es capaz de compactar casi un 96 % del peso real de los cubos de datos. Todo esto siempre guardando la relación del peso y la cantidad de elementos que posee cada cubo de datos. Podemos notar el enorme poder para compactar los cubos de datos que posee la estructura CMHD, esto viene a reforzar la idea de los beneficios que poseen las estructuras de datos compactas en cuanto al espacio de almacenamiento utilizado.

Rendimiento de la compactación		
# Elementos	SGBD (Kb)	CMHD (Kb)
4,096(16x16x16)	736	27,8
32,768(32x32x32)	6,300	205,4
262,144(64x64x64)	48,300	1,500
884,736(96x96x96)	155,100	4,500

Tabla 6.1: Tamaño de los cubos de datos generados

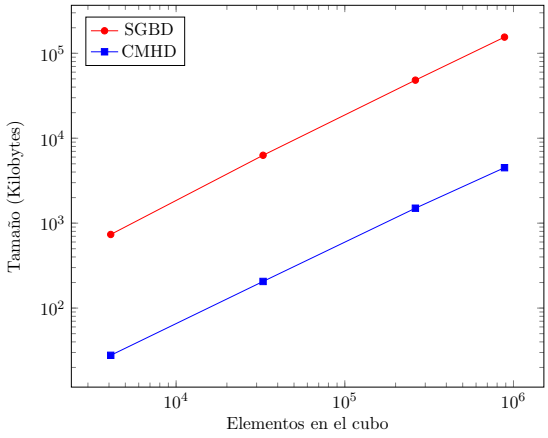


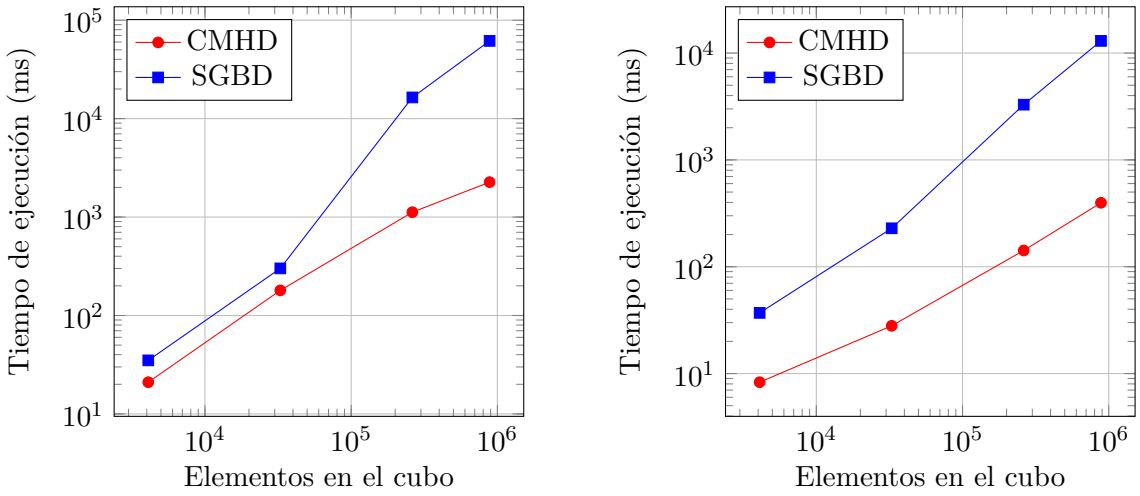
Figura 6.2: Espacio requerido para cada cubo de datos

6.4. Pruebas en rendimiento de consultas

Las Tablas 6.2, 6.3 , 6.4, 6.5 y 6.6 muestran, respectivamente, los tiempos de ejecución para todas las consultas de agregación con funciones SUM, MAX, MIN, COUNT y AVG, ejecutadas sobre los cubos de datos para el DW de la Figura 6.1. Ademas desde la Figura 6.3 hasta la Figura 6.22 se grafican, respectivamente, las consultas mas reletantes, en cuanto a tiempos de ejecución para el CMHD, esto para cada consulta con función de agregación realizada.

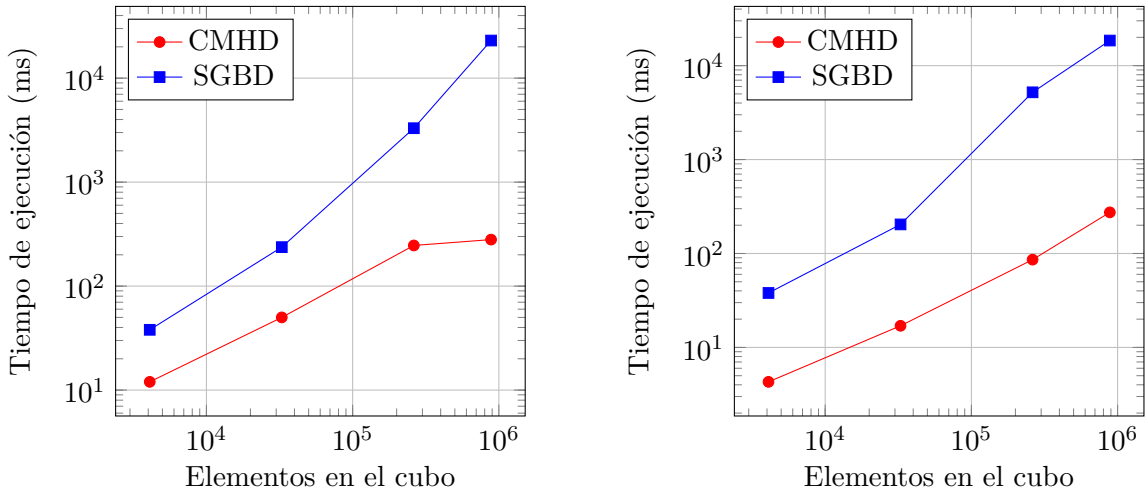
	# Elementos en los cubos							
	4096		32768		262144		884736	
Nivel de consulta	CMHD	SGBD	CMHD	SGBD	CMHD	SGBD	CMHD	SGBD
Tienda - Fecha - Producto	21	35	180	301	1,120	16,420	2,263	61,600
Tienda - Fecha - Tipo	9.7	50	40	264	218	21,050	547	40,000
Tienda - Fecha - Marca	8.5	31	37	247	186	18,730	505	35,400
Tienda - Fecha - All	6.2	20	32	157	107	1,892	314	10,200
Tienda - Mes - Producto	9.5	38	41	227	200	3,700	553	19,400
Tienda - Mes - Tipo	5	40	21	250	76	19,454	226	45,000
Tienda - Mes - Marca	7	45	24	245	73	18,500	205	40,000
Tienda - Mes - All	4	17	15	132	80	18,590	246	52,740
Tienda - Año - Producto	8.3	37	28	229	142	3,290	397	13,000
Tienda - Año - Tipo	4.8	43	16	221	81	3,990	205	15,310
Tienda - Año - Marca	4.5	46	20	246	80	4,040	202	15,700
Tienda - Año - All	3.2	19	16	153	86	4,100	222	16,000
Tienda - All - Producto	6.7	28	35	144	101	2,400	297	11,320
Tienda - All - Tipo	3.7	32	18	174	69	16,300	259	45,400
Tienda - All - Marca	3.5	24	17	172	75	16,500	215	46,300
Tienda - All - All	2.9	15	16	96	114	16,450	231	45,350
Ciudad - Fecha - Producto	12	38	50	237	246	3,300	280	23,000
Ciudad - Fecha - Tipo	5.9	39	19	238	106	4,460	244	40,770
Ciudad - Fecha - Marca	4.9	43	24	213	81	4,650	245	35,600
Ciudad - Fecha - All	4.8	19	18	118	114	4,700	217	36,300
Ciudad - Mes - Producto	5.3	29	20	208	89	4,900	242	13,850
Ciudad - Mes - Tipo	1.7	53	2.5	285	2.2	20,950	2.7	15,000
Ciudad - Mes - Marca	2.2	38	1.4	241	2.4	18,635	1.6	45,760
Ciudad - Mes - All	0.78	30	0.8	114	0.84	2,930	0.9	12,400
Ciudad - Año - Producto	4.3	38	17	204	86	5,200	274	18,500
Ciudad - Año - Tipo	1.4	48	1.4	207	1.5	5,300	1.4	20,370
Ciudad - Año - Marca	1.2	51	1.2	201	1.3	3,800	1.3	15,620
Ciudad - Año - All	0.47	20	0.48	141	0.51	2,400	0.5	13,800
Ciudad - All - Producto	4.7	26	21	141	136	2,800	352	12,350
Ciudad - All - Tipo	0.69	22	0.7	141	0.77	3,000	0.8	14,740
Ciudad - All - Marca	0.61	23	0.6	130	0.68	3,700	0.8	14.260
Ciudad - All - All	0.033	16	0.3	106	0.34	1,900	0.36	8,000
Pais - Fecha - Producto	13	31	32	216	167	3,200	447	13,100
Pais - Fecha - Tipo	5.5	42	18	228	74	3,700	258	14,200
Pais - Fecha - Marca	5.5	38	21	208	79	3,700	229	14,000
Pais - Fecha - All	3.4	21	19	141	90	2,300	222	12,400
Pais - Mes - Producto	4.4	35	18	232	83	3,650	235	14,620
Pais - Mes - Tipo	0.9	38	1.3	250	1.1	3,900	1	18,000
Pais - Mes - Marca	1	58	1.1	238	1	3,950	1.1	15,800
Pais - Mes - All	0.4	20	0.43	132	0.44	2,800	0.5	13,650
Pais - Año - Producto	4.6	43	16	239	77	3,400	230	15,400
Pais - Año - Tipo	0.7	61	0.7	256	0.6	3,600	0.72	15,400
Pais - Año - Marca	0.4	119	0.3	240	0.4	3,850	0.42	16,300
Pais - Año - All	0.1	28	0.1	152	0.1	2,600	0.11	12,550
Pais - All - Producto	3.6	27	14	152	112	2,770	302	16,000
Pais - All - Tipo	0.37	33	0.3	181	0.39	3,140	0.4	15,200
Pais - All - Marca	0.16	21	0.16	157	0.17	3,300	0.18	17,300
Pais - All - All	0.048	16	0.05	104	0.07	1,870	0.05	11,700
All - Fecha - Producto	7.9	24	27	178	116	20,900	340	55,760
All - Fecha - Tipo	4.4	26	20	201	80	18,000	243	49,400
All - Fecha - Marca	4.1	28	19	175	77	17,700	261	42,150
All - Fecha - All	3.4	14	18	87	131	2,070	246	20,500
All - Mes - Producto	4.1	30	18	194	81	2,800	288	15,400
All - Mes - Tipo	0.61	27	0.6	225	0.7	20,000	0.67	50,640
All - Mes - Marca	0.54	27	0.27	203	0.57	16,800	0.59	45,200
All - Mes - All	0.3	14	0.3	107	0.31	2,200	0.31	25,000
All - Año - Producto	3.6	23	19	186	63	2,870	255	11,200
All - Año - Tipo	0.39	29	0.38	177	0.19	3,320	0.2	15,500
All - Año - Marca	0.15	31	0.081	189	0.21	3,095	0.18	14,300
All - Año - All	0.05	15	0.042	101	0.05	2,070	0.053	12,950
All - All - Producto	3.3	20	18	105	114	19,490	252	13,720
All - All - Tipo	0.25	15	0.26	83	0.28	17,300	0.29	15,000
All - All - Marca	0.03	12	0.07	90	0.07	15,970	0.08	15,400
All - All - All	0.008	3.3	0.008	21	0.008	390	0.009	1870

Tabla 6.2: Tiempos de ejecución en milisegundos (ms) en consulta SUM para cubos de datos de tres dimensiones



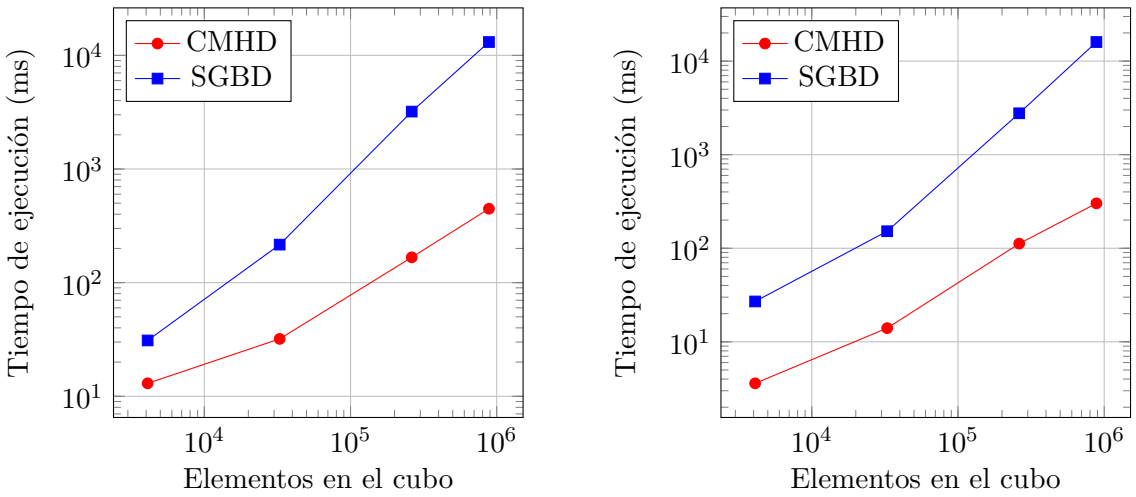
(a) Consulta agrupada por Tienda-Fecha-Producto (b)Consulta agrupada por Tienda-Año-Producto

Figura 6.3: Tiempos de ejecución para consultas de agregación SUM agrupado por los niveles Tienda-Fecha-Producto y Tienda-Año-Producto



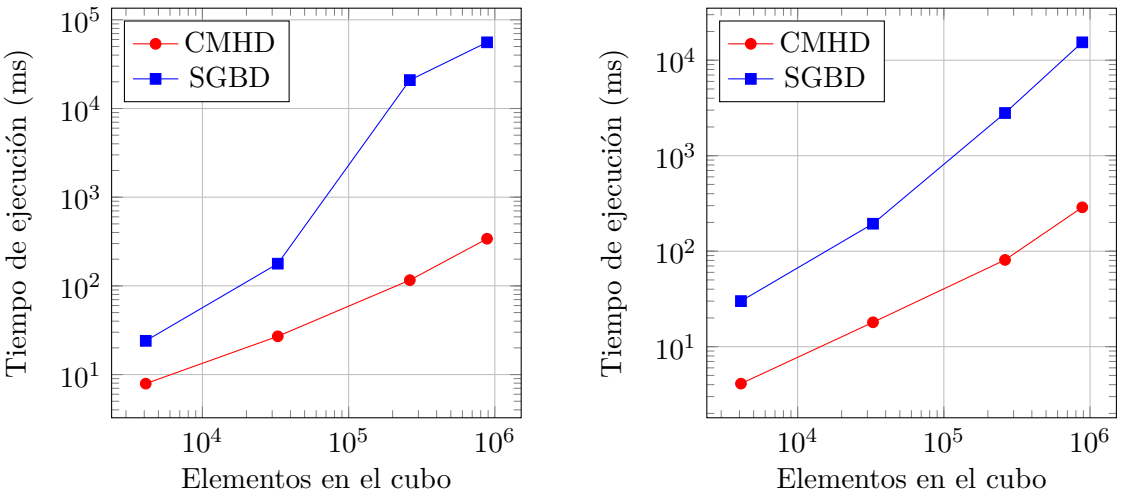
(a) Consulta agrupada por Ciudad-Fecha-Producto (b)Consulta agrupada por Ciudad-Año-Producto

Figura 6.4: Tiempos de ejecución para consultas de agregación SUM agrupado por los niveles Ciudad-Fecha-Producto y Ciudad-Año-Producto



(a) Consulta agrupada por Pais - Fecha - Producto (b)Consulta agrupada por Pais - All - Producto

Figura 6.5: Tiempos de ejecución para consultas de agregación SUM agrupado por los niveles Pais - Fecha - Producto y Pais - All - Producto

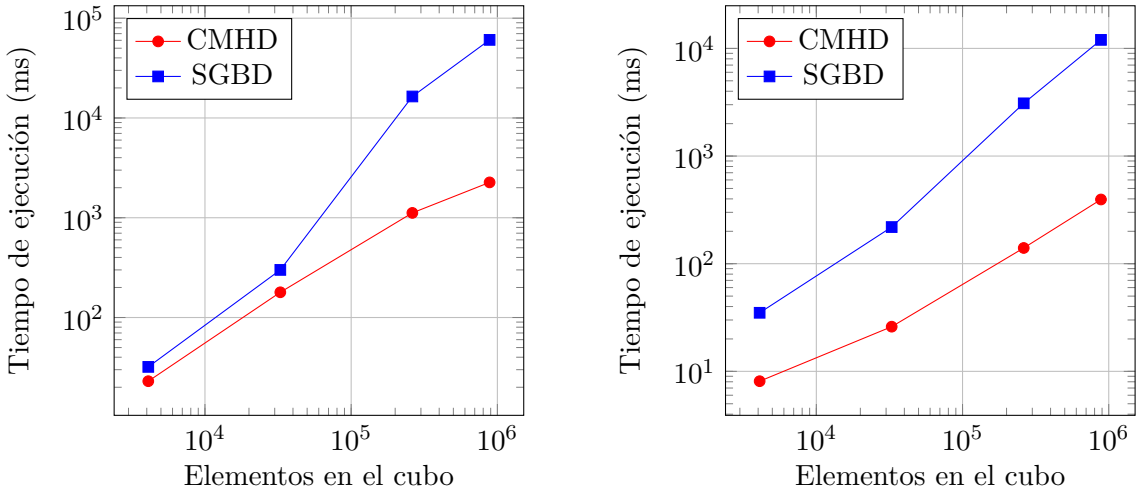


(a) Consulta agrupada por All - Fecha - Producto (b)Consulta agrupada por All - Mes- Producto

Figura 6.6: Tiempos de ejecución para consultas de agregación SUM agrupado por los niveles All - Fecha - Producto y All - Mes- Producto

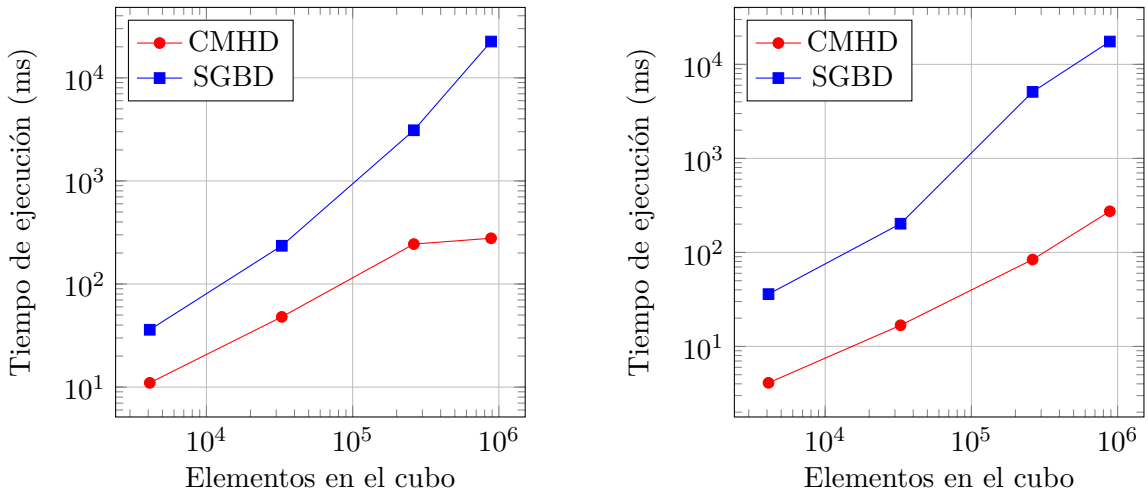
	# Elementos en los cubos							
	4096		32768		262144		884736	
Nivel de consulta	CMHD	SGBD	CMHD	SGBD	CMHD	SGBD	CMHD	SGBD
Tienda - Fecha - Producto	23	32	179	300	1,118	16,419	2,261	60,600
Tienda - Fecha - Tipo	7.5	47	39	263	216	21,049	545	39,000
Tienda - Fecha - Marca	8.5	28,5	36	246	184	18,729	503	34,400
Tienda - Fecha - All	5.2	19	31	156	105	1,891	312	9,200
Tienda - Mes - Producto	8.5	37.25	40	226.25	199	3,600	543	18,300
Tienda - Mes - Tipo	4	39.25	20	249.25	75	19,354	216	44,000
Tienda - Mes - Marca	6	44.25	23	244.25	72	18,400	195	39,000
Tienda - Mes - All	3	16.25	14	131.25	79	18,490	236	51,740
Tienda - Año - Producto	8.1	35	26	219	140	3,090	395	12,000
Tienda - Año - Tipo	4.6	41	14	211	79	3,790	203	14,310
Tienda - Año - Marca	4.3	44	18	236	78	3,840	200	14,700
Tienda - Año - All	3.0	17	14	143	84	3,900	220	15,000
Tienda - All - Producto	6.5	26	33	142	100	2,300	295	10,020
Tienda - All - Tipo	3.5	30	16	172	68	16,200	257	44,100
Tienda - All - Marca	3.3	24	15	170	74	16,400	213	45,000
Tienda - All - All	2.7	13	14	94	113	16,350	229	44,050
Ciudad - Fecha - Producto	11	36	48	235	244	3,100	278	22,500
Ciudad - Fecha - Tipo	4.9	37	17	236	104	4,260	242	40,270
Ciudad - Fecha - Marca	3.9	41	22	211	79	4,450	243	35,100
Ciudad - Fecha - All	3.8	17	16	116	112	4,500	215	35,800
Ciudad - Mes - Producto	5.2	28	19.8	206	88.8	4,800	241.8	12,850
Ciudad - Mes - Tipo	1.6	52	2.3	283	2.0	20,850	2.5	14,000
Ciudad - Mes - Marca	2.1	37	1.2	239	2.2	18,535	1.4	44,760
Ciudad - Mes - All	0.68	29	0.6	112	0.64	2,830	0.7	11,400
Ciudad - Año - Producto	4.1	36	16.8	202	84	5,100	273.8	17,500
Ciudad - Año - Tipo	1.2	46	1.2	205	1.4	5,200	1.2	19,370
Ciudad - Año - Marca	1.0	49	1.0	199	1.2	3,600	1.1	14,620
Ciudad - Año - All	0.27	18	0.28	139	0.41	2,200	0.3	12,800
Ciudad - All - Producto	4.69	25	20.80	140	135.80	2,600	351.80	11,350
Ciudad - All - Tipo	0.68	21	0.5	140	0.57	2,800	0.6	13,740
Ciudad - All - Marca	0.60	22	0.4	129	0.48	3,500	0.6	13.260
Ciudad - All - All	0.023	15	0.1	105	0.14	1,700	0.16	7,000
Pais - Fecha - Producto	12.8	30	31	214	164	3,000	427	12,100
Pais - Fecha - Tipo	5.3	41	17	226	71	3,500	238	13,200
Pais - Fecha - Marca	5.3	37	20	206	76	3,500	209	13,000
Pais - Fecha - All	3.2	20	18	139	87	2,100	202	10,700
Pais - Mes - Producto	4.2	33	17.9	230	82.9	3,600	234.9	13,620
Pais - Mes - Tipo	0.7	36	1.2	249	1.0	3,850	0.9	17,000
Pais - Mes - Marca	0.8	56	0.9	237	0.9	3,900	1.0	14,800
Pais - Mes - All	0.42	18	0.43	131	0.34	2,750	0.4	12,650
Pais - Año - Producto	4.58	42	15.99	219	76.99	3,000	229.99	14,400
Pais - Año - Tipo	0.68	60	0.69	236	0.59	3,200	0.71	14,400
Pais - Año - Marca	0.38	118	0.29	220	0.39	3,650	0.41	15,300
Pais - Año - All	0.08	27	0.09	132	0.09	2,400	0.10	11,550
Pais - All - Producto	3.59	25	13	150	111	1,770	301	15,000
Pais - All - Tipo	0.36	31	0.2	179	0.29	2,140	0.39	14,200
Pais - All - Marca	0.15	19	0.15	155	0.15	2,300	0.17	16,300
Pais - All - All	0.038	14	0.04	102	0.05	0,870	0.04	10,700
All - Fecha - Producto	7.8	22	25	175	114	19,900	320	52,360
All - Fecha - Tipo	4.3	24	18	198	78	17,000	223	46,100
All - Fecha - Marca	4.0	26	17	173	75	16,700	241	39,550
All - Fecha - All	3.3	12	16	85	129	1,070	226	17,100
All - Mes - Producto	3.4	29	17.8	190	80	2,700	285	14,400
All - Mes - Tipo	0.41	26	0.4	221	0.6	19,900	0.66	49,640
All - Mes - Marca	0.34	26	0.51	190	0.55	16,700	0.58	44,200
All - Mes - All	0.31	13	0.31	101	0.31	2,100	0.30	24,000
All - Año - Producto	3.32	22	18	185	75	2,370	243	10,200
All - Año - Tipo	0.38	28	0.37	175	0.41	2,820	0.4	14,500
All - Año - Marca	0.14	30	0.080	187	0.16	2,595	0.17	13,300
All - Año - All	0.05	14	0.050	100	0.05	1,570	0.051	11,950
All - All - Producto	3.29	13.9	18	98	113	19,480	290	16,000
All - All - Tipo	0.24	14.9	0.26	82	0.28	17,370	0.29	16,700
All - All - Marca	0.03	11.9	0.07	89	0.07	16,490	0.08	17.320
All - All - All	0.008	3.2	0.008	20	0.008	460	0.009	1790

Tabla 6.3: Tiempos de ejecución en milisegundos (ms) en consulta MAX para cubos de datos de tres dimensiones



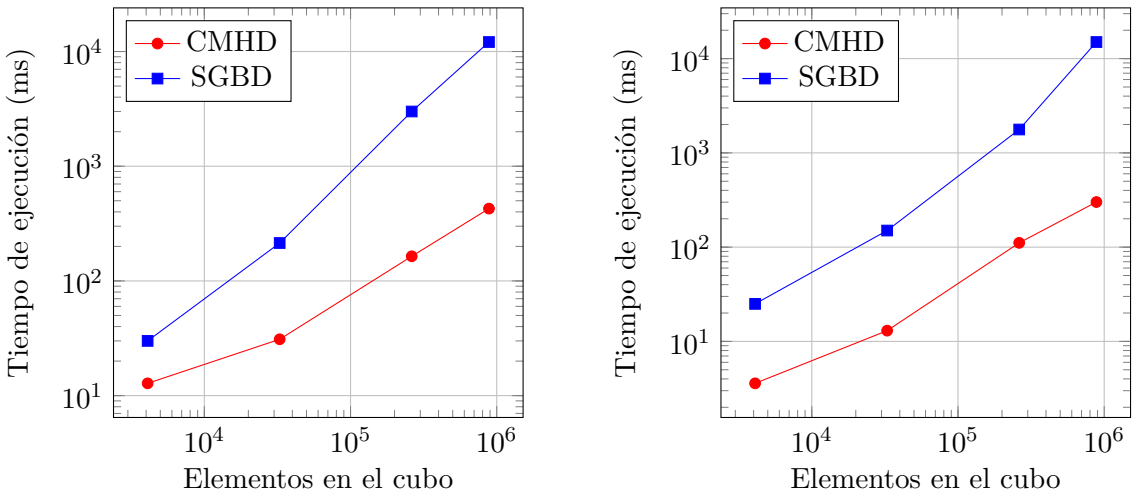
(a) Consulta agrupada por Tienda-Fecha-Producto (b)Consulta agrupada por Tienda-Año-Producto

Figura 6.7: Tiempos de ejecución para consultas de agregación MAX agrupado por los niveles Tienda-Fecha-Producto y Tienda-Año-Producto



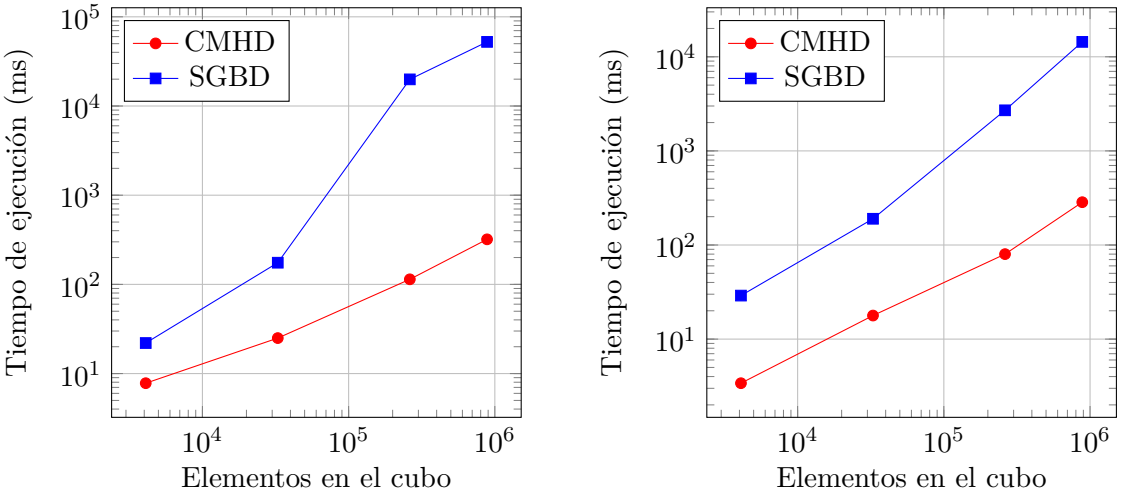
(a) Consulta agrupada por Ciudad-Fecha-Producto (b)Consulta agrupada por Ciudad-Año-Producto

Figura 6.8: Tiempos de ejecución para consultas de agregación MAX agrupado por los niveles Ciudad-Fecha-Producto y Ciudad-Año-Producto



(a) Consulta agrupada por Pais - Fecha - Producto (b)Consulta agrupada por Pais - All - Producto

Figura 6.9: Tiempos de ejecución para consultas de agregación MAX agrupado por los niveles Pais - Fecha - Producto y Pais - All - Producto

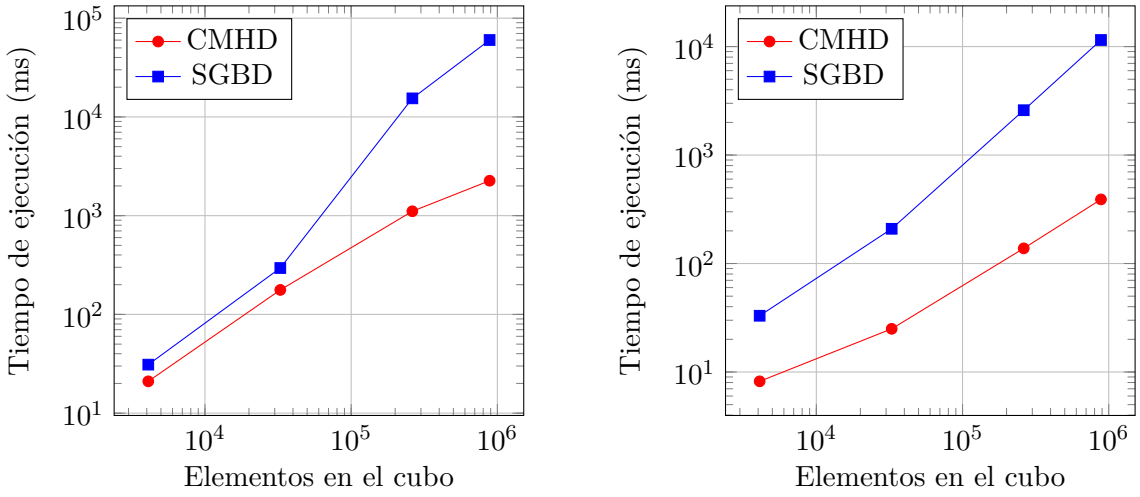


(a) Consulta agrupada por All - Fecha - Producto (b)Consulta agrupada por All - Mes- Producto

Figura 6.10: Tiempos de ejecución para consultas de agregación MAX agrupado por los niveles All - Fecha - Producto y All - Mes- Producto

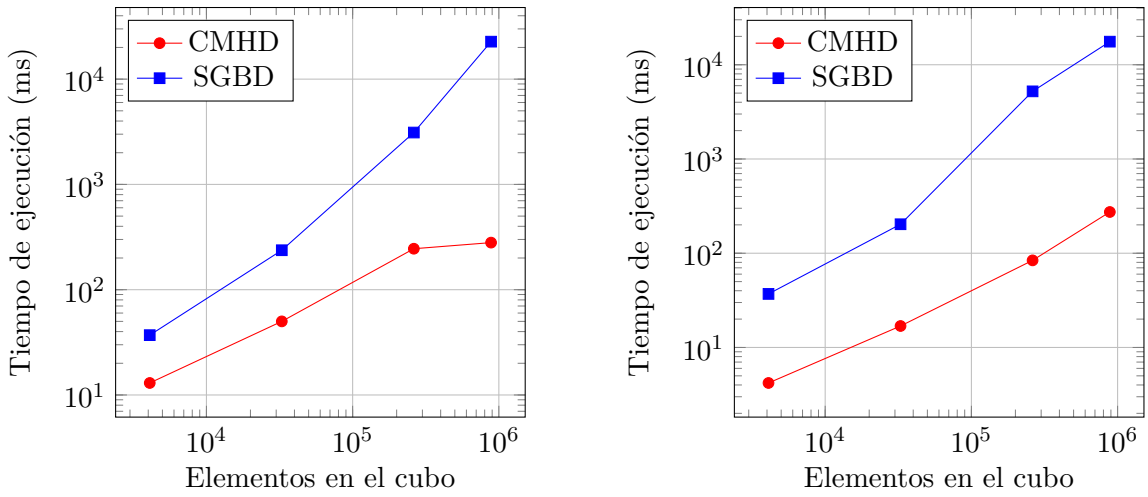
	# Elementos en los cubos							
	4096		32768		262144		884736	
Nivel de consulta	CMHD	SGBD	CMHD	SGBD	CMHD	SGBD	CMHD	SGBD
Tienda - Fecha - Producto	21	31	177	295	1,108	15,419	2,259	60,100
Tienda - Fecha - Tipo	7.4	46	37	260	213	20,049	541	38,500
Tienda - Fecha - Marca	8.4	27,5	34	244	181	17,729	500	33,900
Tienda - Fecha - All	5.1	18	30	153	102	1,091	310	8,700
Tienda - Mes - Producto	8.6	37.29	41	225.25	198	3,500	542	17,500
Tienda - Mes - Tipo	4.1	39.27	19	248.25	74	19,254	215	43,200
Tienda - Mes - Marca	6.05	44.23	23	243.25	71	18,300	194	38,200
Tienda - Mes - All	3.08	16.24	13	130.25	77	18,390	235	50,940
Tienda - Año - Producto	8.2	33	25	209	138	2,590	390	11,500
Tienda - Año - Tipo	4.4	40	13	201	77	3,290	200	13,310
Tienda - Año - Marca	4.25	42	17	226	76	3,340	197	13,200
Tienda - Año - All	3.05	15	13	133	82	3,400	217	14,500
Tienda - All - Producto	6.8	30	35	145	103	2,500	297	10,620
Tienda - All - Tipo	3.8	30	19	174	69	16,400	259	44,700
Tienda - All - Marca	3.6	25	18	173	76	16,600	216	45,600
Tienda - All - All	2.9	15	16	96	114	16,550	231	44,650
Ciudad - Fecha - Producto	13	37	50	237	245	3,110	280	22,700
Ciudad - Fecha - Tipo	5.3	37	19	237	105	4,280	245	40,370
Ciudad - Fecha - Marca	4.1	42	24	212	79	4,450	245	35,232
Ciudad - Fecha - All	3.9	20	18	118	114	4,510	216	35,402
Ciudad - Mes - Producto	5.1	27	19.7	205	88.6	4,770	241.7	11,250
Ciudad - Mes - Tipo	1.5	51	2.2	281	2.0	20,530	2.49	13,320
Ciudad - Mes - Marca	2.02	35	1.25	237	2.1	18,495	1.38	43,760
Ciudad - Mes - All	0.66	28	0.65	110	0.63	2,690	0.7	10,360
Ciudad - Año - Producto	4.2	37	16.9	203	82	5,220	273.9	17,580
Ciudad - Año - Tipo	1.4	48	1.4	206	1.45	5,320	1.32	19,390
Ciudad - Año - Marca	1.3	51	1.3	199	1.23	3,703	1.16	14,770
Ciudad - Año - All	0.37	20	0.38	141	0.41	2,366	0.35	12,910
Ciudad - All - Producto	4.70	26	20.83	143	135.90	2,600	351.80	11,500
Ciudad - All - Tipo	0.69	22	0.55	143	0.56	2,770	0.67	13,830
Ciudad - All - Marca	0.61	22	0.45	129	0.50	3,610	0.67	14.260
Ciudad - All - All	0.043	16	0.15	109	0.16	1,800	0.18	7,100
Pais - Fecha - Producto	12.8	31	33	215	166	3,050	428	12,100
Pais - Fecha - Tipo	5.55	41	18	227	73	3,400	240	13,150
Pais - Fecha - Marca	5.39	39	20	208	77	3,490	211	13,040
Pais - Fecha - All	3.30	26	19	143	88	2,130	203	10,650
Pais - Mes - Producto	4.19	32	17.95	231	83	3,630	234.9	13,600
Pais - Mes - Tipo	0.71	35	1.23	250	1.3	3,860	0.93	17,050
Pais - Mes - Marca	0.9	56	0.94	237	1.1	3,990	1.2	14,820
Pais - Mes - All	0.45	17	0.44	131	0.53	2,790	0.6	12,670
Pais - Año - Producto	4.57	41	15.99	218	76.98	2,000	229.98	13,120
Pais - Año - Tipo	0.67	60	0.68	235	0.58	2,200	0.70	13,300
Pais - Año - Marca	0.37	117	0.30	219	0.38	2,650	0.40	15,100
Pais - Año - All	0.08	27	0.09	131	0.09	1,400	0.10	10,550
Pais - All - Producto	3.60	26	14	151	113	1,470	301	15,100
Pais - All - Tipo	0.38	31	0.32	179	0.33	2,140	0.45	14,300
Pais - All - Marca	0.18	20	0.17	156	0.18	2,200	0.21	16,400
Pais - All - All	0.042	16	0.05	105	0.05	0,900	0.045	10,900
All - Fecha - Producto	7.7	21	24	174	113	18,900	319	52,300
All - Fecha - Tipo	4.2	23	17.5	198	78	16,500	222	46,500
All - Fecha - Marca	4.0	22	16.9	172	74	15,900	240	39,500
All - Fecha - All	3.2	13	15.8	83	126	1,060	225	17,020
All - Mes - Producto	3.3	28	17.7	188	80	1,700	284	13,400
All - Mes - Tipo	0.41	27	0.41	220	0.59	18,900	0.65	47,640
All - Mes - Marca	0.34	25	0.50	189	0.54	16,100	0.58	43,200
All - Mes - All	0.31	14	0.31	100	0.33	1,900	0.32	23,200
All - Año - Producto	3.33	23	19	184	77	2,400	244	10,100
All - Año - Tipo	0.39	30	0.375	175	0.420	2,800	0.43	14,770
All - Año - Marca	0.15	30	0.081	188	0.17	2,595	0.19	13,450
All - Año - All	0.052	15	0.051	101	0.052	1,575	0.053	11,850
All - All - Producto	3.30	13.9	19	99	114	19,180	290	13,850
All - All - Tipo	0.25	14.9	0.28	84	0.295	16,900	0.30	14,020
All - All - Marca	0.03	11.8	0.075	89	0.07	16,290	0.08	15,540
All - All - All	0.008	3.22	0.008	21	0.008	380	0.009	1970

Tabla 6.4: Tiempos de ejecución en milisegundos (ms)) en consulta MIN para cubos de datos de tres dimensiones



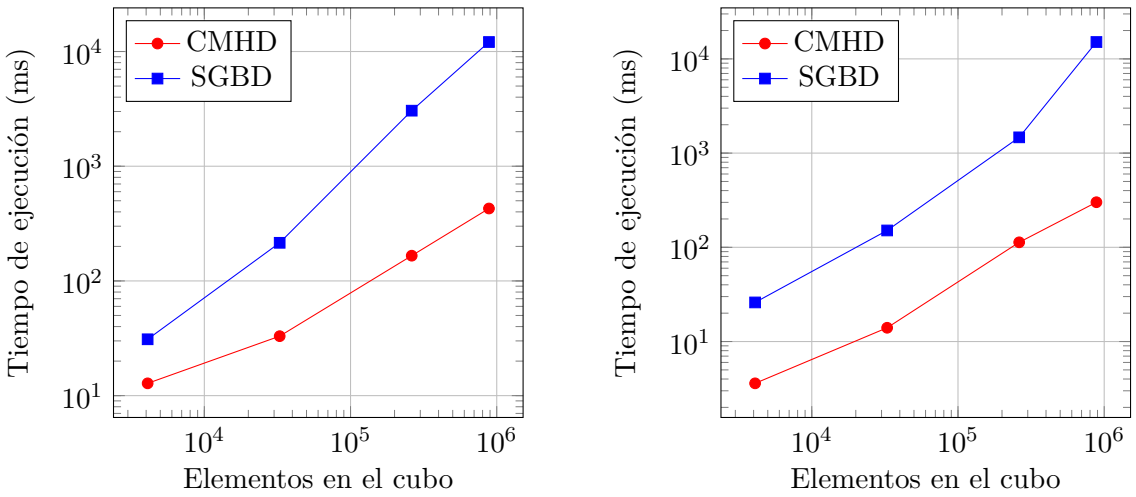
(a) Consulta agrupada por Tienda-Fecha-Producto (b)Consulta agrupada por Tienda-Año-Producto

Figura 6.11: Tiempos de ejecución para consultas de agregación MIN agrupado por los niveles Tienda-Fecha-Producto y Tienda-Año-Producto



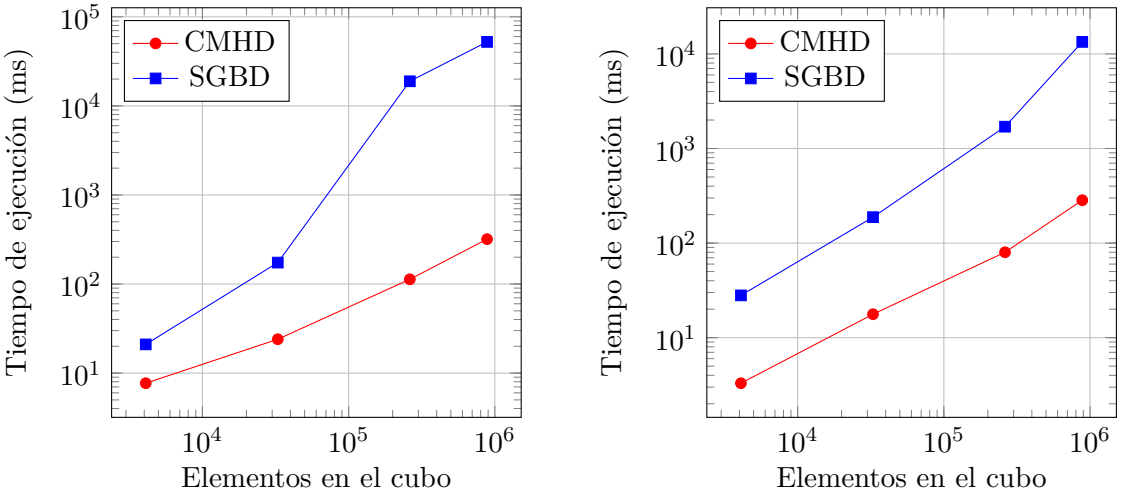
(a) Consulta agrupada por Ciudad-Fecha-Producto (b)Consulta agrupada por Ciudad-Año-Producto

Figura 6.12: Tiempos de ejecución para consultas de agregación MIN agrupado por los niveles Ciudad-Fecha-Producto y Ciudad-Año-Producto



(a) Consulta agrupada por Pais - Fecha - Producto (b)Consulta agrupada por Pais - All - Producto

Figura 6.13: Tiempos de ejecución para consultas de agregación MIN agrupado por los niveles Pais - Fecha - Producto y Pais - All - Producto

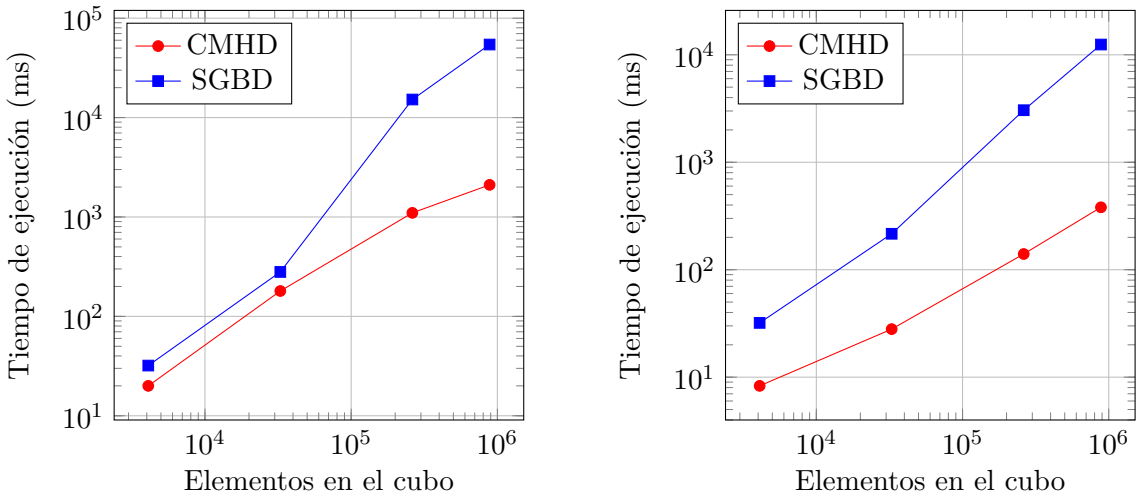


(a) Consulta agrupada por All - Fecha - Producto (b)Consulta agrupada por All - Mes- Producto

Figura 6.14: Tiempos de ejecución para consultas de agregación MIN agrupado por los niveles All - Fecha - Producto y All - Mes- Producto

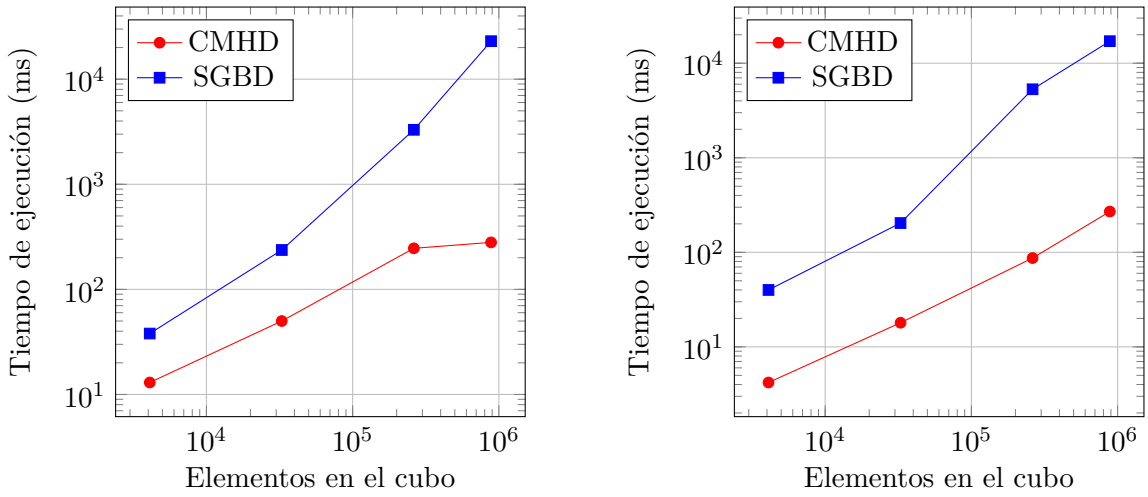
	# Elementos en los cubos							
	4096		32768		262144		884736	
Nivel de consulta	CMHD	SGBD	CMHD	SGBD	CMHD	SGBD	CMHD	SGBD
Tienda - Fecha - Producto	20	32	180	280	1,100	15,200	2,105	54,300
Tienda - Fecha - Tipo	9.2	47	43	260	218	21,630	547	37,000
Tienda - Fecha - Marca	7.9	30	35	245	185	15,560	510	34,600
Tienda - Fecha - All	6.5	19	19	157	105	1,892	315	10,100
Tienda - Mes - Producto	9.5	35	41	210	180	3,700	550	18,200
Tienda - Mes - Tipo	5.4	37	26	216	76	17,990	213	41,000
Tienda - Mes - Marca	4.3	40	21	231	60	16,800	201	40,000
Tienda - Mes - All	3.9	17	17	121	70	18,590	246	50,200
Tienda - Año - Producto	8.3	32	28	216	140	3,050	381	12,460
Tienda - Año - Tipo	4.8	40	16	215	80	3,590	201	14,500
Tienda - Año - Marca	4.5	41	20	231	80	3,780	200	15,100
Tienda - Año - All	3.2	19	16	150	85	3,900	215	15,000
Tienda - All - Producto	6.2	24	21	134	101	2,100	284	10,270
Tienda - All - Tipo	3.4	29	17	174	69	15,200	243	42,980
Tienda - All - Marca	3.1	22	17	165	69	15,100	204	43,300
Tienda - All - All	1.4	15	15	90	106	16,330	219	42,050
Ciudad - Fecha - Producto	13	38	50	237	246	3,300	280	23,000
Ciudad - Fecha - Tipo	5.9	39	19	238	106	4,260	229	38,550
Ciudad - Fecha - Marca	4.9	43	24	213	81	4,550	230	34,600
Ciudad - Fecha - All	4.8	19	18	118	114	4,500	212	35,700
Ciudad - Mes - Producto	5.3	26	20	200	85	4,300	240	13,110
Ciudad - Mes - Tipo	1.7	50	2.4	275	2.1	20,220	2.5	14,400
Ciudad - Mes - Marca	2.2	38	1.3	240	2.2	17,700	1.4	44,700
Ciudad - Mes - All	0.8	30	0.7	110	0.81	2,330	0.87	11,700
Ciudad - Año - Producto	4.2	40	18	204	87	5,300	270	17,100
Ciudad - Año - Tipo	1.2	47	1.5	204	1.5	5,300	1.3	19,300
Ciudad - Año - Marca	1.1	50	1.5	203	1.3	3,500	1.2	14,230
Ciudad - Año - All	0.45	19	0.49	136	0.49	2,200	0.49	12,660
Ciudad - All - Producto	3	24	13	141	114	2,500	226	11,120
Ciudad - All - Tipo	0.69	20	0.61	131	0.73	3,000	0.74	12,440
Ciudad - All - Marca	0.59	22	0.65	130	0.63	3,400	0.65	12.260
Ciudad - All - All	0.033	15	0.33	106	0.34	1,800	0.34	7,400
Pais - Fecha - Producto	8	22	32	202	159	3,000	447	12,400
Pais - Fecha - Tipo	4.9	41	22	205	83	3,100	306	13,780
Pais - Fecha - Marca	4	35	18	200	81	2,900	259	14,000
Pais - Fecha - All	3.7	21	18	140	125	2,300	208	12,500
Pais - Mes - Producto	3	32	17	230	71	3,100	235	12,810
Pais - Mes - Tipo	0.6	32	0.6	230	0.6	3,500	1.1	15,090
Pais - Mes - Marca	1	53	1	218	0.82	4,150	1.1	13,920
Pais - Mes - All	0.42	17	0.42	130	0.43	2,800	0.43	12,600
Pais - Año - Producto	4.6	40	20	229	87	4,100	254	15,000
Pais - Año - Tipo	0.68	52	0.7	250	0.7	4,200	0.72	14,900
Pais - Año - Marca	0.37	103	0.3	245	0.4	4,550	0.39	15,500
Pais - Año - All	0.09	25	0.1	150	0.1	2,930	0.1	13,320
Pais - All - Producto	3.6	27	15	144	88	1,980	230	13,000
Pais - All - Tipo	0.37	33	0.3	160	0.39	3,100	0.39	14,780
Pais - All - Marca	0.15	20	0.15	150	0.17	3,300	0.16	15,300
Pais - All - All	0.047	15	0.05	110	0.05	1,720	0.05	10,500
All - Fecha - Producto	8.1	25	27	180	116	18,960	327	53,600
All - Fecha - Tipo	4.3	27	19	210	71	17,200	321	48,400
All - Fecha - Marca	4.1	27	19	183	90	16,000	252	26,150
All - Fecha - All	3.3	14	18	90	103	3,000	251	25,500
All - Mes - Producto	4.3	33	19	200	82	2,800	235	14,070
All - Mes - Tipo	0.61	30	0.62	220	0.65	20,090	0.65	48,990
All - Mes - Marca	0.54	30	0.55	153	0.56	15,370	0.57	43,240
All - Mes - All	0.3	15	0.3	112	0.3	2,320	0.31	24,040
All - Año - Producto	3.6	22	18	185	66	2,870	233	10,870
All - Año - Tipo	0.37	27	0.37	174	0.38	3,320	0.3	15,000
All - Año - Marca	0.15	30	0.15	170	0.16	3,095	0.16	13,230
All - Año - All	0.04	14	0.04	100	0.05	2,070	0.05	12,300
All - All - Producto	3.3	15	17	98	105	19,600	269	13,600
All - All - Tipo	0.25	15	0.25	83	0.26	18,230	0.26	17,300
All - All - Marca	0.06	12	0.06	87	0.07	17,770	0.067	16,540
All - All - All	0.008	3.2	0.008	22	0.008	420	0.008	1730

Tabla 6.5: Tiempos de ejecución en milisegundos (ms) en consulta COUNT para cubos de datos de tres dimensiones



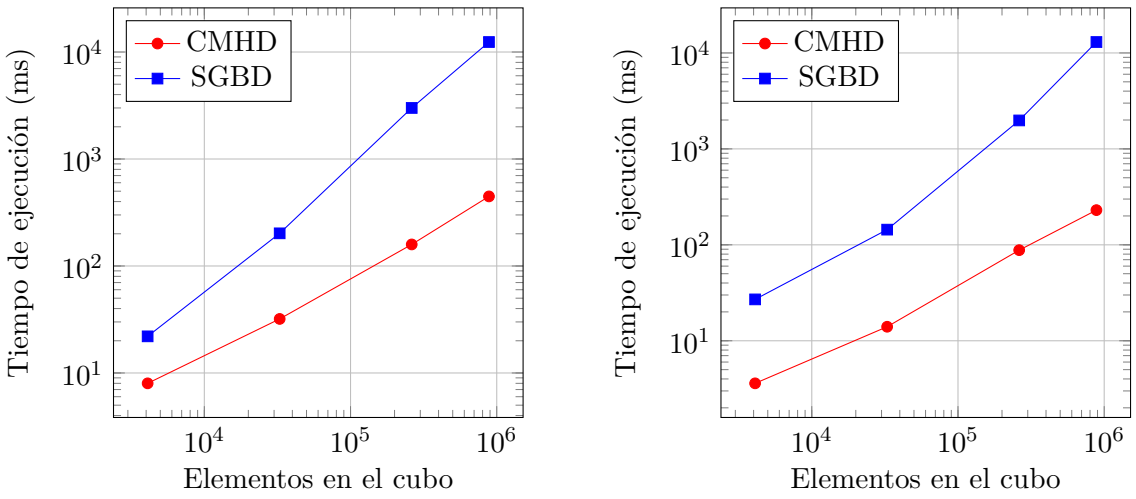
(a) Consulta agrupada por Tienda-Fecha-Producto (b)Consulta agrupada por Tienda-Año-Producto

Figura 6.15: Tiempos de ejecución para consultas de agregación COUNT agrupado por los niveles Tienda-Fecha-Producto y Tienda-Año-Producto



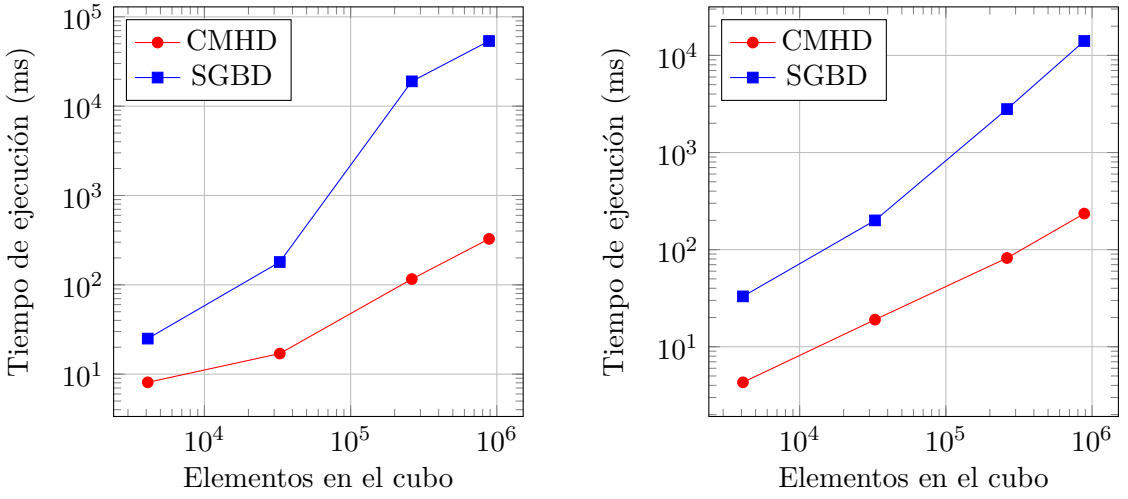
(a) Consulta agrupada por Ciudad-Fecha-Producto (b)Consulta agrupada por Ciudad-Año-Producto

Figura 6.16: Tiempos de ejecución para consultas de agregación COUNT agrupado por los niveles Ciudad-Fecha-Producto y Ciudad-Año-Producto



(a) Consulta agrupada por Pais - Fecha - Producto (b)Consulta agrupada por Pais - All - Producto

Figura 6.17: Tiempos de ejecución para consultas de agregación COUNT agrupado por los niveles Pais - Fecha - Producto y Pais - All - Producto

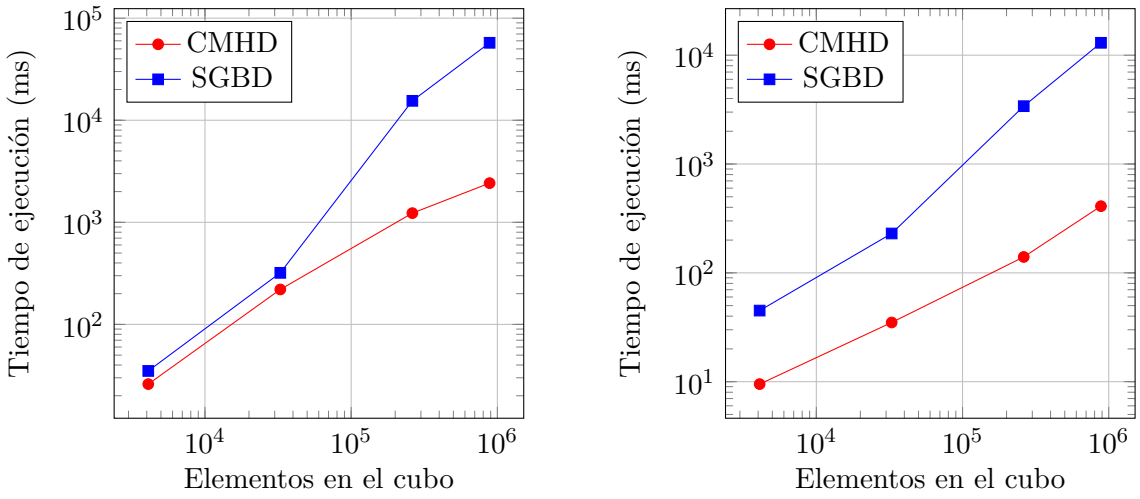


(a) Consulta agrupada por All - Fecha - Producto (b)Consulta agrupada por All - Mes- Producto

Figura 6.18: Tiempos de ejecución para consultas de agregación COUNT agrupado por los niveles All - Fecha - Producto y All - Mes- Producto

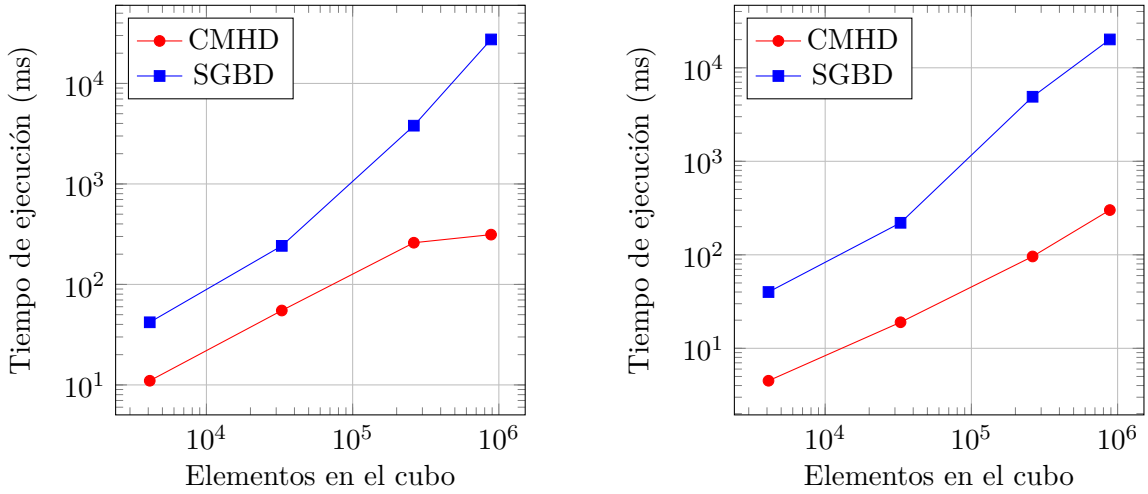
	# Elementos en los cubos							
	4096		32768		262144		884736	
Nivel de consulta	CMHD	SGBD	CMHD	SGBD	CMHD	SGBD	CMHD	SGBD
Tienda - Fecha - Producto	26	35	220	320	1,230	15,500	2,420	57.400
Tienda - Fecha - Tipo	10.2	52	42	255	208	21,500	520	39,300
Tienda - Fecha - Marca	9.5	35	40	250	195	20,100	510	33,700
Tienda - Fecha - All	7	20	36	170	121	1,800	330	11,300
Tienda - Mes - Producto	12	42	42	240	220	3,200	57	20,600
Tienda - Mes - Tipo	6.8	43	25	245	73	21,300	250	47,000
Tienda - Mes - Marca	8.1	45	26	245	65	20,500	190	42,100
Tienda - Mes - All	6.1	15	15	130	76	20,100	230	50,730
Tienda - Año - Producto	9.5	45	35	230	140	3,400	410	13,000
Tienda - Año - Tipo	5.3	43	14	250	76	4,200	210	15,000
Tienda - Año - Marca	5.3	43	19	230	82	3,700	200	15,500
Tienda - Año - All	4.5	20	13	130	90	4,600	230	17,000
Tienda - All - Producto	7.3	28	34	150	111	3,400	350	12,320
Tienda - All - Tipo	4.3	30	16	157	80	17,450	270	42,400
Tienda - All - Marca	3.5	24	16	155	82	18,100	230	42,600
Tienda - All - All	3.2	15	16	92	110	17,100	230	47,600
Ciudad - Fecha - Producto	11	42	55	242	260	3,800	313	27,420
Ciudad - Fecha - Tipo	6.4	45	25	301	100	4,600	250	41,550
Ciudad - Fecha - Marca	5.5	50	27	220	70	5,650	250	31,600
Ciudad - Fecha - All	5	17	18	130	107	5,100	230	33,300
Ciudad - Mes - Producto	5.6	26	22	210	95	5,600	230	13,850
Ciudad - Mes - Tipo	2.1	53	2.5	300	2.3	23,000	3.2	16,440
Ciudad - Mes - Marca	2.5	40	1.8	270	2.4	19,780	2.1	48,200
Ciudad - Mes - All	0.65	33	0.85	100	0.9	3,430	0.9	13,300
Ciudad - Año - Producto	4.5	40	19	220	96	4,900	301	20,100
Ciudad - Año - Tipo	1.6	50	1.5	200	1.5	5,200	1.8	22,240
Ciudad - Año - Marca	1.2	51	1.2	200	1.35	4,200	1.3	15,620
Ciudad - Año - All	0.55	22	0.53	140	0.62	2,700	0.55	14,300
Ciudad - All - Producto	5.2	30	22	150	136	3,800	400	12,000
Ciudad - All - Tipo	0.67	23	0.7	150	0.75	4,320	0.81	14,500
Ciudad - All - Marca	0.62	23	0.6	132	0.80	4,400	0.82	14.000
Ciudad - All - All	0.03	20	0.3	114	0.40	2,400	0.45	7,850
Pais - Fecha - Producto	15	33	30	218	183	3,500	462	12,100
Pais - Fecha - Tipo	6.5	44	20	230	80	3,420	260	12,350
Pais - Fecha - Marca	5.5	40	25	222	82	4,200	240	13,050
Pais - Fecha - All	3.2	25	20	154	89	2,500	232	12,200
Pais - Mes - Producto	5.4	45	20	230	85	4,050	250	14,500
Pais - Mes - Tipo	1.1	45	1.5	270	1.2	4,300	1.3	18,300
Pais - Mes - Marca	1.2	60	1.2	230	1.2	4,300	1.4	15,040
Pais - Mes - All	0.45	22	0.45	135	0.5	3,800	0.55	14,600
Pais - Año - Producto	5	43	20	225	75	3,500	243	14,000
Pais - Año - Tipo	0.84	60	0.75	250	0.65	3,500	0.75	15,030
Pais - Año - Marca	0.45	130	0.3	230	0.5	4,240	0.45	18,200
Pais - Año - All	0.15	30	0.2	160	0.15	3,100	0.15	13,350
Pais - All - Producto	4.4	28	18	160	114	2,550	320	17,000
Pais - All - Tipo	0.35	35	0.5	190	0.40	3,520	0.4	15,500
Pais - All - Marca	0.15	22	0.2	17	0.2	3,360	0.2	16,500
Pais - All - All	0.045	16	0.07	120	0.07	1,850	0.05	12,230
All - Fecha - Producto	8.1	24	30	180	120	19,780	350	58,500
All - Fecha - Tipo	4.5	26	22	223	85	17,930	245	46,040
All - Fecha - Marca	4.3	30	22	173	81	17,500	261	42,320
All - Fecha - All	3.3	15	19	90	134	2,100	250	18,700
All - Mes - Producto	4.4	35	18	207	85	2,500	295	16,450
All - Mes - Tipo	0.66	30	0.65	240	0.8	21,370	0.68	53,200
All - Mes - Marca	0.62	30	0.30	210	0.61	15,700	0.62	46,130
All - Mes - All	0.32	15	0.32	119	0.38	2,500	0.32	26,000
All - Año - Producto	3.7	25	20	186	65	2,870	265	11,600
All - Año - Tipo	0.40	30	0.37	177	0.21	3,320	0.25	15,050
All - Año - Marca	0.15	31	0.082	190	0.22	3,600	0.22	14,800
All - Año - All	0.06	15	0.025	120	0.05	2,460	0.05	13,580
All - All - Producto	3.5	16	20	100	120	20,500	272	14,890
All - All - Tipo	0.32	16	0.25	86	0.28	17,340	0.29	15,060
All - All - Marca	0.04	12	0.07	90	0.07	16,690	0.08	15,970
All - All - All	0.008	3.5	0.008	23	0.008	470	0.009	1950

Tabla 6.6: Tiempos de ejecución en milisegundos (ms) en consulta AVG para cubos de datos de tres dimensiones



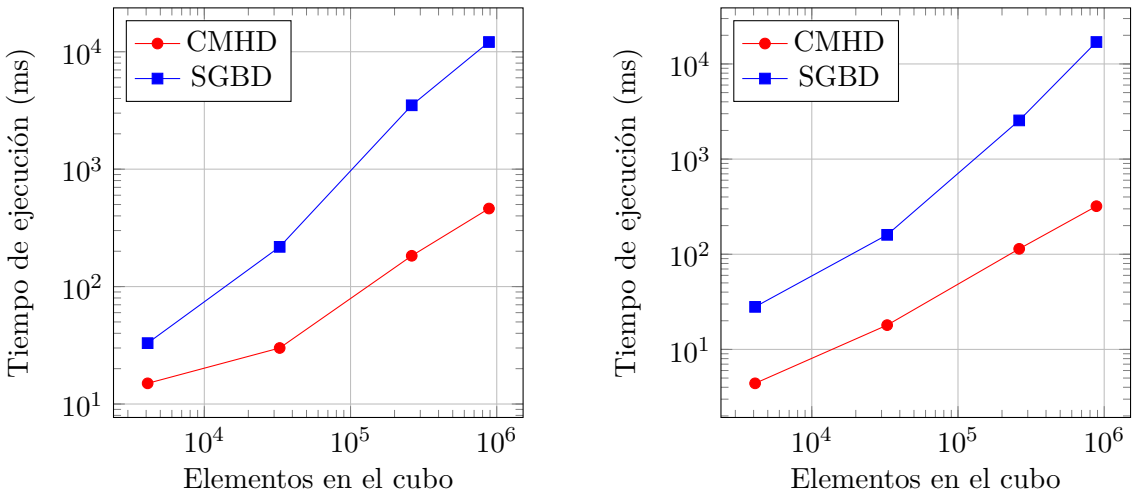
(a) Consulta agrupada por Tienda-Fecha-Producto (b)Consulta agrupada por Tienda-Año-Producto

Figura 6.19: Tiempos de ejecución para consultas de agregación AVG agrupado por los niveles Tienda-Fecha-Producto y Tienda-Año-Producto



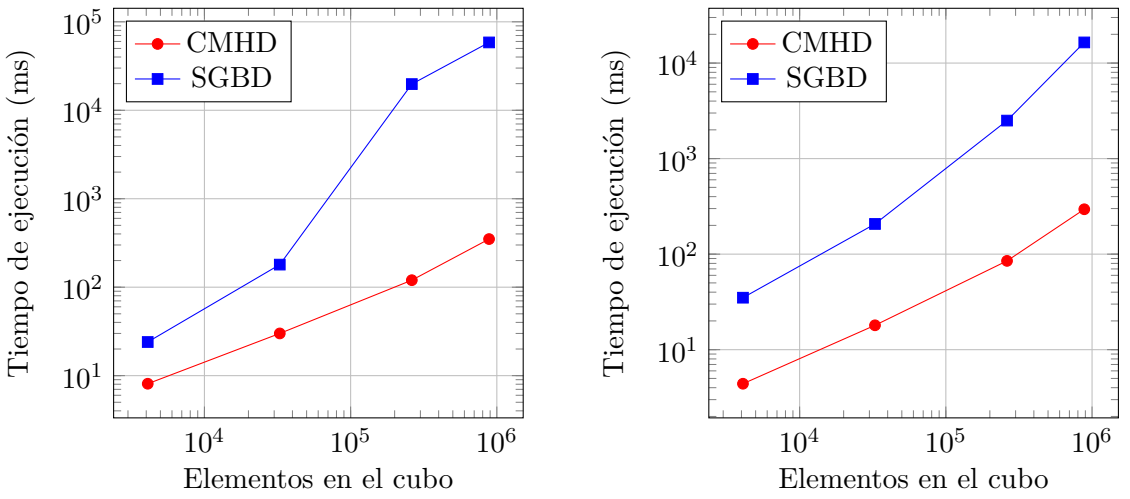
(a) Consulta agrupada por Ciudad-Fecha-Producto (b)Consulta agrupada por Ciudad-Año-Producto

Figura 6.20: Tiempos de ejecución para consultas de agregación AVG agrupado por los niveles Ciudad-Fecha-Producto y Ciudad-Año-Producto



(a) Consulta agrupada por Pais - Fecha - Producto (b)Consulta agrupada por Pais - All - Producto

Figura 6.21: Tiempos de ejecución para consultas de agregación AVG agrupado por los niveles Pais - Fecha - Producto y Pais - All - Producto



(a) Consulta agrupada por All - Fecha - Producto (b)Consulta agrupada por All - Mes- Producto

Figura 6.22: Tiempos de ejecución para consultas de agregación AVG agrupado por los niveles All - Fecha - Producto y All - Mes- Producto

Como se puede observar el CMHD obtiene mejores tiempos de ejecución que MySQL para todos los casos, incluso en los peores escenarios. CMHD sigue dando unos tiempos de ejecución muy bajos. Los peores casos para el CMHD es cuando la consulta generada no es un nodo pre calculado en su árbol, lo que significa que CMHD tiene que calcular todos los posibles casos para aquellas dimensiones, luego recuperar y agrupar dichos datos de

acuerdo a la consulta de agregación realizada. Además vemos que cuando el CMHD debe recuperar nodos de su nivel más bajo, es decir, nodos hojas o hijos este tiende a demorar debido al gran número de datos que tiene que consultar y recuperar. La Tabla 6.2 muestra los tiempos de ejecución obtenidos para todas las consultas con función de agregación SUM ejecutadas sobres los cubos de datos generados para el DW de tres dimensiones de la Figura 6.1. Como se puede observar se obtienen mejores tiempos de ejecución que MySql para todos los casos, la consulta más costosa en cuanto a tiempo de ejecución para el CMHD ocurre cuando debe traer todo su nivel inferior en un consulta, la cual corresponde a $Tienda$ - $Fecha$- $Producto$, pero pese a esto CMHD logra superar a MySql en estas consultas. Existen diferencias considerables en algunos casos puntuales frente a MySql como por ejemplo la consulta $All - All - All$, la cual requiere para este caso la suma total del cubo de datos, pero el CMHD ya tiene almacenado este resultado en la raíz de su árbol y solo debe recuperar dicho valor ya pre calculado. En general mientras mayor sea el nivel de los elementos que conforman la consulta para el CMHD, más rápido responderá este. Las Tablas 6.3, 6.4, 6.5 y 6.6 muestran, respectivamente, los tiempos de ejecución obtenidos para todas las consultas de agregación con funciones MAX, MIN, COUNT y AVG, ejecutadas sobre los cubos de datos generados. Podemos ver que la tendencia en cuanto a tiempos de ejecución del CMHD se mantiene para la mayoría de las consultas realizadas, existe diferencias que son mínimas más aún si tomamos en cuenta que estamos midiendo el tiempo obtenido en milisegundos. Algunos de los peores tiempos obtenidos por el CMHD son efectivamente cuando la diferencia entre los niveles de los elementos que forman la consulta es mayor, por ejemplo la consulta de agregación AVG para los elementos $Pais$ - $Fecha$ - $Producto$ obtiene un tiempo de 15 milisegundos, uno de los peores tiempos obtenidos por el CMHD para esa consulta en el cubo de datos más pequeño, esto se debe a que existe una gran diferencia de nivel entre los elementos de la consulta, además de que los elementos $Fecha$ y $Producto$ se encuentra en el nivel inferior del DW, por lo que se deberá recuperar una gran cantidad de elementos del cubo de datos.

Capítulo 7

Conclusiones y Trabajos Futuros

En este proyecto se presentaron dos sistemas que permiten procesar consultas de agregación sobre DWs almacenados en la estructura de datos compacta CMHD. Este sistema se puede considerar como un prototipo para implementar sobre otras estructuras de datos compactas como el k^2-tree o el k^n-treap, entre otras. En este proyecto nos hemos centrado en la representación compacta y eficiente de conjuntos de datos grandes y complejos. Se propone el uso de la estructura de datos compacta llamada Compact representation of Multidimensional data on Hierarchical Domains (CMHD), la cual se basa en la estructura de datos compacta k^n-treap, pero elimina la restricción de que sus divisiones sean estáticas, es decir, dividir una matriz en K^n sub matrices de igual tamaño, CMHD utiliza las jerarquías de cada dimensión para decidir cómo realizar dicha división en cada nivel. Se presenta la representación de DWs en memoria principal utilizando la estructura de datos compacta CMHD, con el objetivo de reducir de manera eficiente el tamaño de los cubos de datos. Solo fueron considerados cubos de datos con tres y cuatro dimensiones, pero la estructura de datos compacta CMHD puede trabajar con múltiples dimensiones. Inicialmente esta estructura fue diseñada para soportar consultas de agregación solo con función SUM, en este proyecto se extienden dichas consultas a las demás funciones de agregación como lo son MAX, MIN, COUNT y AVG. Se proponen e implementan nuevos algoritmos eficientes que permitan procesar las consultas de agregación sobre los DWs almacenados, de modo que se puedan entregar respuestas exactas y en un breve periodo de tiempo.

Todos los experimentos presentados fueron realizados sobre datos sintéticos. A través de la experimentación realizada podemos concluir que el uso de estructuras de datos compactas en general permite ahorrar espacio de almacenamiento en memoria principal de manera considerable y que al realizar consultas de agregación con las diferentes funciones de agregación se puede responder más eficientemente en todos los casos, que usando un SGBD tradicional (en este caso MySQL). Realizar consultas de agregación para todas las funciones, en general, resultó eficiente ya que la estructura de datos compacta CMHD fue diseñada para responder a este tipo de consultas (SUM), por lo que su arquitectura estaba adaptada de cierta forma, las funciones SUM, MAX, COUNT y MIN demuestran ser un poco más eficientes que AVG a la hora de consultar, esto se debe a que AVG tiene que realizar dos operaciones para su cálculo, la primera es sumar todos los valores calculados y

después dividir el resultado por la cantidad de valores sumados, las principales diferencias en cuanto a tiempo son relativamente menores y casi imperceptibles ya que medimos el tiempo en milisegundos.

Como trabajos futuros se puede comparar la estructura de datos compacta CMHD con otras estructuras de datos compactas, como por ejemplo en (Silva, 2017) donde se compara con la estructura de datos compacta k^n-treap. También se podría medir su rendimiento frente a otros SGBD tradicionales, tales como PostgresSQL y con DWs mas grandes, ya sean DWs con mas dimensiones o con mayor cantidad de elementos. Cabe mencionar que actualmente CMHD solo trabaja con DWs lineales y con dimensiones de iguales niveles, esto se podría modificar, lo que permitiría generar un nuevo campo de pruebas para esta estructura de datos compacta CMHD en cuanto a rendimiento de consultas y espacio de almacenamiento en memoria principal.

Referencias

Nieves R. Brisaboa, Ana Cerdeira-Pena, Narciso López-López, Gonzalo Navarro, Miguel R. Penabad, y Fernando Silva-Coira. Efficient representation of multidimensional data over hierarchical domains. 2016.

Nieves R. Brisaboa, Susana Ladra, y Gonzalo 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. Springer-Verlag, 2009.

Nieves R. Brisaboa, Susana Ladra, y Gonzalo Navarro. Dacs: Bringing direct access to variable-length codes. *Inf. Process. Manage.*, 49(1):392–404, 2013a.

Nieves R. Brisaboa, Miguel R. Luaces, Gonzalo Navarro, y Diego Seco. Space-efficient representations of rectangle datasets supporting orthogonal range querying. *Inf. Syst.*, 38(5):635–655, 2013b.

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

Himanshu Gupta, Venky Harinarayan, Anand Rajaraman, y Jeffrey D. Ullman. Index selection for olap. En *Proceedings of the Thirteenth International Conference on Data Engineering*, ICDE '97, págs. 208–219. IEEE Computer Society, Washington, DC, USA, 1997.

Venky Harinarayan, Anand Rajaraman, y Jeffrey D. Ullman. Implementing data cubes efficiently. En *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, SIGMOD '96, págs. 205–216. ACM, 1996.

Carlos A. Hurtado, Alberto O. Mendelzon, y Alejandro A. Vaisman. Maintaining data cubes under dimension updates. En *Proceedings of the 15th International Conference on Data Engineering, Sydney, Austrialia, March 23-26, 1999*, págs. 346–355. 1999.

Carlos Alberto Hurtado-Larrain. *Structurally Heterogeneous Olap Dimensions*. Tesis Doctoral, 2002.

William H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, Inc., 1992.

G. Jacobson. Space-efficient static trees and graphs. En *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, SFCS '89, págs. 549–554. IEEE Computer Society, 1989.

Inderpal Singh Mumick, Dallan Quass, y Barinderpal Singh Mumick. Maintenance of data cubes and summary tables in a warehouse. En *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, págs. 100–111. ACM, New York, NY, USA, 1997.

Gonzalo Navarro. *Compact Data Structures – A practical approach*. Cambridge University Press, 2016.

Rajeev Raman, Venkatesh Raman, y S. Srinivasa Rao. Succinct dynamic data structures. En *Algorithms and Data Structures*, págs. 426–437. Springer Berlin Heidelberg, 2001.

F. Silva. *Compact Data Structures for Large and Complex Datasets*. Tesis Doctoral, Universidade da Coruña, 2017.

Michael Worboys y Matt Duckham. *GIS: A Computing Perspective, 2Nd Edition*. CRC Press, Inc., 2004.