INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

"COMPACTACIÓN DE CUBOS EN MEMORIA PRINCIPAL CON LA ESTRUCTURA DE DATOS ÁRBLIS"

TESIS

QUE PARA OBTENER EL GRADO DE
MAESTRO EN CIENCIAS
CON ESPECIALIDAD EN COMPUTACIÓN

P R E S E N T A

ING. BELLA CITLALI MARTÍNEZ SEIS

DIRECTORES DE TESIS: DR. GILBERTO LORENZO MARTÍNEZ LUNA
DR. ADOLFO GUZMÁN ARENAS

México, D.F. **Diciembre, 2010**

Contenido

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

RESUMEN

Obtener información a partir de bases de datos de gran volumen es posible a través del procesamiento analítico en línea multidimensional (MOLAP, Multidimensional On-Line Analytical Processing). Este procesamiento se lleva a cabo a través de las denominadas Herramientas MOLAP, entre éstas existe el prototipo de software ANTECUMEM (Analizador Temporal de Cubos en Memoria); la cual realiza un manejo multidimensional de los datos a través de estructuras de datos llamadas Arblis dentro de la lattice correspondiente.

Se presenta la **Estructura Multidimensional AC** (Apuntadores a Catálogo) que parte de la estructura de cubos de datos Arblis comprimida, llevándose a cabo una compactación de tamaño en disco duro y en memoria principal de tal forma que se logre una navegación a través de este cubo de datos además del aumento en la cantidad de datos que pueden ser accedidos y por ello acumulados en el mismo espacio.

El desarrollo del diseño de **compactación de los datos en Arblis** se obtiene a través del estudio de diversos métodos: eliminación de los ítems de datos redundantes, conversión a notación compacta, supresión de caracteres repetidos sucesivamente, evitar espacios vacíos, sustitución de datos repetidos, sustitución de texto idiomáticos, algoritmo de Huffman, algoritmo de Shannon-Fano y algoritmo LZW (Lempel Ziv Welch). La compactación seleccionada se obtiene tomando en cuenta los métodos mencionados así como las características de navegabilidad y orden presentes en Arblis obteniendo cuatro propuestas viables: arreglos con elementos vacíos, arreglos sin elementos vacíos, mapas de bits y apuntadores a catálogo (AC); siendo seleccionada la última ya que se adaptaba más a las necesidades.

Una vez que se tiene materializada la estructura, ésta es cargada en memoria principal haciendo uso de la vista más grande de la lattice, de tal forma que son cargadas las *n* dimensiones del cubo de datos. Una vez que se tiene en memoria principal pueden ser contestadas siete tipo de consultas originarias de ANTECUMEM: pregunta puntual, pregunta de rangos, pregunta de eficiencia global, pregunta de eficiencia grupal, preguntas sobre la conservación y pérdida, pregunta de temporalidad y pregunta de tendencia. A dichos algoritmos se les aplicó una modificación mínima al realizarse la compactación a pesar de que **la búsqueda se realiza en la estructura compresa.**

ANTECUMEM permite tener una estructura de diez dimensiones y la manipulación simultánea de hasta cuatro dimensiones. Por lo que se realizan las modificaciones pertinentes para **manejar *n* dimensiones** en la creación de AC y la manipulación de ésta para la solución de las siete preguntas antes mencionadas. De tal forma que el manejo del cubo de datos es de *n* dimensiones limitado únicamente por la cantidad de memoria.

PALABRAS CLAVE: E.4 Codificación y teoría de la información, H.2.8 Minería de datos, E.2 Representación de almacenamiento de datos, ANTECUMEM, Estructura AC.

ABSTRACT

To get information from a huge data base is possible through the Mutidimensional On-Line Analytical Processing (MOLAP). This processing is done by MOLAP Tools, between them, exist a prototype of software that performs such action, it is called ANTECUMEM (Temporal Analyzer Memory Cubes); that makes a multidimensional data management through data structures called Arblis for each node in the lattice.

The **Multidimensional Data Structure AC** (Catalog Pointers) is presented, based on the concept of the data cube structure Arblis but compress, it is done with a compression in hard disc and main memory, so that navigation thorough that data cube could be possible, in addition of the increase in the amount of data that can be accumulated and therefore accessed in the same space.

The design development of the **data compression in Arblis** is obtained through the study of different methods: elimination of redundant data items, converting to compact notation, removal repeated characters, to avoid empty spaces, replacing redundant data, replacing idiomatic text, Huffman algorithm, Shannon-Fano algorithm and LZW algorithm (Lempel Ziv Welch). The selected compression is obtained by taking into account the above methods and the characteristics of navigability and order that Arblis has, getting four viable proposals: arrays with empty elements, arrangements without empty elements, bitmaps and catalog pointers (AC), being the last one was selected because it filled the requirements.

Once the structure AC is materialized, it is loaded into main memory using the largest view of the lattice, so we would have a *n*-dimensional data cube. Once it is in main memory, seven types of queries originated from ANTECUMEM can be answered: punctual question, ranking question, global efficiency question, group efficiency question, preservation and loss question, and temporary trend question. For those algorithms, it was applied the minimum modifications when the compression was done, even though that **the search is done in the compress data structure** (AC).

ANTECUMEM allows a ten-dimensional structure and the simultaneous use of four dimensions. So therefore the appropriate changes are done in order **to handle a *n*-dimensional structure** in the moment of the creation of AC and manipulation of AC for the solution of the seven questions above. So that the use of the data cube AC of n dimensions is limited only by the amount of memory.

KEYWORDS: E.4 Coding and information theory, H.2.8 Data mining, E.2 Data storage representations, ANTECUMEM, AC Structure

CAPITULO I. INTRODUCCIÓN

La expansión de memoria que se está llevando a cabo [Tremaine, 2001] permite un mayor almacenamiento de los datos, sin embargo, es también necesario buscar almacenar un mayor número de datos en un menor espacio posible permitiendo al usuario aumentar la cantidad de datos que procesa. Es por ello, que hacer uso de técnicas de compresión y compactación de datos puede ayudar a incrementar aún más el número de datos a almacenar en los espacios de memoria ahora disponibles.

Existe un prototipo diseñado y programado para el análisis de datos en bases de datos multidimensionales, también conocidas como cubos de datos [Harinarayan, 1996], [Gray, 1995] y [Han, 2010], dicho software es llamado ANTECUMEM (Analizador Temporal de Cubos en Memoria) propuesta en [Martínez, 2001], el cual se tomará como base para mostrar el uso de estas técnicas para mejorar el uso eficiente del espacio disponible, con lo cual se da lugar a la realización de la tesis titulada "Compactación de Cubos de Datos en Memoria Principal con la estructura Arblis".

1.1 ANTECEDENTES

Los sistemas orientados a OLTP (*On Line Transaction Processing*) apoyan las actividades empresariales a través de la entrada, almacenamiento y salida de datos; posteriormente, la evolución de estos sistemas ha permitido integrar el procesamiento de los datos para obtener información. Una de las técnicas que permite realizarlo es OLAP *(On-Line Analytical Processing*) [Agrawal, 1995], [Ho, 1997], [Kotidis,1998] y [Shanmugasundaram, 1999], el cual permite agilizar la consulta de grandes cantidades de datos a través de estructuras multidimensionales (*Cubos de Datos*) extrayendo información útil para tendencias o patrones que encaminan a las empresas a visualizar nuevas situaciones de negocio.

Dentro de las empresas ha surgido la necesidad de manipular la información para la obtención de reportes, gráficos, tendencias, y/o patrones, permitiendo que el proceso de utilización de cubos de datos sea de vital importancia, exigiendo un corto tiempo de respuesta [Han, 2010].

Esta investigación toma como base un prototipo de software conocido como ANTECUMEM, herramienta que permite realizar análisis en una base de datos, almacenando los cubos de datos en una base de datos multi-dimensional dentro de la memoria principal a través de una estructura con arreglos que están ligados entre sí, dicha estructura es llamada Arblis, la cual ocupa un espacio en memoria que es necesario mejorar su uso a través de métodos de compactación de los datos, aumentando así, el número de registros en memoria y posiblemente su velocidad. Aunado a esto se aportará al tratamiento de dichos datos para *n* dimensiones, donde el número de dimensiones solamente estará limitado por la capacidad de la memoria.

1.2 PLANTEAMIENTO DEL PROBLEMA

ANTECUMEM es una herramienta que permite análisis en cubos de datos a través de la estructura Arblis sobre la cual se realizan consultas que responden a preguntas de tipo: puntual, rango, eficiencia, eficiencia grupal, conservación/perdida, temporalidad y tendencia; definidas en 3.2 ARQUITECTURA DE ANTECUMEM, para mayor detalle véase[Martínez, 2007]; las cuales se llevan a cabo en corto tiempo, sin embargo el almacenamiento en cubos de datos y las consultas en éstos son demandantes en espacio de disco y espacio de memoria principal, por lo que el propósito de este trabajo es buscar mejorar el uso eficiente de dicho espacio en la herramienta, problema ya plateado en [Shanmugasundaram, 1999]. Dada una base de datos con el cubo de datos correspondiente realizar una codificación de los valores de cada variable que requieran un menor espacio en disco y memoria virtual.

El espacio requerido para el almacenamiento podría no ser el óptimo por lo que es necesario realizar una estimación del espacio en memoria principal ocupado por la estructura Arblis, de tal forma que se obtenga una mejora en el desempeño del uso de la memoria principal, sin embargo la compactación de los datos influye negativamente en el tiempo de respuesta.

El aumento de número de dimensiones aumenta la complejidad de los algoritmos de respuesta a las preguntas antes mencionadas, por lo que dichas respuestas se realizan considerando hasta cuatro vistas. Sin embargo en ocasiones en necesario contestar preguntas en cubos de mayores dimensiones, de tal forma que el presente trabajo se enfocará también a resolver esta limitante.

1.3 OBJETIVOS

1.3.1 OBJETIVO GENERAL

Generación e implementación de un diseño de compactación de los datos en memoria principal para la herramienta ANTECUMEM que permita aumentar el número de tuplas en los cubos de datos manteniendo las propiedades de la estructura Arblis.

1.3.1 OBJETIVOS ESPECÍFICOS

Los objetivos específicos para la presente tesis se listan a continuación:

- Compactación y manejo técnico del prototipo de software ANTECUMEM y de la Estructura de Datos Arblis
- Diseñar y comparar diversas codificaciones de valores en las variables de las bases de datos de gran volumen para ocupar el menor espacio posible en disco y memoria principal.

- Desarrollar una mejora para incrementar las vistas que pueden ser almacenadas en la estructura Arblis con la característica de ciclos predefinidos de búsqueda, de tal forma que se conserven las propiedades de Arblis.

- Implementar la compactación de datos dentro de la herramienta ANTECUMEM haciendo uso de tecnología portable (Java).

- Aplicar la compactación a diversas bases de datos con al menos un millón de registros.

1.4 JUSTIFICACIÓN

Atendiendo las necesidades de las organizaciones: empresariales, académicas, comerciales, sociales, políticas y gubernamentales, entre otras; mismas que, por el crecimiento de las bases de datos, el incremento de la capacidad de la memoria principal como lo presenta [Rao, 1999],, la capacidad de procesamiento y el volumen de datos que se necesarias para ayudar a responder preguntas de negocio, requieren que profesionales emprendan, desarrollen o diseñen sistemas que faciliten y abatan tiempos de respuestas para la obtención de datos.

El proceso de utilización de cubos de datos tiene como propósito cubrir esta necesidad, como herramienta, para dar satisfacción al usuario. Por su parte ANTECUMEM como herramienta debería de tener una técnica para el mejoramiento del uso del espacio disponible en memoria principal y así tener una alternativa propia para tratar el problema de crecimiento de las bases de datos.

En este contexto, tal como se establece en el título de tesis “Compactación de Cubos de Datos en Memoria Principal con la estructura Arblis”, se proyecta acceder a la información organizada multi-dimensionalmente mediante un proceso de abstracción de datos en la memoria principal, a través de la estructura de datos Arblis que se presenta en cada uno de los nodos de una lattice, nodos en que se compactarán los datos para mitigar espacio de almacenamiento.

1.5 BENEFICIOS ESPERADOS

- Uso del almacenamiento de cubos de datos en los nodos de una *lattice*, en cuyos nodos son almacenadas las vistas materializadas.
- Desarrollar un diseño de compactación de los datos, para aumentar el número de registros que se almacenan en la memoria principal, así como en el disco.
- Contribuir en el desarrollo del prototipo de software ANTECUMEM con la implementación de un diseño de compactación para el almacenamiento de datos
- Obtener una estimación del tiempo y espacio de carga en la estructura Arblis con los métodos de compactación aplicados al cubo en memoria principal
- Incluir una técnica de búsqueda para reducir el tiempo de respuesta en consultas realizadas sobre el cubo de datos

1.6 ALCANCES

A continuación se muestran los alcances y límites de esta tesis, mostrando una definición más específica de lo que se pretende lograr.

Algunos de los alcances que se tendrán son:

- Mejorar el uso del espacio en el uso de la herramienta ANTECUMEM; así como proporcionar datos estimados para la carga de la estructura. A través de pruebas y prácticas en compactación de datos.
- Crear interfaz gráfica que permita al usuario realizar análisis de la información que éste requiera de manera clara mejorando el tiempo de respuesta y mostrando a través de texto, tablas y graficas la información analizada.
- Utilizar java como tecnología en la que se continuará realizando la aplicación.
- Realizar pruebas de aplicación de algoritmos de compactación en la base de datos ejemplificadas y base de datos de ventas SH (*Sales History,* base de datos obtenida en distribuciones de SABD Oracle).

1.7 ORGANIZACIÓN DE LA TESIS

El presente trabajo se compone de los siguientes capítulos:

- Capítulo 1
  Se presentan los antecedentes, la definición del problema a resolver a lo largo del trabajo de tesis, los objetivos que se esperan lograr, los alcances, límites y beneficios que se desean obtener, así como una justificación.

- Capítulo 2
  Se proporciona una introducción a definiciones relacionadas con OLAP y minería de datos, así como fundamentos teóricos necesarios.
  Se muestra el marco teórico con su estado del arte relacionado con las herramientas existentes que cargan en memoria principal cubos de datos y representación de vistas con estructura de una lattice.

- Capítulo 3
  Se presenta el análisis y diseño bajo la metodología orientada a objetos, haciendo uso de diagramas de casos de uso, diagramas de secuencia, diagramas de actividades y diagramas E-R.

- Capítulo 4
  Se muestra una comparación de algoritmos utilizados para la compactación, permitiendo implementar la mejora para reducción de tiempo e incremento de datos en memoria. Y se realiza entorno gráfico donde se muestran las respuestas a las consultas.

- Capítulo 5
  Se expone y analizan las pruebas realizadas a la implementación de la compactación por medio de tres bases de datos que contengan más de un millón de registros.

- Capítulo 6
  Se manifiestan las conclusiones a las que se ha llegado con el complemento a la herramienta ANTECUMEM, replanteando objetivos y marcando posible trabajo a futuro.

CAPÍTULO II. MARCO TEÓRICO

2.1 INTRODUCCIÓN

El manejo de la información presenta un obstáculo ante el gran volumen, complejidad y heterogeneidad de los datos. En el contexto del marco teórico de la presente tesis se presenta un bosquejo de las bases conceptuales que sustentan el proceso de compactación de cubos en memoria principal con la estructura de datos Arblis.

Se inicia estableciendo la definición de lo que es un cubo de datos, en qué consiste, como se crean, se ejemplifican las operaciones que se pueden presentar y posteriormente se muestra la descripción de la estructura de datos Arblis.

Se continúa con vistas materializadas que permite la simplificación de las consultas realizadas a la base de datos, seguridad, independencia de los datos abordando en la sintaxis utilizada, SQL *(Structured Query Language)*.

Asimismo, se plantea la forma de compactación de datos, técnicas, compactación, eliminación de ítems, conversión a notación compacta, supresión de caracteres repetidos, entre otros

Se finaliza el marco teórico realizando una descripción de las herramientas tipo MOLAP (*Multidimensional Online Analytical Processing)*: Palo, Essbase y Applix, las cuales manejan la creación de cubos de datos para la respuesta a preguntas de negocio [Agrawal, 1995] como lo hace ANTECUMEM siendo ésta última sobre la que se planteará la mejora.

2.2 DEFINICIONES

La minería de datos está formada por un conjunto de conceptos y métodos utilizados para la extracción de conocimiento (patrones) de determinado conjuntos de datos, de tal forma que provee un soporte para la toma de decisiones en industria, negocios, gobierno y ciencia. Esta rama de las Bases de Datos da pie al trabajo "Compactación de Cubos en Memoria Principal con la Estructura de Datos Arblis" por lo que a continuación se dan a conocer los conceptos que lo fundamentan para realización de la compresión y desarrollo del presente trabajo.

2.2.1 BASE DE DATOS

Las bases de datos son una colección de datos, generalmente de gran tamaño, lógicamente relacionados donde los elementos están integrados manteniéndose al mínimo las posibles duplicaciones. Además de guardar los datos se tiene una descripción de éstos conocida como catálogo del sistema o metadatos, es por ello que se suele describir a las bases de datos como una colección autodescriptiva de registros integrados.

La autodescripción de las bases de datos permite independencia entre programas de aplicación y datos. Los primeros son programas que se usan para interactuar con las bases de datos y en su conjunto se tiene un Sistema Administrador de Bases de Datos (SABD)

Al analizar las necesidades de información de una organización se den de identificar: entidades, atributos y relaciones. Una entidad es un objeto distintivo dentro de la organización, un atributo es una propiedad que describe

algún aspecto de dicho objeto y la relación es la asociación entre entidades. El uso de estos permite tener la base de datos lógicamente relacionada. Por ejemplo en la Ilustración 1 se muestra un diagrama Entidad-Relación (ER) para un caso de ventas de productos a determinados clientes. Está compuesto por:

- Seis entidades: Departamento, Productos, Cliente, Cargo, Empleado y Compra
- Cinco relaciones: (los nombres adyacentes a las líneas) Pertenece, es vendido, realiza, tiene y despacha
- Trece atributos: Tres para la entidad de cliente: idCliente, NombreCompleto y Dirección; y dos para el resto de las entidades: idDepartamento, NombreDep, idProductos, Producto, idCargo, DescripciónCargo, idEmpleado, NombreEmpleado, idVenta y Fecha.

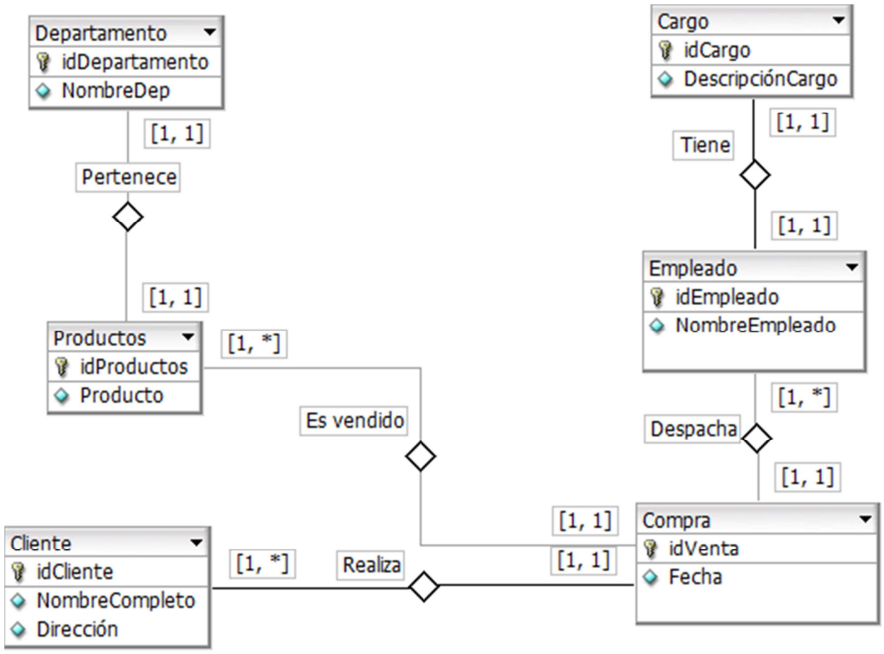


Ilustración 1. Ejemplo del diagrama entidad- relación. Cuenta con 6 entidades, 5 relaciones y 13 atributos

La base de datos representa las relaciones lógicas entre entidades que contienen atributos. Es decir es un almacenamiento de datos ordenado lógicamente que permita acceso a ellos de manera ordenada.

Existen diversos modelos de bases de datos, entre ellas se encuentran:

- **Bases de datos jerárquicas.** Se organiza en una forma jerárquica, similar a un árbol de estructura de datos en donde un nodo padre de información puede tener varios hijos y los hijos tiene un único padre. Son útiles en el caso de aplicaciones que manejan gran volumen de información y datos muy compartidos permitiendo crear estructuras estables sin embargo se presentan redundancia de datos.
- **Base de datos de red**. Fue una mejora al modelo jerárquico ya que se permite que un mismo nodo tenga varios padres ofreciendo solución al problema de redundancia de datos.
- **Base de datos relacional.** Es el modelo más utilizado en la actualidad. Usa una colección de tablas para representar tanto los datos como sus relaciones. Cada tabla contiene registros de un tipo dado. Cada tipo de registro define un número fijo de campos (atributos) representados como columnas en las tablas.

- **Base de datos entidad-relación.** Se basa en una percepción del mundo real que consiste en una colección de objetos básicos denominados entidades y las relaciones entre ellos.
- **Bases de datos orientadas a objetos**. El modelo de datos orientado a objetos es una extensión del modelo E-R con los conceptos de la encapsulación, los métodos y la identidad de los objetos.
- **Base de datos distribuida**. La base de datos está almacenada en varias computadoras conectadas en red. Surgen a la necesidad de tener una organización descentralizada
- **Bases de datos multidimensionales**. Ideadas para el desarrollo de aplicaciones concretas como la creación de cubos OLAP. Conceptualmente se diferencia de las bases de datos relacionales según la representación de los atributos, los cuales pueden ser de dos tipos: dimensiones o métricas que se desean estudiar.

2.2.2 CUBOS DE DATOS

Las bases de datos multidimensionales son bases de datos en las que cada campo (atributo) puede representar: una dimensión (**atributos de dimensión**) o una métrica, hecho o agregado (**atributos de medida**).

Los atributos de medida nos permiten hacer el análisis de datos ya que miden algún valor y pueden agregarse. Por ejemplo el atributo de ventas está representado por los números en la Ilustración 2 ya que mide la cantidad de unidades vendidas de los productos 1 y 2 en los colores azul y verde en todos los tamaños. El resto de los atributos son los atributos dimensión ya que definen las dimensiones en las que se ven los atributos de medida y los resúmenes de los atributos de medida. En la Ilustración 2 el color, producto y tamaño son los atributos dimensión.

Tamaño: Todos (all)

Producto	Color: Azul	Color: Verde	Total (all)
Prod 1	9	4	13
Prod 2	5	2	7
Total (all)	15	7	20

Ilustración 2. Tabulación cruzada de Ventas con Producto y Color para todos los Tamaños

Los **cubos de datos** [Harinarayan, 1996] y [Gray, 1995] permiten a los datos ser modelados y vistos en múltiples dimensiones, y está definida por dimensiones y hechos. Las dimensiones son una entidad respecto a la cual determinada organización almacena los datos los hechos son medidas numéricas a través de las cuales se requieren hacer el análisis de las relaciones entre dimensiones. Son representados a través de **vistas** las cuales permiten un acceso flexible que proporciona la base de datos para el procesamiento de tal forma que el usuario pueda ver las ventas, por ejemplo, de los productos según el departamento, según el tiempo y/o según el vendedor.

Para la creación del cubo es necesario:

1) Definir el cubo de datos. Especificar las mediciones y dimensiones del cubo de datos.
2) Diseñar las agregaciones. Especificar la estrategia en que los datos se conjuntan obteniendo elementos pre-calculados de los datos.

3) Cargar el cubo. Se realiza en memoria principal para procesarlo y posteriormente disminuir tiempos de consulta. Que es sobre lo que se trabajara para reducir el espacio que ocupa en memoria principal así como en disco duro.

A continuación se muestra un cubo de datos de 3 dimensiones: color, producto y tamaño. Con las mediciones (hechos) correspondientes a otra variable que es ventas. Observe que la Ilustración 2 es la vista frontal del cubo mostrado.

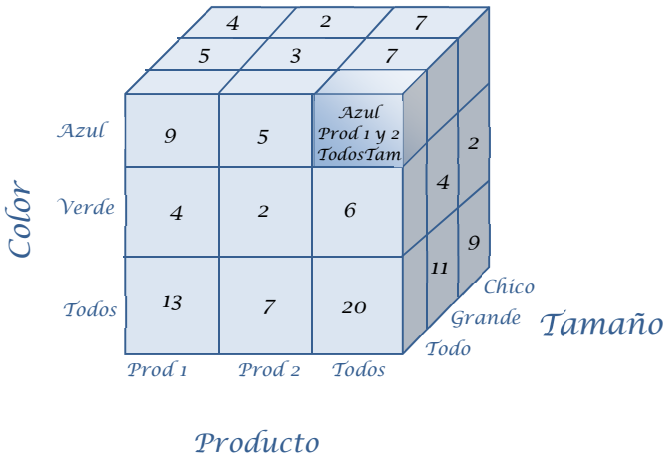


Ilustración 3. Cubo de Datos de 3 Dimensiones: Color, Tamaño y Producto, con agregado de Ventas

Para analizar y visualizar la información dentro de los cubos de datos se requiere realizar las siguientes operaciones:

- Generalizar y especializar (Roll-Up y Drill-Down). Esta operación permite navegar a través de las dimensiones con diferentes niveles de abstracción. De un bajo a un alto nivel tenemos la generalización y en sentido opuesto la especialización. Por ejemplo: en la ilustración un drill-down podría ser accesar al cubo que detalle los tonos de verde
- Corte de cubos (Slice y Dice). Son utilizados para obtener subcubos, como rebanadas o En Ilustración 4 se muestran algunos ejemplos tomando como referencia la Ilustración 3.

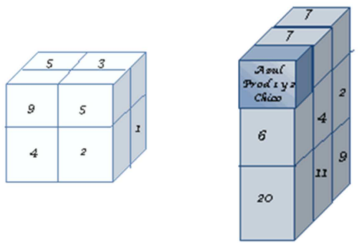


Ilustración 4. Ejemplo de operación Slice y Dice. Del lado izquierdo se observa un subcubo obtenido de una operación dice. Del lado derecho se muestra un subcubo obtenido de una operación Dice, ya que a diferencia del anterior aquí se tiene una rebanada del cubo origen.

- Filtrar. Selecciona datos de un cubo utilizando atributos específicos. Por ejemplo el cubo resaltado en la Ilustración 4 con azul fuerte es un filtrado resultado de pedir todos los productos, todos los tamaños en color azul.
- Pivotear. Permite rotar las dimensiones del cubo.

A pesar de que un cubo hace referencia a una estructura geométrica en 3D, en el almacenamiento de datos un cubo de datos es de n dimensiones. De tal forma que al cubo presentado en la Ilustración 3 podríamos aumentarle una cuarta dimensión de localización obteniendo un cubo de datos de 4D.

2.2.3 ESTRUCTURA DE DATOS ARBLIS

ANTECUMEM hace uso de una estructura de datos ligada comúnmente llamada **lattice** [Harinarayan, 1996] que se construye a través de niveles, estableciendo las combinaciones posibles generadas por medio de las dimensiones que se manejen en determinado cubo. Supongamos que tenemos cuatro dimensiones, donde cada bit representará una dimensión diferente por lo tendríamos una lattice como la que se muestra en la Ilustración 5.

Existen diferentes grados de agrupamiento, de tal forma que dado un conjunto de dimensiones, se pueden generar **cuboides** para cada posible subconjunto de las dimensiones. De tal forma que un cuboide se forma a partir de todas o algunas de las dimensiones que comprenden el cubo de datos, por lo que todas o algunas de las dimensiones pueden estar activan en determinado cuboide.

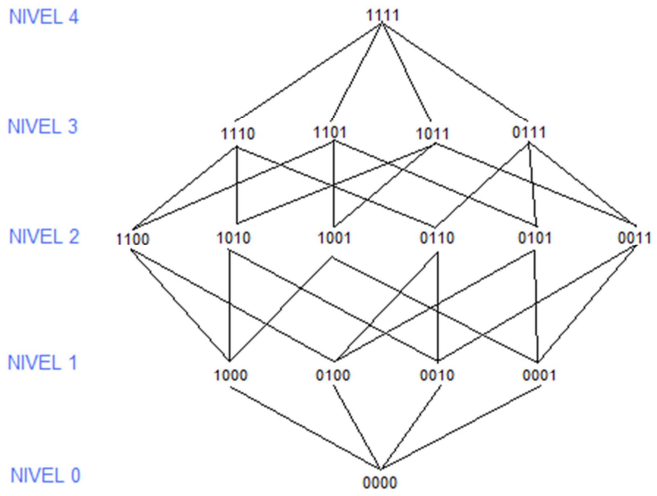


Ilustración 5. Lattice de cuboides hasta de 4D. Los unos representan las dimensiones activas y los ceros las dimensiones que no están activas. En el nivel cero ninguna dimensión esta activa.

En el primer nivel tenemos que solo una dimensión de las cuatro esta activa, de tal forma que en este nivel encontramos los llamados Cuboides de 1D (Una dimensión), en el segundo nivel tenemos dos dimensiones (Cuboides 2D) y así sucesivamente. Para cada nodo de la lattice se genera una estructura de datos ARBLIS en memoria principal, la cual es un almacén de la base de datos multidimensional a través de la cual se permite definir

operaciones con los cubos permitiendo que la herramienta conteste diversas preguntas de negocio del tipo puntual, rango, eficiencia, eficiencia grupal, conservación/pérdida, temporalidad y tendencias.

La estructura Arblis permite buscar en orden dentro de las dimensiones y con los ciclos predefinidos dado un valor en cada dimensión debido a la organización de la que hace uso, veamos el siguiente ejemplo:

Tomando las tres dimensiones de la figura Ilustración 3Ilustración 5 y agregando un valor de *Mediano* a la dimensión de *Tamaño*, tendríamos la tabla de la izquierda en la Ilustración 6 con dos elementos en la dimensión de *Producto*, dos en la de *Color* y tres en *Tamaño* cuyos totales en las combinaciones de estas tres están representados con *Ventas.*

La equivalencia con la estructura Arblis se muestra del lado derecho, donde la dimensión *Producto* está representada como el *Segmento 1*, *Color* como *Segmento 2* y *Tamaño* como *Segmento 3.* El *Segmento 1* apunta al *Segmento 2*, el *Segmento 2* apunta al *Segmento 3* y el *Segmento 3* apunta a los ***Hechos*** es decir las *Ventas* (sombreados con azul claro). Para la navegabilidad en la estructura se hace uso de los apuntadores por ejemplo la posición 0 apunta a la posición 2, la posición 2 apunta a la 6 y la posición 6 tiene los hechos.

Posición	*Producto*	*Color*	*Tamaño*	*Ventas*
0	Prod 1	Verde	Chico	63
1	Prod 1	Verde	Mediano	12
2	Prod 1	Verde	Grande	23
3	Prod 1	Azul	Chico	64
4	Prod 1	Azul	Mediano	72
5	Prod 1	Azul	Grande	18
6	Prod 2	Verde	Chico	46
7	Prod 2	Verde	Mediano	31
8	Prod 2	Verde	Grande	71
9	Prod 2	Azul	Chico	28
10	Prod 2	Azul	Mediano	9
11	Prod 2	Azul	Grande	54

Dimensión	Posición	Valor	Apuntador	Segmento
Producto	0	Prod 1	2	1
	1	Prod 2	4	
Color	2	Verde	6	2
	3	Azul	9	
	4	Verde	12	
	5	Azul	15	
Tamaño	6	Chico	63	3
	7	Mediano	12	
	8	Grande	23	
	9	Chico	64	
	10	Mediano	72	
	11	Grande	18	
	12	Chico	46	
	13	Mediano	31	
	14	Grande	71	
	15	Chico	28	
	16	Mediano	9	
	17	Grande	54	

Ilustración 6. Conversión a Arblis

2.3 VISTAS MATERIALIZADAS Y SU USO

Una vista [Han, 2001] es una configuración (abstracción) de dimensiones, definiendo las medidas que se analizan, los filtros y detalles establecidos. Las vistas no contienen datos en sí mismas, sino una referencia al modelo multidimensional con el cual fueron generadas, de tal forma que no existen físicamente en la base de datos. Para su creación y utilización se deben cumplir determinadas restricciones para la agregación y actualizaciones, a través del mantenimiento de las vistas.

La definición de las vistas permiten simplificar las consultas realizadas a la base de datos y en un SMBD (*Sistema Manejador de Base de Datos*) multiusuario permite tener seguridad en los datos restringiendo a los usuarios tener acceso a determinadas partes de la base de datos. Otra ventaja es la independencia de los datos ya que si la estructura de la base de datos es alterada podría aparentar no tener modificaciones a través de la implementación de vistas.

Las vistas están actualizadas con respecto a las relaciones de las que parten si en su creación no se hizo uso de funciones de agregación y si el SELECT no se realiza sobre todos los campos ya que al agregarse un nuevo campo en la base de datos este no se vería reflejado en la vista. La desventaja principal de las vistas es la reducción del rendimiento en el momento en que se define la vista ya que se deben combinar las tablas cada vez que se accede la vista realizando un procesamiento adicional. Una de las soluciones a dicho problema es la **materialización de vistas**.

La materialización de vistas consiste en almacenar la definición de la vista y los datos que regresa en disco duro, de tal forma que solo se realiza una carga inicial y la actualización de los datos dentro de ésta para llevar un control en el **mantenimiento de la vista**. Debido a que las vistas contienen físicamente los datos de las tablas base, es lógico que si cambian los datos de estas tablas no se reflejarán en la vista materializada. Para lo cual se necesita establecer un mecanismo de actualización automático en el que tendremos que definir el tipo y la forma de hacerlo.

Los sistemas de bases de datos [Perez,2008] proporcionan un lenguaje de definición de datos para especificar el esquema de la base de datos y un lenguaje de manipulación de datos para expresar las consultas y las modificaciones de la base de datos. Ambos se encuentran integrados en lenguajes de base de datos como puede ser SQL.

La sintaxis utilizada en SQL (*Structured Query Language*) [IOS, 1992] para la creación de las vistas materializadas es el siguiente:

```
CREATE MATERIALIZED VIEW nombre_vista
        [TABLESPACE nombre_tablespace]
        [PARALELL (DEGREE n)]
        [BUILD {INMEDIATE|DEFERRED}]
        [REFRESH {FAST|COMPLETE|FORCE|NEVER|ON COMMIT}]
        [{ENABLE|DISABLE} QUERY REWRITE]
AS
        SELECT campos
        FROM tablas
        WHERE condicion
```

BUILD permite establecer la forma en que los datos serán cargados a la vista:

- INMEDIATE permite que los datos se carguen después de crear la vista, si no se establece este parámetro se utilizará ésta como opción por defecto

- DEFERRED hace que se carguen los datos cuando se realice la primera actualización a la base de datos.

REFRESH define el método y la frecuencia de refresco de los datos.

- COMPLETE se refiere a borrar todos los registros contenidos en la vista y volver a insertar todos los registros que sean obtenidos a partir de la sentencia del SELECT definida durante la creación de la vista.

- FAST hace la actualización solo cuando hayan sido realizados cambios sobre las relaciones base de es vista desde la última actualización. Para llevar un registro de dichos cambies es necesario haber creado antes un LOG de la vista materializada, indicando los campos clave en los que se basará el mantenimiento de la vista. Este LOG es creado a través del siguiente comendo en SQL:

  ```
  CREATE MATERIALIZED VIEW LOG ON tabla_base
      WITH PRIMARY KEY
      INCLUDING NEW VALUES;
  ```

- FORCE se realiza una actualización del tipo FAST si es posible, de lo contrario se realizará una actualización tipo COMPLETE.

- NEVER implica que en ningún momento se realizará una actualización de la vista a pesar de las inconsistencias que esto pueda provocar.

- ON COMMIT es una actualización automática realizada al momento en que se hace una modificación a cualquiera de las tablas base

QUERY REWRITE permite que el optimizador de nuestra base de datos pueda reescribir las consultas. Es decir que el optimizador modifique internamente la consulta sobre una tabla dirigiendo la consulta hacia una vista materializada previamente creada de tal forma que permite una mejora en el rendimiento ya que devolverá los mismos datos en ambas formas.

Además existen dos tipos básicos de actualización:

- **Manual:** Manualmente se puede forzar a realizar una actualización usando l *REFRESH.*

  ```
  DBMS_MVIEW.REFRESH ('nombre_vista');
  ```

  Mientras que *REFRESH_DEPENDENT* actualiza todas las vistas materializadas que contengan determinadas tablas base

  ```
  DBMS_MVIEW.REFRESH_DEPENDENT ('tablas');
  ```

  Con la función *REFRESH_ALL_MVIEWS* se refrescarán todas las vistas materializadas de nuestra base de datos.

  ```
  DBMS_MVIEW.REFRESH_ ALL_MVIEWS
  ```

- **Automático:** esta actualización hace uso de *ON COMMIT,* explicado con anterioridad, y refresco programado donde se determina el momento en el tiempo en el que queremos la actualización a través de una fecha determinada y el intervalo de ésta.

En el proyecto la creación de las vistas permite formar estructuras de datos en forma de hiper-cubos minimizando el tiempo de respuesta en consultas complejas; ya que al extraer los datos de varias tablas y colocarlos en una única tabla transforma una consulta de muchas tablas a una consulta realizada sobre una sola tabla. Las vistas materializadas también admiten índices por lo que la implementación de estos permitiría mejorar el rendimiento de las operaciones realizadas a las vistas materializadas.

Para aumentar la eficiencia de los cubos de datos es necesario evaluar el tipo de implementación de las vistas que se tendrá, ya sea materializando el cubo completo, no materializar nada o materializar parte del cubo. Para esta última opción se evalúa a través de *lattices* el beneficio en tiempo de consulta al materializar una u otra vista.

2.4 COMPACTACIÓN DE DATOS

Implementar la compactación de datos en la herramienta ANTECUMEM tiene por objetivo reducir el espacio que ocupan los cubos de datos en la memoria principal, de tal forma que se permita aumentar la cantidad de datos a almacenar.

Las técnicas de compactación pueden ser específicas para su uso o de uso general, en cualquiera de los casos se utiliza la compactación y des-compactación para la entrada y salida de los datos respectivamente.

2.4.1 MÉTODOS DE COMPACTACIÓN

Algunos métodos de compactación son los siguientes:

2.4.1.1 Eliminación de los ítems de datos redundantes

Se basa en la eliminación de la redundancia textual o desglosada de los múltiples datos que se refieren a un solo término, como aquellos que pueden ser calculados a través de otros. Por ejemplo si se tiene la relación de la los campos de fecha de nacimiento, lugar de nacimiento y sexo pueden ser eliminados ya que se encuentran en CURP.

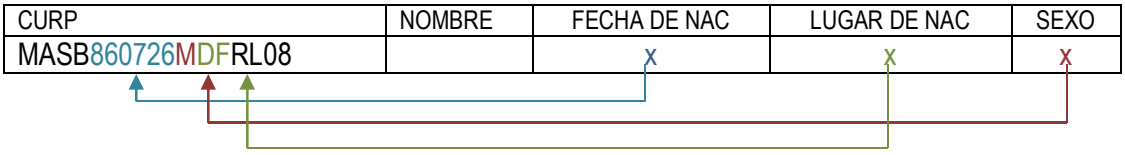


Ilustración 7. Eliminación de Datos Redundantes

De la misma forma se lleva a cabo la eliminación de campos repetidos en las relaciones (Véasellustración 8) de tal forma que tener una doble referencia al mismo campo, como dos veces el nombre del producto para el mismo ID_prod, no solo ocupa mayor espacio sino que también podría provocar inconsistencia en los datos.

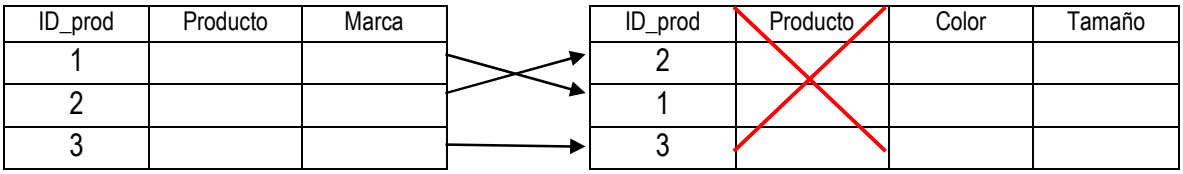


Ilustración 8. Eliminación de datos redundantes

2.4.1.2 Conversión a notación compacta

Utilizar los caracteres mínimos para cada campo aunque no sean entendibles a simple vista para los usuarios finales. Por ejemplo las fechas en lugar de almacenarlas como *15 Nov 1989*, podría equivaler a *151289*. Este tipo de compactación se aplica a elementos como los números de pieza y las direcciones postales.

2.4.1.3 Supresión de caracteres repetidos sucesivamente

Los datos contienen a menudo caracteres repetidos varias veces y pueden ser sustituidos por algún tipo de codificación, para ello se puede hacer uso de 2 bits: el primero que marque si el carácter se encuentra repetido y el segundo la cantidad de veces que se repite de forma consecutiva.

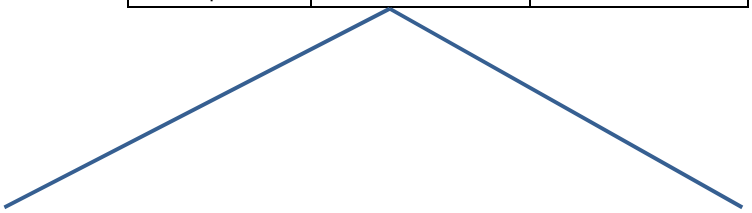
2.4.1.4 Evitar espacios vacíos

Debido a que en muchas ocasiones los registros son de longitud variable, se desperdicia mucho espacio vacío o en blanco, por lo que organizar los registros sin especificar una longitud fija sería una solución viable aunque el manejo de este tipo de archivos es más complejo.

2.4.1.5 Sustitución datos repetidos

Si existen valores repetitivos acotados, se puede utilizar un identificador para éste que permite ocupar menos espacio en el almacenamiento aunque disminuya la legibilidad (Véasellustración 9) Esto se logra mediante *catálogos* que son tablas que relacionan el valor extenso ("Chevrolet") con el valor pequeño ("A").

MARCA	AUTOMOTRIZ	UNIDADES
Chevy	Chevrolet	2563
Pointer	Volkwagen	4531
Fiat	Honda	3982
Optra	Chevrolet	1512



ID	AUTOMOTRIZ
A	Chevrolet
B	Volkwagen
C	Honda

MARCA	AUTOMOTRIZ	UNIDADES
Chevy	A	2563
Pointer	B	4531
Fiat	C	3982
Optra	A	1512

Ilustración 9. Sustitución de datos

2.4.1.6 Sustitución de texto idiomático

A través de este método se pretende sustituir cadenas de caracteres repetitivas por un código en donde se almacena texto escrito de gran longitud; donde al recibir esta sucesión de caracteres los convertirá en su representación con un código previamente determinado, el cual deberá de ser con sucesión de caracteres reservados para evitar errores al hacer el proceso inverso (Véasellustración 10).

Texto	Código
mente	&
sub	#
pre	%

Ilustración 10. Sustitución

La compactación empleada para la generación de la estructura AC hace uso de: eliminación de los ítems de datos redundantes, conversión a notación compacta, evitar espacios vacíos y sustitución de datos habitual. Está ultima junto con eliminación de datos redundantes son la de mayor impacto en la compactación realizada ya que permite trabajar con tipos de datos numéricos y de cadena. Por otra parte la notación compacta es usada principalmente para el manejo de fecha. En todos los caso se hace uso de evitar espacios vacíos.

Se ha considerado que la resolución de preguntas con rango no es eficiente si se realiza con cadenas de caracteres ya que conceptualmente es difícil que se pida un rango de los productos que van de "pan de caja" a "jabón en polvo" por mencionar un ejemplo.

2.4.2 ALGORITMOS DE COMPRESIÓN

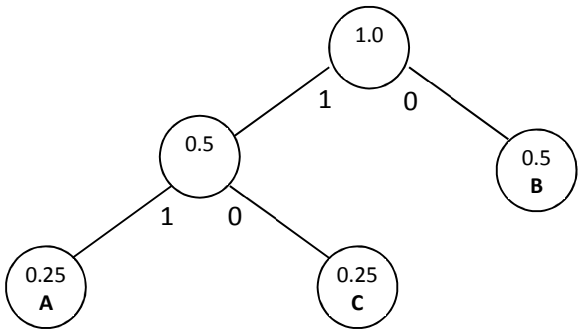
Los métodos de compresión mostrados con anterioridad hacen búsqueda de los bytes, caracteres, palabras o incluso frases más comunes como en las diversas sustituciones en los métodos mostrados anteriormente. En este apartado se describen algoritmos de compresión de longitud variable.

Las representaciones de los caracteres usualmente se maneja con longitud fija, por ejemplo el ASCII se maneja con 8 bits dando pie a 2^8 posibles representaciones de caracteres de la misma longitud. Para disminuir el espacio este algoritmo plantea utilizar códigos de representación de longitud variable.[Wayner, 2000]

2.4.2.1 Algoritmo de Huffman

El algoritmo de Huffman define un código variable para cada carácter sin la necesidad de utilizar un prefijo. De tal forma que aquellos caracteres que son más frecuentes se codifiquen con la menos cantidad de bits posibles mientras que aquellos que no sean tan frecuentes podrán tener una codificación un poco más amplia, pero siempre tendiendo a ser la menor posible.

Para ello se hace uso de una tabla de frecuencia donde se guardan los caracteres empleados en el texto y la cantidad de veces que son utilizados, así como un árbol binario que guardará el código generado por el algoritmo, donde los remplazos son bits dados a través de un árbol binario, donde el elemento de mayor repetición (frecuencia) [Huffman,1952] se encuentra en la cúspide del árbol como se muestra en la Ilustración 11 para lo cual hace uso de una tabla que contiene el elemento, la longitud de bits y el código correspondiente.



Carácter	Frecuencia	Código
A	2	11
B	9	0
C	5	10

Ilustración 11. Árbol y Tabla Generados de Algoritmo de Huffman

La descripción matemática del algoritmo de Huffman es: para un conjunto de símbolos $A = \{a_1, a_2, \dots, a_n\}$ con sus frecuencias (pesos) asociadas $P = \{p_1, p_2, \dots, p_n\}$, donde *n* es la cantidad de símbolos diferentes que existe en la cadena de entrada. Así pues los pesos asociados a los símbolos pertenecientes al conjunto A se encuentran en un rango comprendido entre 1 y n (entendiendo que la entrada no va a ser una cadena vacía), es decir:

$$p_i = peso(a_i), 1 \leq i \leq n$$

Obteniendo un conjunto de código binarios:

$$C(A, P) = \{c_1, c_2, \dots, c_n\}$$

Donde se asume que los mensajes han sido ordenados de tal forma que

$$P(1) \geq P(2) \geq \cdots \geq P(N)$$

De tal forma que para una código óptimo para que al carácter *a* de mayor aparición se le asigne la menor longitud en el código.

$$L(1) \leq L(2) \leq \cdots \leq L(N)$$

Para conseguir que el peso de C sea menor que el de A tal que, por lo que la longitud del mensaha es el número de dígitos asignados, siendo la longitud *L*:

Ecuación 1. Longitud del conjunto de códigos binarios obtenidos a partir del conjunto de símbolos y su peso.

$$L(C) = \sum_{i=1}^{n} p_i \times L(c_i)$$

2.4.2.2 Shannon-Fano

Casi simultáneamente surgen los trabajo de Claude Shannon en los Laboratorios Bell de la AT&T [Shannon, 1948] y Robert Fano en el MIT [Fano, 1949]. El algoritmo de Shannon y Fano construye el árbol de abajo hacia arriba según el siguiente algoritmo:

1) Para una secuencia de símbolos, se calcula la correspondiente lista de frecuencias de aparición de los símbolos
2) Se ordena la lista de símbolos según su frecuencia en orden decreciente
3) Se divide la lista en dos partes, de forma que la suma total de frecuencias de la mitad superior sea lo más cercana posible a la suma total de la parte inferior
4) A la mitad superior de la lista se le asigna el dígito binario 0, y a la mitad inferior se le asigna el dígito binario 1. Esto significa que los códigos de los símbolos en la primera mitad empezarán todos con 0 y los códigos en la segunda mitad empezarán todos con 1
5) Recursivamente se aplica el mismo procedimiento a cada una de las dos mitades, se subdivide en grupos y se agregan bits a las códigos hasta que cada grupo conste de un único símbolo

Posteriormente Shannon introduce cálculos estadísticos introduciendo una forma matemática de medir los nuevos valores de un bit en particular de información dada por:

Ecuación 2. Cálculo de número de bit para la compresión de Shannon y Fano

$$\sum_{x \in A} p(x) \log_2 \frac{1}{p(x)}$$

Aplicando Ecuación 2 al ejemplo de la Ilustración 11 tendríamos:

$$\frac{\log_2 2}{2} + \frac{\log_2 4}{4} + \frac{\log_2 4}{4} = 1.5$$

Donde 1.5 representa la entropía[1], además de ser el número promedio de bits usados por cada carácter al comprimir el archivo con la misma distribución de caracteres.

2.4.2.3 Lempel Ziv Welch

El algoritmo de Lempel Ziv Welch (LZW) fue creado por Lempel Ziv [Ziv, 1977] y mejorado por Terry Welch [Welch, 1984]: Consite en crear un diccionario de cadenas que se encuentran en un texto, al momento en que la cadena es leída de tal forma que evita la doble pasada de analiza y comprimir.

El algoritmo es el siguiente:

1) Leer un carácter
2) Si ese carácter concatenado con una cadena *s* existe en el diccionario entonces *s* será esa concatenación
3) Si ese carácter concatenado con una cadena *s* no existe en el diccionario entonces se añade esa concatenación al diccionario y *s* será el carácter.
4) Repetir 1,2 y 3 mientras exista un carácter.

Este algoritmo permite no guardar la equivalencia para su compresión y descompresión, ya que en el momento que se va leyendo se va generando el diccionario.

A partir de los algoritmos mencionados han surgido mejoras e incrementos aplicables a diversas circunstancias, de tal forma que son parte de los algoritmos base para la creación de sistemas de compactación más complejos. Debido a que éstos sugieren el uso de datos de longitud variable, y el manejo realizado en Arblis no permite el uso de este tipo de datos, estos algoritmos no serán implementados ya que la complejidad y el tiempo de respuesta para la solución de preguntas se incrementaría.

2.5 HERRAMIENTAS TIPO MOLAP

OLAP (Procesamiento analítico en línea) [Agrawal, 1995] y [Ho, 1997] es un término que describe una tecnología que utiliza una vista multidimensional de datos agregados para proporcionar un rápido acceso a la información estratégica, con el propósito de realizar un análisis avanzado [Codd, 1993] ya que a partir de grandes volúmenes de datos multidimensionales se puede realizar un análisis de determinados datos a través de diversas técnicas y herramientas.

[1] Describe la cantidad de orden o desorden en un sistema

Para el procesamiento en línea existen distintas formas de almacenar los datos dándoles la estructura requerida para el análisis. En el caso de ROLAP (*Relacional Online Analytical Processing*) se cuenta con bases de datos **relacionales** que mediante un diseño específico permiten se haga análisis OLAP sobre los datos, por otra parte la arquitectura MOLAP (*Multidimensional Online Analytical Processing*) son bases de datos **multidimensionales** sin soporte relacional.

La arquitectura MOLAP almacena los datos en formato multidimensional a través de estructuras de datos espaciales, implementando cálculos en memoria y la escritura en el mismo cubo. Lo cual permite tener un gran rendimiento en cuanto a las consultas para lo cual los cubos están construidos para la recuperación rápida de datos.

Cuando el cubo es creado todos los cálculos están pre-generados lo cual hace eficiente la consulta. Sin embargo al construir el cubo se pueden derivar un gran volumen de datos por lo que se suele manejar un número limitado de datos. Es por ello que la compresión será aplicada sobre el prototipo ANTECUMEM que hace uso de la estructura Arblis.

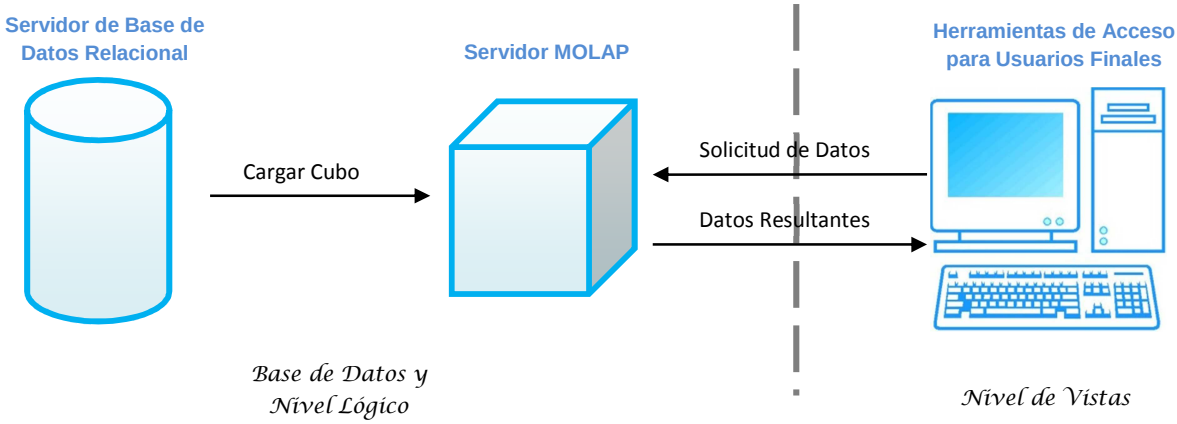


Ilustración 12. Arquitectura MOLAP

Debido a que la estrutura Arblis es del tipo MOLAP, se analizaran herramientas de este tipo a continuación:

2.5.1 HERRAMIENTA PALO

PALO Plan Analyse Report es [Raue, 2008] una herramienta de código abierto (*Open Source*) basada en la arquitectura MOLAP enfocada a soluciones de inteligencia de negocios a través de la planeación, análisis y reporte de datos, usando arquitectura cliente-servidor.

Puede ser utilizado a través de Jpalo que consiste en una implementación en java, así como en otros lenguajes de programación como lo son .NET, PHP y C++. Además cuenta con una integración a Excel a través de un ad-inn que permite la creación de cubos, vistas, elementos y jerarquías.

La creación del cubo de datos se lleve a cabo recibiendo parámetros como la base de datos y las dimensiones. Inicialmente calcula los bytes a ocupar y reserva el espacio necesario en la memoria cache, genera un índice, las dimensiones y finalmente pagina el cubo de datos creado, cargándolo completamente a memoria principal.

Se trabajó con la base de datos ejemplar de Ventas, en la en la cual se crea un cubo de seis dimensiones con 4,406,400 celdas de las cuales 264,591 tuplas contenían datos. Las dimensiones con su respectivo número de elementos es 17 meses, 9 años, 24 registros, 10 tipos de datos, 4 y 30 productos.

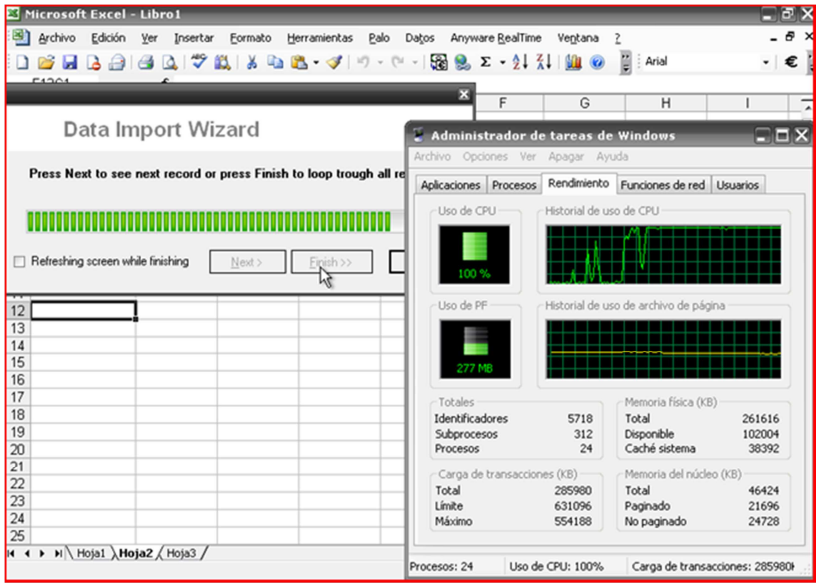
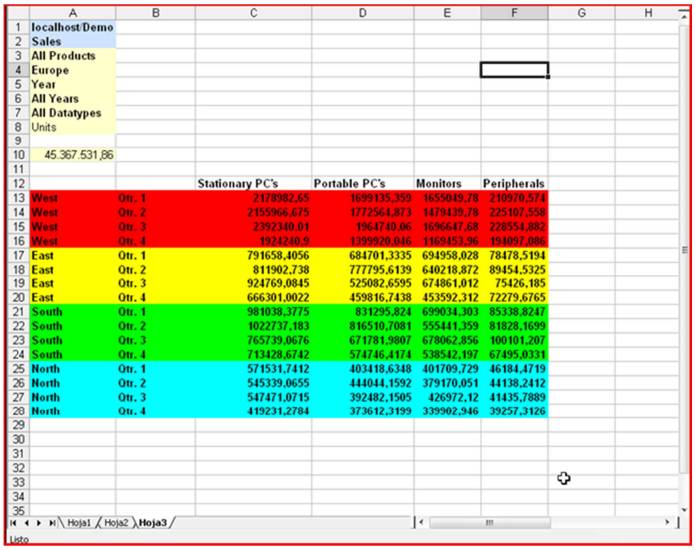


Ilustración 13. PALO, crear cubo

La creación del cubo tarda aproximadamente 32 segundos y ocupa 21572 KB al ser cargado.

Para el segundo ejemplo se cuenta con otra base de datos en la que se crea un cubo de cinco dimensiones: indent, depth, parents, children, children_weight y la creación del cubo ocupa 102572 KB durante un tiempo de 1 min 17 segundos.

Una vez cargado el cubo se insertan elementos en las diferentes jerarquías y se inserta la función correspondiente para que muestre los valores solicitados. Los datos se pueden mostrar en diversas dimensiones.



	A	B	C	D	E	F
1	localhost/Demo					
2	Sales					
3	All Products					
4	Europe					
5	Year					
6	All Years					
7	All Datatypes					
8	Units					
9						
10	45.367.531,86					
11						
12			Stationary PC's	Portable PC's	Monitors	Peripherals
13	West	Qtr. 1	2178982,65	1699135,359	1655049,78	210970,574
14	West	Qtr. 2	2155966,675	1772564,873	1479439,78	225107,558
15	West	Qtr. 3	2392340,01	1964740,06	1696647,68	228554,882
16	West	Qtr. 4	1924240,9	1399920,046	1169453,96	194097,086
17	East	Qtr. 1	791658,4056	684701,3335	694958,028	78478,5194
18	East	Qtr. 2	811902,738	777795,6139	640218,872	89454,5325
19	East	Qtr. 3	924769,0845	525082,6595	674861,012	75426,185
20	East	Qtr. 4	666301,0022	459816,7438	453592,312	72279,6765
21	South	Qtr. 1	981038,3775	831295,824	699034,303	85338,8247
22	South	Qtr. 2	1022737,183	816510,7081	555441,359	81828,1699
23	South	Qtr. 3	765739,0676	671781,9807	678062,856	100101,207
24	South	Qtr. 4	713428,6742	574746,4174	538542,197	67495,0331
25	North	Qtr. 1	571531,7412	403418,6348	401709,729	46184,4719
26	North	Qtr. 2	545339,0655	444044,1592	379170,051	44138,2412
27	North	Qtr. 3	547471,0715	392482,1505	426972,12	41435,7889
28	North	Qtr. 4	419231,2784	373612,3199	339902,946	39257,3126

Ilustración 14. PALO, dimensiones

A partir de los datos abstraídos se crean la graficas con las herramientas que cuenta Excel.

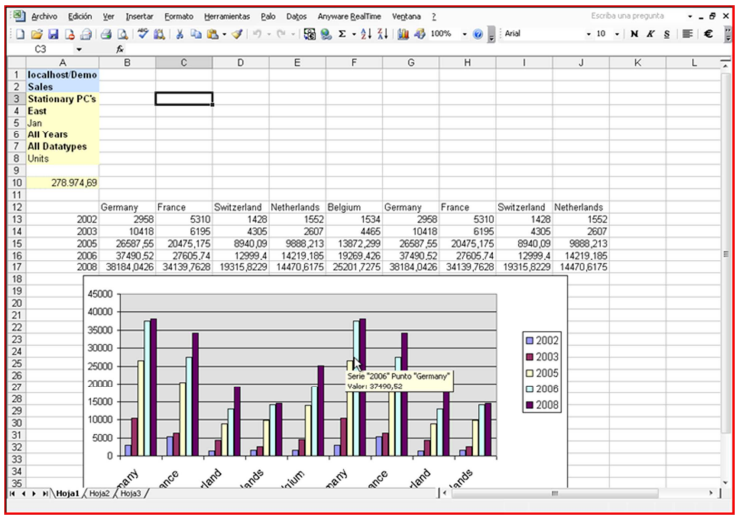


Ilustración 15. PALO, mostrar datos

2.5.2 HERRAMIENTA APPLIX

Esta plataforma que implementa MOLAP permite a los usuarios visualizar y comprender rápidamente grandes conjuntos de complejos datos. La mayor diferencia de Applix con respecto a otras herramientas es que no pre-calcula los cubos de datos en el lado del cliente lo que disminuye el espacio en disco de los usuarios finales.

Applix [Applix 2007] y [Henderson,200] es capaz de dar respuestas de forma casi inmediata por la manipulación de los datos en memoria, incluso en consultas fuera de rango, al igual que PALO cuenta con una integración hacia Excel, la cual la mayoría de las veces es familiar y de fácil entendimiento, como la que se muestra en la Ilustración 16. Applix TM1 en Excel, donde la manipulación de los datos es similar a la usada en PALO.

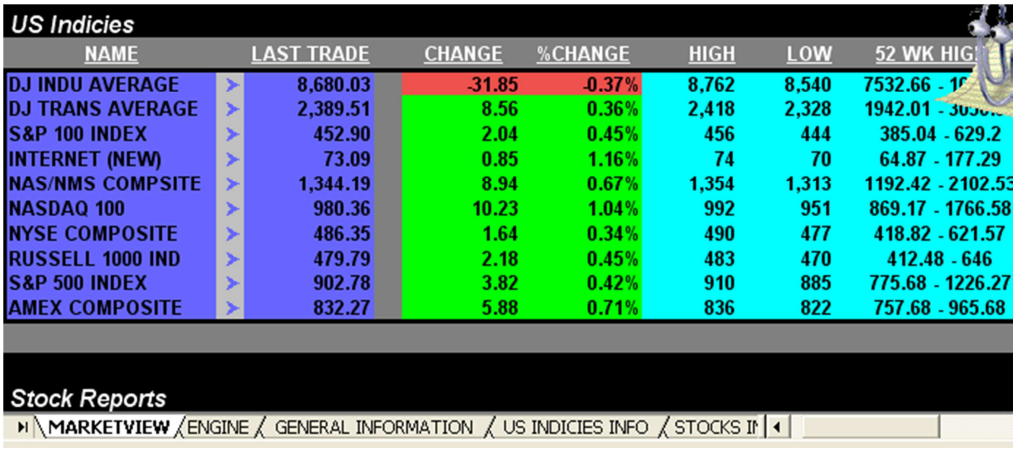


Ilustración 16. Applix TM1 en Excel

Además de la interfaz en Excel se cuenta con Applix TM1 Web, TM1 Integra y Turbo Integrator. Debido a que está disponible gratuitamente en Web las pruebas que se realizaron fueron sobre esta plataforma para sacar mediciones con respecto al tiempo y espacio, que permite visualizar respuestas a las preguntas de negocio predeterminadas donde el tiempo de carga promedio es de 7 segundos.

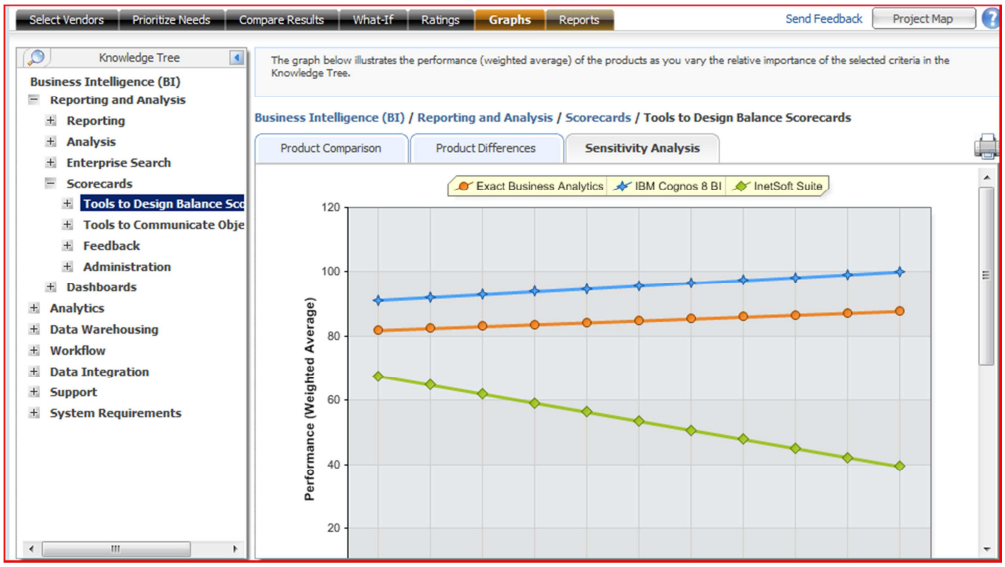


Ilustración 17. Applix TM1 Web

De la misma forma permite resolver comparaciones y preguntas sobre aquellas preguntas que no han sido calculadas con anterioridad, las cuales pueden ser manipuladas (what-if). Debido a que se calcula todo, el tiempo de respuesta es mayor: entre 9 y 27 segundos, con un uso de 212 MB en memoria física.

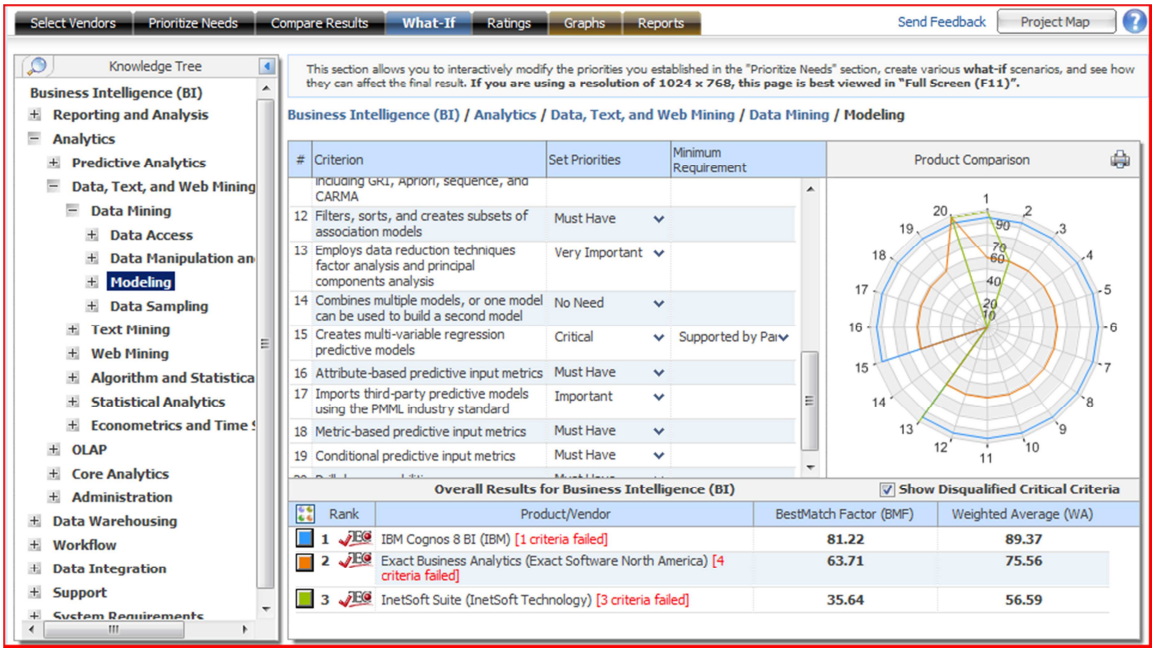


Illustration 18. Applix TM1 Web en What-If

Debido a que deja de hacer pre-cálculos cargando los datos en memoria principal y ejecutando las memorias en el instante sin embargo trajo limitaciones en la RAM donde hace todo el cálculo, aumentando el tiempo de respuesta presentado un problema similar con el que cuenta Arblis actualmente. Presenta el problema que al incrementarse el tamaño, la cantidad de memoria principal requerida también aumenta; y tomando en cuenta que un sistema operativo de 32-bit tiene un límite de 3 GB en RAM (en Windows) o 4 GB de RAM (UNIX), tal vez no solo se deba incrementar la RAM sino sea necesario moverse a un sistema operativo de 64 bit.

2.5.3 HERRAMIENTA ESSBASE

Hyperion Essbase TM1 [Oracle Corporation,2008] es una herramienta MOLAP desarrollada por Oracle que permite la creación de consultas y análisis de los datos para encontrar excepciones y patrones en éstos.

Esta herramienta hace que sus índices sean paginados en la memoria local para evitar cargar completamente a RAM, permitiendo el uso de índices dentro de una base de datos más grande. La optimización de la base de datos en el servidor depende de las reglas bajo las que hayan sido construidas las dimensiones por parte del cliente.

Cuando los datos son cargados, la herramienta re-calcula los valores en cubos multidimensionales y re-construye los índices, cuya indexación es independiente al de las tablas de la base de datos. La pre-construcción permite que no afecte el rápido crecimiento aunque la parte de consolidación de los datos puede ser más tardada. Este pre-cálculo permite que sea más rápida que herramientas como Applix en bases de datos con gran número de dimensiones.

Essbase cuenta con Essbase Visual Explorer a través de la cual se pueden manipular los datos para ser mostrados.

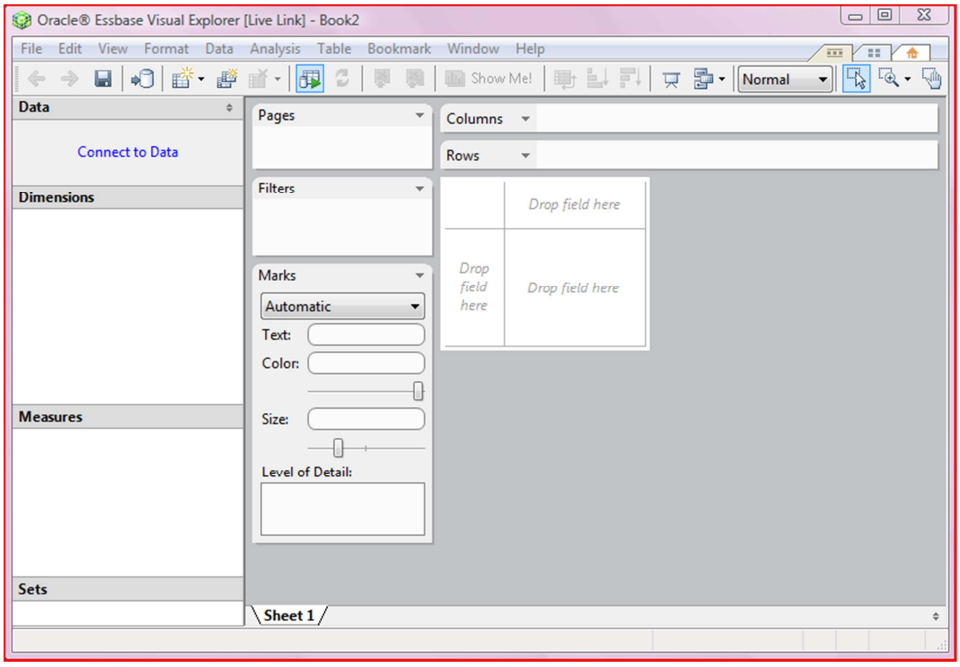


Ilustración 19. Essbase Visual Explorer

Se trabajo con una conexión a Oracle para la visualización de los datos de la base de datos de SH, donde los tiempos variaban dependiendo de las dimensiones del cubo que se creara, las restricciones y el conjunto de tablas que era tomado en cuenta.

2.6 DESCRIPCIÓN DE BASES DE DATOS

Para la realización e implementación de este proyecto, se hará uso de ejemplos de bases de datos, específicamente de: SH (Sales History).

2.6.1 BASE DE DATOS SH

SH cuenta con un historial de ventas de determinada empresa dedicada a la venta de productos cuyos precios varían según la temporada y promociones, además de contar con un registro de sus clientes.

Originalmente la base de datos cuenta con 1,058,884 registros divididos en las siguientes tablas:

- TIMES 1826 registros
- PRODUCTS 72 registros
- COUNTRIES 23 registros
- CHANNELS 5 registros
- PROMOTIONS 503 registros
- CUSTOMERS 55500 registros
- SALES 918843registros
- COSTS 82112 registros

El diagrama que describe esta base de datos se muestra en la Ilustración 20. Diagrama de SH

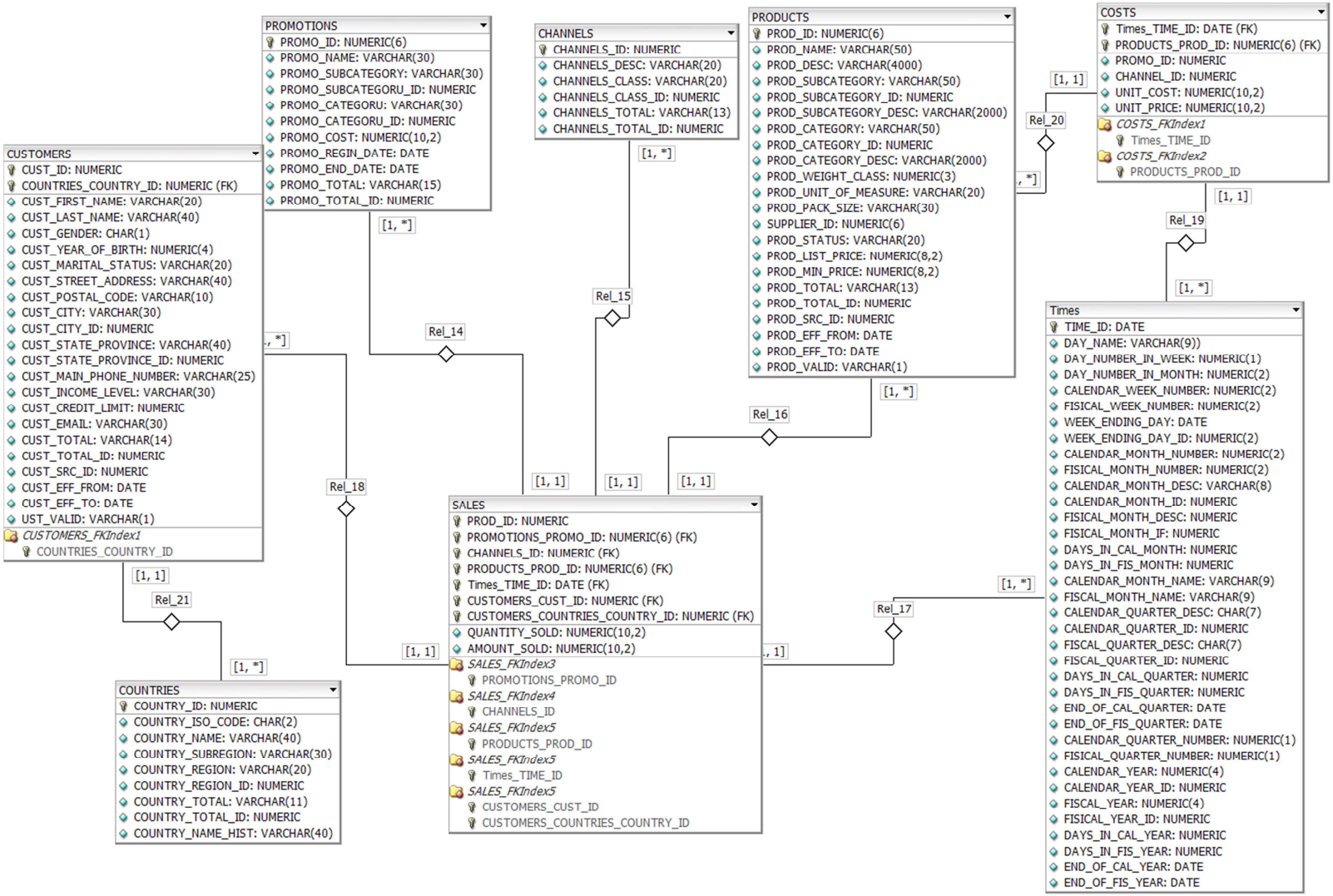


Ilustración 20. Diagrama de SH

2.7 RESUMEN

La necesidad de manipulación de la información en almacenes de datos ha ido aumentando, ya que actualmente no solo se espera contar con el almacenamiento de la información relevante para las empresas e instituciones, sino que es importante contar con datos específicos a buscar en esa gran cantidad de información así como dar respuestas a preguntas de negocio específicas como lo hace ANTECUMEM.

Sin embargo, para resolver estas cuestiones en tiempo real se han ido sacrificando el espacio en memoria principal para lo que es importante conocer las técnicas de compactación planteadas por varios autores enfocadas a bases de datos y aquellos algoritmos usados para comprimir archivos de datos.

ANTECUMEM almacena los datos en Arblis que es una estructura creada en cada nodo de la lattice, la cual guarda la información en una especie de tabla con apuntadores que permiten encontrara rápido datos específicos, así como, la eficiente construcción de cubos de datos (organización de múltiples datos).

Como se ha mencionado ANTECUMEM ocupa espacio en memoria principal, defecto que comparte con otras herramientas como Applix, mientras que otras herramientas como lo son Palo prefieren sacrificar el tiempo por el espacio. Por otra parte Essbase realiza técnicas de indexación al cargar los cubos de datos. Por lo que es evidente optimizar los problemas de tiempo y espacio sin que resulten excluyentes.

Finalmente se hace una breve descripción de la estructura de las bases de datos que serán usadas como ejemplo para la realización del proyecto.

CAPÍTULO III. ANÁLISIS Y DISEÑO DE LA APLICACIÓN

3.1 INTRODUCCIÓN

En este capítulo se muestra el análisis y diseño de las transformaciones a la herramienta ANTECUMEM (Análisis Temporal de Cubos en Memoria), con el objetivo de modificarla para mejorar el espacio en memoria principal y en disco duro así como mejorar la interfaz. Los cambios a la herramienta se centra en la reducción del espacio ocupado por los datos en memoria principal, lo cual se logra a partir del rediseño de la estructura de datos Arblis, la cual además de permitir una navegación rápida sobre ella misma para la resolución de preguntas de negocio, logrará ocupar un espacio reducido en la memoria aumentando así la cantidad de datos que pueda almacenar. La compactación conserva las propiedades de Arblis por lo que la creación de la compactación debe ser monotónica e homomorfa [véase 4.2 Propiedades de Arblis].

La ejecución de la herramienta consta de tres fases, en la primera se realiza la compactación creando la estructura y catálogos requeridos para la manipulación y uso de los datos; la segunda es cargar en memoria principal la estructura y sus descriptores para lo cual se determina los nodos de la lattice que serán creados de manera multidimensional a través de Arblis comprimida. La tercera fase consta en la resolución de preguntas de negocio haciendo uso de los cubos de datos creados.

La descripción de ambas fases con respecto a la compactación de datos, así como el rediseño de la interfaz gráfica, utiliza el modelo orientado a objetos[1] por lo que se hace uso de los diagramas de: casos de uso, diagramas de secuencia, diagramas de clases, diagramas de actividades y diagramas de paquetes que permiten un bosquejo del modelado de la parte de la aplicación a implementar.

3.2 ARQUITECTURA DE ANTECUMEM

La herramienta ANTECUMEM responde preguntas de negocio en un tiempo reducido debido a la forma con la que desarrolla el análisis de datos. Las preguntas de negocio son búsquedas a las bases de datos para hallar datos valiosos en la toma de decisiones. Los datos se almacenan en memoria principal para lo que se emplea la estructura Arblis, la cual representa los cubos de datos. Las preguntas de negocio a las que ANTECUMEM responde son del tipo:

1. ***Puntual.*** Proporcionar el valor de un elemento en un entorno en específico.
2. ***Rango.*** Proporcionar elementos acotado por los rangos.
3. ***Eficiencia.*** Calcular el incremento o decremento en dos períodos con rangos.
4. ***Eficiencia Grupal.*** Mide la eficiencia en los n elementos en dos periodos.
5. ***Conservación/Perdida.*** Mide la eficiencia grupal observando los n elementos que permanecen o que desaparecen en diversos período.
6. ***Temporalidad.*** Proporcionar los n "mejores" elementos que permanecen, en dos o más períodos de tiempo.
7. ***Tendencias.*** Buscar los n elementos de interés que mantienen una tendencia en p lapsos de tiempo.

La reducción del tiempo se basó en la localización y modificación de las fases de OLAP y Minería de Datos que podrían ser reducidas, así como en el diseño de almacén de datos en memoria con la estructura Arblis, modelando en ésta los cubos y bases de datos multidimensionales, por lo que la arquitectura de ANTECUMEM es la que se muestra en la Ilustración 1.

[1] Pressman, Roger**. Ingeniería de Software un Enfoque Práctico,** McGraw Hill, México, 2005 (1982), 6ª edición.

Los apuntadores en memoria y dimensiones en memoria se refieren a la transformación que se le realiza a la base de datos pasando a ser una base de datos multidimensional, por lo que desde ahora se trabajara con dimensiones y los valores a sumar los llamaremos hechos.

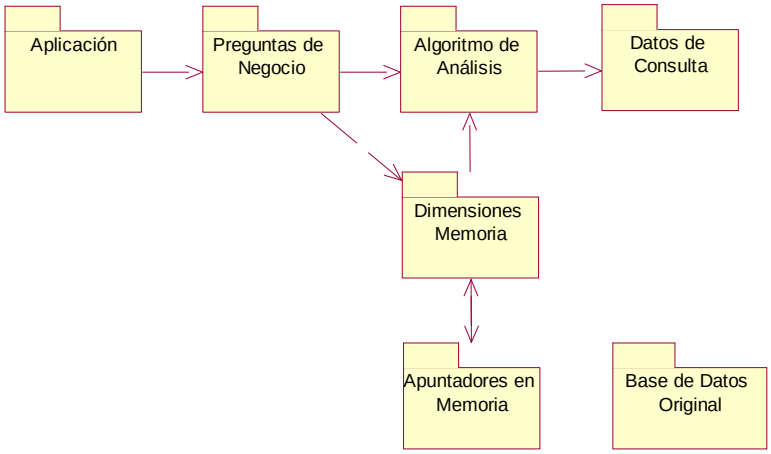


Ilustración 1. Arquitectura Inicial ANTECUMEM

La diferencia de ANTECUMEM con respecto al modelo tradicional OLAP recae en cargar en RAM la base de datos para responder cualquier consulta, a través de la estructura Arblis, ya mencionada. El espacio que ocupa en RAM podría ser mejorado con la implantación adecuada de métodos de compactación, por lo que la arquitectura propuesta para el mejoramiento de ANTECUMEM se presenta en Ilustración 2. Arquitectura Propuesta de ANTECUMEM.

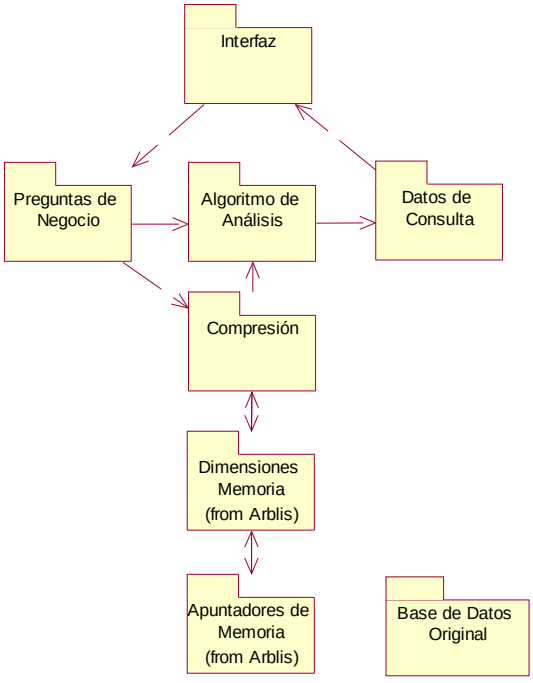


Ilustración 2. Arquitectura Propuesta de ANTECUMEM

Las modificaciones para mejorar el espacio se encuentran en la creación de un módulo para la compactación que influirá en el diseño de los apuntadores en memoria de la estructura Arblis. Además se modifica la Interfaz para hacerla agradable al usuario, con el objetivo de mostrar las respuestas a las preguntas de negocio de manera visualmente comprensible. Dichas modificaciones se detallan a lo largo de este capítulo.

3.2.1. CASOS DE USO POTENCIAL

Las acciones (requisitos potenciales) que se utilizan como base para el desarrollo de la herramienta permite obtener los casos de uso (CdU) potenciales, éstos últimos describen la interacción típica entre un usuario (actores) y el sistema (prototipo a crear). Los requisitos potenciales a desarrollar son básicamente: compactación e interfaz.

Para la compactación tenemos dos casos de uso potenciales, la primera se realiza al elegir la base de datos y formarse la estructura Arblis comprimida para ser cargada en memoria principal, y la segunda es en el momento de realizar la consulta donde la búsqueda debe ser sobre los datos requeridos compresos. En cuanto a la interfaz, son dos los caso de uso potenciales, uno es la de entrada de los datos requeridos para el tipo de pregunta establecida y el otro es la de salida, que servirá para mostrar las respuestas en modo texto y gráficamente si la pregunta lo permite.

A continuación se describen los cuatro casos mencionados con su respectiva descripción:

Creación de la Estructura AC

La base de datos que el usuario selecciona es analizada por una herramienta de lattices que proporciona a ANTECUMEM los nodos que serán usados para la creación de Arblis en memoria y materializada. La estructura Arblis materializada proveniente de ANTECUMEM es tomada por el módulo que se desarrolla para ser compactado.

Arblis es creada a partir de los métodos de compactación antes descritos (véase 2.4 COMPACTACIÓN DE DATOS); para la compresión que se lleva a cabo en las dimensiones se hace uso de: eliminación, 2.4.1.2 Conversión a notación compacta, 2.4.1.4 Evitar espacios vacíos y 2.4.1.5 Sustitución datos repetidos, llevando a cabo un análisis primero por el tipo de datos y después por frecuencias. Para esto se crean los arreglos 3 y 4 adicionales de Arblis mostrados en la Ilustración 51. Ejemplo de . A través de algoritmos mostrados en 5.2 CREACIÓN DE CATÁLOGOS y 5.3 CREACIÓN DE LA ESTRUCTURA AC.

En el caso de uso se muestra que el usuario selecciona la base de datos, ANTECUMEM obtiene de la herramienta de lattices los nodos que creará con Arblis y genera Arblis materializada. El módulo que se aporta a este prototipo tomará Arblis modificada y se compactará generando los descriptores de compresión para cada una de las dimensiones (catálogos) y posteriormente la estructura AC (Arblis Compresa).

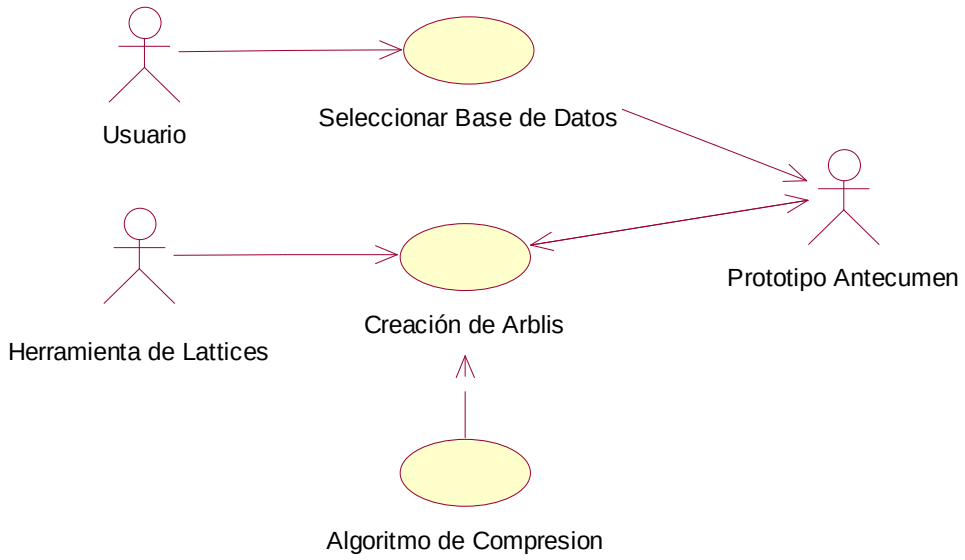


Ilustración 3. CdU Creación ESTRUCTURA AC: ANTECUMEM genera Arblis materializada a partir de la BD dada por el usuario a la cual se le realizará la compresión adecuada.

RF- <id del requisito>	Crear la estructura Arblis comprimida	
Nombre	Creación de Arblis	
Versión	1.3	
Autores	Bella Citlali Martínez Seis	
Fuentes	Especificación de Requisitos	
Objetivos asociados	Comprimir Arblis	
Propósito	Basarse en la estructuración de Arblis para comprimirla, de tal forma que ocupe en el menor espacio posible en memoria principal al ser cargada.	
Descripción o Visión General del caso de uso	Se crea la estructura Arblis por cada nodo de la lattice requerido, y se comprimen los datos que la componen	
Tipo	1. Primario 2. Esencial	
Referencias	Estructura Arblis actual	
Actores participantes	Usuario inicia el caso de uso, Herramienta de Lattices, Prototipo ANTECUMEM	
Precondición o Condiciones Iniciales	Contar con los nodos que se materializaran	
Secuencia Normal o Flujo de Eventos	**Paso**	**Acción**
	1	El usuario selecciona la Base de Datos
	2	El sistema verifica la existencia de la Base de Datos
	3	El sistema obtiene de Herramienta de Lattices los nodos que creará con Arblis compresa.
	4	El sistema comprime dimensione y hechos
	5	El sistema crea Arblis con dimensiones y hechos compresos
	6	El sistema carga a memoria Arblis si así fue requerido para prototipo Ante cume

Pos condición o Condición de Salida	Tener Arblis comprimida en memoria principal
Importancia	Vital
Urgencia	Inmediatamente
Comentarios	Los métodos de compactación dependen de un análisis de frecuencia y tipos de datos.

Tabla 1. CdU Creación de Arblis

Resolución de Preguntas de Negocio

Este caso de uso muestra (véase Ilustración 4 y Tabla 2) la compactación de los valores introducidos a través del usuario en los tipos de preguntas y que son recibidos por el prototipo ANTECUMEM, posteriormente realiza la búsqueda en la estructura AC (Arblis compresa) que retorna los valores de los hechos también compactados y deben de ser descompactados.

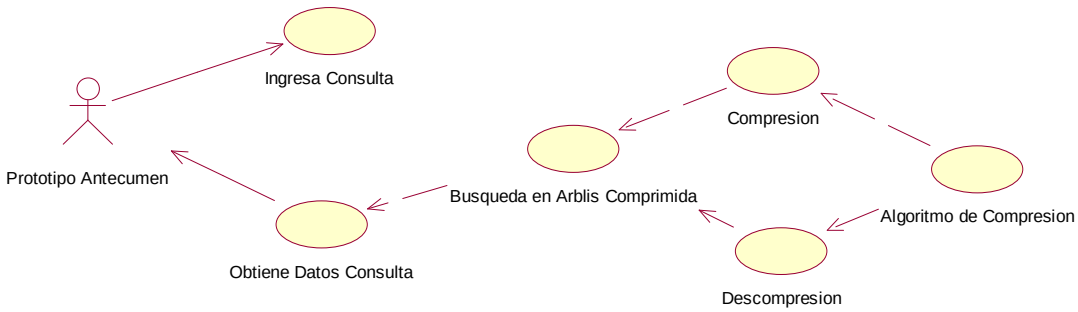


Ilustración 4. CdU Compactación de Valores

RF- <id del requisito>	Compactación y descompactación de valores para la consulta en Arblis comprimida	
Nombre	Compactación de Valores	
Versión	1.3	
Autores	Bella Citlali Martínez Seis	
Fuentes	Especificación de Requisitos	
Objetivos asociados	Comprimir para buscar en Arblis	
Propósito	Aumentar la eficiencia en la búsqueda disminuyendo el espacio que ocupa en memoria principal.	
Descripción o Visión General del caso de uso	El sistema obtiene las dimensiones a utilizar así como los hechos a buscar, lo que es comprimido, buscado en Arblis y finalmente regresa hechos con valores reales.	
Tipo	1. Primario 2. Esencial	
Referencias	Herramienta ANTECUMEM, específicamente los datos de la captura	
Actores participantes	Prototipo ANTECUMEM inicia el caso de uso	
Precondición o Condiciones Iniciales	Tener la Estructura Arblis correspondiente en memoria.	
Secuencia Normal o Flujo de Eventos	**Paso**	**Acción**
	1	Prototipo ANTECUMEM ingresa la consulta, es decir las dimensiones y hechos requeridos
	2	El sistema comprime las dimensiones

	3	El sistema realiza la búsqueda de los hechos a encontrar recorriendo Arblis compactada (AC)
	4	El sistema obtiene hechos compactados
	5	El sistema descompacta hechos según algoritmo usado
	6	El sistema regresa valores reales de los hechos a prototipo ANTECUMEM
Pos condición o Condición de Salida	Regresa datos de la consulta	
Importancia	Vital	
Urgencia	Inmediatamente	
Comentarios	Se regresan valores de datos de las consulta para ser analizados por el prototipo ANTECUMEM	

Tabla 2. CdU Compactación de Valores

Ingresar Preguntas

Este caso de uso representa una interfaz en la que el usuario selecciona el tipo de pregunta a resolver a partir de la base de datos seleccionada, dependiendo de dicha selección se solicitan valores según el tipo de pregunta cómo se muestra en Tabla 3.

Tipo de Pregunta	Dimensión a Analizar	Total de Datos a Mostrar	Total de Temporada 1	Tipo de Dato a Mostrar 1	Dimensión y Tipo de Temporada	Total de Temporada 2	Tipo de Dato a Mostrar 2	Rango 1	Rango 2
Puntual								♦	
Rango								♦	
Eficiencia		♦	♦					♦	♦
E. Grupal	♦	♦	♦			♦	♦	♦	♦
Conservación/ Pérdida	♦	♦	♦			♦	♦	♦	♦
Temporalidad	♦	♦	♦	♦	♦	♦	♦	♦	
Tendencias	♦	♦	♦	♦	♦		♦	♦	

Tabla 3. Parámetros para 7 preguntas

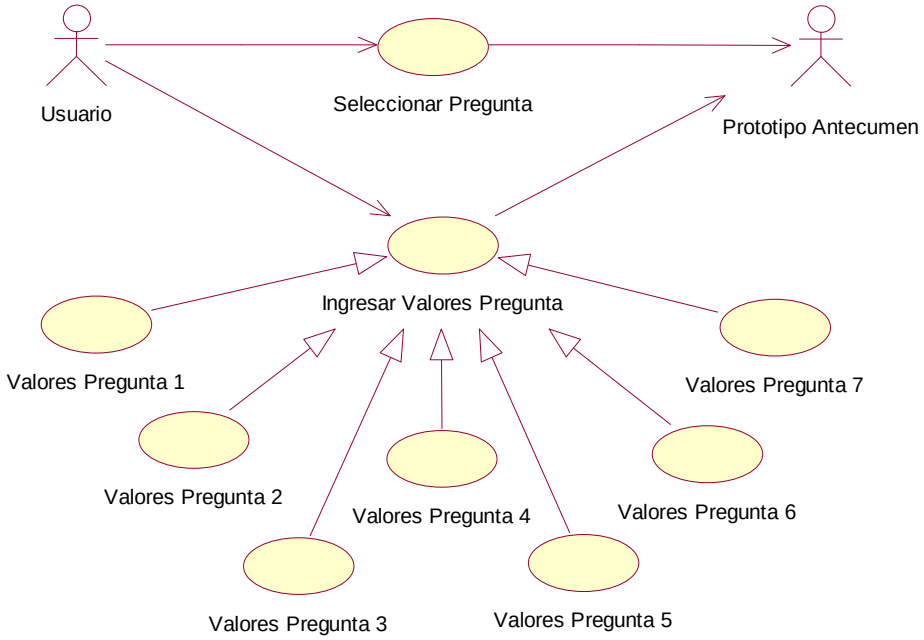


Ilustración 5. CdU ingresar preguntas

RF- <id del requisito>	Seleccionar valores en preguntas de negocio	
Nombre	Ingresar Preguntas	
Versión	1.1	
Autores	Bella Citlali Martínez Seis	
Fuentes	Especificación de Requisitos	
Objetivos asociados	Permitir al usuario seleccionar los parámetros de su interés para el análisis.	
Propósito	Que el usuario seleccione en una interfaz agradable los parámetros necesarios para que ANTECUMEM realice las operaciones correspondientes.	
Descripción o Visión General del caso de uso	El usuario selecciona inicialmente la bases de datos sobre la que desea trabajar para posteriormente seleccionara la pregunta de negocio deseada a responder de las 7 posibles bajo las condiciones que este establezca, ya sea el o los elemento y el periodo de tiempo.	
Tipo	1. Secundario 2. Esencial	
Referencias	Herramienta ANTECUMEM, específicamente la captura de la expresión	
Actores participantes	Usuario es quien inicia caso de uso, y prototipo ANTECUMEM	
Precondición o Condiciones Iniciales	La base de datos requerida debe de existir.	
Secuencia Normal o Flujo de Eventos	**Paso**	**Acción**
	1	El usuario selecciona un tipo de pregunta
	2	El sistema solicita valores dependiendo del tipo de pregunta que haya ingresado

	3	El usuario ingresa los valores solicitados
Pos condición o Condición de Salida	Se cuanta con los datos requeridos para la consulta y el análisis	
Excepciones o Alternativas equiprobables	**Paso**	**Acción**
	1	Si el usuario no introduce todos los valores solicitados el sistema tomara valores por default.
Importancia	importante	
Urgencia	Inmediatamente	
Comentarios	Solo se realizará la interfaz, los valores introducidos por el usuario llegan a la actual herramienta ANTECUMEM.	

Tabla 4. CdU ingresar preguntas

Respuestas Visuales

Este caso de uso se muestra que la graficación debe de realizarse una vez que se tienen los resultados del análisis que realiza ANTECUMEM, para el tipo de pregunta puntual la visualización de los datos es en modo texto y resto de las preguntas serán mostradas en ambos modos: texto y gráficamente. El ingreso de los datos también es un caso de uso ya que la interfaz es dinámica debido a que va cambiando dependiendo del número de dimensiones empleadas.

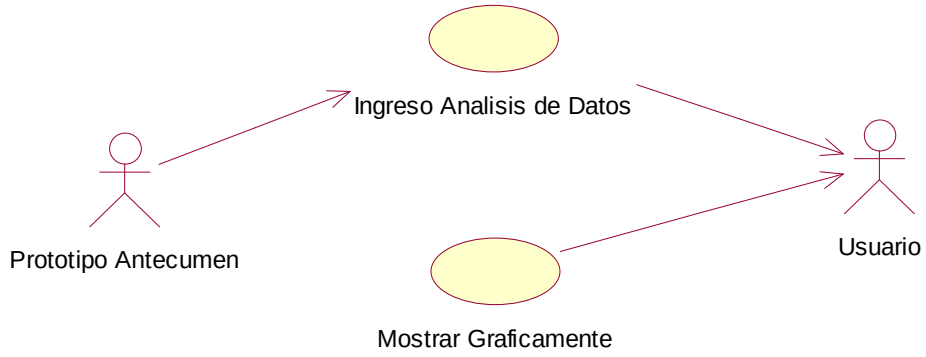


Ilustración 6CdU Respuestas Visuales

RF- <id del requisito>	Mostrar con texto y graficas las respuestas obtenidas	
Nombre	Respuestas visuales	
Versión	1.1	
Autores	Bella Citlali Martínez Seis	
Fuentes	Especificación de Requisitos	
Objetivos asociados	Mostrar respuestas con gráficas y textos	
Propósito	Facilitar la compactación de los resultados a través de entorno visual	
Descripción o Visión General del caso de uso	Cuando el prototipo ANTECUMEM de los resultados del análisis se deberá realizar la graficación de las respuestas y la presentación del texto.	
Tipo	1. Secundario 2. Esencial	
Referencias	Herramienta ANTECUMEM, específicamente el resultado del análisis	
Actores participantes	Prototipo ANTECUMEM y Usuario	
Precondición o Condiciones Iniciales	El prototipo ANTECUMEM debió de haber terminado el Análisis proporcionando la respuesta	
Secuencia	**Paso**	**Acción**

Normal o Flujo de Eventos	1	El actor ANTECUMEM genera respuestas a las preguntas proporcionándolas para inicial este caso de uso
	2	Si la pregunta es puntual el resultado es en texto únicamente, si es de cualquier otro tipo de pregunta se realizará la gráfica correspondiente.
	3	Se muestra lo obtenido
Pos condición o Condición de Salida	El usuario visualiza las respuestas	
Importancia	Importante	
Urgencia	Inmediatamente	
Comentarios	Este caso de uso permite mostrar al usuario los resultados de una forma que puedan ser comprensibles con mayor facilidad	

Tabla 5. CdU Respuestas Visuales

3.2.2 DIAGRAMAS DE ACTIVIDADES

A través de estos diagramas lograremos observar la secuencia de las actividades realizadas en cada caso de uso descrito previamente, observando el flujo lógico de los datos. En el **¡Error! No se encuentra el origen de la referencia.** se detallan los algoritmos empleados para la compresión, descompresión y solución de preguntas para n dimensiones.

Resolución de Preguntas de Negocio

Se obtienen los valores seleccionados a partir del prototipo ANTECUMEM, la aplicación comprimirá esos valores para comenzar la búsqueda en la estructura Arblis previamente comprimida y cargada en memoria, la búsqueda regresará los hechos buscados de manera compresa, por lo que dichos valores deben ser descomprimidos y así obtener los datos reales. (Véase Ilustración 7)

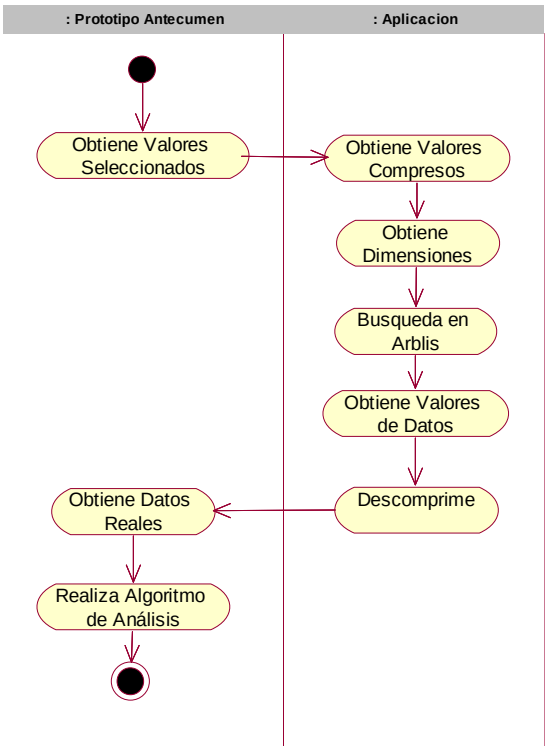


Ilustración 7. Diagrama de Actividades Resolución de Preguntas de Negocio

Creación de la ESTRUCTURA AC

Una vez seleccionada la Base de Datos por el usuario, el prototipo ANTECUMEM verifica su existencia ya que la Base de Datos será analizada por la Herramienta d lattices que regresa los nodos deseables a materializar. En este caso dichos nodos tendrán su equivalente estructura Arblis comprimida a partir de los hechos y dimensiones, posteriormente se carga la estructura a memoria principal o se materializa según el usuario lo haya especificado. (Véase Ilustración 8)

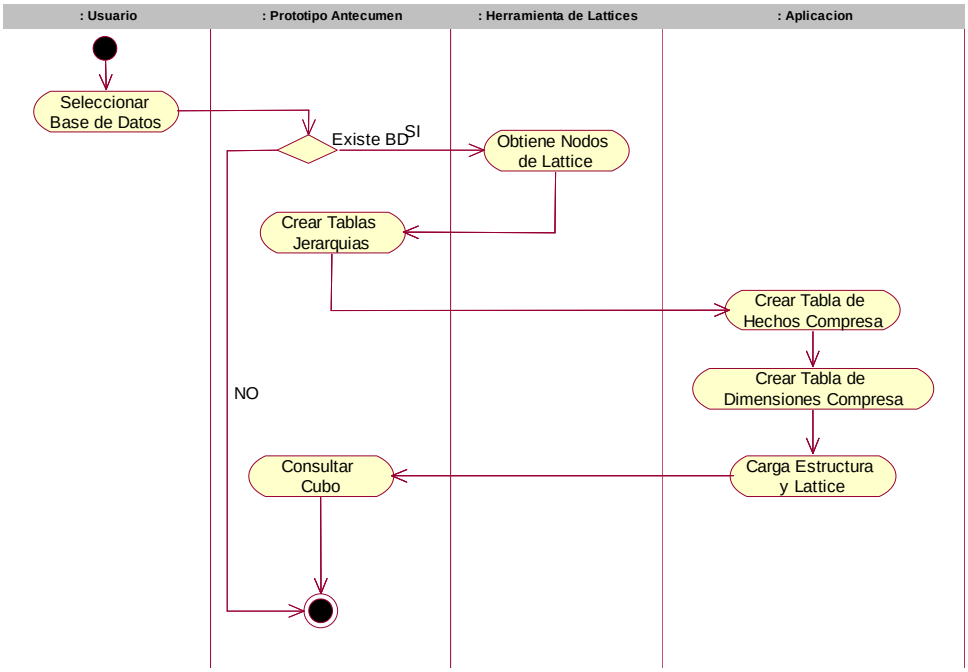


Ilustración 8. Diagrama de Actividades Creación de ESTRUCTURA AC

Ingresar Preguntas

El usuario selecciona el tipo de pregunta que desee para su análisis dentro de la Base de Datos seleccionada y de a acuerdo a esta son los valores que se le solicitan, como ya se mostró anteriormente. (Véase Ilustración 9)

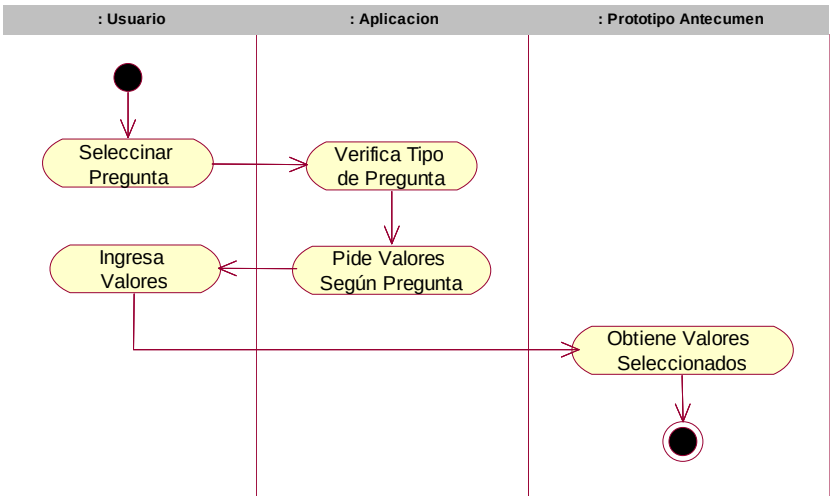


Ilustración 9. Diagramas de Actividades Ingresar Pregunta

Respuestas Visuales

Una vez obtenida la respuesta del análisis realizado por ANTECUMEM, la aplicación obtiene los valores y los muestra a través de texto y gráficas para que el usuario pueda observarlas. (Véase Ilustración 10)

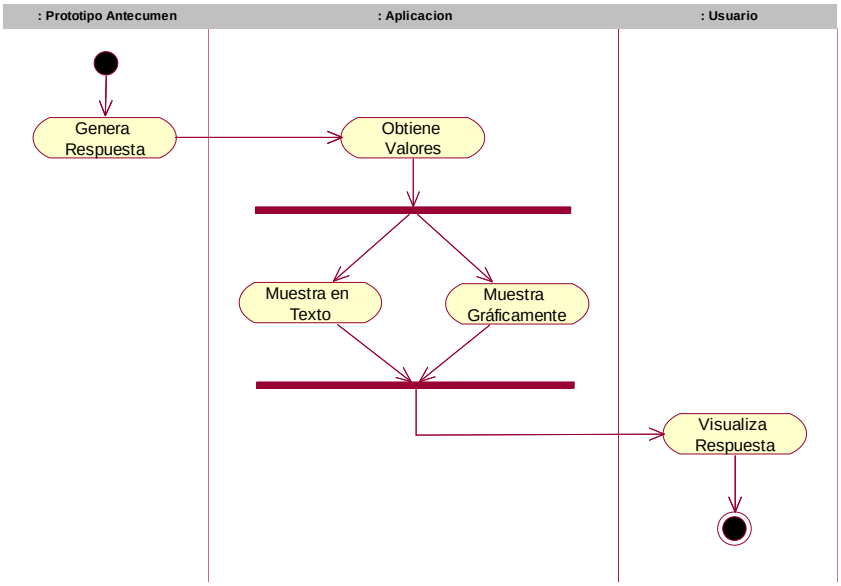


Ilustración 10. Diagrama de Actividades Respuestas visuales

3.2.3 DIAGRAMA DE CLASES

Los diagramas de clases presentados describe la estructura estática del sistema mostrando sus clases, atributos y operaciones. De tal forma que queda definido de manera conceptual la información que se manejará en el sistema, y las clases que se encargaran del funcionamiento. Se muestran dos diagramas de clases expuestos para nuestros dos requerimientos generales: compactación e interfaz.

Diagrama de Clases para la Compactación

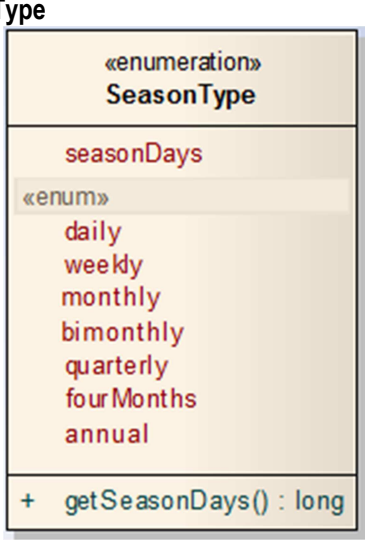
La estructura Arblis es creada a partir de una Base de Datos, no sin antes verificar su existencia para trabajar con ella a través de ANTECUMEM, a partir de esta se genera una *Lattice* con determinado número de dimensiones; uno de los componentes de la Lattice son los *Nodos* de los cuales solo algunos serán representados por *Arblis*.

Arblis es una estructura que contiene apuntadores *Dimensiones* y *Hechos* acomodados de una forma que permite una rápida navegabilidad. Para reducir el espacio en memoria principal que ocupa Arblis se hace uso de *Algoritmos de Compactación* sobre las dimensiones.

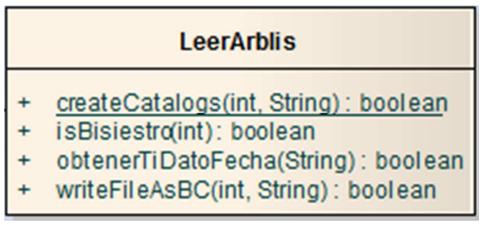
Por otra parte tenemos que el objetivo de ANTECUMEM es responder a diversos tipos de *Pregunta*, y dependiendo del tipo de pregunta es que se realiza la captura de datos, el análisis y la visualización de los resultados. Estas preguntas hacen uso de la estructura AC para obtener la consulta.

Se cuenta con las siguientes clases (véase Tabla 6):

Tabla 6. Clases usadas para la compactación y respuesta de preguntas de negocio

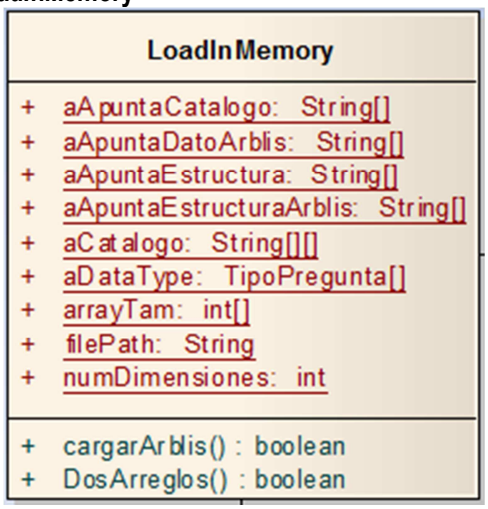
Clases	Descripción
• **TipoPregunta** 	Las enumeraciones son un tipo especial de clases que permiten declarar un grupo de constantes representadas por identificadores, en la clase *TipoPregunta* enumera los siete tipo de preguntas que serán resueltas con la estructura AC para *n* dimensiones.
• **SeasonType** 	Esta clase es una enumeración que muestra los tipos de periodos temporales para comparaciones temporales en rango. Tienen asosciados la cantidad de días que corresponden a cada una de las temporadas. Proporciona el método *getSeasonDays* que permite convertir la representación de la temporada con un valor específico

- **LeerArblis**



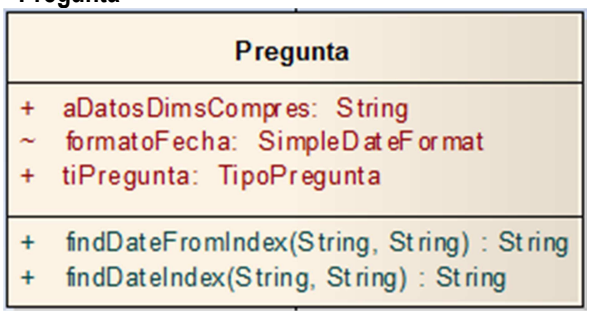
La clase de *LeerArblis* es la encargada de hacer la transformación de la estructura Arblis a AC a través de los métodos de compactación creando catálogos para cada dimensión en el método *createCatalog* que recibe una referencia a la estructura original y el número de dimensiones. Para la generación de AC se hace uso de los catálogos para obtenerla.

- **LoadInMemory**



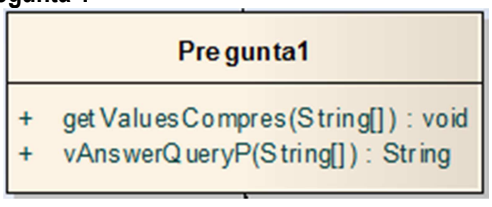
Los catálogos son cargados a memoria en el método *cargarArblis* y en el método de *dosArreglos* se llenana los arreglos correspondientes para la estructura AC los cuales son *aApuntaDatosArblis* y *aApuntaEstructura*. Mientras que el catálogo por dimensión se guardan en una matriz de longitud variables llamada *aCatalogo*.

- **Pregunta**



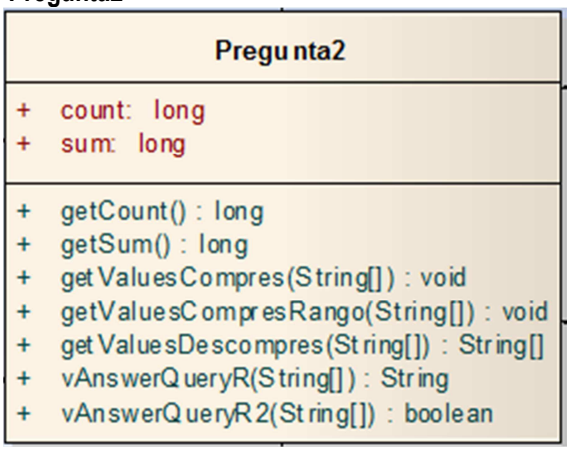
Pregunta es una clase abstracta que tiene los parametros base para realizar una consulta guardados en la variable: *aDatosDimsCompres*. El formato que se maneja de fecha es único para todas las preguntas y todas ellas cuentan con un tipo de pregunta (*tiPregunta*). Se tienen métodos implementados para obtener la compactación de una fecha, así como su descompactación.

- **Pregunta 1**



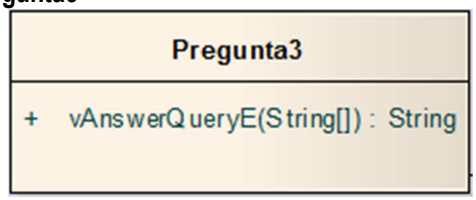
La clase *Pregunta1* es la base para el resto de las preguntas. Es una pregunta puntual en la que se especifican todos los parámetros exactos de la consulta sin dar ningun rango. Debido a que se carga en memoria una sola vez el cubo de datos, la clase *Pregunta1* hace uso de las variables de la clase *LoadInMemory* y realiza sus operaciones en *vAnswerQueryP* comprimiendo los valores recibidos. Sin embargo para este tipo de pregunta no es necesario descomprimir los valores ya que se regresa un único valor para los datos introducidos por el usuario.

- **Pregunta2**



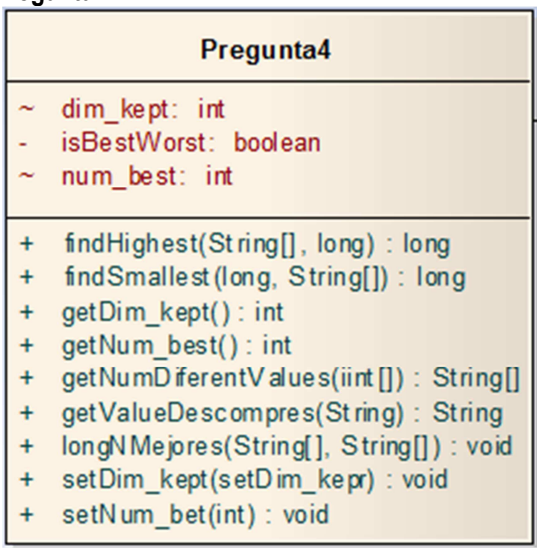
La clase *Pregunta2* hace uso de la clase *Pregunta1* para resolver preguntas por rango dados *2n* parámetros para *n* número de dimensiones. Tiene dos acumulados: *count* y *sum*, el primero se tiene el número de elementos que son sumados en el rango dado y en *sum* se tiene la suma del agregado de todos ellos.
Tiene métodos para compresión y descompresión de los valores, y para contestar tiene el métodos *vAnswerQueryR* y *vAnswerQueryR2*.

- **Pregunta3**



La clase *Pregunta3* contesta preguntas de eficiencia comparando dos cubos de datos dados por rangos. De tal forma que hace uso de la clase *Pregunta2* y posteriormente calcula la eficiencia de éstos.

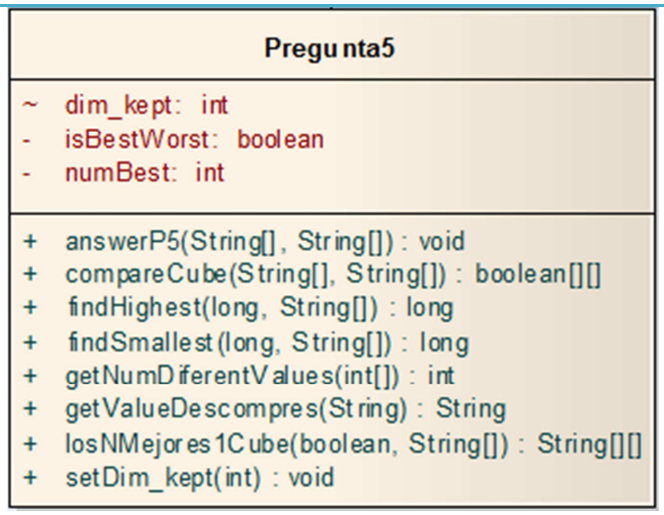
- **Pregunta4**



La clase Pregunta4 contesta preguntas de eficiencia grupal recibiendo dos rangos (que definene los 2 cubos de datos), en el primero se obtienen un número *num_best* de mejores o peores (*isBestWorst*), dichos valores son buscados en el segundo cubo de datos para calcular la eficiencia de éstos, permitiendo la comparación de ellos en dos cubos diferentes a través de los métodos: *getNumDiferentValues*, *longNMejores*, *vAnswerQueryP* y *vAnswerQueryQ*, estos dos últimos provenientes de las clases *Pregunta1* y *Pregunta2*. Sin embargo una de las dimensiones debe ser contante para poderse comparar, la cuál esta definida por *dim_kept*.

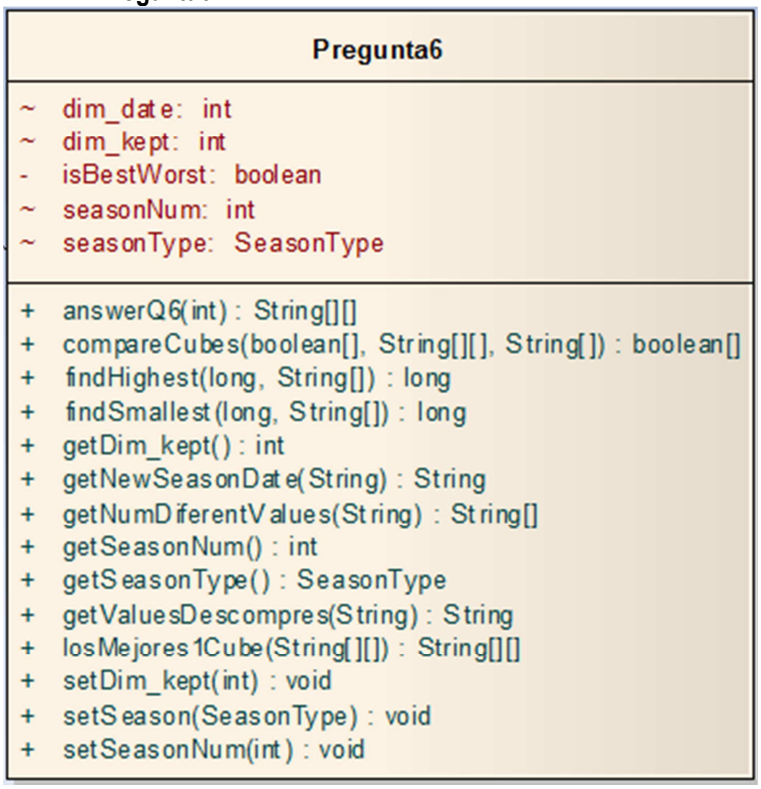
- **Pregunta5**

La clase Pregunta5 contesta preguntas de conservación, de tal fomra que obtiene dos cubos de datos y compara que elementos de la dimensión que se conserva (*dim_kept*) se conservaron en los primeros *numBest* lugares, por lo que es necesario compactar antes de cada consulta con *vAnswerQueryQ,* para comparar los cubos de datos con los métodos *losNMejores1Cube* y *compareCube*, para posteriormente obtener los valores que se

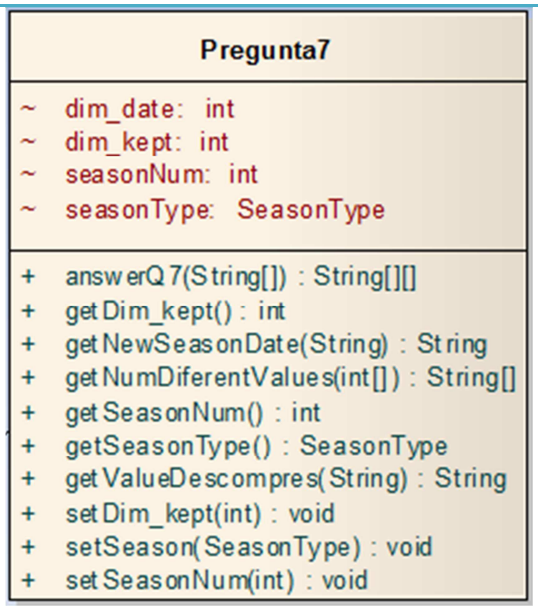


conservaron y descomprimirlos para mostrar al usuario los valores sin compresión.

- **Pregunta6**



La clase Pregunta6 responde preguntas de temporada comparando la cantidad de *seasonNum* cubos cuyos rangos van variando según la temporada *seasonType* dada para saber que elementos de la dimensión que se conserva (*dim_kept*) se conservaron en los primeros *numBest* lugares como los mejores o peores *isBestWorst*, por lo que es necesario compactar antes de cada consulta con *vAnswerQueryQ,* para comparar los cubos de datos con los métodos *losNMejores1Cube* y *compareCubes*, para posteriormente obtener los valores que se conservaron y descomprimirlos para mostrar al usuario los valores sin compresión.



La clase Pregunta7 responde preguntas de tendencia comparando la cantidad de *seasonNum* cubos cuyos rangos van variando según la temporada *seasonType* dada para saber que elementos de la dimensión que se conserva (*dim_kept*) tuvieron un comportamiento creciente, por lo que es necesario compactar antes de cada consulta con *vAnswerQueryQ,* para comparar los cubos de datos con el método *answerQ7*, para posteriormente obtener los valores que se conservaron y descomprimirlos para mostrar al usuario los valores sin compresión.

La relación entre las clases de puede observar en Ilustración 11. Diagrama de Clase Compactación para la compactación y carga en memoria de la estructura AC, así como las clases empleadas para la resolución de preguntas con la estructura cargada en memoria para ***n* dimensiones**.

Para la respuesta de preguntas en las que se comparan varios cubos de datos se le da al usuario la opción de seleccionar la dimensión que se desea conservar, ya que la comparación de cubos de datos debe de tener una dimensión que no cambie y anteriormente era la primera dimensión la que se conservaba; el prototipo de esta herramienta realiza los cálculos según lo requiera el usuario de tal forma que podrá modificar la dimensión que se conserva.

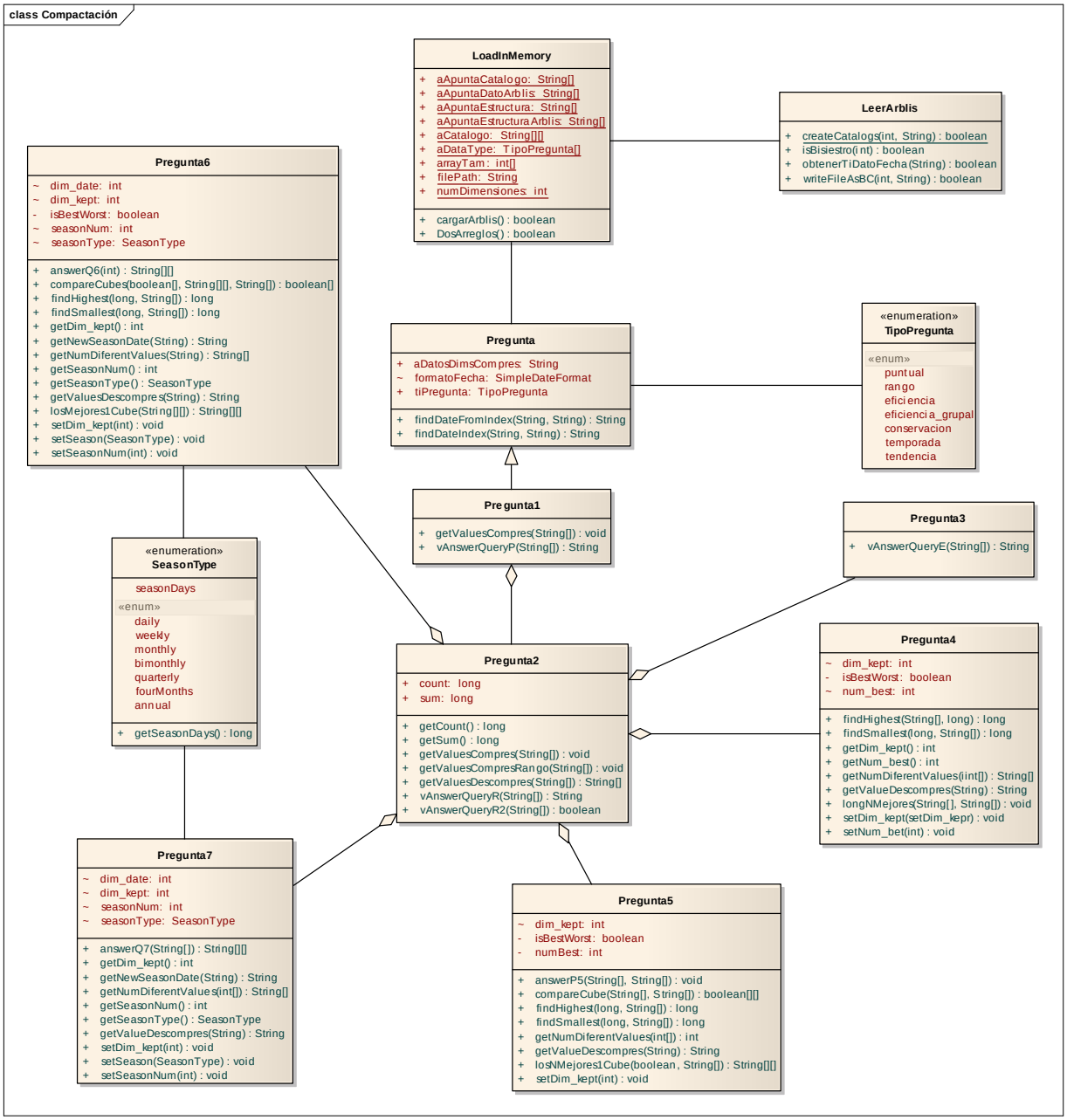


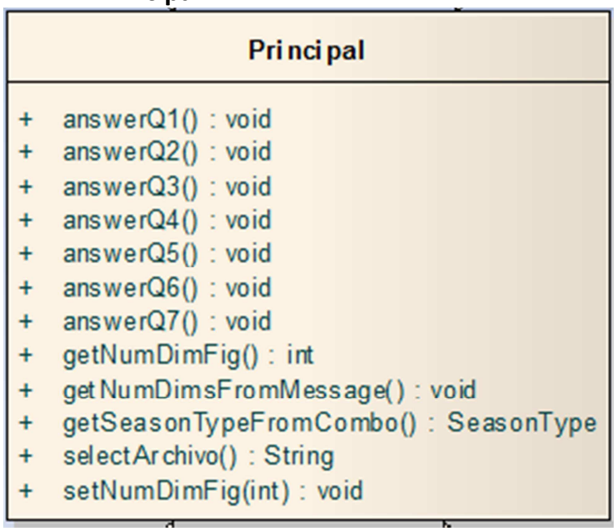
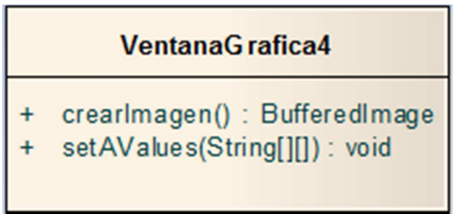
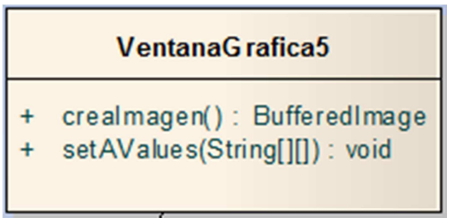
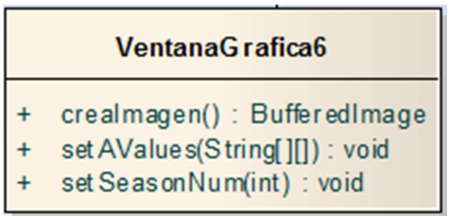
Ilustración 11. Diagrama de Clase Compactación

Diagrama de Clases para la Interfaz

Tenemos dos tipos de pantalla la de *Inicio* donde se selecciona la Base de Datos a utilizar y la de *Selección de Preguntas* que depende del tipo de pregunta que sea solicitada.

Cabe destacar que los resultados mostrados en texto o gráficamente varían según la pregunta de negocio que se haya respondido. Se cuenta con las siguientes clases (véase Tabla 7):

Tabla 7. Clases empleadas para la Interfaz Gráfica

Clase	Descripción
• **Principal** 	Esta es la clase *Principal* de la interfaz gráfica (mostrada en Ilustración 13) en la que además de los componentes gráficos necesarios para la creación se esta se tiene los métodos de responder preguntas: *answerQ1, answerQ2, answerQ3, answerQ4, answerQ5, answerQ6* y *answerQ7* que no reciben parámetros ya que los obtienen de los datos capturados por el usuario en las cajas de texto correspondientes a cada una de las preguntas. Dependiendo del número de dimensiones que sean utilizadas la interfaz **es generada** es por ello que se necesita el método *setNumDimFig* y *getNumDimFig*. La respuesta de las preguntas es mostrada en texto para las primeras 3 preguntas y a partir de la cuarta se generan gráficas haciendo uso de las clases *VentanaGrafica4, VentanaGrafica5, VentanaGrafica6* y *VentanaGrafica7*.
• **VentanaGrafica4** 	La clase *VentanaGráfica4* realiza la gráfica de la respuesta de la clase *Pregunta4* que es usada en la clase *Principal*. Da los parámetros requeridos para las *n* dimensiones y crea la gráfica en *crearImagen*.
• **VentanaGrafica5** 	La clase *VentanaGráfica5* realiza la gráfica de la respuesta de la clase *Pregunta5* que es usada en la clase *Principal*. Da los parámetros requeridos para las *n* dimensiones y crea la gráfica en *crearImagen*.
• **VentanaGrafica6** 	La clase *VentanaGráfica6* realiza la gráfica de la respuesta de la clase *Pregunta6* que es usada en la clase *Principal*. Da los parámetros requeridos para las *n* dimensiones y crea la gráfica en *crearImagen*.
• **VentanaGrafica7**	La clase *VentanaGráfica7* realiza la gráfica de la respuesta de la clase *Pregunta7* que es usada en la clase *Principal*. Da los parámetros requeridos para las *n* dimensiones y crea la gráfica en *crearImagen*.

La relación entre las clases de puede observar en Ilustración 13. Interfaz Gráfica Inicial

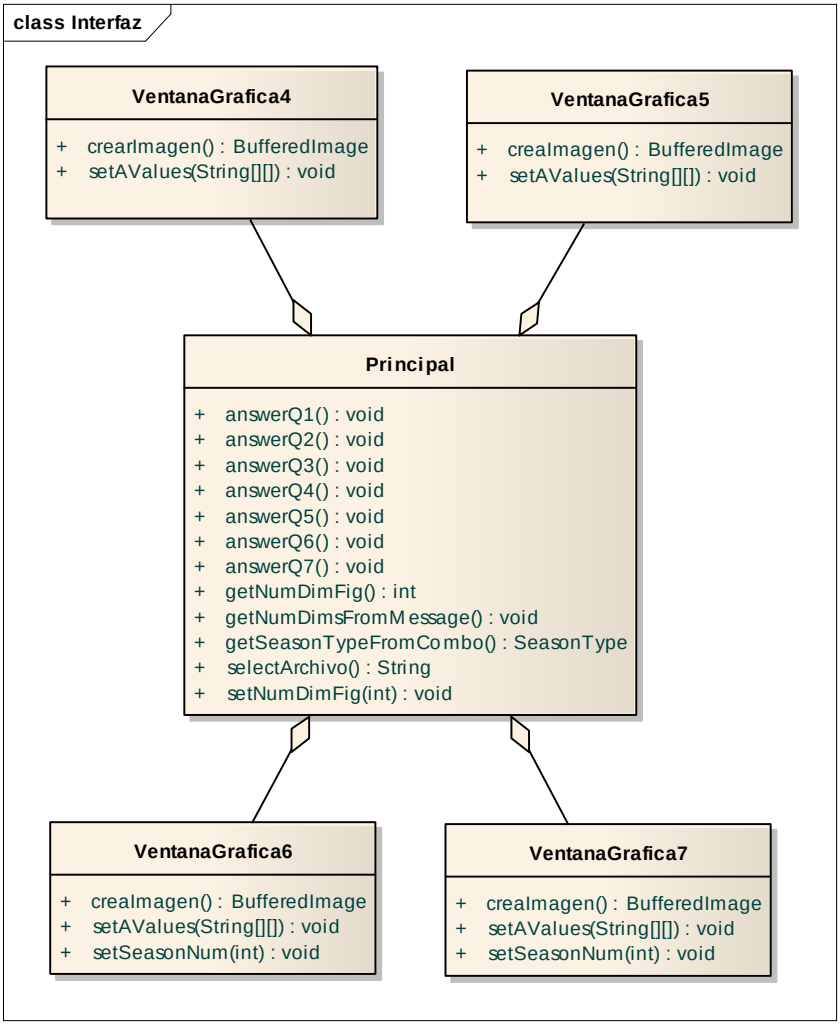


Ilustración 12. Diagrama de Clases Interfaz

La interfaz gráfica inicial generada por estas clases se muestra en la Ilustración 13. Interfaz Gráfica Inicial la cual será explicada en **¡Error! No se encuentra el origen de la referencia.**.

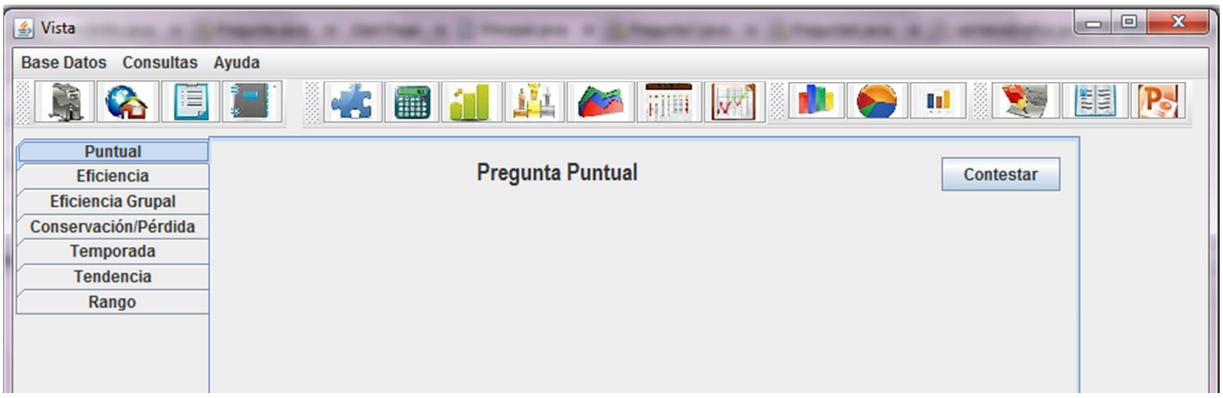


Ilustración 13. Interfaz Gráfica Inicial

3.2.4 DIAGRAMAS DE SECUENCIA

A continuación se muestran los diagramas de secuencia que permiten observar la interacción entre las clases y actores principales, dados para sub-casos de uso para tener una mayor visualización del sistema

Para Diagramas de CdU Compactación de Valores

Ingresar Consulta

Partie del Prototipo de Software ANTECUMEM que se encargará que de ingresar determinados valores para las preguntas que el software mostradas en la *Tabla 3. Parámetros para 7 preguntas* y se regresa una confirmación de lo solicitado. (véase Ilustración 14)

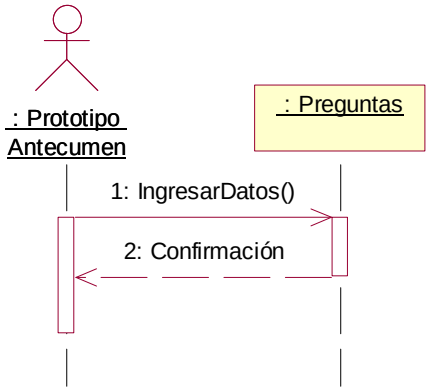


Ilustración 14. Diagrama de Secuencia Ingresar Consulta

Algoritmo de Compactación

Para realizar la compactación y compactación de los datos inicialmente se debe de seleccionar el tipo de compactación que se utilizará para lo cual es necesario hacer antes un análisis de frecuencia de los datos, así como el tipo de datos que se está almacenando en cada uno de los arreglos de la estructura *Arblis inicial* para posteriormente llevar a cabo el método seleccionado generando los siguientes dos arreglos que se muestran en la **¡Error! No se encuentra el origen de la referencia.**. La parte de la compactación y compactación de los datos es la más importante en el desarrollo de esta investigación por lo que posteriormente se detallaran estas funciones.

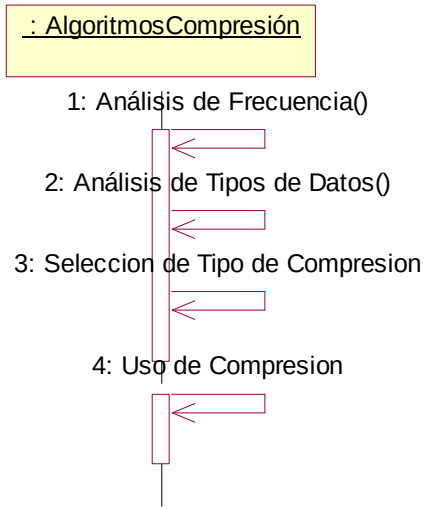


Ilustración 15. Diagrama de Secuencia Algoritmo de Compactación

Compactación

Para realizar la compactación por sustitución de datos habituales combinada con texto idiomático, eliminación de datos redundantes y notación compacta (mencionados en el marco teórico) se necesita saber las dimensiones a usar para crear la estructura Arblis ya definida pero de manera compacta. Por otra parte el algoritmo de compactación devuelve la compactación sobre las dimensiones y las comprime.

De la misma forma se reciben los valores de los hechos que se encuentran en el cubo de datos producto de la combinación de las dimensiones antes recibidas y el algoritmo de compactación regresa nuevamente el tipo de algoritmo de compactación en hechos.

Los dos mencionados anteriormente ingresan los datos para análisis a las preguntas donde se llevará a cabo por otra parte la consulta realizada por el usuario.

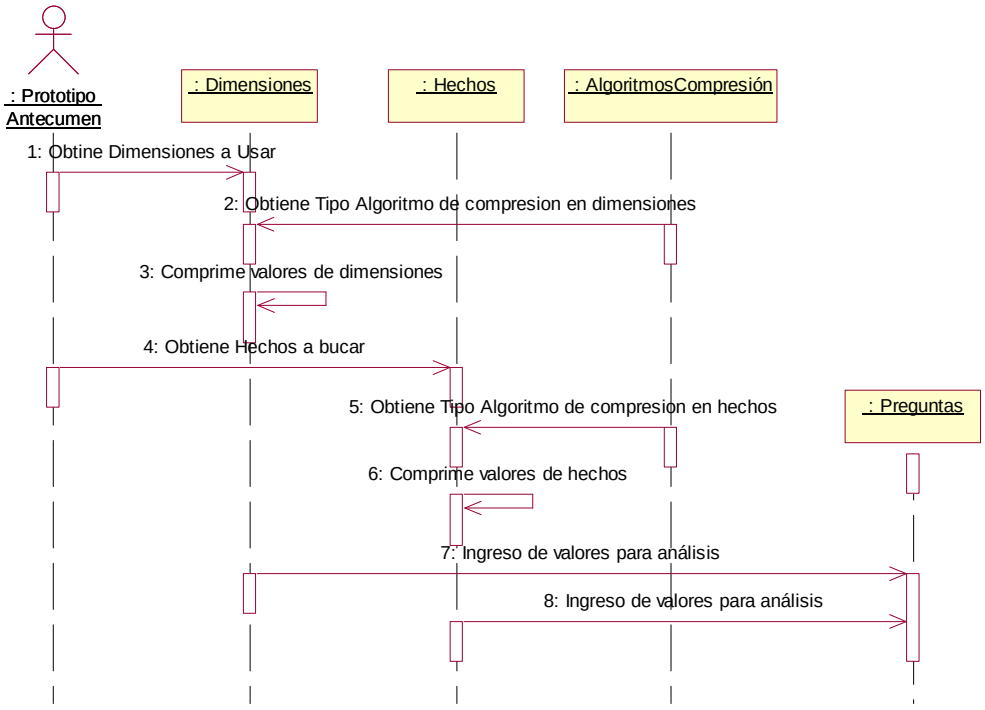


Ilustración 16. Diagrama de Secuencia Compactación

Descompactación

La descompactación se lleva a cabo de forma inversa a la anterior: Preguntas regresará los resultados obtenidos de la consulta de forma compresa, obteniendo el algoritmo empleado en los hechos descomprime los valores obteniendo los resultados de manera legible como inicialmente se solicitaron.

Cabe destacar que en este caso no es necesario llevar a cabo la descompactación de las dimensiones porque el usuario solo verá el resultado presente en los hechos.

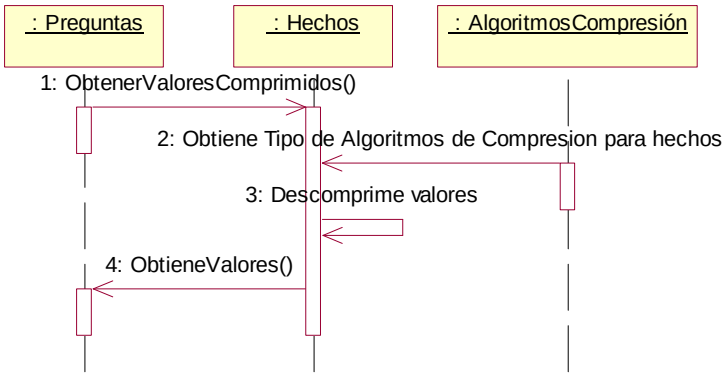


Ilustración 17. Diagrama de Secuencia Descompactación

Búsqueda en Arblis Comprimida

Preguntas pasa los valores compresos requeridos para las preguntas según la tabla hacia la estructura Arblis en la cual se realiza la navegación correspondiente en este nodo de lattice creado con anterioridad para obtener el resultado de la consulta de forma compresa.

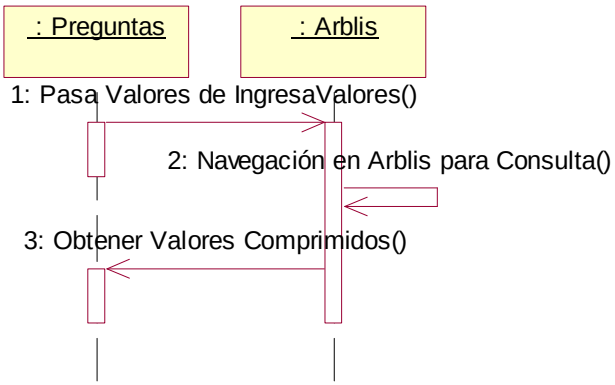


Ilustración 18. Diagrama de Secuencia búsqueda en Arblis

Para Diagramas de CdU Creación de Arblis

Seleccionar BD

Para seleccionar la base de datos se despliega la solicitud al usuario, quien elegirá una BD la cual será buscada para verificar su existencia y poder crear a partir de esta las estructuras Arblis correspondientes a los nodos de lattice sugeridos dentro del prototipo ANTECUMEM

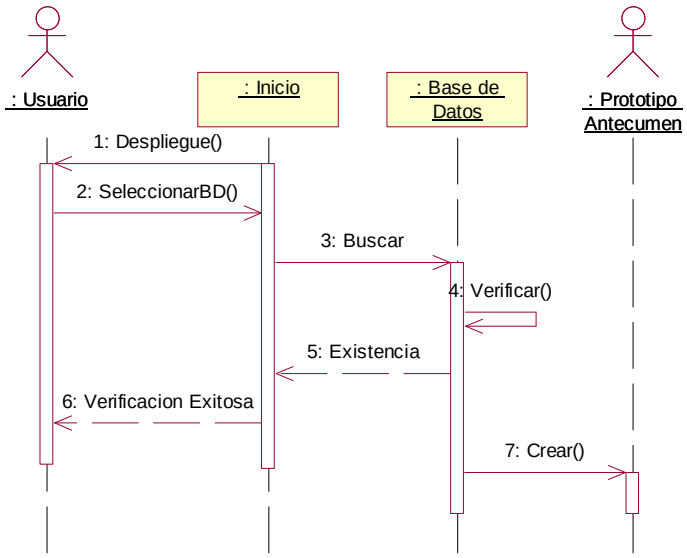


Ilustración 19. Diagrama de Secuencia Seleccionar BD

Creación de Arblis

La creación de Arblis con los datos compresos es de suma importancia. Dependiendo del número de dimensiones se tiene una lattice que es ingresada a la Herramienta de Lattices la cual regresa los nodos óptimos a utilizar para evitar hacer uso de toda la lattice.

A partir de la base de datos se obtienen los valores de las dimensiones y de los hechos que serán compresos tras un análisis de frecuencia y de tipos de datos para aplicar el mejor algoritmo de compactación para esos hechos y dimensiones. En este último se tiene la prioridad de que se realice a través del algoritmo de sustitución de datos habitual.

Una vez realizada la compactación correspondiente se han creado los arreglos de Arblis y teniendo una estructura Arblis por cada nodo de lattice seleccionado se realiza la carga de estos datos a memoria principal.

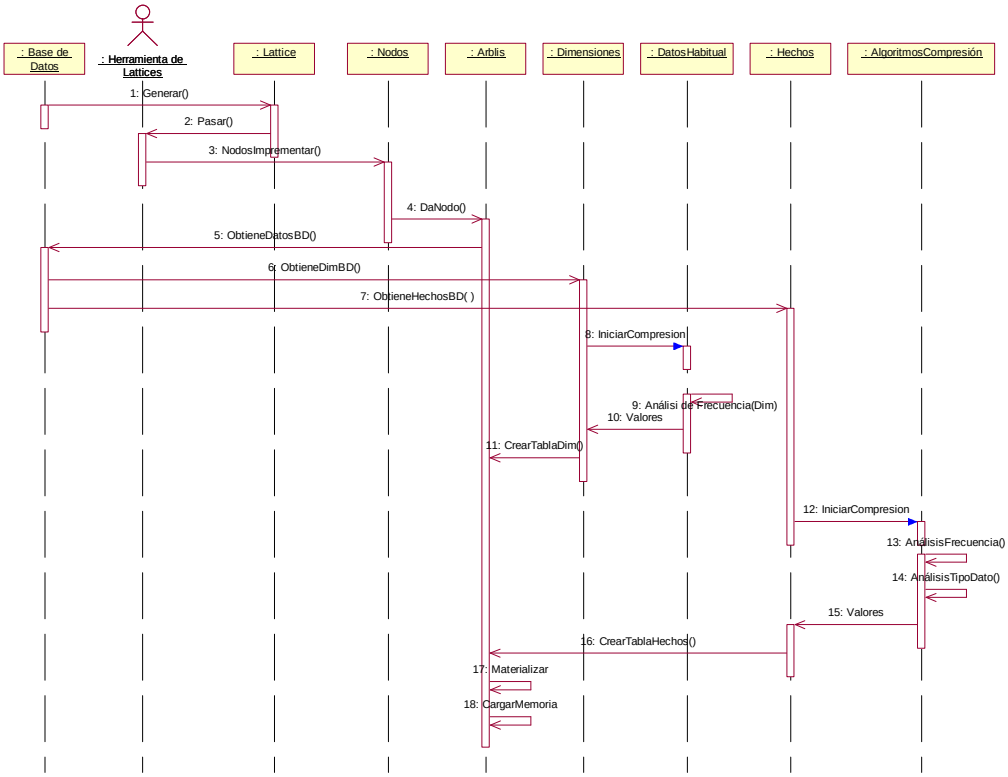


Ilustración 20. Diagrama de Secuencia Creación de Arblis

Para Diagrama de CdU Ingresar Preguntas

Seleccionar Pregunta

Se selecciona el tipo de pregunta que se desea contestar y se le muestra la pantalla correspondiente para el posterior ingreso de valores

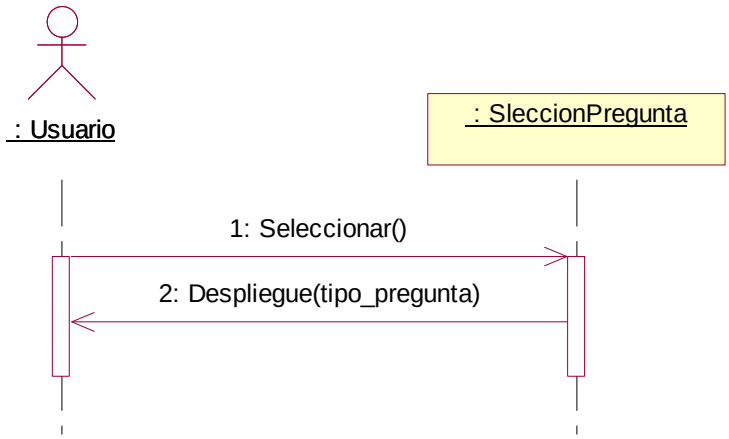


Ilustración 21. Diagrama de Secuencia Seleccionar Pregunta

Ingresar Valores Pregunta

La pantalla que se muestra según el tipo de pregunta permite al usuario ingresar los valores necesarios para responderla según la *Tabla 3. Parámetros para 7 preguntas.*

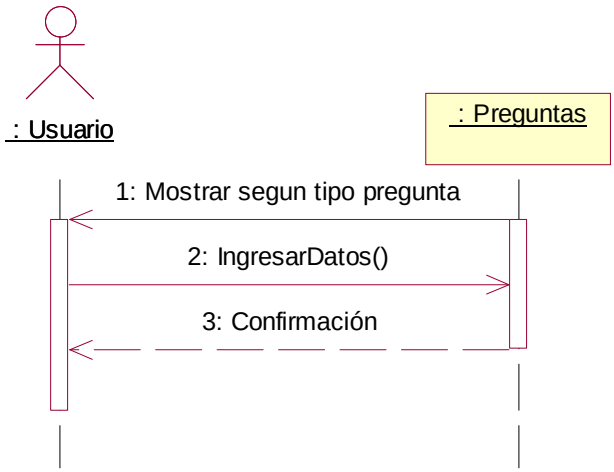


Ilustración 22. Diagrama de Secuencia ingresar valores pregunta

Para Diagrama de CdU respuestas visuales

Mostrar gráficamente

Se recibe el tipo de pregunta que se ha contestado con los valores correspondientes, dependiendo de esto se mostraran valores gráficos a través de diversos tipos de graficas o a través de texto.

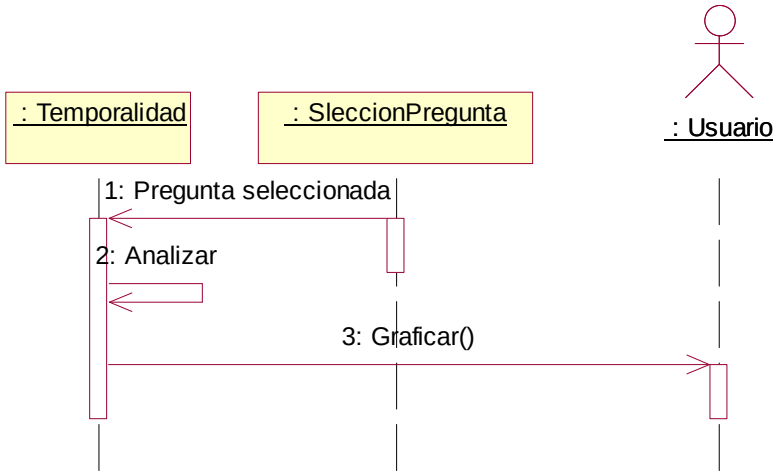


Ilustración 23. Diagrama de Secuencia Mostrar Gráficamente

3.4 RESUMEN

A lo largo de este capítulo se muestra como se realizan las acciones dentro del sistema definiendo operaciones e iteraciones entre éste y los diversoso actores. Se detallan los dos objetivos principales del sistemas: compactación y gaficación.

En la compactación que se realiza al cargar la estuctura en memoria se hase uso de un tercer y cuarto arreglo al que son dirigidos los valores compatados dentro de la estructura Arblis principal, ya que estos nuevos dos arreglos guardan los valores reales que son comprimidos dentro de los dos primeros arreglos en donde se definen unicamente apuntadores cuyo tamaño será definido más adelante.

Debido a que en las dimensiones se repiten se usará en la compactación de estos el algoritmo de Datos Habituales y en los hechos se analizará el tipo de datos y la frecuencia para determinar el tipo de algotimo a utilizar.

La presentación de los resultados de manera visual se lleva a cabo por medio de texto y graficas dependiendo del tipo de pregunta a analizar dentro de las siete posibles, misma razón por la que los valores ingresados en cada una de estas varían.

El uso de los diagramas a lo largo del capítulo permite una mayor comprensión de la arquitectura planteada para el mejoramiento del prototipo Arblis, ya que se logra visualizar escenarios, acciones y objetos estáticos que serán de utilidad durante la implementación.

CAPÍTULO IV. SELECCIÓN DE MÉTODO DE COMPACTACIÓN

4.1 INTRODUCCIÓN

Existen diversos tipos de compactación ya mencionados (véase2.4 COMPACTACIÓN DE DATOS), los cuales son tomados como base para la selección de la forma de compactación más apropiada para la aplicación.

La selección del método de compactación se sustenta en las pruebas de escritorio mostradas a lo largo del presente capítulo, dentro del cual se muestran cuatro propuestas de compactación realizadas para el desarrollo del presente documento compactando la estructura Arblis, las cuales conservan las propiedades de navegabilidad y orden que posee Arblis.

Los métodos mostrados a continuación fueron ideados a partir de los métodos conocidos y adaptados a la necesidad de navegabilidad que exige el manejo de cubos de datos para una rápida respuesta a determinadas preguntas de negocio. Son descritos con sus respectivas observaciones mostrando las ventajas y desventajas de cada uno de ellos.

Las cuatro propuestas que son evaluadas con cálculos sobre así como los resultados de las pruebas de escritorio realizadas sobre un cubo de datos mapeado a la estructura Arblis de la base de datos muestra Historial de Ventas SH.

4.2 ARBLIS

Ablis es un almacén persistente de datos de solo lectura en la que se llevan a cabo búsquedas típicas de minería de datos. La estructura *Arblis* es vista como una base de datos multi-dimensional dentro del prototipo en el que será implementada de forma compacta. La flexibilidad del modelo permite contestar preguntas de negocio como las que se plantea resolver.

Por el diseño de *Arblis*, la consulta toma un tiempo proporcional (lineal) al tamaño de los datos [Martínez, 2007], toda compactación aumenta la complejidad y afecta en el tiempo, por lo que el método de compactación a seleccionar deberá tener un bajo impacto.

Arblis se forma de arreglos ligados entre si permitiendo buscar en orden y con los ciclos predefinidos dentro de las dimensiones dado un valor en cada dimensión debido a la organización de la que hace uso, veamos el siguiente ejemplo:

Tomando tres dimensiones: Producto, Color y Tamaño con el agregado de Ventas, tendríamos la tabla de la izquierda de la Ilustración 1 en la que se cuenta con dos elementos en la dimensión de *Producto*, dos en la de *Color* y tres en *Tamaño* cuyos totales en las combinaciones de estas tres están representados con *Ventas.* Existen valores en null ya que al formarse el cubo de datos hay valores que no existen por diversos motivos, tal vez so hubo ventas de determinado producto o éste no existe.

La equivalencia con la estructura *Arblis* se muestra del lado derecho, donde la dimensión *Producto* está representada como el *Segmento 1*, *Color* como *Segmento 2* y *Tamaño* como *Segmento 3.* El *Segmento 1* apunta al *Segmento 2*, el *Segmento 2* apunta al *Segmento 3* y el *Segmento 3* apunta a los ***Hechos*** es decir las *Ventas* (sombreados). Para la navegabilidad en la estructura se hace uso de los apuntadores por ejemplo la posición 0 apunta a la posición 2, la posición 2 apunta a la 6 y la posición 6 tiene los hechos.

Posición	*Producto*	*Color*	*Tamaño*	*Ventas*
0	Pantalón	Rojo	Chico	23
1	Pantalón	Rojo	Mediano	45
2	Pantalón	Rojo	Grande	34
3	Pantalón	Verde	Chico	71
4	Pantalón	Verde	Mediano	Null
5	Pantalón	Verde	Grande	46
6	Suéter	Rojo	Chico	75
7	Suéter	Rojo	Mediano	Null
8	Suéter	Rojo	Grande	12
9	Suéter	Verde	Chico	Null
10	Suéter	Verde	Mediano	34
11	Suéter	Verde	Grande	56

		Arreglo 1	Arreglo 2	
Dimensión	Posición	Valor	Apuntador	Segmento
Producto	0	Pantalón	2	1
	1	Suéter	4	
Color	2	Rojo	6	2
	3	Verde	9	
	4	Rojo	11	
	5	Verde	13	
Tamaño	6	Chico	23	3
	7	Mediano	45	
	8	Grande	34	
	9	Chico	71	
	10	Grande	46	
	11	Chico	75	
	12	Grande	12	
	13	Mediano	34	
	14	Grande	56	

Ilustración 1. Cubo de Datos transformado a Arblis

Si buscamos la venta de pantalón, verde, chico buscamos pantalón en el segmento 1 el cuál se encuentra en el posición 0 del Arreglo 1 por lo que vemos el Arreglo 2 en la misma posición y da un **2** este número es la posición en la que iniciaremos la búsqueda del siguiente segmento, mientras que la siguiente posición da un **4** el cual indica el final de la búsqueda del siguiente segmento. El siguiente parámetro (verde) se encuentra entre las posiciones 2 y 4 del Arreglo 1, vemos que la posición 2 no es y desplazamos a al siguiente donde coincide con el parámetro buscado, en esta posición vemos que el Arreglo 2 contiene un **9** que indica el inicio y el **11** que indica el fin. Vemos la posición 9 del Arreglo 1 para el siguiente segmento y vemos que coincide con el tamaño chico buscado por lo que hemos llegado al elemento requerido. Vemos en el Arreglo 2 la posición 9 y éste contiene un 71, donde el 71 representa el total de ventas del pantalón, verde, chico.

Para observar que sucede cuando no se encuentra un valor (sea nulo) veamos el siguiente ejemplo:

Se quiere las ventas de la siguiente combinación de parámetros: suéter, verde, chico. Suéter en el Segmento 1 del Arreglo 1 da en el Arreglo 2 un inicio en la posición 4 y un final en 6. Desplazamos a la posición 4 el Segmento 2 del Arreglo 1, vemos que la posición 4 no contiene verde y desplazamos a la posición 5, ésta si coincide con el valor buscado por lo que para el tercer parámetro el Arreglo 2 da un inicio en la posición 13 y un final en el tamaño total del Arreglo 2. Para el tercer segmento desplazamos a la posición 13, vemos que el valor de esta posición en el Arreglo 1 es mediano y no coincide con chico, desplazamos a la posición 14, vemos que el valor de esta posición en el Arreglo 1 es grande y no coincide con chico. Debido a que no se encontró el valor en el rango dado por el segmento 2, la combinación suéter, verde, chico no existe.

El manejo de arreglos en *Arblis* es tomado como base para seleccionar el método de compresión adecuado, tomando como base los métodos propuestos en 4.3 MÉTODOS DE COMPACTACIÓN.

4.2.1 PROPIEDADES EN LA COMPACTACIÓN

La compactación *C* está dada por un conjunto de las diversas formas de transformaciones de compactación tal que $C = \{C_1, C_2, \dots, C_K\}$ donde k es el número de compactaciones posibles.

4.2.1.1 Monotónica

Sea *CD* un cubo de datos con *d* dimensiones (que pueden ser relaciones) y *h* hechos a medir y agregar, un elemento de *CD* tiene la forma $CD = (D_1, D_2, .. D_d, H_1, H_2, .. H_h)$

La representación mínima para la optimización es de la forma $CD' = (D'_1, D'_2, .. D'_d, H'_1, H'_2, .. H'_h)$ donde cada $D'_i = C_m(D_i)$ y $H'_j = C_n(H_j)$; con $i \in [1, d]$, $m \in [1, k]$, $j \in [1, h]$ y $n \in [1, k]$. Tomando en cuenta que cada C_m depende del dominio de la D_i y cada C_n depende del dominio de la H_i.

Las transformaciones C_m o C_n que se elijan además de que realicen la compactación, es decir, reducir en tamaño la representación de un valor v_i de una D_k o de un v_i de una H_k deben mantener las siguientes propiedades:

Es esencial el conservar el **orden en los valores** de cada dominio de las dimensiones, tomando en cuenta que el procedimiento para responder las consultas o preguntas de negocio que ayuda a responder *Arblis* tienen la forma en un lenguaje de consulta estructurado SQL siguiente:

SELECT	*A*	**DISTINCT**	*S(A)*
FROM	*D*		
WHERE	*R(D)*		
GROUP BY	*A*		
ORDER BY	*A, S(A)*		

Donde:

- *A* es un subconjunto de atributos de las relaciones D_i que se hallan en $D = \{D_1, D_2, \dots, D_K\}$ y que forman los datos que se desean analizar y las D_i son las dimensiones o variables de interés.
- *S(A)* es el agrupamiento que se realiza siendo el agregado de interés sobre los atributos en *A*. Donde *S(A)* es del tipo numérico dado por SUM,MAX, MIN, entre otros.
- *R(D)* es el conjunto de rangos $R_i(D_j)$ sobre cada relación D_i tal que cada $R_i(D_j)$ tiene la forma $[v_{i_1}, v_{i_2}]$, con $v_{i_1} \leq v_{i_2}$, por lo que la compactación aplicada a estos valores debe de conservar el orden tal que $C_m(v_{i_1}) \leq C_m(v_{i_2})$ por lo que . C_m es una función monotónica.

Esta propiedad, de no respetarse afectaría la característica de la estructura *Arblis* de que en ella cuando se construye se almacenan **los ciclos predefinidos de búsqueda** por cada uno de los valores de las dimensiones (al pasar de una dimensión a otra).

4.2.1.2 Mínima afectación a algoritmos de consultas

Otra característica de la compactación utilizada es que la **afectación a los algoritmos** que responden las consultas o preguntas de negocio, la cual **debe ser mínima**. La afectación mínima implica que no exista, pues la transformación de la pregunta a resolver se realiza antes de iniciar la ejecución y se vuelve a realizar a regresar los valores originales de búsqueda cuando el algoritmo regresa las respuestas, por lo tanto las afectaciones a los algoritmos es mínima.

Cabe destacar la afectación sobre los algoritmos se debe al manejo de *n* dimensiones pero no a la compactación empleada.

4.2.1.3 Isomorfismo

El homomorfismo[1] nos permitirá mantener el orden, y si logra ser biyectivo se tendría que dado CD como un cubo de datos y CD' el conjunto resultante de la compactación, CD y CD' son isomorfas, denotadas por $CD \cong CD'$, si existe una biyección $C: CD \to CD'$ donde C es la función de compresión y D la función de descompresión tal que:

- $\forall v \in CD$ existe $C_m(v) = v'$ para cada función *m*-aria C donde $v' \ni CD'$
- $C\big(D(v_1, v_2, \dots, v_n)\big) = D\big(C(v_1), C(v_2), \dots, C(v_n)\big)$ donde $v_1, v_2, \dots, v_n \in CD$
- $(v_1, v_2, \dots, v_m) \in R^{CD}$ si y solo si $\big(C(v_1), C(v_2), \dots, C(v_n)\big) \in R^{CD'}$

[1] Si un objeto *X* es un conjunto con orden menor y otro objeto *Y* consiste en un conjunto con orden menor, entonces debe vales para la función $f: X \to Y$ tal que para $u < v \to f(u) < f(v)$.

De tal forma que CD y CD' son idénticas excepto por sus dominios de tal forma que la lógica de primer orden se conserva en ambas estructuras.

4.2.1.4 Optimización

La optimización no se logra ya que no se aplican todas las combinaciones posibles sobre las dimensiones D y los hechos H dentro del cubo de datos CD, además de las restricciones generadas al conservar las propiedades de Arblis por lo que la función de transformación debe de ser monotónica e isomorfa.

4.3 MÉTODOS DE COMPACTACIÓN

La selección de la forma en que se hará la compactación ha considerado la frecuencia de repetición de los datos almacenados así como los tipos de datos que son guardados.

Tomando en cuenta que uno de los objetivos de ANTECUMEM es la resolución de determinadas preguntas de negocio, en la cuales se han considerado preguntas con rango, por lo que los datos que se manejan deben estar orientados para poder responder este tipo de preguntas.

El cubo de datos de Historial de Ventas **SH** (Véase 2.6.1 BASE DE DATOS SH) sobre el que se realizaron la pruebas cuenta con 1,016,271 número de registros, en cuatro dimensiones: Products, Customers, Promotions y Times; y con el agregado de Sales. Cada dimensión cuenta con un rango de valores mostrado en la Tabla 1, donde se puede observar el mínimo y máximo de los valores. Sin embargo ese valor no refleja todos los que son ocupados. Es decir, para Products se cuenta con valores que van de 5 a 49,980; los productos que son ocupados no es la diferencia de este rango (49,975) si no 5,022 mostrados en la columna siguiente.

Tabla 1. Descripción de valores de SH

Dimensión	Rango de Valores		Valores en hechos	Total en Bytes
	Mínimo	Máximo		
Products	5	49,980	5,022	10
Customers	50	189,450	1,569	12
Promotions	1	9.999	376	8
Times	1998/01/01	29/11/2000	1,030	20
Sales	2.10	14,994.00	No es necesario	8
Total/Máximo			7,997	20

En la última columna se hace referencia al espacio ocupado en memoria, considerándolos como arreglos de caracteres las dimensiones y como double las ventas. Se toma en cuenta que un carácter equivale a 2 byte en memoria y un double a 8 bytes (64 bits).

Los métodos mencionados, creados en el desarrollo de la tesis, fueron ideados para seleccionar un algoritmo de compactación y ver cuál de los métodos descritos a continuación sería una mejor aportación al prototipo ANTECUMEM.

4.3.1 ARREGLOS CON ELEMENTOS VACÍOS

Este método permite almacenar los agregados (Ventas) en una columna, en la cual se tienen referencias dentro de otro arreglo (Arreglo 1) que funge como catálogo, el cuál almacena todos los valores del cubo de datos. Este último almacenamiento evita la repetición de datos y supone que se podrían tener agregados en todas las combinaciones posibles; De tal forma que el tamaño del Arreglo 2 corresponde a la Ecuación 1 donde:

- T es el tamaño del Arreglo2 mostrado en Ilustración 2
- *n* es el número de dimensiones y
- E_n es el número de elementos correspondiente a la dimensión *n*.

Ecuación 1. Número de elementos en Arreglo 2

$$T = \prod_{1}^{n} E_n$$

En el ejemplo de Ilustración 2 se cuenta con 2 elementos para la primera dimensión, 2 para la segunda dimensión y 3 para la tercera por lo que el tamaño del Arreglo 2 es de 12 según la Ecuación 1. Como se puede observar (por ejemplo) no existe una venta para el producto pantalón, en color verde, en tamaño mediano; sin embargo se ha almacenado un espacio para éste dentro del Arreglo 2. De tal forma que el inconveniente se presenta en cubos de datos donde se tenga una gran cantidad de elementos vacíos, lo cual es muy común ya que difícilmente se cuenta con todos los productos en todos los tamaños y todos los colores, independientemente de su venta.

Posición	Arreglo 1
0	Pantalón
1	Suéter
2	Rojo
3	Verde
4	Chico
5	Mediano
6	Grande

Producto	Color	Tamaño		Arreglo 2 Venta
Pantalón	Rojo	Chico	0	23
		Mediano	1	45
		Grande	2	34
	Verde	Chico	3	71
		Mediano	4	
		Grande	5	46
Suéter	Rojo	Chico	6	75
		Mediano	7	
		Grande	8	12
	Verde	Chico	9	
		Mediano	10	34
		Grande	11	56

Ilustración 2. Ejemplo de arreglos con elementos vacíos. El pantalón verde chico tiene una venta de 71, suéter rojo grande fueron vendidos sumando 12, y no hubo ventas de suéter verde chico.

La compresión que se lleva a cabo a través de este método es haciendo uso de catálogos ocupados en el Arreglo1, logrando que los diversos elementos solo aparezcan una vez en lugar de varias veces como es el caso de Arblis y para no perder la navegabilidad se ha decidido reservar un espacio de los elementos vacíos de por lo que se hace uso de la 2.4.1.1 Eliminación de los ítems de datos redundantes y 2.4.1.5 Sustitución datos repetidos de tal forma que este método permite navegar sobre la estructura haciendo uso de catálogos donde la navegación es la siguiente:

Se sabe que la primera dimensión comprende 2 elementos, la segunda 2 y la tercera 3; por lo que la dimensión de color tiene un tamaño de 3 ya que las dimensiones que le siguen solo es una: la dimensión de tamaño que cuenta con 3 elementos, mientras que el producto tiene un tamaño de 6 (3x2 ya que son 3 elementos de la tercera dimensión y 2 elementos en la segunda), por lo que el tamaño nunca cambia (véase Ecuación 2). Si no hay alguna

venta el elemento se queda vacío para conservar los tamaños, de tal forma que a pesar de que no hubo una venta de suéter, rojo, mediano se conserva su espacio.

Nótese que el tamaño de una dimensión está dado por la multiplicación del número de elementos en cada una de las dimensiones que le siguen a la dimensión *k* de un total de *n* dimensiones (véase Ecuación 2)

Ecuación 2.Tamaño de una Dimensión k

$$TamDim_k = \prod_k^{n-1} NumElementos_k$$

Dado un elemento E que se requiere encontrar a partir de los parámetros (p_1, p_2, p_3,.... p_n), la posición del elemento en el Arreglo2 P(E) está dado por la suma de la multiplicación del tamaño de cada dimensión con la posición del parámetro de esa dimensión con respecto a ella misma. En caso de la última dimensión no se toma en cuenta el tamaño de la dimensión ya que no tiene.

Ecuación 3. Posición de un elemento en Arreglo2

$$P(E) = \sum_k^{n-1} \left(TamDim_k \times Pos_{P_k}\right) + Pos_{p_n} - 1$$

Si se quiere encontrar el agregado de suéter, rojo y grande. En los 2 primeros elementos del Arreglo 1 se busca el suéter que resulta ser el segundo teniendo la posición 1. En los siguientes 2 elementos se busca el color rojo que es el primero teniendo la posición 0. Finalmente en los siguiente 3 elementos se busca el mediado que es el segundo teniendo la posición 1. Para cada una de las dimensiones se tiene:

- En la primera dimensión (producto) la posición de suéter es 1 y el tamaño de esa dimensión es de 6.
- En la segunda dimensión (color) la posición de rojo es de 0 y el tamaño de esa dimensión es de 3.
- En la tercera dimensión (tamaño) la posición de grande en Arreglo1 es 2 y siendo ésta la última dimensión no se considera su tamaño.

Aplicando la Ecuación 3 tenemos que:

$$(1 \times 6) + (0 \times 3) + 2 - 1 = 7$$

Por lo que suéter, rojo y mediano se encuentra en la posición 7 del Arreglo 2.

Para calcular el tamaño tomaremos los datos de SH dados en Tabla 1. Considerando que en el tamaño máximo de las dimensiones es de 20, se ha considerado este tamaño para el primer arreglo. Mientras que para el segundo arreglo se ha considerado el tamaño de 8 bytes, ya que es donde se guardará el agregado.

El número de registros por cada arreglo (renglones) para el primer arreglo se tiene la suma de los diferentes valores, es decir la suma de la cantidad de valores en los hechos correspondiente a: 7,997. Para el segundo arreglo el total es obtenido según la Ecuación 1, por lo que equivale a:

$$(5{,}022)(1{,}569)(376)(1{,}030) = 3{,}051{,}579{,}731{,}040$$

Por lo que el espacio ocupado sería el siguiente (véase Tabla 2):

Tabla 2. Ejemplo de SH con espacios vacios

Arreglo	Tamaño	Renglones	Total en Byte
1	20	7997	159940
2	8	3,051,579,731,040	24,412,637,848,320
		TOTAL	24,412,638,008,260

Donde el total en bytes resultante es de 181,888.327 GB que es el espacio que ocuparía en memoria si se implementa arreglos con elementos vacíos.

La complejidad de la navegación se mantiene con respecto a Arblis sin embargo el espacio ocupado es muy alto de tal forma que en el siguiente método se propone una solución a éste ya que se observó un que existen muchos espacios vacíos o nulos en el momento en el que se crea el cubo de datos.

4.3.2 ARREGLOS SIN ELEMENTOS VACÍOS

Esta segunda propuesta resuelve el problema anterior de mantener espacios vacíos, En el momento en que los espacios vacíos no son tomados en cuenta, el tamaño de las dimensiones varía en cada segmento y encontrar a un elementos sería complicado si no se cuenta con las referencias necesarias por lo que se agrega una matriz para mostrar el inicio y fin de cada uno de los valores, en este ejemplo el suéter va desde la posición 5 hasta la 9, mientras que el tamaño mediano se encuentra en las posiciones de 1 a 2 y de 7 a 8.

	Arreglo 1	
	Valores	Matriz
0	Pantalón	{(0,5)}
1	Suéter	{(5,9)}
2	Rojo	{(0,3),(5,7)}
3	Verde	{(3,5),(7,9)}
4	Chico	{(0,1),(3,4),(5,6)}
5	Mediano	{(1,2),(7,8)}
6	Grande	{(2,3),(4,5),(6,7),(8,9)}

Producto	Color	Tamaño
Pantalón	Rojo	Chico
		Mediano
		Grande
	Verde	Chico
		Grande
Suéter	Rojo	Chico
		Grande
	Verde	Mediano
		Grande

	Arreglo 2
	Venta
0	23
1	45
2	34
3	71
4	46
5	75
6	12
7	34
8	56

Ilustración 3. Ejemplo de Arreglos sin elementos vacíos

De tal forma que el tamaño del Arreglo 2 corresponde a Ecuación 4 o en su defecto al total de valores del último nivel de su lattice correspondiente:

- T es el tamaño del Arreglo2 mostrado en Ilustración 2
- *n* es el número de dimensiones y
- V_n es el número de valores en hechos correspondiente a la dimensión *n*.

Ecuación 4. Tamaño del Arreglo2 en el método de arreglos sin elementos vacíos

$$T = \sum_{1}^{n} V_n$$

El tamaño del Arreglo 1 es la suma del número de valores diferentes *Val* en cada una de ellas. Mientras que el número de pares que refieren el máximo y mínimo es el tamaño de la matriz, dado por la cantidad de elementos que

se generan del traslape de dimensiones dado por la lattice de la Ilustración 4 perteneciente a la Ilustración 3 tomando en cuenta el orden que se tiene en la estructura.

Observe que se cuenta con 3 datos en cada nodo (vista) de la lattice: el primero hace referencia a la combinación de Producto (P), Color(C) y Tamaño (T), el segundo a las dimensiones activas (1=activa/presente, 0=inactiva), y el tercero a la cantidad de elementos correspondiente al nodo de cada una de ellas. Considerando que el orden de la estructura se toman en cuenta los valores de la vista 100, 110 y 111 representando a la cantidad de elementos de: producto, producto con colores y producto con color y con tamaño. De tal forma que se tienen 2, 4 y 9 elementos los cuales son sumados para representar la cantidad de pares en la matriz de la ilustración.

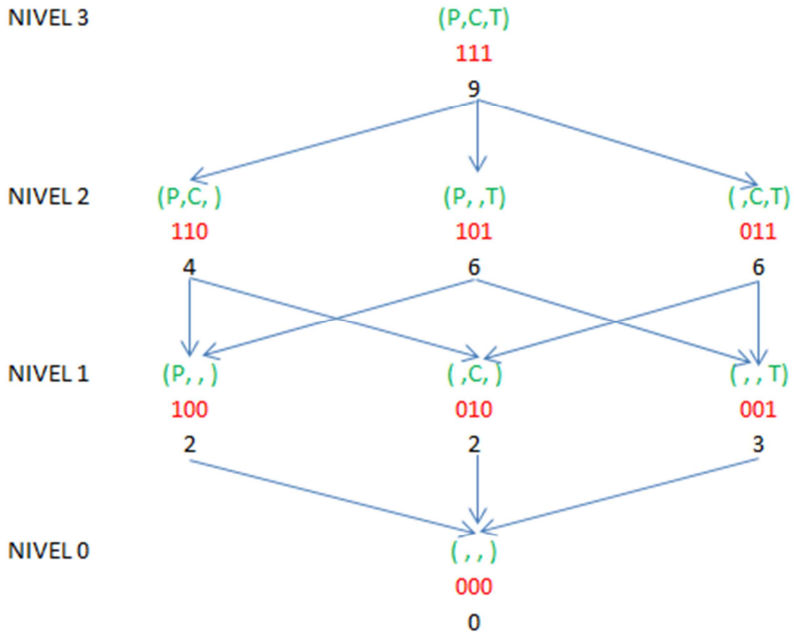


Ilustración 4. Lattice de BD muestra. El número inferior en cada nodo representa el número de elementos en esa vista. Ejemplo: hay 2 elementos de P en el Nivel1, 4 elementos en el nivel2 para la combinación de P y C.

La compresión que se lleva a cabo a través de este método es haciendo uso de catálogos ocupados en el Arreglo1, logrando que los diversos elementos solo aparezcan una vez en lugar de varias veces como es el caso de Arblis y para no perder la navegabilidad se ha decidido reservar un espacio de los elementos vacíos de por lo que se hace uso de la 2.4.1.1 Eliminación de los ítems de datos redundantes y 2.4.1.5 Sustitución datos repetidos de tal forma que este método permite navegar sobre la estructura haciendo uso de catálogos donde la navegación es la siguiente:

La navegación en esta estructura depende de la matriz por lo que si queremos encontrar el de suéter, rojo y grande. Se busca que suéter en el Arreglo1 y se ve en la matriz que va de 5 a 9. Posteriormente se busca el color rojo que dice que va de 0 a 3 y de 5 a 7. Finalmente se busca el tamaño grande que va de 2 a 3, de 4 a 5, de 6 a 7 y de 8 a 9.

Se hace la intersección de todos los anteriores. El resultado de la intersección de suéter rojo es de 5 a 7. La intersección de esto con los rangos para el tamaño grande da de 6 a 7 por lo que el elemento que buscamos se encuentra en la posición 6 del Arreglo 2.

La compactación que se realiza en este método hace uso de: 2.4.1.1 Eliminación de los ítems de datosredundantes, 2.4.1.4 Evitar espacios vacíos y 2.4.1.5 Sustitución datos repetidos.

Haciendo uso de SH se toma en cuenta los valores de la Tabla 1 el tamaño del Arreglo1 es la suma de valores diferentes en cada una de las dimensiones por lo que se tienen 159, 940 bytes resultante de la cantidad de valores en los hechos multiplicado por el tamaño máximo de bytes ocupados por sus elementos lo que corresponde a:

Tabla 3. Cálculo del tamaño de Arreglo1 para método de compresión con elementos vacíos

Dimensión	Valores en hechos	Total en bytes
Products	5,022	100440
Customers	1,569	31380
Promotions	376	7520
Times	1,030	20600
	TOTAL	159940

Para el cálculo de la matriz se hace uso de la lattice de la Ilustración 5, en la que P representa la dimensión de Products, C la de Customers, Pr la de Promotions y T la de Times. Haremos uso por la forma en que se estructura el arreglo 2 del método propuesto las vistas: **1000** con solo P, **1100** de P con C, **1110** de P con C y con Pr y **1111** de P con C, con Pr y T. Por lo que la cantidad de pares ordenados de mínimo y máximo de la matriz corresponde a la suma de las vistas mencionadas lo que equivale a: $5{,}022 + 245{,}504 + 392{,}193 + 1{,}016{,}271 = 1{,}658{,}990$ pares. Tomando en cuenta que corresponden a 2 enteros (máximo y mínimo) los cuales tienen 4 bytes de tamaño el tamaño total para la matriz es de: 13,271,920 bytes.

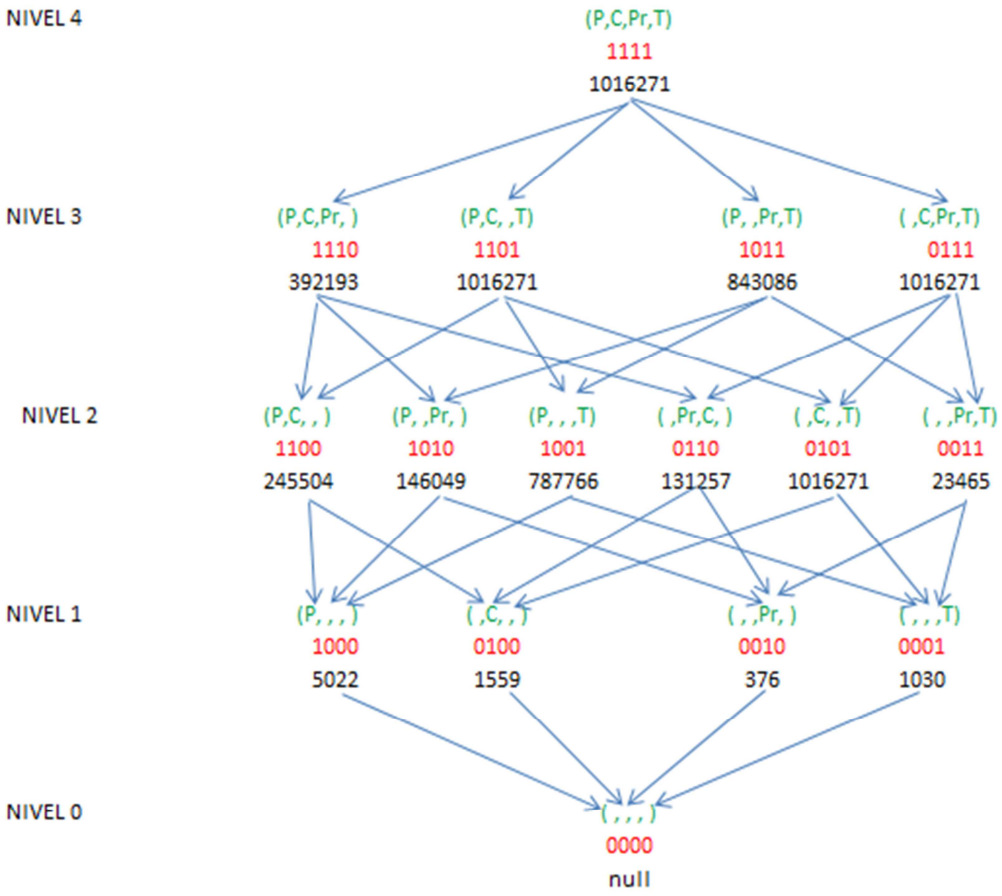


Ilustración 5. Lattice para SH

El tamaño del Arreglo2 equivale a 1016271 renglones con un tamaño de 8 bytes lo que corresponde a 8130168. Sumando el tamaño del Arreglo1, del Arreglo2 y la Matriz se tiene que el tamaño es de 21,562,028 bytes que son aproximadamente 21 MB

Por lo que el total es de 3,371,139,032 bytes, lo que equivales a cerca de 25 GB lo cual está por debajo de lo ocupado por Arreglos con elementos vacíos. Ocupando un 0.0088% de espacio con respecto al método anterior sin embargo la cantidad de operaciones y búsqueda de valores que se hace aumenta con respecto a l anterior.

4.3.3 MAPAS DE BITS

El mapa de bits muestra dos arreglos nuevamente. El primero compuesto por los diferentes valores que pueden obtenerse y el segundo por los hechos existentes. En el intento de disminuir el tamaño de la matriz mostrada en el método anterior se utiliza un mapa de bits que permite saber la existencia o inexistencia del valor de alguna dimensión, cuyas intersecciones darán los hechos requeridos.

En la Ilustración 6 se muestra en verde el Arreglo 1 que contiene los valores de las dimensiones, en rojo se muestra el Arreglo 2 que contendrá los valores de los hechos y en el centro el mapa de bits. (VéaseIlustración 6. Compresión con Mapa de Bits)

	Valores	Hechos								
0	Pantalón	1	1	1	1	1	0	0	0	0
1	Suéter	0	0	0	0	0	1	1	1	1
2	Rojo	1	1	1	0	0	1	1	0	0
3	Verde	0	0	0	1	1	0	0	1	1
4	Chico	1	0	0	1	0	1	0	0	0
5	Mediano	0	1	0	0	0	0	0	1	0
6	Grande	0	0	1	0	1	0	1	0	1
		23	45	34	71	46	75	12	34	56

Arreglo 1 Arreglo 2 Matriz Bits

Ilustración 6. Compresión con Mapa de Bits

La navegación en esta sería haciendo intersecciones según los datos buscados. Si se requiere encontrar el de suéter, rojo y grande, se busca un hecho que tenga activo esos 3 bits y a partir de este obtendríamos el hecho en este caso corresponde a la séptima columna que devuelve el 12. La gran desventaja de este es el tiempo que tomaría contestar preguntas ya que sin importar la pregunta se tendría que leer todo el mapa de bits.

Realizando los cálculos de éste método para la base de datos SH de Oracle se

La Tabla 1 presenta los valores utilizados para el cálculo del mapa de bits. Se sabe que se cuenta con 1,016,271 valores en los hechos lo que permite saber el número de columnas dentro del mapa de bits, mientras que el número de renglones en el mapa de bits corresponde al número de valores de todas las dimensiones por lo que el tamaño esta dado por la multiplicación de éstos:

$$(1{,}016{,}271)(7997) = 8{,}127{,}119{,}187$$

Siendo este el total de mapa de bits equivalente 1,015,889,898 bytes.

Para los Arreglo 1 y 2 el tamaño está dado por lo mostrado en Tabla 1 por lo que el total es de 1,024,180,006 bytes equivalentes a 9767 MB. De tal forma que este método de Mapa de bits corresponde a un 4.1951% del método de manejo *de Arreglos con elementos* vacíos, mientras que el uso de *Arreglos sin elementos* vacíos sigue siendo menor representando un 20.8975% del espacio ocupado por *Mapa de bits*.

4.3.4 APUNTADORES A CATÁLOGO (Estructura AC)

La estructura básica de Arblis descrita en 4.2 ARBLIS está compuesta hasta por 10 dimensiones de las cuales solo pueden estar activas 4 de ellas, conservando el orden requerido para facilitar las búsquedas y operaciones a partir de las preguntas de negocios, cada dimensión posee una cantidad variable de valores a los que corresponde un apuntador que dirige al inicio de la siguiente dimensión como ya se ha descrito.

Retomando el ejemplo de la Ilustración 1 , se cuenta con 2 tipos de producto: pantalon y suéter; con 2 colores: verde y rojo; y 3 tipos de tamaño o talla: chico, mediano y grande. La estructura Arblis se muestra en la Ilustración 7 la cualordena los valores por segmentos que representan las dimensiones y la navegación de ésta permite una complejidad lineal ya que depende del número de dimensiones el número de refrencias que se harían dentro de ésta.

		Arreglo 1	Arreglo 2	
Dimensión	Posición	Valor	Apuntador	Segmento
Producto	0	Pantalón	2	1
	1	Suéter	4	
Color	2	Rojo	6	2
	3	Verde	9	
	4	Rojo	11	
	5	Verde	13	
Tamaño	6	Chico	23	3
	7	Mediano	45	
	8	Grande	34	
	9	Chico	71	
	10	Grande	46	
	11	Chico	75	
	12	Grande	12	
	13	Mediano	34	
	14	Grande	56	

Ilustración 7. Arblis

Haciendo uso de apuntadores a catálogo lo que tendríamos sería la Ilustración 8 en la existe un tercer arreglo que representa todos los valores posibles para cada una de las dimensiones por lo que se requiere se cuente con referencia a este catálogo de tal forma que el Arreglo1 contendrá apuntadores o referencias al Arreglo 3, mientras que el Arreglo2 tiene referencias a lugares del Arreglo 1 como lo hace Arblis originalmente, conservando también el valor de los hechos en este arreglo de tal forma que éste no es modificado con respecto al original.

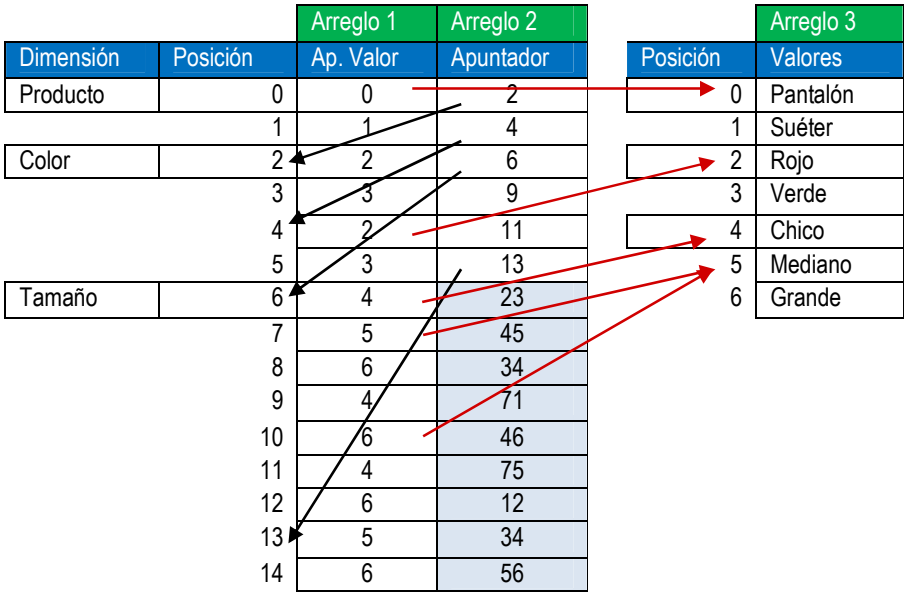


Ilustración 8. Ejemplo de método apuntador a Catálogo 1

Para contestar las preguntas se hace una búsqueda desplazándose a traves de la estructura de la siguiente manera:

Se requiere buscar el la venta de pantalón, verde, chico buscamos el valor de pantalón en el Arreglo3 el cuál se encuentra en la posición 0 por lo que se busca el valor de 0 en el Arreglo1, el cual se encuentra en la posición cero del Arreglo1, leemos en esa posición al Arreglo2 y da un **2** este número es la posición en la que iniciaremos la búsqueda del siguiente segmento, mientras que la siguiente posición (del mismo Arreglo2) da un **4,** el cual indica el final de la búsqueda del siguiente segmento. El siguiente parámetro (verde) corresponde al valor 3 dentro del Arreglo3 que será el valor que se busca entre las posiciones 2 y 4 del Arreglo 1, vemos que la posición 2 no es y desplazamos a al siguiente donde coincide con el parámetro buscado, en esta posición vemos que el Arreglo 2 contiene un **9** que indica el inicio y el **11** que indica el fin. Buscamos el siguiente valor en el Arreglo 3 y se observa que el tamaño chico se encuentra con un valor equivalente a 4 que será buscado entre el 9 y 11. Vemos la posición 9 del Arreglo1 y se ve que coincide con el tamaño chico buscado por lo que hemos llegado al elemento requerido. Vemos en el Arreglo 2 en la posición 9 y éste contiene un 71, donde el 71 representa el total de ventas del pantalón, verde, chico.

Sin embargo los valores en cada una de las dimensiones son diferentes, de tal forma que en algunos casos puede ser cadenas, enteros o fechas los tipos de datos y la compactacción aplicada depende de ésto, de tal forma que el Arreglo3 de la Ilustración 8 será descompuesto en tantas dimensiones como las que se tenga generando una matriz del número de dimensiones por la cantidad de valores en cada una de ellas o con los valores necesarios para su descompactación, obteniendo una matriz como la mostrada en Ilustración 9.

Cabe destacar que la matriz no será de tamaño fijos, de tal forma que este arreglo bidimensional tendra una dimensión con el núero de dimensiones y la segunda será variable dependiendo del número de elementos que se requiera gardar en ellos.

		Matriz		
Dimensión		Valor		
Producto	0	Pantalón	Suéter	
Color	1	Rojo	Verde	
Tamaño	2	Chico	Mediano	Grande
		0	1	2

Ilustración 9. Matriz de Catálogo

Por lo anterior el arreglo1 contendrá el valor empezando de cero de la dimensión a la que pertenezca cada uno de los elementos, reduciendo el tamaño de los elementos dentro del Arreglo1 quedándo como se muestra en el método de apuntadores a catálogos que a partir de este momento llamaremos ***Estrutura AC***. Para contar con una referencia del tipo de compactación realizada a cada una de las dimensiones se emplea un descriptor que será de apoyo para compactar y descompactar datos

		Arreglo 1	Arreglo 2
Dimensión	Posición	Ap. Valor	Apuntador
Producto	0	0	2
	1	1	4
Color	2	0	6
	3	1	9
	4	0	11
	5	1	13
Tamaño	6	0	23
	7	1	45
	8	2	34
	9	0	71
	10	1	46
	11	2	75
	12	0	12
	13	1	34
	14	2	56

Ilustración 10. Estructura AC

En este ejemplo podemos observar las compactaciones 2.4.1.1 Eliminación de los ítems de datos

redundantes, 2.4.1.4 Evitar espacios vacíos y 2.4.1.5 Sustitución datos repetidos al hacer uso de los catálogos . Sin embargo para tipos de dato Date (feha) se aplica el de 2.4.1.2 Conversión a notación compacta. De tal forma que si la base de datos fija con la que se cuenta maneja fechas del 01/01/1998 a 29/11/2000 como es el caso de SH se sustituirá “01/01/1998” por un cero, “02/01/1998” por un uno y así sucesivamente, de tal forma que el catálogo (matriz[numDimFecha]) que se guarda para esta dimensión contiene dos fechas unicamente.

Con la estructura AC se tiene dos arreglos y una matriz, el primero contiene las referencia a la matriz. El segundo es un arreglo de enteros que contiene referencia dentro de la misma estructura, una referencia para cada renglon. La matriz representa los valores de un catálogo de de todos los valores posibles; en SH la cantidad de dichos valores equivale a 7997 los cuales serían multiplicados por 20 si todos fuesen un arreglo pero al ser una matriz. Los bytes usados serán con respecto al de cada una de las dimensiones como se muestra en Tabla 4. Tamaño de matriz para estructura AC.

Tabla 4. Tamaño de matriz para estructura AC

Dimensión en Matriz	Valores en hechos	Bytes	Total en Bytes
Products	5,022	10	50220
Customers	1,569	12	18828
Promotions	376	8	3008
Times	2	20	40
		TOTAL	72096

Para los Arreglos tenemos que el número de renglones o lementos en ellos de la cantidad de elementos con los que se cuenta de tal forma que resulta un tamaño de éstos es de 16,260,336 bytes en la Tabla 5.

Tabla 5. Tamaño de Arreglos para estructura AC

Estructuras (Arreglo/Matriz)	Tamaño	Renglones	Total en Byte
1	8	1016271	8130168
2	8	1016271	8130168
		TOTAL	16260336

De tal forma que el tamaño total está dado por la suma de 92,656 bytes y de 16,260,336 bytes dándonos un total de 16,332,432 bytes lo que equivale a menos de 16MB.

Comparando este tipo de compresión con los métodos empleados con anterioridad tenemos lo que se muestra en Tabla 6.

Tabla 6. Comparación de Métodos

Método	vs Método	Porcentaje de Relación
Con Elementos Vacíos	Sin Elementos Vacíos	0.0088%
Mapa de Bit	Con Elementos Vacíos	4.1951%
Sin Elementos Vacíos	Mapa de Bits	20.8975%
Estructura AC	Sin Elementos Vacíos	76.3123%
Estructura AC	Con Elementos Vacíos	0.0001%
Estructura AC	Mapa de Bits	0.00159474

De acuerdo a los resultados obtenidos la estructura AC será implementada. Ya que ocupa un espacio mínimo con respecto al método de elementos vacíos y el Mapa de bits, y ocupando cerca de un 76% con respecto a lo que ocupa el método sin elementos vacios.

4.4 CONCLUSIÓN

El método de “arreglos con elementos vacíos” hace uso de una gran cantidad de memoria debido a que los elementos en la intersección de las dimensiones no tenían valores, por lo que la mayor cantidad de espacio

reservado corresponde a vacíos: Para evitar este problema se hace uso del método “arreglos sin elementos vacíos” reduciendo consideradamente el uso de memoria, de tal forma que se tuvo un 2% con respecto al método anterior.

El mapa de bits permite trabajar con la unidad mínima de medición, por lo que se esperaba que el espacio ocupado fuese mínimo. Para ello se descartan también espacios vacíos pero se marcan todas las intersecciones existentes, teniendo el mapa de bits con mayor cantidad de ceros que de unos. A pesar de que el resultado en este método es bueno, las operaciones son difíciles de realizar ya que para cualquier tipo de operación se debe de navegar en todo el mapa de bits, lo cual repercute en el tiempo de ejecución.

Debido a las pruebas de escritorio realizadas se decide implementar la estructura AC ya que el espacio ocupado en memoria es menor que el de los otros empleados, además de que la navegación para la solución de preguntas es más rápida.

La estructura AC es una compactación que hace uso de la eliminación de datos redundates, conversación a notación compacta y sustitución de datos repetidos. De tal forma que permite mantener el orden por lo que la función de compactación C_m sobre un valor v es monotónica. Aunado a esto la compresión logra ser uno a uno lo que permite que también ser homomorfica específicamente del tipo isomorfa, con lo cual la búsqueda de datos puede ser realizada sobre la estructura compresa de tal forma que no es necesario descomprimir la estructura para realizar un consulta.

CAPÍTULO V. IMPLEMENTACIÓN

5.1 INTRODUCCIÓN

Para la implementación de este trabajo se ha tomado como base el prototipo ANTECUMEM de tal forma que se comprime la estructura Arblis hacia la estructura AC. Se ha basado en el diseño y análisis mostrado en CAPÍTULO III. ANÁLISIS Y DISEÑO DE LA APLICACIÓN en la plataforma de JAVA, de tal forma que las clases empleadas se encuentran agrupadas en los paquetes como se muestra en la Ilustración 1. Diagrama de Paquetes con Clases.

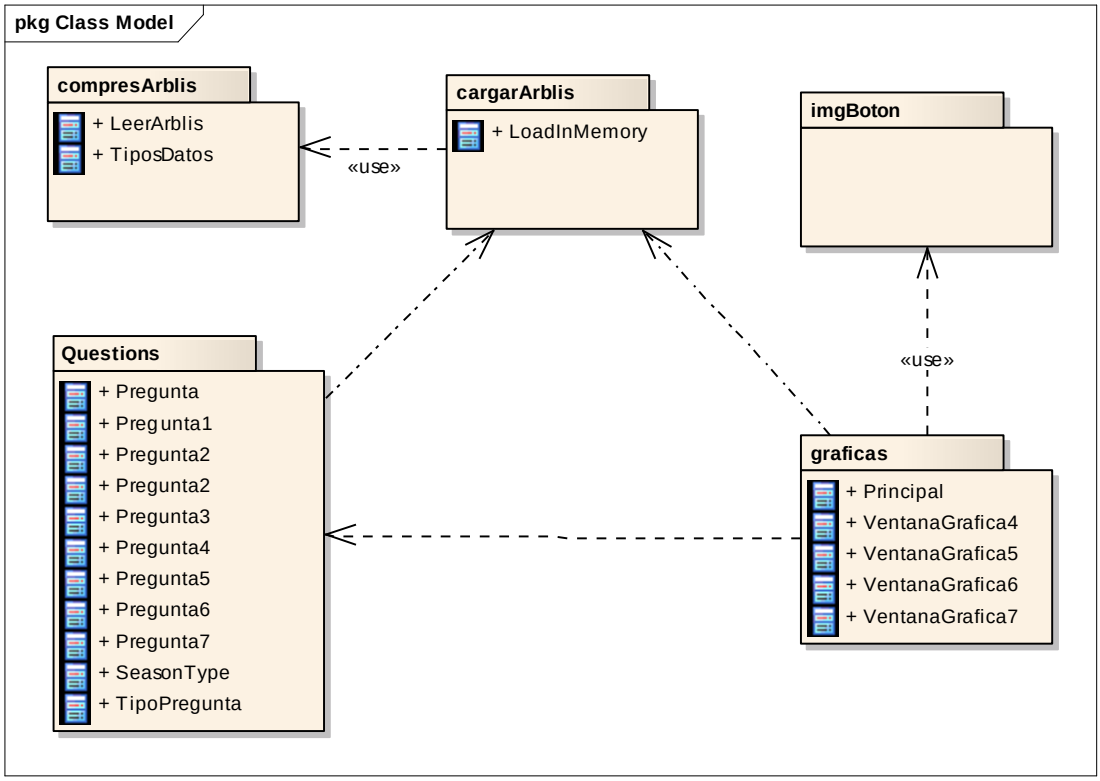


Ilustración 1. Diagrama de Paquetes con Clases

A lo largo del presente capítulo se hará una descripción detallada de las clases empleadas y las relaciones entre ellas, así como de los métodos realizados para la creación de AC, la carga a memoria principal de ésta y la resolución de 7 preguntas de negocio.

La implementación aporta además de la compactación, el uso de *n* dimensiones, tanto en la realización de AC como en la contestación de las preguntas.

5.2 CREACIÓN DE CATÁLOGOS

La creación de los catálogos se lleva a cabo dentro de la clase LeerArblis haciendo uso principalmente de los siguientes algoritmos:

- **Creación de catálogos**
  Este es el algoritmo principal para obtener el catálogo o descripción del tipo de compresión para cada una de las dimensiones. De tal forma que se tiene un desglose del Arreglo 3 de la Ilustración 51 en una arreglo bidimensional de las *n* dimensiones, donde cada una de las dimensiones tendrá el número de elementos

que requiera para su respectivo catálogo como se muestra, en disco duro a cada dimensión le corresponde un archivo donde se guarda el tipo de compresión y el catálogo.

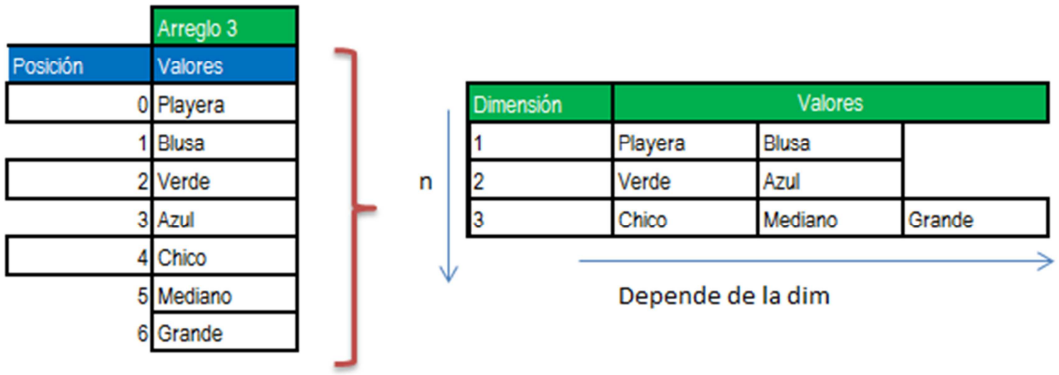


Ilustración 2. Creación de catálogos

Este método devuelve un booleano verificando si se realizó o no correctamente la operación
Requiere como parámetros el nombre del archivo del tipo de dato String de donde procede la estructura Arblis así como el número de dimensiones que puede ser variable (n) dado en enteros.
Las operaciones que se realizan en este son:

1. Se lee la cabecera del archivo para determinar el total de registros en cada una de las dimensiones.
2. Para cada una de las dimensiones
    1 Se lee una muestra para cada dimensión
    2 Se determina el tipo de dato
    3 Se lee todo el archivo dentro de esa dimensión y se genera el catálogo según el tipo de dato y la frecuencia
    4 Se crea el archivo de esa dimensión
3. Fin del ciclo
4. Se cierran todos los archivos
5. Fin de creación de catálogos

- **Determinar el tipo de dato**

Para esta acción se recibe un dato del tipo String

1. Se manda a llamar el método de “obtenerTiDatoFecha”
2. Si es falso
    1. entonces se manda a llamar “obtenerTiDatoEsEntero”
    2. Si es falso
        1. El dato es del tipo cadena
    3. Si es verdadero
        1. El dato es entero
    4. Fin del si
3. Si es verdadero
    1. El dato de del tipo fecha
4. Fin del Si

5. Fin de determinar el tipo de dato

- **Para la generación de los catálogo**

  Se tiene una sobrecarga de métodos de generación de catálogos para los diversos tipo de datos soportados.

  1. Se obtiene el tipo de dato
  2. Si es entero
     1. se manda a crear catálogo pasándole como parámetros una ArrayList
        1. Se crea el archivo
        2. Se escribe la cabecera del archivo indicando el tipo de dato
        3. Mientras existan elementos en la lista
           1. Se escribe el elemento
        4. Fin del mientras
     2. Retorno de la función
  3. Si es fecha
     1. se manda a crear catálogo pasándole como parámetros las fechas máxima y mínima
        1. Se crea el archivo
        2. Se escribe la cabecera del archivo indicando el tipo de dato
        3. Se escribe la fecha máxima
        4. Se escribe la fecha mínima
     2. Retorno de la función
  4. Fin del si
  5. Fin de generación de los catálogos

5.3 CREACIÓN DE LA ESTRUCTURA AC

Para crear la estructura AC se hace uso de los catálogos que se han obtenidos, de tal forma que se cuenta con los siguientes algoritmos para la creación de ésta:

- **Escribir la estructura AC**

  En la estructura AC se escribirán los valores compresos equivalentes a los valores en Arblis. La compresión depende de los catálogos creados con anterioridad.

  Éste método recibe la ruta del archivo original y el número de dimensiones y regresa un booleano para confirmar la realización de la operación.

  1. Abrir el archivo que contiene la estructura Arblis
  2. Escribir cabecera al archivo de AC con los datos del tamaño de cada una de las n dimensiones.
  3. Mientras haya dimensiones
     1 Obtener inicio de la dimensión
     2 Obtener el final de la dimensión
     3 Mientras el contador que parte de inicio no llegue al final
        1. Leer un renglón del archivo
        2. Separar sus tokens en: dato de la dimensión y referencia a arreglo 1 o en su defecto los hechos.
        3. Obtener el valor correspondiente según el catálogo de la presente dimensión.
        4. Escribir en el archivo el valor compreso en lugar del original

4 Fin del mientras
4. Fin del Mientras
5. Fin de escribir la estructura AC

- **Encontrar el valor correspondiente en el catálogo**
  Para obtener el valor correspondiente según el catálogo de cada una de las dimensiones, por lo que recibe la cadena original que debe de ser compreso así como una referencia al catálogo correspondiente.
  1. Según el descriptor del catálogo se obtiene el tipo de compresión empleado
  2. Si es una compresión por catálogo para valores numéricos
     1 Mientras no sea fin del archivo o se encuentre la cadena
        1. Comparar la cadena buscada con la cadena leída
        2. Incrementar el contador
     2 El contador será el valor equivalente
     3 Regresar el valor compreso
  3. Si es una compresión para las fechas
     1 Convertir la fecha buscada al tipo de fecha de Calendario Gregoriano
     2 Convertir la fecha inicial del catálogo al mismo tipo de dato.
     3 Obtener la cantidad de milisegundos para ambas fechas
     4 Sacar la diferencia de milisegundos
     5 Convertir los milisegundos a la cantidad de días correspondiente a la diferencia
     6 Regresar el valor
  4. Si es una compresión por catálogo para valores de cadenas de texto
     1 Mientras no sea fin del archivo o se encuentre la cadena
        1. Comparar la cadena buscada con la cadena leída
        2. Incrementar el contador
     2 El contador será el valor equivalente
     3 Regresar el valor compreso
  5. Fin de encontrar los valores correspondientes según la dimensión

- **Saber si un año es bisiesto**
  Como respaldo a la compresión de fechas es necesario determinar si un año es bisiesto ya que esto aumenta un día a los años bisiestos.
  Para saber si un año es bisiesto:
  1. Se verifica si el año es divisible por 4
     1 Si es verdadero ver si es divisible por 100
        1. Si falso
           - Es bisiesto
        2. Si verdadero
           - No es bisiesto
  2. Si es divisible por 400
     1 Es bisiesto

Para selección la creación de AC y sus respectivos catálogos es necesario seleccionar el botón marcado con el número 1 en la Ilustración 3. Interfaz Gráfica para AC . Esta opción generará la materialización del cubo de datos compreso.

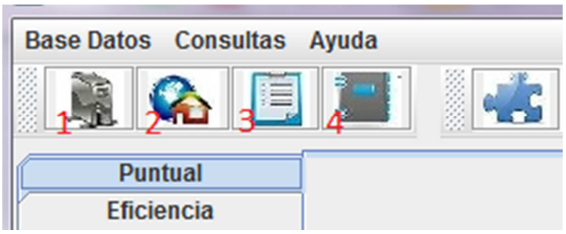


Ilustración 3. Interfaz Gráfica para AC

Para ver los catálogos se selecciona la opción marcada con el número 4 y para visualizar la estructura AC se selecciona el botón con el número tres en la Ilustración 3.

5.4 CARGAR LA ESTRUCTURA AC A MEMORIA

La carga a memoria principal de la estructura AC esta implementada en la clase LoadInMemory, la cual hace uso de las variables estáticas mostradas en laTabla 1. Se ha decidido que las variables sean estáticas para que al resolver las preguntas de negocio, de tal forma que el acceso sea siempre a la misma variable evitando así que se dupliquen en memoria algunas variables.

Tabla 1. Variables para cargar AC a memoria

Tipo de Dato	Nombre	Descripción
static String[]	aApuntaCatalogo	Arreglo 1 donde guardan apuntadores al catalogo
static String[]	aApuntaEstructura	Arreglo 2 donde vienen apuntadores en AC dentro de la estructura
static String[][]	aCatalogo	Matriz de numDim por el num de datos en cada uno (Véase Ilustración 2)
static TiposDatos[]	aDataType	Arreglo que contienen el tipo de dato correspondiente a cada una de las dimensiones
static int[]	arrayTam	Arreglo donde se guardan los tamaños de las dimensiones para facilitar la navegación dentro de ésta
static String	filePath	Ruta de ou
static int	numDimensiones	numero de Dimensiones (entero)

El algoritmo que se lleva acabo para cargar AC a memoria es el siguiente:

1. Se lee el descriptor del archivo AC
2. Se lee el total de registros y el fin de las dimensiones
3. Mientras existan dimensiones desde i = 0
    1. Leer el catálogo correspondiente a la dimensión
    2. Se obtiene tamaño del Catalogo en i
    3. Designar el tipo de dato
    4. Se insertan en el catálogo i los datos correspondientes
    5. Y se carga a memoria
    6. Se cierra el archivo del catálogo i

4. Fin del mientras
5. Se obtiene el tamaño de los dos arreglos principales de AC
6. Mientras j sea menor a la longitud del arreglo 1 o 2 de AC
    1. Separar los tokens
    2. Introducir el primero en los apuntadores al catálogo (Arreglo1)
    3. Introducir el segundo a los apuntadores dentro de la estructura (Arreglo2)
7. Fin del mientras

Para cargar la estructura a memoria para su posterior solución de preguntas se selecciona el botón marcado con el número dos en la figura Ilustración 3. Al mismo tiempo la interfaz gráfica es generada dependiendo del número de dimensiones a ocupar, de tal forma que pedirá al usuario los datos requeridos para cada una de las preguntas adecuada al cubo de datos del que se hará uso.

5.5 MEDICIÓN DE MEMORIA UTILIZADA

En lenguajes como C donde se tiene un acceso directo a memoria, es posible manipular tangiblemente el uso y desuso de la memoria, sin embargo java hace uso de una máquina virtual que reserva automáticamente la memoria según va siendo ocupada.

La medición de memoria ocupada se realiza haciendo uso de la clase Runtime que permite obtener el total de memoria reservada por la máquina virtual. Usando la misma clase obtenemos la memoria libre del total de la reservada de tal forma que obtenemos la memoria ocupada a partir de la diferencia de éstas.

Ecuación 1. Memoria utilizada

$$MemOcupada = MemoriaTotal - MemoriaLibre$$

Dichas mediciones se realizan antes de cargar en memoria los arreglos de AC y después de que éstos han sido cargados; de tal forma que la diferencia refleje la memoria ocupada por AC.

Ecuación 2. Memoria utilizada por AC

$$MemoriaAC = MemOcupadaFinal - MemOcupadaInicial$$

5.5 RESOLUCIÓN DE PREGUNTAS PARA N DIMENSIONES

Se plantea la solución de 7 preguntas de negocio, las cuales abarcan las consultas frecuentes de operaciones en cubos de datos sin el uso de jerarquías. Para responder las preguntas se debe:

Una pregunta de negocio puede ser de cualquiera de los siguientes tipos de preguntas:

1. **Pregunta Puntual.** Localizar el valor de un hecho para valores específicos en cada una de la n dimensiones. Para la solución de ésta es necesario tener un valor en específico para cada una de las dimensiones de tal forma que se debe ingresar una consulta con valores: $Q1(v_1, v_2, \ldots, v_n)$
2. **Pregunta con solo Rangos.** Se tiene un cubo de datos definido por rangos para cada una de las dimensiones, del cual se obtendrá una suma. $S(C)=S(R_1, R_2,\ldots R_n)$
3. **Pregunta de Eficiencia entre dos Cubos.** Calcula un porcentaje de incremento o decremento de dos cubos de datos. $E=100*[S(C_2) /S(C_1) - 1]$.
4. **Pregunta de Eficiencia Grupal** Eficiencia de un conjunto de elementos de un tipo de una dimensión entre dos cubos.

5. **Pregunta sobre Conservación/Pérdida** de Elementos de un Tipo de una Dimensión entre dos Cubos. Localizar objetos de interés que se conservan en una posición determinada dentro de dos cubos de datos.
6. **Pregunta de Temporalidad** Conservación de elementos de un tipo en una dimensión en varios cubos a través de tiempo.
7. **Pregunta de un Tipo de Tendencia de los Elementos de una Dimensión en Varios Cubos.** Localiza elementos que tienen un comportamiento en partículas a través de m momentos continuos de tiempo.

Para la solución de dichas preguntas se tiene una clase padre que tiene el tipo de pregunta y un arreglo de caracteres que contendrá los datos para la solución de las preguntas. A continuación se hace una descripción más detallada de las preguntas y su resolución.

5.5.1 PREGUNTA PUNTUAL

Esta pregunta se define sobre un solo cubo C1. Donde todos los elementos de $R1_i$ que definen al cubo son un único valor para i={1,2,3,…n}, donde d corresponde al número total de dimensiones d_i. De tal forma que C1 es un cubo de datos de un solo punto dado por la combinación de los valores dados.

Para la solución de esta pregunta se parte de que la estructura AC debe de haber ya sido cargada por lo que el algoritmo para responder la pregunta 1 es el siguiente:

1. Mientras existan dimensiones
    1. Obtener el valor compreso a partir del dato dado
2. Fin del mientras
3. Mientras existan datos en el Arreglo 1
    1. Se obtiene el inicio y fin para la primera dimensión
    2. Para i=0 hasta el número total de dimensiones
        1. Para j=inicio hasta que j sea fin o Arreglo 1 es igual al dato compreso buscado
            1. Comparar si el elemento en Arreglo 1 es igual al dato compreso buscado
            2. Si verdadero
                1. Se obtienen inicio y fin para el desplazamiento de la siguiente dimensión
            3. Fin del si
        2. Fin del ciclo
    3. Fin del ciclo
4. Si se llegó al fin y no se encontró regresa vacío
5. Regresa el agregado

Para un mejor detalle observar la Ilustración 4 en donde se puede observar la solución de esta pregunta para n dimensiones.

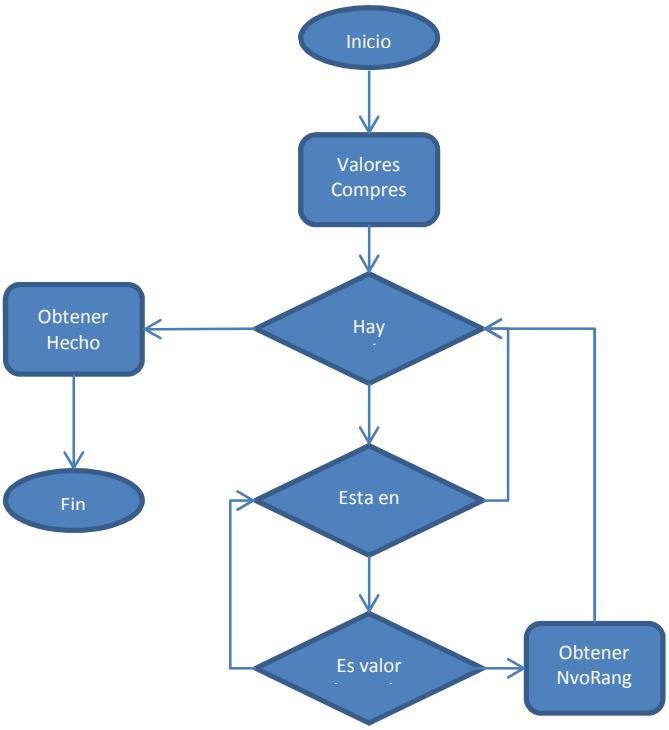


Ilustración 4. Diagrama de Flujo de Pregunta Puntual

Un ejemplo de este tipo de pregunta es:

Cuál es la cantidad vendida del:

Producto: Pantalón (id = 1371)

Color: Rojo (id = 255)

Tamaño: Grande (id = 10)

Y la respuesta es el hecho correspondiente a éste que en este caso según el ejemplo de la Ilustración 44 es de 34.

El cubo de datos generado por el ejemplo de la Ilustración 44 es de 15 registros en la estructura AC con sus respectivos catálogos, la cual cuenta con 3 dimensiones y un agregado de ventas. Por ello en Ilustración 5 se muestra para la pregunta puntal tres espacios a ser llenados para cada una de las dimensiones dándonos como resultado el 34.

En la Ilustración 59 se muestran cuatro espacios para ser llenados ya que para este ejemplo se utilizó un cubo de datos de cuatro dimensiones resultado de agregar una cuarta dimensión de tiempo a la de la Ilustración 44 para poder responder el resto de las preguntas.

Los combos son llenados con todos los valores posibles mientras éstos sean como máximo 100 ya que de otra forma sería tedioso para el usuario buscar entre tantos valores posibles.

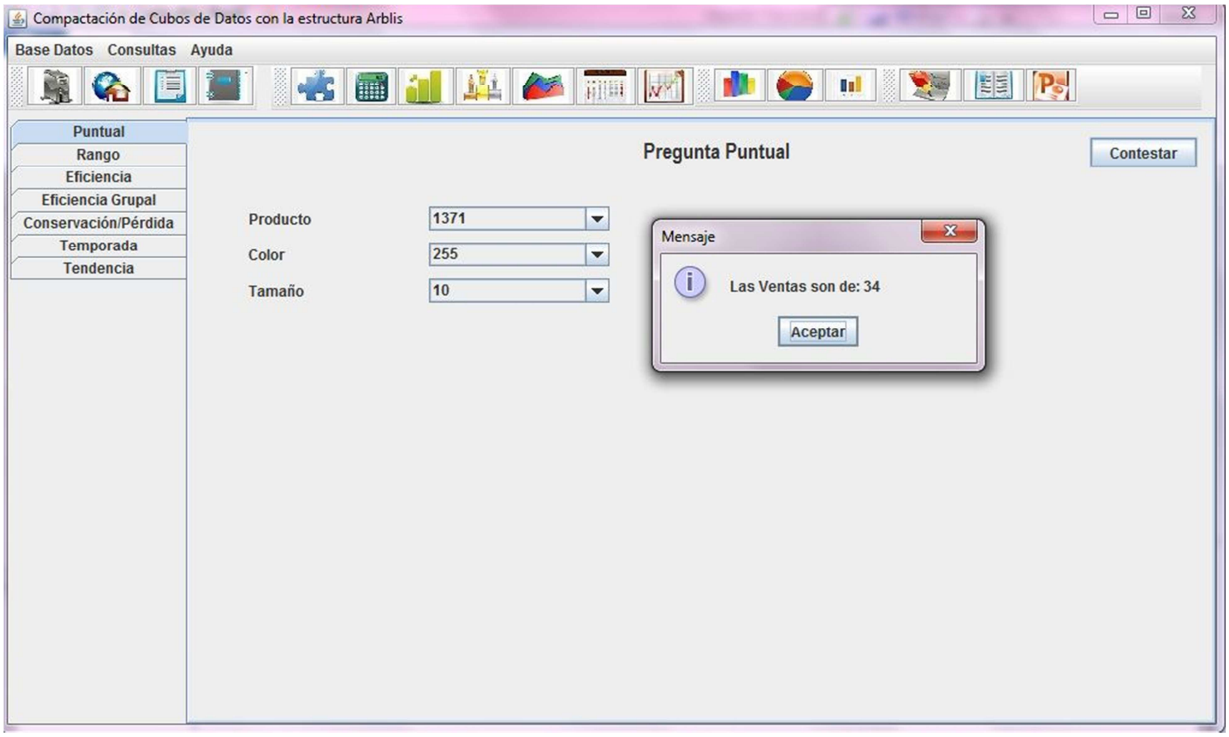


Ilustración 5. Ejemplo de Pregunta Puntual para cubo de datos de 3 dimensiones

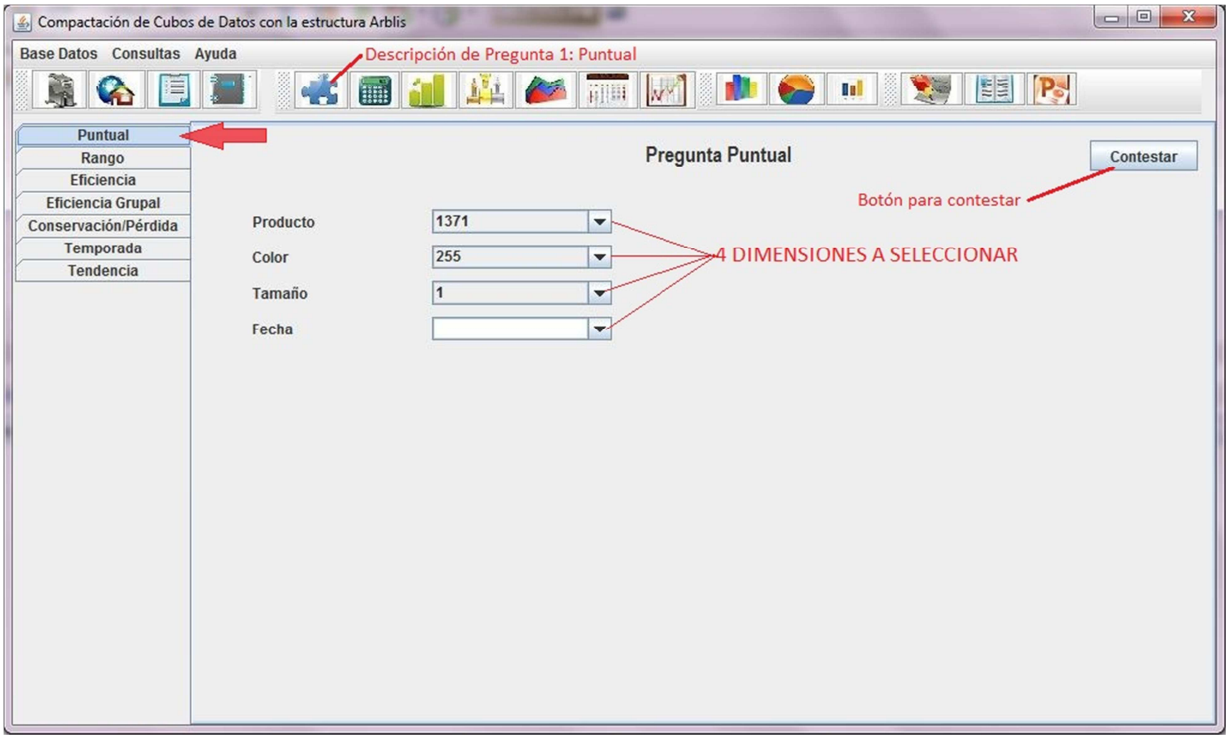


Ilustración 6. Captura de Datos en Pregunta Puntual

5.2 PREGUNTA EN RANGOS

Esta pregunta se define sobre un solo cubo C1. Donde al menos uno de los elementos de $R1_i$ que definen al cubo está dado por un rango de valores $R1_i$ de tal forma que no es un único valor.

Para la solución de esta pregunta se parte de que la estructura AC debe de haber ya sido cargada por lo que el algoritmo para responder la pregunta 2 es el siguiente:

1. Comprimir los valores del rango dado para le cubo C1
2. Inicial ciclo principal
    1. Incrementar el contador de la dimensión n
    2. Iniciar k en 1
    3. Iniciar ciclo interno
        1. Si el contador de la dimension n-k es mayor o igual al máximo de n-k
            1. Si k es igual al número de dimensiones
                1. Salir del ciclo principal
            2. Fin del si
            3. Contador en la dimensión n-k se reinicia
            4. Contador en la dimensión n-(k+1) se incrementa en uno para pasar al siguiente
            5. Se incrementa el valor de k
        2. Si no
            1. Salir del ciclo interno
        3. Fin del si
    4. Fin del ciclo interno
    5. Obtener los valores de dicho dato
    6. Descomprimir los valores
    7. Evaluar el rango de la dimensión anterior
    8. Si los datos están dentro del rango comprendido
        1. Obtener el hecho a partir de la pregunta puntual para estos datos
        2. Incrementar la suma del cubo
        3. Incrementar el contador de número de elementos sumados
    9. Fin del si
3. Fin del ciclo principal
4. Regresar los valores de la suma y el contador

Para un mejor detalle observar la Ilustración 7 en donde se puede observar la solución de esta pregunta para n dimensiones.

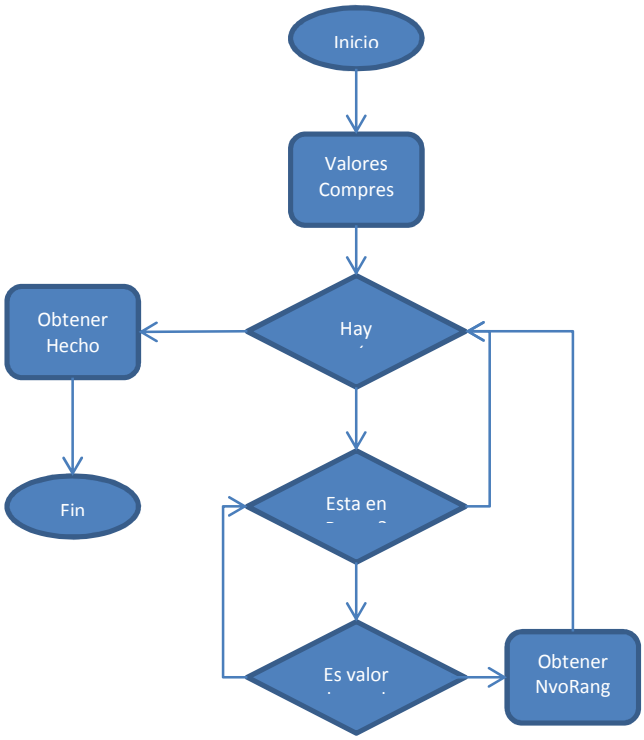


Ilustración 7. Diagrama de Flujo de la Pregunta de Rango

En la Ilustración 8 se muestran los datos a capturar para una pregunta de rango en un cubo de datos de 4 dimensiones. Observe que se cuenta con 8 campos a ser llenados, cada par corresponde a una dimensión de tal forma que se marque el inicio y fin del rango de esa dimensión.

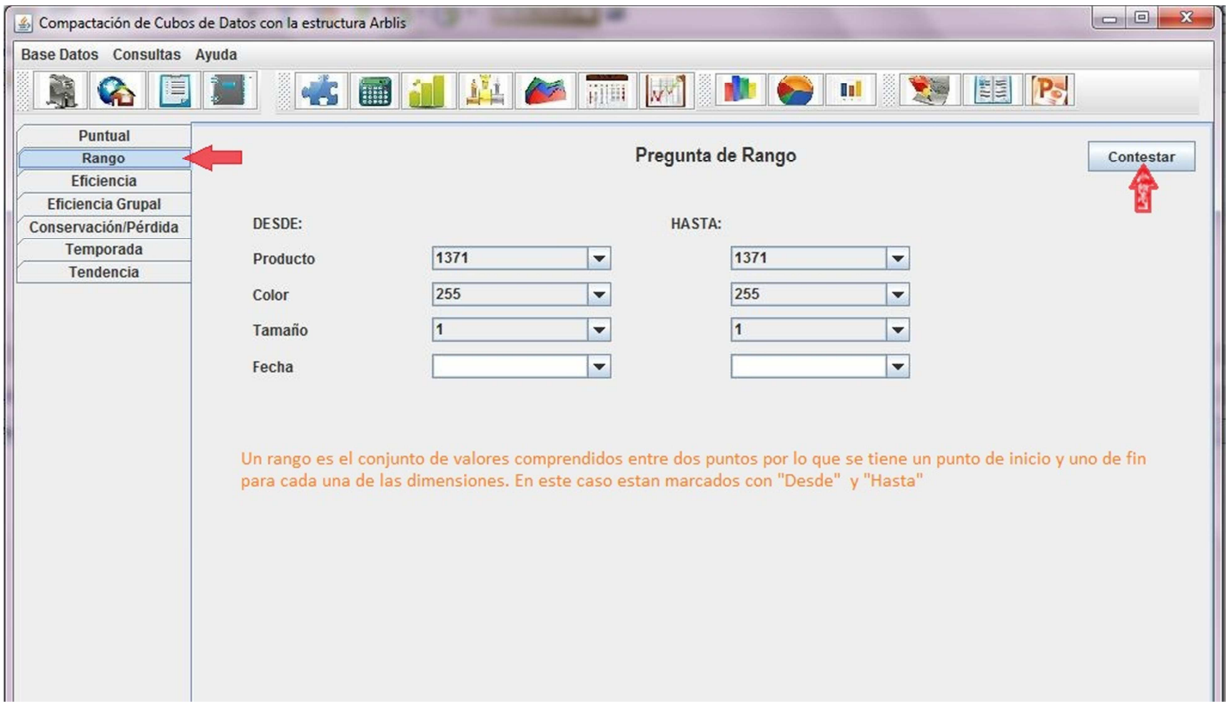


Ilustración 8. Captura de Datos en Pregunta de Rango

5.5.3 PREGUNTA DE EFICIENCIA GLOBAL

Esta Pregunta se define sobre dos cubos por lo que se reciben dos conjuntos de rangos que serán evaluados con la pregunta de rango individualmente y posteriormente se calculará la eficiencia de éstos. Véase el siguiente algoritmo:

1. Comprimir valores de rango del Cubo 1
2. Comprimir valores de rango del Cubo 2
3. Obtener la suma y el contador del Cubo 1 a partir de la llamada a la pregunta de rango para los valores del Cubo 1
4. Obtener la suma y el contador del Cubo 2 a partir de la llamada a la pregunta de rango para los valores del Cubo 2
5. Obtener la eficiencia a partir de la Ecuación 3
6. Regresar el valor de eficiencia

Ecuación 3. Eficiencia Global

$$E = 100 * \left(\frac{sum\ C_2}{sum\ C_1} - 1\right)$$

En la Ilustración 9 se muestran los datos a capturar para una pregunta de eficiencia global en un cubo de datos de 4 dimensiones. Se hacen acotaciones de dos cubos de datos lo cuales serán comparados.

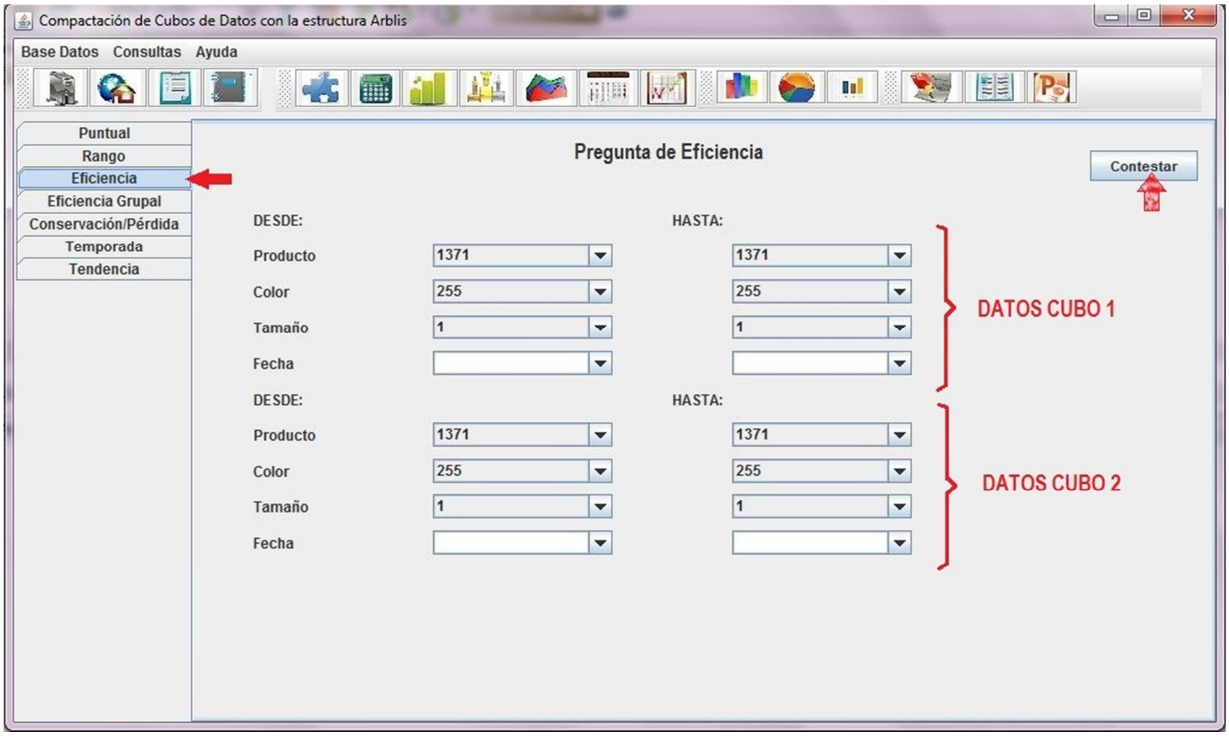


Ilustración 9. Captura de Datos para Pregunta de Eficiencia Global

5.5.4 PREGUNTA DE EFICIENCIA GRUPAL

En esta pregunta se define la dimensión sobre la que se desea calcular la eficiencia de los elementos. Se define el número de elementos *k* que se desean encontrar y saber si se requieren los mejores o peores.

La primera búsqueda es sobre el Cubo 1 que devolverá los k mejores/peores elementos, los cuales serán buscados en el Cubo 2 para comparar en comportamiento de los k elementos en 2 rangos diferentes.

La pregunta se resuelve a través del siguiente algoritmo:

1. Obtener todos los valores posibles de la dimensión sobre la que se desea la comparación
2. Comprimir valores de rango del Cubo 1
3. Mientras existan valores posibles, i
    1. Formar la consulta con el i-esimo valor sustituido a los datos a consultar
    2. Responder con rango los datos a consultar formados
    3. Si i es menor al número de mejores
        1. Guardar el valor del dato
        2. Guardar la Suma en su primer cubo
    4. Si no
        1. Buscar el menor/mayor
        2. Sustituir en éste el valor del dato
        3. Sustituir la suma en su primer cubo
    5. Fin del si
4. Fin del Mientras
5. Para i=0, mientras sea menor de k
    1. Sustituir en los datos a consultar el dato i
    2. Hacer la pregunta por rango con este dato a consultar
    3. Guardar la suma en su segundo cubo
6. Fin del mientras
7. Para i=0, mientras sea menor de k
    1. Calcular la eficiencia para C1 y C2 de i según Ecuación 3 u guardarlo
8. Fin del mientras
9. Regresar los valores con su eficiencia para su gráfica.

En la Ilustración 10 se muestran los datos a capturar para una pregunta de eficiencia grupal en un cubo de datos de 4 dimensiones. Se solicitan dos rangos para cubos de datos. Los datos para el primer cubo de dato son para el cubo en el que se hará la búsqueda de los mejores. Una vez que se cuenta con esta lista se buscan dichos elementos en el segundo cubo de dato y se calcula su eficiencia. Por lo que también se solicita la cantidad de elementos que se quieren en la lista de mejores/peores.

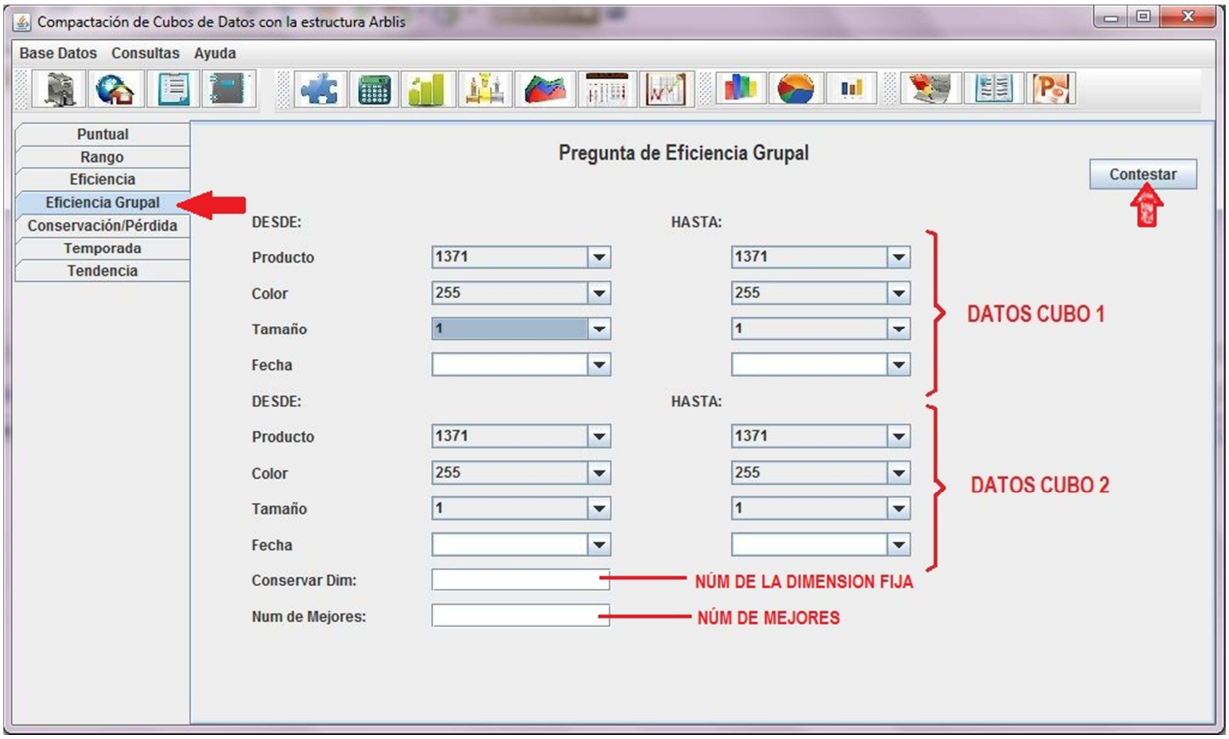


Ilustración 10. Captura de Datos para Pregunta de Eficiencia Global

5.5.5 PREGUNTA SOBRE CONSERVACIÓN Y PÉRDIDA

En esta pregunta se define la dimensión sobre la que se desea calcular la eficiencia de los elementos. Se define el número de elementos *k* que se desean encontrar y saber si se requieren los mejores o peores en un Cubo 1. Se repite la operación para un Cubo 2 y se verifica cuales se conservaron y cuales se perdieron al comparar ambos cubos.

Responde al siguiente algoritmo:

1. Se obtienen los k mejores en el Cubo 1
2. Comprimir valores de rango del Cubo 1
    1. Obtener todos los valores posibles de la dimensión sobre la que se desea la comparación
    2. Mientras existan valores posibles, i
        1. Formar la consulta con el i-esimo valor sustituido a los datos a consultar
        2. Responder con rango los datos a consultar formados
        3. Si i es menor al número de mejores
            1. Guardar el valor del dato
            2. Guardar la Suma en su primer cubo
        4. Si no
            1. Buscar el menor/mayor
            2. Sustituir en éste el valor del dato
            3. Sustituir la suma
        5. Fin del si
    3. Fin del Mientras

3. Repetir para el Cubo 2
4. Comparar cubos:
    1. Para los k elementos del Cubo 1, i=0
        1. Para los k elementos del Cubo 2, j=0
            1. Ver si los mejores en Cubo 1 en i es igual a los mejores en Cubo 1 en j
        2. Fin del ciclo
    2. Fin de ciclo
5. Regresar elementos iguales
6. Graficar

En la Ilustración 11 se muestra que es requerido dos rangos de valores (rango para obtener dos cubos de datos) de tal forma que se obtienen la cantidad de los mejores requeridos a buscar en cada uno de los cubos y posteriormente se hace una comparación para obtener cuáles se encuentran entre los mejores/peores en ambos casos.

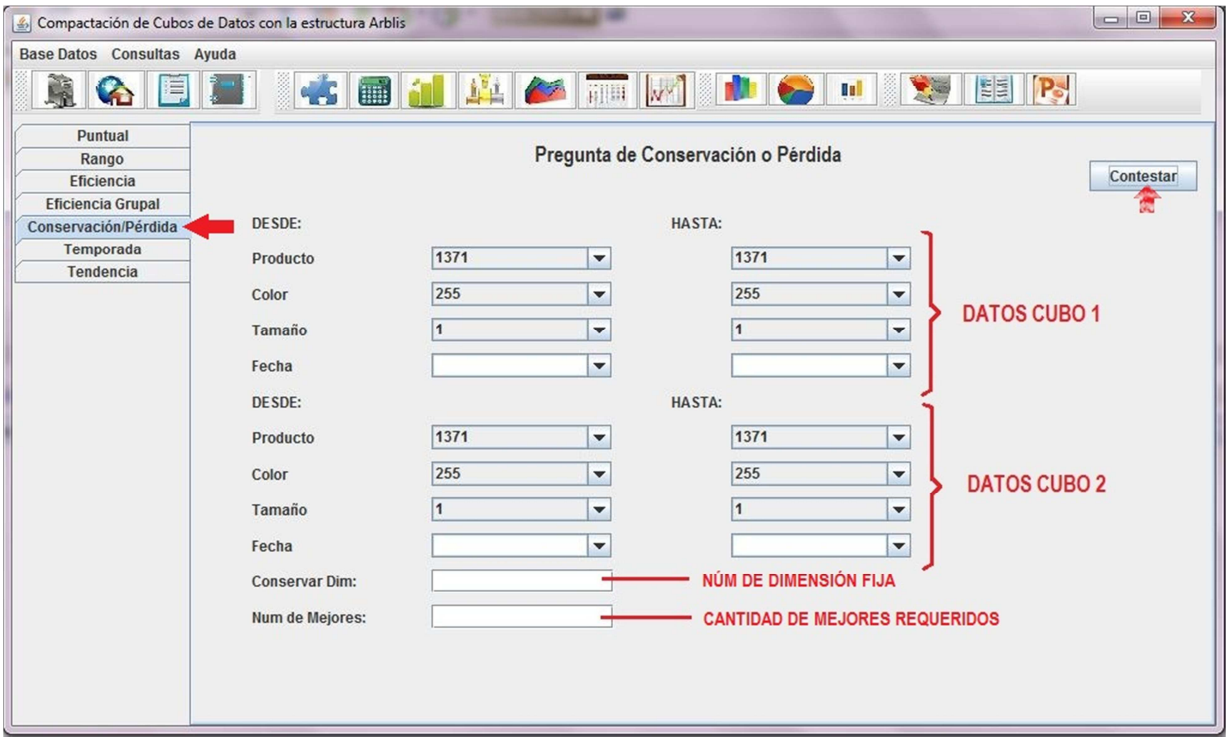


Ilustración 11. Captura de Datos para Pregunta de Conservación o Pérdida

5.5.6 PREGUNTA DE TEMPORALIDAD

Esta pregunta responde a los elementos que permanecen durante temporadas como los mejores/peores. Para ello se define la dimensión donde se buscan los elementos, se definen a cuantos k mejores/peores elementos de la dimensión se desea ver si permanecieron en varios períodos de tiempo. A partir de un cubo inicial Cubo 1.

Los periodos de tiempo están determinados por el tipo de temporadas (daily, weekly, monthly, bimonthly, quarterly, fourMonths, annual) y el número de temporadas que se desea analizar.

1. Obtener la dimensión de tiempo sobre la que se navegará

2. Comprimir valores de rango del Cubo 1
3. Si el número de temporadas es uno entonces
    1. Calcular los k mejores en el cubo 1
4. Si no
    1. Copiar los valores del cubo 1 al cubo 2
    2. Calcular los k mejores en el cubo 1
    3. Para i=0, mientras sea menor al número de temporadas
        1. Reasignar las fechas en el cubo 2
        2. Obtener los k mejores en el cubo 2
        3. Comparar cuáles permanecen
    4. Fin del mientras
5. Fin del si
6. Obtener el número de elementos que se conservaron
7. Obtener la matriz que guarda las respuestas de la intersección de los k cubos
8. Devolver la matriz para obtener su gráfica y tabla de valores correspondiente

Para responder preguntas de temporada es necesario contar un una dimensión de tiempo a través de la cual se hará el desplazamiento un número determinado de veces (Campo de Num de Temps), el desplazamiento depende del tipo de temporada de tal forma que puede ser diario, semanal, mensual, bimestral, trimestral, cuatrimestral, semestral o anual seleccionándolo en el campo de Tipo Temporada de la Ilustración 12. Esto se realiza en un cubo de datos según el número de dimensiones empleadas (4 en el caso de la Ilustración 12), estableciendo la dimensión sobre la que se hará la comparación de la cantidad de los mejores.

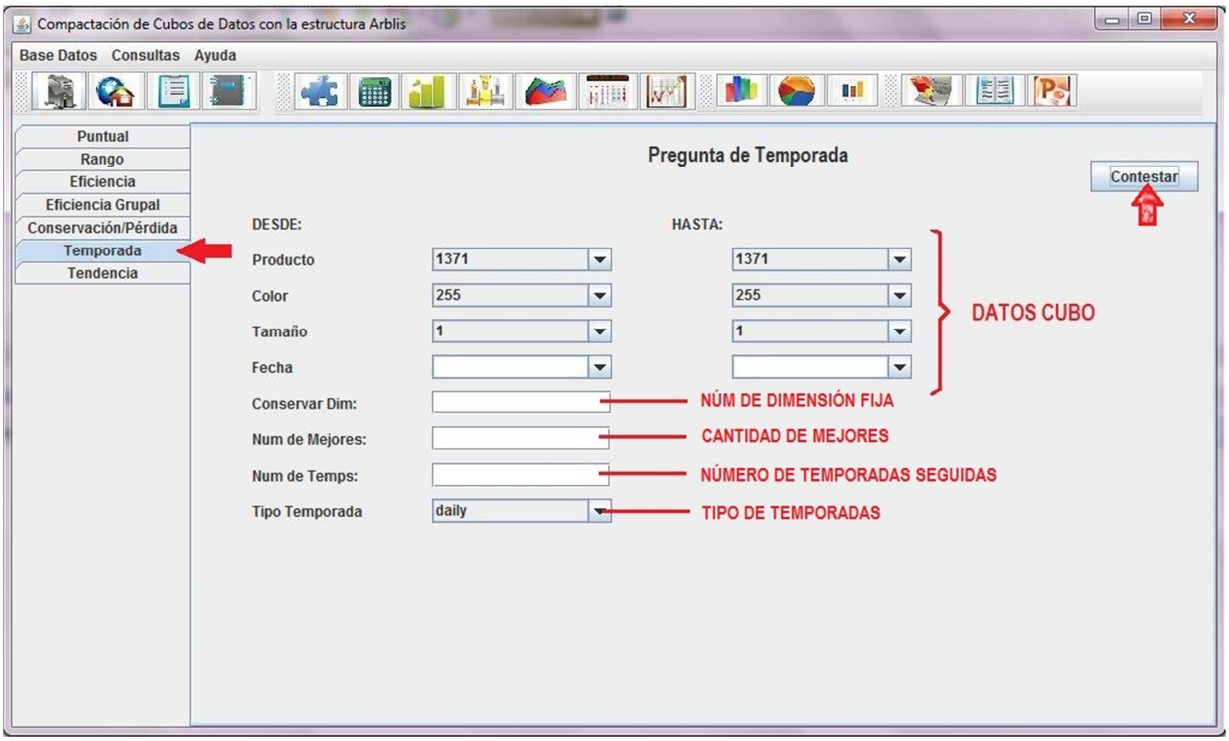


Ilustración 12. Captura de Datos para la Pregunta de Conservación y/o Pérdida

5.5.7 PREGUNTA DE TENDENCIA

Esta pregunta responde a los elementos que permanecen durante temporada. Para ello se define la dimensión donde se buscan los elementos. A partir de un cubo inicial Cubo 1. Se buscan todos los elementos durante cierto número y tipo de temporadas.

Los periodos de tiempo están determinados por el tipo de temporadas (daily, weekly, monthly, bimonthly, quarterly, fourMonths, annual) y el número de temporadas que se desea analizar.

1. Obtener la dimensión de tiempo sobre la que se navegará
2. Comprimir valores de rango del Cubo 1
3. Mientras existan valores en la dimensión donde se buscan los elementos, i=0
    1. Sustituir el valor de la dimensión i
    2. Copiar Valores para Cubo 1 a valores para cubo 2
    3. Obtener fecha para el cubo 2
    4. Mientras j sea menor al número de temporadas, j=0
        1. Responder con rango el Cubo 2
        2. Guardar la suma
        3. Calcular la nueva fecha para Cubo 2
    5. Fin del mientras
    6. Incrementar m
4. Fin del mientras

Para obtener la nueva fecha de los cubos se hace in incremento dependiendo del tipo de la temporada seleccionado y la función regresa la nueva fecha.

Se muestra en la Ilustración 13 los campos a llenar en la pregunta de tendencia para un cubo de 4 dimensiones que ha sido cargado con anterioridad. Para resolver esta pregunta se tiene un rango de valores en el que se buscaran la cantidad de “Num de temporadas” a partir del inicio según el “Tipo de temporada”, de tal forma que se tendremos los elementos de la dimensión fija con sus respectivas sumas en el cada uno de los periodos de tiempo obtenidos.

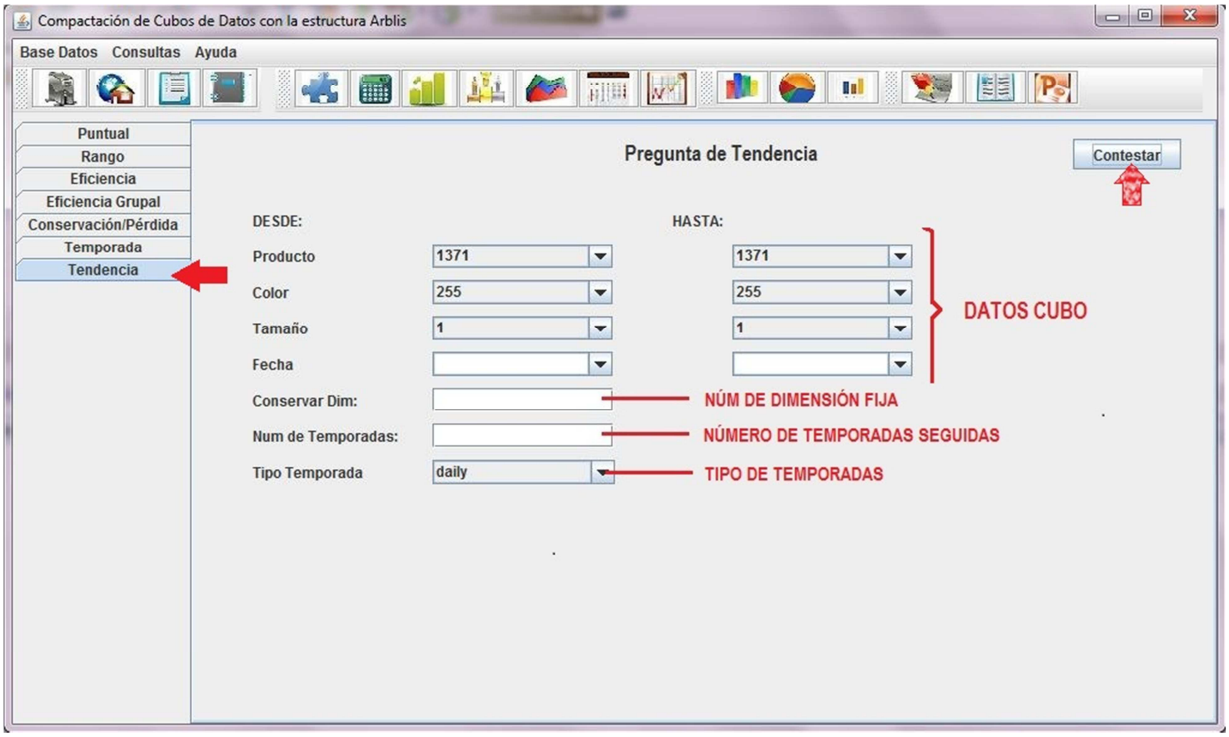


Ilustración 13. Captura de Datos para la Pregunta de Temporalidad

5.6 CONCLUSIÓN

En este capítulo se detalló la forma en que se implementó la compactación de los datos contenidos en la estructura Arblis, haciendo uso de descriptores para cada una de las dimensiones, los cuales contienen el tipo de compactación empleada en dicha dimensión. A partir de estos se lee una segunda vez los datos originales y se empieza la compactación a la estructura AC que es guardada con los datos compresos.

Una vez que se ha creado puede ser cargada a memoria sin la necesidad de volver a realizar la compactación, al cargarla en memoria se hace uso de otros arreglos que facilitan la navegación a través de la estructura y con ello facilitan la respuesta a las preguntas de negocio.

Se proponen siete preguntas de negocio que son resueltas con AC compresa, la resolución de estas está basada en mezclas de las dos primeras: puntual y rango, cuyos algoritmos fueron realizados para el manejo de n dimensiones, aumentando la complejidad de dichos algoritmos.

CAPÍTULO VI. PRUEBAS Y RESULTADOS

La compactación de la estructura Arblis a partir de un archivo *.out permite la generación de catálogos para cada una de las dimensiones así como un nuevo archivo con la estructura Arblis compresa llamada estructura AC, como se muestra en la Ilustración 1.

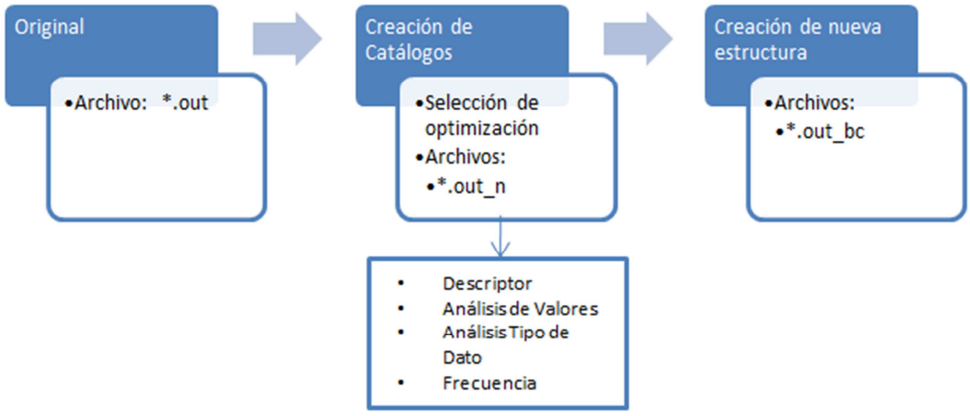


Ilustración 1. Generación de Catálogos y AC

Los archivos generados son guardados en disco por lo que esto permite una medición en la memoria del disco duro. Por otra parte estos son cargados a memoria principal haciendo uso de arrgelos y matrices para su navegación efectiva.

Los resultados que son mostrados a continuación toman como base el Historial de Ventas (SH) en diversos tamaños:

- 1 millón de registros
- 2 millones de registros
- 3 millones de registros
- 5 millones de registros
- 10 millones de registros
- 12 millones de registros
- 15 millones de registros

6.1 TAMAÑO EN DISCO DURO

Al comprimir el Historial de Ventas se obtuvieron los resultados mostrados en la Tabla 1. Como se puede observar la estructura AC con sus respectivos catálogos logra ocupar hasta un 66% del tamaño original, de tal forma que logra reducir la estructura Arblis hasta en un 34%. (Véase Tabla 1. Compresión de SH en DD e Ilustración 2. Gráfica de Compresión de SH en DD)

Tabla 1. Compresión de SH en DD

Registros	Origen	Compresión	Compresión
Millones	en MB		
1	22	18	20.18%
2	36	27	25.00%
3	51	36	29.41%
5	79	55	30.38%
10	149	101	32.21%
12	178	119	33.15%
15	220	147	33.18%

En la Ilustración 2 se muestra la diferencia en megabytes del espacio ocupado de AC con respecto al original. Dichos datos fueron obtenidos al ejecutar la creación de AC, una vez generados los archivos se recopilo la información presentada.

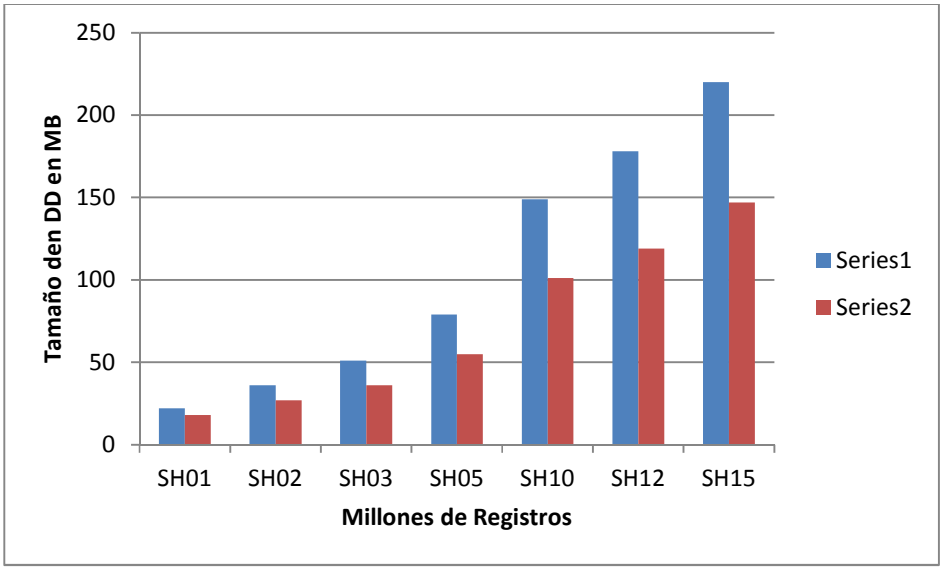


Ilustración 2. Gráfica de Compresión de SH en DD

Como se logra observar la reducción en disco duro es significativa, llegando a ser hasta de un 33%. Aplicando compresión en texto el resultado podría ser mejorado.

6.2 TAMAÑO EN MEMORIA PRINCIPAL

El cálculo de la memoria utilizada en memoria principal ha sido medido con apoyo de la clase Runtime, la cual permite obtener la memoria total y la ocupada, sin embargo no se cuenta con un desglose paso por paso del tamaño real de cada una de las variables en memoria de los diversos arreglos.

Tomando eso en consideración se han obtenido las siguientes lecturas para saber el espacio ocupado. En la Tabla 2 se muestra el tamaño de Arblis (origen) y el tamaño de la estructura AC que implementa la compactación, obteniendo así dichos resultados. (Véase Tabla 2. Tamaño de SH en Memoria Principal e Ilustración 3. Gráfica de Compresión de SH en MP)

Tabla 2. Tamaño de SH en Memoria Principal

Millones de Registros	Origen	Compresión	% de Compactación
1	32	26	20.75%
2	53	40	24.53%
3	72	56	22.22%
5		84	

Como se puede observar, en cinco millones de registros no se cuenta con datos para Arblis. Esto se debe a que en este punto se tuvo un desbordamiento de memoria, debido a la gran cantidad de espacio que ésta requería. En la se observa con mejor detalle la comparación entre Arblis y AC en memoria principal para cada una de las base de datos, cuyo número de registros esta dado en millones.

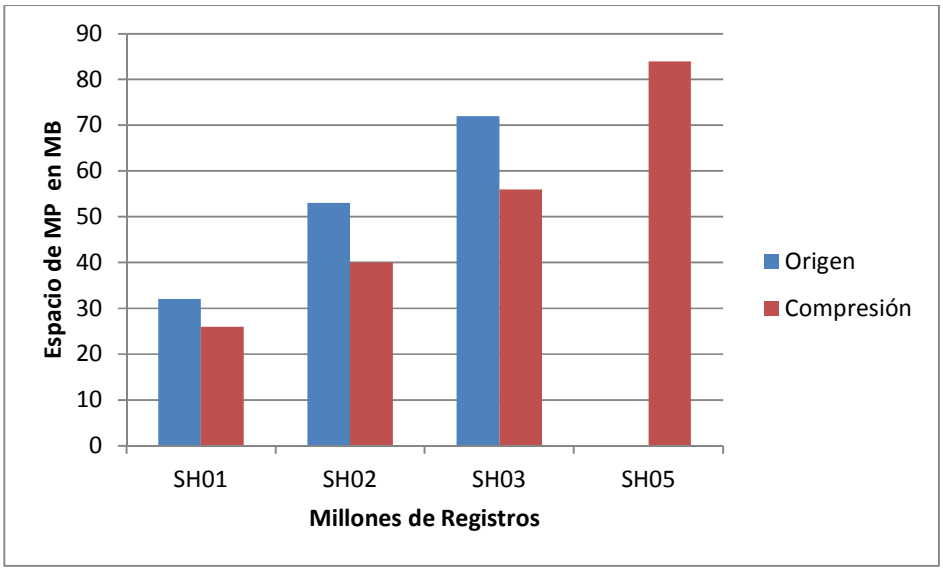


Ilustración 3. Gráfica de Compresión de SH en MP

6.3 TIEMPO DE COMPACTACIÓN

El tiempo de compactación de cada una de las base de datos prueba fue aumentando al formar los catálogos, sin embargo al formar la estructura el incremento fue menor. (Véase Tabla 3. Tiempo de la generación de la compactación en SH e Ilustración 4. Gráfica de tiempo de generación de la compresión de SH)

Tabla 3. Tiempo de la generación de la compactación en SH

	CATALOGOS	ESTRUCTURA
SH01	10	103
SH02	18	100
SH03	26	107
SH05	36	103
SH10	42	108

Por lo que la primera de parte de la compactación hace que el tiempo crezca proporcionalmente al incremento del número de registros, mientras que la generación de la estructura una vez que se cuenta con los catálogos, no muestra un comportamiento creciente tan evidente.

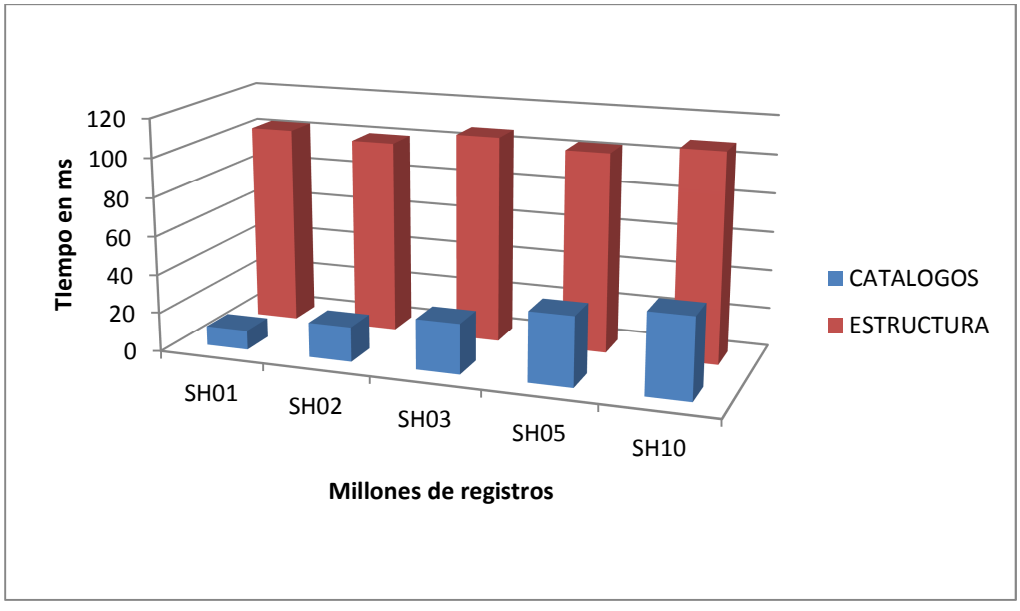


Ilustración 4. Gráfica de tiempo de generación de la compresión de SH

6.4 SOBRE LAS 7 PREGUNTAS

6.4.1 PREGUNTA PUNTUAL

La pregunta puntual permite encontrar datos específicos sobre un catálogo, en la Ilustración 5 se puede ver la captura de datos en pantalla de una base de datos de 3 dimensiones: producto, color y tamaño de tal forma que al darle valores en particular a estos 3 elementos se contestan preguntas como:

¿Cuál fue la venta del producto con ID 1371, de color rojo (255) de tamaño grande (10)?

A un costado se observa el mensaje que muestra que la venta es de 34.

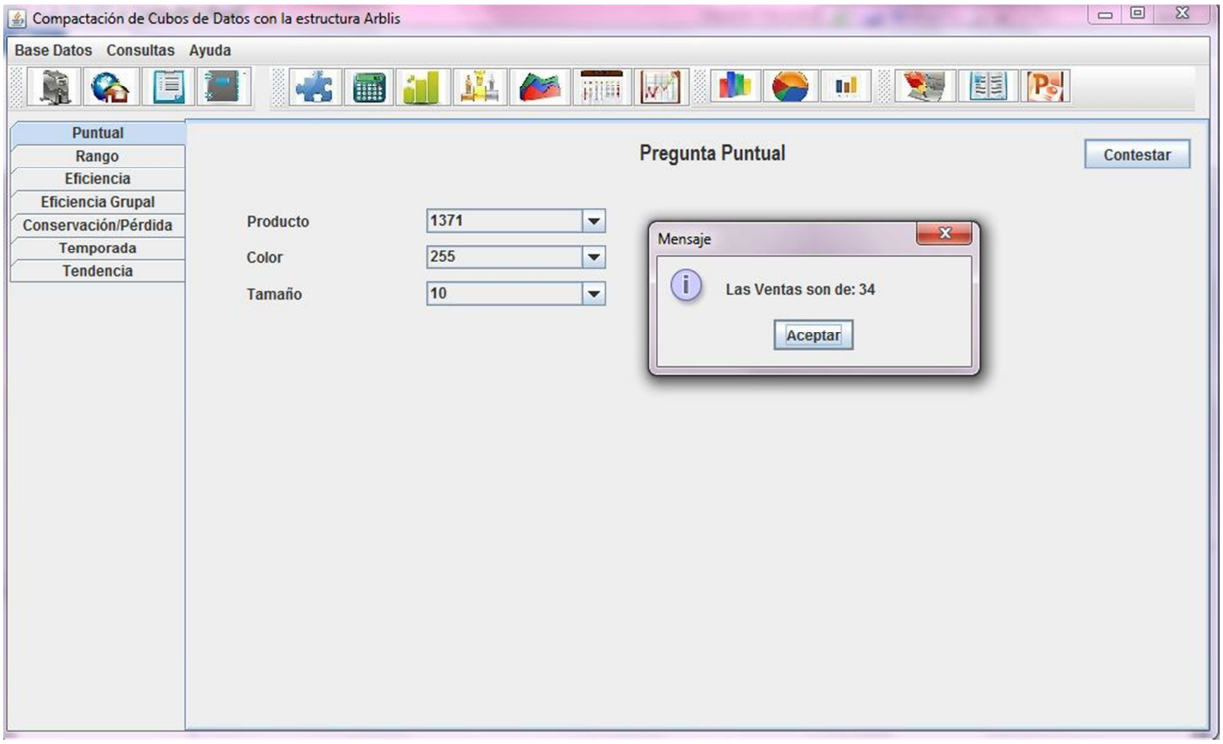


Ilustración 5. Pregunta Puntual 3D

Las respuestas obtenidas con la compresión y sin ella son las mismas como se puede observar en Ilustración 6 aunque el tiempo de respuesta varía ligeramente al aplicar la compactación. Cabe destacar que la variación de tiempo en diversas pruebas fue de 0 a 2 milisegundos, obteniendo un mejor tiempo en la mayoría de los casos la respuesta sin compactación.

```
Memoria diferencia inic: 217729128
Respuesta con compresión:9
Tiempo: 1, tiempoIncio: 1289229486998, tiempoFin: 1289229486999
Memoria ocupada: 246208000
Total: 1658990
Memoria ocupada: 487930736
Memoria diferencia inic: 241722736
Respuesta sin compresión: 9
Tiempo: 0, tiempoIncio: 1289229491883, tiempoFin: 1289229491883
```

Ilustración 6. Respuestas con y sin compactación en pregunta puntual

6.4.2 PREGUNTA DE RANGOS

La pregunta de rangos devuelve el valor de la sumatoria de todos los registros que se encuentren en los límites dados para cada una de las dimensiones. En la Ilustración 7 se muestra un cubo de datos de 4 dimensiones en donde se permite seleccionar los valores de inicio y fin para cada dimensión, en este caso permite que resolvamos preguntas como:

¿Cuál es la suma de los productos que van de un ID 5 hasta 15, desde la tienda 50 hasta la tienda 183960, con la promoción 9999, desde 01/enero/1998 hasta 29/11/2000?

Donde la respuesta se muestra en el cuadro de diálogo en la parte superior que informa que la suma de las ventas es de 179, y que fueron sumados un total de 17 registros.

Ilustración 7. Pregunta de Rango 4D

Las respuestas que se obtienen con y sin compactación son las mismas lo que prueba que el algoritmo fue adaptado de manera satisfactoria a la compactación. En la Ilustración 8 se observa que en ambos casos la respuesta para una pregunta de rango es de 1678, donde fueron sumados dos registros. Sin embargo el tiempo de respuesta varía de 2 a 5 milisegundos obteniendo un mejor tiempo de respuesta la pregunta con compactación, ya que trabaja con elementos de menor tamaño.

```
Suma sin compactación: 1687 de 2 registros
Tiempo sin: 5, tiempoIncio: 1289230432409, tiempoFin: 1289230432414
Entrar a etiqueta
Suma con compactación: 1687 de 2 registros
Tiempo con: 2, tiempoIncio: 1289230432414, tiempoFin: 1289230432416
Suma: 1687 Num de Reg:2
BUILD SUCCESSFUL (total time: 1 second)
```

Ilustración 8. Respuestas con y sin compactación en pregunta de rango

6.4.3 PREGUNTA DE EFICIENCIA GLOBAL

La pregunta de eficiencia global compara dos rangos dándonos la eficiencia del primero con respecto al segundo de tal forma que se pueden contestar preguntas como:

¿Cómo se comporta el producto 5 con respecto al producto 10 en las tiendas de 50 a 183960, con la promoción 9999, en el periodo que va de 01/01/1998 hasta 29/11/2000?

La cual se puede observar en la Ilustración 9 donde para contestarla se han seleccionado los mismo valores en los campos del primer cubo de datos con respecto al segundo variando el ID del producto, de tal forma que en el primer cubo se tiene:

- Producto 5
- Tiendas que van de 50 a 183960
- Promoción 9999
- Periodo que va de 01/01/1998 hasta 29/11/2000

Y para el segundo cubo de datos se tiene:

- Producto 10
- Tiendas que van de 50 a 183960
- Promoción 9999
- Periodo que va de 01/01/1998 hasta 29/11/2000

Obteniendo una respuesta que se muestra en el recuadro inferior, donde muestra los resultados para ambos cubos de datos y posteriormente una comparación de ellos obtenida a través de la eficiencia calculada por la fórmula de la Ecuación 9.

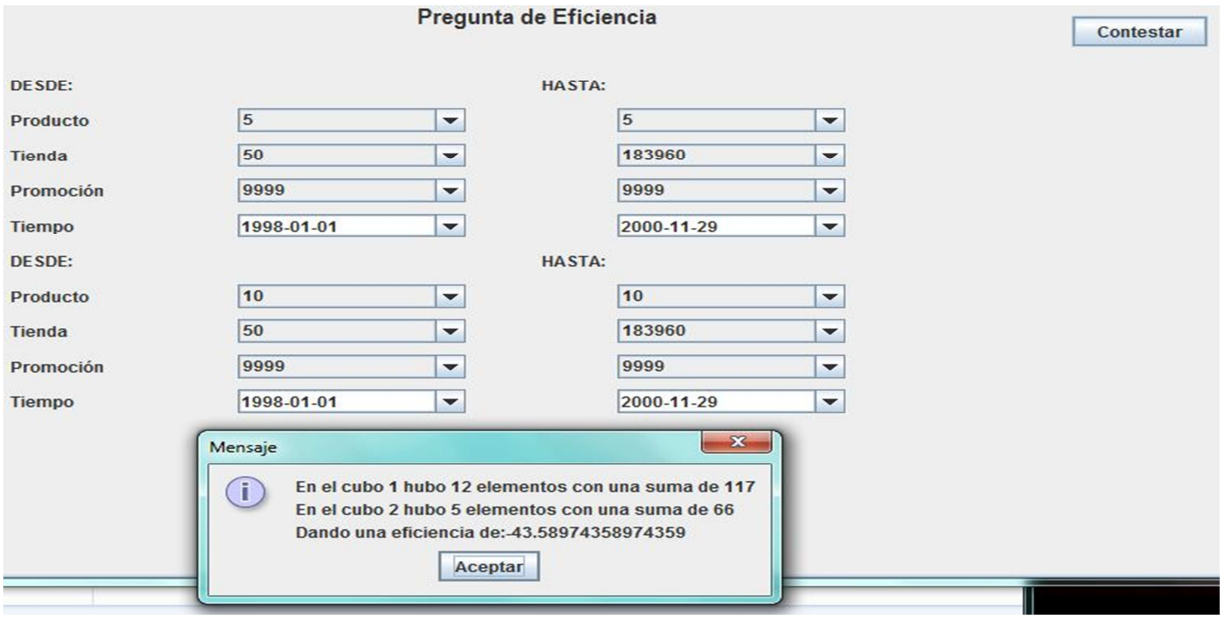


Ilustración 9. Pregunta de Eficiencia de 4D

6.4.4 PREGUNTA DE EFICIENCIA GRUPAL

La eficiencia grupal permite comparar cubos de datos generados en los rangos establecidos, de tal forma que se busca los m mejores en el primer cubo de datos y esos elementos de la dimensión que se conserva son buscados en el segundo cubo de datos para comparar el comportamiento de éstos. De tal forma que se pueden establecer preguntas como:

¿Cuáles son los mejores 2 productos de la tienda 255 en el periodos de 01/01/2000 a 01/01/2004 y que eficiencia muestran con respecto a la tienda 300?

De tal forma que en la figura Ilustración 10 se introduce la misma fecha en ambos cubos de datos y la tienda varía en los cubos de datos para hacer la comparación, en el caso de la dimensión 0, es decir, la de producto no importa los valores seleccionados ya que estos irán variando cuando se comparan los cubos de datos para encontrar los 2 mejores.

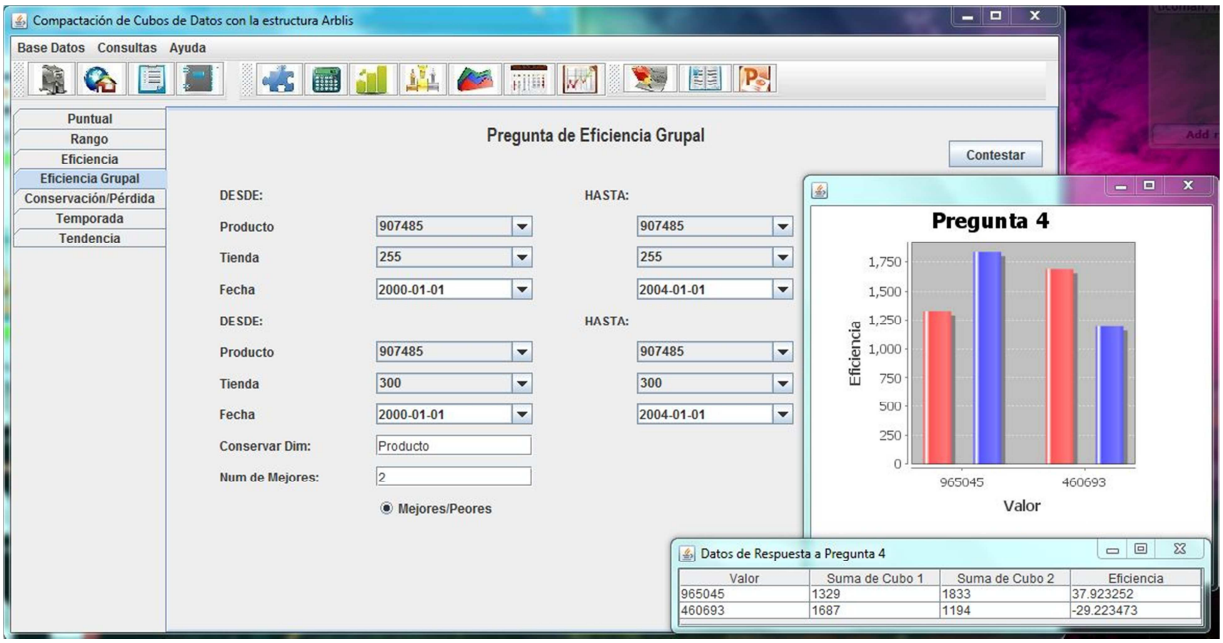


Ilustración 10. Pregunta de Eficiencia Grupal en 3D

La respuesta se muestra en una tabla donde se muestran los m productos con sus respectivos valores en cada uno de los cubos de datos y la eficiencia correspondiente. En la gráfica se observa la suma en ambos cubos de datos (uno dado en rojo y el segundo en azul) para cada uno de los productos de tal forma que para el usuario sea visible el establecimiento de la diferencia entre estos 2 cubos de datos.

6.4.5 PREGUNTA SOBRE CONSERVACIÓN Y PÉRDIDA

Esta pregunta permite observar que elementos se mantienen entre los mejores al comparar dos cubos de datos, de tal forma que nos permite contestar preguntas como:

¿Qué productos se conservan entre los mejores 3 en el periodo de 01/01/2000 hasta 01/01/2004 de la tienda 255 con respecto a la tienda 300?

La cual se puede visualizar en la Ilustración 11 donde es establece los rangos en ambos cubos de datos, de tal forma que se obtienen los m mejores en cada uno de los cubos de datos y son comparados para observar cuál de ellos se mantuvo en los m primeros lugares en ambos rangos. En este caso la sabiendo cuáles eran los mejores 3 en cada cubo de datos se compararon estos datos y se observó que solo se mantuvo un elemento en los primeros 3 lugares por lo que es éste y sus valores en cada cubo de datos los que son mostrados en la gráfica.

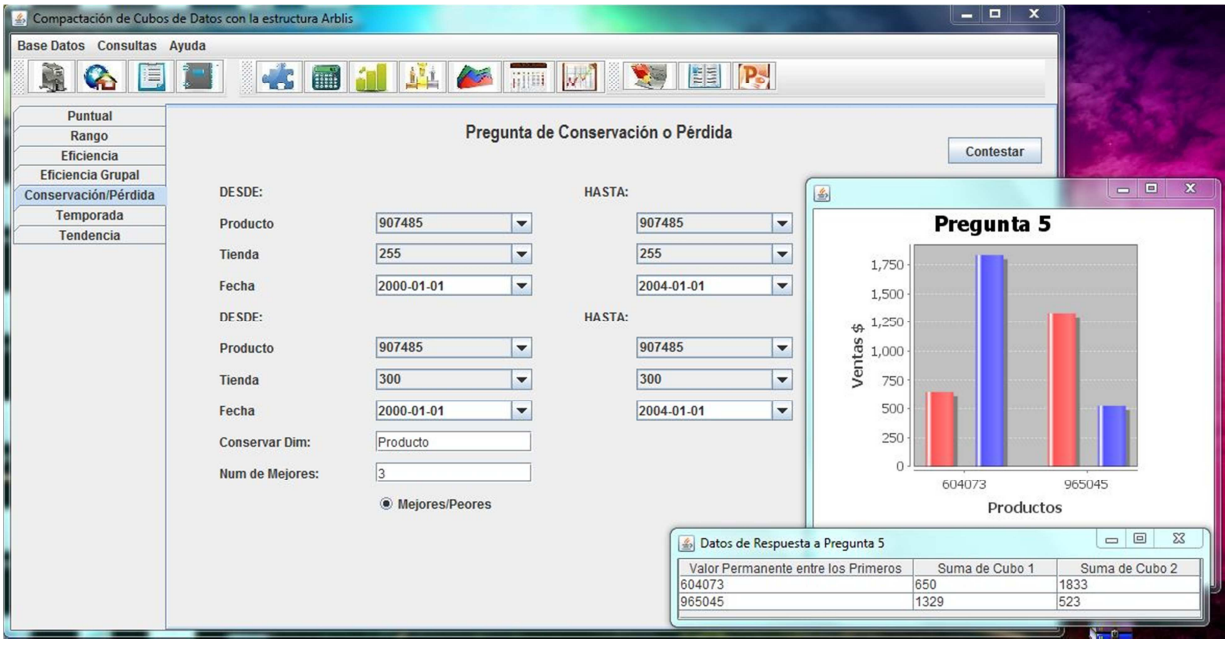


Ilustración 11. Pregunta de conservación y pérdida.

6.4.6. PREGUNTA DE TEMPORALIDAD

En esta pregunta se establece el cubo de datos inicial y posteriormente se generaran automáticamente los cálculos necesarios para generar respuestas en nuevos rangos dados por una variación en tiempo del tipo: anual, cuatrimestral, trimestral, mensual, semanal o diaria. El número de veces que el usuario requiera, estableciéndolo en el número de periodos de tiempo, por lo que se realiza una comparación similar a la existente en la pregunta de conservación y pérdida las veces establecidas en el periodo de tiempo con la determinada variación temporal. De tal forma que se pueden contestar preguntas como:

¿Qué productos se conservan entre los mejores 2 en el de la tienda 255 en 2 periodos de tiempo anuales empezando a partir del 01/01/2001?

En la Ilustración 12 se muestran los datos que fueron introducidos para esta pregunta, de tal forma que el tiempo inicial es el que es considerado la primera vez y para las sucesivas comparaciones se genera un nuevo rango según el tipo de temporada de tal forma que se van comparando los valores del inicial con el segundo calculado y el resultado de éstos se compara con una tercer temporada en caso de haberla y así sucesivamente. En este caso existe un producto que se conserva entre los dos mejores en 2 temporadas anuales partiendo del 01-enero-1001, el cual es mostrado en la tabla así como en la gráfica generada.

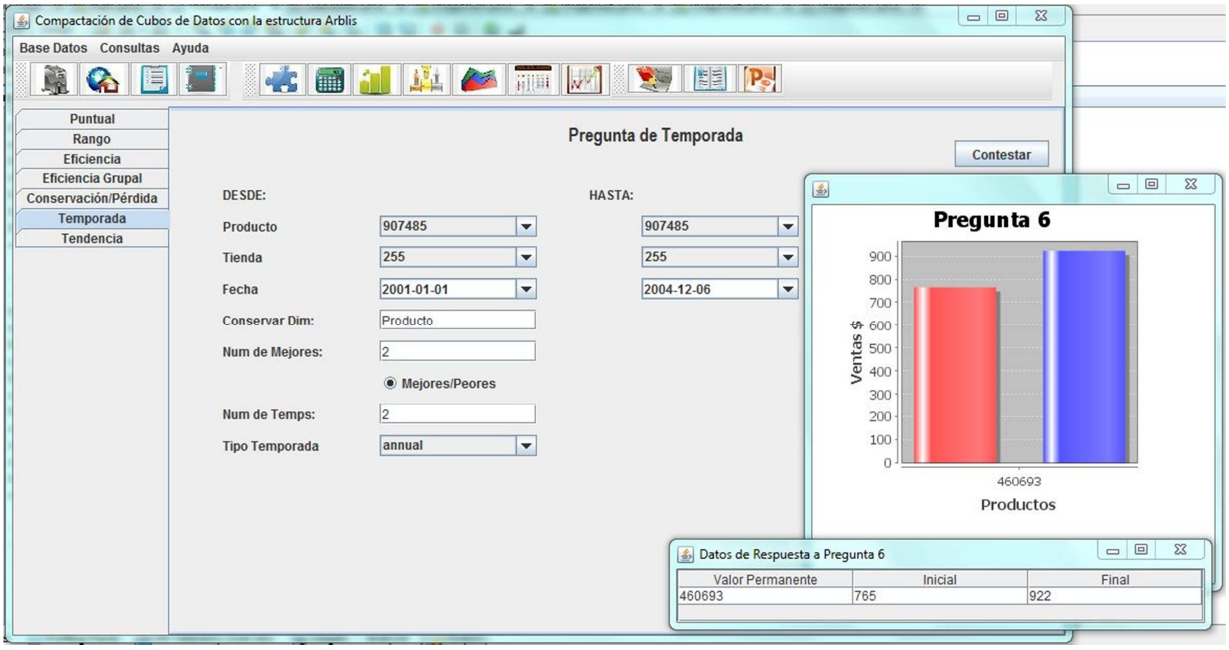


Ilustración 12. Pregunta de Temporada

6.4.7 PREGUNTA DE TENDENCIA

La tendencia es un patrón de comportamiento de los elementos de un entorno particular durante un período que nos determina la dirección del comportamiento de un elemento, sin embargo el término de tendencia en esta pregunta es utilizado para mostrar datos en el tiempo sin determinar un patrón en particular.

La pregunta de tendencia nos muestra en comportamiento de elementos en un periodo de tiempo, de tal forma que se puede observar el agregado (por ejemplo ventas) de los elementos en determinados periodos de tiempo, establecidos por la cantidad de periodos de tiempo que se requieran en temporadas: anual, cuatrimestral, trimestral, mensual, semanal o diaria. De tal forma que como se muestra en la Ilustración 13 se puede establecer preguntas como:

¿Cómo se comportan los productos de la tienda 255 en 3 periodos de tiempo anuales empezando a partir del 01/01/2001?

Donde nos muestra en un gráfico los 3 periodos de tiempo requeridos en el eje horizontal y la cantidad de ventas en el vertical, y en cada color representa un producto por lo que el producto representado por el color verde, incremento su venta en el segundo periodo con respecto al primero pero desapareció en el tercer periodo.

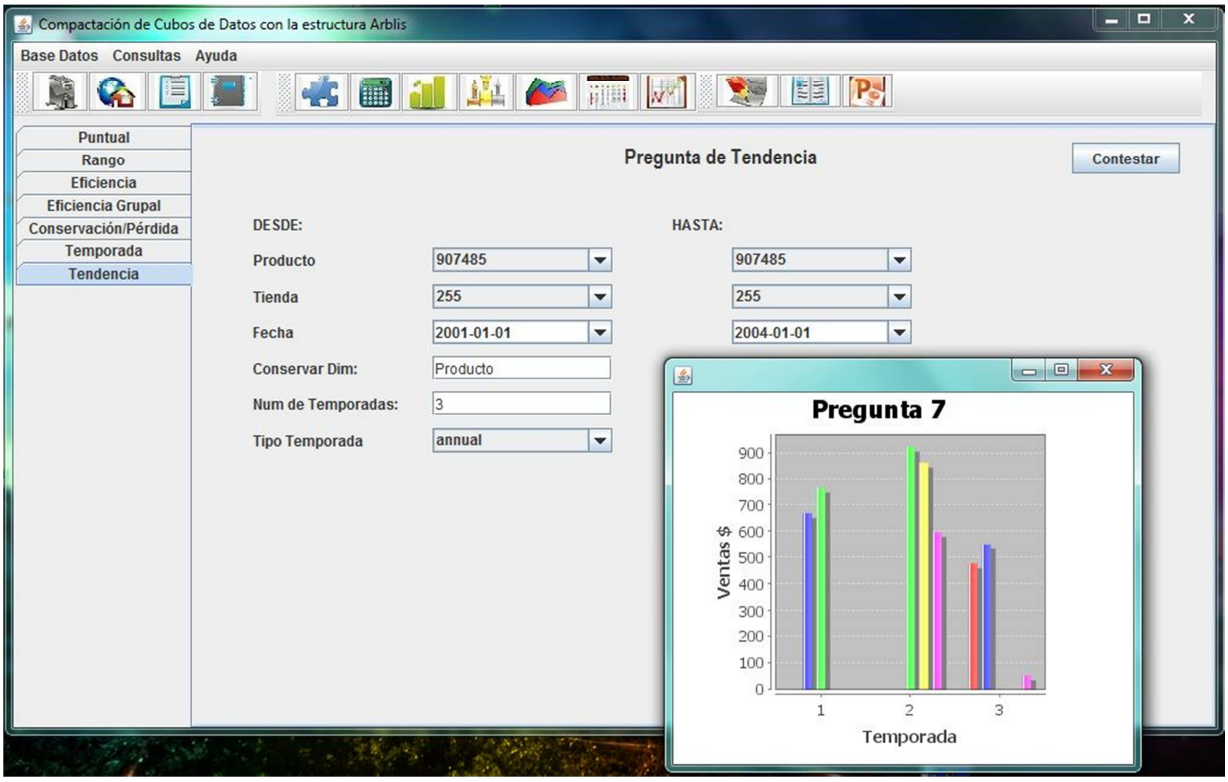


Ilustración 13. Pregunta de tendencia.

6.5 CONCLUSIONES

A lo largo de éste capítulo se muestran las pruebas realizadas sobre una gran cantidad de registros (millones), lo cual permite observar la eficiencia que tiene en espacio en disco comparando el tamaño en la estructura Arblis con el tamaño ocupado en AC.

Se puede observar que existe un incremento en espacio ocupado en disco con respecto al ocupado en la memoria principal. Esto se debe a que en memoria principal se trabaja con arreglos lo que hace que el tipo de dato guardado en el arreglo tome el de su mayor longitud para todo el arreglo a diferencia del disco duro que guarda únicamente los caracteres que se están utilizando.

CAPÍTULO VII. CONCLUSIONES

A lo largo de este capítulo se darán a conocer las conclusiones a las que se llegaron con respecto a la compactación de cubos de datos, así como las fortalezas y debilidades de la presente compactación a través de los alcances. Se considera que en el ámbito del presente trabajo se podría profundizar en algunos temas o llevar a cabo mejoras que permitan un crecimiento del proyecto por lo que es incluida una sección de trabajo a futuro.

7.1 CONCLUSIONES

Tras el estudio de diversos métodos de compactación [véase 2.4.1 MÉTODOS DE COMPACTACIÓN] se obtuvieron diversas propuestas: arreglos con elementos vacíos, arreglos sin elementos vacíos, mapas de bits y apuntadores a catálogos [véase 4.3 MÉTODOS DE COMPACTACIÓN] sobre los cuales se realizan pruebas para observar cual traería mejores resultados en el momento de su implementación, razón por la que es seleccionada la Estructura AC.

Se utiliza AC (Apuntadores a Catálogo) [véase4.3.4 APUNTADORES A CATÁLOGO (Estructura AC)] ya que cumple los requisitos de navegabilidad, el manejo de rangos y orden que plantea Arblis [véase4.2 ARBLIS], de tal forma que la compresión obtenida es monotónica y sin pérdida de información, ya que para todo v_i y v_j tales que $v_i \leq v_j$ se tiene que tiene que $C_m(v_i) \leq C_m(v_j)$ donde C_m es la compactación empleada.

La implementación de la compactación a través de AC se obtiene de los datos contenidos en la estructura Arblis, haciendo uso de descriptores para cada una de las dimensiones, los cuales contienen el tipo de compactación empleada en dichas dimensiones. A partir de éstos se lee una segunda vez los datos originales y se empieza la compactación creando AC que es guardada con los datos compresos. Una vez que se ha creado puede ser cargada a memoria sin la necesidad de volver a realizar la compactación y sobre la estructura compactada se resuelven las consultas de las siete preguntas de ANTECUMEM [véase[Martínez, 2007]].

La compactación obtenida llego a ser hasta de un 33% en disco duro [véase6.1 TAMAÑO EN DISCO DURO] y 25% en memoria principal [véase 6.2 TAMAÑO EN MEMORIA PRINCIPAL] haciendo uso de eliminación de datos redundantes, conversión a notación compacta y sustitución de datos repetidos, lo cual permitió formar la Estructura AC con las propiedades requeridas para su adaptación a la solución de las 7 preguntas planteadas sobre las cuales se logra que el impacto sea mínimo en cuanto a la navegabilidad.

Además de la compactación se logra el manejo de *n* dimensiones, lo cual implicó un análisis más detallado no solo en la creación de AC, también en los algoritmos empleados en las preguntas: puntual, rangos, eficiencia, eficiencia global, eficiencia grupal, conservación y pérdida, temporalidad y tendencia. Ya que se evita la limitación de 4 dimensiones [véase 5.5 RESOLUCIÓN DE PREGUNTAS PARA N DIMENSIONES].

Se obtuvo un código mejorado al manejar n dimensiones así como modificación sobre los algoritmos que permite el uso únicamente de los cubos de datos necesarios para la solución de las preguntas, descartando los cubo de datos que no contienen datos requeridos.

7.2 ALCANCES

La estructura AC permite almacenar mayor cantidad de datos en el mismo espacio, de tal forma que si se tiene un incremento en disco duro o en la RAM, el aprovechamiento del espacio será mejor.

En pequeñas o medianas empresas (PyMES) el incremento de RAM no es tan accesible como en grandes empresas, por lo que hacer uso del espacio disponible para incrementar los datos que pueden procesar simultáneamente es de gran utilidad.

La compactación es realizada una sola vez de tal forma que el tiempo empleado en la creación de Arblis se realiza de forma única quedando materializada por lo que no afecta la solución de la consultas, únicamente la compresión antes y después para dar al usuario los datos sin compresión, debido a que la búsqueda se realiza sobre la estructura compresa, lo cual es de gran ayuda en el desempeño del algoritmo.

La reducción realizada no fue óptima por lo que se podría tener una mejora con respecto a la que se tiene actualmente siempre y cuando se conserven las propiedades de Arblis en cuanto al manejo de cubos de datos.

El manejo de rangos con el que se cuenta es sobre tipo de datos numéricos, de tal forma que cualquier orden que se tenga que se pueda transformar a numérico podría ser usado en AC para su creación y solución de preguntas, sin embargo el manejo directo de tipo de datos como texto (cadenas de caracteres) está limitado a un soporte a través de su representación numérica.

La transformación empleada por medio de la compactación es monotónica, sin embargo se puede tener una compresión no monotónica pero mono-valuada que permita reordenar los valores que resultan de la función de compresión de tal forma que por medio de la permutación de los valores comprimidos se podría considerar como una función monótona. Por lo que para todos v_i y v_j tales que $v_i \leq v_j$ se tiene que tiene que $C_m(v_i) \leq C_m(v_j)$ o $C_m(v_i) > C_m(v_j)$ donde C_m es la compactación empleada por lo que no sería monotónica; si se aplica una función de orden sobre todos los valores compresos $C_m(v)$ tales que existe $C_m(v_k) \leq C_m(v_l)$ tal que $C_m(v_k)$ equivale a v_i y $C_m(v_l)$ a v_j de tal forma que se puede considerar como monótona. Desde luego que hay que guardar esta transformación en una tabla en memoria, para poder invertirla cuando se desee imprimir o desplegar los resultados.

Se responden preguntas de negocios útiles y la variedad de preguntas que se pueden contestar es amplia. Puede adaptarse a todo tipo de preguntas de negocios que trabajen sobre cubos de datos y a los análisis particulares sobre los cubos. Sin embargo actualmente se encuentra limitado a siete formas de realizar las preguntas.

La interfaz gráfica es generada dependiendo del número de dimensiones que se utilicen, por lo que el ingreso de datos es sencillo.

7.3 TRABAJO A FUTURO

- *Manejo de cadenas de caracteres.* El manejo de cadenas de caracteres al igual que los datos numéricos y fechas pueden tener un orden, este orden lexicográfico sería arbitrario pero consistente, de tal forma que una vez que se establezca el orden parcial se pueden aplicar las técnicas ya manejadas así como algunas otras que fueron descartadas

- *Apuntadores relativos.* Actualmente todos los apuntadores del arreglo 2 de la Ilustración 1 simbolizan la posición en la que se encuentra un elemento dentro del arreglo 1, sin embargo estos apuntadores podrían ser relativos si cada inicio de dimensión se reiniciara el conteo de las posiciones de tal forma que el tamaño de los apuntadores empleados en el arreglo 2 sería disminuido.

- *Uso de GPU.* Por una parte se tiene la aceleración por hardware usada principalmente en aplicaciones con una gran carga de gráficos, con lo que la aceleración por GPU le quita trabajo a la CPU, que lo hará más rápido obteniendo una mejora en el tiempo de respuesta. Y por otra parte se tiene que algunas tarjetas gráficas soportan la decodificación por hardware que podría ser utilizado en la transformación de los valores en el momento de la compactación, de tal forma que, así como se decodificar un vídeo por medio de un chip integrado se podría decodificar la compactación a su valor real o inicial

- *Ordenamiento de valores compresos no monótonos*. Se puede tener una compresión no monotónica pero mono-valuada que permita reordenar los valores que resultan de la función de compresión de tal forma que por medio de la permutación de los valores comprimidos se podría considerar como una función monótona.

- *Manejo de Jerarquías.* Dentro del manejo de los cubos de datos se pueden realizar operaciones que involucran el uso de jerarquías almacenadas comúnmente en tablas que ayudan en éste manejo, que podría reflejarse en la creación y uso de la estructura AC.

BIBLIOGRAFÍA

[Agrawal, 1995] Agrawal, R., Gupta, A., Sarawagi, S., "*Modeling Multidimensional Databases*", IBM Almaden Research Center, 650 Harry Road, San José, CA 95120; 1995.

[Applix, 2007] Applix Inc. "Applix TM1 Perspectives", Applix, U.S.,2007

[Berson, 1997] Berson, A., Smith, S. J., "Data warehousing, data mining, and OLAP", Mc Graw Hill, 1997.

[Codd, 1993] Codd E.F., Codd S.B. y Salley C.T. *Providing OLAP(On-line Analytical Processing) to User-Analysis: An IT Mandate. Hyperion Solutions Corporation*, 1993

[Connolly, 2005] Connolly Thonas, Begg Carolyn. "*Sistemas de bases de datos. Un enforque práctico para diseño, implementación y gestión"*. Pearson, España, 2005

[Fano,1949] Fano, R.M., "*The transmission of information***",** Technical Report No 65, Research Laboratory of Electronics, Mit. Cambridge, 1949.

[Gray, 1995] Gray, J., Bosworth, A., Layman, A., Pirahesh, H., "*Data Cube: A Relational Aggregation Operator Generalizing Group By, Cross-Tab, and Sub-Totals*", Technical Report MSR-TR-95-22, Microsoft Research, Advanced Technology Division, Microsoft Corporation, 1995.

[Gupta, 1995] Gupta, A., Murnick, I. S., "*Maintenance of Materialized Views: Problems, Techniques, and Applications*", IBM Almaden Research Center, AT&T Bell Laboratories, 1995.

[Han, 2001] Han, J., Kamber, M., "*Data Mining: concepts and techniques"*, Academic Press, Morgan Kaufmann Publishers, 2001.

[Harinarayan, 1996] Harinarayan V., Rajaraman A., Ullman J., "*Implementing Data Cubes Efficiently*", Stanford University, 1996.

[Henderson, 2000] Henderson Software, "*Inc. Applixware make it happened***",** Applix,U.S, 2000.

[Huffman, 1952] Huffman, David. **"***A Method for the Construction of Minimun Redundancy Codes*", Proceedings of the I.R.E, Massachusetts Institute of Technology, September 1952.

[IOS,1992] "*International Standard for Database Language SQL"*, document ISO/IEC 9075:1992, J. Melton (Ed.)

[Jung, 2003] Jung, K., "*Design and Implementation of Storage Manager in Main Memory Database",* System ALTIBASE", Real-Time Tech Lab, ALTIBASE Co. Seoul, Korea, 2003.

[Martínez, 2007] Martínez-Luna. G. L., Guzmán-Arenas, "*Latices y otras estructuras para acelerar búsquedas en minería de datos***"**, Centro de Investigación en Computación del I.P.N., Tesis de Doctorado, Julio-2007.

[Oracle Corporation, 2008] **"***Oracle Essbase Data Sheet*", Oracle Corporation, U.S.,2008

[Pérez, 2008] Pérez, César, "*Oracle 10g Administración y Análisis de Base de Datos*", Alfaomega, México, 2008.

[Pressman, 1982] Pressman, Roger**. "***Ingeniería de Software un Enfoque Práctico***",** McGraw Hill, México, 2005 (1982), 6ª edición.

[Rao, 1999] Rao, J., "*Cache Conscious Indexing for Decision-Support in Main Memory*", Columbia University, 1999.

[Raue, 2008] Raue, Kristian. Et al. "*Palo 2.5*", Jedox AG, Alemania, 2008

[Shanmugasundaram, 1999] Shanmugasundaram, J., Fayyad, U. M., Bradley, P. S., *"Compressed Data Cubes for OLAP Aggregate Query Approximation on Continuous Dimensions*", Microsoft Research, University of Wisconsin, Technical Report MSR-TR-99-13, 1999.

[Shannon, 1948] Shannon, C.E., "*A mathematical theory of communication"***,** Bell Sys Technology Jour, vol 27, July 1948

[Shukla, 1996] Shukla, P. M. et. al., "*Storage Estimation For Multidimentional Aggregates in The Presence of Hierarchies"***,** Computer Sciences Department University of Wisconsin Madison, 1996.

[Tremaine, 2001] Tremaine, R. B.; Franaszek, P. A.; Robinson, J. T.; Schulz, C. O.; Smith, T. B.; Wazlowski, M. E.; Bland, P. M.;"*Memory Expansion Technology*", IBM Journal of Research and Development, 2001

[Wayner, 2000] Wayner, Peter. "*Compression Algorithms for Real Programmers*", Morgan Kaufmann, Sn Francisco California, 2000

[Welch, 1984] Welch, Terry "A Technique for High-Performance Data Compression", Computer, June 1984

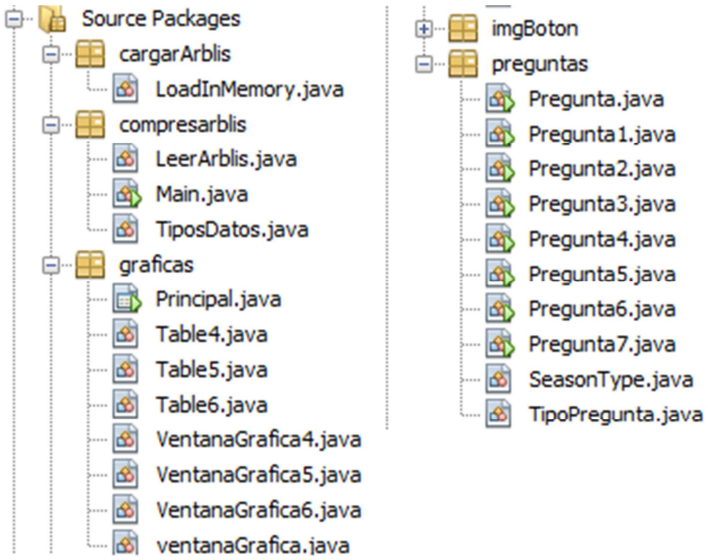
[Ziv, 1977] Ziv, Jacob. Lempel, Abraham. "A Universal Algorithm for Sequential Data Compressin", IEEE Transation on Information Theory, vol it-23, no 3, 1977.

ANEXO A. DOCUMENTACIÓN DEL CÓDIGO

Tabla de Contenido

INTRODUCCIÓN

Se cuenta con cuatro paquetes con sus respectivas clases:



A continuación se da la documentación de cada una de las clases.

cargarArblis

Class LoadInMemory

```
java.lang.Object
  └ cargarArblis.LoadInMemory
```

```
public class LoadInMemory
extends java.lang.Object
```

Clase que permite cargar a memoria principal: a) Arblis o b) Catálogo + AC

Field Summary

`static java.lang.String[]`	**aApuntaCatalogo** Arreglo 1 donde guarda apuntador a catalogo
`static java.lang.String[]`	**aApuntaDatoArblis** Arreglo 1 donde guarda apuntador a catalogo
`static java.lang.String[]`	**aApuntaEstructura** Arreglo 2 donde viene apuntadores en BC dentro de la estructura
`static java.lang.String[]`	**aApuntaEstructuraArblis** Arreglo 2 donde viene apuntadores en Arblis
`static java.lang.String[][]`	**aCatalogo** Matriz de numDim por el num de datos en cada uno
`static TiposDatos[]`	**aDataType** Arreglo que contienen tipos de datos por dim
`static int[]`	**arrayTam** Arreglo donde se guardan los tamaños de las dims (finales)
`static java.lang.String[]`	**arrayTamString** Arreglo donde se guardan los tamaños de las dims (finales) como cadenas
`static java.lang.String`	**filePath** Ruta de out
`static int`	**numDimensiones** numero de Dimensiones (entero)

Constructor Summary

LoadInMemory`(java.lang.String filePath, int numDim)` Constructor LoadInMemory

Method Summary

`boolean`	**cargarArblis**`()` Carga a memoria Arblis tradicional
`boolean`	**DosArreglos**`()` Carga a Memoria AC que comprende: dos arrglos más catálogo
`java.lang.String[]`	**getaApuntaCatalogo**`()` Método get de aApuntaCatálogo
`java.lang.String[]`	**getaApuntaDatoArblis**`()` Obtiene Arreglo 1 donde guarda apuntador a catalogo
`java.lang.String[]`	**getaApuntaEstructura**`()` Regresa Arreglo 2 donde viene apuntadores en BC dentro de la estructura
`java.lang.String[]`	**getaApuntaEstructuraArblis**`()` Regresa Arreglo 2 donde viene apuntadores en BC dentro de la estructura
`java.lang.String[][]`	**getaCatalogo**`()`

	Regresa Matriz de numDim por el num de datos en cada uno donde se guarda los valores necesarios para la compresión y descompresión
TiposDatos[]	**getaDataType**() Método get de Data Type
int[]	**getArrayTam**() Regresa Arreglo donde se guardan los tamaños de las n dims
java.lang.String[]	**getArrayTamString**() Regresa Arreglo donde se guardan los tamaños de las n dims como cadenas
static java.lang.String	**getFilePath**() Método get de Path Obtiene la ruta del archivo out
static int	**getNumDimensiones**() Método get de numDimensiones
static void	**setFilePath**(java.lang.String filePath) Método set de FilePath Determine la ruta del archivo out
static void	**setNumDimensiones**(int numDimensiones) Método set de FilePath Determine la ruta del archivo out

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

numDimensiones
public static int **numDimensiones**
numero de Dimensiones (entero)

---

arrayTam
public static int[] **arrayTam**
Arreglo donde se guardan los tamaños de las dims (finales)

---

aCatalogo
public static java.lang.String[][] **aCatalogo**
Matriz de numDim por el num de datos en cada uno

---

aApuntaEstructura
public static java.lang.String[] **aApuntaEstructura**
Arreglo 2 donde viene apuntadores en BC dentro de la estructura

---

aApuntaCatalogo
public static java.lang.String[] **aApuntaCatalogo**
Arreglo 1 donde guarda apuntador a catalogo

---

aDataType
public static TiposDatos[] **aDataType**
Arreglo que contienen tipos de datos por dim

---

arrayTamString
public static java.lang.String[] **arrayTamString**
Arreglo donde se guardan los tamaños de las dims (finales) como cadenas

---

aApuntaEstructuraArblis
public static java.lang.String[] **aApuntaEstructuraArblis**
Arreglo 2 donde viene apuntadores en Arblis

---

aApuntaDatoArblis
public static java.lang.String[] **aApuntaDatoArblis**
Arreglo 1 donde guarda apuntador a catalogo

---

filePath
`public static java.lang.String filePath`

Ruta de out

Constructor Detail

LoadInMemory
```
public LoadInMemory(java.lang.String filePath,
                    int numDim)
```
Constructor LoadInMemory

Parameters:

`filePath` - `String` Archivo con su dirección con extensión out que es el formato de la estructura Arblis

`numDim` - `int` Número de dimensiones

Method Detail

getFilePath
`public static java.lang.String getFilePath()`

Método get de Path Obtiene la ruta del archivo out

Returns:

`String` filePath

---

setFilePath
`public static void setFilePath(java.lang.String filePath)`

Método set de FilePath Determine la ruta del archivo out

Parameters:

`filePath` - `String` ruta

---

getNumDimensiones
`public static int getNumDimensiones()`

Método get de numDimensiones

Returns:

`int` número de dimensiones

---

setNumDimensiones
`public static void setNumDimensiones(int numDimensiones)`

Método set de FilePath Determine la ruta del archivo out

Parameters:

`numDimensiones` - `int` recibe el número de dimensiones necesarias

---

getaDataType
`public TiposDatos[] getaDataType()`

Método get de Data Type

Returns:

`TipoDatos[]` arreglo de tipos de datos en cada una de las dims

---

getaApuntaCatalogo
`public java.lang.String[] getaApuntaCatalogo()`

Método get de aApuntaCatálogo

Returns:

`String[]` regresa Arreglo 1 donde guarda apuntador a catalogo

---

getaApuntaDatoArblis
`public java.lang.String[] getaApuntaDatoArblis()`

Obtiene Arreglo 1 donde guarda apuntador a catalogo

Returns:

`String[]` Arreglo 1 donde guarda apuntador a catalogo

---

getaApuntaEstructuraArblis
`public java.lang.String[] getaApuntaEstructuraArblis()`

Regresa Arreglo 2 donde viene apuntadores en BC dentro de la estructura

Returns:

String [] Arreglo 2 donde viene apuntadores en BC dentro de la estructura

---

getaCatalogo

```
public java.lang.String[][] getaCatalogo()
```

Regresa Matriz de numDim por el num de datos en cada uno donde se guarda los valores necesarios para la compresión y descompresión

Returns:

`String[][]` Matriz de numDim por el num de datos en cada uno

---

getArrayTam

```
public int[] getArrayTam()
```

Regresa Arreglo donde se guardan los tamaños de las n dims

Returns:

`int[]` Arreglo donde se guardan los tamaños de las n dims

---

getArrayTamString

```
public java.lang.String[] getArrayTamString()
```

Regresa Arreglo donde se guardan los tamaños de las n dims como cadenas

Returns:

`String[]` Arreglo donde se guardan los tamaños de las n dims como cadenas

---

getaApuntaEstructura

```
public java.lang.String[] getaApuntaEstructura()
```

Regresa Arreglo 2 donde viene apuntadores en BC dentro de la estructura

Returns:

`String[]` Arreglo 2 donde viene apuntadores en BC dentro de la estructura

---

cargarArblis

```
public boolean cargarArblis()
                      throws java.io.IOException
```

Carga a memoria Arblis tradicional

Returns:

`String` confirmación de haber cargado a memoria principal Arblis original

Throws:

`java.io.IOException`

DosArreglos

```
public boolean DosArreglos()
                    throws java.io.IOException
```

Carga a Memoria AC que comprende: dos arrglos más catálogo

Returns:

`boolean` confirmación de haber cargado a memoria principal los catálogos y AC

Throws:

`java.io.IOException`

compresarblis

Class LeerArblis

```
java.lang.Object
  └ compresarblis.LeerArblis
```

```
public class LeerArblis
extends java.lang.Object
```

Clase que realiza la transformación de Arblis a AC (Apuntadores a Catálogo), de tal forma que crea los catálogos correspondientes así como la estructura compresa.

Constructor Summary

LeerArblis()

Method Summary

static boolean	**createCatalogs**(java.lang.String filePath, int numDim) Crea los catálogos para el número de dimensiones (nnumDim) dada dentro de un archivo.
static boolean	**isBisiestro**(int year) Saber si un año es bisiestro
static boolean	**obtenerTiDatoFecha**(java.lang.String cadena) Evalúa la expresión cadena para determinar si el tipo de dato de éste elemento es una fecha del tipo AAAA-MM-DD
static boolean	**writeFileAsBC**(java.lang.String filePath, int numDim) Crea la estructura AC y la guarda en disco, recibe el archivo de Arblis original y lo guarda en disco con la extensión outbc.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

LeerArblis

```
public LeerArblis()
```

Method Detail

createCatalogs

```
public static boolean createCatalogs(java.lang.String filePath,
                                      int numDim)
                               throws java.io.IOException
```

Crea los catálogos para el número de dimensiones (nnumDim) dada dentro de un archivo. El nombre del archivo que se recibe es el del archivo original creando la cantidad de numDim catálogos diferentes, ya que la compresión es diferente para cada dimensión.

Parameters:

`filePath` - ruta y nombre de archivo original de Arblis

`numDim` - recibe el número de dimensiones

Returns:

para ver si la creación fue exitosa o no

Throws:

`java.io.IOException`

---

writeFileAsBC

```
public static boolean writeFileAsBC(java.lang.String filePath,
                                    int numDim)
                             throws java.io.IOException
```

Crea la estructura AC y la guarda en disco, recibe el archivo de Arblis original y lo guarda en disco con la extensión outbc. Este método debe ser mandado a llamar si existen los catálogos necesarios para su creación de tal forma que si no existen lanzará una excepción ya que no puede ser creada sin la existencia de éste

Parameters:
`filePath` - nombre y ruta de Arblis
`numDim` - número de dimensiones
Returns:
regresa un booleano de confirmación
Throws:
`java.io.IOException` - debe ser atrapada

---

obtenerTiDatoFecha
`public static boolean obtenerTiDatoFecha(java.lang.String cadena)`

Evalúa la expresión cadena para determinar si el tipo de dato de éste elemento es una fecha del tipo AAAA-MM-DD
Parameters:
`cadena` - cadena a ser evaluada
Returns:
regresa verdadero si el tipo de dato es una fecha

---

isBisiestro
`public static boolean isBisiestro(int year)`

Saber si un año es bisiestro
Parameters:
`year` - año a ser verificado
Returns:
verdadero si es bisiesto y falso si no es bisiesto

---

compresarblis

Class Main

```
java.lang.Object
  └ compresarblis.Main
```

```
public class Main
extends java.lang.Object
```

Clase Principal

Constructor Summary

Main()

Method Summary

static void	**main**(java.lang.String[] args) Método principal

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Main

```
public Main()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

Método principal

Parameters:

`args` - the command line arguments

compresarblis

Enum TiposDatos

```
java.lang.Object
  └ java.lang.Enum<TiposDatos>
      └ compresarblis.TiposDatos
```

All Implemented Interfaces:

java.io.Serializable, java.lang.Comparable<TiposDatos>

```
public enum TiposDatos
extends java.lang.Enum<TiposDatos>
```

Enumeración que contiene los tres tipos de datos soportados

Enum Constant Summary

Cadena Cadena: si el tipo de dato no es ni entero nni fecha
Entero Entero: si el tipo de dato es numérico
Fecha Fecha: si el tipo de dato es una fecha en el formato aaaa/MM/dd

Method Summary

static TiposDatos	**valueOf**(java.lang.String name) Returns the enum constant of this type with the specified name.
static TiposDatos[]	**values**() Returns an array containing the constants of this enum type, in the order they are declared.

Methods inherited from class java.lang.Enum
clone, compareTo, equals, finalize, getDeclaringClass, hashCode, name, ordinal, toString, valueOf

Methods inherited from class java.lang.Object
getClass, notify, notifyAll, wait, wait, wait

Enum Constant Detail

Entero
public static final TiposDatos **Entero**

Entero: si el tipo de dato es numérico

Fecha
public static final TiposDatos **Fecha**

Fecha: si el tipo de dato es una fecha en el formato aaaa/MM/dd

---

Cadena
public static final TiposDatos **Cadena**

Cadena: si el tipo de dato no es ni entero nni fecha

Method Detail

values
public static TiposDatos[] **values**()

Returns an array containing the constants of this enum type, in the order they are declared. This method may be used to iterate over the constants as follows:

```
for (TiposDatos c : TiposDatos.values())
    System.out.println(c);
```

Returns:

an array containing the constants of this enum type, in the order they are declared

---

valueOf
public static TiposDatos **valueOf**(java.lang.String name)

Returns the enum constant of this type with the specified name. The string must match *exactly* an identifier used to declare an enum constant in this type. (Extraneous whitespace characters are not permitted.)

Parameters:

name - the name of the enum constant to be returned.

Returns:

the enum constant with the specified name

Throws:

java.lang.IllegalArgumentException - if this enum type has no constant with the specified name

java.lang.NullPointerException - if the argument is null

---

graficas

Class Principal

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ java.awt.Window
              └ java.awt.Frame
                  └ javax.swing.JFrame
                      └ graficas.Principal
```

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

---

```
public class Principal
extends javax.swing.JFrame
```

Clase que contiene la interfaz gráfica´principal

See Also:

Serialized Form

Constructor Summary

`Principal()`	Crea uun nuevo de Principal

Method Summary

int	`getNumDimFig()` Obtiene el número de dimensiones
void	`getNumDimsFromMessage()` Obtiene el número de dimensiones ingresadas por el usuario
SeasonType	`getSeasonTypeFromCombo()` Obtiene el tipo de periodo del combo de la pregunta 7
SeasonType	`getSeasonTypeFromCombo6()` Obtiene el tipo de periodo del combo de la pregunta 6
java.lang.String	`selectArchivo()` Seleccion de archivo, generalmente se debe seleccionar el archivo .out
void	`setNumDimFig(int numDimFig)` Se asigna el valor del número de dimensiones

Constructor Detail

Principal

```
public Principal()
```

Crea uun nuevo de Principal

Method Detail

getNumDimFig

```
public int getNumDimFig()
```

Obtiene el número de dimensiones

Returns:

la contidad de dimensiones empleadas

---

setNumDimFig

```
public void setNumDimFig(int numDimFig)
```

Se asigna el valor del número de dimensiones

Parameters:
`numDimFig` - numero de dimensiones

---

getSeasonTypeFromCombo
`public SeasonType getSeasonTypeFromCombo()`
Obtiene el tipo de periodo del combo de la pregunta 7
Returns:
SeasonType temporada seleccionada

---

getSeasonTypeFromCombo6
`public SeasonType getSeasonTypeFromCombo6()`
Obtiene el tipo de periodo del combo de la pregunta 6
Returns:
SeasonType temporada seleccionada

---

getNumDimsFromMessage
`public void getNumDimsFromMessage()`
Obtiene el número de dimensiones ingresadas por el usuario

---

selectArchivo
`public java.lang.String selectArchivo()`
Seleccion de archivo, generalmente se debe seleccionar el archivo .out

---

preguntas

Class Pregunta

```
java.lang.Object
  └ preguntas.Pregunta
```

Direct Known Subclasses:

Pregunta1, Pregunta2, Pregunta3, Pregunta4, Pregunta5, Pregunta6, Pregunta7

```
public class Pregunta
extends java.lang.Object
```

Field Summary

`java.lang.String[]`	**aDatosDimsCompres** Datos a buscar en las dims
TipoPregunta	**tiPregunta** Tipo de pregunta de enum TipoPregunta

Constructor Summary

Pregunta()

Method Summary

`java.lang.String`	**findDateFromIndex**(java.lang.String codigo, java.lang.String fechaIncio) Encuentra la fecha y la devuelve en el formato yyyy-MM-dd a partir del código y la fecha de inicio obtenida
`java.lang.String`	**findDateIndex**(java.lang.String inicio, java.lang.String findString) Encuentra el índice correspondiente dependeindo de la fecha de inicio y la fecha buscada
`static void`	**main**(java.lang.String[] arg)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

tiPregunta

```
public TipoPregunta tiPregunta
```

Tipo de pregunta de enum TipoPregunta

---

aDatosDimsCompres

```
public java.lang.String[] aDatosDimsCompres
```

Datos a buscar en las dims

Constructor Detail

Pregunta

```
public Pregunta()
```

Method Detail

findDateIndex

```
public java.lang.String findDateIndex(java.lang.String inicio,
                                      java.lang.String findString)
```

Encuentra el índice correspondiente dependeindo de la fecha de inicio y la fecha buscada

Parameters:

`inicio` - la fecha inicial aaaa/MM/dd

`findString` - cadena buscada en el formato aaaa/MM/dd

Returns:
el valor compreso de findString

---

findDateFromIndex

```
public java.lang.String findDateFromIndex(java.lang.String codigo,
                                          java.lang.String fechaIncio)
```

Encuentra la fecha y la devuelve en el formato yyyy-MM-dd a partir del código y la fecha de inicio obtenida

Parameters:

`codigo` - fecha compresa

`fechaInicio` - valor de la fecha inicial en el formato yyyy-MM-dd

Returns:

valor descompreso de código

---

preguntas

Class Pregunta1

```
java.lang.Object
  └ preguntas.Pregunta
      └ preguntas.Pregunta1
```

```
public class Pregunta1
extends Pregunta
```

Pregunta 1 puntual:
Clase que realiza los calculos necesario para contestar una pregunta puntual. Esta pregunta se define sobre un solo cubo C1. Donde todos los elementos que definen al cubo son un único valor. De tal forma que C1 es un cubo de datos de un solo punto (valor).
Algoritmo
1. Mientras existan dimensiones
a. Obtener el valor compreso a partir del dato dado
2. Fin del mientras
3. Mientras existan datos en el Arreglo 1
a. Se obtiene el inicio y fin para la primera dimensión
b. Para i=0 hasta el número total de dimensiones
aa. Para j=inicio hasta que j sea fin o Arreglo 1 es igual al dato compreso buscado
I. Comparar si el elemento en Arreglo 1 es igual al dato compreso buscado
II. Si verdadero
A. Se obtienen inicio y fin para el desplazamiento de la siguiente dimensión
III.Fin del si
bb. Fin del ciclo
c. Fin del ciclo
4. Si se llegó al fin y no se encontró regresa vacío
5. Regresa el agregado

Field Summary

Fields inherited from class preguntas.Pregunta
aDatosDimsCompres, tiPregunta

Constructor Summary

Pregunta1(int numDimensiones) Constructor de la clase pregunta 1

Method Summary

void	**getValuesCompres**(java.lang.String[] aDatosDims) Obtiene los valores correspondientes en la estructuraAC, para realizar la búsqueda en la estructura compresa.
static void	**main**(java.lang.String[] args)
java.lang.String	**vAnswerQueryP**(java.lang.String[] aDatosDims) Responde una pregunta puntual en la estructura compresa.
java.lang.String	**vAnswerQuerySinP**(java.lang.String[] aDatosDims) Responde una pregunta puntual en la estructura Arblis.

Methods inherited from class preguntas.Pregunta
findDateFromIndex, findDateIndex

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Pregunta1

```
public Pregunta1(int numDimensiones)
```

Constructor de la clase pregunta 1

Parameters:

`numDimensions` - cantidad de dimensiones usadas

Method Detail

vAnswerQueryP

```
public java.lang.String vAnswerQueryP(java.lang.String[] aDatosDims)
```

Responde una pregunta puntual en la estructura compresa. Recibe los datos de la pregunta puntual. Tomando en cuenta que se tendran numDim valores

Parameters:

`aDatosDims` - valores para contestar la pregunta, la cantidad de ellos es proporcional al número de dimensiones, de tal forma que si son usadas 5 dimensioones los datos deben de ser 5

---

vAnswerQuerySinP

```
public java.lang.String vAnswerQuerySinP(java.lang.String[] aDatosDims)
```

Responde una pregunta puntual en la estructura Arblis. Recibe los datos de la pregunta puntual. Tomando en cuenta que se tendran numDim valores

Parameters:

`aDatosDims` - valores para contestar la pregunta, la cantidad de ellos es proporcional al número de dimensiones, de tal forma que si son usadas 5 dimensioones los datos deben de ser 5

---

getValuesCompres

```
public void getValuesCompres(java.lang.String[] aDatosDims)
```

Obtiene los valores correspondientes en la estructuraAC, para realizar la búsqueda en la estructura compresa.

Parameters:

`aDatosDims` - valores originales que serán compresos, la cantidad de ellos es proporcional al número de dimensiones, de tal forma que si son usadas 5 dimensioones los datos deben de ser 5

---

preguntas

Class Pregunta2

```
java.lang.Object
  └ preguntas.Pregunta
      └ preguntas.Pregunta2
```

```
public class Pregunta2
extends Pregunta
```

Pregunta 2 de Rango:
Se define un rango en al menos uno de las dimensiones, De tal forma que podría no constar de un único valor, sino de varios, los cuales son contados y su agregado es sumado. Clase que realiza los calculos necesario para contestar una pregunta de rango

Algoritmo

1. Obtener valores del rango inicial y final para la primera y segunda dimensión para la primera iteracion
2. Obtener valor de inicio de un contador para cada una de las n dimensiones
3. Mientras (contador[0] < rango final[0] y k!=n) entonces
a. Iniciar k en 1
b. Mientras (contador[0] < rango final[0])
aa. Mientras(contador[n-(k+1)]
I.Incrementar en uno el contador[n-(k+1)]
bb.Fin Mientras
cc. Rango inicial [n-k]=arreglo 2 en contador[n-(k+1)]
dd. Rango final[n-k]=arreglo 2 en contador[n-(k+1)+1]
ee. Contadot[n-k]=rango inicial [n-k]
ff. Descomprimir valores en las posiciones dadas por el contador
gg. Responder puntual para esos valores
hh. Sumar resultados a sum y cont
c. Fin Mientras
d. Contador[n-1]++
4. Fin Mientras

Field Summary

long	**count** Cantidad de los valores que se encuentren en rango
long	**sum** Acumulado de la suma de los valores que se encuentren en rango

Fields inherited from class preguntas.Pregunta

aDatosDimsCompres, tiPregunta

Constructor Summary

Pregunta2(int numDimensiones)
Constructor para la pregunta2: pregunta de rango

Method Summary

java.lang.String	**descompresOne**(java.lang.String aDatos, int dim) Regresa el valor descompreso de aDato en la dimension dim
long	**getCount**() Obtiene la cantidad de valores que se encuentran dentro del rango
long	**getSum**() Obtiene la contidad sumada acumulativamente de los valores que se encuentran dentro del rango

void	**getValuesCompres**(java.lang.String[] aDatosDims) Obtiene los valores correspondientes en la estructuraAC Los coloca en aDatosDimsCompres en la misma posisción para numDimensiones por 2 ya que recibe rangos.
void	**getValuesCompresRango**(java.lang.String[] aDatosDims) Obtiene los valores correspondientes en la estructuraAC Los coloca en aDatosDimsCompres en la misma posisción para numDimensiones por 2 ya que recibe rangos.
java.lang.String[]	**getValuesDescompres**(java.lang.String[] aDatos) Descomprime los valores
java.lang.String	**vAnswerQueryP**(java.lang.String[] aDatosDims)
boolean	**vAnswerQueryR**(java.lang.String[] aDatosDims) Responde una pregunta rango en la estructura compresa.
boolean	**vAnswerQuerySinR**(java.lang.String[] aDatosDims) Responde una pregunta rango en la estructura Arblis original.

Methods inherited from class preguntas.Pregunta

findDateFromIndex, findDateIndex

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

sum

```
public long sum
```

Acumulado de la suma de los valores que se encuentren en rango

---

count

```
public long count
```

Cantidad de los valores que se encuentren en rango

Constructor Detail

Pregunta2

```
public Pregunta2(int numDimensiones)
```

Constructor para la pregunta2: pregunta de rango

Parameters:

`numDimensiones` - número de dimensiones

Method Detail

getCount

```
public long getCount()
```

Obtiene la cantidad de valores que se encuentran dentro del rango

Returns:

long cout

---

getSum

```
public long getSum()
```

Obtiene la contidad sumada acumulativamente de los valores que se encuentran dentro del rango

Returns:

long sum

---

vAnswerQueryP

```
public java.lang.String vAnswerQueryP(java.lang.String[] aDatosDims)
```

---

vAnswerQueryR

```
public boolean vAnswerQueryR(java.lang.String[] aDatosDims)
                     throws java.text.ParseException
```

Responde una pregunta rango en la estructura compresa. Recibe los datos de la pregunta de rango. Tomando en cuenta que se tendran el doble de numDim valores. Regresa false si no encontro

Parameters:

`aDatosDims` - arreglo donde vienen los valores inicial y final de cada dimensión. aDatosDims[n]=val rango inicio de dimension n. aDatosDims[n+numDimensiones]= val rango fin de dimension n.

Returns:

confirmación de que se realizó con éxito

Throws:

`java.text.ParseException` - errores en conversión

---

vAnswerQuerySinR

```
public boolean vAnswerQuerySinR(java.lang.String[] aDatosDims)
                         throws java.text.ParseException
```

Responde una pregunta rango en la estructura Arblis original. Recibe los datos de la pregunta de rango. Tomando en cuenta que se tendran el doble de numDim valores. Regresa false si no encontro

Parameters:

`aDatosDims` - arreglo donde vienen los valores inicial y final de cada dimensión. aDatosDims[n]=val rango inicio de dimension n. aDatosDims[n+numDimensiones]= val rango fin de dimension n.

Returns:

confirmación de que se realizó con éxito

Throws:

`java.text.ParseException` - errores en conversión

---

getValuesCompresRango

```
public void getValuesCompresRango(java.lang.String[] aDatosDims)
```

Obtiene los valores correspondientes en la estructuraAC Los coloca en aDatosDimsCompres en la misma posisción para numDimensiones por 2 ya que recibe rangos.

Parameters:

`aDatosDims` - valores a comprimir (son dos cubos de datos por lo que el número de datos recibidos es del doble de la cantidad de dimensiones manejadas)

---

getValuesCompres

```
public void getValuesCompres(java.lang.String[] aDatosDims)
```

Obtiene los valores correspondientes en la estructuraAC Los coloca en aDatosDimsCompres en la misma posisción para numDimensiones por 2 ya que recibe rangos.

Parameters:

`aDatosDims` - valores a comprimir del número de datos equivalente dimensiones a las dims manejadas

---

getValuesDescompres

```
public java.lang.String[] getValuesDescompres(java.lang.String[] aDatos)
```

Descomprime los valores

Parameters:

`aDatos` - Del largo del num de dims

Returns:

valores originales

---

descompresOne

```
public java.lang.String descompresOne(java.lang.String aDatos,
                                      int dim)
```

Regresa el valor descompreso de aDato en la dimension dim

Parameters:

`aDatos` - dato a descomprimir

`dim` - dimensión a la que pertenece

Returns:

valor descompreso de aDatos

preguntas

Class Pregunta3

```
java.lang.Object
  └ preguntas.Pregunta
      └ preguntas.Pregunta3
```

```
public class Pregunta3
extends Pregunta
```

Pregunta 3 de eficiencia global:
Esta Pregunta se define sobre dos cubos por lo que se reciben dos conjuntos de rangos que serán evaluados con la pregunta de rango individualmente y posteriormente se calculara la eficiencia de éstos
Algoritmo
1. Comprimir valores de rango del Cubo 1
2. Comprimir valores de rango del Cubo 2
3. Obtener la suma y el contador del Cubo 1 a partir de la llamada a la pregunta de rango para los valores del Cubo 1
4. Obtener la suma y el contador del Cubo 2 a partir de la llamada a la pregunta de rango para los valores del Cubo 2
5. Obtener la eficiencia a partir de la Ecuación
6. Regresar el valor de eficiencia

Field Summary

Fields inherited from class preguntas.Pregunta
aDatosDimsCompres, tiPregunta

Constructor Summary

Pregunta3(int numDimensiones) Constructor de la clase de la pregunta 3	

Method Summary

void	**answerQ3**(java.lang.String[] aDatosDims, java.lang.String[] aDatosDims2) Método para contestar la pregunta 3 descrito en la clase
long	**getCount1**() Obtener el contador del primer cubo de datos
long	**getCount2**() Obtener el contador del segundo cubo de datos
double	**getEficiencia**() Obtiene el valor de la eficiencia
long	**getSum1**() Obtener la suma del primer cubo de datos
long	**getSum2**() Obtener la suma del segundo cubo de datos
void	**setCount1**(long count1) Asignar valor al contador del primer cubo de datos
void	**setCount2**(long count2) Asignar valor al contador del segundo cubo de datos
void	**setEficiencia**(double eficiencia) Asigna valor a la eficiencia
void	**setSum1**(long sum1) Asignar la suma del primer cubo de datos
void	**setSum2**(long sum2)

	Asignar la suma del segundo cubo de datos

Methods inherited from class preguntas.Pregunta
findDateFromIndex, findDateIndex

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Pregunta3
`public Pregunta3(int numDimensiones)`

Constructor de la clase de la pregunta 3

Parameters:

`numDimenciones` - entero para el numero de dimensiones

Method Detail

getCount1
`public long getCount1()`

Obtener el contador del primer cubo de datos

Returns:

long regresa el contador

---

setCount1
`public void setCount1(long count1)`

Asignar valor al contador del primer cubo de datos

Parameters:

`count1` - valor para el contador

---

getCount2
`public long getCount2()`

Obtener el contador del segundo cubo de datos

Returns:

long regresa el contador

---

setCount2
`public void setCount2(long count2)`

Asignar valor al contador del segundo cubo de datos

Parameters:

`count1` - valor para el contador

---

getEficiencia
`public double getEficiencia()`

Obtiene el valor de la eficiencia

Returns:

double valor de la eficiencia

---

setEficiencia
`public void setEficiencia(double eficiencia)`

Asigna valor a la eficiencia

Parameters:

`eficiencia` - double que contiene la eficiencia

---

getSum1
`public long getSum1()`

Obtener la suma del primer cubo de datos

Returns:

long regresa el contador

---

setSum1

```
public void setSum1(long sum1)
```

Asignar la suma del primer cubo de datos

Parameters:

`sum1` - long de la suma para el 1er cubo

---

getSum2

```
public long getSum2()
```

Obtener la suma del segundo cubo de datos

Returns:

long regresa el contador

---

setSum2

```
public void setSum2(long sum2)
```

Asignar la suma del segundo cubo de datos

Parameters:

`sum1` - long de la suma para el 2o cubo

---

answerQ3

```
public void answerQ3(java.lang.String[] aDatosDims,
                     java.lang.String[] aDatosDims2)
```

Método para contestar la pregunta 3 descrito en la clase

Parameters:

`aDatosDims` - datos que establecen los rangos del primer cubo de datos

`aDatosDims2[]` - datos que establecen los rangos del segundo cubo de datos

---

preguntas

Class Pregunta4

```
java.lang.Object
  └ preguntas.Pregunta
      └ preguntas.Pregunta4
```

```
public class Pregunta4
extends Pregunta
```

Pregunta 4 de eficiencia grupal:
En esta pregunta se define la dimensión sobre la que se desea calcular la eficiencia de los elementos. Se define el número de elementos k que se desean encontrar. La primera búsqueda es sobre el Cubo 1 que devolverá los k mejores/peores elementos, los cuales serán buscados en el Cubo 2 para comparar en comportamiento de los k elementos en 2 rangos diferentes

Algoritmo
1. Obtener todos los valores posibles de la dimensión sobre la que se desea la comparación
2. Comprimir valores de rango del Cubo 1
3. Mientras existan valores posibles, i
a. Formar la consulta con el i-esimo valor sustituido a los datos a consultar
b. Responder con rango los datos a consultar formados
c. Si i es menor al número de mejores
aa. Guardar el valor del dato
bb. Guardar la Suma en su primer cubo
d.Si no
aa. Buscar el menor/mayor
bb. Sustituir en éste el valor del dato
cc. Sustituir la suma en su primer cubo
e. Fin del si
4. Fin del Mientras
5. Para i=0, mientras sea menor de k
a. Sustituir en los datos a consultar el dato i
b. Hacer la pregunta por rango con este dato a consultar
c. Guardar la suma en su segundo cubo
6. Fin del mientras
7. Para i=0, mientras sea menor de k
a. Calcular la eficiencia para C1 y C2 de i según Ecuación 9 u guardarlo
8. Fin del mientras
9. Regresar los valores con su eficiencia para su gráfica.

Field Summary

Fields inherited from class preguntas.Pregunta
aDatosDimsCompres, tiPregunta

Constructor Summary

Pregunta4(int numDimensiones) Constructor para la pregunta 4	

Method Summary

long	**findHighest**(java.lang.String[] aSums, long value) Sustituir el arreglo en busca de los n mayores
long	**findSmallest**(java.lang.String[] aSums, long value) Encuentra el valor más pequeño que será desplazado por uno mayor en caso de que value sea mayor los los m mejores

int	**getDim_kept**() Obtiene el número de la dimensión que permanece
int	**getNum_best**() Obtiene el número de mejores que el usuario requiera
java.lang.String[]	**getNumDiferentValues**(int[] questArrayTam, java.lang.String[][] aQCatalogo) Obtiene los diversos valores para la dimensión que se mantiene
java.lang.String	**getValueDescompres**(java.lang.String aDato) Descomprime un valor de la dimensión que se conserva
java.lang.String[][]	**losNMejores**(java.lang.String[] aDatosDims, java.lang.String[] aDatosDims2, int numMejores) Obtiene los N mejores en determinado.
void	**setDim_kept**(int dim_kept) Establece el número de la dimensión comenzando por cero, el cual es dado regularmente por el usuario
void	**setNum_best**(int num_best) Establece el número de mejores valores, el cual es dado regularmente por el usuario

Methods inherited from class preguntas.Pregunta
findDateFromIndex, findDateIndex

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Pregunta4
public **Pregunta4**(int numDimensiones)
Constructor para la pregunta 4

Method Detail

getNum_best
public int **getNum_best**()
Obtiene el número de mejores que el usuario requiera
Returns:
int regresa el valor de la cantidad de mejores/peores

---

setNum_best
public void **setNum_best**(int num_best)
Establece el número de mejores valores, el cual es dado regularmente por el usuario
Parameters:
num_best - cantidad de mejores

---

getDim_kept
public int **getDim_kept**()
Obtiene el número de la dimensión que permanece
Returns:
int regresa el valor de la dimension pivote

---

setDim_kept
public void **setDim_kept**(int dim_kept)
Establece el número de la dimensión comenzando por cero, el cual es dado regularmente por el usuario
Parameters:
dim_kept - dimension que se conserva

---

losNMejores
public java.lang.String[][] **losNMejores**(java.lang.String[] aDatosDims,
java.lang.String[] aDatosDims2,

```
                                        int numMejores)
```

Obtiene los N mejores en determinado. Constestando la pregunta con el algoritmo antes descrito.

Parameters:

`aDatosDims` - String [] del rango de valores del primer cubo de datos

`aDatosDims2` - String [] del rango de valores del segundo cubo de datos

`numMejores` - int establecera la cantidad de mejores

Returns:

String[][] Es una matriz de 4 por la cantidad de mejores/peores deseados

1. Valor del mejor dato
2. Suma del primer cubo
3. Suma del segundo cubo
4. Eficiencia calculada a partir de las sumas anteriores

---

getNumDiferentValues

```
public java.lang.String[] getNumDiferentValues(int[] questArrayTam,
                                               java.lang.String[][] aQCatalogo)
```

Obtiene los diversos valores para la dimensión que se mantiene

Parameters:

`questArrayTam` - int[] tamaño de cada una de las dimensiones

`aQCatalogo` - String [][] catálogo

Returns:

String[] cadena de String que contiene los diversos valores a intercambiarse en la dimensión que se mantiene, que servirá para generar los sub-cubos del cubo 1 y 2

---

getValueDescompres

```
public java.lang.String getValueDescompres(java.lang.String aDato)
```

Descomprime un valor de la dimensión que se conserva

Parameters:

`aDato` - String

Returns:

String aDato descomprimido

---

findSmallest

```
public long findSmallest(java.lang.String[] aSums,
                         long value)
```

Encuentra el valor más pequeño que será desplazado por uno mayor en caso de que value sea mayor los los m mejores

Parameters:

`aSums` - String[] conjunto de elementos que contiene la suma de los n mejores

`value` - long del dato a ser evaluado dentro de aSums para ver si desplazará a algun dato o no

Returns:

long regresa el número del menor

---

findHighest

```
public long findHighest(java.lang.String[] aSums,
                        long value)
```

Sustituir el arreglo en busca de los n mayores

Parameters:

`aSums` - String[] conjunto de elementos que contiene la suma de los n mejores

`value` - long del dato a ser evaluado dentro de aSums para ver si desplazará a algun dato o no

Returns:

long regresa el número del mayor

---

preguntas

Class Pregunta5

```
java.lang.Object
  └ preguntas.Pregunta
      └ preguntas.Pregunta5
```

```
public class Pregunta5
extends Pregunta
```

Pregunta 5 sobre conservación y pérdida:
En esta pregunta se define la dimensión sobre la que se desea calcular la eficiencia de los elementos. Se define el número de elementos k que se desean encontrar. Se repite la operación para un Cubo 2 y se verifica cuales se conservaron y cuales se perdieron al comparar ambos cubos.
Algoritmo
1. Se obtienen los k mejores en el Cubo 1
2. Comprimir valores de rango del Cubo 1
a. Obtener todos los valores posibles de la dimensión sobre la que se desea la comparación
b. Mientras existan valores posibles, i
aa. Formar la consulta con el i-esimo valor sustituido a los datos a consultar
bb. Responder con rango los datos a consultar formados
cc.Si i es menor al número de mejores
I. Guardar el valor del dato
II. Guardar la Suma en su primer cubo
dd. Si no
I. Buscar el menor/mayor
II. Sustituir en éste el valor del dato
III. Sustituir la suma
ee.Fin del si
b. Fin del Mientras
3. Repetir para el Cubo 2
4. Comparar cubos:
a. Para los k elementos del Cubo 1, i=0
aa. Para los k elementos del Cubo 2, j=0
I. Ver si los mejores en Cubo 1 en i es igual a los mejores en Cubo 1 en j
bb. Fin del ciclo
b. Fin de ciclo
5. Regresar elementos iguales
6. Graficar

Field Summary

Fields inherited from class preguntas.Pregunta
aDatosDimsCompres, tiPregunta

Constructor Summary

Pregunta5(int numDimensiones) Constructor para la pregunta 5	

Method Summary

java.lang.String[][]	**answerP5**(java.lang.String[] aDatosDims, java.lang.String[] aDatosDims2) Contesta la pregunta con el algoritmo antes descrito.
boolean[]	**compareCubes**(java.lang.String[] aTheBestCube1Data, java.lang.String[] aTheBestCube2Data) Regresa Verdadero o falso del primer arreglo si es que dicho dato se encuentra en ambos

long	**findHighest**(java.lang.String[] aSums, long value) Sustituir el arreglo en busca de los n mayores
long	**findSmallest**(java.lang.String[] aSums, long value) Sustituir el arreglo en busca de los n menores
int	**getDim_kept**() Obtiene el número de la dimensión que permanece
java.lang.String[]	**getNumDiferentValues**(int[] questArrayTam, java.lang.String[][] aQCatalogo) Obtiene los diversos valores para la dimensión que se mantiene
int	**getNumMejores**() Obtiene el número de mejores que el usuario requiera
java.lang.String	**getValueDescompres**(java.lang.String aDato) Descomprime un valor de la dimensión que se conserva
java.lang.String[][]	**losNMejores1Cube**(java.lang.String[] aDatosDims) Obtiene los N mejores en determinado rango de valores (cubo de datos)
void	**setDim_kept**(int dim_kept) Establece el número de la dimensión comenzando por cero, el cual es dado regularmente por el usuario
void	**setNumMejores**(int numMejores) Establece el número de mejores valores, el cual es dado regularmente por el usuario

Methods inherited from class preguntas.Pregunta
findDateFromIndex, findDateIndex

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Pregunta5
`public Pregunta5(int numDimensiones)`
Constructor para la pregunta 5

Method Detail

getNumMejores
`public int getNumMejores()`
Obtiene el número de mejores que el usuario requiera
Returns:
int regresa el valor de la cantidad de mejores/peores

---

setNumMejores
`public void setNumMejores(int numMejores)`
Establece el número de mejores valores, el cual es dado regularmente por el usuario
Parameters:
`num_best` - cantidad de mejores

---

getDim_kept
`public int getDim_kept()`
Obtiene el número de la dimensión que permanece
Returns:
int regresa el valor de la dimension pivote

---

setDim_kept
`public void setDim_kept(int dim_kept)`
Establece el número de la dimensión comenzando por cero, el cual es dado regularmente por el usuario
Parameters:

`dim_kept` - dimension que se conserva

---

answerP5

```
public java.lang.String[][] answerP5(java.lang.String[] aDatosDims,
                                     java.lang.String[] aDatosDims2)
```

Contesta la pregunta con el algoritmo antes descrito.

Parameters:

`aDatosDims` - String [] del rango de valores del primer cubo de datos

`aDatosDims2` - String [] del rango de valores del segundo cubo de datos

Returns:

String[][] Es una matriz de 4 por la cantidad de mejores/peores deseados

1. Valor del mejor dato
2. Suma del primer cubo
3. Suma del segundo cubo
4. Eficiencia calculada a partir de las sumas anteriores

---

losNMejores1Cube

```
public java.lang.String[][] losNMejores1Cube(java.lang.String[] aDatosDims)
```

Obtiene los N mejores en determinado rango de valores (cubo de datos)

Parameters:

`aDatosDims` - String [] del rango de valores del primer cubo de datos

`aDatosDims2` - String [] del rango de valores del segundo cubo de datos

Returns:

String[][] Es una matriz de 2 por la cantidad de mejores/peores deseados

1. Valor del mejor dato
2. Suma del primer cubo

---

compareCubes

```
public boolean[] compareCubes(java.lang.String[] aTheBestCube1Data,
                              java.lang.String[] aTheBestCube2Data)
```

Regresa Verdadero o falso del primer arreglo si es que dicho dato se encuentra en ambos

Parameters:

`aTheBestCube1Data` - String [] los n mejores en el primer cubo

`aTheBestCube2Data` - String [] los n mejores en el segundo cubo de datos

Returns:

boolen[][] determina cuales se mantienen y cuales no

---

getNumDiferentValues

```
public java.lang.String[] getNumDiferentValues(int[] questArrayTam,
                                               java.lang.String[][] aQCatalogo)
```

Obtiene los diversos valores para la dimensión que se mantiene

Parameters:

`questArrayTam` - int[] tamaño de cada una de las dimensiones

`aQCatalogo` - String [][] catálogo

Returns:

String[] cadena de String que contiene los diversos valores a intercambiarse en la dimensión que se mantiene, que servirá para generar los sub-cubos del cubo 1 y 2

---

getValueDescompres

```
public java.lang.String getValueDescompres(java.lang.String aDato)
```

Descomprime un valor de la dimensión que se conserva

Parameters:

`aDato` - String

Returns:

String aDato descomprimido

---

findSmallest

```
public long findSmallest(java.lang.String[] aSums,
                         long value)
```

Sustituir el arreglo en busca de los n menores

Parameters:

`aSums` - String[] conjunto de elementos que contiene la suma de los n mejores

`value` - long del dato a ser evaluado dentro de aSums para ver si desplazará a algun dato o no

Returns:

long regresa el número del menor

---

findHighest

```
public long findHighest(java.lang.String[] aSums,
                        long value)
```

Sustituir el arreglo en busca de los n mayores

Parameters:

`aSums` - String[] conjunto de elementos que contiene la suma de los n mejores

`value` - long del dato a ser evaluado dentro de aSums para ver si desplazará a algun dato o no

Returns:

long regresa el número del mayor

---

preguntas

Class Pregunta6

```
java.lang.Object
  └ preguntas.Pregunta
      └ preguntas.Pregunta6
```

```
public class Pregunta6
extends Pregunta
```

Pregunta 6 de temporalidad:
Esta pregunta responde a los elementos que permanecen durante temporadas como los mejores/peores. Para ello se define la dimensión donde se buscan los elementos, se definen a cuantos k mejores/peores elementos de la dimensión se desea ver si permanecieron en varios períodos de tiempo. A partir de un cubo inicial Cubo 1.

Algoritmo

1. Obtener la dimensión de tiempo sobre la que se navegará
2. Comprimir valores de rango del Cubo 1
3. Si el número de temporadas es uno entonces
a. Calcular los k mejores en el cubo 1
4. Si no
a. Copiar los valores del cubo 1 al cubo 2
b, Calcular los k mejores en el cubo 1
c. Para i=0, mientras sea menor al número de temporadas
I. Reasignar las fechas en el cubo 2
II. Obtener los k mejores en el cubo 2
III. Comparar cuáles permanecen
d. Fin del mientras
5. Fin del si
6. Obtener el número de elementos que se conservaron
7. Obtener la matriz que guarda las respuestas de la intersección de los k cubos
8. Devolver la matriz para obtener su gráfica y tabla de valores correspondiente

Field Summary

Fields inherited from class preguntas.Pregunta
aDatosDimsCompres, tiPregunta

Constructor Summary

Pregunta6(int numDimensiones) Constructor para la pregunta 5

Method Summary

java.lang.String[][]	**answerQ6**(java.lang.String[] aDatosDims) Contesta la pregunta con el algoritmo antes descrito.
boolean[]	**compareCubes**(java.lang.String[] aTheBestCube1Data, java.lang.String[] aTheBestCube2Data, boolean[] aEqual) Regresa Verdadero o falso del primer arreglo si es que dicho dato se encuentra en ambos
long	**findHighest**(java.lang.String[] aSums, long value) Sustituir el arreglo en busca de los n mayores
long	**findSmallest**(java.lang.String[] aSums, long value) Sustituir el arreglo en busca de los n menores
int	**getDim_kept**() Obtiene el número de la dimensión que permanece

java.lang.String	**getNewSeasonDate**(java.lang.String dateStr) Obtiene la sigueinte temporada a partir de una fecha recibida
java.lang.String[]	**getNumDiferentValues**(int[] questArrayTam, java.lang.String[][] aQCatalogo) Obtiene los diversos valores para la dimensión que se mantiene
int	**getNumMejores**() Obtiene el número de mejores que el usuario requiera
int	**getSeasonNum**() Obtiene el número de temporadas
SeasonType	**getSeasonType**() Obtiene el tipo de temporadas
java.lang.String	**getValueDescompres**(java.lang.String aDato) Descomprime un valor de la dimensión que se conserva
java.lang.String[][]	**losNMejores1Cube**(java.lang.String[] aDatosDims) Obtiene los N mejores en determinado rango de valores (cubo de datos)
void	**setDim_kept**(int dim_kept) Establece el número de la dimensión comenzando por cero, el cual es dado regularmente por el usuario
void	**setNumMejores**(int numMejores) Establece el número de mejores valores, el cual es dado regularmente por el usuario
void	**setSeason**(SeasonType seasonType) Establece el tipo de temporadas
void	**setSeasonNum**(int seasonNum) Establece el numero de temporadas

Methods inherited from class preguntas.Pregunta
findDateFromIndex, findDateIndex

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Pregunta6
public **Pregunta6**(int numDimensiones)
Constructor para la pregunta 5

Method Detail

getNumMejores
public int **getNumMejores**()
Obtiene el número de mejores que el usuario requiera
Returns:
int regresa el valor de la cantidad de mejores/peores

---

setNumMejores
public void **setNumMejores**(int numMejores)
Establece el número de mejores valores, el cual es dado regularmente por el usuario
Parameters:
num_best - cantidad de mejores

---

getSeasonType
public SeasonType **getSeasonType**()
Obtiene el tipo de temporadas
Returns:
SeasonType regresa el valor del tipo de temporada

---

setSeason
`public void setSeason(SeasonType seasonType)`

Establece el tipo de temporadas

Parameters:

`SeasonType` - tipo de temporadas

---

getSeasonNum
`public int getSeasonNum()`

Obtiene el número de temporadas

Returns:

int regresa la cantidad de temporadas

---

setSeasonNum
`public void setSeasonNum(int seasonNum)`

Establece el numero de temporadas

Parameters:

`int` - valor del numero de temporadas

---

getDim_kept
`public int getDim_kept()`

Obtiene el número de la dimensión que permanece

Returns:

int regresa el valor de la dimension pivote

---

setDim_kept
`public void setDim_kept(int dim_kept)`

Establece el número de la dimensión comenzando por cero, el cual es dado regularmente por el usuario

Parameters:

`dim_kept` - dimension que se conserva

---

answerQ6
`public java.lang.String[][] answerQ6(java.lang.String[] aDatosDims)`

Contesta la pregunta con el algoritmo antes descrito.

Parameters:

`aDatosDims` - String [] del rango de valores del primer cubo de datos

Returns:

String[][] Es una matriz del numero de dimensiones que se conservan por 3
1. Valor del mejor dato
2. Suma del primer cubo
3. Suma del segundo cubo

---

getNewSeasonDate
`public java.lang.String getNewSeasonDate(java.lang.String dateStr)`

Obtiene la sigueinte temporada a partir de una fecha recibida

Parameters:

`dateStr` - La fecha es recibida en formato de cadena (String)

Returns:

String fecha de la siguiente temporada

---

losNMejores1Cube
`public java.lang.String[][] losNMejores1Cube(java.lang.String[] aDatosDims)`

Obtiene los N mejores en determinado rango de valores (cubo de datos)

Parameters:

`aDatosDims` - String [] del rango de valores del primer cubo de datos

Returns:

String[][] Es una matriz de 2 por la cantidad de mejores/peores deseados
1. Valor del mejor dato
2. Suma del primer cubo

---

compareCubes
`public boolean[] compareCubes(java.lang.String[] aTheBestCube1Data,`

```
                        java.lang.String[] aTheBestCube2Data,
                        boolean[] aEqual)
```

Regresa Verdadero o falso del primer arreglo si es que dicho dato se encuentra en ambos

Parameters:

`aTheBestCube1Data` - String [] los n mejores en el primer cubo

`aTheBestCube2Data` - String [] los n mejores en el segundo cubo de datos

`aEqual` - boolean[] para ver cuales habia sido iguales en temporadas anteriores

Returns:

boolen[][] determina cuales se mantienen y cuales no

---

getNumDiferentValues

```
public java.lang.String[] getNumDiferentValues(int[] questArrayTam,
                                               java.lang.String[][] aQCatalogo)
```

Obtiene los diversos valores para la dimensión que se mantiene

Parameters:

`questArrayTam` - int[] tamaño de cada una de las dimensiones

`aQCatalogo` - String [][] catálogo

Returns:

String[] cadena de String que contiene los diversos valores a intercambiarse en la dimensión que se mantiene, que servirá para generar los sub-cubos del cubo 1 y 2

---

getValueDescompres

```
public java.lang.String getValueDescompres(java.lang.String aDato)
```

Descomprime un valor de la dimensión que se conserva

Parameters:

`aDato` - String

Returns:

String aDato descomprimido

---

findSmallest

```
public long findSmallest(java.lang.String[] aSums,
                         long value)
```

Sustituir el arreglo en busca de los n menores

Parameters:

`aSums` - String[] conjunto de elementos que contiene la suma de los n mejores

`value` - long del dato a ser evaluado dentro de aSums para ver si desplazará a algun dato o no

Returns:

long regresa el número del menor

---

findHighest

```
public long findHighest(java.lang.String[] aSums,
                        long value)
```

Sustituir el arreglo en busca de los n mayores

Parameters:

`aSums` - String[] conjunto de elementos que contiene la suma de los n mejores

`value` - long del dato a ser evaluado dentro de aSums para ver si desplazará a algun dato o no

Returns:

long regresa el número del mayor

---

preguntas

Class Pregunta7

```
java.lang.Object
  └ preguntas.Pregunta
      └ preguntas.Pregunta7
```

```
public class Pregunta7
extends Pregunta
```

Pregunta 7 de tendencia:
Esta pregunta responde a los elementos que permanecen durante temporada. Para ello se define la dimensión donde se buscan los elementos. A partir de un cubo inicial Cubo 1. Se buscan todos los elementos durante cierto número y tipo de temporadas

Algoritmo

1. Obtener la dimensión de tiempo sobre la que se navegará
2. Comprimir valores de rango del Cubo 1
3. Mientras existan valores en la dimensión donde se buscan los elementos, i=0

a. Sustituir el valor de la dimensión i
b. Copiar Valores para Cubo 1 a valores para cubo 2
c. Obtener fecha para el cubo 2
d. Mientras j sea menor al número de temporadas, j=0

I. Responder con rango el Cubo 2
II. Guardar la suma
III. Calcular la nueva fecha para Cubo 2

e. Fin del mientras
f. Incrementar m

4. Fin del mientras

Field Summary

Fields inherited from class preguntas.Pregunta
aDatosDimsCompres, tiPregunta

Constructor Summary

Pregunta7(int numDimensiones) Constructor de para la pregunta 7: Tendencia.	

Method Summary

java.lang.String[][]	**answerQ7**(java.lang.String[] aDatosDims) Contesta la pregunta con el algoritmo antes descrito.
int	**getDim_kept**() Obtiene el número de la dimensión que permanece
java.lang.String	**getNewSeasonDate**(java.lang.String dateStr) Obtiene la sigueinte temporada a partir de una fecha recibida La fecha es recibida en formato de cadena (String)
java.lang.String[]	**getNumDiferentValues**(int[] questArrayTam, java.lang.String[][] aQCatalogo)
int	**getSeasonNum**() Obtiene el número de temporadas
SeasonType	**getSeasonType**() Obtiene el tipo de temporadas
java.lang.String	**getValueDescompres**(java.lang.String aDato)

void	**setDim_kept**(int dim_kept) Establece el número de la dimensión comenzando por cero, el cual es dado regularmente por el usuario
void	**setSeason**(SeasonType seasonType) Establece el tipo de temporadas
void	**setSeasonNum**(int seasonNum) Establece el numero de temporadas

Methods inherited from class preguntas.Pregunta
findDateFromIndex, findDateIndex

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Pregunta7

```
public Pregunta7(int numDimensiones)
```

Constructor de para la pregunta 7: Tendencia. Establece que tiPregunta sera tendencia y establece el número de dimnsiones que le sean pasado en los parámetros

Parameters:

`numDimensiones` - int cantidad de dimensiones

Method Detail

getSeasonType

```
public SeasonType getSeasonType()
```

Obtiene el tipo de temporadas

Returns:

SeasonType regresa el valor del tipo de temporada

---

setSeason

```
public void setSeason(SeasonType seasonType)
```

Establece el tipo de temporadas

Parameters:

`SeasonType` - tipo de temporadas

---

getSeasonNum

```
public int getSeasonNum()
```

Obtiene el número de temporadas

Returns:

int regresa la cantidad de temporadas

---

setSeasonNum

```
public void setSeasonNum(int seasonNum)
```

Establece el numero de temporadas

Parameters:

`int` - valor del numero de temporadas

---

getDim_kept

```
public int getDim_kept()
```

Obtiene el número de la dimensión que permanece

Returns:

int regresa el valor de la dimension pivote

---

setDim_kept

```
public void setDim_kept(int dim_kept)
```

Establece el número de la dimensión comenzando por cero, el cual es dado regularmente por el usuario

Parameters:
`dim_kept` - dimension que se conserva

---

answerQ7
```
public java.lang.String[][] answerQ7(java.lang.String[] aDatosDims)
```
Contesta la pregunta con el algoritmo antes descrito.
Parameters:
`aDatosDims` - String [] del rango de valores del primer cubo de datos
Returns:
String[][] Es una matriz de 2 por el numero de elementos
1. Valor del mejor dato
2. Suma del cubo

---

getNewSeasonDate
```
public java.lang.String getNewSeasonDate(java.lang.String dateStr)
```
Obtiene la sigueinte temporada a partir de una fecha recibida La fecha es recibida en formato de cadena (String)
Parameters:
`dateStr` - La fecha es recibida en formato de cadena (String)
Returns:
String fecha de la siguiente temporada

---

getNumDiferentValues
```
public java.lang.String[] getNumDiferentValues(int[] questArrayTam,
                                               java.lang.String[][] aQCatalogo)
```

---

getValueDescompres
```
public java.lang.String getValueDescompres(java.lang.String aDato)
```

---

preguntas

Enum SeasonType

```
java.lang.Object
  └ java.lang.Enum<SeasonType>
      └ preguntas.SeasonType
```

All Implemented Interfaces:

java.io.Serializable, java.lang.Comparable<SeasonType>

```
public enum SeasonType
extends java.lang.Enum<SeasonType>
```

Enumeración para definir los periodos temporales

Enum Constant Summary

annual	Anual, consta de 365 días
bimonthly	Bimestral, consta de 60 días
daily	Diaria, consta de 1 día
fourMonths	Cuatrimestral, consta de 120 días
monthly	Mensual, consta de 30 días
quarterly	Trimestral, consta de 90 días
weekly	Semanal, consta de 7 días

Method Summary

long	**getSeasonDays**() Devuelve la cantidad de días dependiendo de la estación
static SeasonType	**valueOf**(java.lang.String name) Returns the enum constant of this type with the specified name.
static SeasonType[]	**values**() Returns an array containing the constants of this enum type, in the order they are declared.

Methods inherited from class java.lang.Enum

clone, compareTo, equals, finalize, getDeclaringClass, hashCode, name, ordinal, toString, valueOf

Methods inherited from class java.lang.Object

getClass, notify, notifyAll, wait, wait, wait

Enum Constant Detail

daily

```
public static final SeasonType daily
```

Diaria, consta de 1 día

---

weekly

```
public static final SeasonType weekly
```

Semanal, consta de 7 días

---

monthly
`public static final SeasonType monthly`

Mensual, consta de 30 días

---

bimonthly
`public static final SeasonType bimonthly`

Bimestral, consta de 60 días

---

quarterly
`public static final SeasonType quarterly`

Trimestral, consta de 90 días

---

fourMonths
`public static final SeasonType fourMonths`

Cuatrimestral, consta de 120 días

---

annual
`public static final SeasonType annual`

Anual, consta de 365 días

Method Detail

values
`public static SeasonType[] values()`

Returns an array containing the constants of this enum type, in the order they are declared. This method may be used to iterate over the constants as follows:

```
for (SeasonType c : SeasonType.values())
    System.out.println(c);
```

Returns:

an array containing the constants of this enum type, in the order they are declared

---

valueOf
`public static SeasonType valueOf(java.lang.String name)`

Returns the enum constant of this type with the specified name. The string must match *exactly* an identifier used to declare an enum constant in this type. (Extraneous whitespace characters are not permitted.)

Parameters:

`name` - the name of the enum constant to be returned.

Returns:

the enum constant with the specified name

Throws:

`java.lang.IllegalArgumentException` - if this enum type has no constant with the specified name

`java.lang.NullPointerException` - if the argument is null

---

getSeasonDays
`public long getSeasonDays()`

Devuelve la cantidad de días dependiendo de la estación

Returns:

cantidad de días para cada estación

---

preguntas

Enum TipoPregunta

```
java.lang.Object
  └ java.lang.Enum<TipoPregunta>
      └ preguntas.TipoPregunta
```

All Implemented Interfaces:

java.io.Serializable, java.lang.Comparable<TipoPregunta>

```
public enum TipoPregunta
extends java.lang.Enum<TipoPregunta>
```

Enumeración para los tipos de pregunta que son manejados

Enum Constant Summary

CONSERVACION
Pregunta 5 sobre conservación y pérdida: En esta pregunta se define la dimensión sobre la que se desea calcular la eficiencia de los elementos.

EFICIENCIA
Pregunta 3 de eficiencia global: Esta Pregunta se define sobre dos cubos por lo que se reciben dos conjuntos de rangos que serán evaluados con la pregunta de rango individualmente y posteriormente se calculara la eficiencia de éstos

EFICIENCIA_GRUPAL
Pregunta 4 de eficiencia grupal: En esta pregunta se define la dimensión sobre la que se desea calcular la eficiencia de los elementos.

PUNTUAL
Pregunta 1 puntual: Esta pregunta se define sobre un solo cubo C1.

RANGO
Pregunta 2 de Rango: Se define un rango en al menos uno de las dimensiones, De tal forma que podría no constar de un único valor, sino de varios, los cuales son contados y su agregado es sumado.

TEMPORADA
Pregunta 6 de temporalidad: Esta pregunta responde a los elementos que permanecen durante temporadas como los mejores/peores.

TENDENCIA
Pregunta 7 de tendencia: Esta pregunta responde a los elementos que permanecen durante temporada.

Method Summary

`static TipoPregunta`	**valueOf**`(java.lang.String name)` Returns the enum constant of this type with the specified name.
`static TipoPregunta[]`	**values**`()` Returns an array containing the constants of this enum type, in the order they are declared.

Methods inherited from class java.lang.Enum

`clone, compareTo, equals, finalize, getDeclaringClass, hashCode, name, ordinal, toString, valueOf`

Methods inherited from class java.lang.Object

`getClass, notify, notifyAll, wait, wait, wait`

Enum Constant Detail

PUNTUAL

```
public static final TipoPregunta PUNTUAL
```

Pregunta 1 puntual: Esta pregunta se define sobre un solo cubo C1. Donde todos los elementos que definen al cubo son un único valor. De tal forma que C1 es un cubo de datos de un solo punto (valor).

RANGO
`public static final TipoPregunta RANGO`

Pregunta 2 de Rango: Se define un rango en al menos uno de las dimensiones, De tal forma que podría no constar de un único valor, sino de varios, los cuales son contados y su agregado es sumado.

EFICIENCIA
`public static final TipoPregunta EFICIENCIA`

Pregunta 3 de eficiencia global: Esta Pregunta se define sobre dos cubos por lo que se reciben dos conjuntos de rangos que serán evaluados con la pregunta de rango individualmente y posteriormente se calculara la eficiencia de éstos

---

EFICIENCIA_GRUPAL
`public static final TipoPregunta EFICIENCIA_GRUPAL`

Pregunta 4 de eficiencia grupal: En esta pregunta se define la dimensión sobre la que se desea calcular la eficiencia de los elementos. Se define el número de elementos k que se desean encontrar. La primera búsqueda es sobre el Cubo 1 que devolverá los k mejores/peores elementos, los cuales serán buscados en el Cubo 2 para comparar en comportamiento de los k elementos en 2 rangos diferentes.

CONSERVACION
`public static final TipoPregunta CONSERVACION`

Pregunta 5 sobre conservación y pérdida: En esta pregunta se define la dimensión sobre la que se desea calcular la eficiencia de los elementos. Se define el número de elementos k que se desean encontrar. Se repite la operación para un Cubo 2 y se verifica cuales se conservaron y cuales se perdieron al comparar ambos cubos.

TEMPORADA
`public static final TipoPregunta TEMPORADA`

Pregunta 6 de temporalidad: Esta pregunta responde a los elementos que permanecen durante temporadas como los mejores/peores. Para ello se define la dimensión donde se buscan los elementos, se definen a cuantos k mejores/peores elementos de la dimensión se desea ver si permanecieron en varios períodos de tiempo. A partir de un cubo inicial Cubo 1.

TENDENCIA
`public static final TipoPregunta TENDENCIA`

Pregunta 7 de tendencia: Esta pregunta responde a los elementos que permanecen durante temporada. Para ello se define la dimensión donde se buscan los elementos. A partir de un cubo inicial Cubo 1. Se buscan todos los elementos durante cierto número y tipo de temporadas

Method Detail

values
`public static TipoPregunta[] values()`

Returns an array containing the constants of this enum type, in the order they are declared. This method may be used to iterate over the constants as follows:

```
for (TipoPregunta c : TipoPregunta.values())
    System.out.println(c);
```

Returns:
an array containing the constants of this enum type, in the order they are declared

---

valueOf
`public static TipoPregunta valueOf(java.lang.String name)`

Returns the enum constant of this type with the specified name. The string must match *exactly* an identifier used to declare an enum constant in this type. (Extraneous whitespace characters are not permitted.)

Parameters:
`name` - the name of the enum constant to be returned.

Returns:
the enum constant with the specified name

Throws:
`java.lang.IllegalArgumentException` - if this enum type has no constant with the specified name
`java.lang.NullPointerException` - if the argument is null

ANEXO B. MANUAL DE USUARIO

TABLA DE CONTENIDO

1. INTRODUCCIÓN

La herramienta está basada en el Analizador Temporal en Cubos en Memoria ***Antecumem***. El cual permite un análisis de datos por medio de preguntas. De tal forma que haciendo uso de éstas se puede contestar diversas preguntas de negocio. Dicho procedimiento es realizado haciendo uso de la Estructura AC la cual ocupa menor espacio en disco y memoria principal, dentro de la cual se realizan las consultas para n dimensiones.

El conjunto de preguntas que se realizan son preguntas específicas sobre un dato en específico, un conjunto de ellos, patrones de comportamiento en cambios de condiciones y tiempos. Para esto fueron establecidas 7 preguntas enlistadas a continuación:

Las preguntas de negocios que puede responder son las siguientes:

1. ***Puntual.*** Proporcionar el valor de un elemento en un entorno en específico.
2. ***Rango.*** Proporcionar elementos acotado por los rangos.
3. ***Eficiencia.*** Calcular el incremento o decremento en dos períodos con rangos.

4. ***Eficiencia Grupal.*** Mide la eficiencia en los *n* elementos en dos periodos.
5. ***Conservación/Perdida.*** Mide la eficiencia grupal observando los *n* elementos que permanecen o que desaparecen en diversos período.
6. ***Temporalidad.*** Proporcionar los *n* "mejores" elementos que permanecen, en dos o más períodos de tiempo.
7. ***Tendencias.*** Buscar los n elementos de interés que mantienen una tendencia en p lapsos de tiempo.

2. OBJETIVO DEL MANUAL

El presente documento tiene como fin explicar la operación de la herramienta para:

- Creación de la Estructura AC
- Mostrar la Estructura AC y los catálogos de la Estructura AC
- La forma de realizar las preguntas de negocios.
- Revisar sus resultados de las pruebas de negocio
- Ayuda

3. REQUERIMIENTOS Y ACCESO

3.1 SOFTWARE NECESARIO

- Máquina Virtual de Java (JRE).
- Un navegador para Internet

3.2 FORMA DE ACCEDER

A continuación se describen dos formas para ejecutar el programa:

a) Acceso por el ícono. Dar doble clic sobre el ícono con el nombre de CompresArblis.jar

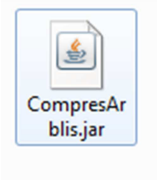


Ilustración 1. Icono

b) Acceso desde consola. Se escribe el siguiente comando:

java -jar "*ruta*\CompresArblis.jar"

Sustituyendo *ruta* por la ruta correspondiente. Por ejemplo si el archivo CompresArblis.jar se encuentra en el escritorio la ruta quedaría:

C:\Users\Owner\Desktop\

Donde el comando correcto para su acceso sería:

java -jar "C:\Users\Owner\Desktop\CompresArblis.jar"

De tal forma que se tendría lo mostrado en XX y se presiona ENTER

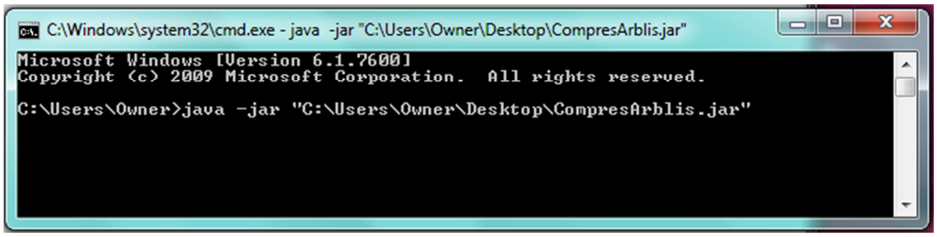


Ilustración 2. Acceso en cmd

3.3 VENTANAS DE PRESENTACIÓN

Una vez que se ejecute el programa la pantalla que se debe de desplegar es la mostrada en Ilustración 3

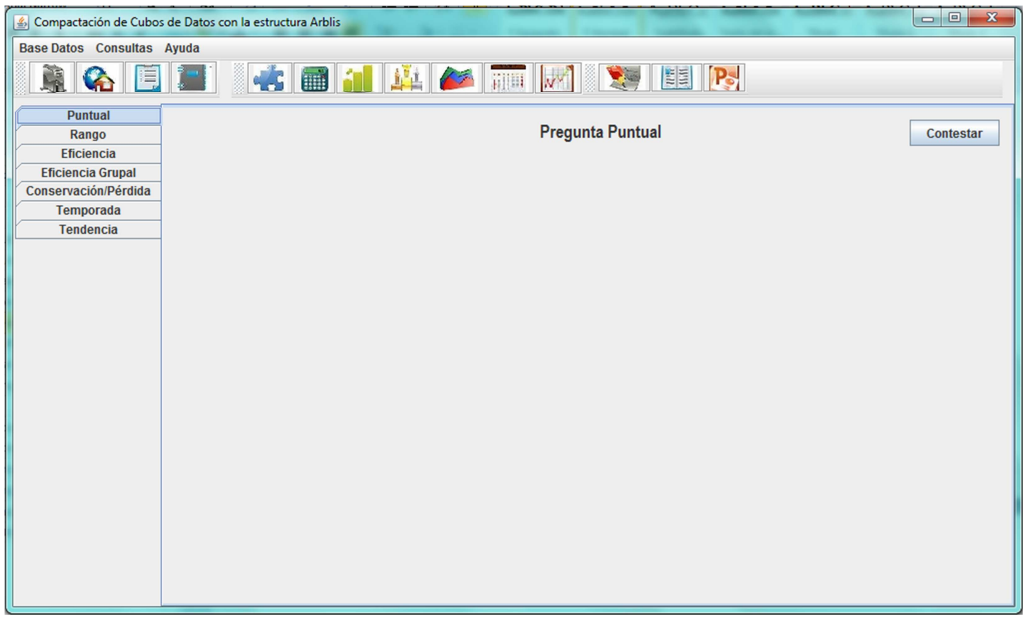


Ilustración 3. Pantalla de Inicio

4. DESCRIPCIÓN DE LA CREACIÓN DE LA ESTRUCTURA AC

La creación de la Estructura AC requiere de la estructura Arblis original la cual se encuentra en un formato *.out.

Para realizar la compresión a Arblis es necesario seleccionar el icono mostrado en Ilustración 4

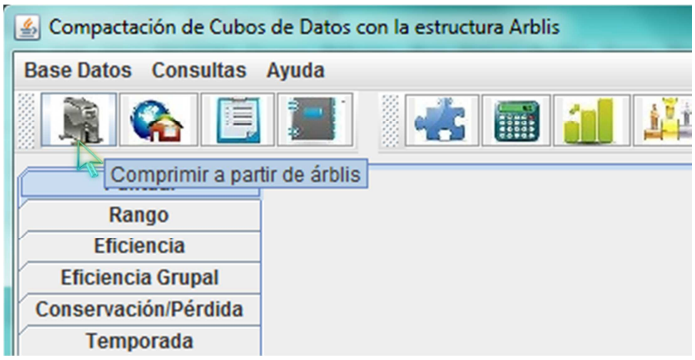


Ilustración 4. Botón de Creación de AC

Una vez que sea oprimido, es necesario seleccionar un archivo con el formato antes descrito, y oprimir el botón de Abrir, de la Ilustración 5, una vez que sea seleccionado se le solicitará al usuario el número de dimensiones.

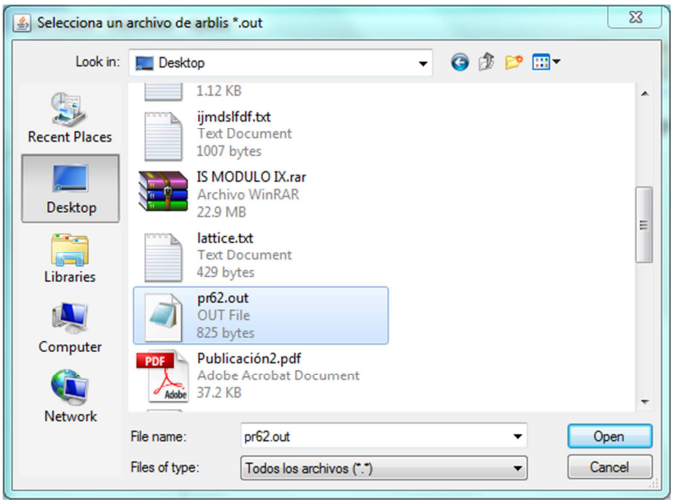


Ilustración 5. Seleccionar archivo de Arblis

Una vez que la creación de catálogos y de la estructura AC hayan sido creadas se mostrará un mensaje como el mostrado en la Ilustración 6.

Ilustración 6. Mensaje de compresión exitosa

5. DESCRIPCIÓN DE LA VISUALIZACIÓN DE ESTRUCTURAS

La estructura creada AC, así como los catálogos requeridos para cada una de las dimensiones son guardados en archivos, de tal forma que se tendrán el número de archivos correspondiente al número de dimensiones, además del de la estructura creada.

La herramienta permite tener acceso a estos archivos como se describe a continuación.

5.1 MOSTRAR LA ESTRUCTURA AC

Para ver la estructura AC compresa se selecciona el botón mostrado en la Ilustración 7. Lo que permitirá se abra el archivo en un bloc de notas.

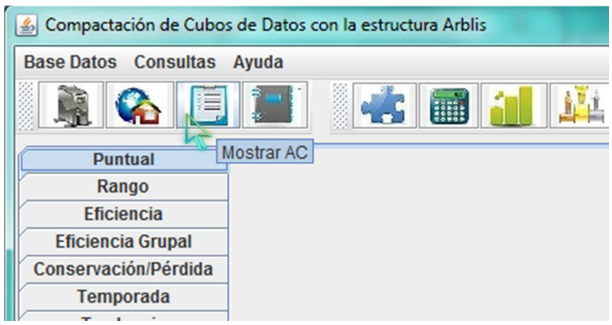


Ilustración 7. Mostrar AC

5.2 MOSTRAR LOS CATÁLOGOS USADOS EN LA ESTRUCTURA AC

Si *n* es el número de dimensiones que se tienen, existe el mismo número de catálogos, los cuales describen el tiepo de compresión empleada así como los elementos necesarios para su descompresión, de tal forma que si se desean visualizar los n catálogos para examinarlos se debe de seleccionar el botón mostrado en la Ilustración 8, el cual hará que se abran los archivos correspondientes a la estructura seleccionada.

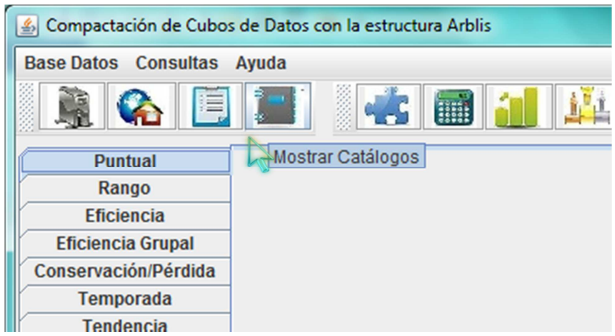


Ilustración 8. Mostrar Catálogos

6. DESCRIPCIÓN DE CAPTURA Y EJECUCIÓN DE PREGUNTAS DE NEGOCIOS

Una vez que se tiene materializada la Estructura AC es necesario cargarla en memoria la primera vez que es utilizada para lo cual se selecciona el botón mostrado en la Ilustración 9

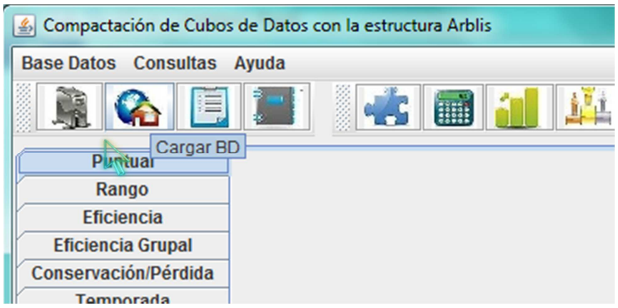


Ilustración 9. Cargar la estructura AC

Se solicita el número de las dimensiones, así como el nombre de las dimensiones y el agregado, como se observa en los ejemplos de la Ilustración 10.

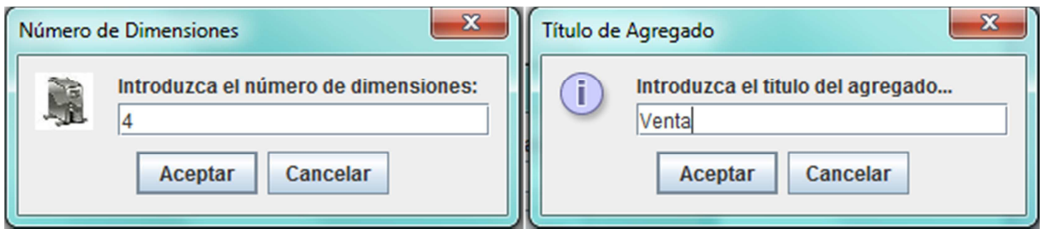


Ilustración 10. Introducción de datos

Nos muestra un mensaje de confirmación cuando se ha cargado en memoria, y se informa que se cargará la interfaz gráfica correspondiente al número de dimensiones y las descripciones introducidas.

Una vez que se tiene en memoria la Estructura AC es posible contestar siete preguntas. (Para mayor detalle ver XX), para las cuales se describe su acceso a continuación.

6.1 Pregunta Puntual

Para ingresar una pregunta puntual se selecciona la pestaña que dice **Puntual** en el lado izquierdo y se introducen los datos.

Una pregunta de este tipo sería "*Se desea saber la venta el 21 de enero de 1999 del producto 5, del cliente 50 y la promoción 47*".

Por lo que se selecciona o se alimenta los siguientes valores en las respectivas dimensiones:

- *products*=5
- *customers*=50
- *promotion*=47 y
- *times*=1999-01-21

Después de oprime **Contestar** y se observaría el resultado en un cuadro de diálogo como el que está en la derecha de la Ilustración 11.

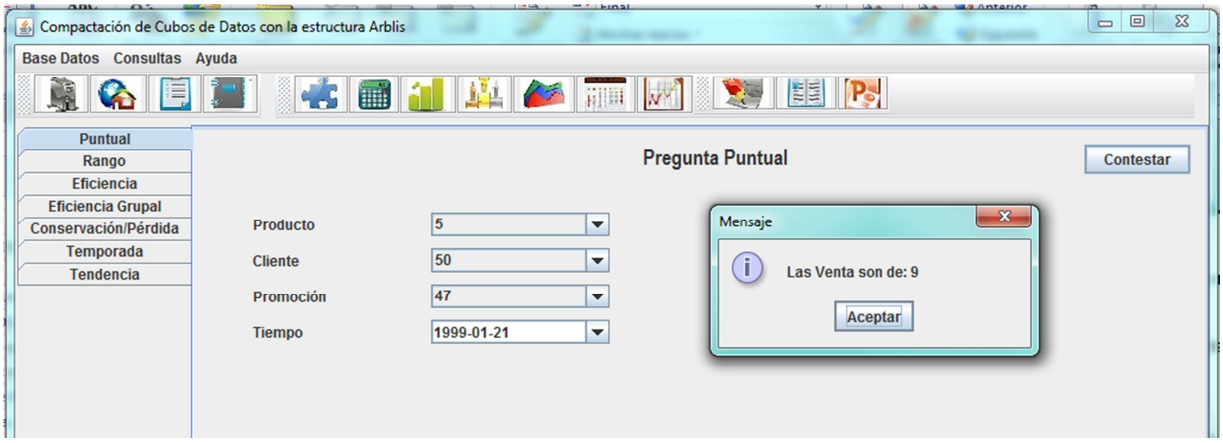


Ilustración 11. Pregunta Puntual

6.2 Pregunta de Rango

Para ingresar una pregunta puntual se selecciona la pestaña que dice **Rango** en el lado izquierdo y se introducen los datos.

Una pregunta de este tipo sería "*Se desea saber la venta generada por la promoción 9999 en todos los productos y todos los clientes en todos los años que se tienen almacenados en la estructura*".

Se selecciona o se alimenta los siguientes valores en las respectivas dimensiones:

- *products*=[5- 49980]
- *customers*=[50-189450]
- *promotion*=[9999-9999]
- *times*=[1998/01/01-2000/11/29]

Se oprime **Contestar** y los resultados se observan en Ilustración 12.

Ilustración 12. Pregunta de Rango

6.3 Pregunta de Eficiencia

Para ingresar una pregunta puntual se selecciona la pestaña que dice **Eficiencia** en el lado izquierdo y se introducen los datos.

Una pregunta de este tipo sería "*Se desea saber el incremento de ventas de 1998 a 1999 en el producto 5 en todos los clientes y todas las promociones*.

Se seleccionan o se alimentan los siguientes valores en las respectivas dimensiones para el primer cubo:

- *products*=[5- 5]
- *customers*=[50-189450]
- *promotion*=[1-9999]
- *times*=[1998/01/01-1998/12/230]

Y para el segundo cubo de datos se seleccionan o se alimentan los siguientes valores en las respectivas dimensiones:

- *products*=[5- 5]
- *customers*=[50-189450]
- *promotion*=[1-9999]
- *times*=[1999/01/01-1999/01/21]

Se oprime **Contestar** y los resultados se observan en Ilustración 13.

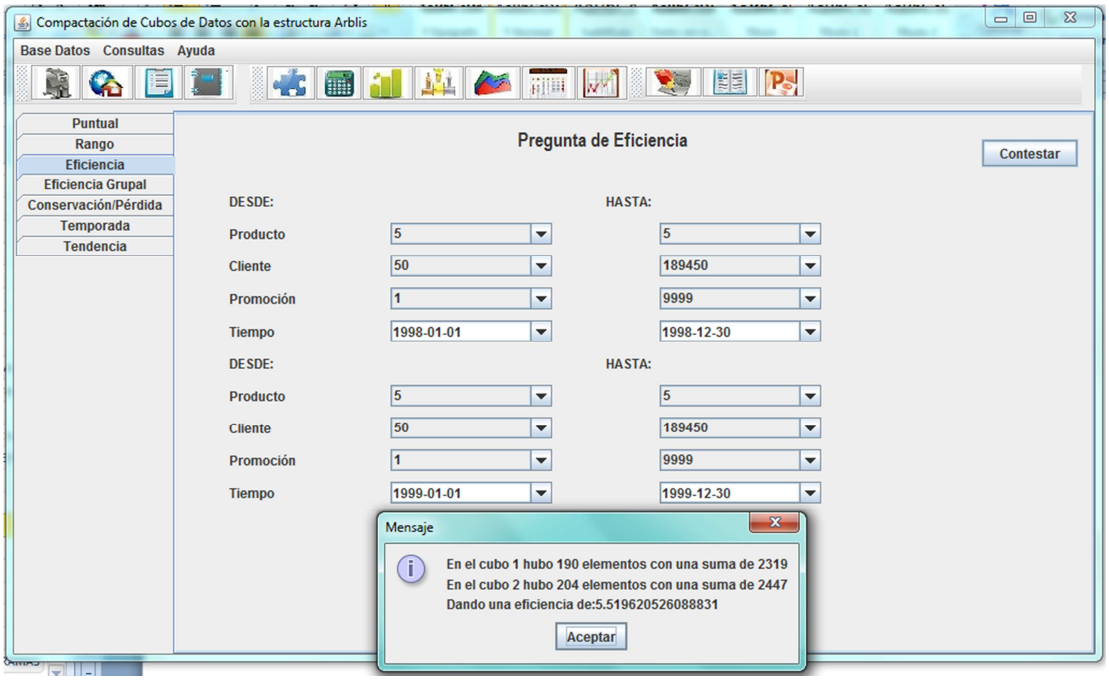


Ilustración 13. Pregunta de Eficiencia

6.4 Pregunta de Eficiencia Grupal

Para ingresar una pregunta puntual se selecciona la pestaña que dice **Eficiencia Grupal** en el lado izquierdo y se introducen los datos.

Una pregunta de este tipo sería "*¿Cuáles son los mejores 2 productos de la tienda 255 en el periodos de 01/01/2000 a 01/01/2004 y que eficiencia muestran con respecto a la tienda 300?*".

Se seleccionan o se alimentan los siguientes valores en las respectivas dimensiones para el primer cubo de dato:

- *products*=todos
- *tienda*=[255-255]
- *times*=[2000/01/01-2004/01/01]

Se seleccionan o se alimentan los siguientes valores en las respectivas dimensiones para el segundo cubo de datos:

- *products*=todos
- *customers*=[300-300]
- *times*=[1999/01/01-1999/01/21]

Donde la dimensión a conservar es la dimensión de **Productos** por lo que se introduce el número correspondiente a esta dimensión por lo que en **Conservar Dim** se introduce un **Producto**.

Ya que se quieren los mejores dos productos en **Num de Mejores** se introduce un **2**. Así como saber si se quieren los mejores o peores, en caso de ser los mejores se **selecciona** Mejores/Peores

La respuesta muestra el comportamiento de los primeros 3 productos del primer cubo de datos y su desempeño en el segundo cubo de datos, dándose una comparación mostrada en la tabla con la eficiencia. Mientras que en la gráfica de la Ilustración 14 se observa la suma para ambos productos en los dos cubos de datos.

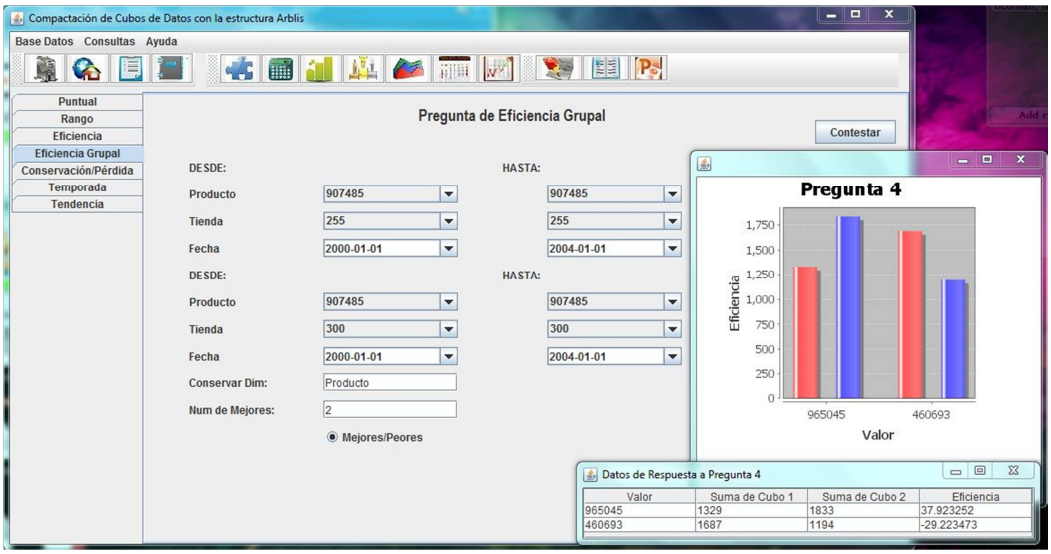


Ilustración 14. Pregunta de Eficiencia Grupal

En caso de que los que se hubiesen solicitado Mejores/Peores debería de no estar seleccionado por lo que salida a pantalla hubiese sido la de la Ilustración 15.

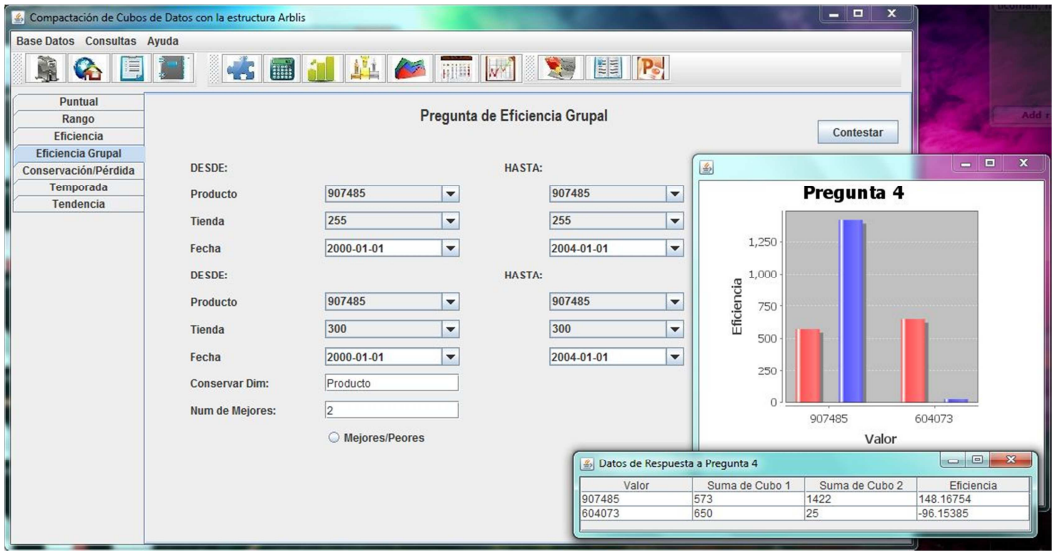


Ilustración 15. Pregunta Eficiencia Grupal Peores

6.5 Pregunta de Conservación

Para ingresar una pregunta puntual se selecciona la pestaña que dice **Conservación** en el lado izquierdo y se introducen los datos.

Una pregunta de este tipo sería "*¿Qué productos se conservan entre los mejores 3 en el periodo de 01/01/2000 hasta 01/01/2004 de la tienda 255 con respecto a la tienda 300?*".

Se seleccionan o se alimentan los siguientes valores en las respectivas dimensiones para el primer cubo de dato:

- *products*=todos
- *tienda*=[255-255]
- *times*=[2000/01/01-2004/01/01]

Se seleccionan o se alimentan los siguientes valores en las respectivas dimensiones para el segundo cubo de datos:

- *products*=todos
- *customers*=[300-300]
- *times*=[1999/01/01-1999/01/21]

Donde la dimensión a conservar es la dimensión de **Productos** por lo que se introduce el número correspondiente a esta dimensión por lo que en **Conservar Dim** se introduce **Producto**.

Ya que se quieren los mejores dos productos en **Num de Mejores** se introduce un **3**. Así como saber si se quieren los mejores o peores, en caso de ser los mejores se **selecciona** Mejores/Peores

La respuesta se puede observar en la gráfica y la tabla que muestra el producto que se conservo dentro de los primeros 3 elementos en ambos cubos de datos, por lo que la respuesta se observa en la Ilustración 16.

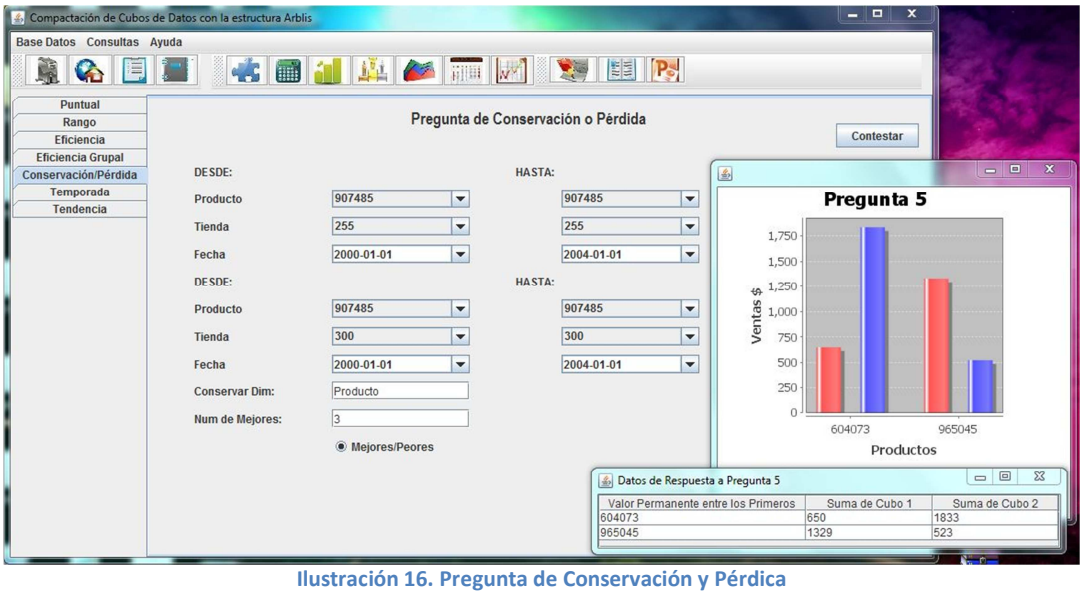


Ilustración 16. Pregunta de Conservación y Pérdica

6.6 Pregunta de Temporalidad

Para ingresar una pregunta puntual se selecciona la pestaña que dice **Temporalidad** en el lado izquierdo y se introducen los datos.

Una pregunta de este tipo sería "*¿Qué productos se conservan entre los mejores 2 en el de la tienda 255 en 2 periodos de tiempo anuales empezando a partir del 01/01/2001?".*

Se seleccionan o se alimentan los siguientes valores en las respectivas dimensiones para el primer cubo de dato:

- *products*=todos
- *tienda*=[255-255]
- *times*=[2000/01/01-2004/01/01]

Donde la dimensión a conservar es la dimensión de **Productos** por lo que se introduce el número correspondiente a esta dimensión por lo que en **Conservar Dim** se introduce **Producto**.

Ya que se quieren los mejores dos productos en **Num de Mejores** se introduce un **2**. Así como saber si se quieren los mejores o peores, en caso de ser los mejores se **selecciona** Mejores/Peores.

El número de temporadas que se quieren son dos, por lo que en **Num de Temps** se introduce un **2**.

El tipo de temporada es anual por lo que en **Tipo Temporada** se selecciona **annual.**

La respuesta se puede observar en la gráfica y la tabla que muestra el producto que se conservó dentro de los primeros 2 elementos en ambos cubos de datos, por lo que la respuesta se observa en la Ilustración 17.

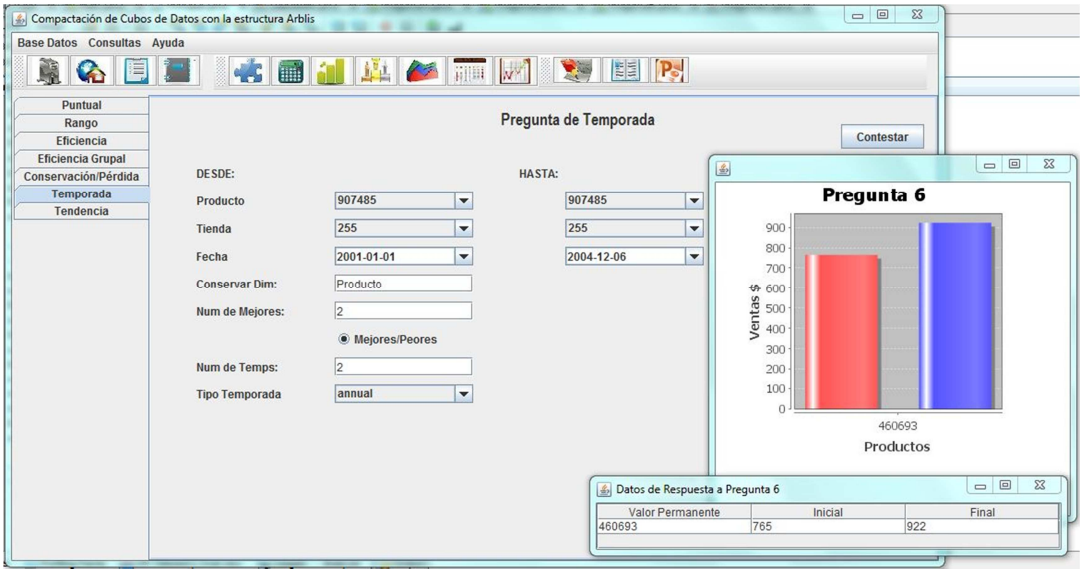


Ilustración 17. Pregunta de Temporada

6.7 Pregunta de Tendencia

Para ingresar una pregunta puntual se selecciona la pestaña que dice **Conservación** en el lado izquierdo y se introducen los datos.

Una pregunta de este tipo sería "*¿Cómo se comportan los productos de la tienda 255 en 3 periodos de tiempo anuales empezando a partir del 01/01/2001?*".

01/01/2000 a 01/01/2004 y que eficiencia muestran con respecto a la tienda 300?".

Se seleccionan o se alimentan los siguientes valores en las respectivas dimensiones para el primer cubo de dato:

- *products*=todos
- *tienda*=[255-255]
- *times*=[2000/01/01-2004/01/01]

Donde la dimensión a conservar es la dimensión de **Productos** por lo que se introduce el número correspondiente a esta dimensión por lo que en **Conservar Dim** se introduce **Producto**.

El número de temporadas que se quieren son dos, por lo que en **Num de Temps** se introduce un 3.

El tipo de temporada es anual por lo que en **Tipo Temporada** se selecciona **annual.**

La respuesta se puede observar en la gráfica que muestra las tres temporadas, y cada color representa un producto por lo que fácil observar el comportamiento de los productos en la gráfica como se ve en la Ilustración 18.

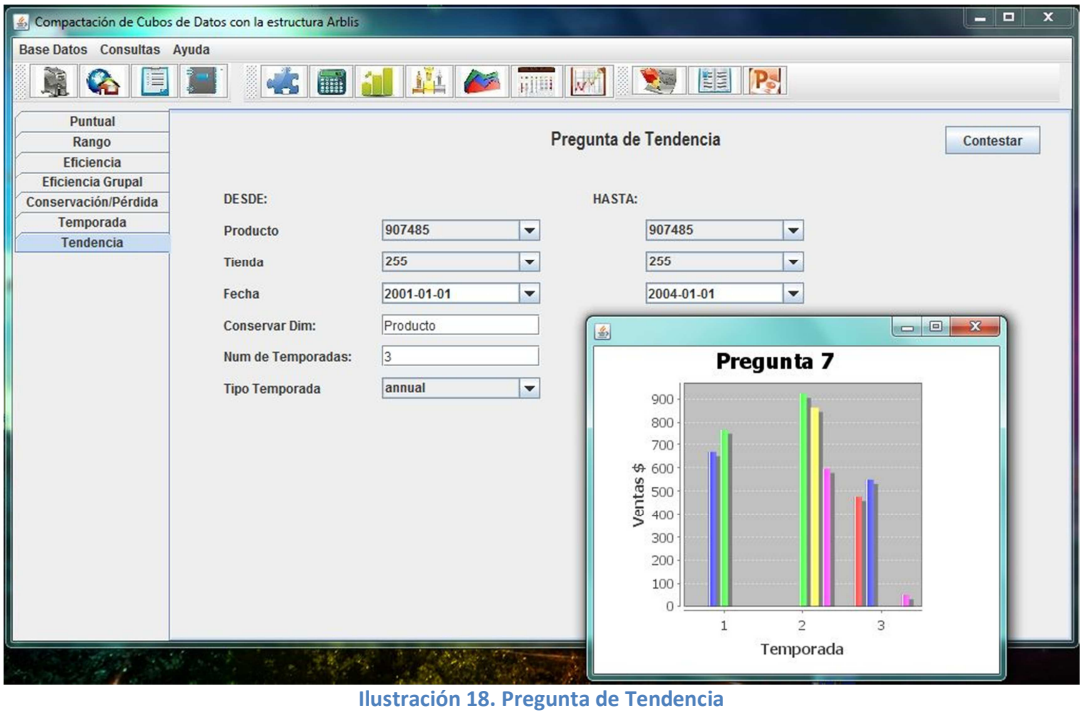


Ilustración 18. Pregunta de Tendencia

7. Ayuda

Si se requiere saber más sobre las preguntas y su implementación se selecciona la pregunta deseada en la parte superior central. En la Ilustración 19 se muestra que botón apretar para cada una de las preguntas.

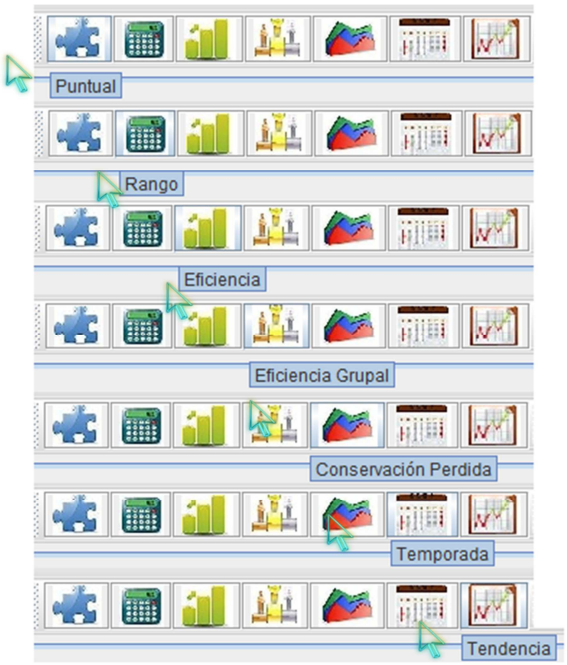


Ilustración 19. Ayuda de Preguntas

8. RESOLUCIÓN A PROBLEMAS EN LOS PROGRAMAS

Para problemas en cuanto a la ejecución del software se puede enviar un correo a:

Centro de Investigación en Computación del I.P.N.

Bella Citlali Martínez Seis

Correo electrónico bellams@gmail.com

GLOSARIO

ANTECUMEM. Prototipo de software que trabaja en modo mono-usuario, llena la estructura Arblis, captura las preguntas deseadas, realiza los correspondientes análisis de datos y obtiene los resultados.

Arblis. Estructura de datos que funge como un almacén persistente de datos de solo lectura donde se llevan a cabo búsquedas típicas de minería de datos. Por el diseño de Arblis, la consulta toma un tiempo proporcional (lineal) al tamaño de los datos. Sus respuestas son hasta 50 veces más rápidas que un manejador de bases de datos. Los datos fueron ya validados y no cambian (no hay actualización).

Base de Datos. Una colección de datos lógicamente relacionados, junto con una descripción de estos datos, que están diseñados para satisfacer las necesidades de información de una organización.

Compactación. Reducir el tamaño de datos de forma tal que estos ocupen menos espacio y que dado el dato comprimido pueda recuperarse el dato original. Existe compactación con y sin pérdida de datos: la compactación sin pérdida permite que los datos antes y después de comprimirlos son exactos en la compresión sin pérdida; la compactación con pérdida puede eliminar datos para reducir aún más el tamaño, con lo que se suele reducir la calidad, dee tal forma que una vez realizada la compresión, no se puede obtener el dato original, aunque sí una aproximación cuya semejanza con la original dependerá del tipo de compresión.

Compresión. Véase compactación.

CPU. Unidad Central de Procesos.

Cubo de Datos. Permite que los datos sean modelados y visualizados en múltiples dimensiones. Esta definida por dimensiones y hechos.

Cuboide. Diferentes niveles de agrupamiento de una lattice un cubo de datos.

Dimensión. Perspectiva o entidad con respecto a la cual la organización quiere mantener su almacén o archivo.

Disco Duro. Dispositivo de almacenamiento de datos no volátil. Normalmente un disco duro consiste en varios discos o platos. Cada disco requiere dos cabezales de lectura/grabación, uno para cada lado. Todos los cabezales de lectura/grabación están unidos a un solo brazo de acceso, de modo que no puedan moverse independientemente. Cada disco tiene el mismo número de pistas, y a la parte de la pista que corta a través de todos los discos se le llama cilindro.

Estructura AC (Estructura). Estructura basada en apuntadores a catálogos que retoma conceptos de navegabilidad de Arblis ocupando un menor espacio en disco duro y memoria principal a través de métodos de compactación.

Función monotónica. (función monótona) Función que preserva el orden dado. Este concepto primero se presentó adentro cálculo, y fue generalizado más adelante al ajuste más abstracto de teoría de la orden.

GPU. (Graphics Processing Unit) procesador dedicado exclusivamente al procesamiento de gráficos, para aligerar la carga de trabajo del procesador central en aplicaciones como los videojuegos y o aplicaciones 3D.

Jerarquía. Organiza los calores de atributos o dimensiones en diversos niveles de abstracción. Son usados en agrupamientos de múltiples niveles de abstracción.

Memoria Principal. Dispositivo donde se almacenan temporalmente tanto los datos como los programas que la CPU está procesando o va a procesar en un determinado momento. Esta clase de memoria es volátil.

Minería de Datos. Proceso de extraer información válida, previamente desconocida, comprensible y útil de bases de datos de gran tamaño y utilizar dicha información para tomar decisiones de negocio cruciales. De tal forma que descrubre reglas y patrones de los datos.

MOLAP. OLAP multidimensional.

OLAP. (Online Transaction Processing) Síntesis, análisis y consolidación dinámicas de grandes volúmenes de datos.

Optimización. Proceso de búsqueda y mejoramiento con el objetivo de obtener el mejor entre un conjunto de elementos.

ROLAP. OLAP relacional.

SQL. (Sructured Query Language) Es un lenguaje orientado a la transformación, es decir, un lenguaje diseñado para usar relaciones con el fin de transformar los datos de entrada en las salidas requeridas. El estándar SQL tiene dos componentes principales: DDL (Data Definition Languaje) y DML (Data Manipulation Language). El primero es el lenguaje para definir la estructura de la base de datos y controlar el acceso a los datos y el segundo es la manipulación de los datos al extraerlos o actualizarlos.

Tendencia. Movimiento suave de la serie a largo plazo. Cuando se observa que los datos estudiados presentan preferencia a estar de una forma u otra, es decir cuando vemos datos que tienden a elevarse en el grafico esa es una tendencia al aumento en largo plazo. En el presente documento la tendencia es usada para describir el comportamiento (variación) de un elemento.

Vista. Es el resultado dinámico de una o más operaciones relacionales que operan sobre las relaciones base para producir otra relación. Una vista es una relación virtual que no tiene por qué existir necesariamente en la base de datos, sino que puede producirse cuando se solicite por parte de un usuario concreto, generándosela en el momento de la solicitud.