CAPÍTULO V

Implementación

En este capítulo se muestra la integración de las herramientas que forman al sistema PIE, la justificación de su selección y detalles técnicos sobre la implementación del sistema. El capítulo V se encuentra organizado de acuerdo a la arquitectura propuesta en el capítulo IV: en la sección 5.1 se describe la implementación de la capa de integración, en la sección 5.2 se describe la implementación de la capa de análisis, en la sección 5.3 se describe la implementación de la capa de visualización, en la sección 5.4 se describe las optimizaciones realizadas al modelo y a la implementación y por último, en la sección 5.5 las conclusiones del capítulo.

La figura 5.1 muestra el flujo de datos a través de las 3 capas: integración, análisis y visualización que integran al sistema PIE. Los datos son extraídos de la base de datos transaccional y pasan por el proceso de ETL para poder ser almacenados en el *datawarehouse*.

Posteriormente los datos almacenados en el *datawarehouse* son explotados por la capa de análisis en donde se encuentran las técnicas de OLAP y *data mining*, por medio de las cuales se manipulará la información para ofrecer resultados interesantes al usuario final sobre su propia información.

Por último los datos son desplegados en la capa de visualización en donde el usuario puede manipular gráficamente los resultados que se le hayan entregado, esta información le servirá como base y justificación para el apoyo a la toma de decisiones dentro de la empresa.

Este flujo nos da un panorama general del funcionamiento del sistema PIE.

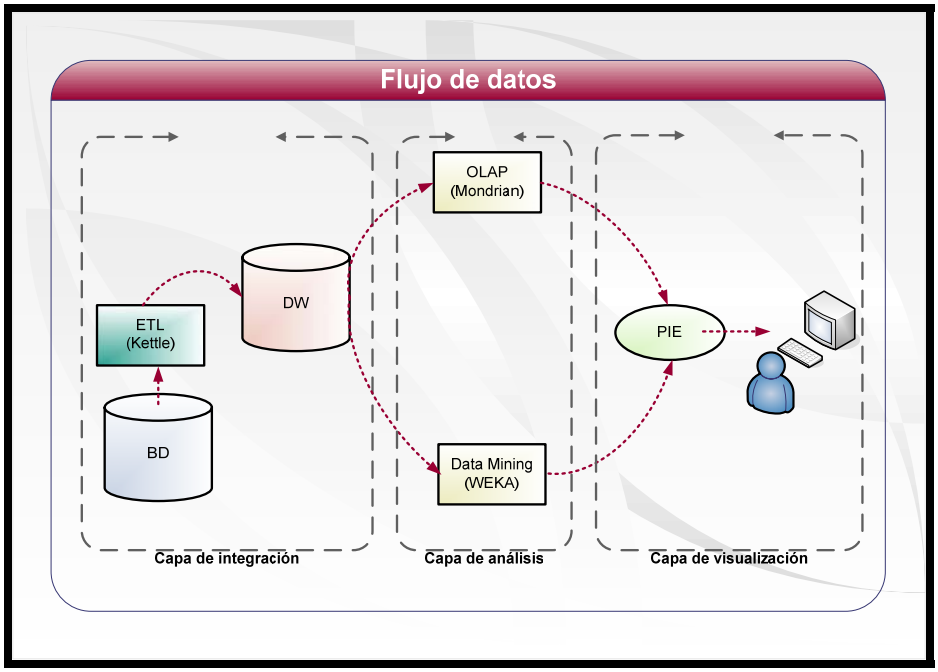

Figura 5.1 Flujo de datos del sistema PIE

En las siguientes secciones veremos la implementación de cada una de las capas más detalladamente.

5.1 Creación de capa de integración

Para la creación de la capa de integración, se extrajeron los datos de la base de datos de la PyME que se encuentra en Paradox versión 7.0 y se almacenaron en el *datawarehouse* que se

implementó en el manejador MySQL 5.0. Paradox tiene la limitante de no ser un manejador de base de datos relacional con todas sus funciones ya que, carece de algunas características como el manejo de llaves foráneas y concurrencia de usuarios, esto propicia un marco ideal porque seguramente se encontrarán escenarios similares en otras PyMEs.

Por otro lado, se decidió utilizar MySQL porque, además de ser un manejador de código abierto, es uno de los más rápidos que existen para grandes volúmenes de datos y por ello es ideal para ambientes de *datawarehousing*, aún en empresas que cuentan con algún manejador robusto como Oracle [Bezinger, 2005] [Scalzo, 2005].

De todas las herramientas vistas en el capítulo III para realizar el proceso de ETL se eligió Kettle para la integración de datos en el *datawarehouse*. Las versiones de Kettle que se utilizaron en la tesis fueron: Kettle 2.5.0 y Kettle 3.0.0 RC1. Kettle fue seleccionado por las siguientes razones:

- Permite crear transformaciones y trabajos que pueden hacer procesos lote/batch.
- Los procesos batch pueden ejecutarse desde línea de comando con los scripts que genera PAN.
- Su interfaz gráfica permite visualizar todo el flujo de la información.
- Posee una gran cantidad de fuentes de origen y destinos de almacenamiento de información.
- Su manejo de memoria, ya que permite leer y escribir por bloques datos con ayuda de memoria caché.

Por otro lado, algunos de los inconvenientes que presentó Kettle durante su uso para la implementación del *datawarehouse* fueron:

- No utiliza índices en los productos cartesianos para formar las parejas de tuplas en las tablas, es decir, por cada tupla de la tabla A hace un barrido completo de la tabla B para ver cuales tuplas forman parejas.
- Los productos cartesianos propios de Kettle son lentos y poco optimizados, es preferible utilizar los de los manejadores de bases de datos.
- Kettle 2.5.0 no es muy estable, después de varias interacciones el programa se cerraba. Sin embargo, se observaron mejoras, en este aspecto, para la versión 3.0.0 RC1.
- Es importante la configuración del *commit size*, para que no se escriban registros de manera inmediata al disco duro, pero tampoco que almacene muchos registros en memoria, ya que esto repercute de manera directa en el rendimiento del proceso que se esté realizando.

Una vez descritas las ventajas y desventajas que observamos durante el funcionamiento de Kettle, a continuación mostraremos la implementación del modelo de datos multidimensional descrito en la sección 4.3. En la tabla 5.1 se pueden observar los tipos de datos que se manejan para los atributos del modelo y algunas especificaciones. Cabe mencionar que a las tablas de tiempo_venta y descuentos, se les genera su propia llave con un número entero como lo establece el modelo multidimensional.

Nombre de la tabla	Nombre de la Columna	Tipo de dato	Longitud	Nulos	Primary Key	Foreign Key
Clientes	Cliente_No	DOUBLE	-	NO	SI	NO
	Nombre	VARCHAR	120	NO	NO	NO
	Estado	VARCHAR	20	NO	NO	NO
	Ciudad	VARCHAR	40	NO	NO	NO
Empleados	Empleado_No	DOUBLE	-	NO	SI	NO
	Nombre	VARCHAR	120	NO	NO	NO
Productos	Producto_No	DOUBLE	-	NO	SI	NO
	Departamento	VARCHAR	30	NO	NO	NO
	Categoría	VARCHAR	30	NO	NO	NO
	Unidad	VARCHAR	4	NO	NO	NO
	Descripción	VARCHAR	255	NO	NO	NO
Descuentos	Descuento_No	DOUBLE	-	NO	SI	NO
	Porcentaje	INT	-	NO	NO	NO
Tiempo_venta	Fecha_No	INT	-	NO	SI	NO
	Año	VARCHAR	4	NO	NO	NO
	Mes	VARCHAR	2	NO	NO	NO
	Día	VARCHAR	2	NO	NO	NO
Venta_fact	Venta_No	DOUBLE	-	NO	SI	NO
	Cliente_No	DOUBLE	-	NO	SI	SI
	Producto_No	DOUBLE	-	NO	SI	SI
	Fecha_No	DOUBLE	-	NO	SI	SI
	Descuento_No	DOUBLE	-	NO	SI	SI
	Empleado_No	DOUBLE	-	NO	SI	SI
	Cantidad	FLOAT	-	NO	NO	NO
	Precio	FLOAT	-	NO	NO	NO
	Costo_Unitario	FLOAT	-	NO	NO	NO
	Costo	FLOAT	-	NO	NO	NO

Tabla 5.1 Tipos de datos para la implementación del *datawarehouse*

El *datawarehouse* fue alimentado siguiendo las transformaciones (proceso ETL) que se describen en la tabla 5.2 y en la figura 5.2 podemos ver un ejemplo de las transformaciones.

Transformación	Tabla origen Paradox (P) o MySQL (M)	Tabla destino MySQL (M)	Archivo	Observaciones
Dimensión de clientes	Cliente/(P)	Clientes/(M)	clientes_dimension.ktr	--
Dimensión de empleados	Empleado/(P)	Empleados/(M)	empleados_dimension.ktr	--
Dimensión de productos	Mnlus/(P)	Productos/(M)	productos_dimension.ktr	--
Dimensión de descuentos	Descuento/(P)	Descuentos/(M)	descuentos_dimension.ktr	--
Dimensión de tiempo	V012007/(P)	Tiempo_venta/(M)	tiempo_dimension.ktr	Un ejemplo puede verse en la figura 5.2

Sumar productos iguales	DV012007/(P)	Suma_productos_ iguales/(M)	Suma_productos_ iguales_venta.ktr	Sumariza los productos iguales que aparecen en distinto orden en una misma venta, la tabla resultante es una tabla temporal
Venta detalle	V012007, DV012007/(P)	Venta_detalle/(M)	Juntar_venta_ ventadetalle_ mysql.ktr	Agrupa las trablas de origen en una sola porque el los datos de la venta y su detalle se encuentran en 2 tablas diferentes, la tabla resultantes es una tabla temporal.
Venta detalle y productos iguales	Suma_productos_iguales, venta_detalle/(M)	Ventadetalle_ productosiguales/(M)	Ventadetalle_ productosiguales.ktr	Agrupa las tablas de origen para tener una tabla consolidada, la tabla resultante es una tabla temporal.
Venta_fact	Ventadetalle_ productos_iguales, tiempo_venta, descuentos/(M)	Venta_fact/(M)	Venta_fact.ktr	La tabla resultante es la tabla hecho del modelo multidimensional.

Tabla 5.2 Transformaciones de alimentación del datawarehouse

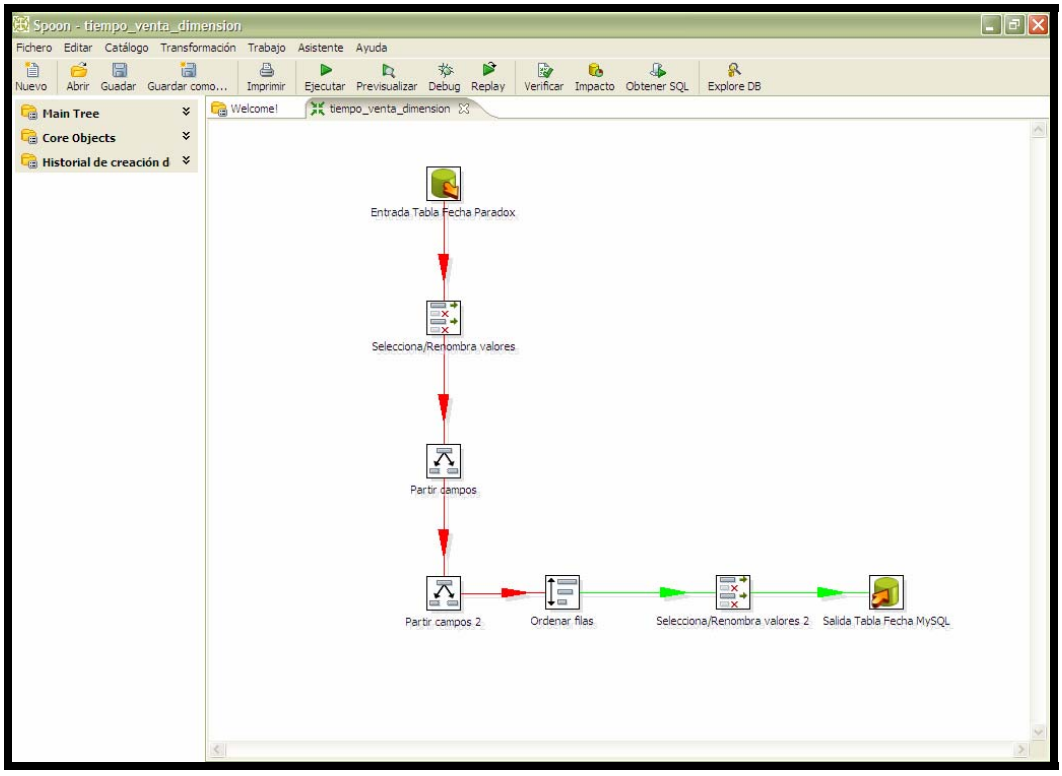

Figura 5.2 Transformación de la dimensión tiempo.

Las figuras de las demás transformaciones se encuentran en el apéndice B al final del documento.

En cada una de las transformaciones se verifica la consistencia de los datos, así como su nombre y tipo para garantizar la estandarización de los mismos.

Para la automatización de las transformaciones mencionadas anteriormente, se diseñó un trabajo (*job*), como se puede observar en la figura 5.3, en el que se ejecutan de manera secuencial las transformaciones siguiendo el orden mencionado, agregando un paso adicional antes de crear la tabla hecho, que es un script de SQL que agrega aquellos datos que existen en la tabla hecho pero que han sido dados de baja de las dimensiones.

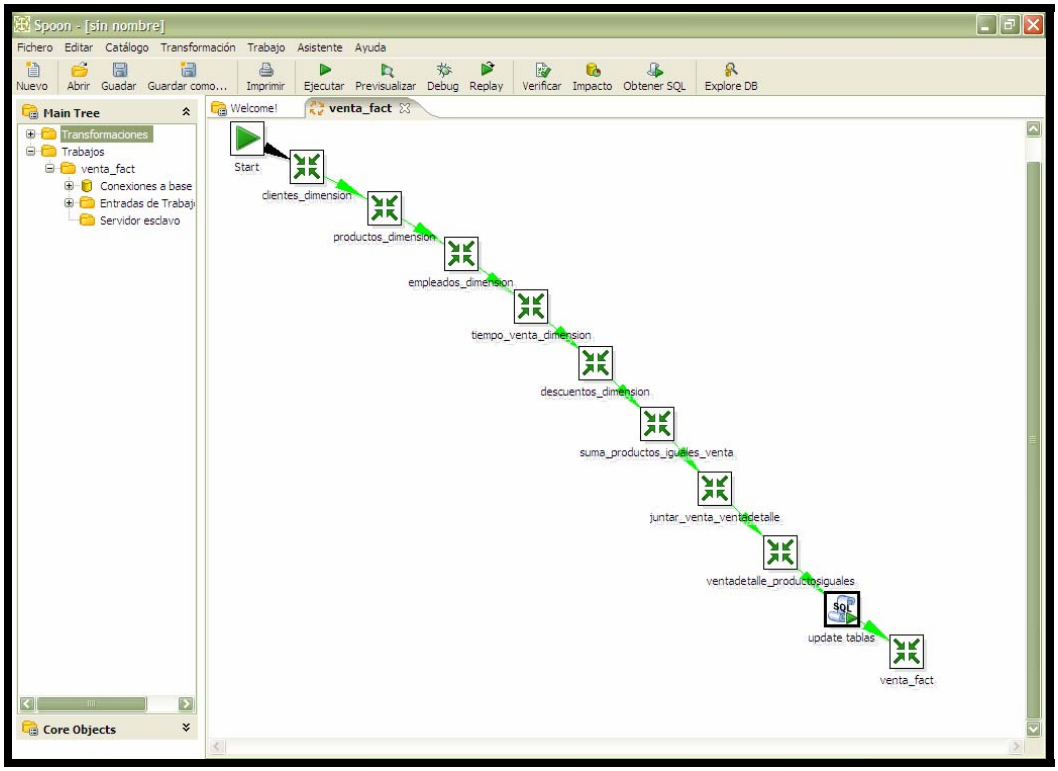

Figura 5.3 Trabajo de automatización del proceso ETL.

El tiempo de ejecución promedio de las transformaciones para la creación del *datawarehouse* es de 4 minutos aproximadamente, dando un total de realización de todo el trabajo de 30 minutos.

5.2 Creación de capa de análisis

La capa de análisis está conformada por dos componentes, el análisis de OLAP implementado con *Mondrian* y el análisis de *data mining* implementado con *WEKA.*

5.2.1 Mondrian

De las herramientas vistas en el capítulo 3 para aplicar OLAP a los datos almacenados en el *datawarehouse* se escogió Mondrian.

Las principales ventajas de Mondrian provienen de su implementación ROLAP, ya que no tiene que generar cubos estáticos ahorrando el tiempo que cuesta generarlos y la memoria que ocupan. Tiene la posibilidad de utilizar los datos que se encuentran en la base de datos, de esta manera se trabaja con los datos actualizados [Mondrian, 2007].

Pese a que tradicionalmente los sistemas de OLAP implementados con MOLAP tienen una cierta ventaja de rendimiento, el uso de caché y de tablas agregadas que tiene Mondrian, hacen que se puedan obtener muy buenos rendimientos con él, sin perder las ventajas del modelo ROLAP.

Para la elaboración del documento XML que representa al cubo de OLAP utilizamos la herramienta *CubeDesigner*, descrita en el capítulo III, siguiendo una serie de 5 pasos que se describen a continuación:

1. Se selecciona la localización del *datawarehouse* y se nombra el cubo que vamos generar en este caso *ventas*, como vemos en la figura 5.4.

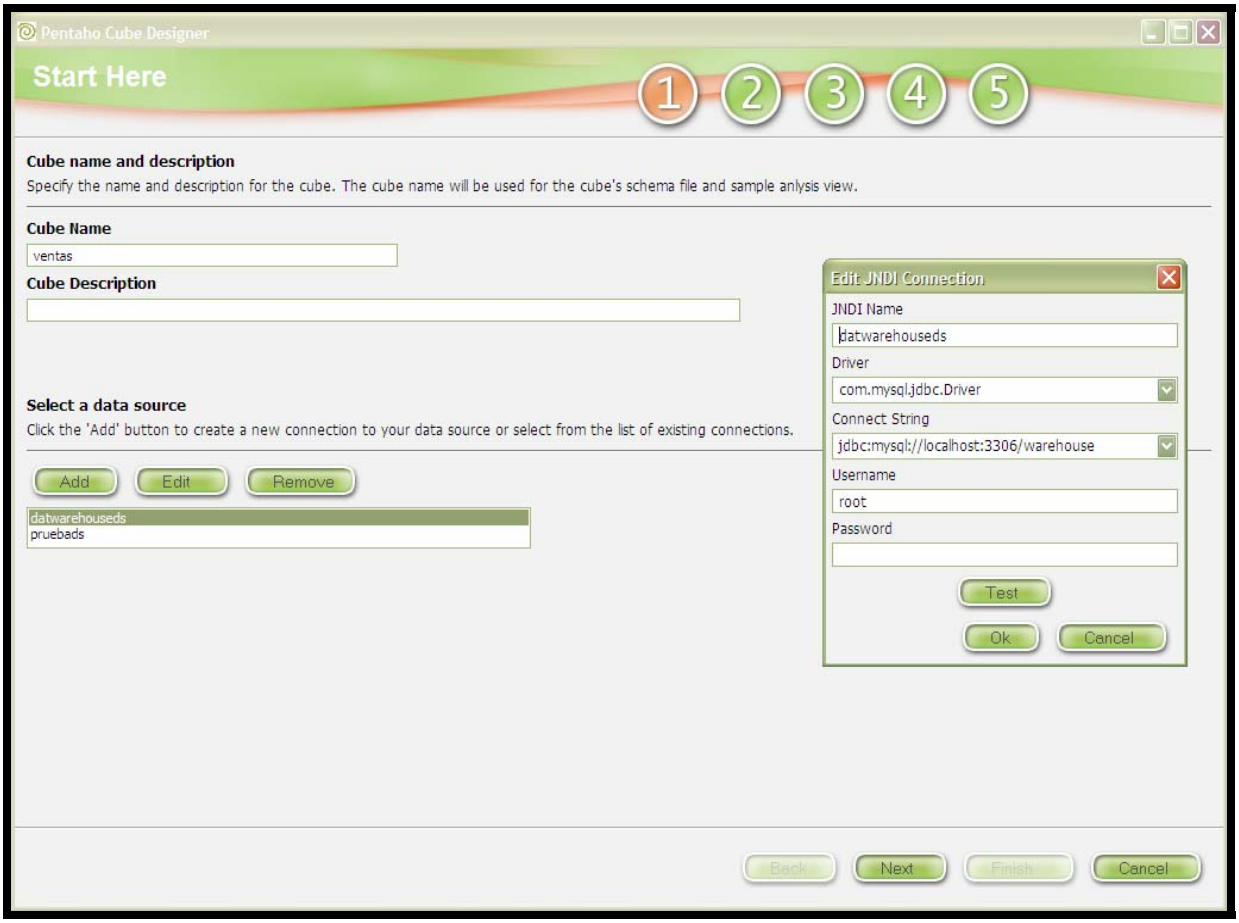

Figura 5.4 Localización del datasource.

2. Se arrastran la tabla hecho y las tablas dimensión, al área de modelado y se seleccionan los atributos de cada dimensión, que vamos a manejar en cubo, como se ve en la figura 5.5.

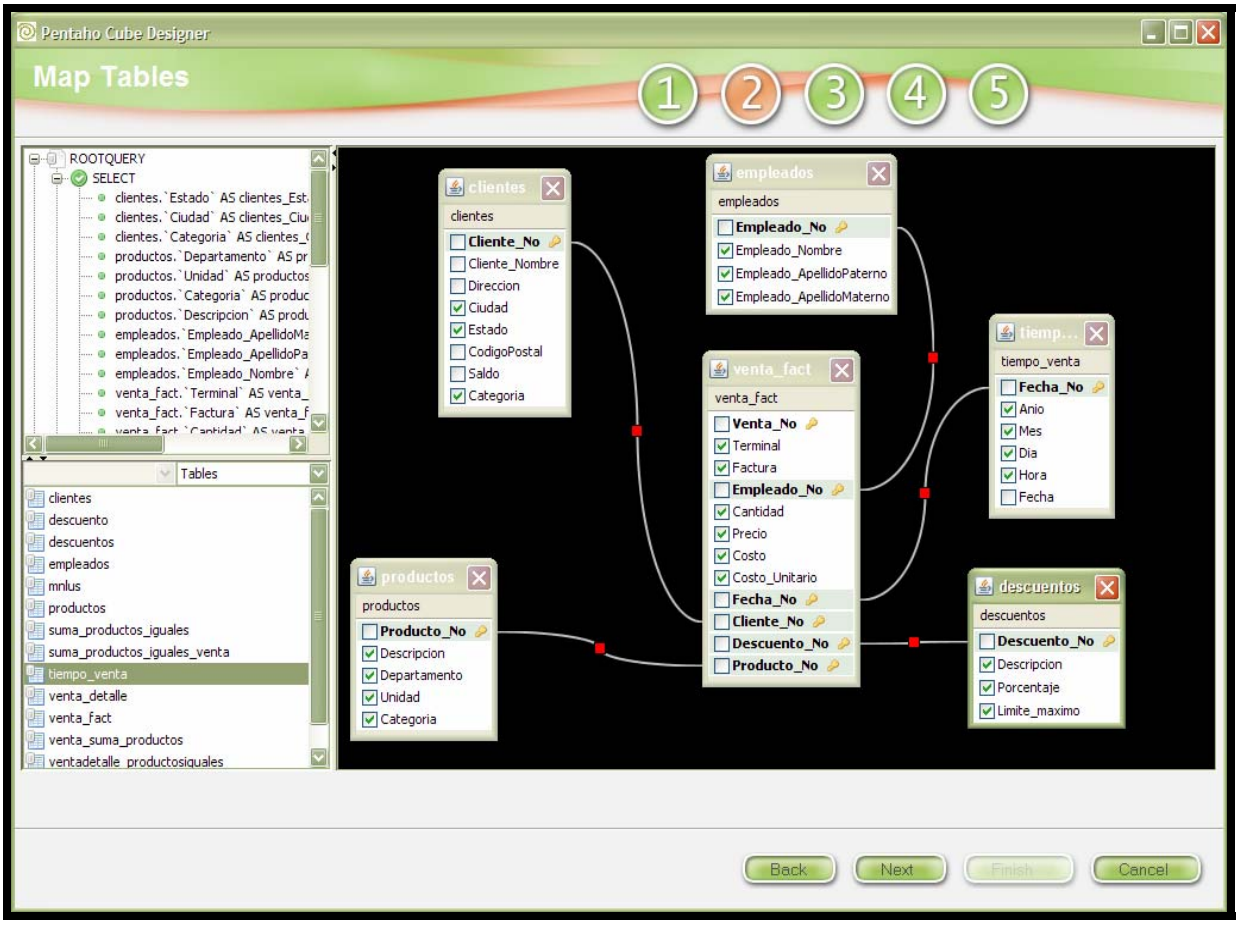

Figura 5.5 Selección de tablas y atributos

3. Se elige la tabla hecho, se seleccionan los atributos que fungirán como medidas o funciones de agregación, y se especifica cúal es el tipo de agregación que deseamos, en este caso la sumatoria como podemos ver en la figura 5.6.

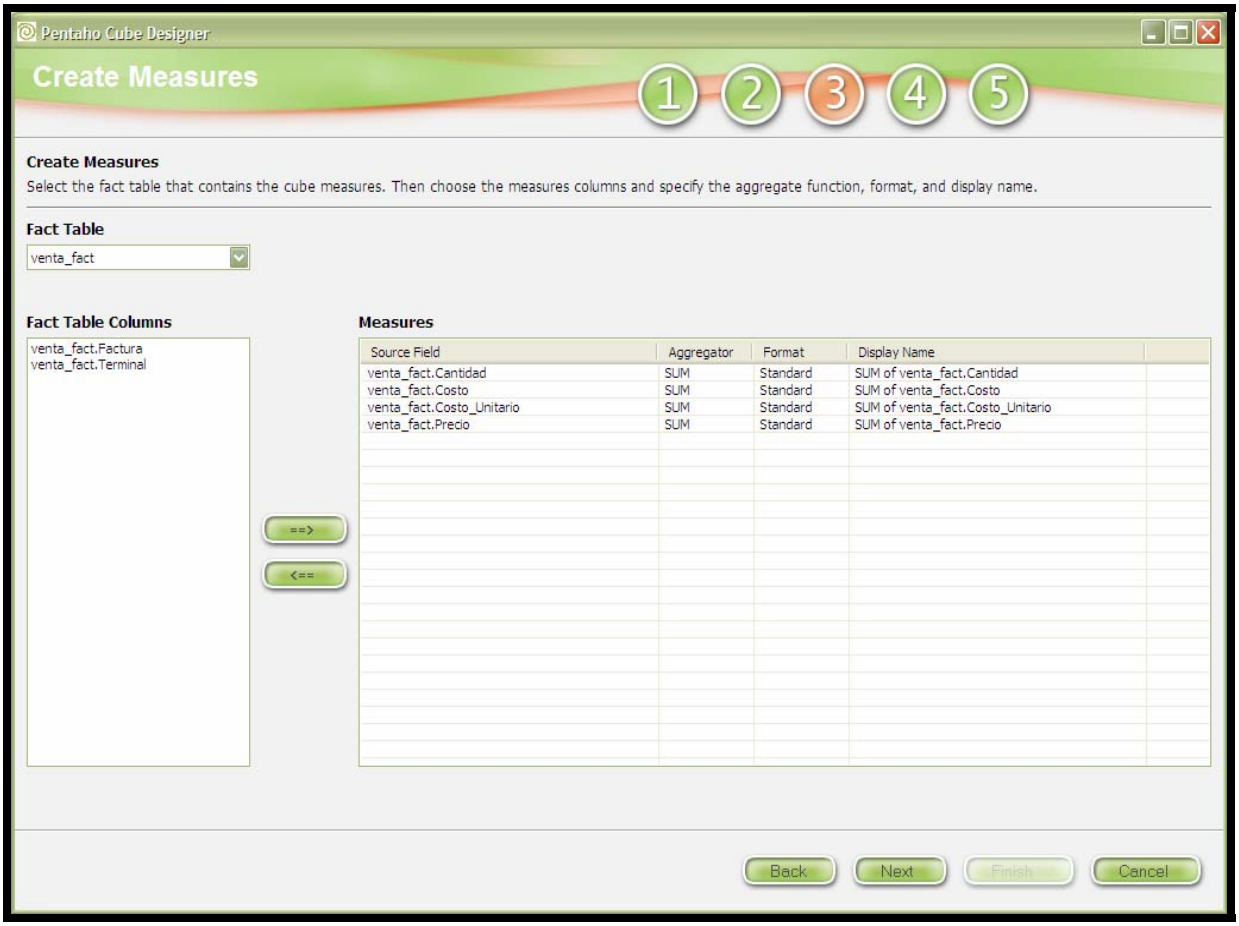

Figura 5.6 Selección de funciones de agregación.

4. Se definen las jerarquías de las dimensiones, en este paso podemos ver una característica peculiar de Mondrian, porque nos permite tener jerarquías a partir de un esquema denormalizado. Es decir, crear jerarquías entre los diversos atributos de una misma dimensión, lo que elimina la necesidad de usar estrictamente copos de nieve para manejar jerarquías, como se ve en la figura 5.7.

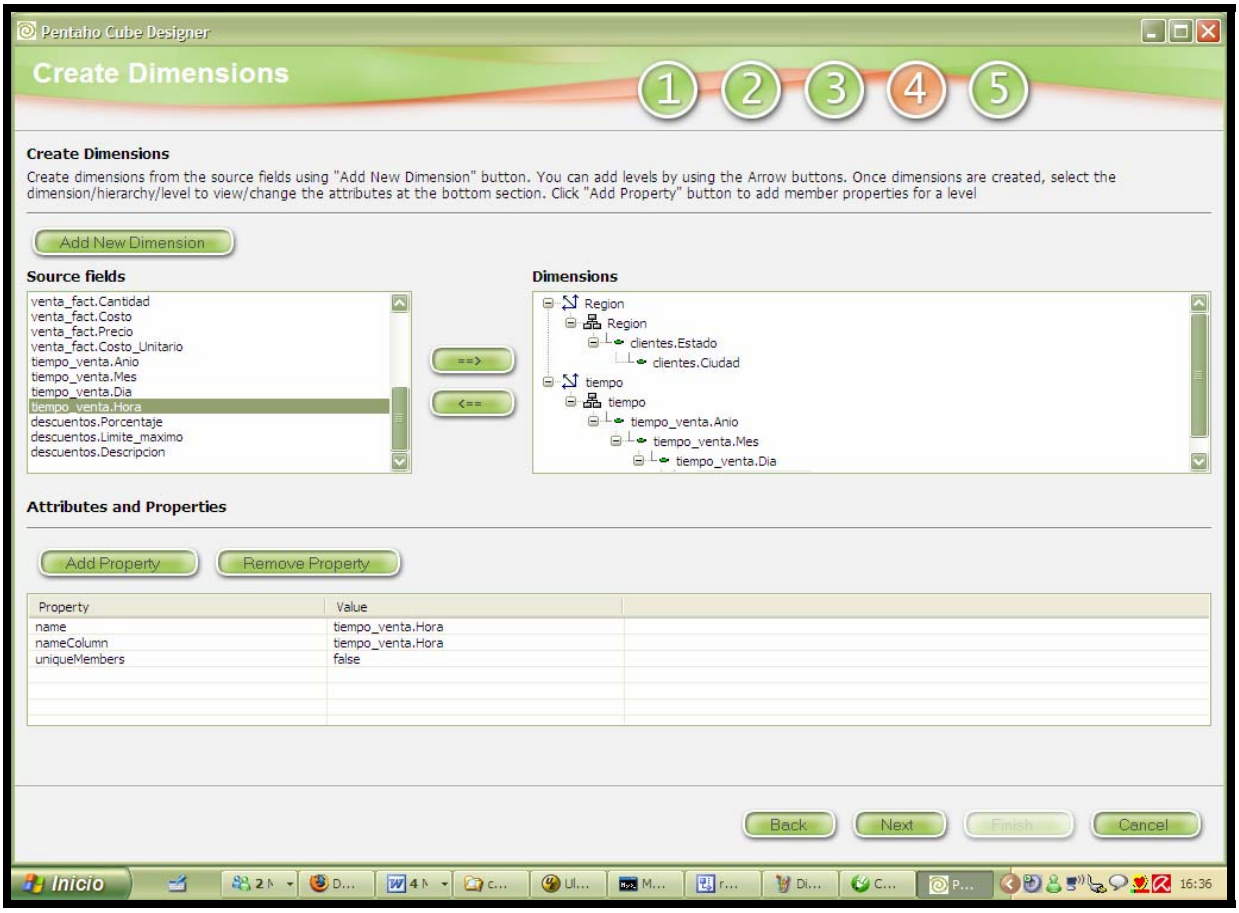

Figura 5.7 Definición de las jerarquías de las dimensiones.

5. Por último ya que se obtuvo el archivo XML, se guarda en la localización deseada, como vemos en la figura 5.8.

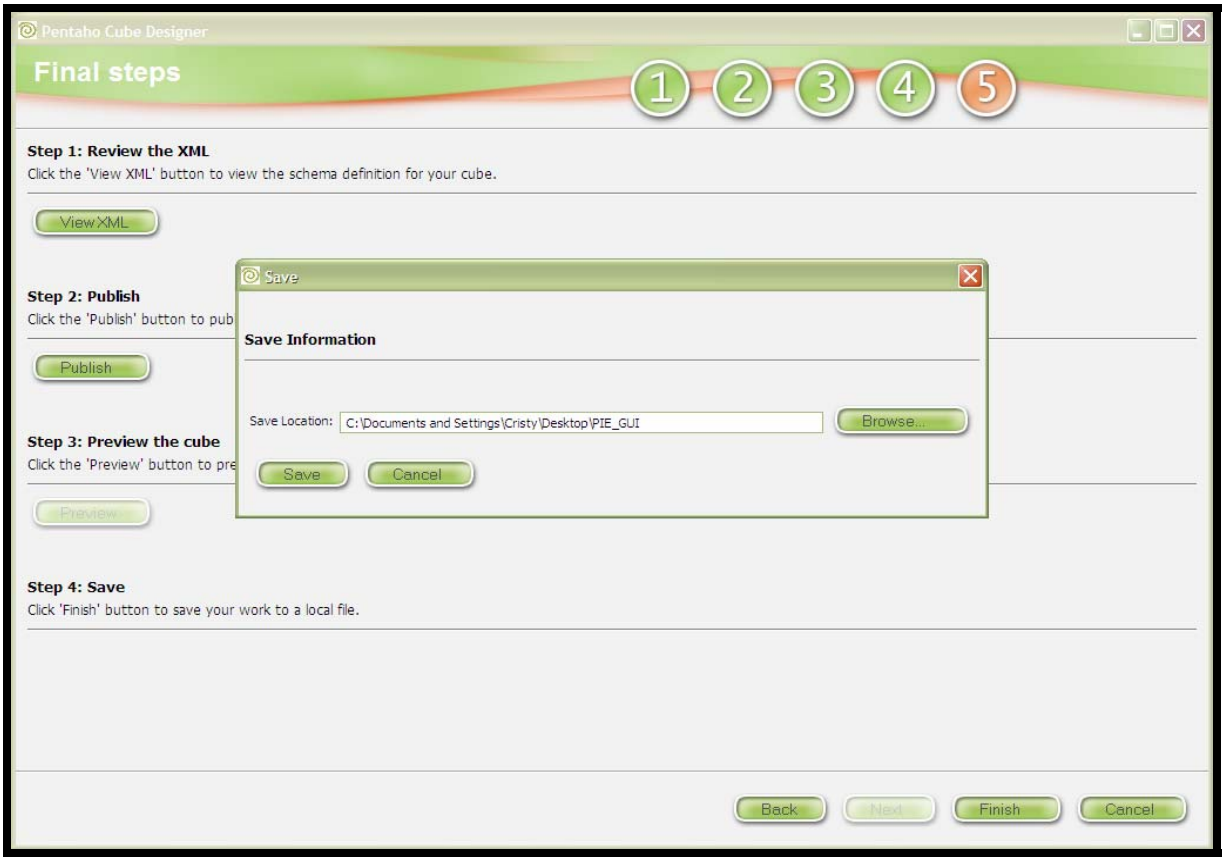

Figura 5.8 Guardar el documento XML.

En la figura 5.9 podemos ver el documento XML que obtuvimos de los pasos anteriores el cual fue diseñado para responder a las consultas que se apliquen al modelo multidimensional mencionado en la sección 4.3.

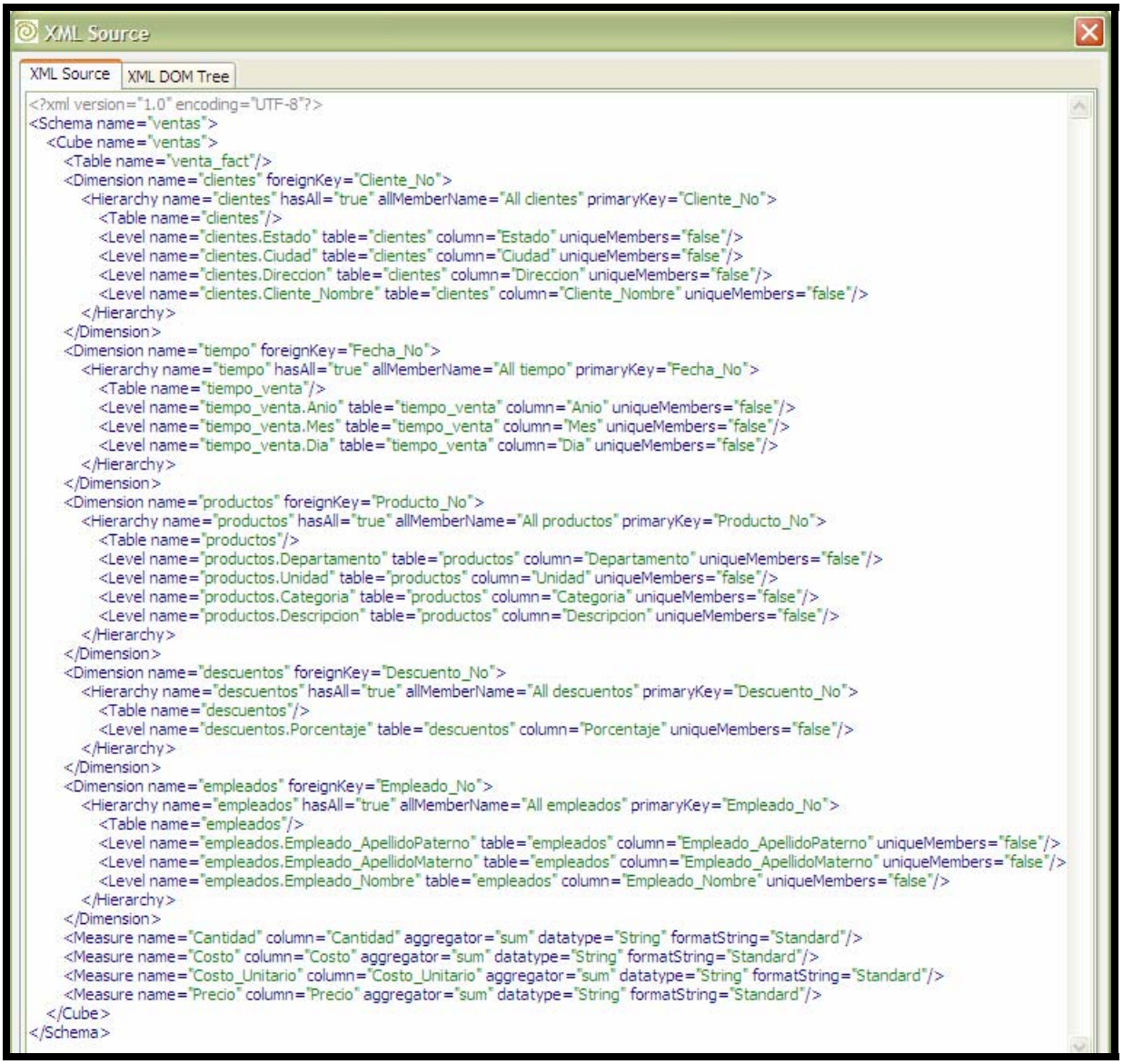

Figura 5.9 Documento XML que representa al cubo de OLAP.

Como podemos observar se define primero el nombre del esquema y posteriormente el nombre del cubo que se va a diseñar. Se especifica cual es el nombre de la tabla hecho *venta_fact*, que debe corresponder al nombre de la tabla hecho que se encuentra en la base de datos, el cual se puede verificar en el modelo de datos multidimensional.

En el archivo se definen 5 dimensiones con el tag <*Dimension*>, las cuales corresponden a las tablas dimensión del modelo de datos.

Posteriormente para cada dimensión se define su jerarquía mediante de los tags <*Level*> y también se indica cual es el nombre de la tabla en donde se encuentra la información, dichos nombres también deben corresponder a los nombre de las tablas en la base de datos, como se ven en el modelo de datos.

Por último se definen las medidas con el tag <*Measures*> y se especifica el tipo de agregación, en este caso todas las agregaciones son sumatorias, pero también se pueden elegir promedios, conteos, mínimos y máximos.

Como se mostró en el archivo XML, para algunas de las dimensiones se establecieron jerarquías basándose en los atributos de la misma dimensión, en la tabla 5.3 se muestran las jerarquías definidas para cada dimensión.

Dimensión	Jerarquías
Clientes	Estado Ciudad Nombre
Productos	Departamento Categoría Unidad Descripción
Empleados	Nombre
Descuentos	Porcentaje
Tiempo	Año Mes Día Hora

Tabla 5.3 Jerarquías de las dimensiones

Ya que se ha implementado el cubo de OLAP, podemos definir las consultas predeterminadas en lenguaje MDX, las cuales responderán a los grupos de consultas definidos en el capítulo 4. Se definieron 5 consultas predeterminadas que se muestran en la tabla 5.4.

Pregunta	Sentencia en lenguaje MDX
¿Cuál es el importe total de ventas del departamento B?	select {[Measures].[Cantidad (unidades)], [Measures].[Costo Promedio], [Measures].[Precio (pesos)]} ON COLUMNS, NON EMPTY {[productos].[Departamento].Members} ON ROWS from [ventas]
¿Cuál fue el importe total vendido del departamento A por el empleado B?	select {[Measures].[Precio (pesos)], [Measures].[Cantidad (unidades)], [Measures].[Costo Promedio]} ON COLUMNS, {[empleados.empleados].[All empleados].[#null], [empleados.empleados].[All empleados].[CARVAJAL], [empleados.empleados].[All empleados].[CASTRO], [empleados.empleados].[All empleados].[HERNANDEZ], [empleados.empleados].[All empleados].[LOYO], [empleados.empleados].[All empleados].[MARTINEZ], [empleados.empleados].[All empleados].[MORA], [empleados.empleados].[All empleados].[NO DEFINIDO], [empleados.empleados].[All empleados].[TORRES MARIN]} ON ROWS from [ventas]
¿Cuál fue el importe total comprado del departamento A por el cliente B?	select {[Measures].[Cantidad (unidades)], [Measures].[Costo Promedio], [Measures].[Precio (pesos)]} ON COLUMNS, {[clientes_region.clientes_region].[All clientes_region].[MORELOS], [clientes_region.clientes_region].[All clientes_region].[PUEBLA], [clientes_region.clientes_region].[All clientes_region].[VERACRUZ]} ON ROWS from [ventas]
¿Cuántos productos del departamento A compraron los clientes de la región B?	select Crossjoin({[Measures].[Cantidad (unidades)], [Measures].[Costo Promedio], [Measures].[Precio (pesos)]}, {[clientes_region.clientes_region].[All clientes_region].[MORELOS], [clientes_region.clientes_region].[All clientes_region].[PUEBLA], [clientes_region.clientes_region].[All clientes_region].[VERACRUZ]}) ON COLUMNS, {[productos.productos].[All productos].[ABARROTES], [productos.productos].[All productos].[DEPARTAMENTAL], [productos.productos].[All productos].[DULCES], [productos.productos].[All productos].[FRUTAS Y LEGUMBRES], [productos.productos].[All productos].[LACTEOS], [productos.productos].[All productos].[MEDICAMENTOS], [productos.productos].[All productos].[PERFUMERIA], [productos.productos].[All productos].[SALCHICHONERIA], [productos.productos].[All productos].[VINOS]} ON ROWS from [ventas]
¿Cuántos productos del departamento A se vendieron por el empleado B?	select Hierarchize({([Measures].[Cantidad (unidades)], [empleados.empleados].[All empleados]), ([Measures].[Costo Promedio], [empleados.empleados].[All empleados]), ([Measures].[Precio (pesos)], [empleados.empleados].[All empleados])}) ON COLUMNS, {[productos.productos].[All productos].[ABARROTES], [productos.productos].[All productos].[DEPARTAMENTAL], [productos.productos].[All productos].[DULCES], [productos.productos].[All productos].[FRUTAS Y LEGUMBRES], [productos.productos].[All productos].[LACTEOS], [productos.productos].[All productos].[MEDICAMENTOS], [productos.productos].[All productos].[PERFUMERIA], [productos.productos].[All productos].[SALCHICHONERIA], [productos.productos].[All productos].[VINOS]} ON ROWS from [ventas]

Tabla 5.4 Sentencias en lenguaje MDX para las consultas predefinidas.

Es importante recordar que las consultas en MDX son mapeadas posteriormente a lenguaje SQL, para obtener los datos requeridos desde el *datawarehouse* que se encuentra implementado en un manejador de base de datos relacional. Un ejemplo de este mapeo es el que mostramos en la figura 5.10.

La sentencia en MDX:

```
select {[Measures].[Cantidad (unidades)], [Measures].[Costo Promedio],
[Measures].[Precio(pesos)]} ON COLUMNS, NON EMPTY
{[productos].[Departamento].Members} ON ROWS from [ventas]
```

Se mapea en las siguientes sentencias SQL:

```
select `productos`.`Departamento` as `c0` from `productos` as `productos`
group by `productos`.`Departamento` order by
ISNULL(`productos`.`Departamento`), `productos`.`Departamento` ASC

select count(distinct `productos`.`Departamento`) as `c0` from `productos` as
`productos`

select `productos`.`Departamento` as `c0`, sum(`venta_fact`.`Cantidad`) as
`m0`, sum(`venta_fact`.`Costo`) as `m1`, sum(`venta_fact`.`Precio`) as `m2`
from `productos` as `productos`, `venta_fact` as `venta_fact` where
`venta_fact`.`Producto_No` = `productos`.`Producto_No` group by
`productos`.`Departamento`
```

Figura 5.10 Mapeo de sentencias en lenguaje MDX a lenguaje SQL.

La última parte de OLAP está conformada por el cliente. En el capítulo 3 mencionamos dos de los clientes de Mondrian más importantes: JPivot y JRubik.

En las figura 5.11 se encuentra un ejemplo de los resultados que entrega JPivot.

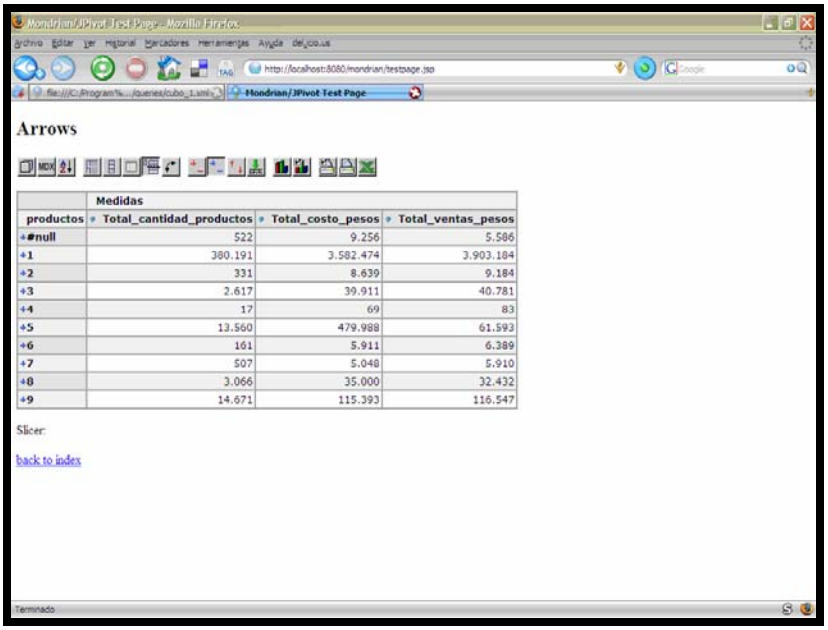

Figura 5.11 Resultados de OLAP en JPivot.

Y en la figura 5.12 podemos ver un ejemplo de los resultados que entrega JRubik.

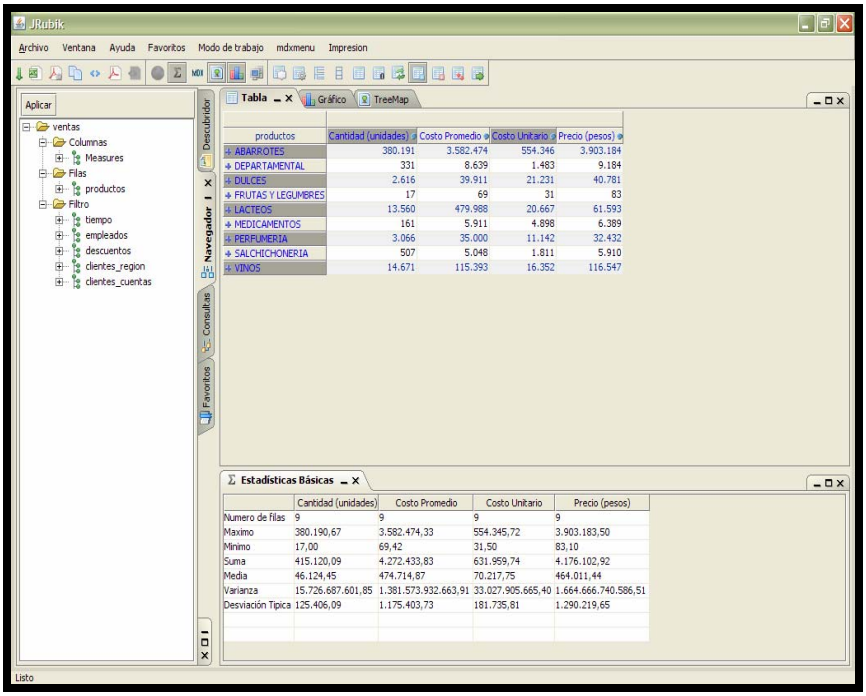

Figura 5.12 Resultados de OLAP en JRubik.

En la tesis estuvimos trabajando con ambos clientes, sin embargo escogimos JRubik como cliente para Mondrian, debido a que tiene una interfaz mucho más intuitiva y amigable que JPivot, esto es importante considerando el tipo de usuario que utilizará el sistema.

Se observó mejor tiempo de respuesta en JRubik que en JPivot, y dado que el programa de administración de la PyME (SAN) tiene una interfaz de escritorio tiene mucho más sentido que el sistema PIE también tenga una interfaz de escritorio.

Por otro lado las opciones extras que tiene JRubik, permiten la presentación de resultados, a través de diferentes gráficos, lo que facilita de manera considerable la interacción con el usuario. Los resultados de las consultas presentados de manera gráfica facilitan la comprensión e interpretación de los mismos.

Para agregar las consultas predeterminadas al sistema PIE, se debe modificar el archivo XML que contiene el índice de las consultas *Alias.xml* y posteriormente generar el documento XML de la consulta multidimensional con la sentencia en lenguaje MDX, las propiedades con las que se quiere que aparezca el reporte, como se muestra en la figura 5.13

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<RubikMdx>
  select {[Measures].[Cantidad (unidades)], [Measures].[Costo Promedio], [Measures].[Precio (pesos)]}
ON COLUMNS,
NON EMPTY {[productos].[Departamento].Members} ON ROWS from [ventas]

  <Alias>Totales por depto</Alias>
  <RubikConnection>
    <DataSourceInfo>Provider=mondrian;JdbcDriver=com.mysql.jdbc.Driver;Jdbc=jdbc:mysql://localhost/warehouse;JdbcUser=root;
    JdbcPassword=;Catalog=file:./data/resources/catalogs/pie_cubo.xml;</DataSourceInfo>
  </RubikConnection>
  <View plugin="es.aeat.eett.rubik.tableRubik@tableRubikViewMain">
    <TableSetting modeNavegacion="0" showProperties="false" mondrianDrillThrough="true" showMeasuresRow="true" showParentsRow="false"
    identRow="true" modeHierarchyHeaderRow="0" modeRowHeaderRow="1" modeMemberSpanRow="3" modeHeaderSpanRow="3" showMeasuresCol="false
    showParentsCol="false" identCol="false" modeHierarchyHeaderCol="1" modeMemberSpanCol="3" modeHeaderSpanCol="3" />
  </View>
  <Levels />
</RubikMdx>
```

Figura 5.13 Documento XML para inserción de consultas MDX

El tiempo de ejecución promedio observado para cada consulta es de 3 segundos aproximadamente.

5.2.2 WEKA

De las herramientas vistas en el capítulo 3 para aplicar algoritmos de *data mining* a los datos almacenados en el *datawarehouse* se escogió Weka por las siguientes razones:

- Posee el algoritmo *Apriori.*
- Puede manejar grandes volúmenes de datos.
- Se puede integrar a otras aplicaciones o acceder a sus métodos a través de líneas de comando.
- Puede tomar los datos desde cualquier fuente con archivos de datos.

Algunos de los inconvenientes que presentó Weka durante su funcionamiento fueron los siguientes:

- Maneja todos los conjuntos de datos que está analizando en memoria, lo que hace que sus requerimientos de memoria en ocasiones puedan ser muy grandes.
- No puede conectarse con un manejador de bases de datos relacional para extraer los datos a minar directamente desde él, se tienen que usar archivos intermedios.

Ya que justificamos la selección de Weka, para el análisis de *data mining* podemos proceder a aplicar los algoritmos a nuestros datos almacenados en el *datawarehouse.*

Para poder aplicar los métodos en Weka, la información debe estar en un formato llamado *.arff* (*Attribute-Relation File Format*) que contiene la información de los atributos con sus posibles valores y las instancias de estos, como podemos ver la figura 5.14. En nuestro caso los atributos son los productos y las instancias son las ventas y clientes.

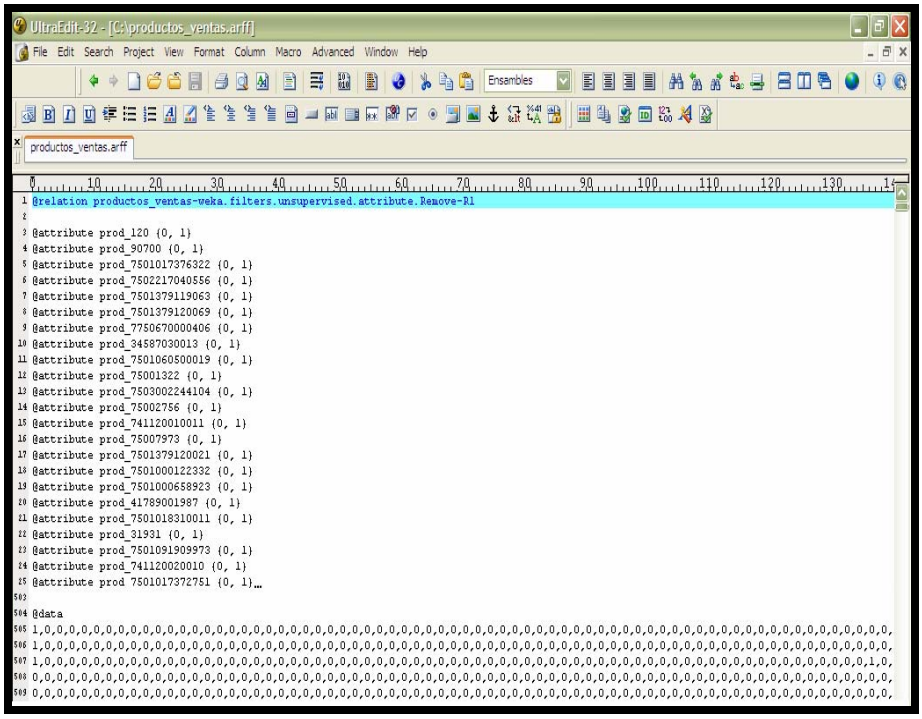

Figura 5.14 Formato de archivo *.arff*

Como se muestra el formato del archivo *.arff* hace referencia al formato de la tabla que se definió en la sección 4.6.

Se creó una aplicación llamada *ArffGenerator* que trabaja en línea de comandos para poder extraer la información de la base de datos y representarla en el formato *arff*. El *ArffGenerator* extrae únicamente los 500 productos más vendidos, ya que no es relevante aplicar el algoritmo a todos los productos existentes, esto solamente ocasionaría una sobrecarga innecesaria al modelo y las reglas que obtienen de la aplicación del algoritmo sobre los 500 productos y el total de los productos son extremadamente similares.

Como se mencionó en el diseño, se realizaron 4 archivos para aplicar el algoritmo de *data mining* de reglas de asociación *Apriori* dividiendo los datos en:

- Ventas al menudeo vs. productos
- Ventas al mayoreo vs. productos
- Clientes al menudeo vs. productos
- Clientes al mayoreo vs. Productos

Cumpliendo con lo establecido en el diseño en la sección 4.1, se manipularon los parámetros del algoritmo para poder obtener el mayor número de reglas posibles en un tiempo aceptable y consumiendo no más de 512Mb de memoria RAM.

Posteriormente se procedió a aplicar el algoritmo a cada uno de los distintos archivos, generando un archivo con las reglas de asociación resultantes, como se ve en la figura 5.15.

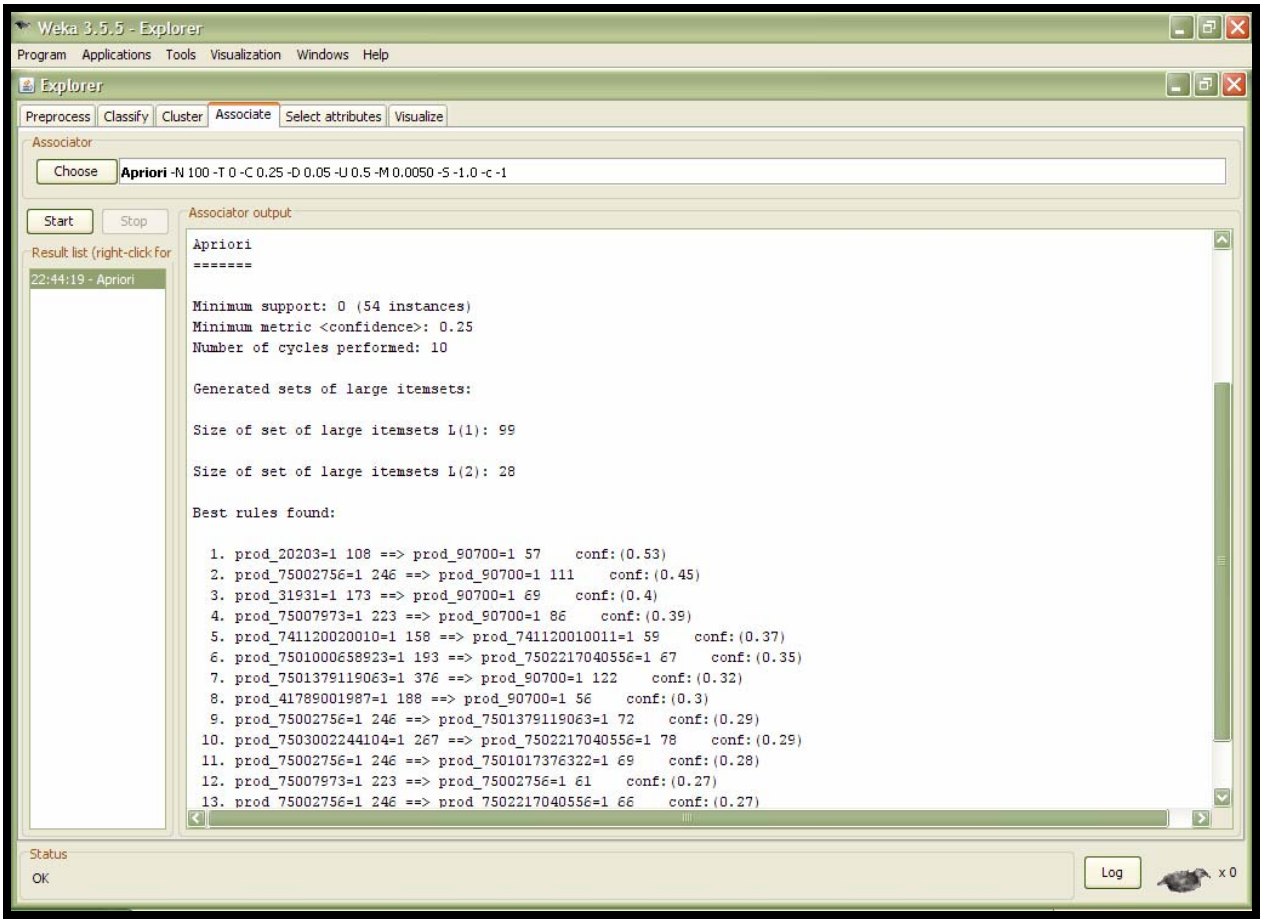

Figura 5.15 Reglas de asociación resultantes en Weka.

Finalmente, el archivo de resultados pasa por una aplicación llamada *ReportFormatter*, la cual reemplaza en el archivo de resultados, los códigos de los productos con su descripción y da un mejor formato a las reglas resultantes, con el fin de hacerlas más comprensibles e interpretables para el usuario. El listado de reglas resultantes se encuentra en el apéndice C. El tiempo de ejecución promedio del algoritmo *Apriori* para cada archivo es de 13.5 minutos.

5.3 Ambiente integrado

Como se busca crear un ambiente integrado que implemente las técnicas de la inteligencia empresarial: *datawarehousing*, OLAP y *data mining*, se decidió desarrollar un prototipo que facilite la interacción con todas las herramientas mencionadas anteriormente.

Este prototipo requiere, para brindar el ambiente integral que permita aplicar el proceso de la inteligencia empresarial en las PyMES, de la realización de varios pasos que desde el punto de vista de programación permitan la compatibilidad entre las herramientas.

Se desarrolló un nuevo módulo, mediante la arquitectura de código que presenta JRubik (Java Plug-ins), para la incorporación de los métodos de las APIs de Weka que permiten aplicar el algoritmo *Apriori* a los datos del *datawarehouse*. Este módulo fue agregado a la plataforma existente de JRubik convirtiendolo en un prototipo diferente.

Para la interacción con el usuario se decidió modificar la interfaz de JRubik, aprovechando la ventaja de su tipo de licencia de código abierto, para convertirlo en un ambiente consolidado. Dentro de los cambios que se hicieron se encuentran:

- La figura 5.16 muestra la personalización de la interfaz, con el nombre del sistema PIE (PyMEs Inteligencia Empresarial).
- Se agregó un panel, al lado izquierdo con las consultas predeterminadas mencionadas anteriormente en la sección 5.2.1, como podemos ver la figura 5.17.

- Se agregó un menú, como se ve en las figuras 5.18 y 5.19, que presenta los resultados de la aplicación del algoritmo de *data mining* a los cuatro archivos, llamado "DataMining", empleando la infraestructura de Java Plug-in framework [JPF, 2007].

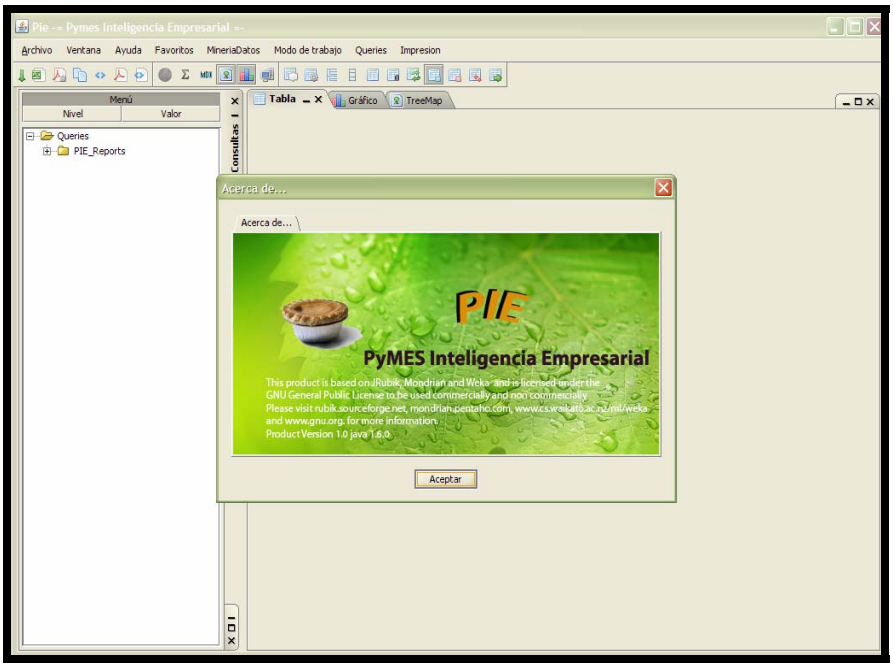

Figura 5.16 Personalización de la interfaz del sistema PIE.

- Se agregó el *datasource* que permite a un usuario más experimentado sus propias consultas en MDX arrastrando las dimensiones y medidas hacia las filas o columnas y desde la pestaña del navegador, seleccionar los atributos que desee visualizar, como vemos en la figura 5.20.

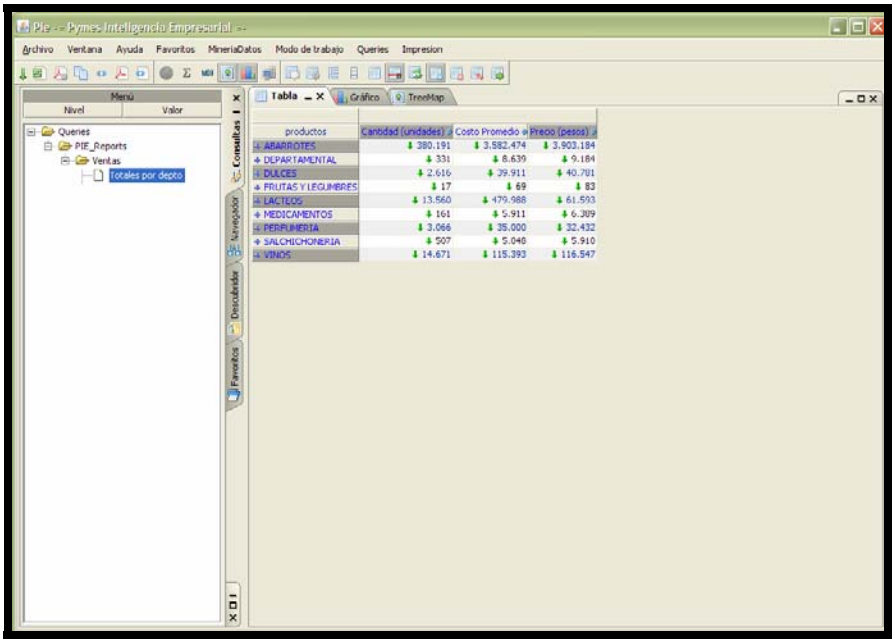

Figura 5.17 Panel de consultas predeterminadas.

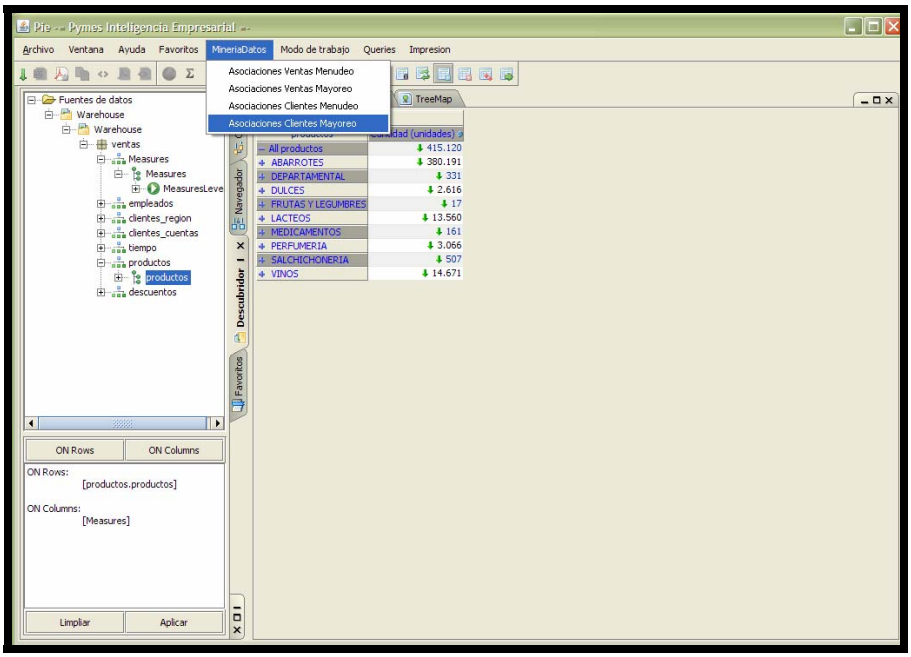

Figura 5.18 Menú de *data mining* agregado al sistema PIE.

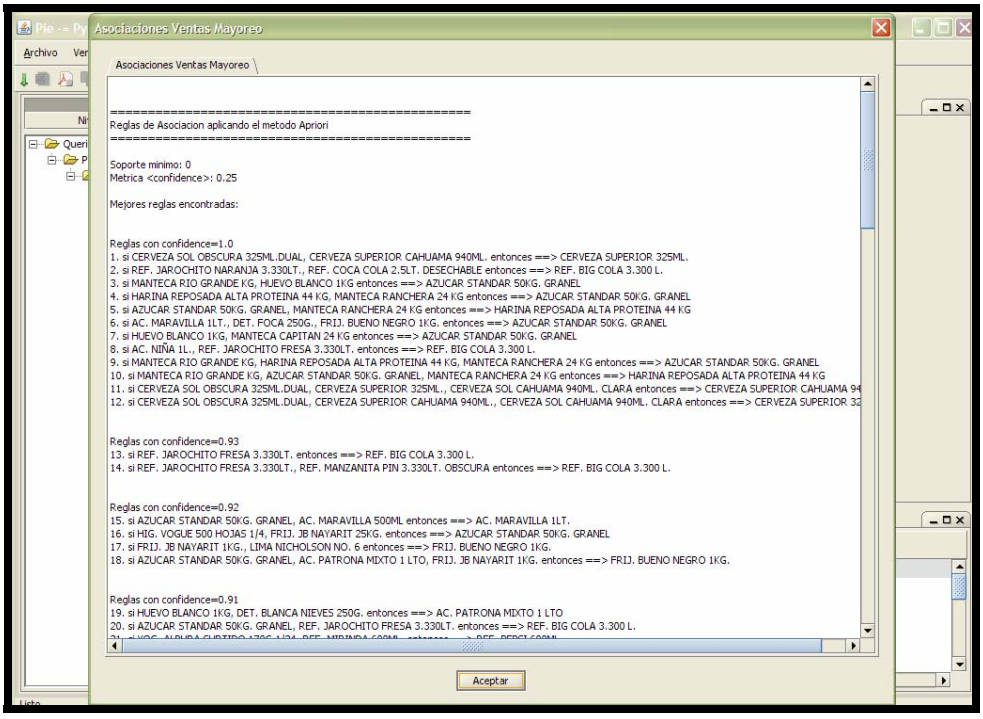

Figura 5.19 Reporte de reglas resultantes de la aplicación del algoritmo *Apriori.*

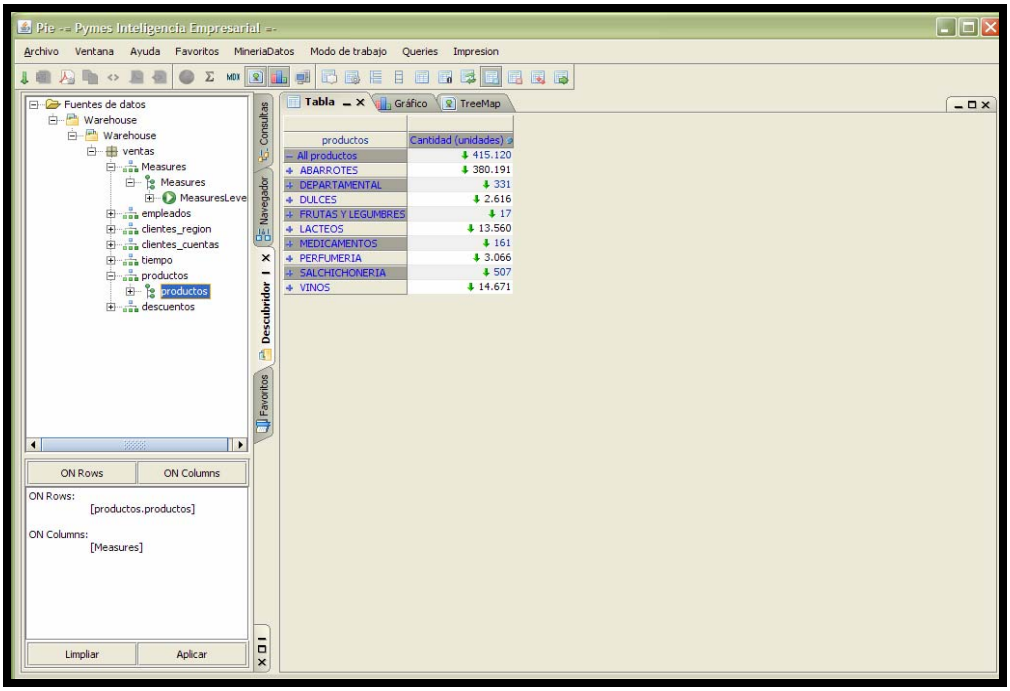

Figura 5.20 Panel de consultas para usuarios más experimentados.

En la siguiente sección veremos las acciones que se realizan para optimizar los procesos de ETL, OLAP y *data mining*.

Como vemos, la implementación del prototipo requirió de mucho más que la simple unión de las herramientas, se hizo un estudio de los conceptos necesarios que hicieran posible la integración de las 3 herramientas, se analizaron las opciones de modelado y arquitectura del sistema, más adecuadas para resolver la problemática de la tesis, así mismo se definieron y determinaron parámetros y procedimientos para hacer posible la aplicación del proceso de inteligencia empresarial.

5.4 Optimización

Un aspecto importante de todo sistema de inteligencia empresarial dentro de una organización, es que no sólo debe proveer un buen ambiente de consultas de ágil respuesta, también necesita ser rápido y preciso para satisfacer las necesidades de su comunidad de usuarios. Para ello no solamente se deben examinar aspectos de creación de una base de datos física eficiente, sino también examinar cómo mejorar el proceso de desarrollo y el modelo. Sobre todo en una PyME donde los recursos son limitados como se mencionó en los capítulos I y IV.

En lo que respecta al modelo multidimensional, se aplicó la denormalización al esquema de copo de nieve, esta optimización consiste en emplear un esquema estrella en vez de un esquema copo de nieve, ya que esto puede disminuir el rendimiento de las consultas de OLAP, en donde se involucren diversas tablas.

Con respecto al *datawarehouse*, el volumen de información que se maneja es del orden de 10,000 ventas al mes, el equivalente a 1Mb aproximadamente lo cual significa, que aún manipulando los datos de un cierto periodo de tiempo, por ejemplo: 12 meses = 12Mb sigue siendo factible para las características de memoria de las máquinas utilizadas en las PyMEs. Por lo anterior podemos descartar técnicas avanzadas de particionamiento y clustering de tablas [Silberchatz, et.al., 2006], limitando la optimización a nivel físico únicamente con el buen empleo de índices.

Los índices que se establecieron para cada una de las tablas los podemos ver la tabla 5.5.

Tablas	Índices
Clientes	Cliente_No
Productos	Producto_No, Departamento
Empleados	Empleado_No
Tiempo_venta	Fecha_No
Descuentos	Descuento_No
Suma_Productos_Iguales_Venta	Venta_No, Producto_No
Venta_fact	Venta_No, Fecha_No, Empleado_No, Cliente_No, Descuento_No, Producto_No

Tabla 5.5 Índices establecidos para las tablas del *datawarehouse*.

En relación al proceso de ETL se consideraron las siguientes medidas de optimización:

- Lectura y escritura en bloques de 1000 registros.
- Utilización de tablas temporales en el manejador de base de datos, indexadas para las operaciones de productos cartesianos.
- Caché de registros.

Por otro lado, para el proceso de OLAP se consideraron las variables, que mostramos en la tabla 5.6:

Propiedad / Valor	Descripción
mondrian.rolap.aggregates.Use=true	Se utiliza para la agregación de tablas. Si se encuentra en *true*, entonces esta propiedad es examinada previamente para cada consulta de agregación. Las agregaciones pueden ser leídas desde la base de datos usando la propiedad `mondrian.rolap. aggregates.Read,` pero la propiedad no será utilizada a menos que esta última se encuentre en *true*.
mondrian.rolap.aggregates.Read=true	Se utiliza para leer las agregaciones de las tablas. Si su valor es *true*, entonces Mondrian explora la base de datos para las tablas agregadas. A menos que `mondrian.rolap.aggregates.Use` se encuentre en *true*, las agregaciones encontradas no serán usadas.
mondrian.native.topcount.enable=true	Si se encuentra habilitada, algunas sentencias `TopCount` de MDX serán ejecutadas en la base de datos y no con Mondrian/Java.
mondrian.native.filter.enable=true	Si se encuentra habilitas, algunas sentencias `Filter()` de MDX serán ejecutadas en la base de datos y no con Mondrian/Java.
mondrian.rolap.star.disableCaching=false	Se utiliza para limpiar la memoria cache de RolapStar después de cada consulta ejecutada. Si se encuentra habilitada esta propiedad RolapStar

	no agregará datos a la memoria de una consulta a la siguiente, la memoria se limpiará después de cada consulta.
mondrian.expCache.enable=true	Controla, si se utiliza o no, la memoria caché para los resultados de las expresiones que se utilizan comúnmente. Con esta propiedad deshabilitada una expresión como: `Rank([Product]. CurrentMember, Order([Product].MEMBERS, [Measures].[Unit Sales]))` realizará muchas ordenaciones redundantes.

Tabla 5.6 Variables de optimización de Mondrian para el proceso de OLAP. [Mondrian, 2007]

Por último para la parte de *data mining* con Weka se definieron los siguientes criterios:

- Dado que los tiempos de ejecución son de aproximadamente 13 minutos, las reglas de asociación se precalculan al momento de actualizar el *datawarehouse* para que al momento de que el usuario despliegue el reporte en la interfaz no se tengan que calcular, nuevamente
- Se asignaron 512Mb de memoria RAM para la realización del algoritmo.
- Se asignó el parámetro de soporte máximo a 0.5 del algoritmo *Apriori*, para que no consumiera demasiados recursos.
- Weka soporta dos tipos de representación de los datos: la no esparcida y la esparcida. La no esparcida mantiene todos los valores de todos los atributos incluyendo los “0” mientras que la esparcida simplemente incluye los número de las columnas y los valores distintos de “0”, como podemos ver la figura 5.21 [Witten, et.al., 2000].

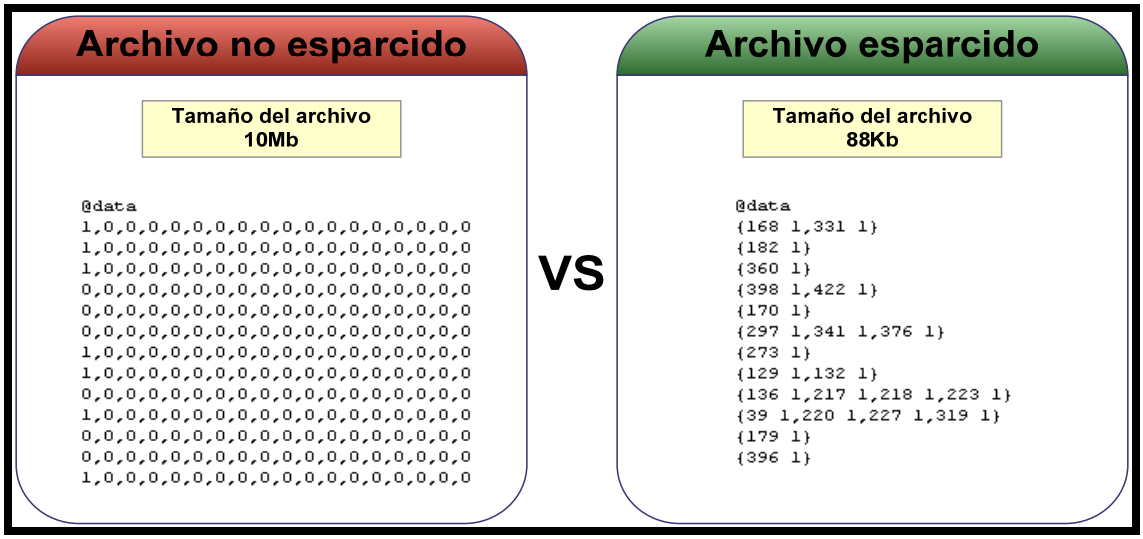

Figura 5.21 Archivo no esparcido vs. archivo esparcido.

Para fines de optimización se utilizó el archivo esparcido, ya que Weka almacena todos los datos en memoria, y de esta manera se reducen el número de datos que se tienen que procesar.

5.5 Discusión final

A lo largo del capítulo se mencionó la implementación de cada una de las capas que conforman al sistema PIE, así como los detalles técnicos que se cuidaron y tomaron en cuenta. También mencionamos las acciones de optimización para mejorar los procesos de ETL, OLAP y *data mining*.

En este capítulo abarcamos las etapas de validación y despliegue, del estándar de CRISP-DM, aunque podríamos realizar varias iteraciones de estas fases para mejorar más los procedimientos, como mencionaremos en el siguiente capítulo.