Un caso de estudio sobre diseño lógico de Data Warehouses

Verónika Peralta

Universidad de la República, Uruguay.
vperalta@fing.edu.uy

Resumen: Un Data Warehouse (DW) es una base de datos que almacena información para la toma de decisiones. Las características de los DWs hacen que los modelos de datos y estrategias de diseño sean diferentes a los utilizados para las bases de datos operacionales, requieren de nuevas técnicas y herramientas de diseño.

Este reporte presenta un caso de estudio sobre diseño lógico de DWs, en particular se presenta una metodología para traducir un esquema conceptual multidimensional a un esquema relacional.

El diseñador define un conjunto de lineamientos de diseño que compelmentan el esquema conceptual con estrategias de diseño, y relaciona los objetos del esquema conceptual con la base fuente a través de mapeos. Luego se aplica un algoritmo de traducción que genera como resultado el esquema lógico del DW mediante transformaciones al esquema fuente.

1. Introducción

Un Data Warehouse (DW) es una base de datos que almacena información para la toma de decisiones. Dicha información es construida a partir de bases de datos que registran las transacciones de los negocios de las organizaciones (bases operacionales). El objetivo de los DWs es consolidar información proveniente de diferentes bases operacionales y hacerla disponible para la realización de análisis de datos de tipo gerencial.

La prioridad es el acceso interactivo e inmediato a información estratégica de un área de negocios. Las operaciones preponderantes no son las transacciones, como en las bases de datos operacionales, sino consultas que involucran gran cantidad de datos y agrupaciones de los mismos.

Las características de los DWs hacen que las estrategias de diseño para las bases de datos operacionales generalmente no sean aplicables para el diseño de DW ([Kim96], [Inm96]). Los modelos de datos para representar los datos almacenados en el DW también son diferentes.

A nivel conceptual resurgen los modelos multidimensionales ([Ken96], [Car00]), que representan la información como matrices multidimensionales o cuadros de múltiples entradas denominados *cubos*. A los ejes de la matriz se los llama *dimensiones* y representan los criterios de análisis, y a los datos almacenados en la matriz se los llama *medidas* y representan los indicadores o valores a analizar.

A nivel lógico surgen implementaciones de los cubos tanto para bases de datos relacionales como multidimensionales. Para el caso de bases relacionales surgen nuevas técnicas y estrategias de diseño que apuntan esencialmente a optimizar la performance en las consultas introduciendo redundancia, lo cual eventualmente sacrifica la performance en las actualizaciones. ([Kim96], [Bal98], [Kor99]).

Una de las tareas más importantes en la construcción de un DW es la construcción de su *esquema lógico*. El esquema lógico es una especificación más detallada que el esquema conceptual, donde se incorporan nociones de almacenamiento, performance y estructuración de los datos.

Un componente adicional a tener en cuenta son las bases fuentes. Un DW no es una base de datos para construir desde cero, sino que debe construirse con información extraída de un cierto conjunto de bases fuente. Durante el diseño lógico deben considerarse dichas bases y cómo se corresponden con el esquema conceptual. Por lo tanto es escencial poder relacionar los elementos del esquema conceptual con las tablas y atributos de las bases fuentes.

Para las bases de datos operacionales existen varias propuestas para traducir un esquema E/R en un esquema relacional ([Mar89], [Teo86], [Jaj83]). Debido a la utlización de nuevos modelos de datos y técnicas de diseño para DWs, tanto a nivel conceptual como lógico, se necesitan también nuevas técnicas de traducción. La traducción debe tomar como entrada no sólo el esquema conceptual sino también las bases fuente.

Este reporte presenta un caso de estudio en diseño de DWs, en particular presenta una metodología para traducir un esquema conceptual multidimensional a un esquema relacional ([Per01]).

El esquema conceptual se complementa con lineamientos de diseño que abstraen estrategias de diseño de DWs y restricciones de performance y almacenamiento para el mismo. Los lineamientos permiten al diseñador elegir la estrategia que mejor se adecúa a su problema concreto. El disñador además especifica correspondencias o mapeos entre el esquema conceptual y la fuente, de manera de saber de dónde se obtienen los datos para cada elemento del esquema conceptual.

La metodología consiste en la aplicación de transformaciones de esquema a las bases fuentes, hasta lograr un esquema lógico que se adecúe a los requerimientos expresados a través del esquema conceptual y los lineamientos.

Como transformaciones básicas se utilizará una extensión al conjunto de *Transformaciones de Esquemas* propuestas por Marotta en [Mar00]. Dichas transformaciones construyen paso a paso el DW generando una traza de aplicación que permite documentar el proceso y facilita su evolución.

La elección de que transformaciones aplicar puede ser muy amplia, y cada problema puede ser eventualmente resuelto por muchas trazas de aplicación. Algunas de esas trazas llevarán a mejores soluciones que otras, tanto desde el punto de vista de la calidad del esquema resultado, como de la performance en su construcción.

Este trabajo presenta un conjunto de reglas de transformación y un algoritmo que dan como resultado una traza de aplicación. Como el algoritmo debe resolver casos generales, muchos de los pasos aplicados que puedan ser imprescindibles para un ejemplo concreto puedan ser innecesarios para otro ejemplo.

Para la traducción no es necesaria la intervención del diseñador, por lo que el proceso es totalemente automatizable. El diseñador interviene mediante los lineamientos y mapeos en una etapa anterior a la traducción. Esto permite separar los aspectos semánticos, como estrategias e interpretación de los datos de la fuente, de los aspectos de implementación como el algoritmo de traducción.

Las siguientes secciones siguen paso a paso el proceso de diseño del DW. En la sección 2 se introduce el caso de estudio, tanto los requerimientos y el esquema conceptual como las bases fuentes. Se discuten además algunas estructuras que permiten resolver ambigüedades en la consulta a las fuentes. En la sección 3 se definen lineamientos y en la sección 4 se definen los mapeos entre el modelo conceptual y las fuentes. En la sección 5 se muestra la aplicación del algoritmo y en la sección 6 se concluye.

2. Presentación del caso de estudio

En esta sección se presenta un caso de estudio sobre una empresa distribuidora de productos que desea implantar un Data Warehouse.

La sección 2.1 cuenta la problemática de la empresa y resume los requerimientos. La sección 2.2 presenta un esquema conceptual que resuelve dichos requerimientos.

La sección 2.3 presenta la base de datos fuente con que cuenta la empresa.

2.1. Requerimientos

La empresa distribuidora de productos alimenticios *Gran Distribuidor* desea instalar un sistema de DW para hacer un seguimiento más eficiente de sus productos.

Se trata de una empresa nacional, que cuenta con diversos centros de fabricación y/o elaboración de productos alimenticios y trabaja también en cooperación con productores agrícolas de la región. La empresa se encarga también de la distribución de los productos en todo el territorio nacional.

Se comenzó con la distribución de productos envasados y bebidas, incorporándose luego los lácteos y panificados. Recientemente, gracias a los acuerdos con cooperativas agrarias se incluyó la distribución de productos agrícolas. Muchos de los productos que se distribuyen son muy perecederos (la mayor parte de los lácteos, panificados y vegetales), por lo que se debe ajustar muy bien las cantidades en stock de estos productos.

La empresa trabaja con empresas mayoristas y supermercados, pero también con almacenes y restaurantes. Algunos de estos clientes tienen casas en varias ciudades del país por lo que debe resolverse el traslado de mercaderías al interior. Actualmente se está apuntando a incrementar las ventas en las ciudades del interior y ganar mercado incorporando comercios locales.

La empresa desea resolver los siguientes requerimientos:

Evolución de las ventas:

- Se desea hacer un seguimiento de las ventas comparando los distintos meses del año, y del año anterior, estudiando la evolución por familia de productos, y pudiéndola refinar hasta un producto en concreto.
- Se desea también observar las variaciones en las ventas para las distintas ciudades del país.

Análisis de mercado.

- Las diferentes promociones están orientadas a un determinado perfil de clientes, por lo que es necesario medir los volúmenes de venta para los diferentes rubros (mayoristas, supermercados, almacenes y restaurantes) estudiando los efectos positivos y/o negativos de la promoción en cada sector. No interesa comparar cliente por cliente, alcanza con un fraccionamiento vertical por rubros.

Distribución geográfica.

- Interesa comparar las ventas por departamentos y ciudades. Esto nos indica las regiones que están en riesgo y necesitan de mayor atención.

Desempeño de vendedores.

- Se necesita comparar el desempeño de los distintos vendedores, y la evolución de dicho desempeño a lo largo del tiempo.
- Un estudio de ventas por producto ayuda a planificar qué productos se asignarán a qué vendedores, y un estudio de ventas por ciudad ayuda a planificar las giras a las que se asignarán los mismos.

2.2. Esquema Conceptual

Se utilizará el modelo conceptual multidimensional CMDM definido por Carpani en [Car00].

A continuación se presenta un esquema conceptual diseñado a partir de los requerimientos. El diseño de dicho esquema escapa a los objetivos de este reporte.

Se relevaron 4 dimensiones: artículos, clientes, vendedores y fechas. La dimensión cantidades representa a las medidas, pero es tratada como una dimensión más ya que CMDM trabaja con dimensionalidad genérica ([Car00]). La Figura 1 muestra la representación gráfica de las dimensiones.

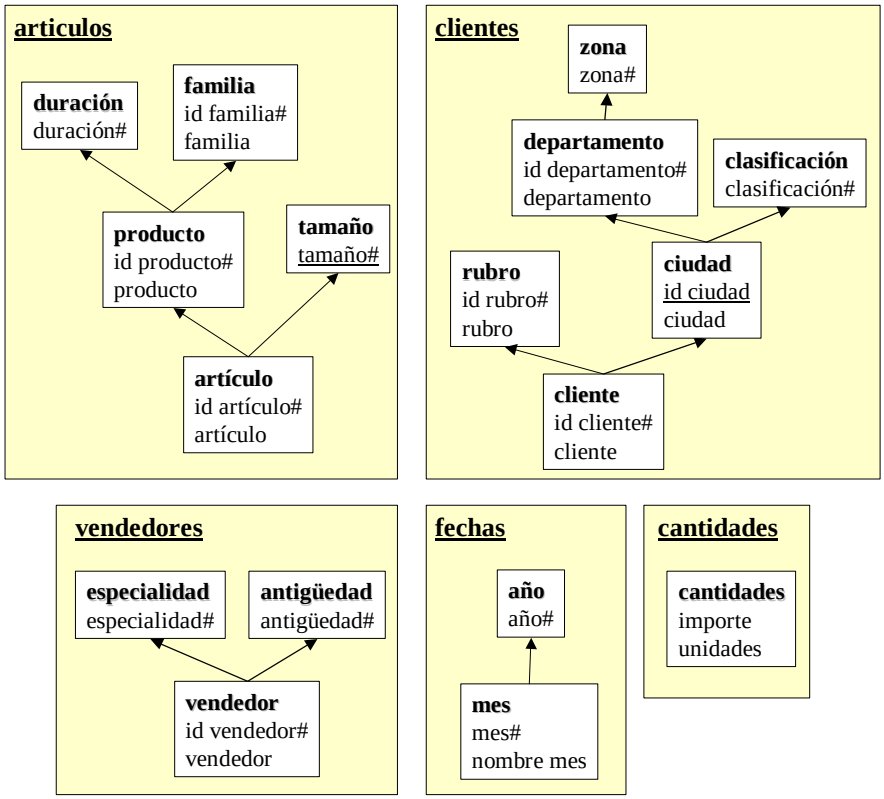

Figura 1 – Dimensiones y Niveles

Se relevó una única relación dimensional: venta, que vincula todas las dimensiones relevadas. La Figura 2 muestra la representación gráfica de la relación dimensional.

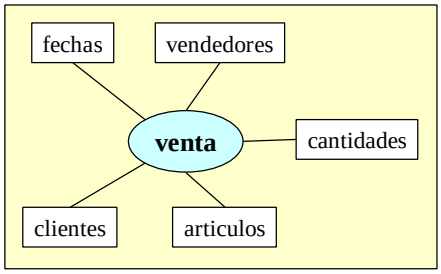

Figura 2 – Relaciones Dimensionales

2.3. Base fuente

Actualmente la empresa cuenta con una base de datos fuente con información de facturación y ventas.

2.3.1. Tablas

A continuación se describen las tablas que componen la base de datos de producción de la empresa. Para cada tabla se presentan sus atributos y su clave primaria (atributos subrayados).

- **Departamentos** (Id depto, Nom depto, Zona)

  Contiene información sobre los departamentos de nuestro país. Para cada uno se guarda (en el orden de los atributos) identificador, nombre y zona.

- **Ciudades** (Id ciudad, Id depto, Nom ciudad, Población, Clasificación)

  Contiene información sobre las ciudades o localidades de nuestro país, ya sea que hay clientes en esa ciudad o no. Para cada una se guarda (en el orden de los atributos) identificador del departamento en que está, identificador de la ciudad, nombre de la ciudad, población y clasificación.

- **Rubros** (Id rubro, Nom rubro)

  Contiene información sobre los rubros de los clientes (por ejemplo: almacenes, supermercados). Para cada uno se guarda (en el orden de los atributos) identificador y nombre.

- **Clientes** (Id cliente, Nombre, Dirección, Teléfono, Ciudad, Departamento, Rubro, Categoría, Fecha alta)

  Contiene información sobre los clientes o empresas a las que se vende. Para cada uno se guarda (en el orden de los atributos) identificador, nombre, dirección actual, teléfono, ciudad, departamento, rubro, categoría, y fecha de alta en el sistema.

- **Facturas** (Factura, Fecha, Cliente, Vendedor)

  Contiene información sobre las ventas realizadas a los clientes. Cada registro corresponde a una factura o boleta. Para cada uno se guarda (en el orden de los atributos) número de factura, fecha, cliente y vendedor.

- **Registros-Facturas** (Factura, Artículo, Importe, Unidades)

  Contiene información sobre el detalle de las facturas, es decir, el desgloce por artículo vendido. Para cada artículo se guarda (en el orden de los atributos) número de factura, identificador del artículo, importe total y unidades vendidas.

- **Artículos** (Id artículo, Id producto, Id tamaño)

  Contiene información sobre los artículos que vende la empresa. Para cada uno se guarda (en el orden de los atributos) identificador del artículo, identificador de producto (agrupación de artículos) e identificación del tamaño (clasificación de los tamaños).

- **Productos** (Id producto, Id familia, Id duracion)

  Contiene información sobre los productos de la empresa. Son agrupaciones de artículos (por ejemplo: un producto puede ser "Salsa portuguesa" y uno de sus artículos "Salsa portuguesa, lata de 1/2 kg"). Para cada producto se guarda (en el orden de los atributos) identificador del producto, identificación de la familia (agrupación de productos) e identificación de la duración (clasificación según su grado de perecedad).

- **Códigos** (Tipo, Código, Descripción)

  Contiene descripciones de códigos utilizadas por el sistema. El campo tipo indica a qué se refiere el código (se encuentran codigos de artículos, productos, tamaños, duraciones y familias). El campo código indica el código o identificador, y el campo descripción una descripción del mismo.

- **Vendedores** (Id vendedor, Nombre, Dirección, Teléfono, Especialidad, Antigüedad)

  Contiene información sobre los vendedores de la empresa. Para cada uno se guarda (en el orden de los atributos) identificador, nombre, dirección actual, teléfono, especialidad y antigüedad en la empresa.

2.3.2. Links

La Figura 3 bosqueja la vinculación entre las tablas de la base fuente. Las líneas entre las tablas (links) indican por qué atributos se deben joinear dichas tablas.

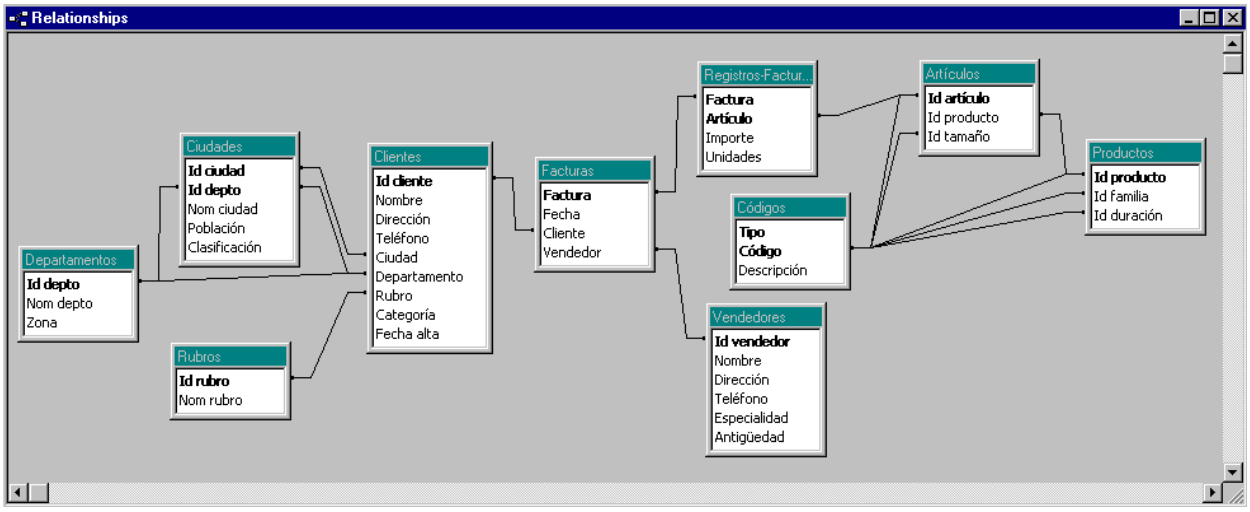

Figura 3 – Links entre tablas de la base fuente.

En este ejemplo todos los links representan la igualdad entre los atributos. Múltiples líneas entre dos tablas (por ejemplo entre *Ciudades* y *Clientes*) representan un link con el *and* (y lógico) de las dos igualdades.

Se presenta un problema con las múltiples líneas entre las tablas *Artículos* y *Códigos*, ya que la intención es joinear con dos instancias de la tabla *Códigos*, no cumplir ambas condiciones. Para ello se definen alias de las tablas. La motivación y definición de links y alias se presenta en [Per01] en el anexo 3.

En el ejemplo hay dos casos en los que se necesita utilizar alias:

- El atributo *Código* de la tabla *Códigos* joinea con dos atributos de la tabla *Artículos*.
- El atributo *Código* de la tabla *Códigos* joinea con tres atributos de la tabla *Productos*.

Se resolverán las ambigüedades generando cinco alias de la tabla *Codigos*:

```
CÓDIGOS-ID ARTICULO (Tipo, Id artículo, Descripción)
CÓDIGOS-ID TAMAÑO (Tipo, Id tamaño, Descripción)
CÓDIGOS-ID PRODUCTO (Tipo, Id producto, Descripción)
CÓDIGOS-ID FAMILIA (Tipo, Id familia, Descripción)
CÓDIGOS-ID DURACIÓN (Tipo, Id duración, Descripción)
```

Los atributos de las tablas *Artículos* y *Productos* referencian a los respectivos alias (por ejemplo: se define un link entre el atributo *Id articulo* de la tabla *Artículos* con el atributo *Id articulo* de la tabla *Codigos-id articulo*). La Figura 4 muestra la definición de links para las nuevas tablas.

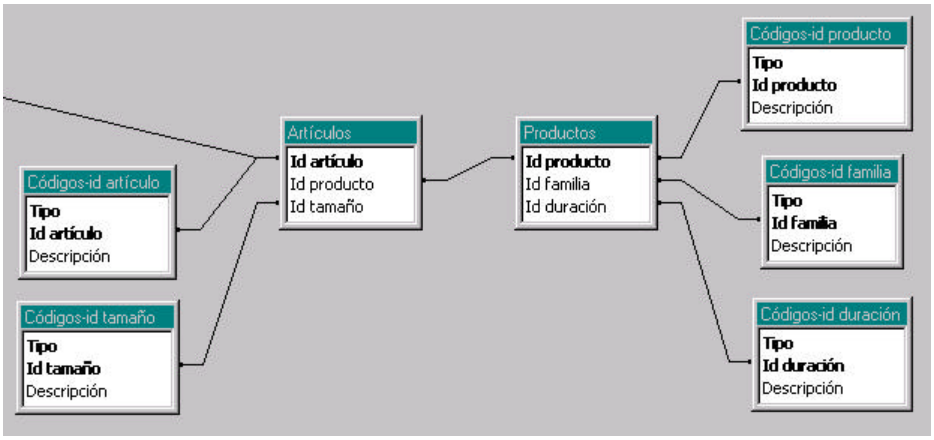

Figura 4 – Alias y Re-definición de los Links (fracción de los links)

3. Lineamientos

Los lineamientos son información de diseño que complementan al modelo conceptual y a las bases fuentes, y permiten al diseñador dar pautas sobre el esquema deseado para el DW.

Hay 3 tipos de lineamientos: materialización de relaciones, fragmentación de dimensiones y fragmentación de cubos. La materialización de relaciones permite indicar qué cubos se quieren materializar, atendiendo a los requerimientos de performance y almacenamiento. La fragmentación de dimensiones permite elegir el estilo de diseño deseado para el DW, esto incluye obtener un esquema estrella, snowflake, o estrategias intermedias, en este último caso indicando que dimensiones denormalizar, normalizar o fragmentar. La fragmentación de cubos permite almacenar por separado datos históricos, o dividir la instancia de los cubos de acuerdo a criterios del diseñador.

A continuación se presentan los lineamientos definidos para el ejemplo.

3.1. Materialización de Relaciones

Se elige materializar tres cubos para la relación dimensional *Venta*:

1- Con detalle de artículos, clientes, vendedores y meses.
2- Con detalle de artículos, rubros, vendedores y meses.
3- Con detalle de artículos y meses.

La Figura 5 muestra la representación gráfica de los cubos. El nombre está dentro del cubo, y entre paréntesis el nombre de la relación que materializa. Los rectángulos blancos representan los niveles de detalle. Las medidas corresponden al nivel marcado por una flecha.

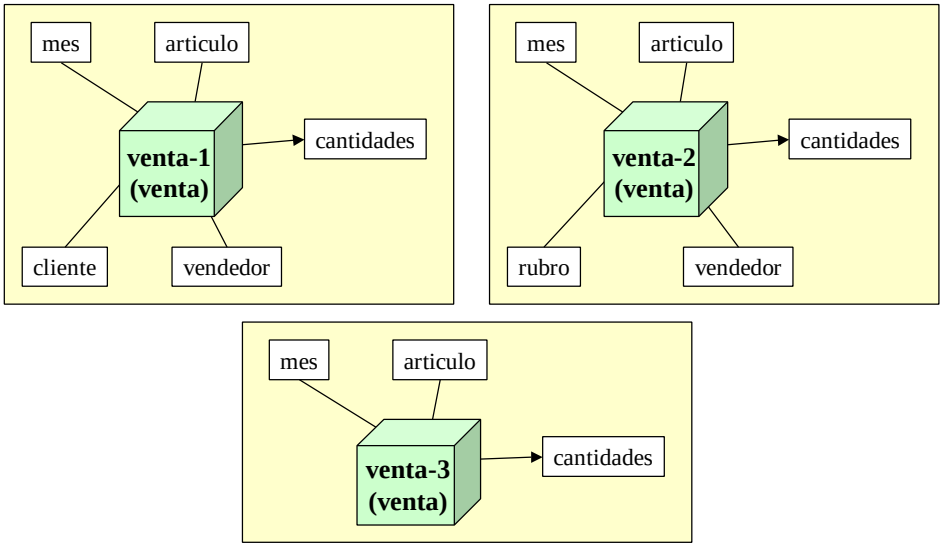

Figura 5 – Cubos

3.2. Fragmentación de Cubos

Se decide fragmentar los cubos de la siguiente manera:

1- Una banda para las ventas del año actual, y otra con el resto de la historia.
2- Una única banda.
3- Una única banda.

La Figura 6 muestra la representación gráfica de las bandas definidas. Las bandas se indican en la llamada celeste mediante predicados.

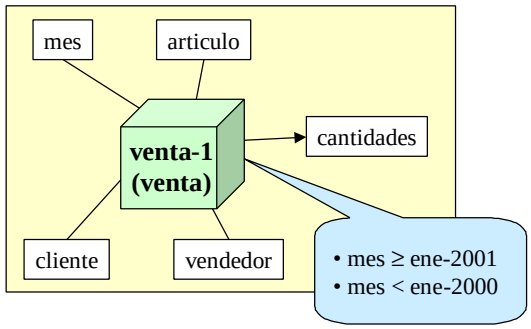

Figura 6 – Bandas

3.3. Fragmentación de Dimensiones

Se decide seguir las siguientes estrategias de diseño para las dimensiones:

- Clientes: 2 fragmentos, uno con cliente y rubro, y el otro con los restantes .
- Artículos: 3 fragmentos, uno con artículo y tamaño, otro con producto y duración, y el otro con familia.
- Vendedores: denormalizada.
- Fechas: denormalizada.
- Cantidades: No se implementará, será utilizada como medidas.

La Figura 7 muestra la representación gráfica de los fragmentos. Los niveles coloreados con el mismo color pertenecen al mismo fragmento, lo que significa que quieren almacenarse juntos.

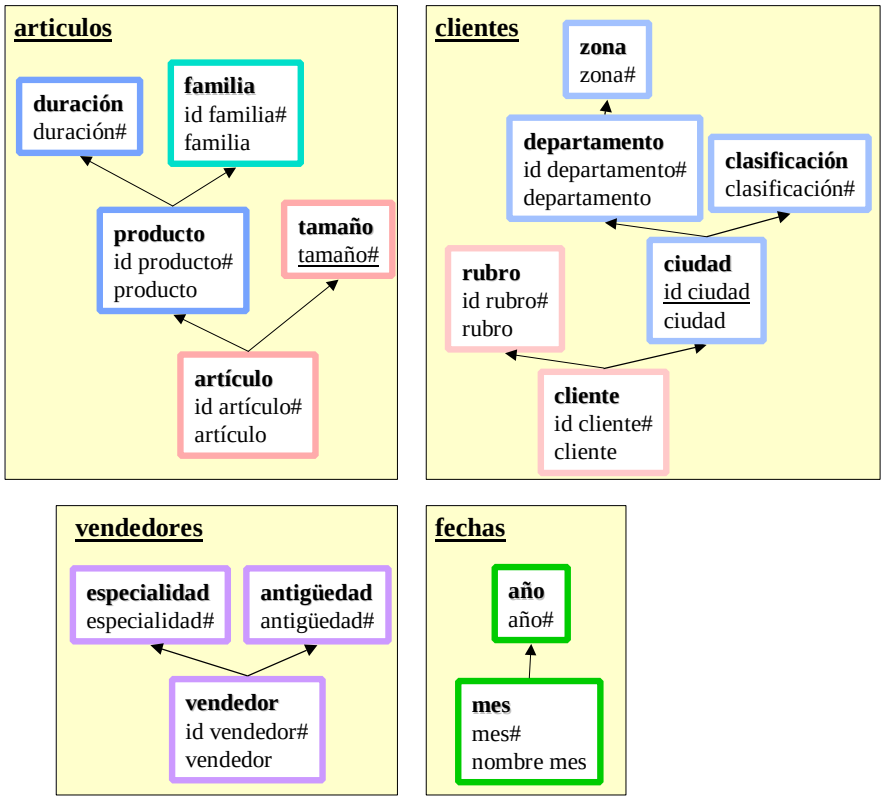

Figura 7 – Fragmentos

4. Mapeos

Un mapeo es una función que muestra como se corresponden los objetos del modelo conceptual con la base fuente.

Los mapeos son funciones que asocian a a cada item de un objeto del esquema conceptual una expresión de mapeo, construida en base a las tablas y atributos de la fuente. Estas funciones son definidas por el diseñador en forma explícita. Se tendrá una función de mapeo para cada fragmento de dimensión, y una función de mapeo para cada cubo.

Un cubo puede mapearse a las fuentes mediante una función de mapeo (mapeo explícito) o puede indicarse como efectuar un drill-up respecto a otro cubo ya mapeado, que tiene más detalle.

Una expresión de mapeo puede ser un atributo de una tabla fuente (directo), o un cálculo que involucra varios atributos de una tupla (cálculo simple), o una totalización que involucra varios atributos de varias tuplas (cálculo agregado) o algo externo a las fuentes como una constante, una estampa de tiempo o dígitos de versión.

4.1. Mapeos de Fragmentos

A continuación se presentan los mapeos definidos para los fragmentos de las dimensiones.

La Figura 8 y la Figura 9 muestan los mapeos de los fragmentos de la dimensión clientes. La Figura 10, la Figura 11 y la Figura 12 muestran los mapeos de los fragmentos de la dimensión artículos. La Figura 13 muestra el mapeo del único fragmento de la dimensión vendedores, y la Figura 14 muestra el de la dimensión fechas.

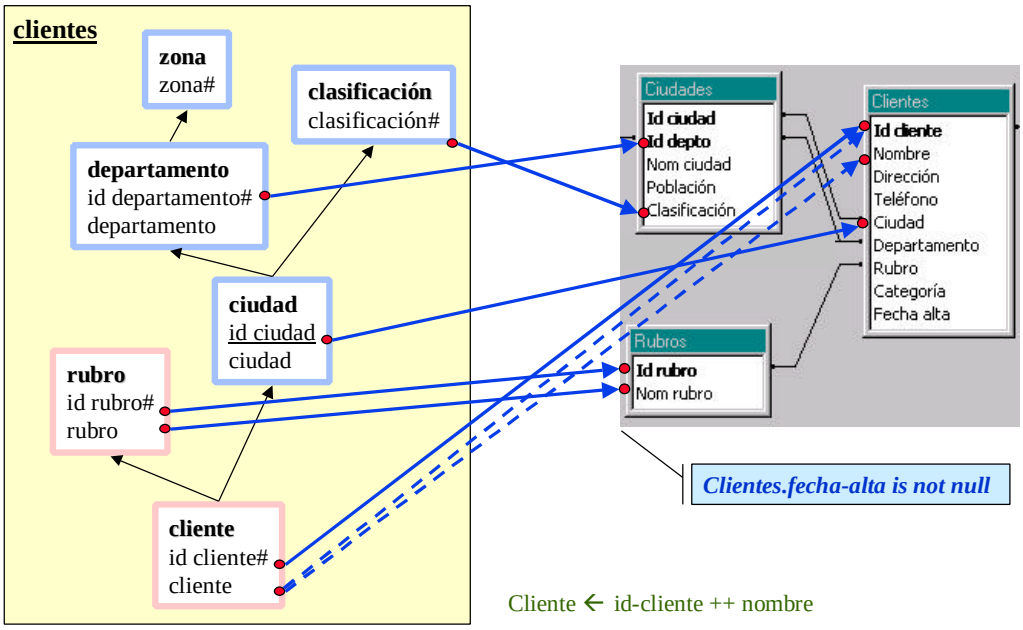

Figura 8 – Mapeo del fragmento rosa de la dimensión clientes

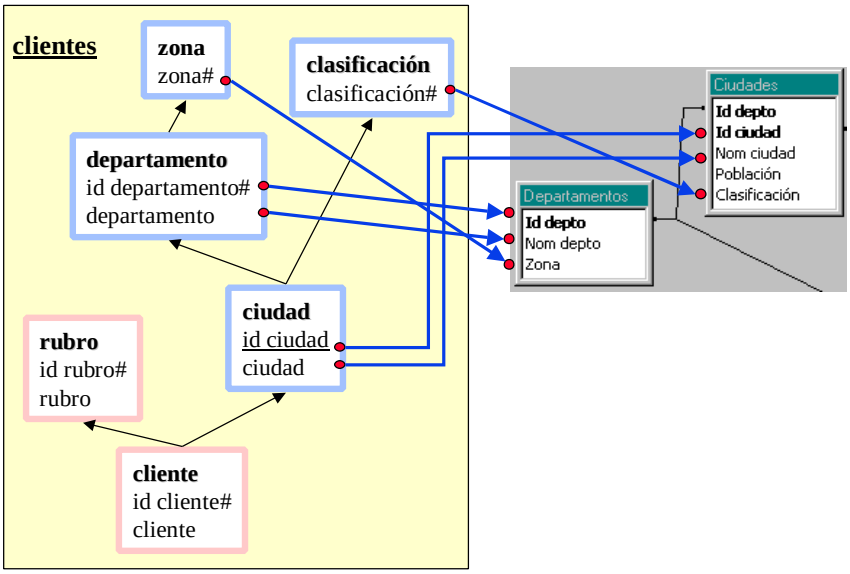

Figura 9 – Mapeo del fragmento celeste de la dimensión clientes

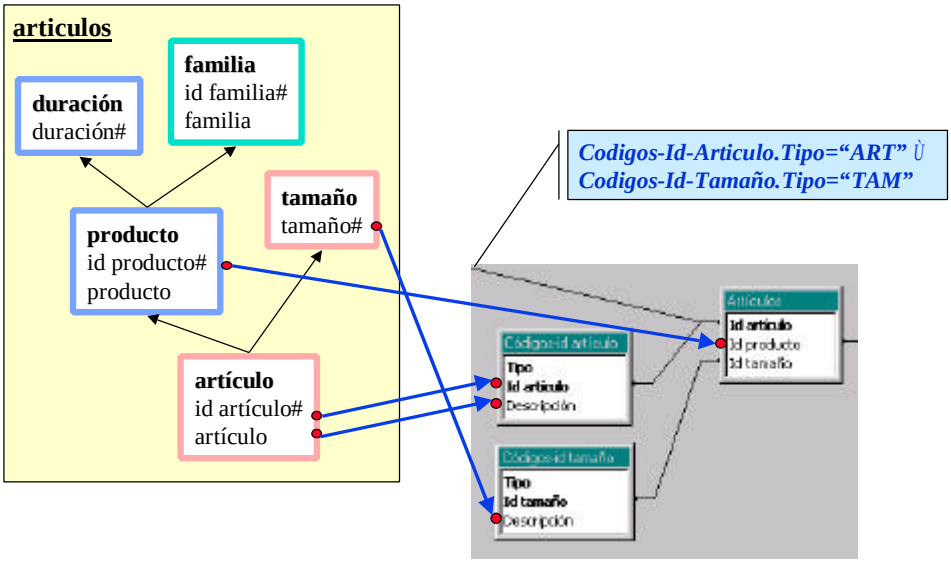

Figura 10 – Mapeo del fragmento rosa de la dimensión articulos

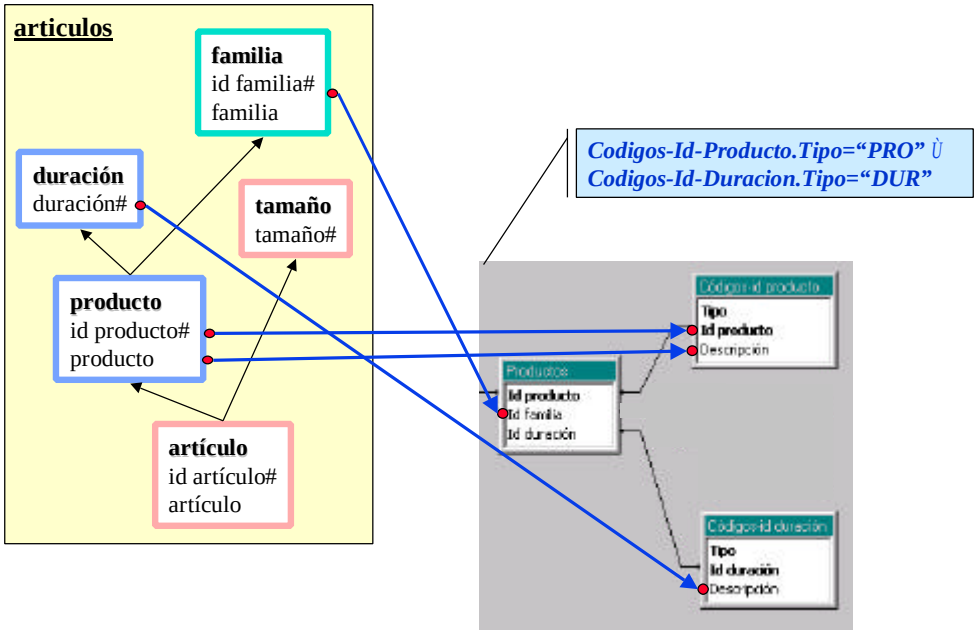

Figura 11 – Mapeo del fragmento azul de la dimensión artículos

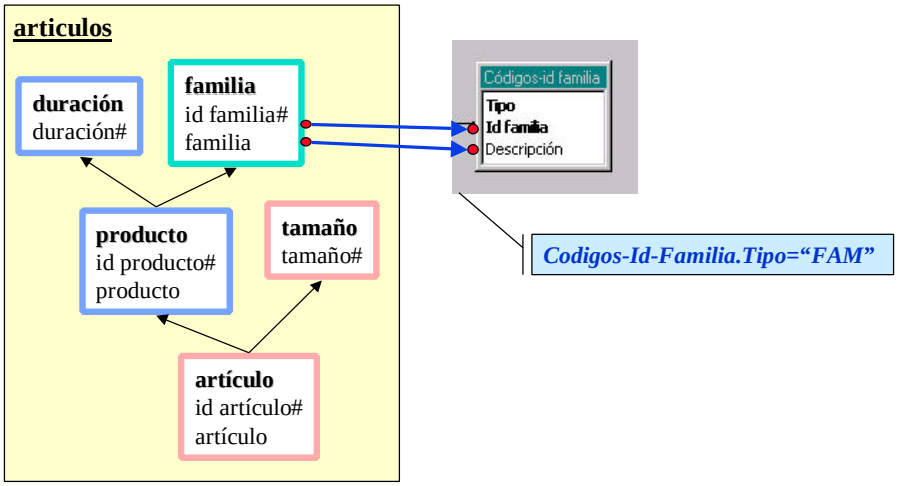

Figura 12 – Mapeo del fragmento verde de la dimensión articulos

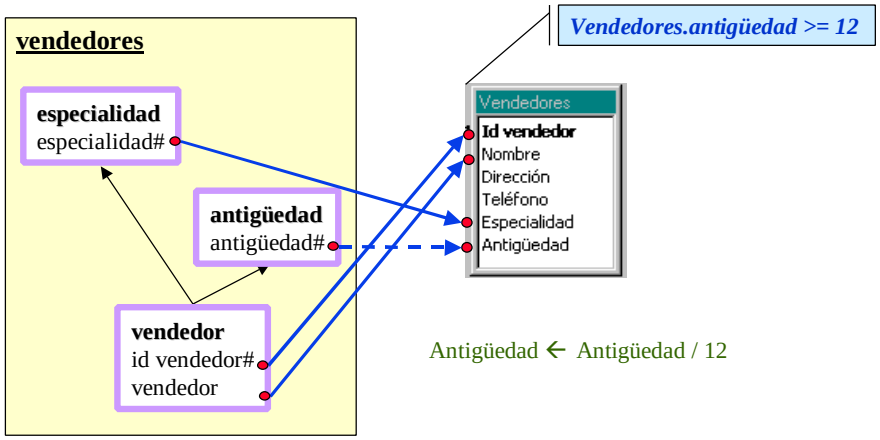

Figura 13 – Mapeo del fragmento de la dimensión vendedores

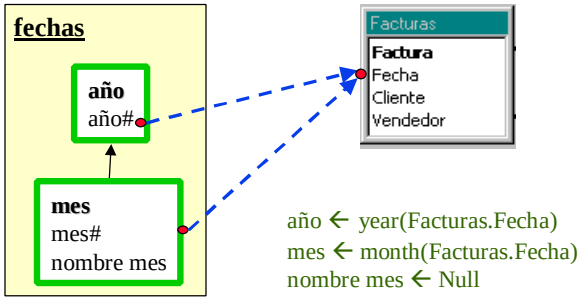

Figura 14 – Mapeo del fragmento de la dimensión fechas

4.2. Mapeos de Cubos

A continuación se presentan los fragmentos definidos para los cubos.

La Figura 15 muestra el mapeo del cubo venta-1. La Figura 16 muestra el mapeo del cubo venta-2 como drill-up del cubo venta-1 por la dimensión clientes, por ello se indica mediante un mapeo como realizar el drill-up. La Figura 17 muestra el mapeo del cubo venta-3 como drill-up del cubo venta-1 eliminando dimensiones.

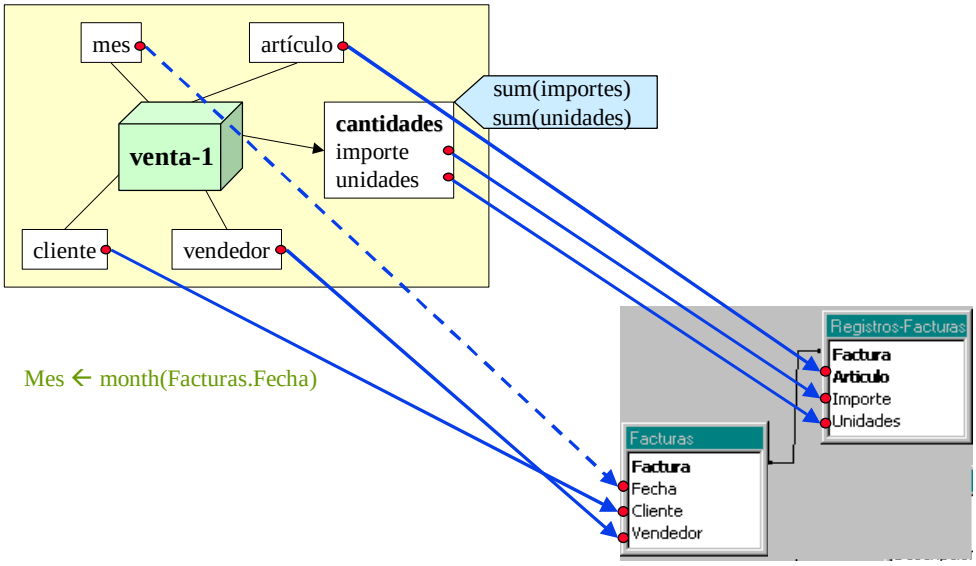

Figura 15 – Mapeo del cubo venta-1

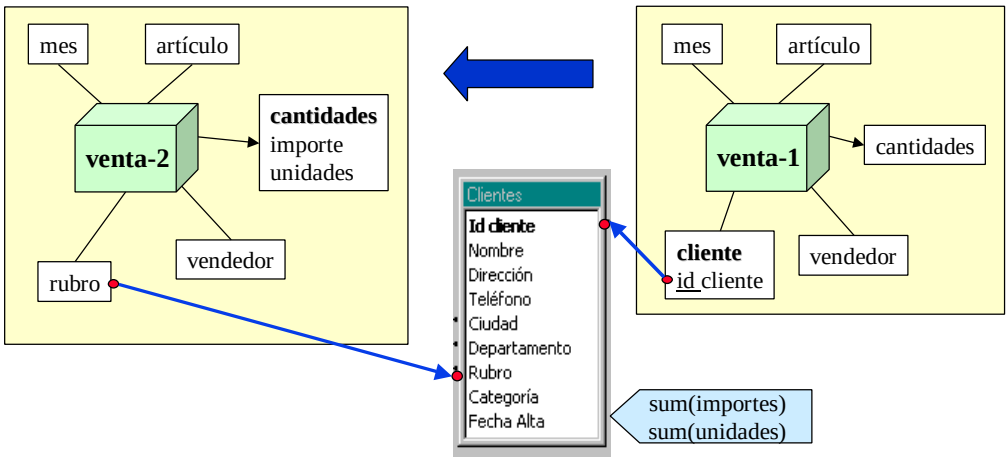

Figura 16 – Mapeo del cubo venta-2

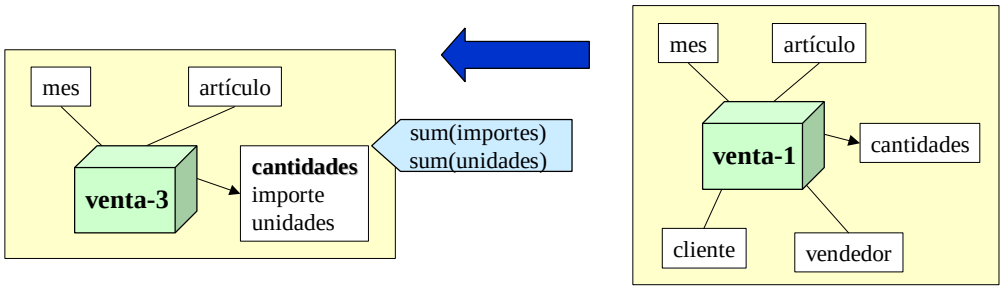

Figura 17 – Mapeo del del cubo venta-3

5. Aplicación del algoritmo

A partir del esquema intermedio y los mapeos se va transformando el esquema fuente aplicando las primitivas de transformación de esquemas definidas por Marotta en [Mar00]. La elección de cuando corresponde aplicar las primitivas y los parámetros adecuados se hace en base a reglas de transformación.

En [Per01] se presenta un algoritmo que da un orden a las reglas.

A continuación se muestra en el ejemplo la aplicación de los pasos del algoritmo, los parámetros utilizados y las tablas que se generan como resultado. Por cuestiones de espacio se omiten las funciones de mapeo.

CONSTRUCCIÓN DE TABLAS DE DIMENSIÓN

Para la construcción de las tablas de dimensión se aplican los pasos 1 a 6 para cada fragmento de dimensión.

STEP 1 - CONSTRUIR EL ESQUELETO

El fragmento rosa de la dimensión *clientes* (se le llamará DwClientes) mapea a tres tablas fuentes: *Clientes*, *Ciudades* y *Rubros*. Se itera aplicando la regla *Join* de a dos tablas. Sea M = SchFMapping(DwClientes).Map, la función de mapeo del fragmento.

- Ejecutar Join (ClientesRubros, M, Clientes, Rubros).

  Resultado DwClientes01 (id-cliente, nombre, direccion, telefono, ciudad, departamento, rubro, categoria, fecha-alta, id-rubro, nom-rubro)

- Ejecutar Join (ClientesRubros, M, DwClientes01, Ciudades).

  Resultado DwClientes02 (id-cliente, nombre, direccion, telefono, ciudad, departamento, rubro, categoria, fecha-alta, id-rubro, nom-rubro, id-depto, id-ciudad, nom-ciudad, poblacion, clasificacion)

Análogamente se aplica la regla *Join* al fragmento celeste de la dimensión *clientes* (DwCiudades) y a los fragmentos rosa (DwArticulos) y azul (DwProductos) de la dimensión *artículos.* Se obtiene como resultado las tablas:

- DwCiudades01 (id-depto, id-ciudad, nom-ciudad, poblacion, clasificacion, id-depto2, nom-depto, zona)
- DwArticulos01 (tipo, id-articulo1, descripcion, id-articulo2, id-producto, id-tamaño)
- DwArticulos02 (tipo1, id-articulo1, descripcion1, id-articulo2, id-producto, id-tamaño1, tipo2, id-tamaño2, descripcion2)
- DwProductos01 (id-producto1, id-familia, id-duracion, tipo, id-producto2, descripcion)
- DwProductos02 (id-producto1, id-familia1, id-duracion1, tipo1, id-producto2, descripcion1, tipo2, id-duracion2, descripcion2)

No se aplica la regla al fragmento verde de la dimensión *articulos* (DwFamilias), al fragmento de la dimensión *vendedores* (DwVendedores) y al fragmento de la dimensión *fechas* (DwFechas) ya que mapean a una única tabla y por tanto no cumplen la precondición de la regla.

STEP 2 - RENOMBRAR ATRIBUTOS PARA ITEMS CON MAPEO DIRECTO.

Los ítems *rubro* e *id-departamenteo* del fragmento DwClientes tienen mapeos directos a los atributos *nom-rubro* y *id-depto* respectivamente, cuyos nombres difieren de los nombres de los ítems. Se aplica la regla *Rename*. Sea M = SchFMapping(DwClientes).Map, la función de mapeo del fragmento.

- Ejecutar Rename (DwClientes, M, DwClientes02).

  Resultado DwClientes03 (id-cliente, nombre, direccion, telefono, ciudad, departamento, rubro1, categoria, fecha-alta, id-rubro, rubro, id-departamento, id-ciudad, nom-ciudad, poblacion, clasificacion)

Análogamente se aplica la regla *Rename* a los fragmentos DwCiudades, DwArticulos, DwProductos, DwFamilias y DwVendedores. Se obtiene como resultado las tablas:

- DwCiudades02 (id-depto, id-ciudad, ciudad, poblacion, clasificacion, id-departamento, departamento, zona)
- DwArticulos03 (tipo1, id-articulo, articulo, id-articulo2, id-producto, id-tamaño1, tipo2, id-tamaño2, tamaño)
- DwProductos03 (id-producto1, id-familia, id-duracion1, tipo1, id-producto, producto, tipo2, id-duracion2, duracion)
- DwFamilias01 (tipo, id-familia, familia)
- DwVendedores01 (id-vendedor, vendedor, direccion, telefono, especialidad, antiguedad)

No se aplica la regla al fragmento DwFechas ya que no tiene ítems con mapeo directo.

STEP 3 - GENERAR ATRIBUTOS PARA ITEMS CON MAPEO CALCULADO O EXTERNO.

El ítem *cliente* del fragmento DwClientes tiene mapeo de cálculo simple (1calcME) a los atributos *id-cliente* y *nombre*. Se aplica la regla *Simple-Calculate*. Sea M = SchFMapping(DwClientes).Map, la función de mapeo del fragmento.

- Ejecutar Simple-Calculate (DwClientes, cliente, M, DwClientes03).

  Resultado DwClientes04 (id-cliente, nombre, direccion, telefono, ciudad, departamento, rubro1, categoria, fecha-alta, id-rubro, rubro, id-departamento, id-ciudad, nom-ciudad, poblacion, clasificacion, cliente)

Análogamente se aplica la regla *SimpleCalculate* a los fragmentos DwVendedores y DwFechas. Se obtiene como resultado las tablas:

- DwVendedores02 (id-vendedor, vendedor, direccion, telefono, especialidad, antiguedad1, antiguedad)
- DwFechas01 (factura, fecha, cliente, vendedor, año)
- DwFechas02 (factura, fecha, cliente, vendedor, año, mes)

El ítem *nombre-mes* del fragmento DwFechas tiene mapeo externo de tipo constante. Se aplica la regla *Constant-Extern-Value*. Sea M = SchFMapping(DwClientes).Map, la función de mapeo del fragmento.

- Ejecutar Constant-Extern-Value (DwFechas, nombre-mes, M, DwFechas02).

  Resultado DwFechas03 (factura, fecha, cliente, vendedor, año, mes, nombre-mes)

Los fragmentos DwCiudades, DwArticulos, DwProductos y DwFamilias no tienen ítems con mapeo calculado ni externo por lo que no se aplica ninguna regla.

STEP 4 - APLICAR FILTROS.

El fragmento DwClientes tiene una condición en su función de mapeo: *Clientes.fecha-alta is not null*. Se aplica la regla *Filter*. Sea M = SchFMapping(DwClientes).Map, la función de mapeo del fragmento.

- Ejecutar Filter (DwClientes, "DwClientes04.fecha-alta is not null", M, DwClientes04).

  Resultado DwClientes05 (id-cliente, nombre, direccion, telefono, ciudad, departamento, rubro1, categoria, fecha-alta, id-rubro, rubro, id-departamento, id-ciudad, nom-ciudad, poblacion, clasificacion, cliente)

Análogamente se aplica la regla *Filter* a los fragmentos DwArticulos, DwProductos, DwFamilias y DwVendedores. Se obtiene como resultado las tablas:

- DwArticulos04 (tipo1, id-articulo, articulo, id-articulo2, id-producto, id-tamaño1, tipo2, id-tamaño2, tamaño)
- DwProductos04 (id-producto1, id-familia, id-duracion1, tipo1, id-producto, producto, tipo2, id-duracion2, duracion)
- DwFamilias02 (tipo, id-familia, familia)
- DwVendedores03 (id-vendedor, vendedor, direccion, telefono, especialidad, antiguedad1, antiguedad)

Los fragmentos DwCiudades y DwFechas no tienen condiciones en sus mapeos por lo que no se aplica la regla.

STEP 5 - ELIMINAR ATRIBUTOS SOBRANTES.

Los atributos *nombre, direccion, telefono, ciudad, departamento, rubro1, categoria, fecha-alta* y *nom-ciudad* no están mapeados al fragmento DwClientes. Se aplica la regla *Fragment-Group*. Sea M = SchFMapping(DwClientes).Map, la función de mapeo del fragmento.

- Ejecutar Fragment-Group (DwClientes, M, DwClientes05).

  Resultado DwClientes06 (id-cliente, id-rubro, rubro, id-departamento, id-ciudad, cliente)

Análogamente se aplica la regla *Fragment-Group* a los restantes fragmentos. Se obtiene como resultado las tablas:

- DwCiudades03 (id-ciudad, ciudad, clasificacion, id-departamento, departamento, zona)
- DwArticulos05 (id-articulo, articulo, id-producto, tamaño)
- DwProductos05 (id-familia, id-producto, producto, duracion)
- DwFamilias03 (id-familia, familia)
- DwVendedores04 (id-vendedor, vendedor, especialidad, antiguedad)
- DwFechas04 (año, mes, nombre-mes)

STEP 6 - AJUSTAR LAS CLAVES.

La clave del fragmento DwClientes es *id-cliente* (clave del nivel inferior en la jerarquía), y la clave de la tabla que lo mapea (*DwClientes06)* está formada por los atributos *id-cliente, id-rubro, id-departamento* e *id-ciudad*. Se aplica la regla *Primary-Key*. Sea M = SchFMapping(DwClientes).Map, la función de mapeo del fragmento.

- Ejecutar Primary-Key (DwClientes, M, DwClientes06).

  Resultado DwClientes07 (id-cliente, id-rubro, rubro, id-departamento, id-ciudad, cliente)

Análogamente se aplica la regla *Primary-Key* al fragmento DwFechas. Se obtiene como resultado la tabla:

- DwFechas04 (año, mes, nombre-mes)

No es necesario ajustar la clave de los fragmentos: DwCiudades, DwArticulos, DwProductos, DwFamilias y DwVendedores.

Se tienen como resultados finales las tablas:

- DwClientes07 (id-cliente, id-rubro, rubro, id-departamento, id-ciudad, cliente)
- DwCiudades03 (id-ciudad, ciudad, clasificacion, id-departamento, departamento, zona)
- DwArticulos05 (id-articulo, articulo, id-producto, tamaño)
- DwProductos05 (id-familia, id-producto, producto, duracion)
- DwFamilias03 (id-familia, familia)
- DwVendedores04 (id-vendedor, vendedor, especialidad, antiguedad)
- DwFechas04 (año, mes, nombre-mes)

CONSTRUCCIÓN DE TABLAS DE HECHOS PARA CUBOS CON MAPEO BASE

Para la construcción de las tablas de hechos se aplican los pasos 7 a 12 para cada cubo con mapeo base. En el ejemplo el único cubo con mapeo base es *venta-1*.

STEP 7 - CONSTRUIR EL ESQUELETO

El cubo venta-1 mapea a dos tablas fuentes: *Facturas* y *Registros-Facturas*. Se aplica la regla *Join*. Sea M = SchCMapping(venta-1).Map, la función de mapeo del cubo.

- Ejecutar Join (venta-1, M, Facturas, Registros-Facturas).

  Resultado DwVenta101 (factura1, fecha, cliente, vendedor, factura2, articulo, importe, unidades)

STEP 8 - RENOMBRAR ATRIBUTOS PARA ITEMS CON MAPEO DIRECTO.

Los ítems *id-cliente*, *id-vendedor* e *id-articulo* del cubo venta-1 tienen mapeos directos a los atributos *cliente*, *vendedor* y *articulo* respectivamente, cuyos nombres difieren de los nombres de los ítems. Se aplica la regla *Rename*. Sea M = SchCMapping(venta-1).Map, la función de mapeo del cubo.

- Ejecutar Rename (venta-1, M, DwVenta101).

  Resultado DwVenta102 (factura1, fecha, id-cliente, id-vendedor, factura2, id-articulo, importe, unidades)

STEP 9 - GENERAR ATRIBUTOS PARA ITEMS CON MAPEO CALCULADO O EXTERNO.

El ítem *mes* del cubo venta-1 tiene mapeo de cálculo simple (1calcME) al atributo *fecha*. Se aplica la regla *Simple-Calculate*. Sea M = SchCMapping(venta-1).Map, la función de mapeo del cubo.

- Ejecutar Simple-Calculate (venta-1, mes, M, DwVenta102).

  Resultado DwVenta103 (factura1, fecha, id-cliente, id-vendedor, factura2, id-articulo, importe, unidades, mes)

STEP 10 - APLICAR FILTROS.

El cubo no tiene condiciones en el mapeo por lo que no se aplica la regla *Filter*.

STEP 11 - ELIMINAR ATRIBUTOS SOBRANTES.

Los atributos *factura1*, *fecha* y *factura-2* no están mapeados al cubo venta-1. Se aplica la regla *Cube-Group*. Sea M = SchCMapping(venta-1).Map, la función de mapeo del cubo.

- Ejecutar Cube-Group (venta-1, {sum(importe), sum(unidades)}, M, DwVenta103).

  Resultado DwVenta104 (id-cliente, id-vendedor, id-articulo, importe, unidades, mes)

STEP 12 - AJUSTAR LAS CLAVES.

La clave del cubo venta-1 está formada por los ítems *id-cliente, id-vendedor, id-artículo* y *mes* (clave de los niveles que no son medida), y la clave de la tabla que lo mapea (*DwVenta104)* es *id-articulo*. Se aplica la regla *Primary-Key*. Sea M = SchCMapping(venta-1).Map, la función de mapeo del cubo.

- Ejecutar Primary-Key (venta-1, M, DwVenta104).

  Resultado DwVenta105 (id-cliente, id-vendedor, id-articulo, importe, unidades, mes)

Se tienen como resultado final la tabla:

- DwVenta105 (id-cliente, id-vendedor, id-articulo, importe, unidades, mes)

CONSTRUCCIÓN DE TABLAS DE HECHOS PARA CUBOS CON MAPEO RECURSIVO

Para la construcción de las tablas de hechos se aplican los pasos 13 y 14 para cada cubo con mapeo recursivo.

STEP 13 - ARMAR LA TABLA DE LA JERARQUÍA

El cubo venta-2 tiene un mapeo recursivo en el que se hace drill-up por la dimensión *clientes* para reducir el detalle de la dimensión (de *cliente* a *rubro*). Se construye un fragmento ficticio (al que se llamará *ClientesAuxiliar*) formado por los niveles *cliente* y *rubro*, y se define como función de mapeo la función de mapeo del drill-up, que sólo mapea los ítems clave de los niveles. Se aplican los pasos 1 a 6 al fragmento ClientesAuxiliar.

STEP 1 – CONSTRUIR EL ESQUELETO

El fragmento ClienteAuxiliar mapea a una única tabla por lo que no verifica la precondición de la regla *Join*. La regla no se aplica.

STEP 2 – RENOMBRAR ATRIBUTOS PARA ITEMS CON MAPEO DIRECTO.

El ítem *id-rubro* del fragmento ClientesAuxiliar tiene un mapeo directo al atributo *rubro*, cuyo nombre difiere del nombre del ítem. Se aplica la regla *Rename*. Sea M = SchFMapping(ClientesAuxiliar).Map, la función de mapeo del fragmento.

- Ejecutar Rename (ClientesAuxiliar, M3, Clientes).

  Resultado DwClienteAuxiliar01 (id-cliente, nombre, direccion, telefono, ciudad, departamento, id-rubro, categoria, fecha-alta)

STEP 3 – GENERAR ATRIBUTOS PARA ITEMS CON MAPEO CALCULADO O EXTERNO.

El fragmento ClienteAuxiliar no tiene ítems con mapeo calculado o externo por lo que no se aplica ninguna regla.

STEP 4 – APLICAR FILTROS.

El fragmento ClienteAuxiliar no tiene condiciones en su función de mapeo por lo que no se aplica la regla *Filter*.

STEP 5 – ELIMINAR ATRIBUTOS SOBRANTES.

Los atributos *nombre, direccion, telefono, ciudad, departamento, categoria* y *fecha-alta* no están mapeados al fragmento ClientesAuxiliar. Se aplica la regla *Fragment-Group*. Sea M = SchFMapping(ClientesAuxiliar).Map, la función de mapeo del fragmento.

- Ejecutar Fragment-Group (ClientesAuxiliar, M3, DwFicticio01).

  Resultado DwClienteAuxiliar02 (id-cliente, id-rubro)

STEP 6 – AJUSTAR LAS CLAVES.

No es necesario ajustar la clave del fragmento ClienteAuxiliar por lo que no se aplica la regla *PrimaryKey*.

El cubo venta-3 tiene un mapeo recursivo en el que se hacen dos drill-ups por las dimensiones *clientes* y *vendedor* pero ambos para eliminar el detalle de las dimensiones, por lo que no se aplica ninguna regla.

STEP 14 - APLICAR EL DRILL-UP.

El cubo venta-2 tiene un mapeo recursivo respecto al cubo venta-1, con un drill-up por la dimensión *clientes* para reducir el detalle de la dimensión. Se aplica la regla *Hierarchy-Drill-Up*.

Sea M = SchCMappings(venta-2).Map la función de mapeo del cubo venta-1. Sea {Dup} = SchCMappings(venta-2).Dups el drill-up respecto a la dimensión *clientes*.

- Ejecutar Hierarchy-Drill-Up (venta-1, venta-2, Dup, M, Dup.Map, DwVenta105, DwClientesAuxiliar02).

  Resultado DwVenta201 (id-rubro, id-vendedor, id-articulo, importe, unidades, mes)

El cubo venta-3 tiene un mapeo recursivo respecto al cubo venta-1, con dos drill-ups por las dimensiones *clientes* y *vendedores* para eliminar el detalle de las dimensiones. Se aplica la regla *Total-Drill-Up*.

Sea M = SchCMappings(venta-2).Map la función de mapeo del cubo venta-1. Sean {DupClientes, DupVendedores} = SchCMappings(venta-3).Dups los drill-ups respecto a las dimensiones *clientes* y *vendedores* respectivamente.

- Ejecutar Total-Drill-Up (venta-1, venta-3, DupClientes, M, DwVenta105).

  Resultado DwVenta301 (id-ventdedor, id-articulo, importe, unidades, mes)

- Ejecutar Total-Drill-Up (venta-1, venta-3, DupVendedores, M, DwVenta301).

  Resultado DwVenta302 (id-articulo, importe, unidades, mes)

Se tienen como resultados finales las tablas:

- DwVenta201 (id-rubro, id-vendedor, id-articulo, importe, unidades, mes)
- DwVenta302 (id-articulo, importe, unidades, mes)

CONSTRUCCIÓN DE TABLAS DE HECHOS PARA FRANJAS DE CUBOS

Para la construcción de las tablas de hechos se aplican los pasos 13 y 14 para cada cubo con mapeo recursivo. En el ejemplo el único cubo con bandas definidas es *venta-1*.

STEP 15 - ARMAR LA TABLA DE CADA BANDA

El cubo venta-1 tiene definidas dos bandas: *mes >= ene-2001* (a la que se llamará banda1) y *mes < ene-2001* (a la que se llamará banda2). Se aplica la regla *Filter* para cada banda. Sea M = SchCMapping(venta-1).Map, la función de mapeo del cubo.

- Ejecutar Filter (venta-1, "DwVenta105.mes >= ene-2001", M, DwVenta105).

  Resultado DwVenta1Banda01 (id-cliente, id-vendedor, id-articulo, importe, unidades, mes)

- Ejecutar Filter (venta-1, "DwVenta105.mes < ene-2001", M, DwVenta105).

  Resultado DwVenta1Banda02 (id-cliente, id-vendedor, id-articulo, importe, unidades, mes)

Se tienen como resultados finales las tablas:

- DwVenta1Banda01 (id-cliente, id-vendedor, id-articulo, importe, unidades, mes)
- DwVenta1Banda02 (id-cliente, id-vendedor, id-articulo, importe, unidades, mes)

Se obtiene como resultado el esquema lógico que se muestra en la Figura 18:

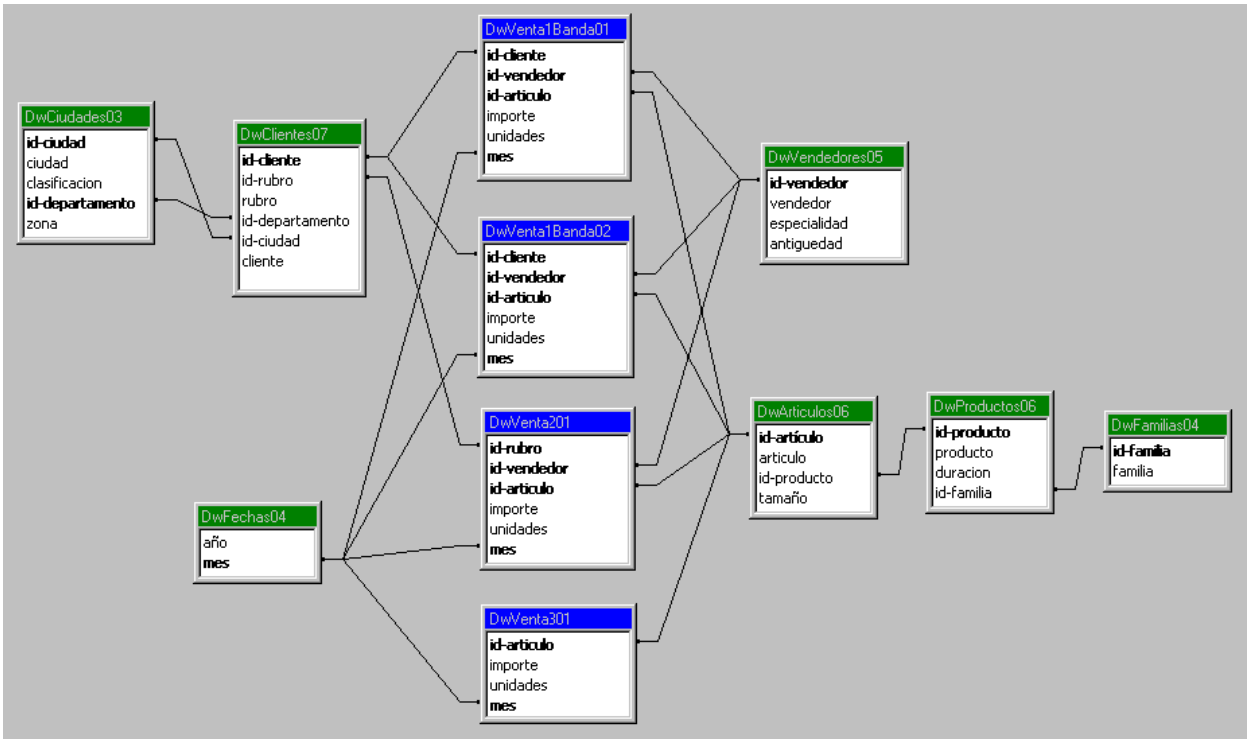

Figura 18 – Esquema lógico del DW

6. Conclusiones

En este reporte se un caso de estudio en diseño de DWs. Es particular se mostró una metodología para traducir el esquema conceptual de un DW a un esuqema lógico relacional. La metodología se divide en dos grandes etapas, la primera de definición de propiedades y correspondencias, que es realizada por el diseñador, y la segunda de transformación de los esquemas fuente, que se realiza automáticamente.

El caso de estudio ilustra la definición de lineamientos y mapeos y permite seguir una ejecución del algoritmo de traducción.

Actualmente se cuenta con un prototipo de una herramienta CASE para aplicar dicha metodología. Se está trabajando en la interconexión del prototipo con otras herramientas de diseño de DWs.

7. Bibliografía

[Ada98] Adamson, C. Venerable, M.: "Data Warehouse Design Solutions". J. Wiley & Sons, Inc.1998.

[Alc01] Alcarraz, A. Ayala, M. Gatto, P.: "Diseño e implementación de una herramienta para evolución de un Data Warehouse Relacional". Undergraduate project. InCo, Universidad de la República, Uruguay, 2001.

[Arz00] G. Arzúa, G. Gil, S. Sharoian. "Manejador de Repositorio para Ambiente CASE". Under-graduate Project. InCo, Universidad de la República, Uruguay, 2000.

[Bal98] Ballard, C. Herreman, D. Schau, D. Bell, R. Kim, E. Valncic, A.: "Data Modeling Techniques for Data Warehousing". SG24-2238-00. IBM Red Book. ISBN number 0738402451. 1998.

[Cab98] Cabibbo, L. Torlone, R.:"A Logical Approach to Multidimensional Databases", EDBT, 1998.

[Car00] Carpani, F.: "CMDM: A conceptual multidimensional model for Data Warehouse". Master Thesis. InCo - Pedeciba, Universidad de la República, Uruguay, 2000.

[Gar00] Garbusi, P. Piedrabuena, F. Vázquez, G.: "Diseño e Implementación de una Herramienta de ayuda en el Diseño de un Data Warehouse Relacional". Undergraduate project. InCo, Universidad de la República, Uruguay, 2000.

[Gol98] Golfarelli, M. Rizzi, S.:"Methodological Framework for Data Warehouse Design.", DOLAP'98, USA, 1998.

[Gol98a] Golfarelli, M. Maio, D. Rizzi, S.:"Conceptual Design of Data Warehouses from E/R Schemes.", HICSS'98, IEEE, Hawaii,1998.

[Inm96] Inmon, W.: "Building the Data Warehouse". John Wiley & Sons, Inc. 1996.

[Inm96a] Inmon, W.: "Building the Operational Data Store". John Wiley & Sons, Inc. 1996.

[Jaj83] Jajodia, S. Ng, P. Springsteel, F.: "The problem of equivalence for entity-relationship diagrams", IEEE Trans. on Software Engineering SEO,5. September 1983.

[Ken96] Kenan Technologies:"An Introduction to Multidimensional Databases". White Paper, Kenan Technologies, 1996.

[Kim96] Kimball, R.:"The Datawarehouse Toolkit". John Wiley & Son, Inc., 1996.

[Mar99] Marotta, A. Peralta, V.: "Diseño de Data Warehouses: Un Enfoque Basado en Transformación de Esquemas". Info-UY'99, Uruguay, 1999.

[Mar00] Marotta, A.: "Data Warehouse Design and Maintenance through Schema Transformations". Master Thesis. InCo - Pedeciba, Universidad de la República, Uruguay, 2000.

[Moo00] Moody, D. Kortnik, M.: "From Enterprise Models to Dimensionals Models: A Methodology for Data Warehouse and Data Mart Design". DMDW'00, Sweden, 2000.

[Per99] Peralta, V. Marotta, A. Ruggia, R.: "Designing Data Warehouses through schema transformation primitives". ER'99 Posters and Demonstrations, France, 1999.

[Per00] Peralta, V. Garbusi, P. Ruggia, R.: "DWD: Una herramienta para diseño de Data Warehouses basada en transformaciones sobre esquemas". Technical Report. InCo, Universidad de la República, Uruguay, 2000.

[Per00a] Peralta, V.: "Sobre el pasaje del esquema conceptual al esquema lógico de DW". JIIO'00, Uruguay, 2000.

[Pic00] Picerno, A. Fontan, M.: "Un editor para CMDM". Undergraduate Project. InCo, Universidad de la República, Uruguay. 2000.

[Teo86] Teorey, T. Yang, D. Fry, J.: "A logical design methodology for relational databases using: the extended entity-relationship model", Cornput&Surveys 18,2. June 1986.