Carga de un Data Warehouse a partir de la Traza de Diseño

Tesis de Maestría

Ignacio Larrañaga

Tutor: Dr. Alejandro Gutiérrez

Grupo de Concepción de Sistemas de Información (CSI)
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
PEDECIBA Informática

2006

Agradecimientos:

A mi amada esposa Cecilia Guillenea que supo entenderme y apoyarme constantemente, a mi tutor, el Dr. Alejandro Gutierrez que creyó en mí y me acompaño durante este largo trayecto. Al grupo de Concepción de Sistema de Información (CSI) que siempre estuvo listo a darme su apoyo. Y finalmente a la empresa Ideasoft que ayudo a formar el profesional que soy hoy en día.

Resumen

Data Warehousing es el término generalmente usado para definir la tecnología de los sistemas de soporte a la toma de decisiones y aplicaciones OLAP. En particular se denomina Data Warehouse (DW) al repositorio de datos integrados, orientados a un dominio específico, no volátiles y variables en el tiempo, que ayudan a la toma de decisiones de una empresa u organización. La estructura de dicho DW se obtiene como resultado de un proceso de diseño generalmente guiado por alguna metodología. El proceso de extraer los datos desde donde residen y transformarlos para almacenarlos en el DW se denomina proceso de carga. El proceso de mantener estos datos actualizados se denomina actualización. La carga y actualización de un DW que fue diseñado utilizando alguna metodología es el foco de esta tesis.

Este trabajo aborda el problema de la carga y actualización del DW reutilizando el conocimiento generado durante el diseño conceptual y lógico de éste. En particular, se toma como base un algoritmo existente que genera el esquema de la base de datos relacional de un DW, partiendo de un diseño conceptual del mismo, de una base de datos fuente integrada y lineamientos de diseño. Utilizando la información disponible del algoritmo este trabajo analiza los resultados que se obtendrían con un enfoque naive (basándose exclusivamente/directamente en dicho algoritmo), identifica los errores que podrían producirse con este enfoque y propone una solución que presenta un mejor desempeño y resuelve los errores encontrados.

La propuesta continúa la línea de trabajo del grupo CSI en el área de diseño lógico y conceptual de Data Warehouses, complementando las técnicas y algoritmos existentes con soluciones específicas a los problemas de carga y actualización hasta ahora no abordados en dichos trabajos.

Palabras Clave: Data Warehouses, Diseño Lógico de Esquemas Relacionales, Carga de Datos, SQL, Álgebra Relacional.

Contenido

Tabla de Figuras

Tabla de Ejemplos

1 Introducción

En este capítulo se presentan conceptos generales como Data Warehousing, Online Analytical Processing (OLAP), ETL, Data Mart's, etc. A continuación se introducen la motivación y objetivos de esta tesis, se detallan los principales aportes, la organización del resto del documento y se culmina con las convenciones utilizadas.

1.1 Conceptos Generales

Data Warehousing es el término generalmente usado, para definir la tecnología que provee la infraestructura de software utilizada, para los sistemas de soporte a la toma de decisiones y aplicaciones OLAP [BFM99][VGD00]. [Gar98] sin embargo prefiere definir el Data Warehousing como un proceso, no un producto, orientado a ensamblar y manejar información proveniente de múltiples fuentes, con el propósito de ganar una única y detallada vista de una parte o de todo el negocio. La arquitectura clásica de un sistema de DW puede observarse en la Figura 1.

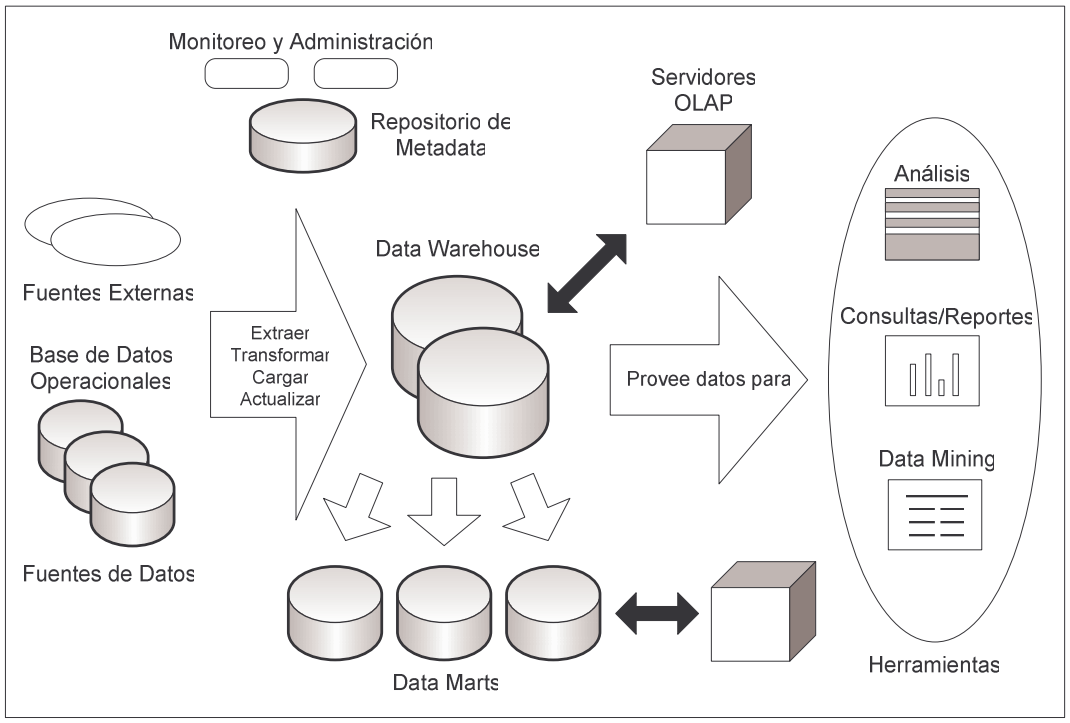


Figura 1: Arquitectura clásica de un sistema de Data Warehousing según [CD97].

En la Figura 1 sobre la parte izquierda se observan las fuentes de datos. Dichas fuentes suelen ser bases de datos (BD) conteniendo información operacional, también llamadas por este motivo **Bases de Datos Operacionales** (BDO). Apoyados sobre las BDO se encuentran los sistemas que permiten el ingreso confiable de información y el procesamiento eficiente de transacciones. A tales sistemas se les llama **OLTP**, por sus

siglas en inglés: On Line Transaction Processing [Kim96]. Por último notar que [CD97] también considera fuentes externas diversas (ej.: sistemas legados, archivos, etc.) que además pueden contener información.

La información de las fuentes es extraída, transformada y posteriormente vuelta a almacenar en otro repositorio. Las herramientas de Extracción Transformación y Carga (**ETL** por sus siglas en inglés) son las piezas de software responsables de dicha tarea. La actualización de los datos extraídos es un problema estrechamente relacionado al ETL (por esto la Figura 1 lo incluye en la flecha) dada la necesidad de mantener al día la información. Las actividades asociadas a la tarea de ETL abarcan: la identificación de la información relevante, su extracción, la unificación en un formato común, la limpieza en función de diversos criterios o reglas de negocio y la propagación hasta el repositorio. El lector interesado puede consultar [VSS02a][VVSKS00] [RH01][GFSS00][RD00] por más información en esta área.

El repositorio en el cual se almacena la información luego de su transformación es denominado **Data Warehouse** (DW). Un DW tal como fue definido por Inmon en [Inm96] es una colección de datos integrados, orientados a un dominio especifico, no volátiles y variables en el tiempo, que ayudan a la toma de decisiones de la empresa u organización. La existencia de este DW abre una serie de problemas relacionados como ser el diseño conceptual y físico de éste [TLS99][GMR98][HLV00][Gar98], su actualización [VGD00][BFM99], el procesamiento de consultas [HRU96][Gup97], etc.

Además del DW principal pueden existir un conjunto de Data Mart's para diferentes necesidades. Un **Data Mart** (DM) contiene información personalizada y sumarizada procedente del DW, orientada a satisfacer requerimientos de análisis específicos de una unidad de negocio (departamento, conjunto de usuarios, etc.).

La consulta interactiva de los DW o DM se conoce como **OLAP**, término que fue introducido por Codd y asociados en [Cod93]. OLAP es el acrónimo en inglés de procesamiento analítico en línea (Online Analytical Processing) y define una solución que suministra respuestas rápidas a consultas analíticas.

La información en el DW o los DMs es almacenada y manejada por uno o más servidores, los cuales presentan una visión multidimensional a una variedad de herramientas de usuario (herramientas de consulta, análisis, reporte, minería de datos, etc.). El término visión multidimensional está asociado al de **base de datos multidimensional** (BDM). Una BDM en lugar de almacenar la información en múltiples tablas bidimensionales (como lo hacen las bases de datos relacionales), representa las claves de las entidades relacionadas como diferentes dimensiones. Esto es, una base de datos multidimensional ofrece una extensión a los sistemas relacionales que provee una visión multidimensional de los datos. El lector interesado puede ampliar en [CD97][Inm96][Kim96][Sho97].

Finalmente existe un repositorio para almacenar y manejar metadata además de herramientas para monitorear y administrar el sistema de DW. Una reseña sobre el tema de Data Warehousing puede ser encontrada en [Wid95][WB97][CD97]. [Vas00] clasifica los trabajos relacionados al tema, publicados desde 1995 hasta 1999, en tres de las más significativas conferencias en el área de base de datos (PODS, SIGMOD, VLDB).

De los problemas aquí mencionados esta tesis se centra en el área de ETL, en particular en la transformación de los datos (T).

1.2 Motivación y Objetivos

El Grupo de Concepción de Sistemas de Información (CSI) ha trabajado en el área de diseño lógico y conceptual de esquemas del DW, pero hasta el momento el tema de ETL no se ha tratado explícitamente. En particular en lo que es diseño lógico y conceptual, el trabajo de [Per01], propone que un modelo multidimensional representado en **CMDM**[Car00], puede ser implementado sobre una base de datos relacional, aplicando un algoritmo que genera transformaciones en los esquemas de las fuentes de datos.

En [Per01] se presenta un algoritmo que, tomando como entrada el modelo conceptual CMDM del DW, el esquema de la base de datos fuente (BDF), un conjunto de mapeos o correspondencias entre estos dos (CMDM y BDF) y lineamientos de diseño sobre la solución deseada, genera el esquema relacional del DW. Un resultado adicional del algoritmo es la secuencia de operaciones que transforma el esquema fuente en el esquema del DW.

Inmediatamente después de tener el esquema aparece la necesidad de implementar las transformaciones para los datos. Esto es, generar el proceso de carga del DW que toma los datos de la BDF y los coloca en el éste.

La carga y actualización de los datos del DW diseñado es un problema que no fue abordado en [Per01]. Utilizar los trabajos tradicionales en el área de carga sin considerar lo ya realizado en [Per01], supondría que el diseñador, luego de preparar la especificación del DW y lo necesario para su traducción a un esquema relacional, debe además enfrentar desde cero el problema de la carga y actualización de los datos. Por otro lado existe un conjunto de información ya recogida o generada (CMDM, BDF, lineamientos, etc.) que seguramente sea requerida para elaborar cualquier propuesta de carga para el esquema.

Este trabajo plantea y analiza una solución para resolver el problema de carga y actualización, aprovechando la información utilizada para generar el esquema de la base de datos. Para esto se exploran los posibles esquemas generados, a través del estudio de las secuencias de operaciones de transformación producidas. Luego a partir de este estudio, se propone un algoritmo para generar una solución de carga, tomando como entrada la misma información utilizada en [Per01], o sea la que ayudó a construir el esquema de la base de datos.

1.3 Principales Aportes

El principal aporte de este trabajo es la presentación de una solución al problema de carga y actualización para un DW diseñado utilizando la metodología propuesta por [Per01].

Partiendo de que el enfoque utilizado por [Per01] plantea la transformación de esquemas mediante primitivas, este trabajo realiza una recorrida por las propuestas que los distintos autores han realizado en diferentes contextos. Luego de esto se posiciona este trabajo con respecto a las demás propuestas realizadas.

Por otro lado se analizan en profundidad los resultados que pueden ser alcanzados utilizando un enfoque naive para la resolución del problema. En este análisis se descubren problemas y oportunidades para mejorar la propuesta que a primera vista pasaban inadvertidos.

Este trabajo presenta un algoritmo que permite generar una sentencia SQL para cada relación en el DW, llevando ahora el problema a un contexto donde existe más literatura y posibilidad para trabajos futuros. Este algoritmo utiliza como entrada la misma información que en su momento se usó para generar el esquema, lo cual significa que es directamente aplicable luego de haber obtenido el diseño del esquema. Dicho algoritmo también maneja correctamente los problemas detectados en el enfoque naive y propone una solución que en ciertos casos cuenta con un mejor desempeño (considerando ejecución sobre un RDBMS).

1.4 Organización de la Tesis

Esta tesis ha sido organizada en cinco capítulos (incluyendo éste) y cinco apéndices. Los capítulos 3 y 4 contienen las principales contribuciones. A continuación se presenta una breve descripción de cada capítulo y los apéndices.

1.4.1 Capítulo 2: Trabajos Relacionados

Este capítulo analiza trabajos relacionados con los temas abordados en este documento. Primeramente se reseñan distintos enfoques para el problema de transformación, con el objetivo de establecer cuales son las posibilidades para implementar la transformación de los datos (principal problema considerado por este trabajo). Luego se discuten las soluciones planteadas para los problemas típicos: ejecución, optimización, etc. Se cierra con las conclusiones y el posicionamiento de este trabajo con respecto a los distintos enfoques.

1.4.2 Capítulo 3: Enfoque Naive

En este capítulo se analiza una primera alternativa (Naive) para realizar la carga. Ésta resulta bastante intuitiva y además puede desprenderse de la propuesta de [Per01] (trabajo sobre el cual se apoya esta tesis). Este análisis se lleva a cabo siguiendo paso a paso el algoritmo de [Per01] durante su ejecución, observando finalmente que construcciones se generaron y con que propósito.

Luego de analizar el resultado obtenido por la propuesta se identifican distintos problemas. Estos errores dan la entrada necesaria para el capítulo 4, donde con el objetivo de corregir dichos problemas y mejorar aspectos que se fueron observando, se introduce una nueva alternativa.

1.4.3 Capítulo 4: Propuesta de Carga

El capítulo 4 presenta la propuesta para la solución de carga postulada por este trabajo. Esta propuesta surge de la corrección de diferentes problemas que se pueden suceder en la propuesta Naive, al mismo tiempo que responde a posibilidades de mejora y optimización que se fueron identificando previamente.

En este capítulo se vuelve a hacer una recorrida por el algoritmo pero esta vez con el interés de analizar cual es el resultado de la nueva propuesta. En el transcurso es necesario analizar los resultados y componerlos en un resultado final que se denomina Vista Unificada, la cual apunta a representar la transformación precisamente de una forma unificada.

Para finalizar se analiza cómo se solucionan cada uno de los problemas identificados, se establece un algoritmo para generar la Vista Unificada y se realiza un breve análisis de costos que permite mostrar la mejora obtenida.

1.4.4 Capítulo 5: Conclusiones

El último capítulo de esta tesis contiene las conclusiones y trabajos futuros.

1.4.5 Apéndices

Esta tesis incluye cinco apéndices. En el apéndice A se presenta el algoritmo propuesto en [Per01], dado que es citado repetidas veces y resulta esencial para el entendimiento de este trabajo.

En el apéndice B se incluyen las reglas en función de las cuales trabaja el algoritmo presentado en el apéndice A. Además del hecho de que hay modificaciones realizadas especialmente para contemplar situaciones presentadas por este trabajo, también resulta esencial contar con este apéndice para el entendimiento del algoritmo.

En el apéndice C se presentan las primitivas utilizadas por las reglas para describir las transformaciones realizadas por el algoritmo. Como antes, además de la necesidad de incluirlas para el entendimiento de la propuesta, se han realizado también modificaciones. Las primitivas cuentan con una descripción de la transformación de instancias que realizan[1], que en casos particulares tuvo que ser modificada, ya que o no resultaba totalmente útil, o no estaba expresada en SQL.

El apéndice D presenta distintas extensiones y lemas que fueron necesarias a lo largo del documento.

En el apéndice E se presenta un caso de estudio, que es muy similar al presentado en [Per01]. La razón para la similitud es tener un punto de comparación con el resultado de la ejecución del algoritmo de generación de esquema. La razón para que no sean iguales es que ciertos aspectos (NCalcME) que se comentan en el documento, se analizan con situaciones puntuales que el caso original no incluía.

1.5 Convenciones de Escritura

Con el fin de mejorar la legibilidad de este documento se ha tomando el siguiente conjunto de convenciones:

- **Negrita**, las definiciones de nuevos términos, así como los principales términos no convencionales, se remarcan de esta forma la primera vez que son utilizados.
- "*Cursiva*", las citas sobre el contenido de otro trabajo, se distinguirán por el uso de comillas dobles y letra cursiva.
- Fuente Arial, ha sido utilizada para marcar palabras y conceptos referidos a figuras o ejemplos.
- `Fuente Courier New`, ha sido utilizada para las secciones de código presentadas.
- Los ejemplos no se indentan como los párrafos normales del documento para facilitar su identificación.

[1] Recordar que las primitivas son primariamente transformaciones de esquema.

2 Trabajos Relacionados

Las herramientas de Extracción Transformación y Carga (ETL por sus siglas en inglés) son las piezas de software responsables de la extracción de información desde las fuentes, su transformación según las necesidades del problema y su inserción en el repositorio destino.

En este trabajo se ataca un problema que perfectamente encaja con el concepto de ETL. Siguiendo una secuencia de transformaciones, que primariamente fueron utilizadas para la construcción del esquema, este trabajo se aboca a proponer cómo manipular los datos subyacentes, focalizándose en las transformaciones, más que en la extracción y la carga.

Este capítulo primero hace una reseña sobre cómo otros autores enfocaron el problema de transformación, para luego presentar las conclusiones sobre los trabajos presentados y cómo se posiciona este trabajo con respecto a ellos. Las presentaciones que se harán a continuación, tratan el tema de transformación pero en contextos que varían desde el diseño de DW, la limpieza de datos[(2)], la actualización, etc.

2.1 Reseña de Enfoques

2.1.1 Theodoratos – Ligoudistianos – Sellis

Theodoratos et al. trabajan sobre el enfoque de diseño orientado a vistas materializadas[TLS99]. En [TS99] se señala que la arquitectura clásica de un DW es como la que se muestra en la Figura 2. En ésta, la información de cada capa es derivada de la información de las capas anteriores, y en los niveles más bajos se encuentran las fuentes operacionales.

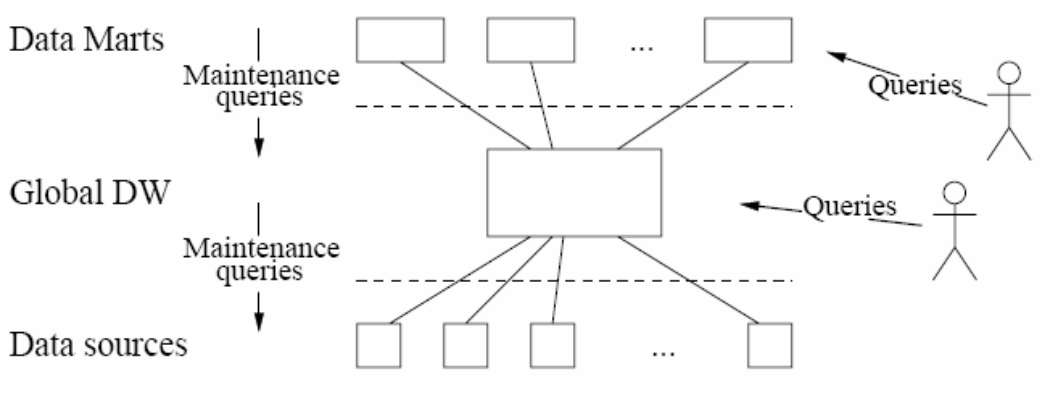


Figura 2: Arquitectura de un DW por Theodoratos y Sellis.

[2] Para el lector interesado [RD00] presenta los problemas y enfoques en el área de limpieza.

De esta forma, el problema de diseño de un DW consiste en encontrar y mantener las vistas que lo definen. Las transformaciones necesarias para llegar desde la base fuente hasta el DW se expresan en las vistas a diseñar. En palabras de los autores [TS99]:

> "*The DW design problem consists of selecting a set of views to materialize in the DW such that:*
> *(1) The materialized views fit in the space available at the DW.*
> *(2) All the queries can be answered using this set of materialized views (with-out accessing the remote source relations).*
> *(3) The combination of the query evaluation cost and the view maintenance cost (operational cost) of this set of views is minimal.*"

Otros autores argumentan que este enfoque resulta muy simplista para el problema en cuestión [BFM99], [VSS02a], [VGD99]. En [WB97] se discuten las ventajas y desventajas de este enfoque.

2.1.2 Peralta (DWDesigner)

En [Per01] se presenta un algoritmo para la generación semiautomática de un esquema relacional para un DW a partir de una base de datos fuente integrada, un esquema conceptual modelado con CMDM[Car00] y un conjunto de correspondencias entre la base de datos fuente y el esquema integrado. El algoritmo produce como resultado un esquema relacional y además una **Traza de Primitivas**[Mar00] también llamada **Traza de Diseño** (o simplemente Traza).

Las **primitivas** (presentadas en el apéndice C) son construcciones que abstraen las operaciones más típicas de la tarea de diseño de DW[MR02]. Estas primitivas toman sub-esquemas relacionales y producen otros sub-esquemas relacionales, resultado de la aplicación de éstas. La Tabla 1 muestra las primitivas definidas por [Mar00] y [Per01]. Una traza de primitivas (Figura 3) es esencialmente un DAG [3], cuyos nodos son primitivas y aristas que conectan las salidas de una primitiva con la entrada de otra.

P1: Identity	P2: Data Filter	P3: Temporization	P4: Key Generalization
P5: Foreign Key Update	P6: DD-Adding	P7: Attribute Adding	P8: Hierarchy Roll Up
P9: Aggregate Generation	P10: Data Array Creation	P11: Partition by Stability	P12: Hierarchy Generation
P13: Multidimensional Break Off	P14: New Dimension Crossing	Q1: Relation Join[4]	Q2: Attribute Renaming[4]
Q3: Instance Filtering[4]	Q4: Primary Key Modification[4]		

Tabla 1: Primitivas de [Mar00].

[3] Grafo dirigido y acíclico.
[4] Primitiva agregada por [Per01].

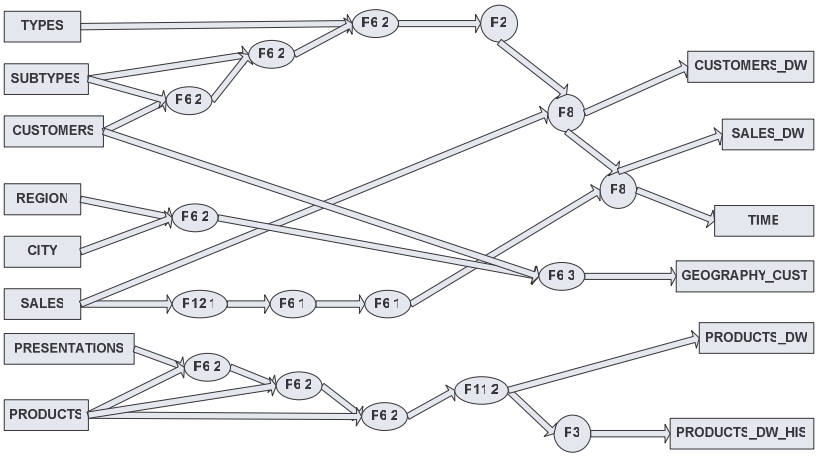


Figura 3: Ejemplo de traza de primitivas de [Mar00].

La traza de primitivas representa una transformación entre el esquema fuente y el DW, la transformación está expresada en primitivas que son operaciones principalmente de esquema pero que además contienen información sobre cómo transformar los datos que puede derivar en una expresión del proceso de carga. La herramienta sobre la que se implementa esta propuesta y como se referirá a continuación lleva el nombre de DWDesigner.

2.1.3 Vassiliadis – Vagena – Skiadopoulos (ARKTOS)

En [VSS02] el problema de ETL se modela con el llamado Architecture Graph, considerándose éste como una combinación de actividades de ETL y almacenamiento de datos. Para modelar el Architecture Graph se utiliza un conjunto de entidades (Figura 4) unidas por cinco tipos particulares de aristas que recogen las características de las interacciones posibles entre éstas. Las aristas pueden ser: Instance-Of (qué tipo), Part-Of (atributos y parámetros asociados a las actividades), Regulator (población de los parámetros), Provider (flujo de datos) o Derived Provider Relationships (cálculo, cómputo).

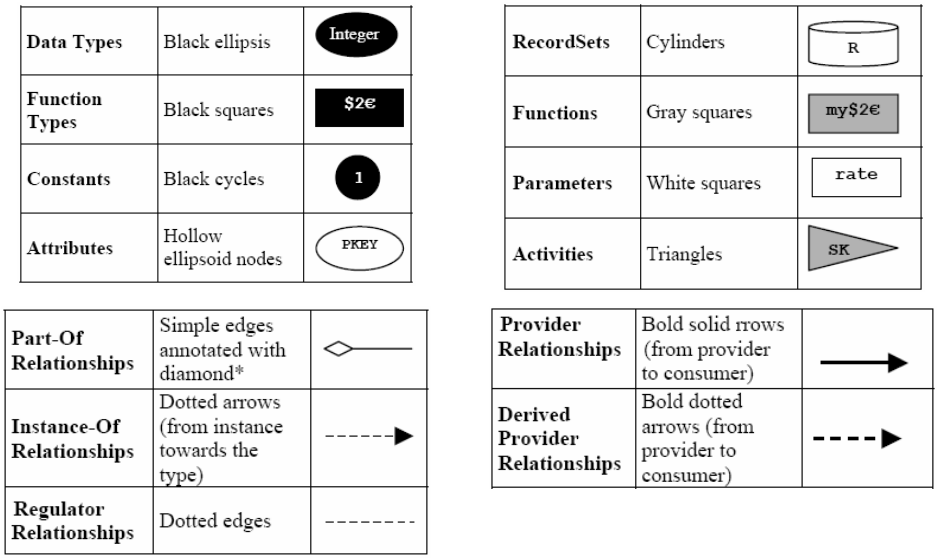


Figura 4: Entidades de [VSS02].

La Figura 5 muestra un ejemplo donde se está propagando la información de una tabla PARTSUPP(PKEY, DATE, QTY, COST) de la fuente a otra tabla DW.PARTSUPP(PKEY, SUPPKEY, DATE, QTY, COST). Más allá de los detalles el principal aspecto a considerar en esta figura es la forma en que se construye el grafo a partir de los elementos. Notar la presencia de los atributos a lo largo de toda la transformación (PKEY, QTY, etc.), las fuentes de datos (DS.PS1, LOOKUP_PS), las actividades de transformación (AddSPK1, SK1) y su descomposición (Add_const1), así como la obtención de parámetros (PAR).

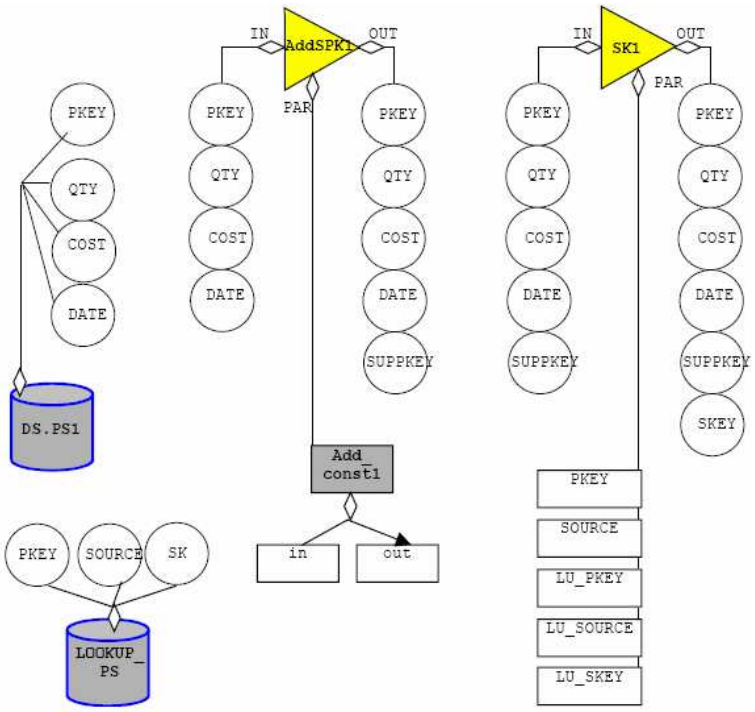


Figura 5: Ejemplo de transformación [VSS02].

En [VSS02a], los autores trabajan sobre la misma idea, presentando más claramente el modelo subyacente y la paleta de actividades (triángulos) propuestas para el común de los casos. La Tabla 2 presenta la paleta de actividades, pero cabe mencionar que el modelo puede ser extendido por el usuario.

Filtros: Selección. No nulo. Violación de Clave Primaria. Violación de Clave Foránea. Valor Único. Incompatibilidad de Dominio.	**Transformaciones Binarias**: Unión. Join. Diferencia. Detección de Actualizaciones.	**Operaciones sobre Archivos**: Conversión de EBCDIC a ASCII. Ordenamiento de Archivo.
Transformaciones Unitarias: Push. Agregación. Proyección. Aplicación de Función. Asignación de Clave Sustituida. Normalización de Tupla. De-normalización de Tupla.	**Operaciones de Transferencia**: FTP. Comprimir / Descomprimir. Encriptar / Desencriptar.	**Transformaciones Compuestas**: Cambio lento de dimensión. Incompatibilidad de Formatos. Conversión de tipo de dato. Selección. Unión extendida.

Tabla 2: Actividades propuestas para el proceso en [VSS02a].

Finalmente hemos de mencionar que ARKTOS es la herramienta resumen de lo anterior, presentada por Vassiliadis y compañía en [VVSKS00].

2.1.4 Raman – Hellerstein (Potter's Wheel)

En [RH01] los autores presentan un sistema interactivo de limpieza. El usuario paulatinamente construye una transformación de los datos agregando, alterando o eliminando transformaciones en una interfase al estilo hoja de cálculo, donde éste puede observar inmediatamente el efecto de cada acción. La Figura 6 muestra el conjunto de transformaciones al tiempo que la Figura 7 muestra un ejemplo de la aplicación de éstas.

Transform	Definition		
Format	$\phi(R, i, f)$	$=$	$\{(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n, f(a_i)) \mid (a_1, \ldots, a_n) \in R\}$
Add	$\alpha(R, x)$	$=$	$\{(a_1, \ldots, a_n, x) \mid (a_1, \ldots, a_n) \in R\}$
Drop	$\pi(R, i)$	$=$	$\{(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n) \mid (a_1, \ldots, a_n) \in R\}$
Copy	$\kappa((a_1, \ldots, a_n), i)$	$=$	$\{(a_1, \ldots, a_n, a_i) \mid (a_1, \ldots, a_n) \in R\}$
Merge	$\mu((a_1, \ldots, a_n), i, j, \text{glue})$	$=$	$\{(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_{j-1}, a_{j+1}, \ldots, a_n, a_i \oplus \text{glue} \oplus a_j) \mid (a_1, \ldots, a_n) \in R\}$
Split	$\omega((a_1, \ldots, a_n), i, \text{splitter})$	$=$	$\{(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n, \text{left}(a_i, \text{splitter}), \text{right}(a_i, \text{splitter})) \mid (a_1, \ldots, a_n) \in R\}$
Divide	$\delta((a_1, \ldots, a_n), i, \text{pred})$	$=$	$\{(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n, a_i, \text{null}) \mid (a_1, \ldots, a_n) \in R \wedge \text{pred}(a_i)\} \cup$ $\{(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n, \text{null}, a_i) \mid (a_1, \ldots, a_n) \in R \wedge \neg\text{pred}(a_i)\}$
Fold	$\lambda(R, i_1, i_2, \ldots i_k)$	$=$	$\{(a_1, \ldots, a_{i_1-1}, a_{i_1+1}, \ldots, a_{i_2-1}, a_{i_2+1}, \ldots, a_{i_k-1}, a_{i_k+1}, \ldots, a_n, a_{i_l}) \mid (a_1, \ldots, a_n) \in R \wedge 1 \leq l \leq k\}$
Select	$\sigma(R, \text{pred})$	$=$	$\{(a_1, \ldots, a_n) \mid (a_1, \ldots, a_n) \in R \wedge \text{pred}((a_1, \ldots, a_n))\}$

Figura 6: Transformaciones de [RH01].

La transformación Format aplica una función a todos los valores en una columna. Add, Drop y Copy permiten agregar, eliminar o copiar columnas. Merge concatena los valores en dos columnas y Split divide una columna en dos o más. Divide a su vez divide condicionalmente una columna, enviando el resultado de acuerdo al predicado a una de dos nuevas columnas. Select como es de esperarse selecciona filas a partir de un predicado. Fold convierte una fila en múltiples, al mismo tiempo que existe Unfold para la operación opuesta.

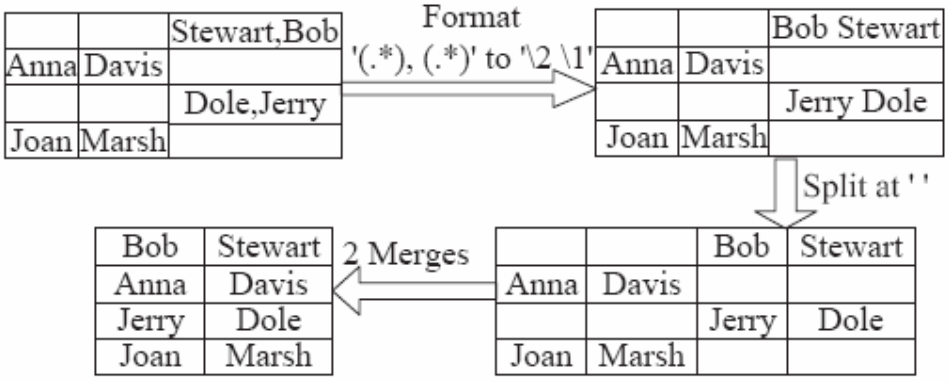


Figura 7: Aplicación de las transformaciones de [RH01].

Finalmente debemos comentar que una vez que el usuario está satisfecho con la transformación que ha logrado, el enfoque de la herramienta es compilar la transformación en un programa ya sea: C, Perl o un formato propietario para reutilizar transformaciones.

2.1.5 Galhardas – Florescu – Shasha – Simon (AJAX)

En [GFSS00] los autores presentan una solución para el problema de limpieza de información. En este trabajo los autores proponen un framework que modela el proceso de limpieza de datos como un grafo dirigido y acíclico de transformaciones atómicas.

El grafo se conforma con 5 tipos de transformaciones: Mapping (estandariza los formatos de los datos), Matching (busca pares de registros que muy probablemente se correspondan con el mismo objeto), Clustering (agrupa parejas con alto grado de similitud), Merging (colapsa cada cluster individual en una tupla resultante), y finalmente SQL View (se corresponde con una operación de join o unión SQL). Para la especificación de cada transformación se enriquece el lenguaje SQL. La Figura 8 y la Figura 9 muestran parte de la sintaxis de dicha extensión.

```
<data cleaning program> ::
      (<declaration of complex type>)*
      (<registration of functions and algorithms>)*
      (<declaration of input data flow>)+
      (<transformation definition>)+
```

Figura 8: Estructura de un programa AJAX.

```
<transformation definition> ::
   <CREATE> (<MAPPING>|<MATCHING>|<MERGING>|<CLUSTERING>|<SQLVIEW>)
      <output data flow>
   (<FROM><input data flow>+)|(<USING><input data flow>)
   [<LET>(<variable>=<expression>(,<expression>)*)+]
   [<BY><algorithm>
   [<WHERE><predicate>(<AND>|<OR><predicate>)*]
   ({<SELECT>(<variable or attribute name>)+[<INTO><output data flow>]
      [<KEY><name>(,<name>)*]
      (<CONSTRAINT>(<expression><DEP><expression>|
         <CHECK><expression>|
         <UNIQUE><column name>|
         <NOT NULL><column name>|
         <FOREING KEY><column name>
            <REFERENCES><output data flow>(< attribute name>))*
   })*
```

Figura 9: Transformación en un programa AJAX.

Las operaciones básicas también pueden ser encadenadas para conformar actividades más complejas. En el trabajo se presentan tres transformaciones complejas: Formatting (que intenta extraer la estructura de los datos o convertir formatos), Duplicate Elimination (encuentra y remueve registros duplicados), Multiple Table Matching (join en base a su similitud entre distintos flujos de datos). La Figura 10 presenta un ejemplo de los grafos generados por AJAX: a la izquierda se aprecian las transformaciones atómicas que a la derecha son utilizadas en transformaciones complejas. En [GFSS00a] también se puede obtener más detalle sobre algunos de los puntos de este trabajo.

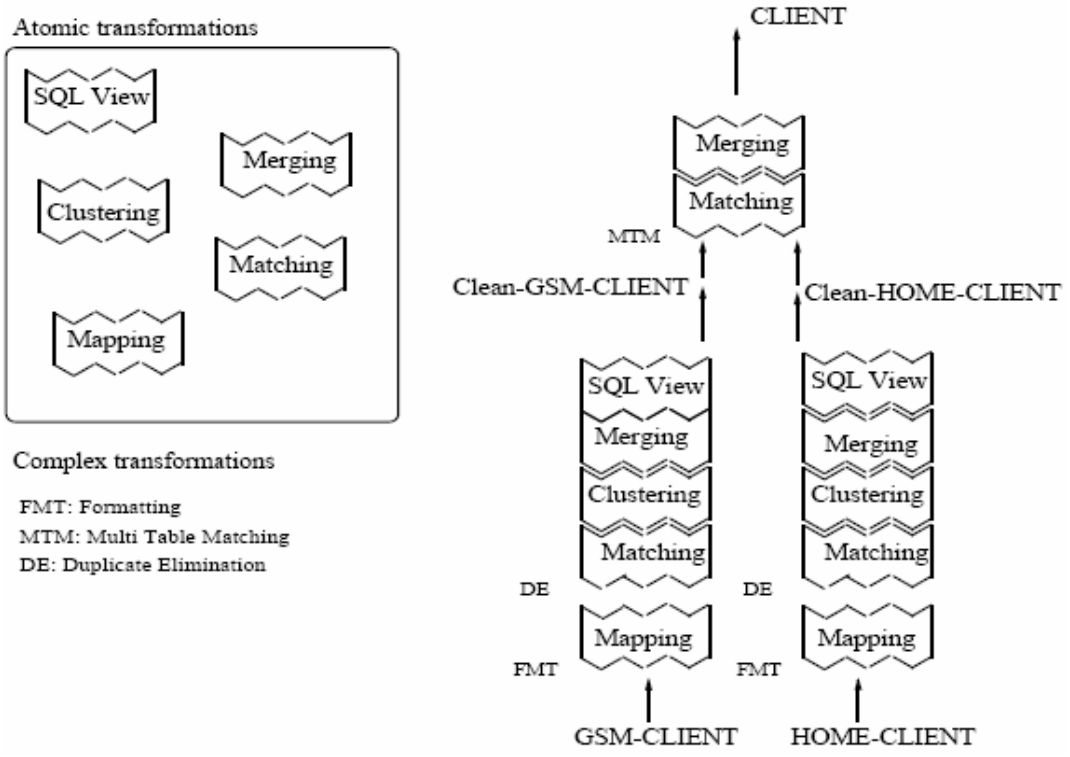


Figura 10: Ejemplo de aplicación de AJAX.

2.1.6 Claypool – Rundestein (Sangam)

En [CR03] Claypool y Rundestein presentan un enfoque que plantea modelar el problema de transformación sobre un grafo, llamado Grafo de Sangam y luego realizar las transformaciones sobre éste. El enfoque se basa en que el Grafo de Sangam es un meta-modelo sobre el cual se pueden expresar los otros modelos existentes (relacional, XML, etc.).

En la Figura 11 se puede apreciar un ejemplo donde el DTD de la izquierda se ve representado como un Grafo de Sangam a la derecha. En este ejemplo tanto los elemento como los atributos son representados como nodos en el grafo. La arista 1 entre el nodo etiquetado como item y el etiquetado como location representa la relación entre el elemento item y el sub-elemento location; de forma similar la arista 4 entre item y id representa la relación del elemento con su atributo. La notación [1:1] indica las cardinalidades de los extremos de la arista y observando la arista 1 del nodo mailbox se puede encontrar un ejemplo de otra posible cardinalidad [0:n]. Finalmente, más allá del ejemplo es importante rescatar que el modelo de datos definido por el DTD está siendo representado como Grafo de Sangam.

```
<!ELEMENT item (localtion,
mailbox, name>
<!ATTLIST item id ID
#REQUITED featured CDATA
#IMPLIED>
<!ELEMENT location
(#PCDATA)>
<!ELEMENT mailbox (mail*)>
<!ATTLIST mailbox id CDATA>
<!ELEMENT mail (from, to,
date)>
<!ATTLIST mail text CDATA>
<!ELEMENT from (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT name (firstName,
lastName)>
<!ELEMENT firstName
(#PCDATA)>
<!ELEMENT lastName
(#PCDATA)>
```

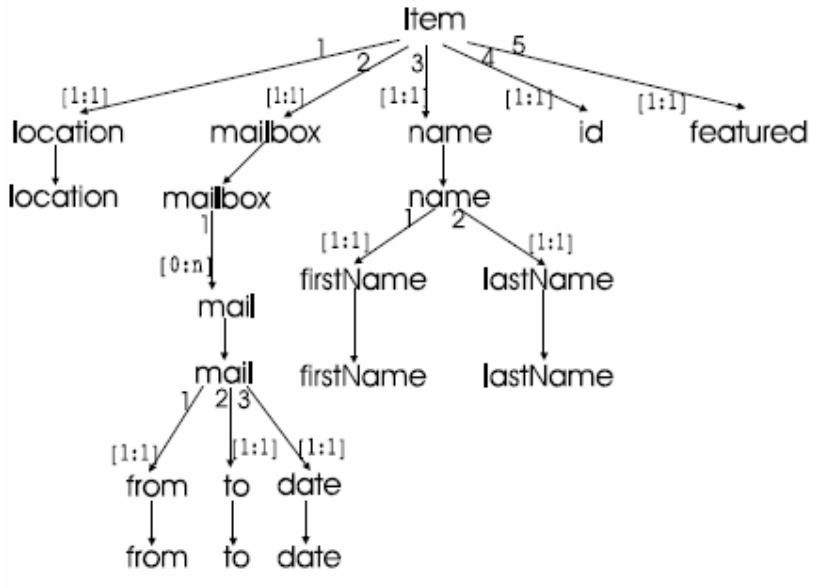


Figura 11: Ejemplo de Grafo de Sangam.

Sobre este meta-modelo (Grafo de Sangam), Claypool y Rundestein plantean dos tipos de herramientas para realizar las transformaciones: una llamada The Bricks o Cross Algebra y otra llamada The Mortar. El álgebra (The Bricks) contiene 4 operadores básicos: cross(Figura 12a), connect(Figura 12b), smoth(Figura 13a) y subdivide(Figura 13b), los cuales podrían ser clasificados como operadores para la transformación de grafos (correspondencia de nodos, aristas, operaciones de join y split entre grafos).

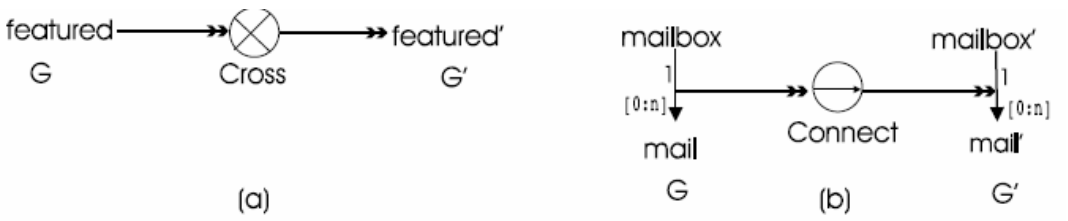


Figura 12: Operadores Cross (a) y Connect (b).

Figura 13: Operadores Smooth (a) y Subdivide (b).

La segunda herramienta consta de la aplicación de dos técnicas de composición: Dependencia de Contexto y Derivación. Las dependencias de contexto (Figura 14) posibilitan a un conjunto de operadores del álgebra trabajar en conjunto sobre un sub-grafo para producir otro grafo. La técnica de derivación por su parte permite el anidamiento o encadenamiento de operadores de forma de que la salida de un operador se convierta en entrada de otro.

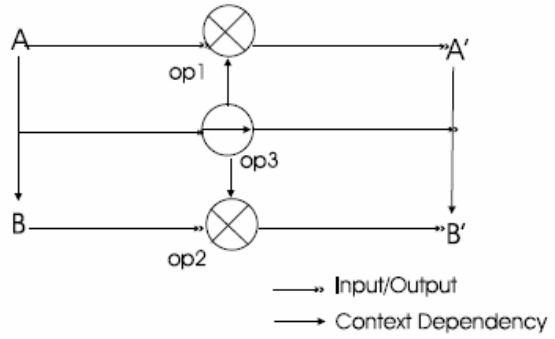


Figura 14: Ejemplo de Dependencia de Contexto.

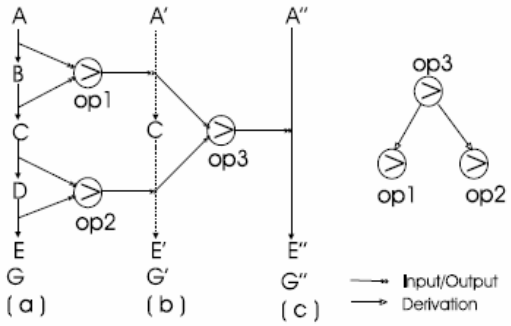


Figura 15: Ejemplo de Derivación.

Finalmente los autores plantean entonces, llevar los esquemas definidos (relacional, XML, etc.) a su representación en grafos de Sangam, para luego transformarlos y volverlos a poner en su representación original. En [CR02] se puede obtener más detalle sobre este trabajo.

2.1.7 Vavouras – Gatziu – Dittrich (SIRUS)

En [VGD99], [VGD99a], [VGD00] se presenta SIRIUS (Supporting Incremental Refreshment of Information Warehouses) un proyecto que plantea un enfoque para definir y ejecutar procesos de actualización de Data Warehouses. En el particular que afecta a este trabajo SIRUS propone un conjunto de opciones para definir las transformaciones entre las fuentes y el DW. En la Figura 16 se puede apreciar el meta-modelo propuesto por SIRUS para modelar el problema de actualización y en particular sobre la parte inferior izquierda el modelado de transformaciones entre la fuente y el DW.

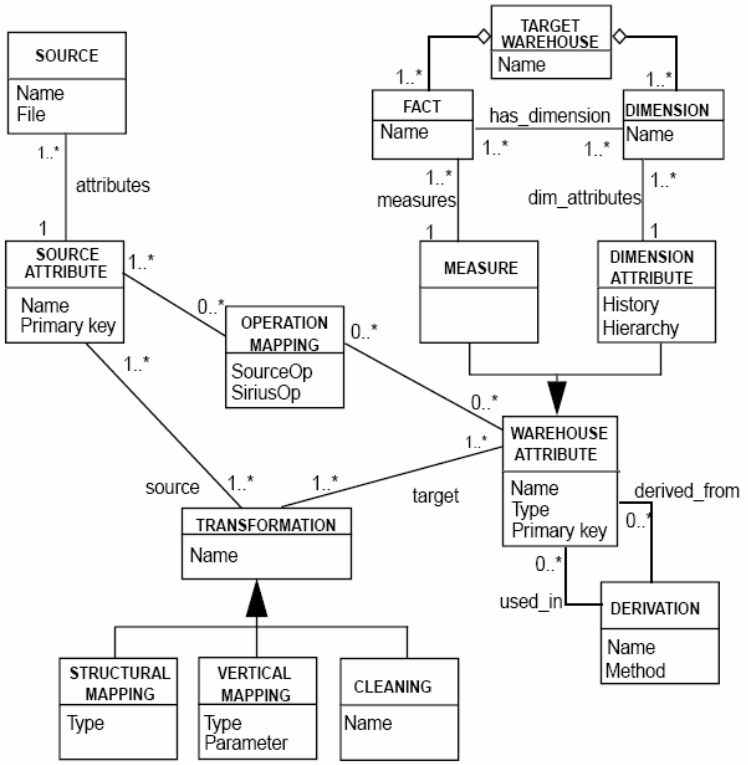


Figura 16: Meta-modelo propuesto por SIRUS.

En SIRUS se proponen tres tipos de transformaciones entre la fuente y el DW. Primeramente los llamados Structural Mapping's, que representan correspondencias entre atributos de exactamente una fuente y atributos del DW. Se distinguen tres tipos: 1:1 (un atributo de la fuente define su correspondiente en el DW), 1:n (obtener n atributos del DW a partir de uno de la fuente) y n:1 (fusionar n atributos de la fuente para obtener uno en el DW).

La siguiente propuesta para representar la transformación son los Vertical Mapping's, en este caso a diferencia del anterior se alimenta el atributo del DW a partir de más de una fuente. El objetivo de este tipo de transformaciones es la integración de múltiples fuentes. Se mencionan como opciones para este tipo de transformación la unión, intersección, diferencia, el filtrado (selección) y priorization[5].

Por último el tercer tipo de transformación Cleaning refiere a la posibilidad de: corregir anomalías, eliminar duplicados e información inconsistente, transformar o enriquecer la información proveniente de las fuentes, etc. SIRUS menciona que provee un conjunto básico de operaciones pero se menciona que la limpieza de información no es el foco principal del proyecto y se refieren otros trabajos.

Además de los tres tipos de transformación antes presentados SIRUS presenta otro tipo de correspondencias relativas al impacto de las operaciones en las fuentes (Operation Mapping). Finalmente vale indicar que la solución de SIRUS trabaja en función de Wrappers donde diversos componentes interactúan con éstos para popular el esquema global, refrescar los datos, etc.

2.2 Clasificación de los Enfoques

En esta sección se discuten y clasifican los trabajos mencionados anteriormente. La clasificación se realiza primeramente en función de la naturaleza del problema (construcción de DW, limpieza, etc.). También se analiza si utilizan SQL o algún tipo de operación o correspondencia, si las operaciones que cada propuesta define están asociadas al problema de DW, si existe un álgebra, cómo se realiza la ejecución y si existen propuestas de optimización. Para finalizar se presentan conclusiones y se posiciona este trabajo con respecto a los demás analizados.

2.2.1 Problema Principal

La primera clasificación que se puede hacer sobre los trabajos citados anteriormente refiere a la naturaleza del problema principal. Aunque todos fueron incluidos por tratar el tema de transformación desde distintos enfoques, se puede apreciar que cada uno tiene su foco en diferentes problemas. Por un lado la propuesta de Theodoratos y el DWDesigner se centran en la construcción de DW, mientras que AJAX y Potter's Wheel se enfocan en la limpieza. ARKTOS por su lado trabaja en el problema de ETL, mientras que Sangam principalmente es un framework de transformación. Finalmente a SIRUS lo impulsa el problema de actualización de DW.

[5] Brevemente esta operación permite frente a la presencia de múltiples fuentes para un mismo dato establecer prioridades al considerar cual utilizar (ver [VGD00]).

2.2.2 Orientación

Los trabajos pueden dividirse en tres grupos según si su orientación es hacia utilizar / extender SQL, definir nuevos lenguajes u operaciones o trabajar con correspondencias. Bajo esta clasificación la propuesta de Theodoratos y AJAX[6] representan soluciones orientadas a SQL mientras que DWDesigner, ARKTOS, Potter's Wheel y Sangam introducen sus propios operadores a fin de definir cómo transforman la información (primitivas, actividades, transformaciones y operadores de la cross algebra respectivamente). Finalmente SIRUS trabaja estableciendo correspondencias entre los esquemas y dejando que un motor se encargue de transformar en función de dicha especificación.

Las soluciones basadas en SQL como la propuesta de Theodoratos necesitan incorporar extensiones como las funciones definidas por el usuario (UDF's) presentadas en SQL'99 de forma de permitir realizar ciertas operaciones que no se incluyen en el SQL estándar (ej.: split o merge de atributos). La propuesta de AJAX es una extensión al lenguaje SQL por lo cual el mecanismo de extensión ya fue utilizado. Debería poder usarse nuevamente para las UDF sin inconvenientes. También se podría observar que si se desean soportar genéricamente reestructuraciones de esquema, es necesario incorporar extensiones como las de SchemaSQL[LSS96].

Las ventajas de las soluciones basadas en SQL son la familiaridad con el lenguaje que tendrán los usuarios y la literatura relacionada (materialización, optimización, etc.). Mientras que la desventaja atribuida a esta propuesta es que la complejidad de todas las tareas inherentes al proceso (limpieza, transformación, agregación, manejo de la historia, etc.) puede ser excesiva para un modelo basado en vistas SQL, sobre todo considerando el proceso de actualización [BFM99].

Las soluciones basadas en operaciones tienen la ventaja de una mayor capacidad expresiva, si se observa en el contexto del Data Warehousing. Considerando el enfoque del DWDesigner donde la primitiva Hierarchy Roll-Up (hacer Roll-Up por una jerarquía) expresa claramente la transformación asociada a la operación, si se hace una comparación con la sentencia SQL equivalente presentada por Theodoratos, no resultará claro que ambas expresen la misma transformación. La justificación para esta mayor expresividad es que mientras SQL es un lenguaje de consulta genérico, las otras propuestas enfocan sus operaciones en la transformación específica que desean realizar. Esto es, para cada acción relevante al contexto se tiene una operación, mientras que en SQL el diseñador se debe servir de un conjunto de herramientas ideadas para otro problema, la resolución de consultas.

Entre las desventajas de contar con un nuevo conjunto de operadores, la primera está relacionada a la necesidad de aprender a utilizarlos, dado que se trata de una nueva propuesta. En segundo lugar se puede observar la poca aplicabilidad (en general) de trabajos existentes en problemas conocidos como son la optimización de la transformación, la materialización de resultados, el paralelismo, etc. Mientras que con el enfoque de Theodoratos es directo aplicar una propuesta de optimización a las consultas subyacentes, optimizar el Architecture Graph introducido por ARKTOS es un nuevo problema.

El uso de correspondencias en lugar de SQL u operaciones levanta la discusión a un plano más declarativo, restando importancia a cómo se llegará al resultado. Por esta

[6] AJAX introduce operaciones Mapping, Matching, etc. sin embargo se ve mejor clasificada como extensión de SQL.

razón este enfoque nuevamente puede considerarse más expresivo que SQL e incluso que los enfoques orientados a operaciones. En contraparte, cómo se realiza la transformación resulta ser una suerte de caja negra, que como en el caso de operaciones, difícilmente pueda utilizar trabajos existentes.

2.2.3 Operaciones de DW

Las soluciones pueden ser estudiadas a través de las operaciones que definen, o en el caso de SIRUS a través del objetivo de las correspondencias que establece. Una forma de clasificar los tipos de operaciones (o correspondencias) es en relación al problema tratado, o sea, la transformación de datos en el contexto de Data Warehousing. La clasificación se realiza en función de si las operaciones persiguen objetivos de este contexto o no.

Las primitivas utilizadas por el DWDesigner se presentan como operaciones comunes de la tarea de construcción de DW (Aggregate Generation, Temporization, etc.). En contraposición las operaciones de Potter's Wheel son mucho menos relacionadas con el problema de DW y más asociadas al entorno de hoja de cálculo (Add, Drop, Copy, etc.); el caso de Sangam es similar pero en el contexto de grafos. Los Mappings presentados por SIRUS son totalmente genéricos por lo que no clasifican como operaciones de DW.

El DWDesigner parte de correspondencias como en el caso de SIRUS sólo que da un paso más y plantea una secuencia de transformaciones que equivale a la operativa seguida para obtener un DW que respete las correspondencias. Ésta precisamente es la principal observación que se le puede hacer a SIRUS en lo que tiene que ver con la claridad de la operativa seguida.

ARKTOS por su parte puede clasificarse como mixto, dado que hay operaciones como la agregación (relacionada al Data Warehousing) y al mismo tiempo de conversión entre formatos de caracteres. Las operaciones presentadas por AJAX están asociadas principalmente al proceso de limpieza que es parte de las tareas de DW, por lo cual es más lógico clasificarlas como operaciones de DW. En cuanto al enfoque de Theodoratos, SQL es un lenguaje genérico de consulta claramente utilizado en el contexto del Data Warehousing.

2.2.4 Álgebra

Bajo este título se analiza el hecho de que las propuestas cuenten o no con un álgebra. Dado que gran parte de las propuestas son orientadas a definir un conjunto de operaciones, es conveniente analizar si es posible o no operar de modo algebraico. En este trabajo se hace uso extenso de la capacidad de operar algebraicamente en particular utilizando Álgebra Relacional. No contar con un álgebra dificulta hacer simplificaciones, sustituciones o alteraciones a las transformaciones planteadas, lo cual es conveniente para lograr optimizaciones o alternativas de ejecución.

En el caso de Theodoratos se sabe que las consultas SQL pueden ser expresadas mediante Álgebra Relacional y por lo tanto claramente disponen de un álgebra. Potter's Wheel si bien no declara públicamente un álgebra, a los efectos de este trabajo se considerará que sí cuenta con una, tomando en cuenta la forma en la que se operan, componen, etc. las diferentes transformaciones. En el caso de Sangam se define explícitamente la llamada Cross Algebra.

AJAX por su parte no posee un álgebra aunque extiende de SQL y cuenta con algunas de sus cualidades. DWDesigner, ARKTOS y SIRUS tampoco cuentan con un álgebra. En el caso de este último tampoco cuenta explícitamente con operaciones sino que manejan correspondencias entre esquemas.

2.2.5 Ejecución

Las propuestas se diferencian también por la forma en la cual ejecutan las transformaciones de datos, ya sea para hacer la carga, ejecutar el proceso de limpieza o de ETL. Mientras que la línea de Theodoratos ejecuta directamente en el SGBD, todas las otras propuestas tienen una propuesta propia para ejecutar.

En AJAX aparece la figura de la Execution Engine responsable por la ejecución de las transformaciones. En el DWDesigner, no se ataca directamente la ejecución de la transformación, sin embargo se indica que cada primitiva indica una transformación de las instancias asociadas y que por esa vía podría lograrse la ejecución. En ARKTOS se comenta que las transformaciones son resueltas mediante programación, pero no se menciona explícitamente cómo o a través de que componentes. Potter's Wheel por su parte menciona un Transformation Engine que tiene la posibilidad de generar un programa C o Perl que ejecuta la transformación, previéndose para el futuro permitir también generar la transformación sobre SQL/XSLT (lenguajes declarativos). En Sangam se mencionan diferentes componentes SAG-Loader, CAG-Builder, CAG-Evaluator y CAG-Generator que son los responsables de la ejecución. Recordar que en esta propuesta se transforma del modelo relacional al denominado Grafo de Sangam, se opera y finalmente se lo vuelve a transformar a relacional. Finalmente en SIRUS se presenta el Data Warehouse Refreshment Manager (DWRM) quien es el encargado de coordinar los mediadores cuando realizan las tareas.

2.2.6 Optimización

En el trabajo de Theodoratos la optimización se enfoca desde el punto de vista del SGBD. En el enfoque de AJAX por su parte, además de esto (dado que AJAX es una extensión de SQL) se consideran optimizaciones propias del proceso de limpieza. El DWDesigner no trata el problema de la optimización así como tampoco lo hacen SIRUS Sangam o ARKTOS, sin embargo en estos dos últimos se indica como un trabajo futuro que hasta el momento no ha sido publicado. En el caso de Potter's Wheel, se menciona la existencia de optimizaciones pero no fue posible encontrar las referencias. También se menciona que en trabajos futuros se plantea la posibilidad de generar SQL de forma que un SGBD pueda realizar optimizaciones a la transformación.

2.2.7 Conclusión y Posicionamiento

De lo presentado anteriormente se puede extraer la siguiente tabla a modo de resumen:

Propuesta	Problema Principal	Orientación	Operaciones de DW	Álgebra	Ejecución	Optimización
Theodoratos	Constr. DW	SQL	Si	Si	SGBD	Si
AJAX	Limpieza	SQL	Si	No	Propuesta Explícita	Si
DWDesigner	Constr. DW	Operaciones	Si	No	Propuesta Explícita	No
ARKTOS	ETL	Operaciones	Mixto	No	Propuesta Explícita	No
Potter's Wheel	Limpieza	Operaciones	No	Si	Propuesta Explícita	No
Sangam	Transformación	Operaciones	No	Si	Propuesta Explícita	No
SIRUS	Refresque DW	Correspondencias	No	No	Propuesta Explícita	No

Tabla 3: Clasificación de los diferentes enfoques.

Este trabajo como ya se mencionó se basa en el enfoque de Peralta (DWDesigner). Éste aborda el problema de construcción de un DW realizando una transformación de la base de datos fuente para convertirla en el DW. Todas las propuestas aquí mencionadas realizan dicha transformación sólo que en cada caso persiguiendo objetivos distintos (construcción, limpieza, etc.).

La propuesta que se verá en este trabajo toma las operaciones (Primitivas) del DWDesigner y las traduce a SQL, para de esta forma poder disponer de un álgebra (Álgebra Relacional) que permita operar con éstas y además plantear una solución que habilite la utilización de la literatura relacionada. Esta opción es tanto conveniente por el álgebra, la literatura o el hecho de que puede ejecutar en cualquier SGBD, como porque sigue conservando la capacidad expresiva mencionada en 2.2.2, pues nunca desaparecen las operaciones, sólo se traducen a SQL para su implementación.

Si esta tesis hubiera seguido la línea de trabajar directamente con las primitivas al estilo de las soluciones basadas en sus propias operaciones, se tendrían que haber abierto una serie de líneas de trabajo (proveer implementaciones de las primitivas, diseñar un ambiente de ejecución, etc.) con pocas expectativas de obtener una solución interesante. Recordar también que la naturaleza y propósito de las primitivas es la de representar las tareas típicas de construcción de un DW desde el punto de vista lógico, que mucho dista del problema de cargar los datos.

El foco de este trabajo se centra en analizar las transformaciones realizadas a los datos. No es el objetivo del mismo hacer algún aporte en las áreas de extracción de datos, carga en el repositorio o ejecución del proceso, limitándose simplemente a usar el enfoque de [TLS99] para dar marco a la implementación de la carga y actualización.

3 Enfoque Naive

Este capítulo analiza una primer propuesta para realizar la carga de un DW generado con el algoritmo de [Per01] (presentado en el apéndice A) que de aquí en adelante denominaremos **Algoritmo de Generación del Esquema**. Este análisis es interesante porque muestra una primera alternativa de un proceso de carga candidato, permitiendo así entenderlo, criticarlo y mejorarlo. A continuación se presenta esta primera propuesta y dado que se basa en el algoritmo de generación del esquema las siguientes secciones tratan cada una de las partes de éste, concluyendo con un resumen. Para finalizar se analizan problemas encontrados y se presentan las conclusiones.

3.1 Presentación del Enfoque Naive

Como se mencionó en la sección 2.1.2, las primitivas[Mar00] producen alteraciones sobre un conjunto de esquemas de relación que toman como entrada, para de esta forma definir los esquemas de relación que entregan como resultado. Además de esto, asociada a cada una de las primitivas se encuentra la definición de cuál será la instancia de datos generada tras su aplicación, expresada en función de sentencias SQL[7].

Dado que el algoritmo utilizado para generar el esquema del DW produce como resultado una secuencia de aplicaciones de estas primitivas, una de las formas más simples de enfrentar el problema de carga, es suponer que cada una de las primitivas define una vista que expresa la transformación asociada. El resultado obtenido por el encadenamiento de estas vistas, en particular las últimas vistas de la secuencia, representan el DW deseado. A este enfoque se lo denominará **Naive**.

Ejemplo 1: Enfoque Naive

A modo de ejemplo se utiliza la traza `T = P1(P1(R))`, donde para obtener la relación R_DW en el DW se aplica la primitiva P1 a la relación R y posteriormente a esto se le vuelve a aplicar la primitiva P1 (ver Figura 17).

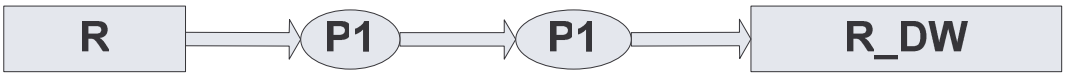


Figura 17: Traza de ejemplo para enfoque Naive.

[7] Existen casos en que las instancias asociadas a las primitivas no se definen como sentencias SQL (primitivas Q2 y Q4), u otros en los cuales la sentencia propuesta no era conveniente (primitivas P4.1, P7); para estos casos propondremos una sentencia SQL. Ver el apéndice Primitivas para más información.

Primitiva P1 – Identity
Descripción: Dada una relación, ésta genera otra que es exactamente la misma de la que se partió.
Entrada:
- Esquema Origen: $R \in Rel$
- Instancia Origen: r

Esquema Resultante:
$R \in Rel \,/\, R' = R$
Instancia Generada:

```
r' = select * from R
```

Una aplicación de la primitiva P1 se transformará en la creación de la vista asociada. El resultado obtenido será:

```
create view V1 as select * from R
```

Luego con la misma estrategia, la segunda aplicación de P1 también genera la misma sentencia SQL, pero en este caso utilizando V^1 en lugar de R, dado que la segunda aplicación de P1 se realiza sobre el resultado anterior.

```
create view V2 as select * from V1
```

Finalmente el DW estaría constituido por la vista V^2 y se podría decir que internamente utiliza una vista auxiliar V^1.

♦

El enfoque Naive, no es una "buena" solución para el problema de carga, pero para dar contexto a esta aseveración es necesario su desarrollo, evidenciando sus falencias. Al mismo tiempo el análisis del enfoque Naive, sienta las bases para formular una mejor propuesta. En este capítulo se analiza el algoritmo de generación del esquema con el objetivo de observar las trazas producidas (el algoritmo se adjunta en el apéndice A). Éstas nos permitirán observar las bondades y debilidades de la propuesta Naive.

El primer objetivo de este capítulo será constatar cuáles serán las primitivas aplicadas en cada paso de la generación del esquema, porqué cada una de éstas luego indica cuál es la vista asociada y de esta forma podremos apreciar cómo se construye la solución Naive (sección 3.4). En grandes líneas el algoritmo de generación del esquema (Figura 18) está dividido en dos grandes secciones:

- Pasos 1-6: Tratamiento de Dimensiones. Genera el esquema para las dimensiones definidas en el esquema conceptual. Analizado en la sección 3.2.
- Pasos 7-15: Tratamiento de Cubos. Genera el esquema para los cubos definidos en el esquema conceptual. Analizado en la sección 3.3.

Los **pasos** son utilizados en [Per01] para organizar la resolución del problema en etapas, enfocadas en obtener resultados puntuales. Cada uno de estos pasos a su vez, se resuelve aplicando una de un conjunto de reglas, pensadas para cada uno de los problemas encontrados. Cada **regla** indica qué primitiva se utiliza y con qué parámetros

se realiza la invocación (las reglas son presentadas en el apéndice B). El análisis contemplará todos los casos posibles, para obtener cuáles son todas las combinaciones de primitivas que se utilizan, o lo que es lo mismo, todas las posibles trazas que se generan.

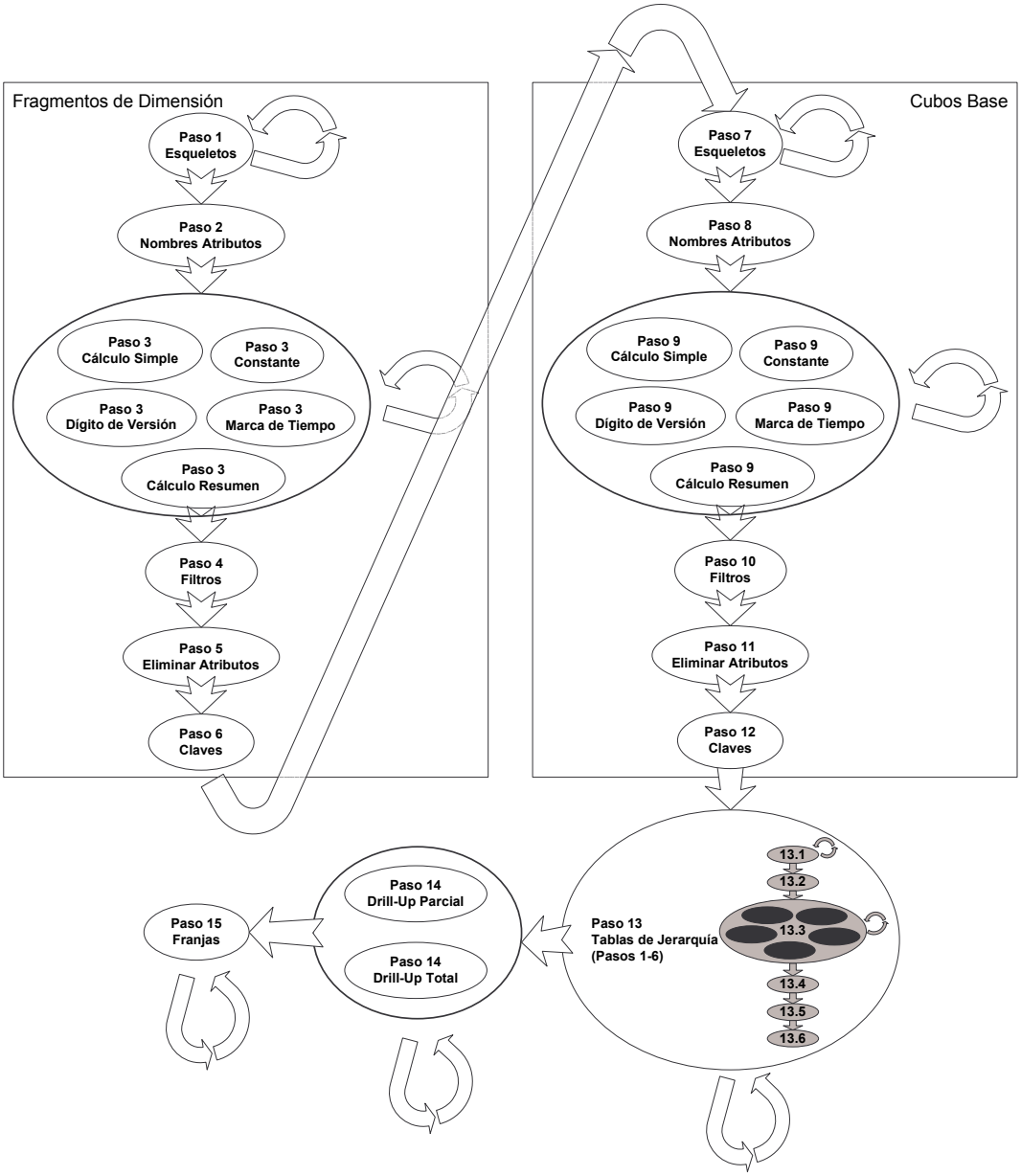


Figura 18: Algoritmo de generación del esquema.

Los pasos mencionados por sus títulos en la Figura 18 (Esqueletos, Nombres de Atributos, etc.) se abordarán en las dos secciones siguientes. Una observación interesante que guiará la presentación, es que el algoritmo contiene partes similares (pasos 1 al 6, 7 al 12, 13.1 al 13.6), debido a que se utilizan las mismas estrategias para resolver problemas similares (materialización de dimensiones y cubos). En la sección siguiente se comenzará analizando las dimensiones que conforman la primer parte del

algoritmo, para posteriormente utilizando las similitudes entre las partes, analizar las restantes.

3.2 Tratamiento de Dimensiones

Esta sección analiza la generación del esquema relacional exclusivamente para las dimensiones del esquema conceptual. Mirando más en detalle los pasos del 1 al 6 que conforman esta parte, se observa que los objetivos de cada uno de éstos son: armar los esqueletos, renombrar los atributos, definir atributos calculados, filtrar las tuplas que no correspondan, eliminar los atributos sobrantes y ajustar la clave.

Estos pasos considerarán cada **Fragmento de Dimensión**[Per01], es decir cada una de las secciones en las cuales se decidió materializar la dimensión del modelo conceptual (CMDM), y tienen el objetivo de producir una relación para cada uno. Las dimensiones están compuestas por **ítems**[Per01] que les dan contenido y permiten entre otras cosas definir los niveles y los fragmentos agrupando estos niveles. Para la vinculación de los ítems con los atributos de la BDF, se utiliza el concepto de mapeo o **correspondencia**[Per01]**,** donde se asocia a un ítem del CMDM una de cuatro tipos de correspondencia posibles.

Las correspondencias definen la forma en que se desean obtener los datos para el ítem, que en definitiva implica la forma en que se cargará el atributo de la relación del DW. Hay correspondencias de tipo **DirectME** (el ítem toma el mismo valor que su atributo correspondiente en la BDF), **1CalcME** (el ítem se obtiene realizando un cálculo en función de atributos de la BDF), **NCalcME** (el ítem se obtiene agregando un conjunto de tuplas de la BDF) y **ExternME** (el ítem se obtiene utilizando una funcionalidad externa, distinguiéndose constantes, marcas de tiempo y dígitos de versión). Estas correspondencias fueron levemente alteradas en el contexto de este trabajo y esto se presenta en el Apéndice D en la sección de Extensiones a las Correspondencias.

Reviendo el algoritmo se observa que estos pasos son centrales, dado que, a no ser por pequeñas modificaciones, se repiten en distintas secciones del algoritmo (pasos 7 a 12 y paso 13).

3.2.1 Paso 1 - Esqueletos

El objetivo del primer paso es crear los **esqueletos**, entendiéndose por esto unificar todas las relaciones de la BDF que tienen correspondencias a ítems del fragmento de dimensión (sólo se consideran DirectME y 1CalcME según [Per01] ver definición de MapTables). Al analizar el paso se puede observar que, para cada fragmento se podrían generar cero o más aplicaciones de la primitiva Q1, dependiendo de la cantidad de relaciones de las cuales se obtengan atributos: si todos los atributos se obtuvieran de una misma relación no es necesario aplicar la primitiva; si los atributos se obtuvieran de N relaciones es necesario generar N-1 aplicaciones de la primitiva. Cada una de las aplicaciones de primitiva generará una vista de la forma[8]:

[8] Ver el Apéndice Algoritmo en la sección A.1 para ver en detalle el paso 1 y el Apéndice Reglas en la sección B.1 para ver la regla utilizada en este paso, el Apéndice Primitivas da a su vez detalles para las sentencias asociadas a cada primitiva y la aquí aplicada se encuentra en la sección C.9.

```
create view V1i as select (Att(Ri1) - Yi1) ∪ (Att(Ri2) - Yi2)
       from Ri1, Ri2
       where f(Ci1, .., Cik)
```

Esta última puede ser simplificada en función de la forma en que se invoca la primitiva Q1 en el algoritmo, dado que no se elimina ningún atributo de las relaciones involucradas (aspecto por el cual se restaban los conjuntos Y_{i1} e Y_{i2}). Por ende, las vistas generadas finalmente serán de la forma:

```
create view V1i as select Att(Ri1) ∪ Att(Ri2)
       from Ri1, Ri2
       where f(Ci1, .., Cik)
```

En la vista anterior 'i' hace referencia al i-ésimo caso donde se aplicó, dado que como se indicó, se podrían generar n vistas. El superíndice 1 colocado en el nombre de la vista creada hace referencia a que es generada por el paso 1 del algoritmo. Esto permitirá referirla con más claridad en pasos posteriores. Finalmente R_{i1}, R_{i2} son relaciones de la BDF, siendo $f(C_{i1}, .., C_{ik})$ la función de join entre éstas. Dicha función proviene del hecho de que las relaciones deben estar vinculadas por un **link**, concepto que es definido en [Per01] con precisamente el objetivo de denotar la forma en que se relacionan las tablas de la BDF. Se analizará el siguiente ejemplo para entender este paso más en detalle.

Ejemplo 2: Construcción de Esqueletos

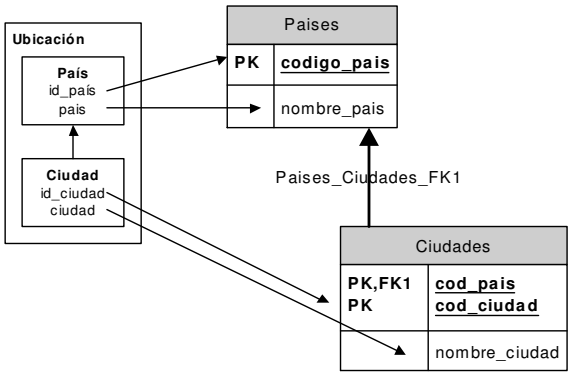


Figura 19: Ejemplo de Aplicación del Paso 1.

Se supone un fragmento Ubicación donde se unen (desnormalizan) dos relaciones: Países y Ciudades. Las relaciones se encuentran vinculadas por un link Países_Ciudades_FK1 en la Figura 19, o sea Ciudades.código_país = Países.cod_país. En tal caso la vista generada será:

```
create view V1i as select Países.código_país, Países.nombre_país,
     Ciudades.cod_país, Ciudades.cod_ciudad, Ciudades.nombre_ciudad
   from Países, Ciudades
   where Ciudades.cod_país = Países.código_país
```

Notar que el atributo Ciudades.cod_país no tiene correspondencia y aún así se encuentra en V^1_i; esto será ajustado posteriormente por el paso 5 del algoritmo. Si los nombres de los atributos no fueran como en este caso diferentes también se encontraría un problema; éste será comentado un poco más adelante en esta sección.

♦

Es conveniente observar con un poco de detalle los casos en los cuales la generación del fragmento involucra a más de dos relaciones, para ello se muestra el siguiente ejemplo:

Ejemplo 3: Relación entre primitivas

Suponiendo que una dimensión Productos, con una jerarquía familia / sub-familia / producto, es modelada con un solo fragmento y obtenida de tres relaciones distintas: familias(idfamilia, descfamilia), subfamilias(idsubfamilia, descsubfamilia, idfamilia) y productos(idproducto, descproducto, idsubfamilia). En particular las correspondencias entre la dimensión productos y estas relaciones son:

```
idfamilia → DirectME(<idfamilia, familias, familias.idfamilia>)
descfamilia → DirectME(<descfamilia, familias, familias.descfamilia>)
idsubfamilia → DirectME(<idsubfamilia, subfamilias,
  subfamilias.idsubfamilia>)
descsubfamilia → DirectME(<descsubfamilia, subfamilias,
  subfamilias.descsubfamilia>)
idproducto → DirectME(<idproducto, productos, productos.idproducto>)
descproducto → DirectME(<descproducto, productos,
  productos.descproducto>)
```

Suponiendo que existen links entre los atributos con igual nombre (familias <-> subfamilias y subfamilias <-> productos) el resultado obtenido al tomar las vistas correspondientes con cada primitiva será:

```
create view V¹1 as
      select idfamilia, descfamilia, idsubfamilia, descsubfamilia
      from familias, subfamilias
      where familias.idfamilia = subfamilias.idfamilia
create view V¹2 as
      select idfamilia, descfamilia, idsubfamilia,
            descsubfamilia, idproducto, descproducto
      from V¹1, productos
      where V¹1.idsubfamilia = productos.idsubfamilia
```

La observación clave es que se utiliza la vista creada primeramente (V^1_1) dado que el algoritmo compone la aplicación de las primitivas al estar constantemente alterando las correspondencias. En el ejemplo esto hace que el link que inicialmente existía entre subfamilias y productos sea alterado luego de la aplicación de la primera primitiva (construcción de la primera vista), para quedar asociando V^1_1 con productos. Dicha acción se realiza mediante un recálculo de las relaciones que tienen correspondencia con el fragmento.

♦

Debe tenerse presente que las correspondencias se van actualizando tras la aplicación de cada primitiva, por lo cual las vistas de la secuencia que se está produciendo se encuentran encadenadas como ya se mostró en el Ejemplo 3. Este trabajo no entra en detalle en ese análisis ya que el enfoque Naive no altera el algoritmo con el que se generan las primitivas y se remite a trabajar sobre el resultado. Cómo se llega a cada vista, es un aspecto que se puede ver en detalle en [Per01].

Antes de continuar, es necesario hacer una pequeña observación sobre el tema de los nombres de los atributos. Es claro que al aplicar la primitiva Q1 a dos relaciones podrían existir dos o más atributos con igual nombre, que incluso pueden representar distintos objetos de la realidad (descfamilia y descsubfamilia podrían llamarse ambos descripción en el Ejemplo 3). Si bien los atributos serán manipulados e incluso algunos eliminados en pasos posteriores del algoritmo, éstos crean un problema en las vistas intermedias a crear ya que no se pueden crear vistas con nombres de atributos repetidos.

Para resolver el problema de los atributos, se recurre a un pequeño cambio en cada una de las vistas a crear que consta de, cambiar el nombre de los atributos sin correspondencia del fragmento (para conservar los nombres que el algoritmo sí asigna a los atributos presentes en el resultado) anteponiéndoles el nombre de la relación y un guión bajo. Entonces dado el atributo A de la relación R se utilizará en la vista creada el nombre R_A a través de la utilización de la cláusula "as" del lenguaje SQL, si éste no es alguno de los atributos involucrados en el resultado final.

Claramente este aspecto cambiaría todos los resultados ya presentados y a presentar, pero por simplicidad no se hará explícito el cambio a menos que sea necesario (dígase hay un conflicto de nombres) para no oscurecer la presentación de los resultados.

Como resultado del paso 1 el algoritmo de generación del esquema asegura que, se obtendrá una relación por fragmento a través de iterar específicamente por todos éstos, en el DW parcial o temporal que se está construyendo. Considerando el enfoque Naive el DW parcial producido luego de la aplicación del paso 1 está conformado por las vistas que se han citado en esta sección.

3.2.2 Paso 2 - Nombres Atributos

El siguiente paso apunta al tratamiento de las correspondencias directas y en particular al tratamiento de los nombres de los atributos. En el paso 2, es necesario generar vistas si existen ítems del fragmento con correspondencia directa que tengan distinto nombre en la relación de la BDF de la que provienen (nombre(ítem) <> nombre(atributo)). Observar que por más que se hubiera aplicado la primitiva Q1 en el paso anterior, ésta no cambia el nombre del atributo para que coincida con el del ítem. Además si más de un atributo requiere el cambio de nombre, éste se realiza en una única vista por fragmento, o sea todos los cambios de nombre se hacen en conjunto. En el paso 2, utilizando posiblemente las vistas creadas en el paso 1, y dado que se utiliza la primitiva Q2, se generarían de ser necesario vistas como la que se muestra a continuación:

create view V^2_i **as select** A_{i1} **as** $f_i(A_{i1})$, .., A_{in} **as** $f_i(A_{in})$ **from** V^1_i

En la sentencia anterior V^1_i hace referencia a que se utiliza la construcción del paso 1, si no se hubieran generado vistas en el paso 1 dado que la aplicación de la primitiva no fuera necesaria, V^1_i sería directamente una relación de la BDF en lugar de una vista del paso anterior. Este comentario vale para cualquiera de los pasos posteriores, en los cuales si no se encontrara una vista del paso anterior, se referiría a la última anterior en la secuencia, o incluso a la relación de la BDF. El siguiente ejemplo demuestra lo expresado anteriormente:

Ejemplo 4: Cambio de nombres de atributos

Tomando el ejemplo anterior de países y ciudades (Ejemplo 2), se puede observar que los atributos de la dimensión son obtenidos mediante correspondencias directas[9] y los nombres de los atributos en las relaciones de la BDF no coinciden con los nombres utilizados en el fragmento. La vista generada en tal caso sería:

```
create view V²ᵢ as select código_país as id_país, nombre_país as país,
     cod_país, cod_ciudad as id_ciudad, nombre_ciudad as ciudad
  from V¹ᵢ
```

Notar que en la consulta anterior, el atributo cod_país no fue tratado con la cláusula as, dado que no tiene correspondencia al fragmento por lo tanto no está en las hipótesis.

♦

3.2.3 Paso 3 - Correspondencias

Este paso está dedicado a tratar las correspondencias que no fueron consideradas anteriormente, es decir todas menos las directas. En el paso 3, el algoritmo de generación del esquema plantea una iteración sobre todas las correspondencias definidas donde, según el tipo de correspondencia se aplica una determinada estrategia. A continuación se ven cada una de las estrategias, observando qué primitiva y qué vista se genera para cada tipo de correspondencia. Notar que, según cuantas correspondencias de cierto tipo se encuentren definidas para un fragmento, tantas veces se aplicará la estrategia asociada.

3.2.3.1 Marca de Tiempo

Suponiendo que la correspondencia del ítem en cuestión es una marca de tiempo (ExternME de tipo **Timestamp**), el algoritmo de generación introducirá la primitiva P3 para agregar al resultado del paso anterior, un atributo nuevo que refleje dicha marca. En tal caso, suponiendo que el nombre del ítem a agregar es A', la vista generada será de la forma:

```
create view V³ᵢ as select Aᵢ₁, .., Aᵢₙ, tᵢ() as A' from V²ᵢ
```

En la vista anterior $t_i()$ es la función con la que se introduce la marca de tiempo y ésta debe expresarse al momento de indicar la correspondencia (por más detalles ver la sección C.2 del Apéndice C en donde se presenta la primitiva asociada). El siguiente ejemplo muestra cómo podría constituirse un caso de aplicación de este paso:

Ejemplo 5: Resolución de marcas de tiempo

Suponiendo que existe la siguiente correspondencia para el fragmento Ventas.

[9] Esto se deduce a través de la notación gráfica utilizada. Ésta es la presentada en [Per01].

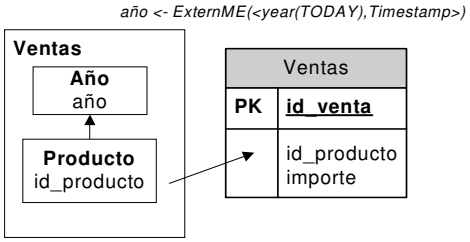


Figura 20: Ejemplo de correspondencia externa de tipo Timestamp.

En tal caso, observando que en los pasos anteriores no se debió generar ninguna vista dado que no se cumplen ninguna de las condiciones para esto, la sentencia generada será:

```
create view V3i as
  select id_venta, id_producto, importe, year(TODAY) as A'
  from Ventas
```

El ejemplo aquí presentado es un clásico caso de manejo de la historia y merecería un análisis mucho más amplio, sin embargo se encuentra fuera del alcance de este trabajo.

♦

3.2.3.2 Dígito de Versión

Si por el contrario alguno de los atributos tuviera una correspondencia a un dígito de versión (ExternME de tipo **Versión**), aparecería la necesidad de aplicar la Primitiva P4.1. En tal caso las vistas que se generarían son:

```
create view V3i as select Ai1, ..., f() || Aij as A', ..., Ain from V2i
```

En la vista anterior f() es la función con la que se versiona el atributo A_{ij} y es parte de las decisiones a tomar al momento de establecer la correspondencia (por más detalles ver la sección C.3 del Apéndice C donde se presenta la primitiva aplicada).

Ejemplo 6: Correspondencias de tipo dígito de versión.

Suponiendo que extendemos el Ejemplo 4 definiendo la siguiente correspondencia:

```
versión → ExternME(<"1", Países, Paises.código_país, Version>)
```

Notar que se está tomando la decisión de cuál será la función f() particular al dar la correspondencia. Tener presente también que la definición de las correspondencias de tipo dígito de versión fue alterada en el apéndice D sección D.4, para indicar la relación y atributo que se está versionando. En base a estas correspondencias la vista será:

```
create view V3i as select V2i.id_país, V2i.país, V2i.cod_país,
  V2i.id_ciudad, V2i.ciudad, "1" || código_país as Versión from V2i
```

Se debe notar que el problema de versionado no es menor y está relacionado al problema de manejo de la historia en la sección anterior. Este problema se encuentra fuera del alcance de esta tesis.

♦

3.2.3.3 Constante

En el caso de que la correspondencia fuera a valores externos constantes (ExternME de tipo **Constant**) la primitiva que el algoritmo aplicaría sería la P7. En tal caso las vistas generadas serían de la forma:

```
create view V3i as select Ai1, ..., Ain, c as A' from V2i
```

Donde c es el valor constante que se establece en el mapeo (en la sección C.6 del Apéndice C es posible ver en detalle la primitiva en cuestión). Un caso particular es que todos los mapeos de una dimensión sean por ejemplo constantes como muestra el ejemplo a continuación:

Ejemplo 7: Resolución de correspondencias constantes

Suponiendo el caso de las siguientes correspondencias:

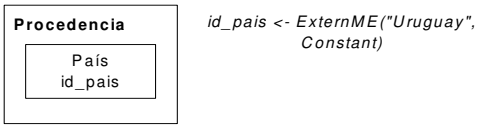


Figura 21: Ejemplo de correspondencias constantes.

En tal caso la sentencia generada sería:

```
create view V3i as select c as A'
```

Es de notar que, este caso podría presentarse también para cualquier tipo de correspondencias, por más que se está presentado en este tipo.

♦

3.2.3.4 Cálculo Simple

Si existiera una correspondencia 1CalcME que indicara un cálculo entre distintos atributos de la relación tomada como entrada, se estaría frente a un caso de la aplicación de la primitiva P6.1. En tal caso las sentencias generadas serían de la forma:

```
create view V3i as select Ai1, ..., Ain, f(Ak1, ..., Akm) as A' from V2i
```

En lo anterior $f(A_{k1}, ..., A_{km})$ representa la función utilizada para el cálculo del atributo, en la sección C.4 del Apéndice C se puede ver en detalle la primitiva.

3.2.3.5 Cálculo de Resumen

Por último, si existiera una correspondencia que indicara que un ítem debe ser obtenido con un cálculo en forma de resumen (NCalcME), sobre atributos de una relación de la BDF, el algoritmo indicaría que la primitiva a aplicar sería la P6.3 y las sentencias generadas serían de la forma:

```
create view V3i as  select V2i.Ai1, ..., V2i.Ain, e(R'i.Bi) as A'
   from V2i, R'i
   where V2i.A = R'i.A
   group by V2i.Ai1, ..., V2i.Ain, X
```

Tomando en cuenta la forma en que la primitiva es invocada, dado que no se agrupa por atributos de la segunda relación, se puede simplificar la sentencia anterior de la siguiente forma:

```
create view V3i as select V2i.Ai1, ..., V2i.Ain, e(R'i.Bi) as A'
   from V2i, R'i
   where V2i.A = R'i.A
   group by V2i.Ai1, ..., V2i.Ain
```

En lo anterior $e(R'_i.B_i)$ representa la función utilizada para el cálculo del atributo, R'_i es la relación sobre la que se está resumiendo y se produce un join para incorporar el atributo calculado a la nueva vista. El aspecto a resaltar acerca del join, es cómo se decide el link a utilizar, que es equivalente a preguntarse cuál es la condición de join. Hay que notar que la definición del NCalcME no contiene el nombre del link, ni tampoco la relación contra la que se realiza el join, sólo está la relación sobre la que se está resumiendo (R'_i). Para llegar a obtener $V^2_i.A = R'_i.A$, se debe considerar que en el esqueleto (V^2_i), alguna de las relaciones debe tener link a R'_i, supongamos R''_i. Como en el armado del esqueleto todas las correspondencias se re-expresaron en función de V^2_i, el link que antes conectaba R''_i con R'_i ahora conecta V^2_i con R'_i. Si fuera el caso que hubieran varios links, o sea que varias relaciones conectaran con R'_i, utilizar cualquiera de ellos debería resultar equivalente. En la sección C.4 del Apéndice C se puede ver en detalle la primitiva. El siguiente ejemplo muestra una posible aplicación de este caso:

Ejemplo 8: Resolución de atributos resumidos

Suponiendo la existencia de las siguientes correspondencias:

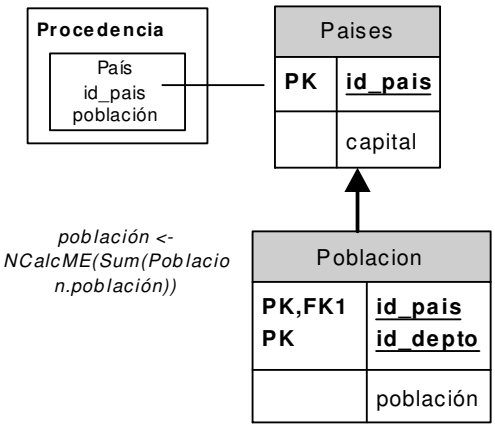


Figura 22: Ejemplo de correspondencia NCalcME.

Para obtener la población, se debe efectuar un cálculo de resumen (NCalcME) sobre la relación que contiene el valor de la población por departamento. Notar que sólo se produce una vista en este caso dado que los pasos anteriores no lo requieren dada la situación planteada. La sentencia generada en este caso será:

```
create view V3i as
      select Paises.id_pais, Paises.capital,
        sum(Poblacion.población) as población
      from Paises, Poblacion
      where Paises.id_pais = Poblacion.id_pais
      group by Paises.id_pais, Paises.capital
```

♦

Al finalizar la aplicación de paso 3, todas las correspondencias que existieran para cada fragmento deben ser resueltas. Si no existiera ninguna, el paso 3 no agregaría ninguna vista a la secuencia.

3.2.4 Paso 4 - Filtros

Continuando con el algoritmo en el paso 4 puede ser necesario aplicar la primitiva Q3. La necesidad radica en que las instancias deseadas para el DW es posible que deban cumplir ciertas restricciones ($f(C_{i1}, ..., C_{ik})$), expresadas como condiciones sobre los datos de la BDF. En tal caso este paso generaría vistas de la forma:

```
create view V4i as select Ai1, ..., Ain from V3i where f(Ci1, ..., Cik)
```

3.2.5 Paso 5 - Eliminar Atributos

A continuación, el paso 5 aplicaría la primitiva P9 si existen atributos en el resultado parcial que no están dentro de las correspondencias definidas, es decir, que no deben estar en el DW. Puede ocurrir que los atributos estén, pero no sean requeridos por ejemplo porque los atributos estaban en la relación fuente, o porque fueron agregados por las primitivas anteriores pero por aspectos puramente operacionales (paso 1, Join, no todos los atributos de la relaciones tienen porque pertenecer al DW). En caso que fuera necesario eliminar atributos el paso 5 generaría construcciones de la forma:

```
create view V5i as select X1, .., Xm, e1(Z1), …, ek(Zk)
      from V4i
      group by X1, .., Xm
```

Dadas las características de la invocación de la primitiva podemos notar que las sentencias generadas en realidad pueden ser simplificadas de la siguiente forma:

```
create view V5i as select Xi1, .., Xim from V4i group by Xi1, .., Xim
```

Esto se debe a que el conjunto de medidas y funciones de resumen que recibe la primitiva en este caso siempre es vacío. El conjunto $\{X_{i1}, .., X_{im}\}$ se conforma de los atributos con correspondencia a ítems del fragmento que pueden ser calculados ya que en cada paso el algoritmo se preocupa por actualizar el conjunto de correspondencias. Un análisis fino de la situación permite notar que, llegado este paso todos los ítems tienen correspondencias sólo de tipo directo, es decir se sustituyen los otros tipos de correspondencia por correspondencias de tipo DirectME. El siguiente ejemplo muestra una posible vista generada por este paso:

Ejemplo 9: Eliminación de atributos

Suponiendo que se continúa con la situación propuesta por el Ejemplo 8. Por un lado id_pais es un atributo con correspondencia directa (notar que además no cambia de nombre en el fragmento de dimensión, por lo cual no generó ninguna vista en el paso 2). Por otro lado si se observa la actualización de los mapeos que realiza la regla AGGREGATE CALCUATE se notará que sustituye la correspondencia de tipo NCalcME por una directa al atributo población recién creado.

En base a esto último se puede decir que el paso 5 generará la siguiente vista:

```
create view V5i as select V3i.id_pais, V3i.población
      from V3i
      group by V3i.id_pais, V3i.población
```

♦

3.2.6 Paso 6 - Claves

Finalmente para cada fragmento cuya clave difiera de la clave en el DW resultado se aplica la primitiva Q4 asociada a este paso. En caso que fuera necesario el algoritmo generaría construcciones de la forma:

```
create view V6i as select A1, ..., An from V5i
```

Hay que notar que esta vista no tiene ningún efecto sobre las instancias, sin embargo el enfoque hasta ahora es incluir una vista por cada aplicación de primitiva. Si se realizara aquí alguna simplificación/excepción/etc. se establecería la pregunta de por qué no hacerlo en otro lugar en que exista la posibilidad (analizando en detalle se puede ver que hay oportunidades de realizar simplificaciones y esto es parte de lo que justifica la propuesta presentada en la sección 4). Además de si hay varios tipos de situaciones donde se pueden hacer simplificaciones, la pregunta sería: ¿cuándo correspondería hacer una simplificación y cuándo no? Si se decidiera hacer todas las simplificaciones ya no se podría considerar este enfoque como Naive o simple; si se optara por un punto medio la discusión se traslada a cuál es ese punto. Tratando de mantener claro el enfoque se ha elegido no realizar ninguna simplificación en el enfoque Naive, aunque ésta sea tan trivial como la decisión de no realizar acción en este paso.

Otro aspecto que es necesario comentar es qué ocurre si los datos no respetan la clave que el diseñador intentaba establecer. La vista propuesta en el paso 6 no tiene ningún efecto sobre las instancias, esto se traduce en que podría ocurrir un error durante la carga. Este aspecto será comentado en la sección 3.5.1 pero se ha de adelantar que éste se considera un error de diseño y se encuentra fuera del alcance de este trabajo.

3.2.7 Resumen de Pasos 1 al 6

Los pasos 1 a 6, desde el aspecto de las vistas que crean se pueden resumir en la Figura 23. En ésta se observa como el paso 1 puede crear vistas de J (de Join), el paso 2 vistas P (de Proyección), el paso 3 vistas G (Agrupamiento) para mapeos NCalcME y vistas P para cualquier otro tipo de mapeos, por su lado el paso 4 crearía vistas S (Selección), el paso 5 vistas G y finalmente el paso 6 vistas P.

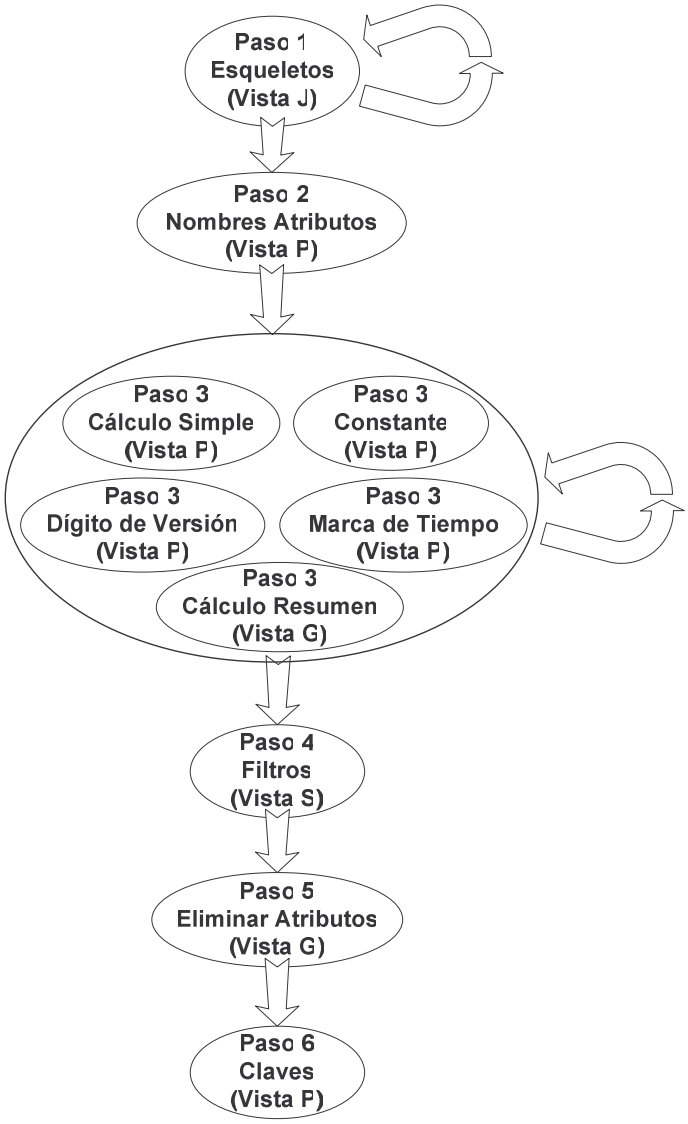


Figura 23: Vistas utilizadas en los pasos 1-6 del algoritmo de generación del esquema.

3.3 Tratamiento de Cubos

En esta sección se centra la atención en los pasos donde se construyen los cubos, o sea desde el paso 7 en adelante. Desde el punto de vista de [Per01] los cubos pueden ser definidos de dos formas, la primera muy similar a la parte de dimensiones, indicando cómo se obtienen a partir de las fuentes mediante el uso de correspondencias. La segunda, es definiendo el cubo en base a otro cubo, también utilizando correspondencias, pero en este caso entre ambos cubos y no con la BDF. Estos últimos

terminan dando origen a los llamados "*Cubos con Mapeo Recursivo*" o como se les llamarán en este trabajo **Cubos Recursivos**. Otro concepto abordado ya en el último paso del algoritmo es el de la **Fragmentación de Cubos**, que refiere a la posibilidad de dividir cubos en múltiples relaciones de forma similar a la Fragmentación de Dimensiones.

En general se ha de decir que a menos de diferencias menores, los pasos 7 al 12, donde se tratan los cubos definidos en función de la BDF, vuelven a ejecutar los pasos 1 al 6 vistos anteriormente, por ende se construye una secuencia similar de vistas. Las diferencias radican en que se toman las correspondencias de cubos, en lugar de las de fragmentos, además de que en el paso 11, donde se eliminan los atributos sin correspondencia (similar al paso 5), se considera la existencia de medidas y por esto cambia la regla aplicada.

A continuación y observando los comentarios anteriores se centrará la atención en analizar los cambios del paso 11, los pasos 13 y 14 que manejan los Cubos Recursivos y el paso 15 que se concentra en la Fragmentación de Cubos.

3.3.1 Paso 11 - Eliminar Atributos en Cubos

En el paso 11 se aplicará la Primitiva P9 si existen atributos en el resultado parcial que no tienen correspondencia (de igual forma que en las dimensiones). En caso que fuera necesario eliminar atributos el paso 11 generaría construcciones de la forma:

```
create view V11_i as select X_1, .., X_m, e_1(Z_1), …, e_k(Z_k)
      from V10_j
      group by X_1, .., X_m
```

En este paso, a diferencia del paso 5, no es posible simplificar la sentencia, ya que Z_1, ..., Z_k efectivamente son proporcionadas, dado que serán las medidas del cubo a construir. A excepción de este aspecto el resultado es idéntico y valen las mismas observaciones sobre cómo se obtiene el conjunto $\{X_1, .., X_m\}$.

3.3.2 Pasos 13 y 14 - Cubos Recursivos

A continuación, si bien se continúa trabajando para generar las relaciones para los cubos, se hará hincapié en los obtenidos recursivamente, es decir definidos en función de otros cubos. De la misma forma que el cubo obtenido recursivamente recibe el nombre de Cubo Recursivo, el cubo del cual se parte recibe el nombre de **Cubo Base**.

El enfoque para construir los cubos recursivos consta de dos etapas. En una primera (paso 13) se construyen fragmentos de dimensión denominados **Fragmentos Ficticios**, dado que sólo se utilizan para construir los cubos recursivos. En la segunda etapa (paso 14) estos fragmentos son utilizados por el algoritmo para efectuar las operaciones de Drill-Up sobre el cubo base y así obtener el cubo recursivo. Estos fragmentos ficticios se construyen sólo para las dimensiones en que sea necesario hacer un **Drill-Up Parcial**, donde esto significa no eliminar la dimensión en el cubo sino ir a un nivel superior. Dichos fragmentos también están especialmente diseñados para contener sólo los niveles origen y el de destino del Drill-Up.

En el paso 13 con el fin de construir los fragmentos ficticios se ejecutan nuevamente los pasos 1 al 6 del algoritmo por lo cual las vistas que se construyen son equivalentes a las ya vistas para estos pasos, por esta razón no se abordará este paso.

A continuación, como se dijo, el paso 14 aplica las operaciones de Drill-Up al cubo base para obtener el cubo recursivo, con la particularidad de que las operaciones de Drill-Up se hacen en una dimensión a la vez. Para realizar cada Drill-Up se utilizan primitivas que se aplicarán tantas veces como sea necesario. Para cada Drill-Up en una dimensión existen dos posibles primitivas a aplicar, la primera se utiliza cuando no hay detalle en el cubo objetivo para la dimensión en cuestión (se elimina la dimensión), en tal caso se aplica la primitiva P9, la cual genera vistas de la forma:

```
create view V14_i as select X_1, .., X_m, e_1(Z_1), …, e_k(Z_k)
       from V12_j
       group by X_1, .., X_m
```

En lo anterior utilizamos V^{12}_j dado que se utilizan como base los cubos generados precisamente al final de este paso. Los atributos $\{X_1, .., X_m\}$ en función de los que se genera el agrupamiento son determinados por la operación de Drill-Up que se está efectuando, hay que notar que se hace el Drill-Up una dimensión a la vez y en esa aplicación $\{X_1, .., X_m\}$ corresponde a los atributos de los niveles de las dimensiones restantes, pues la dimensión en cuestión se está eliminando.

Por otro lado, en caso que hubiera detalle en la dimensión en la cual se desea hacer el Drill-Up (subo a un nivel superior) se aplicaría la primitiva P8, con lo cual se generarían vistas de la forma:

```
create view V14_i as select V_1, …, V_m, e_1(Z_1), …, e_k(Z_k)
     from V12_j, V13_j
     where V12_j.A = V13_j.A
     group by V_1, …, V_m
```

En la vista anterior podemos comentar que $\{V_1, \ldots, V_m\}$ está compuesto por los atributos de los niveles resultantes del Drill-Up. Además notar que se está utilizando V^{13}_j, que es una de las relaciones de los fragmentos ficticios construidos en el paso 13.

Por último hemos de notar que al utilizar V^{12}_j en la expresión para construir V^{14}_i, implícitamente estamos indicando que al observar la traza o secuencia de vistas producida para el cubo recursivo, ésta antepone primero la de la construcción del cubo base. Existen otras alternativas pero por razones de simplicidad y claridad ésta será la utilizada.

Esta decisión de anteponer la traza del cubo base puede ser cuestionable. También podría proponerse otra alternativa en la cual los cubos recursivos se construyeran tomando como fuente la relación del DW que alberga el cubo base. Esto último es, no anteponer las primitivas del cubo base, sino en su lugar, suponer que éste ya está materializado y utilizarlo directamente. Como ya se indicó, esta última sin embargo no es la opción elegida para el enfoque Naive.

3.3.3 Paso 15 - Franjas

Por último el paso 15 se aplicará en caso que exista fragmentación de cubos para generar las distintas **Franjas del Cubo**, es decir las distintas relaciones en las cuales se dividen los datos de cubo. Las franjas son similares al concepto de fragmento en el caso de dimensión y representan una división física de la información a almacenar en el cubo. La diferencia radica en que no se separan atributos sino datos. En general aunque no exista fragmentación de cubos, se puede indicar que existe una sola franja en el cubo,

por lo cual se puede decir que el resultado de los pasos 7 al 15 son las distintas franjas del cubo más que el cubo en si.

La regla aplicada por este paso es la misma que la aplicada en el paso 10, que a su vez ya fue presentada en la sección 3.2.4 al analizar el paso 4, por lo cual las vistas generadas serán las mismas y se omitirá escribirlas nuevamente.

3.4 Resumen

A modo de resumen se presenta la Figura 24 que esquematiza las primitivas y vistas aplicadas a lo largo del algoritmo.

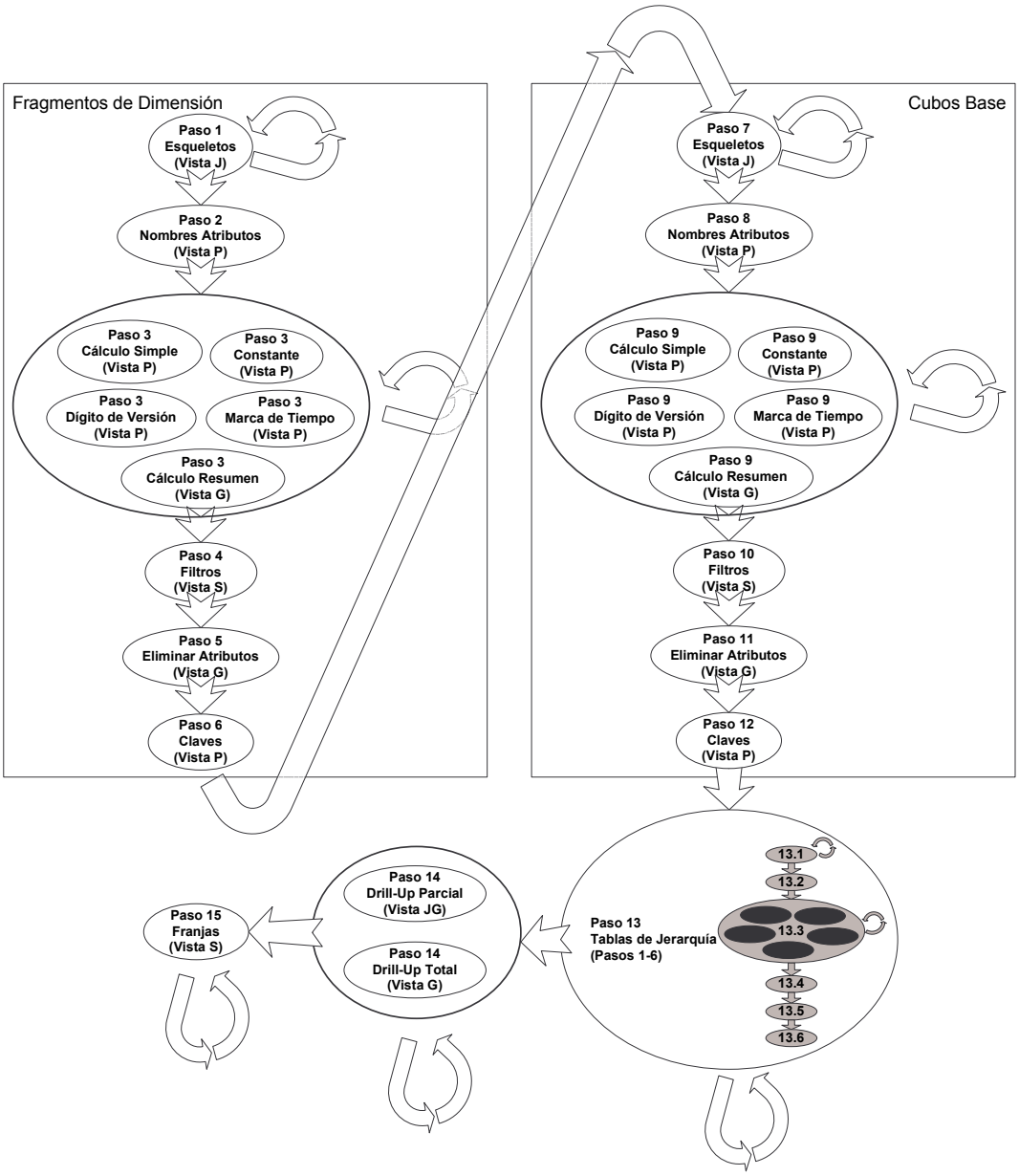


Figura 24: Vistas utilizadas en el algoritmo de generación del esquema.

A continuación la Tabla 4 muestra qué construcciones se generan como consecuencia del uso del enfoque Naive:

Paso	Regla	Primitiva	Vista
1	R1 – Join	Q1 – Join	**create view** V^1_i **as select** $Att(R_{i1}) \cup Att(R_{i2})$ **from** R_{i1}, R_{i2} **where** $f(C_{i1}, .., C_{ik})$
2	R2 – Rename	Q2 – Attribute Renaming	**create view** V^2_i **as select** A_{i1} **as** $f_i(A_{i1})$, .., A_{in} **as** $f_i(A_{in})$ **from** V^1_i
3	R3.1 – Simple Calculate	P6.1 – DD-Adding 1-1	**create view** V^3_i **as select** A_{i1}, ..., A_{in}, $f(A_{k1}, ..., A_{km})$ **as** A' **from** V^2_i
	R3.2 – Aggregate Calculate	P6.3 – DD-Adding N-N	**create view** V^3_i **as select** $V^2_i.A_{i1}$, ..., $V^2_i.A_{in}$, $e(R'_i.B_i)$ **as** A' **from** V^2_i, R'_i **where** $V^2_i.A = R'_i.A$ **group by** $V^2_i.A_{i1}$, ..., $V^2_i.A_{in}$, X
	R4.1 – Constant Extern Value	P7 – Attribute Adding	**create view** V^3_i **as select** A_{i1}, ..., A_{in}, c **as** A' **from** V^2_i
	R4.2 – Timestamp Extern Value	P3 – Temporization	**create view** V^3_i **as select** A_{i1}, .., A_{in}, $t_i()$ **as** A' **from** V^2_i
	R4.3 – Version Extern Value	P4.1 – Version Digits	**create view** V^3_i **as select** A_{i1}, ..., $f()$ **||** A_{ij} **as** A', ..., A_{in} **from** V^2_i
4	R8 – Filter	Q3 – Instance Filter	**create view** V^4_i **as select** A_{i1}, ..., A_{in} **from** V^3_i **where** $f(C_{i1}, ..., C_{ik})$
5	R5.1 – Fragment Group	P9 – Aggregate Generation	**create view** V^5_i **as select** X_{i1}, .., X_{im} **from** V^4_i **group by** X_{i1}, .., X_{im}
6	R7 – Primary Key	Q4 – Primary Key	**create view** V^6_i **as select** A_1, ..., A_n **from** V^5_i
7	Ídem paso 1	Ídem paso 1	Ídem paso 1
8	Ídem paso 2	Ídem paso 2	Ídem paso 2
9	Ídem paso 3	Ídem paso 3	Ídem paso 3
10	Ídem paso 4	Ídem paso 4	Ídem paso 4
11	R5.2 – Cube Group	P9 – Aggregate Generation	**create view** V^{11}_i **as select** X_1, .., X_m, $e_1(Z_1)$, …, $e_k(Z_k)$ **from** V^{10}_j **group by** X_1, .., X_m
12	Ídem paso 6	Ídem paso 6	Ídem paso 6
13	Ídem pasos 1 al 6	Ídem pasos 1 al 6	Ídem pasos 1 al 6
14	R6.1 – Hierarchy Drill-Up	P8 – Hierarchy Roll-Up	**create view** V^{14}_i **as select** V_1, …, V_m, $e_1(Z_1)$, …, $e_k(Z_k)$ **from** V^{12}_j, V^{13}_j **where** $V^{12}_j.A = V^{13}_j.A$ **group by** V_1, …, V_m
	R6.2 – Total Drill-Up	P9 – Aggregate Generation	**create view** V^{14}_i **as select** X_1, .., X_m, $e_1(Z_1)$, …, $e_k(Z_k)$ **from** V^{12}_j **group by** X_1, .., X_m
15	Ídem paso 4	Ídem paso 4	Ídem paso 4

Tabla 4: Construcciones generadas por el enfoque Naive.

De una rápida revisión surge que a lo largo del algoritmo de generación de esquema se utilizan las siguientes primitivas: P3, P4.1, P6.1, P6.3, P7, P8, P9, Q1, Q2, Q3, Q4, sin embargo tomando como base las primitivas presentadas en [Mar00] las siguientes no son utilizadas: P1, P2, P4.2, P5, P6.2, P10, P11, P12, P13 y P14. El hecho de que no todas las primitivas sean utilizadas se debe a dos aspectos. Por un lado existen primitivas para problemas que no están incluidos dentro de las capacidades del algoritmo de generación de esquema (ej.: P10 Data Array Creation ver [Mar00] por más detalles). Por otro lado dado que ciertas operaciones pueden realizarse de varias formas, la opción tomada en el algoritmo de generación de esquema puede dejar fuera ciertas primitivas a favor de utilizar otras (ej.: P2 Data Filter elimina atributos de las relaciones pero eventualmente podría dejar instancias repetidas, en ese sentido el algoritmo toma como alternativa P9 Aggregate Generation para filtrar los atributos y no generar repetidos).

Por último es necesario citar a los efectos de contemplar todas las posibilidades el caso de la **Traza Vacía**. Una traza vacía puede ser producida si ninguno de los pasos del algoritmo necesita agregar ninguna construcción al resultado, es decir si ninguno es aplicable. El caso particular en el cual se da, es que la relación en la BDF sea exactamente la que necesitamos en el DW. Suponiendo que una relación en la BDF contiene exactamente el fragmento de dimensión o tabla de hechos del DW, en tal caso no se genera ninguna construcción a lo largo del algoritmo.

Para el tratamiento de las trazas vacías se generará una vista que simplemente hará corresponder el resultado de la fuente tal y como está. La vista generada para el caso 'i' donde la relación R_i es exactamente la que se corresponde con un fragmento de dimensión o tabla de hechos será:

```
create view Vi as select Att(Ri) from Ri
```

V_i es la relación a obtener en el DW, es decir la relación a la que la traza vacía apunta a obtener. Dado que no se genera traza para este caso su resolución termina en este punto.

Todo lo presentado hasta el momento permite conocer exactamente qué se produce al aplicar el enfoque Naive, es decir, que vistas, en que orden, con que información, etc. Analicemos ahora si existen problemas que no se hayan considerado para luego elaborar las conclusiones.

3.5 Problemas

El análisis anterior trató de enfocarse en cómo llevar adelante la carga utilizando el enfoque Naive sin desviarse con los problemas que pudiesen aparecer. En esta sección, por el contrario, se abordarán los problemas que antes se omitieron, pero estrictamente considerando los originados por la utilización del enfoque Naive. Esto refiere a que la carga puede fallar si los datos utilizados son incorrectos (ej.: se indicó mal la clave primaria de una relación), sin embargo, esto se considera error de diseño y no de la carga, por lo cual no está dentro del alcance de esta sección.

La estrategia utilizada para desarrollar esta sección tiene que ver con la posibilidad de generar resultados equivocados a causa de utilizar el enfoque Naive para realizar la carga. Como se verá puede ocurrir que las operaciones asociadas a las primitivas y en

particular ciertas combinaciones de éstas, produzcan resultados erróneos desde el punto de vista de las instancias que se hubiera esperado obtener.

3.5.1 Violación de clave Primaria

Aunque un caso esté correctamente expresado existe la posibilidad de que la forma en que se organizan las operaciones asociadas a las primitivas, induzca un error que produce una violación de la clave primaria, como se ve en el siguiente ejemplo:

Ejemplo 10: Violación de clave primaria en el enfoque Naive

Sea una dimensión Países compuesta por un nivel País con atributos país y temperatura, la clave del nivel es el atributo país. Sean dos relaciones en la base datos: Ciudad_País (ciudad, país) y Temperaturas (fecha, ciudad, temperatura) donde existe un link entre ellas en el atributo ciudad. Se suponen dos mapeos:

1) Mapeo directo entre país de la dimensión y país de Ciudad_País.
2) NCalcME: temperatura de la dimensión es avg(Temperaturas.temperatura).

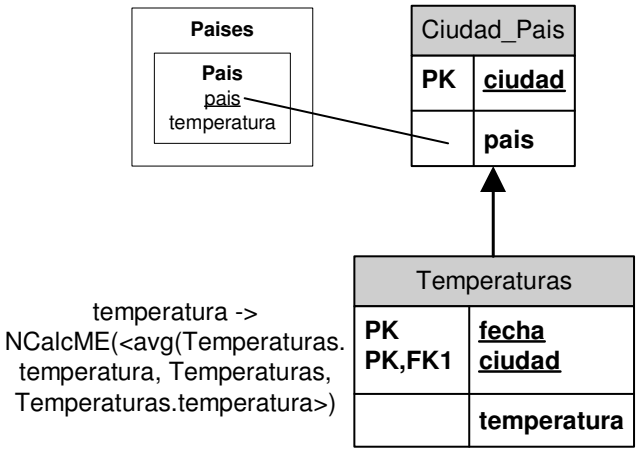


Figura 25: Correspondencias utilizadas en el Ejemplo 10.

Al calcular las vistas que se producirán en cada paso tenemos:
Paso 1: No es necesario aplicar ninguna primitiva dado que sólo se considera la correspondencia de Ciudad_País.país para construir el esqueleto.
Paso 3: El NCalcME produce la siguiente vista:

```
create view V3i as
   select a.ciudad, a.pais, avg(b.temperatura) as temperatura
   from Ciudad_Pais a, Temperaturas b
   where a.ciudad = b.ciudad
   group by a.ciudad, a.pais
```

Paso 5: Al eliminar los atributos se obtiene la siguiente vista:

```
create view V5i as
   select país, temperatura
   from V3i
   group by país, temperatura
```

El resultado sin embargo no es el esperado. Se obtienen promedios por ciudad y no por país como sugiere V^5_i. Podrían aparecer tuplas en V^5_i como (Uruguay, 1.5) y (Uruguay, 2.5) con los siguientes datos:

Ciudad_País	
Ciudad	País
Canelones	Uruguay
Montevideo	Uruguay

Temperaturas		
Fecha	Ciudad	Temperatura
1/1/2005	Canelones	1.5
1/1/2005	Montevideo	2.5

♦

El problema con el ejemplo anterior es que ni el paso 5 ni el 6 están asegurando la unicidad de la clave del fragmento de dimensión sobre las instancias. El paso 5 se concentra en eliminar los atributos sobrantes, pero potencialmente como se ha visto deja repetidos con respecto a la clave del fragmento, y el paso 6 no tiene efecto sobre las instancias. Este problema podría haber sido solucionado si el agrupamiento se hubiera planteado en primera instancia en función de País que es la clave del fragmento, o de otras formas, pero no sin afectar el enfoque Naive.

A la luz de lo anterior surge la pregunta de, cuál es la posibilidad de que existan otros problemas relacionados con la clave primaria en el enfoque Naive, debido a que el paso 6 no realiza ninguna acción explícita al respecto y tampoco existe un análisis que permita afirmar que la clave planteada para el fragmento es válida para la transformación de instancias a realizar. La respuesta a esta pregunta es que no es posible afirmar que no existen otros problemas de este tipo; en este trabajo sólo se pudo identificar el antes planteado. La principal dificultad para identificar este tipo de anomalías recae en que no debe tratarse de un problema de diseño sino de ejecución de la carga.

3.5.2 Asociatividad de las Operaciones

Otro problema existente con el enfoque Naive tiene que ver con la asociatividad de las operaciones. Como se vio anteriormente en el paso 14 se resuelven los cubos con mapeo recursivo utilizando la estrategia de hacer Roll-Up's sobre los cubos base. La combinación de este paso con otros previamente ejecutados puede llevar a errores cuando las operaciones de agregación de las medidas no son asociativas.

En [GR00] se estudian las secuencias de operaciones así como los operadores de agregación([GCBL97]) desde el punto de vista de la materialización de vistas anidadas con agrupamientos[10]. En este trabajo sólo basta con referir al más simple de los problemas mencionados en [GR00] (asociatividad), puesto que al detectarlo se detectan

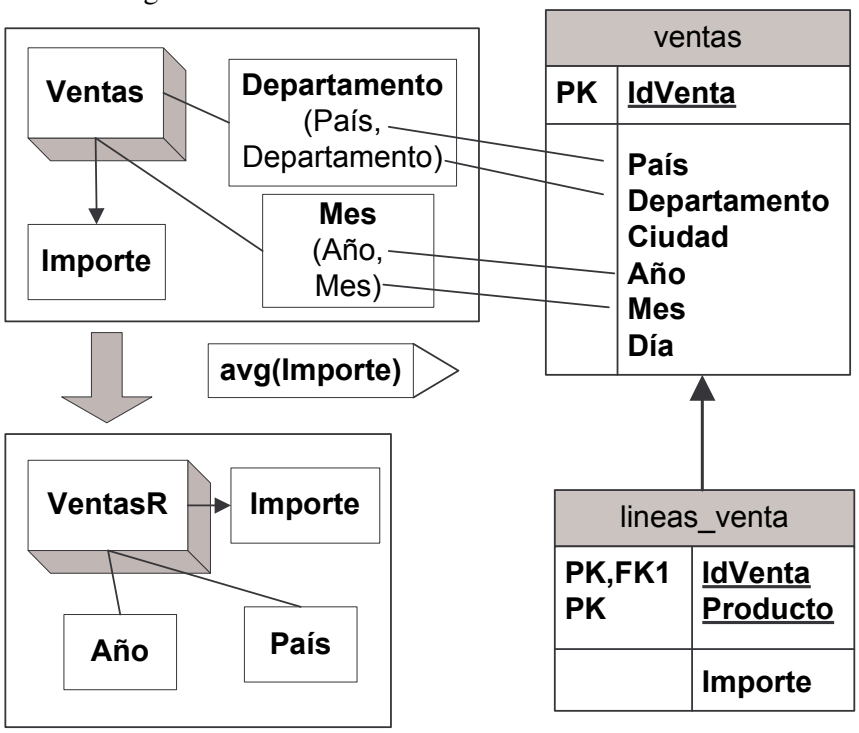
[10] En [GR00] y [GCBL97] la propiedad de asociatividad aquí mencionada se refiere como propiedad distributiva. En este trabajo se considera mejor opción utilizar el término asociatividad y no distributividad, tomando en cuenta que ésta condice con la definición de la propiedad en el álgebra clásica (http://en.wikipedia.org/wiki/Associative, http://en.wikipedia.org/wiki/Distributive).

también los otros por la naturaleza de éstos (una función de agregación no asociativa tampoco es algebraica u hologistica).

En el siguiente ejemplo se puede observar el problema que puede aparecer:

Ejemplo 11: Error de asociatividad en el enfoque Naive

Suponiendo un cubo Ventas y un cubo VentasR construido recursivamente sobre Ventas como muestra la Figura 26:



Importe -> NCalcME(<avg(lineas_venta.Importe), lineas_venta, lineas_venta.Importe>)

Figura 26: Ejemplo de cubo recursivo.

En la Figura 26 se puede apreciar que el cubo Ventas se construye básicamente tomando información de la relación ventas de la BDF, para la medida Importe es necesario además resolver un NCalcME resumiendo las N posibles líneas asociadas a cada venta. Conformando el ejemplo también se encuentran dos dimensiones (presentadas en la Figura 27), por un lado la dimensión Ubicaciones que sirve de soporte para País y Departamento en los cubos anteriores, por otro lado la dimensión Fechas que da soporte a Año y Mes.

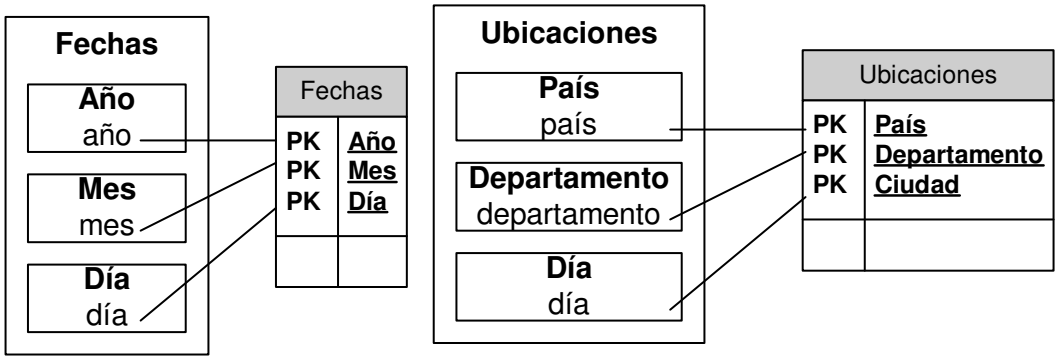


Figura 27: Dimensiones asociadas al ejemplo de Cubo Recursivo.

Al ejecutar el algoritmo, primero debemos atender los fragmentos de dimensión (pasos 1 al 6), sin embargo para la especificación anterior no se construye ninguna vista dado que las relaciones en la BDF son exactamente las que resultarán en el DW (ver en la sección 3.4, Traza Vacía), por lo cual se pasa directamente al procesamiento de cubos. Al continuar con el algoritmo abordando ahora el cubo Ventas (pasos 7 al 12), para éste se obtiene el primer resultado en el paso 9, pues los anteriores no generan ninguna vista, el resultado obtenido en dicho paso será:

```
create view V9i as
  select ventas.IdVenta, ventas.País, ventas.Departamento,
    ventas.Ciudad, ventas.Año, ventas.Mes, ventas.Día,
    avg(lineas_ventas.Importe) as Importe
  from ventas, lineas_venta
  where lineas_venta.IdVenta = ventas.IdVenta
  group by ventas.IdVenta, ventas.País, ventas.Departamento,
    ventas.Ciudad, ventas.Año, ventas.Mes, ventas.Día
```

En la vista generada en el paso 9 se está resolviendo la correspondencia de tipo NCalcME que se plantea para obtener la medida Importe. Posteriormente en el paso 11 se obtendrá:

```
create view V11i as
  select V9i.Mes, V9i.Año, V9i.Departamento, V9i.País,
    avg(V9i.Importe) as Importe
  from V9i
  group by V9i.Mes, V9i.Año, V9i.Departamento, V9i.País
```

La vista anterior está al mismo tiempo eliminando los atributos sobrantes y agregando la medida como corresponde. En el paso 12 se genera también una vista, pero dado que este no tiene impacto alguno sobre las instancias se omite su presentación. Esto es todo lo que se realiza para el cubo Ventas, posteriormente en la ejecución del algoritmo le toca el turno al cubo VentasR y en el paso 13 se genera lo siguiente:

```
create view V13.5i1 as
  select Fechas.Año, Fechas.Mes
  from Fechas
  group by Fechas.Año, Fechas.Mes

create view V13.5i2 as
  select Ubicaciónes.País, Ubicaciónes.Departamento
  from Ubicaciónes
  group by Ubicaciónes.País, Ubicaciónes.Departamento
```

Estas vistas corresponden a la construcción de las dos jerarquías ficticias, éstas si bien no son estrictamente necesarias para hacer el Roll-Up por las dimensiones Fechas y Ubicaciones, son construidas con este fin. Cabe notar que si la dimensión se hubiera encontrado fragmentada y además las claves de los niveles no fueran débiles, sí hubiera sido imprescindible contar con estas vistas para poder hacer el Roll-Up. Al ejecutar el paso 6 dentro del paso 13 se vuelven a generar 2 vistas que no tienen efecto sobre las instancias, por lo tanto se omiten para facilitar la presentación, a continuación en el paso 14 se generan las siguientes vistas:

```
create view V14i1 as
   select V13.5i1.Año, V11i.Departamento, V11i.País,
      avg(V11i.Importe) as Importe
   from V11i, V13.5i1
   where V11i.Mes = V13.5i1.Mes and V11i.Año = V13.5i1.Año
   group by V13.5i1.Año, V11i.Departamento, V11i.País

create view V14i2 as
   select V14i1.Año, V13.5i2.País, avg(V14i1.Importe) as Importe
   from V14i1, V13.5i2
   where V14i1.País = V13.5i2.País and
      V14i1.Departamento = V13.5i2.Departamento
   group by V13.5i1.Año, V11i.País
```

La vista V^{14}_{i2} es la obtenida como resultado del cubo recursivo planteado al comienzo de este ejemplo. Esta vista (al igual que V^{14}_{i1} y la V^{11}_{i}) presenta un problema clave respecto al uso de la función promedio, pues esta función no es asociativa y por lo tanto al evaluar la vista no se obtendrá un resultado correcto. Para que este resultado fuese correcto el promedio debería haberse realizado una única vez y no al construir el cubo base y luego al hacer el Roll-Up de cada dimensión.

♦

Este ejemplo se muestra al mismo tiempo tres aspectos asociados a que alguna de las operaciones de agregación de las medidas de un cubo recursivo no sea asociativa:

1. Es cuestionable separar el cálculo de la correspondencia (paso 9) y la eliminación de atributos (paso 11).
2. No es posible separar en pasos la operación de eliminación de atributos (paso 11) de las operaciones de Roll-Up (paso 14).
3. No es posible dividir en pasos la operación de Roll-Up, es decir hacer Roll-Up por cada dimensión.

Se debe aclarar que, la primera de estas observaciones está marcada como cuestionable, porque podría ser el caso que el resultado intermedio igual fuera válido tomando en cuenta el problema de asociatividad del operador. En el caso de las dos últimas observaciones no hay ninguna razón para dividir el cálculo en N etapas por lo cual este sí es un resultado erróneo inducido por la forma de resolución.

Otra observación que cabe realizar es relativa a la construcción de los fragmentos ficticios. Las principales razones para construir un fragmento de dimensión auxiliar son dos. Por un lado la dimensión original puede encontrarse fragmentada de una forma que no resulte útil para hacer el Drill-Up (nivel origen y destino en relaciones separadas). Por otro lado la primitiva utilizada requiere que el nivel origen del Drill-Up pertenezca a la clave de la relación que sirve de jerarquía (el fragmento de dimensión) lo cual

tampoco tiene porque ser así. Estas restricciones que dejan muy poco margen para reutilizar alguno de los fragmentos de dimensión existentes, pero tomando otro enfoque tal vez no sea necesario construir los fragmentos ahorrando los costos de este paso.

3.5.3 Múltiple Conteo

A no ser por las diferencias en el paso 11 como se vio en la sección 3.3.1, no existe esencial diferencia entre las vistas producidas por los pasos 1-6 y 7-12, sin embargo, debemos notar que, en este paso, existe un problema potencial de múltiple conteo (o problema de sumarizabilidad [LS97]), como muestra el siguiente ejemplo.

Ejemplo 12: Múltiple conteo en el enfoque Naive

Sean dos relaciones: Población (País, Departamento, Ciudad, Población) y Territorio (País, Departamento, Territorio), suponiendo que los links entre las relaciones que determinan los joins son Población.País = Territorio.País y Población.Departamento = Territorio.Departamento y considerando ahora las siguientes instancias de datos:

Población			
País	Departamento	Ciudad	Población
Uruguay	Canelones	Canelones	1
Uruguay	Canelones	San Ramón	2

Territorio		
País	Departamento	Territorio
Uruguay	Canelones	2

Supóngase que se plantea un cubo donde las dimensiones del mismo son País y Departamento y las medidas Población y Territorio, estando las funciones de promedio y suma asociadas correspondientemente a las operaciones de Roll-Up.

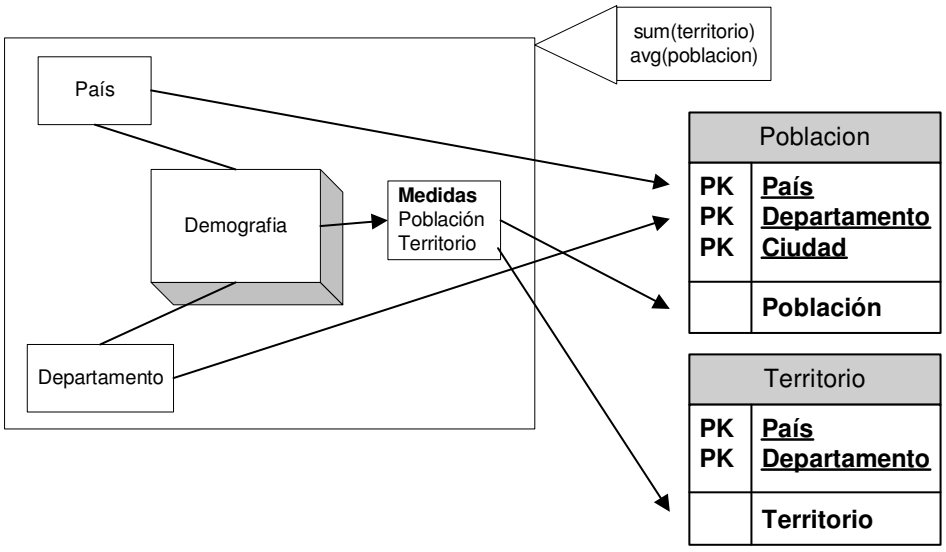


Figura 28: Correspondencias para el Cubo del Ejemplo 12.

Si la información proviene de las relaciones antes mencionadas se tendrá un problema de doble conteo al llegar al paso 11. Primero en el paso 7 se generaría la siguiente vista:

```
create view V7i as
   select Población.País, Población.Departamento,
     Población.Ciudad as Población_Ciudad, Población.Población,
     Territorio.País as Territorio_País,
     Territorio.Departamento as Territorio_Departamento,
     Territorio.Territorio
   from Población, Territorio
   where Población.País = Territorio.País and
     Población.Departamento = Territorio.Departamento
```

En lo anterior recordar la estrategia mencionada anteriormente para las colisiones en los nombres de atributo. A continuación en el paso 11 se generaría la siguiente vista que resultaría de esta forma:

```
create view V11i as
   select V7i.País, V7i.Departamento, avg(V7i.Población),
     sum(V7i.Territorio)
   from V7i
   group by V7i.País, V7i.Departamento
```

Si se resuelve la vista anterior, ésta contendrá la tupla [País = Uruguay, Departamento = Canelones, Población = ½, Territorio = 4] que no es correcta dado que el territorio de Canelones es 2.

♦

El ejemplo anterior muestra un problema de múltiple conteo producido por hacer join de relaciones de diferente nivel de granularidad para además agrupar los resultados. Este ejemplo podría haber sido enfocado desde otro punto de vista para evitar el problema de múltiple conteo, para esto la opción sería obtener la Población como un mapeo NCalcME. De todas formas hay que notar que esto incurriría en el problema de asociatividad presentado en la sección 3.5.2 dado que el promedio no es una operación asociativa, por lo tanto el problema tampoco sería resuelto.

3.6 Conclusiones

A lo largo de este capítulo se ha analizado cómo se comporta el enfoque Naive para realizar la carga del DW creado tras la aplicación del algoritmo de [Per01]. Se lo ha denominado de esta forma pues se basa en aprovechar la traza de primitivas originada por el algoritmo de generación de esquema, dejando que las transformaciones de las instancias asociadas a cada primitiva y la propia organización de las primitivas, guíe el proceso de carga.

Este enfoque demostró ser viable para un espectro de situaciones, pues a lo sumo con pequeñas observaciones se pudieron abordar casi todas las situaciones. Sin embargo, también evidenció problemas en ciertos casos donde, ofrecer una solución, exigiría un cambio sustancial al enfoque y no una simple observación. En la sección 3.5.1 se mostró como se podía producir una violación de la clave primaria al ocurrir que algunas de las tuplas obtenidas luego de la transformación, no cumplían con la clave propuesta para la relación del DW. Lo más trascendente de este problema es el hecho de que no se trata de un error de diseño, sino de un error producido por la ejecución de la carga con el enfoque Naive. También en la sección 3.5.2 se comentó el problema acarreado por la

no asociatividad de algunas de las operaciones de agregación, así como la sección 3.5.3 demostró que podían existir situaciones en las cuales se produce múltiple conteo.

La afirmación de que el enfoque Naive es incorrecto que se puede realizar en base a lo anterior plantea una disyuntiva. Al analizar [Per01] se puede notar que las instancias correctas o esperadas en términos de lo que el usuario deseaba definir al momento de establecer sus correspondencias, están definidas por la secuencia de primitivas y sus operaciones sobre las instancias. Ahora bien, ésta es la base del enfoque Naive, por lo que nos lleva a la conclusión de que si el enfoque Naive es incorrecto, también lo es la secuencia de primitivas generadas, lo cual plantea una contradicción.

La observación que es necesario realizar al razonamiento anterior es la siguiente: el algoritmo de generación del esquema presentado en [Per01] se concentra principalmente en la generación del esquema del DW e intenta inducir cómo se realizará la carga de éste. Las observaciones realizadas en la sección 3.5, así como otras sobre la eficiencia del enfoque Naive que se presentarán posteriormente, quedan fuera de lo que puede contemplar un trabajo que no trate específicamente el aspecto de la carga. Por ende, sin perjuicio del lo realizado por [Per01] este trabajo aprovecha este espacio para realizar un aporte. El siguiente capítulo muestra cómo esto se lleva a cabo.

4 Propuesta de Carga

En el algoritmo de generación del esquema, cada relación del DW se obtiene de una secuencia de aplicaciones de primitivas. Continuando esta línea, el enfoque Naive propone realizar la carga a través de la secuencia de vistas asociada a la secuencia de primitivas. Esta primera alternativa es suficiente en muchos casos, pero no en todos como se notó en la sección 3.5. Además como se mostrará en este capítulo no es la mejor solución que se puede proponer.

Al analizar el algoritmo se puede notar que éste propone por lo menos seis pasos para el procesamiento de cada relación del DW. En general cada uno produce una vista a raíz de la utilización de una primitiva. Dependiendo de la complejidad del procesamiento (ej.: joins, correspondencias, etc.) la cantidad de vistas utilizadas para una relación puede aumentar aún más. Para el DW lo importante es el resultado final que es representado por la última vista de la secuencia. Tener tantas vistas complica la legibilidad al mismo tiempo que puede llegar a consumir más recursos. Una mejor alternativa es construir una única vista o **vista unificada** para cada relación; esto mejora su presentación pues toda la transformación puede verse expresada en dicha vista. Al mismo tiempo como veremos más adelante, las consideraciones que se realizan para crear una única vista no están al alcance de un optimizador, por lo cual ésta tendrá mejor desempeño.

Este capítulo presenta la propuesta de carga impulsada por este trabajo como alternativa al enfoque Naive, argumentando que ésta resuelve los problemas del enfoque Naive, mejora la presentación en términos de legibilidad e introduce mejoras de desempeño en la solución obtenida. Esta propuesta tiene como objetivo obtener la vista unificada para cada fragmento de dimensión o franja de cubo, que sólo esté definida en función de las relaciones de la BDF, tomando en consideración que éstas serán las vistas con las cuales se definirán los datos a almacenar en el DW.

Para llegar al objetivo de una única vista se mostrará que, al considerar la secuencia de vistas del enfoque Naive, es posible proponer en cada caso una vista que resuma toda la cadena. De esta forma no sólo se logrará obtener la vista sino que también se demostrará su equivalencia, a excepción de las correcciones con el enfoque Naive. Por ejemplo, suponiendo que en el paso 1 se produce una vista V^1 que está definida en función de las relaciones de la BDF y en el paso 2 una V^2 que utiliza V^1, se presentará una vista alternativa $V^{2'}$ que sea equivalente a V^2 pero que no requiera de V^1, o sea que esté definida en función de relaciones de la BDF. Sucesivas repeticiones de esta estrategia finalmente producirán una única vista que reproduce la secuencia de vistas producidas por todos los pasos del algoritmo. Finalmente luego de tener este resultado, se mostrará cómo los problemas identificados en el enfoque Naive fueron solucionados y porqué se argumenta que la solución propuesta es más óptima que la anterior.

Para elaborar esta propuesta se cambiará de representación y se utilizará la expresión en álgebra relacional (AR)[GUW00] de cada una de las vistas, ya que las demostraciones de equivalencia resultan más claras si se realizan utilizando propiedades

del AR que haciendo lo mismo con vistas. Para probar que dos vistas son equivalentes se debería probar que todas las tuplas de una, están en la otra y viceversa. Sin embargo utilizando expresiones del álgebra la equivalencia se obtiene por las propiedades de la transformación aplicada.

4.1 Tratamiento de Dimensiones

Tomando como base lo realizado con el enfoque Naive, en esta sección se analizan cada una de las vistas propuestas para el procesamiento de fragmentos de dimensiones. El objetivo es construir una sola vista para cada uno de éstos, que resuma todas las transformaciones que éste sufre hasta formar parte del DW.

4.1.1 Paso 1 - Esqueletos

Para el paso 1 donde se construyen los esqueletos se puede realizar una primera observación dado que en realidad este paso no tiene porqué producir una única vista, sino que podría ser en potencia un conjunto de éstas (ver en la sección 3.2.1). Dicha secuencia se expresa en AR de la siguiente forma[11]:

$V^1_i \leftarrow R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2}$
$V^1_i, \leftarrow V^1_i \otimes_{cond(V1i, Ri3)} R_{i3}$
...

Sobre la formulación anterior que presenta el join entre R_{i1}, R_{i2} y en un paso posterior R_{i3} se puede realizar la siguiente deducción:

$V^1_i, \leftarrow V^1_i \otimes_{cond(V1i, Ri3)} R_{i3} \Rightarrow (1)$
$V^1_i, \leftarrow R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes_{cond(V1i, Ri3)} R_{i3} \Rightarrow (2)$
$V^1_i, \leftarrow R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes_{cond(Ri2, Ri3)} R_{i3}$
...

1) Sustitución.
2) $cond(V^1_i, R_{i3}) = cond(R_{i2}, R_{i3})$ ya que las condiciones (links) originalmente eran entre estas relaciones en la BDF, se convirtieron a condiciones entre V^1_i y R_{i3} al crear la vista por el ajuste de las correspondencias (ver Regla Join en el apéndice B, sección B.1).

De esta forma se está construyendo una expresión equivalente que incluye todas las operaciones de join. Continuando esta línea de deducciones se puede ver que cualquiera sea la cantidad de relaciones que se deba procesar en el paso 1, siempre se puede escribir una sola vista que será de la forma:

```
create view V1in as select Ri1.*, Ri2.*, ...
   from Ri1, Ri2, ...
   where cond(Ri1, Ri2) and cond(Ri2, Ri3) and ..
```

[11] Originalmente $cond(R_{i1}, R_{i2})$ se expresaba con $f(C_1, ..., C_k)$ sin embargo para su mejor presentación en este capítulo se adoptó esta representación.

Ejemplo 13: Ejemplo de vista producida al procesar sólo el armado de esqueletos.

Tomando como base la situación del Ejemplo 3 se obtendría la vista:

```
create view V¹2 as
   select idfamilia, descfamilia, idsubfamilia, descsubfamilia,
     idproducto, descproducto
   from familias, subfamilias, productos
   where familias.idfamilia = subfamilias.idfamilia and
        subfamilias.idsubfamilia = productos.idsubfamilia
```

♦

4.1.2 Paso 2 - Nombres Atributos

Si en el paso 2 se pretende tener una sola vista como resultado final, se debe generar una vista que recoja tanto el resultado anterior como el de la propia transformación del paso 2. La primera opción es que el algoritmo no haya generado una vista para el paso 1, pero en tal caso la vista del paso 2 directamente sería la vista final, por lo cual no sería necesario hacer ninguna deducción. El caso más interesante es suponer que existe una vista generada para el paso 1. En este caso se sustituirá en la expresión asociada a la primitiva utilizada en el paso 2 la expresión encontrada para el paso 1. Luego proporcionando una vista que represente la expresión obtenida finalmente se habrá logrado el objetivo.

$V^2_i \leftarrow \prod_{Ai1 \rightarrow f(Ai1),\ ..,\ Ain \rightarrow f(Ain)} V^1_i \Rightarrow (1)$
$V^2_i \leftarrow \prod_{Ai1 \rightarrow f(Ai1),\ ..,\ Ain \rightarrow f(Ain)} (R_{i1} \otimes_{cond(Ri1,\ Ri2)} R_{i2} \otimes \ ...)$

1) Sustitución.

La vista asociada a la expresión anterior es:

```
create view V²i as select Ai1 as f(Ai1), ..., Ain as f(Ain)
   from Ri1, Ri2, ...
   where cond(Ri1, Ri2) and cond(Ri2, Ri3) and ..
```

Para manejar los nombres de atributos en la proyección se utilizan operadores del AR extendidos presentados en [GUW00][12]. Con esta notación al escribir $A_{i1} \rightarrow f(A_{i1})$ dentro de la proyección se indica que el atributo A_{i1} pasa a llamarse $f(A_{i1})$. Se debe notar que el uso del AR permite concluir rápidamente que las expresiones son equivalentes y por tanto que la vista que se tenía para el paso 2 (que internamente utilizaba a la del paso 1) es equivalente a la nueva vista propuesta.

Ejemplo 14: Vista resultado de unir el paso 1 y el paso 2

Tomando como base ahora el Ejemplo 4 y el Ejemplo 2:

[12] En las expresiones aquí presentadas se recuerda que $f(A_{i1})$ .. $f(A_{in})$ son utilizadas a los solos efectos de indicar que se asigna un nombre a cada uno de los atributos.

```
create view V²i as select Países.código_país as id_país,
     Países.nombre_país as país, Ciudades.cod_país,
     Ciudades.cod_ciudad as id_ciudad, Ciudades.nombre_ciudad as ciudad
   from Países, Ciudades
   where Ciudades.cod_país = Países.código_país
```

Se puede observar que los potenciales conflictos que se habían mencionado al final de la sección 3.2.1 en los nombres de atributos ya no están presentes, dado que ahora se puede referir a los atributos utilizando el nombre de la relación de la que se obtienen (ej.: Países.código_país en lugar de sólo código_país) pues éstas siempre van a estar disponibles.

♦

Para el análisis de los pasos siguientes es conveniente realizar una observación. En todos los análisis posteriores se deberá considerar una vista que resuma los resultados obtenidos hasta el paso inmediatamente anterior. Como ya se ha observado al analizar este paso, se plantean en general varias posibilidades respondiendo al hecho de que no en todos los pasos se tiene porqué aplicar una primitiva. Por otro lado es posible observar que en general siempre existe un caso que es más representativo o que de alguna forma recoge las complejidades y situaciones que implican los pasos anteriores. Atendiendo al hecho de que no resultaría práctico enumerar todas las posibilidades, dado que en general se resuelven con los mismos argumentos y por ende no aportan valor a la presentación, se optará por tomar siempre en consideración este caso representativo al momento de considerar el resultado del paso anterior. El caso más representativo del paso 2 es el que supone la aplicación de por lo menos alguna primitiva en el paso 1. En general el caso más representativo será el único abordado, siendo los otros comentados brevemente en caso de ser necesario.

4.1.3 Paso 3 - Correspondencias

En el paso 3 hay un conjunto de casos distintos según el tipo de correspondencia encontrada (Marca de Tiempo, 1CalcME, NCalcME, etc.). Primeramente se analizará por separado qué ocurre con cada correspondencia, suponiendo que sólo existe una de éstas y luego se verá qué ocurre cuando hay más de una. La estrategia abordada será la misma que para el paso 2, por sustitución de un paso en el otro se tratará de encontrar una expresión que permita construir una única vista.

4.1.3.1 Marca de Tiempo

Para comenzar se analizará el caso de la Primitiva P3 (marca de tiempo):

$V^3_i \leftarrow \prod_{Ai1, .., Ain, ti()\rightarrow A'} V^2_i \Rightarrow (1)$

$V^3_i \leftarrow \prod_{f(Ai1), .., f(Ain), ti()\rightarrow A'} (\prod_{Ai1\rightarrow f(Ai1), .., Ain\rightarrow f(Ain)} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes ...)) \Rightarrow (2)$

$V^3_i \leftarrow \prod_{Ai1\rightarrow f(Ai1), .., Ain\rightarrow f(Ain), ti()\rightarrow A'} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes ...)$

1) Sustitución. Notar que como ya se mencionó al final de la sección 3.2.2 se toma un caso representativo de la expresión asociada al paso 2 a fin de no considerar todas las posibilidades, que como se dijo, no aportarían valor a la presentación. Advertir también que se instancia A_{i1}, ..., A_{in}, con los nombres de los atributos de la relación sobre la cual se aplica la proyección $f(A_{i1})$, ..., $f(A_{in})$.

2) Por la propiedad de Cascada del operador $\prod$ se obtiene la última proyección[EN97], replanteada en función de los atributos correspondientes ($A_{i1} \rightarrow f(A_{i1})$, .., $A_{in} \rightarrow f(A_{in})$).

La vista será de la forma:

```
create view V3i as select Ai1 as f(Ai1), ..., Ain as f(Ain), ti() as A'
   from Ri1, Ri2, ...
   where cond(Ri1, Ri2) and cond(Ri2, Ri3) and ..
```

La vista formulada recoge los pasos 1, 2 y 3 pero sólo para el caso de marcas de tiempo. Es de notar que para lo anterior es necesario extender el operador de proyección definido en [GUW00], ya que no se están restringiendo a las expresiones que éste introduce para el operador. Esto se hace en el Apéndice D, sección D.1.

Ejemplo 15: Vista resultado de unir el paso 1, 2 y el 3 para marcas de tiempo

Tomando como base el Ejemplo 4, el Ejemplo 2 y agregando a éstos la existencia de una correspondencia que introduzca un atributo Fecha definido como marca de tiempo (ver Ejemplo 5) se obtiene:

```
create view V3i as select Países.código_país as id_país,
     Países.nombre_país as país, Ciudades.cod_país,
     Ciudades.cod_ciudad as id_ciudad,
     Ciudades.nombre_ciudad as ciudad, year(TODAY) as Fecha
   from Países, Ciudades
   where Ciudades.cod_país = Países.código_país
```

♦

4.1.3.2 Dígito de Versión

El caso de la Primitiva P4.1 (dígitos de versión) tiene una resolución muy similar ya que se utilizan las mismas justificaciones en cada paso:

$V^3_i \leftarrow \prod_{Ai1, .., f()\|Aij \rightarrow A', .., Ain} V^2_i \Rightarrow$ (1)
$V^3_i \leftarrow \prod_{f(Ai1), .., f()\|f(Aij) \rightarrow A', .., f(Ain)} (\prod_{Ai1 \rightarrow f(Ai1), .., Ain \rightarrow f(Ain)} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes ...)) \Rightarrow$ (2)
$V^3_i \leftarrow \prod_{Ai1 \rightarrow f(Ai1), .., f()\|f(Aij) \rightarrow A', .., Ain \rightarrow f(Ain)} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes ...)$

4.1.3.3 Atributo Constante

También la Primitiva P7 (agregado de atributo) tiene una resolución muy similar:

$V^3_i \leftarrow \prod_{Ai1, .., Ain, c \rightarrow A'} V^2_i \Rightarrow$ (1)
$V^3_i \leftarrow \prod_{f(Ai1), .., f(Ain), c \rightarrow A'} (\prod_{Ai1 \rightarrow f(Ai1), .., Ain \rightarrow f(Ain)} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes ...)) \Rightarrow$ (2)
$V^3_i \leftarrow \prod_{Ai1 \rightarrow f(Ai1), .., Ain \rightarrow f(Ain), c \rightarrow A'} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes ...)$

4.1.3.4 1CalcME (cálculo simple)

La Primitiva P6.1 (1CalcME) sigue la misma línea:

$V^3_i \leftarrow \prod_{Ai1, .., Ain, f(Ak1,..,Akm)\rightarrow A'} V^2_i \Rightarrow (1)$
$V^3_i \leftarrow \prod_{f(Ai1), .., f(Ain), f(Ak1,..,Akm)\rightarrow A'} (\prod_{Ai1\rightarrow f(Ai1), .., Ain\rightarrow f(Ain)} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes ...)) \Rightarrow (2)$
$V^3_i \leftarrow \prod_{Ai1\rightarrow f(Ai1), .., Ain\rightarrow f(Ain), f(Ak1,..,Akm)\rightarrow A'} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes ...)$

4.1.3.5 NCalcME (cálculo resumen)

A continuación se analizará la primitiva P6.3. Para comenzar es necesario realizar una observación dado que a diferencia de los otros casos no se tomará la representación de la instancia propuesta para la primitiva, pues ésta no conducirá a un resultado.

Recordando que el objetivo es obtener una única vista como resultado de todas las operaciones de la traza, analicemos el caso en el que existen dos correspondencias de tipo NCalcME en forma consecutiva en la traza. Si bien aún no se habían introducido este tipo de complejidades al análisis, es claro que estas situaciones pueden ocurrir. Suponiendo que se ha llegado hasta este punto tomando la instancia generada para un mapeo NCalcME de la propuesta original de la primitiva, si se realizara el planteo, se llegaría a una situación como la siguiente (γ representa el operador de agregación según **[GUW00]**):

$V^3_i \leftarrow \gamma_{Ai1\rightarrow f(Ai1), .., Ain\rightarrow f(Ain), e(R'i.Bi)\rightarrow A'} (V^{3'}_i \otimes_{V3'i.A=R'i.A} R'_i) \Rightarrow (1)$
$V^3_i \leftarrow \gamma_{Ai1\rightarrow f(Ai1), .., Ain\rightarrow f(Ain), e(R'i.Bi)\rightarrow A'} ((\gamma_{Ai1\rightarrow f(Ai1), .., Ain\rightarrow f(Ain), e'(R''i.B'i)\rightarrow A'} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes ... \otimes_{V2i.A=R''i.A} R''_i)) \otimes_{V3'i.A=R'i.A} R'_i)$

1) Sustitución de un caso NCalcME a su vez en otro caso NCalcME.

El problema con dicha expresión es que no es posible fusionar los dos agrupamientos, aspecto que es imprescindible si deseamos obtener una sola vista como resultado. Si se analiza el camino de una transformación de la forma:

$$\gamma\ (\gamma\ (R \otimes R') \otimes R'') = \gamma\ (R \otimes R' \otimes R'')$$

se encuentra que es difícil realizar algún tipo de simplificación. El Ejemplo 16 muestra las dificultades encontradas con un planteo de esta forma.

Ejemplo 16: Dificultades al fusionar los NCalcME con la representación original

Se suponen tres relaciones Países, Población y Territorio, de tal forma que el objetivo es calcular la Población y el Territorio total mediante correspondencias de tipo NCalcME. Como se desea realizar todo en una vista, se intentarán juntar las dos correspondencias NCalcME en una sola vista mediante la siguiente suposición:

$\gamma_{País, Población, sum(Territorio)} (\gamma_{País, sum(Población)} (Países \otimes Población) \otimes Territorio) = \gamma_{País, sum(Población), sum(Territorio)} (Países \otimes Población \otimes Territorio)$

En lo anterior la izquierda de la igualdad representa lo que plantearía el algoritmo y la derecha intenta ser una simplificación donde no hay dos agrupamientos sino que se colocó todo en uno. Suponiendo que los links entre las relaciones que determinan los join's entre las relaciones son Países.País = Población.País y Países.País = Territorio.País y considerando las siguientes instancias de datos:

Países
País
Uruguay

Población		
País	Departamento	Población
Uruguay	Montevideo	1
Uruguay	Canelones	2

Territorio		
País	Departamento	Territorio
Uruguay	Montevideo	1
Uruguay	Canelones	2

Se puede constatar que si se resuelve $\gamma_{\text{País, Población, sum(Territorio)}}$ ($\gamma_{\text{País, sum(Población)}}$ (Países $\otimes$ Población) $\otimes$ Territorio) entonces el resultado obtenido será [Población=3, Territorio=3], mientras que si se resuelve $\gamma_{\text{País, sum(Población), sum(Territorio)}}$ (Países $\otimes$ Población $\otimes$ Territorio) el resultado será [Población=6, Territorio=6]. La razón es simple: en la expresión de la derecha el join produce más tuplas que las esperadas. Este problema se denomina Múltiple Conteo[13] y permite observar que no es posible hacer la transformación propuesta, aunque claro existe una solución y ésta se analiza a continuación.

♦

El ejemplo anterior muestra que es necesario explorar otro camino. Con el fin de obtener una solución al problema se introduce la posibilidad de representar las correspondencias de tipo NCalcME de la siguiente forma:

$$\Pi_{Att(R),e}(R \otimes_{cond(R,R')} \gamma_{CR',E\rightarrow e}(R'))$$

en lugar de como se planteó originalmente en la definición de la primitiva:

$$\gamma_{Att(R),E\rightarrow e}\ (R \otimes_{cond(R,R')} R')$$

En lo último, *Att*(R) es el conjunto de todos los atributos de R, E es una expresión de agregación que sólo involucra atributos de R' y $C_{R'}$ son los atributos de R' que aparecen en cond(R, R'). En este cambio de representación se está aplicando un caso particular del concepto de Eager Aggregation[YL94]. La nueva representación puede ser escrita en una única sentencia SQL de la forma:

[13] A diferencia de la sección 4.4 donde el problema de múltiple conteo era inherente al enfoque Naive, en esta ocasión el problema se produce por la estrategia de simplificación. En el enfoque Naive esta situación no ocurre dado que la simplificación no se aplica.

```
select R.*, e
from R,
   (select C_R', E(R'.B) as e
    from R'
    group by C_R') S
where cond(R, S)
```

Como se puede ver en el apéndice D en la sección D.3 es posible demostrar que estas expresiones son equivalentes bajo la hipótesis de que R no tiene repetidos, o sea que la expresión de la vista que se ha generado hasta antes del paso 3 no debe tener repetidos. Para justificar esto se asumirá que las relaciones de la BDF no pueden tener repetidos (lo cual significa suponer que al menos todas tienen clave que es una restricción muy liviana). Además se puede constatar que las operaciones utilizadas en los pasos anteriores (join y proyección pero siempre agregando atributos) tampoco pueden introducir repetidos.

La ventaja de contar con este tipo de construcción es que a diferencia de la otra versión, sí es posible encadenar correspondencias de tipo NCalcME y expresar el resultado en una sola vista. Esto se verá posteriormente cuando se analice la posibilidad de tener varias correspondencias a fin de no alterar el orden de presentación. A continuación se verá el caso en que luego del paso 2 se deba aplicar un mapeo de tipo NCalcME. El planteo con la nueva propuesta de esta situación será el siguiente:

$V^3_i \leftarrow \prod_{Att(V2i),A'} (V^2_i \otimes_{V2i.C=S'i.C} (\gamma_{C,\ e(R'i.Bi)\rightarrow A',\ R'_i})) \Rightarrow (1)$

$V^3_i \leftarrow \prod_{Ai1\rightarrow f(Ai1),\ ..,\ Ain\rightarrow f(Ain),A'} ((\prod_{Ai1\rightarrow f(Ai1),\ ..,\ Ain\rightarrow f(Ain)} (R_{i1} \otimes_{cond(Ri1,\ Ri2)} R_{i2} \otimes ...)) \otimes_{V2i.C=S'i.C} (\gamma_{C,\ e(R'i.Bi)\rightarrow A',\ R'_i})) \Rightarrow (2)$

$V^3_i \leftarrow \prod_{Ai1\rightarrow f(Ai1),\ ..,\ Ain\rightarrow f(Ain),A'} ((\prod_{Ai1\rightarrow f(Ai1),\ ..,\ Ain\rightarrow f(Ain)} (R_{i1} \otimes_{cond(Ri1,\ Ri2)} R_{i2} \otimes ...)) \otimes_{V2i.C=S'i.C} (\prod_{S'i.C,\ A'} (\gamma_{C,\ e(R'i.Bi)\rightarrow A',\ R'_i}))) \Rightarrow (3)$

$V^3_i \leftarrow \prod_{Ai1\rightarrow f(Ai1),\ ..,\ Ain\rightarrow f(Ain),A'} (\prod_{Ai1\rightarrow f(Ai1),\ ..,\ Ain\rightarrow f(Ain),\ S'i.C,\ A'} (R_{i1} \otimes_{cond(Ri1,\ Ri2)} R_{i2} \otimes ... \otimes_{V2i.C=S'i.C} (\gamma_{C,\ e(R'i.Bi)\rightarrow A',\ R'_i}))) \Rightarrow (4)$

$V^3_i \leftarrow \prod_{Ai1\rightarrow f(Ai1),\ ..,\ Ain\rightarrow f(Ain),\ A'} (R_{i1} \otimes_{cond(Ri1,\ Ri2)} R_{i2} \otimes ... \otimes_{V2i.C=S'i.C} (\gamma_{C,\ e(R'i.Bi)\rightarrow A',\ R'_i}))$

1) Sustitución de vista del paso 2. Notar que S'_i hace referencia al alias que se le da a la sub-consulta de agrupamiento.
2) Se aplica $\prod_{R.*}(R) = R$, o sea la identidad del operador $\prod$.
3) Se aplica la conmutatividad entre el operador $\prod$ y $\otimes$ [EN97]:

   "Conmutación de $\prod$ con $\otimes$: Supongamos que la lista de proyección es $L = A_1, .., A_n, B_1, .., B_m\}$, donde $A_1, .., A_n$ son atributos de R y $B_1, .., B_m$ son atributos de S. Si en la condición de reunión c intervienen sólo los atributos comprendidos en L, las dos operaciones pueden conmutarse como sigue:
   $\prod_L (R \otimes_C S) \equiv (\prod_{A1, .., An}(R)) \otimes_C (\prod_{B1, .., Bm}(S))$"

   En este caso se puede aplicar ya que $V^2_i.C \in \{f(A_{i1}), .., f(A_{in})\} = \{A_1, .., A_n\}$ (tiene que ser alguno de los atributos) y $S'_i.C$ se proyecta explícitamente en $\{S'_i.C, A'\} = \{B_1, .., B_m\}$, $L = \{f(A_{i1}), .., f(A_{in}), S'_i.C, A'\}$.

4) Por la propiedad de Cascada del operador $\prod$ se obtiene la última proyección.

En la derivación no se ha reemplazado $V^2_i.C$ ni $S'_i.C$ por una expresión en términos de las relaciones de la BDF pero esto puede hacerse fácilmente observando que atributos se utilizan dado que las relaciones de la BDF están presentes.

Aquí terminaría el análisis del paso 3 a no ser por el detalle de que las primitivas pueden ser aplicadas repetidas veces y en cualquier orden en el paso 3. Se analizarán entonces los siguientes casos demostrativos de las combinaciones que pueden suceder. No se citan expresamente todas dado que esto no aporta valor a razón de que son muy similares a las que aquí se esbozan.

4.1.3.6 NCalcME y nuevamente NCalcME

La deducción para cuando es necesario encadenar dos aplicaciones de NCalcME es:

$V^3_i \leftarrow \prod_{Att(V3'i),A'} (V^{3'}_i \otimes_{V3'i.C=S'i.C} (\gamma_{C, e(R'i.Bi)\rightarrow A'} R'_i)) \Rightarrow (1)$

$V^3_i \leftarrow \prod_{Ai1\rightarrow f(Ai1), .., Ain\rightarrow f(Ain), A'', A'} ((\prod_{Ai1\rightarrow f(Ai1), .., Ain\rightarrow f(Ain), A''} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes ... \otimes_{V2i.C'=S''i.C'} (\gamma_{C', e(R''i.Bi)\rightarrow A''} R''_i))) \otimes_{V3'i.C=S'i.C} (\gamma_{C, e(R'i.Bi)\rightarrow A'} R'_i)) \Rightarrow (2)$

$V^3_i \leftarrow \prod_{Ai1\rightarrow f(Ai1), .., Ain\rightarrow f(Ain), A'', A'} ((\prod_{Ai1\rightarrow f(Ai1), .., Ain\rightarrow f(Ain), A''} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes ... \otimes_{V2i.C'=S''i.C'} (\gamma_{C', e(R''i.Bi)\rightarrow A''} R''_i))) \otimes_{V3'i.C=S'i.C} (\prod_{S'i.C, A'} (\gamma_{C, e(R'i.Bi)\rightarrow A'} R'_i))) \Rightarrow (3)$

$V^3_i \leftarrow \prod_{Ai1\rightarrow f(Ai1), .., Ain\rightarrow f(Ain), A'', A'} (\prod_{Ai1\rightarrow f(Ai1), .., Ain\rightarrow f(Ain), A'', S'i.C, A'} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes ... \otimes_{V2i.C'=S''i.C'} (\gamma_{C', e(R''i.Bi)\rightarrow A''} R''_i) \otimes_{V3'i.C=S'i.C} (\gamma_{C, e(R'i.Bi)\rightarrow A'} R'_i))) \Rightarrow (4)$

$V^3_i \leftarrow \prod_{Ai1\rightarrow f(Ai1), .., Ain\rightarrow f(Ain), A'', A'} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes ... \otimes_{V2i.C'=S''i.C'} (\gamma_{C', e(R''i.Bi)\rightarrow A''} R''_i) \otimes_{V3'i.C=S'i.C} (\gamma_{C, e(R'i.Bi)\rightarrow A'} R'_i))$

1) Sustitución de NCalcME.
2) Se aplica $\prod_{R.*}(R) = R$, o sea la identidad del operador $\prod$.
3) Se aplica la conmutatividad entre el operador $\prod$ y $\otimes$. En este caso se puede aplicar ya que $V^{3'}_i.C \in \{f(A_{i1}), .., f(A_{in}), A''\} = \{A_1, .., A_n\}$ (tiene que ser alguno de los atributos) y $S'_i.C$ se proyecta explícitamente en $\{S'_i.C, A'\} = \{B_1, .., B_m\}$, $L = \{f(A_{i1}), .., f(A_{in}), A'', S'_i.C, A'\}$.
4) Por la propiedad de Cascada del operador $\prod$ nos se obtiene la última proyección.

En definitiva con una sentencia de la siguiente forma se puede representar cualquier secuencia hasta el paso 3 del algoritmo donde sólo se hayan utilizado correspondencias de tipo NCalcME:

```
create view V3'i as select Ai1 as f(Ai1), ..., Ain as f(Ain)
  from Ri1, Ri2, ...
    (select C', e(R''i.Bi) as A''
      from R''
      group by C') S''i,
    (select C, e(R'i.Bi) as A'
      from R'
      group by C) S'i, ...
  where cond(Ri1, Ri2) and cond(Ri2, Ri3) and ..
    V2i.C'=S''i.C' and V3'i.C=S'i.C and ...
```

Ejemplo 17: Solución al problema de fusión de correspondencias NCalcME.

Retomando el Ejemplo 16 la vista resultado de aplicar lo visto en esta sección sería:

```
create view V3i as
   select País, Población'.Población, Territorio'.Territorio
   from Países,
      (select Población.País, sum(Población.Población) as Población
        from Población
        group by Población.País) Población',
      (select Territorio.País, sum(Territorio.Territorio) as Territorio
        from Territorio
        group by Territorio.País) Territorio'
   where Países.País=Población'.País and Países.País=Territorio'.País
```

♦

4.1.3.7 Marca de tiempo y NCalcME

$V^3_i \leftarrow \prod_{Ai1, .., Ain, ti()\rightarrow A'} V^{3'}_i \Rightarrow (1)$

$V^3_i \leftarrow \prod_{f(Ai1), .., f(Ain), A'', ti()\rightarrow A'} (\prod_{Ai1\rightarrow f(Ai1), .., Ain\rightarrow f(Ain), A''} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes ... \otimes_{V2i.C=S'i.C} (\gamma_{C, e(R'i.Bi)\rightarrow A''} R'_i))) \Rightarrow (2)$

$V^3_i \leftarrow \prod_{Ai1\rightarrow f(Ai1), .., Ain\rightarrow f(Ain), A'', ti()\rightarrow A'} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes ... \otimes_{V2i.C=S'i.C} (\gamma_{C, e(R'i.Bi)\rightarrow A'} R'_i))$

1) Sustitución de un caso NCalcME.
2) Considerando que A_{i1}, …, A_{in} serán todos los atributos de $V^{3'}_i$ y aplicando la propiedad de Cascada del operador $\prod$ se obtiene la última proyección[EN97].

Si en lugar de marca de tiempo la correspondencia hubiera sido dígito de versión, constante o 1CalcME, el análisis y su resolución es extremadamente similar, por esto no se presenta.

4.1.3.8 NCalcME y Dígito de versión

$V^3_i \leftarrow \prod_{Att(V3'i), A'} (V^{3'}_i \otimes_{V3'i.C=S'i.C} (\gamma_{C, e(R'i.Bi)\rightarrow A'} R'_i)) \Rightarrow (1)$

$V^3_i \leftarrow \prod_{Ai1, .., A'', .., Ain, A'} ((\prod_{Ai1, .., f()\|Aij\rightarrow A'', .., Ain} V^2_i) \otimes_{V3'i.C=S'i.C} (\gamma_{C, e(R'i.Bi)\rightarrow A'} R'_i)) \Rightarrow (2)$

$V^3_i \leftarrow \prod_{Ai1, .., A'', .., Ain, A'} ((\prod_{Ai1, .., f()\|Aij\rightarrow A'', .., Ain} V^2_i) \otimes_{V3'i.C=S'i.C} \prod_{S'i.C, A'} (\gamma_{C, e(R'i.Bi)\rightarrow A'} R'_i)) \Rightarrow (3)$

$V^3_i \leftarrow \prod_{Ai1, .., A'', .., Ain, A'} (\prod_{Ai1, .., f()\|Aij\rightarrow A'', .., Ain, S'i.C, A'} (V^2_i \otimes_{V3'i.C=S'i.C} (\gamma_{C, e(R'i.Bi)\rightarrow A'} R'_i))) \Rightarrow (4)$

$V^3_i \leftarrow \prod_{Ai1, .., f()\|Aij\rightarrow A'', .., Ain, A'} (V^2_i \otimes_{V3'i.C=S'i.C} (\gamma_{C, e(R'i.Bi)\rightarrow A'} R'_i))$

1) Sustitución de un caso de dígito de versión
2) Se aplica $\prod_{R.*}(R) = R$, o sea la identidad del operador $\prod$.
3) Se aplica la conmutatividad entre el operador $\prod$ y $\otimes$ [EN97]. Aplicarla porque $V^{3'}_i.C \in \{f(A_{i1}), ..,A'', .., f(A_{in})\} = \{A_1, .., A_n\}$ (tiene que ser alguno de los atributos) y $S'_i.C$ se proyecta explícitamente en $\{S'_i.C, A'\} = \{B_1, .., B_m\}$, $L = \{f(A_{i1}), .., A'', .., f(A_{in}), S'_i.C, A'\}$.
4) Se obtiene la última proyección por la propiedad de Cascada del operador $\prod$.

Nuevamente no se plantea el caso de marca de tiempo, constante o 1CalcME dado que la resolución es extremadamente similar.

4.1.3.9 Marca de tiempo y Dígito de Versión

$V^3_i \leftarrow \prod_{Ai1, .., Ain, ti()\rightarrow A'} V^3'_i \Rightarrow (1)$
$V^3_i \leftarrow \prod_{Ai1, .., Ain, ti()\rightarrow A'} (\prod_{Ai1, .., f()\|Aij\rightarrow A'', .., Ain} V^2_i) \Rightarrow (1)$
$V^3_i \leftarrow \prod_{Ai1, .., f()\|Aij\rightarrow A'', .., Ain, ti()\rightarrow A'} V^3'_i$

1) Sustitución de un caso de dígito de versión.
2) Por la propiedad de Cascada del operador $\prod$ se obtiene la última proyección.

Los casos que restan analizar (marca de tiempo con constante, marca de tiempo con 1CalcME, marca de tiempo con marca de tiempo, etc.) tienen la particularidad de ser muy similares a este último, dado que todos son proyecciones de proyecciones, por esta razón no se los analizarán explícitamente.

4.1.3.10 Resumen del Paso 3

En la sección 4.1.3 se comienza analizando todos los tipos de correspondencias, pues cada uno de éstos recibe un tratamiento distinto (secciones 4.1.3.1 a 4.1.3.5). A continuación se observa el caso en que se defina más de una correspondencia, y por tanto que sea necesario encadenar los resultados. Como resultado de este análisis se presenta la siguiente expresión del álgebra relacional y sentencia SQL, a modo de ilustración de las expresiones o vistas que se pueden obtener al aplicar la estrategia hasta el paso 3:

$V^3_i \leftarrow \prod_{Ai1\rightarrow f(Ai1),\ ..,\ f()\|Aij\rightarrow A''',\ ..,\ Ain\rightarrow f(Ain),\ A',\ ti()\rightarrow A'',\ c\rightarrow Aiv,\ f(\ldots)\rightarrow Av} (R_{i1} \otimes_{cond(Ri1,\ Ri2)} R_{i2} \otimes \ldots \otimes_{V2i.C=S'i.C} (\gamma_{C,\ e(R'i.Bi)\rightarrow A'} R'_i) \ldots)$

```
create view V3i as
      select Ai1 as f(Ai1), ..., f() || Aij as A''', ...,
             Ain as f(Ain), A', ti() as A'', c as AIV,
             f(..) as AV
      from Ri1, Ri2, ...
            (select C, e(R'i.Bi) as A'
      from R'
      group by C) S'i, ...
      where cond(Ri1, Ri2) and cond(Ri2, Ri3) and ..
     V3'i.C=S'i.C and ...
```

Estas últimas provienen del análisis de los resultados presentados y la observación de qué aporta cada tipo de correspondencia al resultado generado. Como se discutió previamente, dependiendo de qué primitivas se apliquen, la expresión o vista puede llegar a no ser tan compleja como la antes enunciada. La siguiente tabla resume lo que aporta cada correspondencia a la vista final del paso 3.

Correspondencia	Aporte o Impacto
NCalcME	Agrega un atributo a la cláusula SELECT (A') que se obtiene de una sub-consulta en la cláusula FROM ($S'_i.C$). Además se agregan las condiciones de join con la sub-consulta en la cláusula WHERE (ver secciones 4.1.3.5, 4.1.3.6 o 4.1.3.8).
1CalcME	Agrega un atributo a la cláusula SELECT (A^V) que es donde se encuentra la expresión de cálculo del mapeo (f(..)).
ExternME ∈ Timestamp	Cada correspondencia de tipo ExternME / Timestamp aporta un atributo a la cláusula SELECT (A'') que es donde se encuentra la función que calcula la marca de tiempo del mapeo ($t_i()$) (ver secciones 4.1.3.1, 4.1.3.7 o 4.1.3.9).
ExternME ∈ Version Digits	Agrega un atributo a la cláusula SELECT (A''') que es donde se encuentra la función que agrega el dígito de versión de la correspondencia (f() || A_{ij}) (ver secciones 4.1.3.2, 4.1.3.8 o 4.1.3.9).
ExternME ∈ Constant Value	Agrega un atributo a la cláusula SELECT (A^{IV}) que es donde se encuentra la constante del mapeo (c) (ver sección 4.1.3.3).

Tabla 5: Aporte o Impacto de cada tipo de correspondencia en la sentencia del Paso 3.

Ejemplo 18: Ejemplo de vista para el paso 3 con múltiples tipos de correspondencia.

Agregando al planteo del Ejemplo 17 las siguientes correspondencias:

```
Fecha → ExternME(<TODAY, Timestamp>)
Versión → ExternME(<País, Version>)
Usuario → ExternME(<"Ignacio", Constant>)
Código → 1CalcME(<IF País == "Uruguay" THEN "UY" ELSE "Desconocido",
  {}, {Países.País}>)
```

Tenemos la siguiente vista resultado al finalizar el paso 3:

```
create view V3i as
  select País, Población'.Población, Territorio'.Territorio,
    TODAY as Fecha, "1_" || País as Versión, "Ignacio" as Usuario,
    IF País == "Uruguay" THEN "UY" ELSE "Desconocido" as Código
  from Países,
    (select Población.País, sum(Población.Población) as Población
      from Población
      group by Población.País) Población',
    (select Territorio.País, sum(Territorio.Territorio) as Territorio
      from Territorio
      group by Territorio.País) Territorio'
  where Países.País=Población'.País and Países.País=Territorio'.País
```

La vista anterior claramente puede cambiar según el orden en el cual se procesen las correspondencias pero esto no afecta el resultado.

♦

4.1.4 Paso 4 - Filtros

En el paso 4 se fusiona la operación de selección con la expresión obtenida del paso anterior, de la siguiente forma:

$V^4_i \leftarrow \sigma_{f(Ci1, .., Cik)} V^3_i \Rightarrow (1)$

$V^4_i \leftarrow \sigma_{f(Ci1, .., Cik)} (\prod_{Ai1 \rightarrow f(Ai1), .., f() \| Aij \rightarrow A''', .., Ain \rightarrow f(Ain), A', ti() \rightarrow A'', c \rightarrow Aiv, f(\ldots) \rightarrow Av} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes \ldots \otimes_{V2i.C=S'i.C} (\gamma_{C, e(R'i.Bi) \rightarrow A'} R'_i) \ldots)) \Rightarrow (2)$

$V^4_i \leftarrow \prod_{Ai1 \rightarrow f(Ai1), .., f() \| Aij \rightarrow A''', .., Ain \rightarrow f(Ain), A', ti() \rightarrow A'', c \rightarrow Aiv, f(\ldots) \rightarrow Av} (\sigma_{f(Ci1, .., Cik)} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes \ldots \otimes_{V2i.C=S'i.C} (\gamma_{C, e(R'i.Bi) \rightarrow A'} R'_i) \ldots))$

1) Sustitución por la expresión propuesta para el paso 3.
2) Se aplica la conmutatividad entre el operador $\prod$ y σ [EN97].

> "*Conmutación de σ con $\prod$: Si en la condición de selección c intervienen sólo los atributos A_1, …, A_n de la lista de proyección, se pueden conmutar ambas operaciones:*
> $\prod_{A1, \ldots, An} (\sigma_C (R)) \equiv \sigma_C (\prod_{A1, \ldots, An} (R))$"
>
> En este caso se puede aplicar ya que C_{i1}, .., C_{ik} (los elementos utilizados en la selección) son atributos de alguna de las relaciones fuente $\{R_{i1}, \ldots\}$, que obviamente son proyectados, dado que de lo contrario ya la sentencia original estaría mal expresada.

Hay que notar que es necesario utilizar los nombres que tienen los atributos en las relaciones de la BDF en lugar de los nombres proyectados al momento de plantear la condición de la selección, dado que los nombres de los atributos podrían ser alterados por la proyección. Esto es posible si se toma la expresión original de la condición de selección, o sea se evita corregir las expresiones al modificar las correspondencias en los pasos anteriores, tal como lo hace el algoritmo de generación del esquema.

Se aclara especialmente que las condiciones siempre son establecidas sobre los atributos de la BDF. No es posible que esta selección se realice por ejemplo sobre un atributo calculado. Esto permite afirmar que no hay problema al colocar la condición de selección directamente sobre esta sentencia.

4.1.5 Paso 5 - Eliminar Atributos

El paso 5 plantea la siguiente situación:

$V^5_i \leftarrow \gamma_{Xi1, .., Xim} (V^4_i) \Rightarrow (1)$

$V^5_i \leftarrow \gamma_{Xi1, .., Xim} (\prod_{Xi1, .., Xim} (V^4_i)) \Rightarrow (2)$

$V^5_i \leftarrow \gamma_{Xi1, .., Xim} (\prod_{Xi1, .., Xim} (\prod_{Ai1 \rightarrow f(Ai1), .., f() \| Aij \rightarrow A''', .., Ain \rightarrow f(Ain), A', ti() \rightarrow A'', c \rightarrow Aiv, f(\ldots) \rightarrow Av} (\sigma_{f(Ci1, .., Cik)} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes \ldots \otimes_{V2i.C=S'i.C} (\gamma_{C, e(R'i.Bi) \rightarrow A'} R'_i) \ldots)))) \Rightarrow (3)$

$V^5_i \leftarrow \gamma_{Xi1, .., Xim} (\prod_{Xi1, .., Xim} (\sigma_{f(Ci1, .., Cik)} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes \ldots \otimes_{V2i.C=S'i.C} (\gamma_{C, e(R'i.Bi) \rightarrow A'} R'_i) \ldots))) \Rightarrow (4)$

$V^5_i \leftarrow \gamma_{Xi1, .., Xim} (\sigma_{f(Ci1, .., Cik)} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes \ldots \otimes_{V2i.C=S'i.C} (\gamma_{C, e(R'i.Bi) \rightarrow A'} R'_i) \ldots))$

1) Se agrega proyección aplicando:
   "$\gamma_L(R) = \gamma_L(\prod_M (R))$
   Si M es una lista de todos aquellos atributos R que se mencionan en L [GUW00]".
2) Sustitución.
3) Por la propiedad de Cascada del operador $\prod$ se obtiene la última proyección.

4) Se elimina la proyección aplicando nuevamente "$\gamma_L(R) = \gamma_L(\prod_M(R))$" pero ahora al revés.

Este paso tiene como objetivo eliminar los atributos sin correspondencia, para esto se debe conocer con que atributo tiene correspondencia cada ítem del fragmento al llegar a este paso. De esta forma se define el conjunto $\{X_{i1}, .., X_{im}\}$ que se utiliza en la expresión anterior. Las correspondencias que el usuario define (dato de entrada) van sufriendo alteraciones hasta llegar a esta parte del algoritmo. Dichas alteraciones serán analizadas a continuación.

En el paso 1 es necesario observar que, por cada aplicación de la Regla Join, se ajustan las correspondencias para que los ítems del modelo conceptual que apuntaban a la relaciones a unir, apunten a los atributos de la nueva relación creada (en este caso vista). Al finalizar el paso se cuenta con una única vista (V^1_{in}), a donde se refieren todos las correspondencias de tipo DirectME y 1CalcME, es decir, todas las correspondencias de estos tipos se definen exclusivamente en función de atributos de esta vista (V^1_{in}).

Observando la definición de las reglas se puede constatar que, como caso general cada vez que se aplica alguna regla se produce una nueva relación (en este caso vista), y todas las correspondencias que referían a la anterior se transportan a la nueva. En los párrafos subsiguientes el lector debe tener presente que cualquier correspondencia DirectME, 1CalcME o ExternME ∈ Versión[14] apunta siempre a la última vista creada, dado que cada vez que se crea una nueva todos las correspondencias se transportan a ésta.

En el paso 2 serán tratados los ítems del fragmento de dimensión con correspondencia DirectME, siempre y cuando exista una diferencia entre el nombre del atributo y el nombre del ítem. El tratamiento consiste en ajustar el nombre del atributo al cual refiere la correspondencia DirectME, por lo cual la correspondencia luego del paso se actualiza reflejando este cambio.

En el paso 3 las correspondencias de tipo ExternME, 1CalcME y NCalcME son alteradas para que se transformen en correspondencias DirectME. En cada caso la expresión de cálculo (función de tiempo, atributo concatenado con la versión, valor constante, cálculo simple o atributo resultado del resumen) se coloca en un atributo y la correspondencia directa se establece contra éste.

Notar que luego del paso 3 sólo existen correspondencias de tipo DirectME refiriendo atributos de la relación (en este caso vista) en construcción. Éstos no son alterados en el paso 4, simplemente se transportan hacia la vista que se produce en dicho paso.

Finalmente se puede concluir cómo se constituye el conjunto $\{X_{i1},..,X_{im}\}$ del paso 5. Está formado por atributos referidos por correspondencias DirectME de la relación (en este caso vista) producida en el paso 4. Éstas a su vez representaban como ya se explicó todas las correspondencias definidas en primera instancia. Hay que tener presente que algunos de estos atributos son resultado de expresiones introducidas en el paso 3, en cuyo caso en el paso 5 simplemente se agrupan/proyectan en función de esta expresión.

Siguiendo este análisis se puede también observar la correspondencia entre los ítems del modelo conceptual y los atributos de las relaciones del DW. Si se observa el paso 6 se notará que éste no realiza cambios sobre las correspondencias ya establecidas, sino que se limita a transportarlas a la nueva relación creada. Esto permite concluir que hay

[14] Éstas son las que refieren atributos de la BDF.

una relación directa entre los ítems del modelo conceptual y los atributos de la relación creada.

4.1.6 Paso 6 - Clave

Para el paso 6 la situación es muy simple:

$V^{6}_{i} \leftarrow \Pi_{Ai1,..,\,Ain}(V^{5}_{i}) \Rightarrow (1)$
$V^{6}_{i} \leftarrow V^{5}_{i}$

1) Se aplica $\prod_{R.*}(R) = R$, o sea la identidad del operador $\prod$, ya que por definición la primitiva aplicada en este paso realiza esta operación.

La sección 4.1 procesó todas las vistas producidas por los pasos 1 al 6 del algoritmo de generación del esquema. A continuación la sección 4.2 hace lo propio con el resto de los pasos del algoritmo. La elaboración de la vista unificada producto del análisis de estas secciones se presenta posteriormente en la sección 4.4 luego de analizar las consideraciones para los problemas encontrados en la sección 3.5.

4.2 Tratamiento de Cubos

En esta sección se considera qué ocurre con el tratamiento de los cubos en los pasos 7 al 12 del algoritmo de [Per01]. Como ya se comentó en la sección 3.3 sólo es necesario observar los cambios en el paso 11, los pasos 13, 14 y finalmente el paso 15.

4.2.1 Paso 11 - Eliminar Atributos en cubos

En el paso 11 se plantea una situación muy similar al paso 5 con la diferencia que ahora existen funciones de agregación. Estos cambios no llegan a afectar la estrategia de resolución más que en la consideración de los atributos extra.

Notar que al ser procesado este paso se genera un agrupamiento por todos los atributos con correspondencia que no son medias. Las medidas a su vez son reemplazadas por las funciones de agregación asociadas a éstas.

$V^{11}_{i} \leftarrow \gamma_{Xi1,..,Xim,ei1(Zi1),..,eik(Zik)}(V^{10}_{i}) \Rightarrow (1)$
$V^{11}_{i} \leftarrow \gamma_{Xi1,..,Xim,ei1(Zi1),..,eik(Zik)}(\prod_{Xi1,..,Xim,Zi1,..,Zik}(V^{10}_{i})) \Rightarrow (2)$
$V^{11}_{i} \leftarrow \gamma_{Xi1,..,Xim,ei1(Zi1),..,eik(Zik)}(\prod_{Xi1,..,Xim,Zi1,..,Zik}(\prod_{Ai1\rightarrow f(Ai1),\ ..,\ f()\|Aij\rightarrow A'',\ ..,\ Ain\rightarrow f(Ain),\ A',\ ti()\rightarrow A'',\ c\rightarrow Aiv,\ f(\ldots)\rightarrow Av}(\sigma_{f(Ci1,\ ..,\ Cik)}(R_{i1} \otimes_{cond(Ri1,\ Ri2)} R_{i2} \otimes \ldots \otimes_{V2i.C=S'i.C} (\gamma_{C,\ e(R'i.Bi)\rightarrow A'} R'_{i}) \ldots)))) \Rightarrow (3)$
$V^{11}_{i} \leftarrow \gamma_{Xi1,..,Xim,ei1(Zi1),..,eik(Zik)}(\prod_{Xi1,..,Xim,Zi1,..,Zik}(\sigma_{f(Ci1,\ ..,\ Cik)}(R_{i1} \otimes_{cond(Ri1,\ Ri2)} R_{i2} \otimes \ldots \otimes_{V2i.C=S'i.C} (\gamma_{C,\ e(R'i.Bi)\rightarrow A'} R'_{i}) \ldots))) \Rightarrow (4)$
$V^{11}_{i} \leftarrow \gamma_{Xi1,..,Xim,ei1(Zi1),..,eik(Zik)}(\sigma_{f(Ci1,\ ..,\ Cik)}(R_{i1} \otimes_{cond(Ri1,\ Ri2)} R_{i2} \otimes \ldots \otimes_{V2i.C=S'i.C} (\gamma_{C,\ e(R'i.Bi)\rightarrow A'} R'_{i}) \ldots))$

1) Se agrega proyección aplicando como en el paso 5 "$\gamma_L(R) = \gamma_L(\prod_M(R))$".
2) Sustitución.
3) Por la propiedad de Cascada del operador $\prod$ se elige la última proyección. Por simplicidad no se reescriben las expresiones asociadas a los atributos.
4) Se elimina la proyección aplicando nuevamente "$\gamma_L(R) = \gamma_L(\prod_M(R))$" pero ahora al revés.

4.2.2 Pasos 13 y 14 - Cubos Recursivos

Los pasos 13 y 14 se encargan de la construcción de los cubos recursivos, como ya se mencionó la estrategia para éstos es aplicar sucesivos Drill-Up's a un cubo base, hasta obtener la granularidad deseada. Este enfoque resulta útil para la definición de cubos, pero como se vio en la sección 3.5.2 tratar de operar con las instancias de esta forma introduce problemas, dado que supone dividir la agregación de un atributo en múltiples pasos, lo cual no es posible si las operaciones de agregación no son asociativas.

Dado que el problema se basa en la división en pasos de la agregación, sería difícil hacer alguna propuesta general que permita reutilizar los cubos anteriores y esté libre de problemas de asociación. En el caso de la primitiva P9, que se da al tratar de eliminar una dimensión en el paso 14 tendríamos un planteo similar al siguiente:

$V^{14.1}{}_i \leftarrow \gamma_{Xi1,..,Xim,ei1(Zi1),..,eik(Zik)}(V^{12}{}_{ij}) \Rightarrow (1)$

$V^{14.1}{}_i \leftarrow \gamma_{Xi1,..,Xim,ei1(Zi1),..,eik(Zik)}(\gamma_{X'i1,..,X'im,ei1(Zi1),..,eik(Zik)}(\sigma_{f(Ci1,\ ..,\ Cik)}(R_{i1} \otimes_{cond(Ri1,\ Ri2)} R_{i2} \otimes \ ... \ \otimes_{V2i.C=S'i.C} (\gamma_{C,\ e(R'i.Bi)\rightarrow A'} R'_i) \ ...))) \Rightarrow (?)$

1) Sustitución (V^{12}_{ij}).
2) ?

En (2) no existe una propiedad en el AR que permita hacer alguna transformación orientada a eliminar la doble agregación, mas aún es posible enumerar contraejemplos como el de la sección 3.5.2. Si se hubiera explorado el caso de la primitiva P8 cuando se intenta hacer el Drill-Up la situación sería similar:

$V^{14.1}{}_i \leftarrow \gamma_{Vi1,..,Vim,ei1(Zi1),..,eik(Zik)}(V^{12}{}_{ij} \otimes_{cond(V12ij,\ V13ij)} V^{13}{}_{ij}) \Rightarrow (1)$

$V^{14.1}{}_i \leftarrow \gamma_{Vi1,..,Vim,ei1(Zi1),..,eik(Zik)}((\gamma_{Xi1,..,Xim,ei1(Zi1),..,eik(Zik)}(\sigma_{f(Ci1,\ ..,\ Cik)}(R_{i1} \otimes_{cond(Ri1,\ Ri2)} R_{i2} \otimes \ ... \ \otimes_{V2i.C=S'i.C} (\gamma_{C,\ e(R'i.Bi)\rightarrow A'} R'_i) \ ...))) \otimes_{cond(V12ij,\ V13ij)} V^{13}{}_{ij}) \Rightarrow (2)$

$V^{14.1}{}_i \leftarrow \gamma_{Vi1,..,Vim,ei1(Zi1),..,eik(Zik)}((\gamma_{Xi1,..,Xim,ei1(Zi1),..,eik(Zik)}(\sigma_{f(Ci1,\ ..,\ Cik)}(R_{i1} \otimes_{cond(Ri1,\ Ri2)} R_{i2} \otimes \ ... \ \otimes_{V2i.C=S'i.C} (\gamma_{C,\ e(R'i.Bi)\rightarrow A'} R'_i) \ ...))) \otimes_{cond(V12ij,\ V13ij)} (\gamma_{Xi1',..,Xim'}(\sigma_{f(Ci1',\ ..,\ Cik')}(R_{i1'} \otimes_{cond(Ri1',\ Ri2')} R_{i2'} \otimes \ ... \ \otimes_{V2i.C=S'i.C} (\gamma_{C,\ e(R'i.Bi)\rightarrow A'} R'_i) \ ...)))) \Rightarrow (?)$

1) Sustitución (V^{12}_{ij}).
2) Sustitución (V^{13}_{ij}).
3) ?

Esta complejidad inherente a enfocar el problema mediante la reutilización del resultado anterior, junto con el hecho de que no está claro que se obtenga algún beneficio, lleva a tomar otro camino. Una alternativa más simple es obviar la estrategia de resolución de los pasos 13 y 14 utilizando la misma que para los cubos base, dado que es posible plantearse obtener el cubo recursivo, calculando las correspondencias con la BDF que éste tendría y utilizando la estrategia de los pasos 7 al 12. Puesto que definir un cubo en forma recursiva, o utilizando directamente las correspondencias a la BDF es equivalente por hipótesis, no es posible que esto afecte el resultado obtenido.

Lo anterior en resumen significa solamente un cambio de estrategia. La cadena de primitivas obtenida por los pasos 13 y 14 refieren a una estrategia para alterar un cubo y obtener otro. Previamente al paso 13 se antepone una cadena de primitivas para obtener un resultado intermedio (cubo base). El cambio de estrategia es simple: porqué no hacer que el resultado intermedio sea el final, uniendo la especificación del resultado intermedio con la de la transformación.

Para obtener las correspondencias con la BDF un breve análisis de los pasos 13 y 14 muestra que se pude proceder como sigue. Cada cubo recursivo tiene asociado un cubo base y por lo tanto existen correspondencias para éste. Tomemos estas correspondencias como punto de partida. Cada cubo recursivo tiene asociado un conjunto de Drill-Up's donde para cada uno de éstos podemos observar lo siguiente. En caso de un Drill-Up total (eliminar la dimensión del cubo recursivo), es necesario eliminar el o los ítems del cubo asociados a la dimensión en cuestión. Esto se traduce en eliminar las correspondencias asociadas a los ítems. En caso de un Drill-Up parcial (subir niveles en la dimensión) es necesario considerar las correspondencias de los niveles origen y destino. Quitando las correspondencias del nivel origen y agregando las del nivel destino será suficiente. Luego de utilizar esta estrategia para todos los Drill-Up's del cubo recursivo habremos obtenido las correspondencias de éste.

Esta estrategia presenta solamente un problema. Éste se da por las funciones de Roll-Up asociadas a las medidas, en particular cuando éstas son diferentes de las funciones de agregación asociadas a las medidas. El problema es que en una sola vista no se pueden incluir N agrupamientos distintos, de forma de poder tener múltiples agrupamientos. Este problema no será tratado en este documento, sin embargo, hemos de decir que en [RA05] al mencionar el problema de "*Multiple Aggregation Problem*" se plantea una extensión al lenguaje SQL que permite manejar esta situación.

El siguiente procedimiento resume lo antes dicho, procesando todos los cubos recursivos y alterando las correspondencias según se estableció[15]:

```
RecursiveToBaseMappings:
∀ C ∈ SchCubes.(SchCMappings(C) ∈ RecursiveCMappings
     ∧ SchCMappings(C).Cub ∈ BaseCMappings)
   /* un cubo con mapeo recursivo, en función de un cubo con mapeo base */
   Sea <Cbase,Dups, Map> = SchCMappings(C) /* el mapeo del cubo */

   ∀ Dup ∈ Dups /* para cada drill-up del mapeo del cubo */
      Map_Lbase = SchCMappings(ObjectItems(Dup.Lbase)).Map
         /* correspondencias de los niveles base */
      Map_Lnews = SchCMappings(ObjectItems(Dup.Lnews)).Map
         /* correspondencias de los niveles destino */
      Si Map_Lnews ∈ NoDetailDrillUps(Cbase,C) /* si se elimina la dimensión */
         Map = Map − Map_Lbase      /*elimino el nivel base */
      Sino /* si tiene detalle para la dimensión */
         Map = Map − Map_Lbase /*elimino correspondencias del nivel base */
         Map = Map ∪ Map_Lnews /*agrego correspondencias del nuevo nivel */
      Fin /* si */
   Fin /* para todo */

   SchCMapping(C) ≡<Map, ⊥, ⊥> /* Ahora el cubo tiene un mapeo base */
Fin /* para todo */
```

[15] La notación utilizada para el algoritmo es la propuesta por [Per01]. Ésta se comenta brevemente en la sección 4.4.1 al presentar el algoritmo para la construcción de la vista unificada.

Ejemplo 19: Cubos recursivos con la nueva propuesta de carga

Si tomamos el siguiente caso:

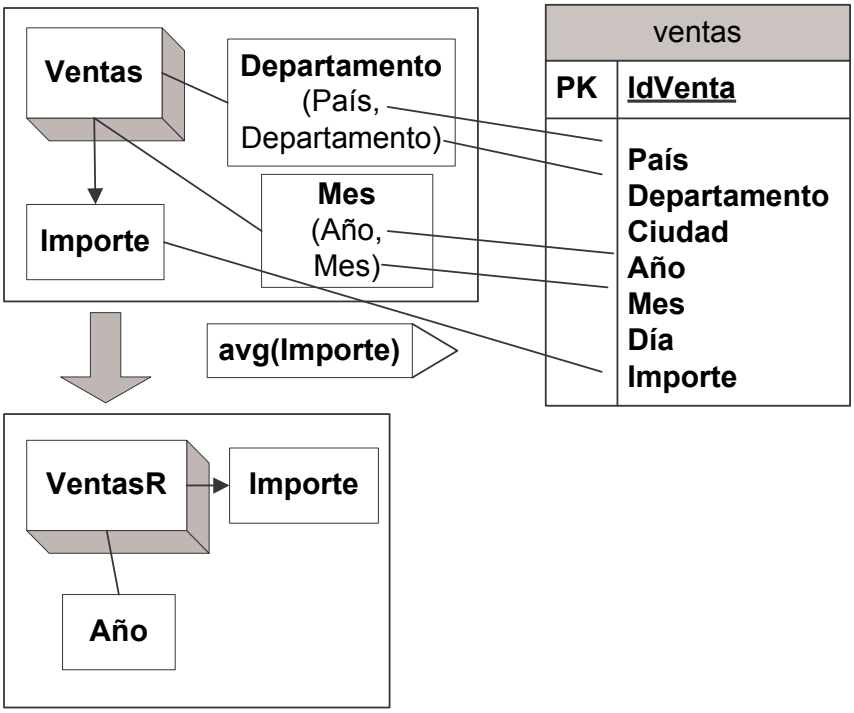


Figura 29: Ejemplo de nueva propuesta de carga con cubos recursivos.

El Ejemplo 19 es muy similar al Ejemplo 11 sólo que en este caso una de las dimensiones (Departamento) desaparece completamente, dando lugar a utilizar todas las estrategias antes mencionadas. Además de esto, para simplificar se eliminó el problema causado por el NCalcME (será abordado posteriormente) para calcular el Importe, en su lugar se utilizó una correspondencia directa como se puede ver en la Figura 29. El resultado de calcular las nuevas correspondencias para el cubo recursivo se ve en la siguiente figura:

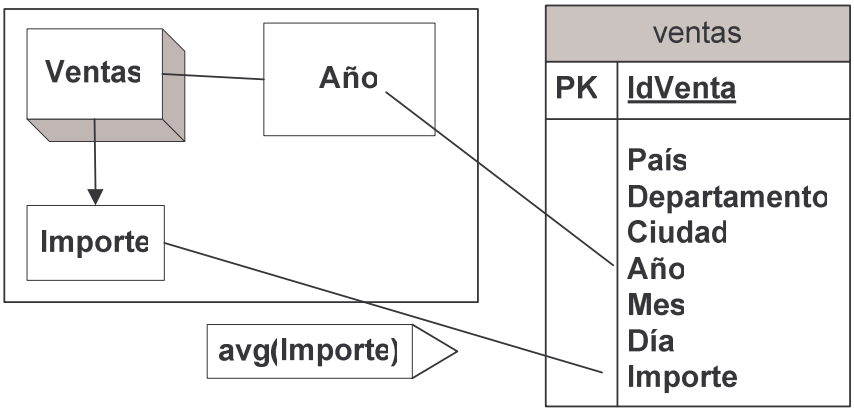


Figura 30: Nuevas correspondencias para el Ejemplo 19.

Por último si aplicáramos los pasos 7 al 12 para las correspondencias anteriores obtendríamos sólo una vista en el paso 11 (la del paso 12 no es significativa) que sería:

```
create view V11i as
   select ventas.Año, avg(ventas.Importe) as Importe
   from ventas
   group by ventas.Año
```

♦

Para finalizar sólo queremos remarcar que dado que este procedimiento observa las transformaciones realizadas en las correspondencias y utiliza el mismo algoritmo, es necesario que tenga el mismo resultado que si hubiera sido planteado directamente en términos de correspondencias con la BDF. Ésta es la razón por la cual no se plantea demostración sobre la equivalencia o igualdad de los resultados.

4.2.3 Paso 15 - Franjas

Por último sólo resta por mencionar el paso 15 donde se dividen los cubos en las franjas que sean necesarias. Un aspecto interesante es que las expresiones que determinan las franjas se escriben en función de los ítems del cubo, por lo cual será necesario cambiar esta expresión por una en función de los atributos de la fuente (ver Ejemplo 20). Para esto consideremos las posibles expresiones asociadas a un ítem: si se trata de una correspondencia directa se utiliza el atributo de la correspondencia, en caso de un cálculo simple se utiliza la expresión asociada así como para el caso de constante, dígito de versión o marca de tiempo. Si fuera el caso que se tratara de un NCalcME bastaría con utilizar el atributo donde se prevé que esté el valor resumido (ej.: e en la SQL de la sección 4.1.3.5).

Considerando que en el caso de vistas entrelazadas (enfoque Naive) esto se realiza automáticamente al utilizar el atributo calculado una secuencia de pasos más atrás, podemos deducir cómo se justifica la estrategia. El planteo para este caso es el siguiente:

$V^{15}_i \leftarrow \sigma_{f(Ci1, .., Cik)} V^{12}_i \Rightarrow (1)$

$V^{15}_i \leftarrow \sigma_{f(Ci1, .., Cik)} (\gamma_{Xi1,..,Xim,ei1(Zi1),..,eik(Zik)} (\sigma_{f(Ci1', .., Cik')} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes ... \otimes_{V2i.C=S'i.C} (\gamma_{C, e(R'i.Bi)\rightarrow A'} R'_i) ...)) \Rightarrow (2)$

$V^{15}_i \leftarrow \gamma_{Xi1,..,Xim,ei1(Zi1),..,eik(Zik)} (\sigma_{f(Ci1, .., Cik) \text{ and } f(Ci1', .., Cik')} (R_{i1} \otimes_{cond(Ri1, Ri2)} R_{i2} \otimes ... \otimes_{V2i.C=S'i.C} (\gamma_{C, e(R'i.Bi)\rightarrow A'} R'_i) ...))$

1) Sustitución, utilizamos V^{12}_i pero claramente puede tratarse de un cubo recursivo que como se vio es idéntico al caso de un cubo base.
2) Fácilmente es demostrable que la selección puede introducirse en el agrupamiento, dado que se realiza sobre los atributos expuestos $\{X_{i1},..,X_{im}\}$.

Ejemplo 20: Franjas de cubos.

Considerando el Ejemplo 18 y suponiendo que una franja se define por el predicado Código = "UY", la vista generada para la franja será:

```
create view V3i as
   select País, Población'.Población, Territorio'.Territorio,
     TODAY as Fecha, "1_" || País as Versión, "Ignacio" as Usuario,
     IF País == "Uruguay" THEN "UY" ELSE "Desconocido" as Código
   from Países,
     (select Población.País, sum(Población.Población) as Población
       from Población
       group by Población.País) Población',
     (select Territorio.País, sum(Territorio.Territorio) as Territorio
       from Territorio
       group by Territorio.País) Territorio'
   where Países.País=Población'.País and Países.País=Territorio'.País
     and (IF País == "Uruguay" THEN "UY" ELSE "Desconocido") == "UY"
```

Notar que en la vista anterior como la condición Código = "UY" es sobre atributos del cubo (que en particular se obtienen por cálculo simple) al traducirla se utilizó la expresión asociada a la correspondencia.

♦

Esta sección y la anterior contribuyeron a analizar cómo se deben manejar las vistas generadas por cada uno de los pasos con el objetivo de crear una única vista. Antes de presentar cómo esta vista puede construirse aprovechando el conocimiento generado por este análisis, la sección siguiente analiza los problemas mencionados en el enfoque Naive.

4.3 Problemas del Enfoque Naive

En la sección 3.5 hicimos mención a una serie de problemas introducidos por el enfoque Naive. Ésta en parte fue una de las razones que impulsó a realizar una nueva propuesta. Ahora es tiempo de demostrar o completar la nueva propuesta para contemplar los problemas planteados. Esta sección se dedica a comentar estos temas.

4.3.1 Violación de Clave Primaria

El problema de violación de clave primaria que se presentó en el Ejemplo 10, ocurre dado que el enfoque Naive no prevé que la relación obtenida tras el paso 5, no tenga la clave que fue declarada para el fragmento de dimensión, asumiendo que si esto no es así se trata de un error de diseño.

La diferencia sustancial en el Ejemplo 10 es que no hay un error en la clave del fragmento, efectivamente país es clave del fragmento y si el cálculo se hubiera realizado en un solo paso agrupando por la clave del fragmento, no hubiera ocurrido ningún inconveniente. La siguiente sentencia muestra lo antes descrito:

```
create view Sol as select Ciudad_Pais.pais,
     avg(Temperaturas.temperatura) as temperatura
   from Ciudad_Pais, Temperaturas
   where Ciudad_Pais.ciudad = Temperaturas.ciudad
   group by Ciudad_Pais.pais
```

Con lo que hemos presentado hasta el momento de la nueva propuesta, la vista generada para la situación del Ejemplo 10 mantendrá los problemas de la vista del enfoque Naive. Si se calcula la vista con la nueva propuesta ésta es:

```
create view V5i as select Ciudad_Pais.pais, S.temperatura
  from Ciudad_Pais,
     (select ciudad, avg(temperatura) as temperatura
     from Temperaturas
     group by ciudad) S
  where Ciudad_Pais.ciudad = S.ciudad
  group by Ciudad_Pais.pais, S.temperatura
```

Tanto la vista del enfoque Naive como la que hemos presentado para la nueva propuesta no permiten una solución basada en volver a agregar el atributo (agregar avg(S.temperatura) y quitar este atributo del GROUP BY). Como la operación de agregación no es asociativa el resultado al volver a agregar no tiene porqué ser correcto, como se mostró en el Ejemplo 11.

Al observar la vista Sol se pude tender a pensar que la solución es seguir la estrategia que ésta sugiere (realizar la agregación del NCalcME junto con la principal). Sin embargo hay que observar que si se tuviera una tercer relación la estrategia ya no sería aplicable, dado que induciría un problema de múltiple conteo, similar al presentado en el Ejemplo 12.

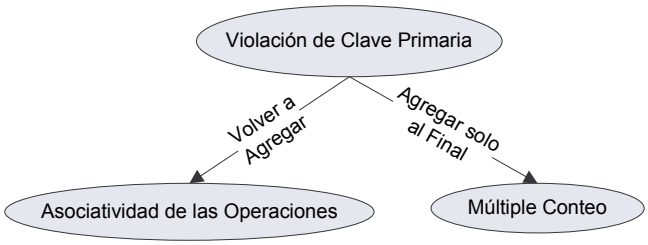


Figura 31: Problemas asociados a la violación de clave primaria.

Para estudiar la razón por la cual se produce el problema debemos considerar porqué se viola la clave primaria del fragmento, es decir analizar porqué no se cumple la dependencia funcional Clave Fragmento -> $A_{NCalcME}$, donde $A_{NCalcME}$ es el atributo resultado del NCalcME. Al observar el problema se puede notar que existe una diferencia entre el nivel de granularidad (o detalle) del fragmento (Países) y el que posee el resultado de agregar el atributo a incorporar (Temperaturas.temperatura). Mientras que el fragmento está definido por país, el NCalcME se calcula por ciudad.

Como se supone que este atributo ya está agregado y con granularidad menor o igual al fragmento y esto no es así [(16)], se está violando la dependencia funcional. Por ende el resultado intermedio es incorrecto y a la larga genera que el resultado final también lo sea. El Ejemplo 10 aprovecha esta vulnerabilidad proponiendo un juego de datos que precisamente rompe la dependencia funcional para mostrar el error. Hay que señalar que este análisis tiene sentido si el atributo $A_{NCalcME}$ no pertenece a la clave del fragmento, dado que si pertenece a la clave del fragmento la dependencia se cumple trivialmente.

[16] Para hacer esta suposición hay que basarse en cómo está diseñado el algoritmo. Si no estuviera tomando esta hipótesis (Clave Fragmento -> $A_{NCalcME}$) se debería hacer algo al respecto para evitar este problema (agrupar, filtrar, etc.).

Notar que nos referimos al nivel de granularidad del fragmento y no del esqueleto. En el Ejemplo 10 el nivel de detalle del esqueleto Ciudad_País está definido por <ciudad, país> mientras que en el fragmento está definido sólo por país. El nivel de granularidad del resultado del NCalcME está como en el caso del esqueleto definido por <ciudad, país> (por más que sólo se encuentre el atributo ciudad). El error se produce porque el atributo temperatura, proveniente del NCalcME, no se encuentra al nivel de detalle del fragmento (país) y el algoritmo supuso que sí se encontraría a este nivel.

A partir de lo anterior se deduce que una solución es asegurar que el nivel de granularidad del fragmento y el de los resultados del NCalcME sean iguales, es decir, disminuir el nivel de granularidad del NCalcME hasta alcanzar el del fragmento. Para lograr esto una alternativa es forzar a que el resultado del NCalcME cumpla con la clave del fragmento. Luego realizando el join en base a dicha clave se elimina la posibilidad de obtener repetidos en el agrupamiento del paso 5. La razón por la cual ya no hay repetidos es que ahora sí es válida la dependencia funcional país -> temperatura o en forma genérica Clave Fragmento -> $A_{NCalcME}$. Esta dependencia funcional se está forzando en el cálculo del NCalcME, por estar forzando que la clave del fragmento también sea la del NCalcME.

La vista que se produciría en el Ejemplo 10 replanteando el NCalcME en función de la clave del fragmento (país) y definiendo los links correspondientes, sería la siguiente:

```
create view V5i as select Ciudad_Pais.pais, S.temperatura
   from Ciudad_Pais,
      (select a.pais, avg(b.temperatura) as temperatura
      from Ciudad_Pais a, Temperaturas b
      where a.ciudad = b.ciudad
      group by a.pais) S
   where Ciudad_Pais.pais = S.pais
   group by Ciudad_Pais.pais, S.temperatura
```

Esta transformación que puede parecer muy simple vista rápidamente, tiene problemas para su implementación en forma automática. Por lo tanto se prefirió advertir la situación de violación de clave primaria y dejar al diseñador el replanteo del problema, utilizando si lo desea esta guía sobre una alternativa para solucionarlo. En definitiva la propuesta que estamos realizando se puede enunciar de la siguiente forma:

> *Siempre que un atributo no perteneciente a la clave del fragmento sea obtenido mediante NCalcME, se debe analizar qué relación guarda el nivel de granularidad del fragmento con respecto al NCalcME. Siempre que el NCalcME tenga mayor granularidad que el fragmento, se está frente a una situación de violación de clave primaria y se debe replantear el problema.*

La diferencia en el nivel de granularidad es fácilmente detectable considerando la clave del fragmento y los atributos por los cuales se agrupa el NCalcME. Siempre que haya algún atributo del agrupamiento que no pertenezca a la clave del fragmento habrá problema, pues el fragmento tiene menor granularidad que el NCalcME. En el Ejemplo 10 ciudad no pertenecía a la clave del fragmento (país).

Al párrafo anterior es necesario hacerle una precisión sobre el significado de "*algún atributo del agrupamiento que no pertenezca a la clave del fragmento*". Los atributos del NCalcME en general no pertenecen al esqueleto, que es donde está la clave del fragmento, dado que se trata de relaciones diferentes, por lo cual sin agregar algo más esta cita no tiene sentido. El aspecto que resta mencionar es cómo comparar atributos de

relaciones diferentes. Para esto diremos que los links definen clases de equivalencia para los atributos de las relaciones de la BDF, dado que establecen la igualdad de atributos de diferentes relaciones. Comparando en base a estas clases de equivalencia es que podemos decir que un atributo de una relación está en un conjunto de atributos de otra relación. Por ejemplo, si un link establece que A.a = B.a (un atributo a de la relación A es igualado a un atributo a de una relación B), las clases de equivalencia nos permiten decir que considerando la relación B, tener o comparar contra el atributo a, es lo mismo que tener o comparar contra el atributo a de la relación A.

4.3.2 Asociatividad de las Operaciones

En la sección 3.5.2 se presenta otro de los problemas que ocurren al llevar a cabo la carga utilizando el enfoque Naive. El problema está asociado al hecho de que si la operación de agregación no es asociativa, no se puede dividir el cálculo del resultado. En el Ejemplo 11 se pueden observar todos los puntos del algoritmo donde se puede presentar el problema. Tomando una operación no asociativa como el promedio, se mostró que en el extremo, para promediar una medida, se dividía el cálculo en por lo menos en 3 instancias. Primero se calculaba una vez en una correspondencia de tipo NCalcME, otra vez al eliminar atributos en el cubo base, y luego múltiples veces al hacer Roll-Up para alcanzar el cubo recursivo.

En la sección 4.2.2, al abordar el tema de los cubos recursivos, se redujo el problema presentado a sólo tener que resolver el problema de los cubos base, ya que se optó por utilizar la misma estrategia para éstos que para los cubos recursivos. Esto elimina del Ejemplo 11 sólo los problemas ocasionados por el Roll-Up, porque ahora éstos no se hacen, pero no afecta los otros puntos de problema. Esta sección presenta cuales serán las acciones a tomar al encontrar el problema de asociatividad al armar los cubos base, que es la parte que ha quedado pendiente.

La situación problemática ocurre siempre que, una medida, con función de agregación no asociativa, es obtenida utilizando la misma función de agregación por NCalcME. En dicho caso la resolución de la agregación se realiza en dos etapas, al resolver el NCalcME y al eliminar atributos, lo que claramente es un error. Si las funciones de agregación del NCalcME y de la medida son distintas, esta discusión no aporta valor pues no existe otro camino para realizar la transformación de los datos. Si la operación de agregación es asociativa esta propiedad habilita la división del cálculo.

Una alternativa para dar una solución a este problema, es utilizar el mismo mecanismo para calcular el NCalcME que la sección 4.3.1, dado que este calculaba toda la agregación en un solo paso, evitando el problema en cuestión. Es necesario considerar que como se trata de medidas, esto implica eliminar la operación de agregación del paso 11, pues como el NCalcME ya estará al nivel de agregación del cubo, no es correcto volver a agregar. Además la clave utilizada para realizar el join será parte de la del cubo, o sea parte de las claves de los niveles que componen el cubo. Se trata sólo de una parte porque sólo se deben involucrar las claves de los niveles en las que tiene atributos el NCalcME.

Hay que notar que la solución de la sección 4.3.1 no siempre es aplicable, en particular como mostrará el Ejemplo 21 porque pueden no estar disponibles los atributos para hacer el join. Tanto en esta ocasión como en la anterior no hay otra solución que pedirle al diseñador que plantee el caso de una forma que no presente estos problemas.

Si observamos ambos problemas en forma abstracta, se puede notar que el problema de la violación de la clave primaria, está relacionado con el problema de asociatividad, dado que ambos tienen en común las diferencias en los niveles de granularidad entre el resultado (cubo / fragmento) y el NCalcME. En ambos casos el problema siempre se presenta en los ítems que no están en la clave (fragmento / cubo). Y en ambos casos una solución es siempre calcular el NCalcME al nivel del resultado.

En resumen la propuesta para resolver el problema de asociatividad será:

Siempre que una medida, con agregación no asociativa, sea obtenida utilizando la misma función de agregación por NCalcME y además los niveles de granularidad del cubo y el NCalcME sean diferentes, se indicará el error de asociatividad solicitando que se replantee el caso en cuestión. Considerando el mismo caso pero si los niveles de granularidad fueran iguales, se considerará correcto y se omitirá volver a agregar el atributo correspondiente a la medida, dado que éste ya fue agregado al nivel deseado.

Ejemplo 21: Solución al problema de asociatividad.

En el Ejemplo 11, si revemos el paso 9 para el cubo VentasR (recursivo), considerando las precisiones de la sección 4.2.2 sobre como calcular el cubo recursivo, pero sin considerar la propuesta de esta sección tendremos la siguiente vista:

```
create view V9i as
   select ventas.IdVenta, ventas.País, ventas.Departamento,
     ventas.Ciudad, ventas.Año, ventas.Mes, ventas.Día, S.Importe
   from ventas,
      (select IdVenta, avg(Importe) as Importe
      from lineas_venta) S
   where ventas.IdVenta = S.IdVenta
   group by ventas.IdVenta, ventas.País, ventas.Departamento,
     ventas.Ciudad, ventas.Año, ventas.Mes, ventas.Día
```

En lo anterior hay que notar que esta vista plantea la resolución del NCalcME original y esto ocasiona el problema de asociatividad de la misma forma que la propuesta Naive. Si agregamos a ésta las consideraciones realizadas en esta sección, e intentamos la solución propuesta en la sección 4.3.1, la sentencia cambiaría calculando el NCalcME al nivel de granularidad del cubo. Para hacer esto es necesario navegar hasta la tabla ventas de forma de obtener el nivel de granularidad necesario, dado que lineas_venta no tiene los atributos año, mes:

```
create view V9i as
   select ventas.IdVenta, ventas.País, ventas.Departamento,
     ventas.Ciudad, ventas.Año, ventas.Mes, ventas.Día, S.Importe
   from ventas,
      (select ventas.Año, ventas.País,
        avg(lineas_venta.Importe) as Importe
      from lineas_venta, ventas
      where lineas_venta.IdVenta = ventas.IdVenta) S
   where ventas.Año = S.Año and ventas.País = S.País
   group by ventas.IdVenta, ventas.País, ventas.Departamento,
     ventas.Ciudad, ventas.Año, ventas.Mes, ventas.Día
```

El problema con lo anterior es que no es válido dado que no representa un NCalcME. Es necesario entonces ensayar otra solución. Para este caso el camino es simple, en lugar

de plantear una correspondencia de tipo NCalcME planteemos una directa para el importe. De esta forma la primera vista se presentará en el paso 7 de forma de armar el esqueleto:

```
create view V7i as
   select ventas.IdVenta, ventas.País, ventas.Departamento,
      ventas.Ciudad, ventas.Año, ventas.Mes, ventas.Día,
      lineas_venta.IdVenta, lineas_venta.producto, lineas_venta.Importe
   from ventas, lineas_venta
   where lineas_venta.IdVenta = ventas.IdVenta
```

Posteriormente en el paso 11 se tendrá:

```
create view V11i as
   select V7i.Año, V7i.País, avg(V7i.Importe) as Importe
   from V7i
   group by V7i.Año, V7i.País
```

En definitiva, la alternativa que se sugirió en la sección 4.3.1 no pudo ser utilizada, pero en su lugar se propuso otra alternativa que resolvió el problema. El valor aportado por el algoritmo es no dejar que se produzcan resultados inconsistentes, impidiendo que los diseños que los provocarían se lleven a cabo.

♦

4.3.3 Múltiple Conteo

El problema de múltiple conteo del Ejemplo 12 se ocasiona por construir el esqueleto de un cubo utilizando links de cardinalidad 1-N, teniendo al mismo tiempo medidas en ambos extremos del link. En el ejemplo la medida ubicada en el extremo con cardinalidad 1, se ve incrementada N veces luego de realizado el join entre ambas relaciones(17).

En un fragmento de dimensión, construir esqueletos con links de cualquier cardinalidad no tiene mayor importancia, dado que no se agrega información y los duplicados simplemente se eliminan (paso 5). En un cubo el valor de una medida es resultado de una agregación, por lo que tener una tupla más, implica un valor más a agregar y por ende un resultado final generalmente distinto (paso 11).

En el Ejemplo 12 se utilizaron ítems con correspondencia directa, pero aparecería un problema similar si se definiera una medida con una correspondencia constante o de cálculo simple y el esqueleto no tuviera cardinalidad 1. El problema aparece porque el cálculo de la agregación se verá desvirtuado por las repeticiones de tuplas en el esqueleto, tal y como ocurre con el Ejemplo 12. En el caso de las correspondencias constantes es posible asumir que ésta es el resultado de la agregación y omitir colocar la expresión de agregación de la medida. En el caso de cálculo simple se optó por reportar el error, dejando para un trabajo futuro hacer un análisis detallado del tema que permita identificar casos correctos a pesar de la cardinalidad N del esqueleto.

Las correspondencias de tipo versión y marca de tiempo no se considera que puedan ser medidas, por lo cual no se trataron en el párrafo anterior. Las correspondencias de tipo resumen (NCalcME), en la nueva propuesta se resuelven independientemente (la

[17] El problema de múltiple conteo esta relacionado con la violación de la condición necesaria de separación (disjointness) de las tuplas a agregar, establecida en las condiciones de sumarización [LS97].

agregación se hace en una subconsulta, ver sección 4.1.3.5) y luego se hace el join del resultado con el esqueleto, por lo cual tampoco se afectan.

Una opción para este problema es clasificarlo como error de diseño, dado que se puede poner en tela de juicio si es correcto utilizar links de cardinalidad distinta de 1-1 en la construcción del esqueleto. Siguiendo esta línea estableceríamos que todos los links deben ser de cardinalidad 1-1, para asegurar que no haya cambios de granularidad y dejaríamos que los atributos de distinta cardinalidad se resolvieran utilizando el NCalcME, que es el medio natural para este problema.

El inconveniente es que la propuesta es demasiado restrictiva, muchas de las relaciones presentes en una base de datos normal son de cardinalidad 1-N (pe.: claves foráneas) y por otro lado no siempre se utilizan los links para obtener atributos resumidos, por lo cual el NCalcME no es una solución para éstos. Si en el Ejemplo 11 agregamos una dimensión Producto, con el atributo producto de las líneas de venta, tenemos una situación en la que es imposible no utilizar link 1-N entre las ventas y las lineas_venta. Por otro lado tampoco podemos utilizar un NCalcME para solucionarlo dado que no tiene sentido aplicarlo para el atributo producto.

Dado entonces que la opción de error de diseño no puede ser considerada como solución, debemos elaborar una nueva propuesta. En el caso del Ejemplo 12, se puede plantear una alternativa como la presentada en el Ejemplo 22, estudiemos esta solución para ver qué se puede extraer de ella.

Ejemplo 22: Solución de cambio de correspondencias al problema de múltiple conteo.

Una solución al problema del Ejemplo 12 es:

- modificar las correspondencias directas de País y Departamento para referir a la relación Territorio.
- eliminar la correspondencia directa a Población y en su lugar agregar una de tipo resumen: Población -> NCalcME(<avg(Población.Población), Población, Población.Población>).
- Conservar el link que une Territorio y Población en función de País y Departamento.

De esta forma se elimina el join problemático, ya que no es necesario utilizar el link de cardinalidad 1-N. El planteo con estas modificaciones comienza en el paso 9 con la incorporación del atributo resumido, pues el esqueleto es sólo la relación Territorio (paso 7) y no es necesario hacer correcciones de nombres de atributos (paso 8):

```
create view V9i as
  select Territorio.País, Territorio.Departamento, S.Población,
    Territorio.Territorio
  from Territorio,
    (select País, Departamento, avg(Población.Población) as Población
    from Población
    group by País, Departamento) S
  where Territorio.País = S.País and
    Territorio.Departamento = S.Departamento
```

Notar que se está aplicando la observación de la sección 4.3.2, por la cual, ya que el NCalcME y el cubo están al mismo nivel de granularidad, no se producirá un problema de asociatividad. Para comprobar la igualdad en el nivel de granularidad debemos puntualizar qué, a partir de las clases de equivalencia inducidas por el link, los atributos

con los que contamos en el NCalcME (País, Departamento de Población) coinciden con la clave del cubo (País, Departamento de Territorio), por ende están al mismo nivel de granularidad. Las clases de equivalencia inducidas por las igualdades planteadas por el link entre Población y Territorio son:

- c_1 = {Población.País, Territorio.País},
- c_2 = {Población.Departamento, Territorio.Departamento}

Recordemos que si no fuera por estas clases de equivalencia, no habría fundamento para decir que los atributos del NCalcME son los mismos que la clave del cubo, dado que los atributos son de distintas relaciones.

A continuación el paso 11 genera la siguiente vista, donde no se está volviendo a agrupar a causa de la observación de la sección 4.3.2:

```
create view V^11_i as
   select V^9_i.País, V^9_i.Departamento, V^9_i.Población,
      sum(V^9_i.Territorio) as Territorio
   from V^9_i
   group by V^9_i.País, V^9_i.Departamento, V^9_i.Población
```

♦

Como ocurrió en la sección 4.3.1 y 4.3.2, existe una solución al problema, lo cual muestra que utilizando otra forma de resolución es suficiente. Hay que notar que independientemente de la solución, será necesario relevar más información sobre los links, para poder identificar la situación problemática. Es necesario contar con la cardinalidad del link, dado que como ya se mencionó si la cardinalidad es 1-1, el problema no existe. La extensión para contar con la cardinalidad se hace en el apéndice D, sección D.5.

Notemos que si se utiliza la estrategia inducida por el Ejemplo 22 (cambiar correspondencias y utilizar NCalcME), entonces el problema de múltiple conteo nunca se presenta, ya que el join problemático nunca se realiza. El problema con esta estrategia, es que puede no ser posible eliminar el join, dado que todos los atributos del esqueleto no tienen porque estar en alguna de las relaciones.

Otra estrategia es calcular todas las medidas problemáticas mediante NCalcME. De esta forma no es necesario eliminar el join del esqueleto, aunque los links podrían seguir causando la repetición de tuplas (en el esqueleto). El hecho de que las medidas se calculen por separado (NCalcME), evita la obtención de resultados incorrectos, salvo que éstos ya fueran incorrectos desde su definición (pe.: un link definido incorrectamente), en cuyo caso estaríamos frente a un error de diseño. En algunos casos es posible simplificar y algunas medidas podrían seguir siendo calculadas por el paso 9, sin embargo, para no oscurecer el planteo no entraremos en este tema.

Ejemplo 23: Solución con medidas NCalcME al problema de múltiple conteo.

En el paso 7 se construye el esqueleto como pasaba inicialmente, sólo hay que tener presente que ahora ni el atributo Población ni Territorio es la fuente de la medida, sino que se obtienen por NCalcME, por esto se le cambian los nombres a los atributos para que no colisionen:

```
create view V7i as
   select Población.País, Población.Departamento,
     Población.Ciudad as Población_Ciudad,
     Población.Población as Población_Población,
     Territorio.País as Territorio_País,
     Territorio.Departamento as Territorio_Departamento,
     Territorio.Territorio as Territorio_Territorio
   from Población, Territorio
   where Población.País = Territorio.País and
     Población.Departamento = Territorio.Departamento
```

En el paso 9 se calculan todas las medidas por NCalcME. Observar como se reexpresan los cálculos de la medida, utilizando la relación fuente de la correspondencia (anteriormente directa) y la función de Roll-Up de la medida. Notar también como se establece el join al nivel de granularidad del cubo:

```
create view V9i1 as
   select V7i.País, V7i.Departamento, V7i.Población_Ciudad,
     V7i.Población_Población, V7i.Territorio_País,
     V7i.Territorio_Departamento, V7i.Territorio_Territorio,
     S.Población
   from V7i,
     (select País, Departamento, avg(Población) as Población
     from Población
     group by País, Departamento) S
   where V7i.País = S.País and V7i.Departamento = S.Departamento

create view V9i2 as
   select V9i1.País, V9i1.Departamento, V9i1.Población_Ciudad,
     V9i1.Población_Población, V9i1.Territorio_País,
     V9i1.Territorio_Departamento, V9i1.Territorio_Territorio,
     V9i1.Población, S.Territorio
   from V9i1,
     (select País, Departamento, avg(Territorio) as Territorio
     from Territorio
     group by País, Departamento) S
   where V7i.País = S.País and V7i.Departamento = S.Departamento
```

Notar que a partir de las clases de equivalencia inducidas por el link, los atributos con los que contamos en V^9_{2i} coinciden con la clave del cubo. Las clases de equivalencia inducidas por las igualdades planteadas por el link entre Población y Territorio son las mismas que en el Ejemplo 22; la diferencia es que ahora Población es la relación que define el esqueleto y la clave del fragmento (puesto que no estamos alternado el ejemplo original, cosa que sí se hacía en el Ejemplo 22).
Finalmente en el paso 11 se eliminan los repetidos y los atributos sobrantes:

```
create view V11i as
   select V9i2.País, V9i2.Departamento, V9i2.Población, V9i2.Territorio
   from V9i2
   group by V9i2.País, V9i2.Departamento
```

♦

En resumen entonces la propuesta será:

Siempre que exista algún link utilizado para armar el esqueleto que tenga cardinalidad distinta de 1-1, hacer lo siguiente:

- *Verificar que todas las medidas obtenidas con correspondencia directa estén al mismo nivel de granularidad. Si esto no ocurre indicar que ha ocurrido un error de múltiple conteo y solicitar que se replantee el problema.*
- *Si alguna de las medidas fuera obtenida por correspondencia de tipo constante, no aplicarle la operación de agregación y notificar la acción tomada.*
- *Si alguna de las medidas fuera obtenida por cálculo simple, indicar que ha ocurrido un error de múltiple conteo y solicitar que se replantee el problema.*

Si fuera necesario replantear el problema los ejemplos de esta sección intentan facilitar el desarrollo.

El problema de múltiple conteo también es mencionado en la literatura como "Fan-Shaped Problem" [RA05]. En [RA05] aunque se enfoca el problema desde otro ángulo (planteo de consultas multidimensionales en soluciones ROLAP), se caracteriza de forma muy similar[18]. Comparando las soluciones propuestas podemos concluir que el utilizar consultas anidadas es equivalente al planteo de las medidas obtenidas por NCalcME, las extensiones al SQL siempre resultan una alternativa y el manejo de la materialización en nuestro caso no corresponde.

4.4 Vista Unificada

En esta sección se presenta un nuevo algoritmo de carga. En éste la atención se centra en construir una única vista o vista unificada para cada fragmento de dimensión o franja de cubo. Esto es posible gracias a que las secciones 4.1 y 4.2 presentan las bases para hacerlo. La nueva versión no admite que ocurran los problemas enumerados para el enfoque Naive, dado que se utiliza lo presentado en la sección 4.3 para identificarlos y solicitar la corrección en los datos de entrada si es necesario.

El nuevo algoritmo resume un conjunto de observaciones, sobre qué aporta cada primitiva utilizada en el enfoque Naive, a la construcción de la vista unificada. A su vez también obedece las puntualizaciones realizadas en las secciones 4.1, 4.2 y 4.3, sobre casos particulares y estrategias para evitar obtener datos erróneos. Cada vista, de la misma forma que ocurría en el caso Naive, apunta a definir los fragmentos de las dimensiones o las franjas de cubos. También es de notar que se está utilizando la misma información que en el enfoque Naive, o sea la misma que se utilizó en la construcción del esquema de la base de datos. La siguiente tabla enumera las observaciones más importantes consideradas en el algoritmo:

[18] En [RA05] se presentan problemas muy relacionados con este trabajo. Por un lado se menciona el problema denominado "Multiple Aggregation", que aquí se trató en la sección 4.2.2 bajo el nombre de Asociatividad de Operaciones. También se menciona el problema denominado "Selection Granularity", pero éste se consideraría un error de diseño en el contexto de este trabajo. Más detalle sobre [RA05] puede encontrarse en [ASS03].

<table>
<tr><td>Esqueleto / Joins</td><td colspan="3">Si el fragmento de dimensión o franja de cubo se vinculara (correspondencias DirectME, 1CalcME y ExternME de tipo Version) con más de una relación en la BDF, entonces todas las operaciones de join que se realizan en el paso 1, se encontrarán en la vista unificada.</td></tr>
<tr><td colspan="4">Correspondencias</td></tr>
<tr><td colspan="2">Directas
Todos los ítems con correspondencia directa estarán como el correspondiente atributo proyectado en la vista unificada (ver Medidas), si fuera necesario se ajustará el nombre.</td><td>Marca de Tiempo
Cada marca de tiempo que se utilice aparecerá en la vista unificada en forma de un atributo calculado mediante una función que define el valor de la marca de tiempo.</td><td>Dígito de Versión
Cada ítem producto de agregar un dígito de versión aparecerá en la vista unificada como un atributo que se calcula como lo indica la primitiva P4.1 (sección 4.1.3).</td></tr>
<tr><td colspan="2">Constante
Cada constante que se utilice aparecerá en la vista unificada como un atributo con valor constante. (ver Medidas)</td><td>Cálculo Simple
Cada cálculo simple aparecerá en la vista unificada como un atributo que toma su valor al evaluar la expresión correspondiente (ver Medidas).</td><td>Cálculo Resumen
Cada cálculo de resumen aparecerá en la vista unificada como un atributo que toma su valor de una subconsulta en la cláusula FROM. Está se vincula a través de un join con las relaciones del esqueleto (sección 4.1.3).</td></tr>
<tr><td>Condiciones</td><td colspan="3">Si existieran condiciones para los datos contenidos en el fragmento de dimensión o franja de cubo, éstas se agregarán a la vista unificada.</td></tr>
<tr><td>Atributos sobrantes y duplicados</td><td colspan="3">Sólo los ítems con correspondencia definirán los atributos de la vista, todos los otros serán filtrados mediante una operación de agrupamiento. Notar que este agrupamiento también producirá el efecto de eliminar eventuales tuplas repetidas.</td></tr>
<tr><td>Medidas</td><td colspan="3">Para todos los atributos provenientes de correspondencias directas o expresiones provenientes de constantes o cálculos simples que fueran a agregarse a la vista, debe tomarse en cuenta si se trata de una medida. En caso afirmativo, en lugar de agregarse el atributo o expresión de cálculo, se agregará su expresión de agregación (sección 4.2.1).</td></tr>
<tr><td>Cubos Recursivos</td><td colspan="3">Se ha de utilizar la misma estrategia que para los cubos base, transformando la definición en una no recursiva utilizando el procedimiento RecursiveToBaseMappings.</td></tr>
<tr><td>Franjas</td><td colspan="3">Cada franja de cubo agrega más condiciones a las introducidas por las otras observaciones, éstas son precisamente las que definen la franja.</td></tr>
<tr><td>Consideración de Clave Primaria</td><td colspan="3">Siempre que un atributo no perteneciente a la clave del fragmento sea obtenido mediante NCalcME, se debe analizar qué relación guarda el nivel de granularidad del fragmento con respecto al NCalcME. Siempre que el NCalcME tenga mayor granularidad que el fragmento, se está frente a una situación de violación de clave primaria y se debe replantear el problema.</td></tr>
</table>

Consideración de Asociatividad	Siempre que una medida, con agregación no asociativa, sea obtenida utilizando la misma función de agregación por NCalcME y además los niveles de granularidad del cubo y el NCalcME sean diferentes, se indicará el error de asociatividad solicitando que se replantee el caso en cuestión. Considerando el mismo caso pero si los niveles de granularidad fueran iguales, se considerará correcto y se omitirá volver a agregar el atributo correspondiente a la medida, dado que éste ya fue agregado al nivel deseado.
Consideración de Múltiple Conteo	Siempre que exista algún link utilizado para armar el esqueleto que tenga cardinalidad distinta de 1-1, hacer lo siguiente: • Verificar que todas las medidas obtenidas con correspondencia directa estén al mismo nivel de granularidad. Si esto no ocurre indicar que ha ocurrido un error de múltiple conteo y solicitar que se replantee el problema. • Si alguna de las medidas fuera obtenida por correspondencia de tipo constante, no aplicarle la operación de agregación y notificar la acción tomada. • Si alguna de las medidas fuera obtenida por cálculo simple, indicar que ha ocurrido un error de múltiple conteo y solicitar que se replantee el problema.

Tabla 6: Observaciones que guian la construcción del algoritmo de carga.

4.4.1 Algoritmo Principal

A continuación se presenta el algoritmo utilizado para obtener las vistas asociadas a cada fragmento de dimensión o franja de cubo. Dicho algoritmo está escrito en función de las mismas definiciones utilizadas en [Per01] para el algoritmo de generación del esquema del DW. Esto hace que requiera para su entendimiento del conocimiento del vocabulario introducido en [Per01]. Para permitir la lectura de lo siguiente se comienza refrescando informalmente el mencionado vocabulario de definiciones.

- SchFragments: Abreviatura que representa el conjunto de todos los fragmentos de dimensión del esquema conceptual. $O \in$ SchFragments es un fragmento de dimensión en particular (ver apéndice E, sección E.3 por ejemplos de fragmentos de dimensión).
- SchCubes: Conjunto de todos los cubos del esquema conceptual definidos por el diseñador. $O \in$ SchCubes es uno de estos cubos (ver apéndice E, sección E.3 por ejemplos de cubos).
- SchLinks: Es uno de los componentes de la definición de la base de datos fuente (LogialSchema en el vocabulario de [Per01]) y representa el conjunto de links entre tablas que define como se realizan los joins. Formalmente es una función de pares de tablas en un predicado. SchLinks(T1,T2) puede ser idealizado como un predicado aplicable a una cláusula WHERE en una vista SQL (ver apéndice E, sección E.2 por ejemplos de links)
- SchFMappings: Es una función que a cada fragmento le hace corresponder una función de correspondencia y opcionalmente una condición. <Map, Cond> = SchFMappings(O). Esto es las correspondencias (Map) asociadas al fragmento (O) y la condición (Cond) sobre los datos a considerar de la

fuente (ver apéndice E, sección E.4.1 por ejemplos de correspondencias de fragmentos).

- SchCMappings: Es una función que a cada cubo le hace corresponder un mapeo base o un mapeo recursivo de cubo. Particularmente en el algoritmo se utilizará el mapeo base por lo cual será de la forma: <Map, Cond, Rup> = SchCMappings(O), donde: Map son las correspondencias asociadas al cubo (O), Cond es la condición sobre los datos a considerar de la fuente y Rup son las expresiones de Roll Up asociadas (ver apéndice E, sección E.4.2 por ejemplos de correspondencias de cubos).
- SchCFragmentantion: Es una función que a cada cubo le hace corresponder un conjunto de franjas en las cuales se fragmenta el cubo. Una franja es un predicado que delimita un conjunto de los datos del cubo (ej.: mes >= enero, ver apéndice E, sección E.3 por ejemplos de fragmentación de cubos).
- MapTables: Es una función que devuelve el conjunto de tablas que tienen correspondencias con un conjunto de ítems dados. Recordar que en el apéndice D, sección D.4 se está modificando levemente esta función sin afectar su objetivo.
- MapAttributes: Es una función que devuelve los atributos que tienen correspondencia a ítems de un conjunto dado. Recordar que en el apéndice D, sección D.4 se está modificando levemente esta función sin afectar su objetivo.
- ObjectItems: Es una función que devuelve los ítems asociados al objeto indicado (ej.: los ítems de una dimensión). Recordar que un item es un atributo de un nivel en CMDM.
- ObjectKeyItems: Es una función que devuelve los ítems que identifican al objeto dado. (ej.: los ítems que identifican a un nivel de una dimensión).
- DirectME, ExternME, 1CalcME, NCalcME: Son los conjuntos que definen los distintos tipos de correspondencias. Por ejemplo Map(I) ∈ DirectME significa que el ítem tiene una correspondencia de tipo directa.

```
(1) Execute RecursiveToBaseMappings

(2) ∀ O ∈ SchFragments ∪ SchCubes
      If O ∈ SchFragments
        <Map,Cond> = SchFMappings(O)
(3)     Ranges = { ⊥ }
      Else If O ∈ SchCubes
        <Map,Cond,Rup> = SchCMappings(O)
        MeasureItems = ObjectItems(O.Measure)
        Ranges = SchCFragmentation(O)
      End-If

(4)   ∀ F ∈ Ranges
        V_select = "select "
        V_from = "from "
        V_select = "where "
        V_groupby = "group by "

        Execute Skeleton
        Execute DirectMappings
```

```
      Exectue TimeMappings
      Execute VersionMappings
      Execute ConstantMappings
      Execute SimpleMappings
      Execute AggregateMappings
      Execute Conditions
      Execute ViewAssembling
    End-∀
End-∀
```

Primeramente observar que a causa de (1) no hay cubos definidos recursivamente, dado que RecursiveToBaseMappings (ver sección 4.2.2) redefine todos los cubos definidos de esta forma, eliminando todas las correspondencias recursivas.

El código anterior recorre tanto los fragmentos como los cubos (2), instanciando en cada caso las correspondencias, franjas (Ranges) y medidas según corresponde (observar que la variable MeasureItems se inicializa con los ítems asociados a las medidas). Dada la similitud en el tratado de los fragmentos y cubos esta estrategia resulta más clara y sencilla. El único artilugio que debió introducirse para proceder de esta forma es relativo a las franjas en los fragmentos de dimensión. Las franjas son un concepto propio de los cubos y no de los fragmentos, sin embargo en el código anterior, se genera una franja ficticia (3) con el único objetivo de permitir la ejecución en la línea (4).

Finalmente para cada franja se inicializan las variables y se invocan los procedimientos que tratan cada una de las posibles situaciones. El último de éstos (ViewAssembling) es el responsable del armado de la vista.

4.4.2 Tratamiento del Esqueleto

```
   Skeleton:
      card_Esqueleto = 1
      Ts = {}
(1)   While Ts ≠ MapTables(Map, ObjectItems(O))
(2)     Let T ∈ MapTables(Map, ObjectItems(O)) - Ts;
        If V_from ≠ "from "
          V_from = V_from || ", "

(3)     V_from = V_from || T

        If #Ts > 0
(4)       Let T' ∈ { t ∈ Ts / SchLinks(T, t) ≠ ⊥ }
(5)       <card_T, card_T'> = SchLinksCards(T, T')

(6)       card_Esqueleto = Max(card_Esqueleto, card_T, card_T')

          If V_where ≠ "where "
            V_where = V_where || " and "
          End-If

(7)       V_where = V_where || SchLinks(T, T')
        End-If

        Ts = Ts ∪ T
      End-While
```

```
    If (O ∈ SchCubes ∧ card_Esqueleto ≠ 1 ∧
(8)       #GranularitySetsOfMeasures(O, Ts) ≠ 1)
      Error "No todas las medidas de " || O || " estan al mismo
        nivel de granularidad, ocasionando un problema de
        múltiple conteo".
    End-If
  End-Skeleton
```

El código anterior está recorriendo las tablas que tienen correspondencias con los ítems para agregarlas a la cláusula FROM de la vista (3). Al mismo tiempo coloca las condiciones de join necesarias en la cláusula WHERE (7) y calcula la cardinalidad máxima de los links del esqueleto (6). Esto último es útil para detectar un caso de múltiple conteo.

Para recorrer las tablas con correspondencias se realiza un cubrimiento del grafo formado por las relaciones y los links del esqueleto (ver líneas 2 y 4). Dicho cubrimiento alcanza todas las relaciones del esqueleto, utilizando sólo los links que sean necesarios (algunos pueden no utilizarse). Eventualmente podrían existir formas alternativas de realizar el cubrimiento pero esto no debería afectar el resultado.

En la línea (1) vale la pena notar que si se observa la definición de MapTables, se verá que utiliza las correspondencias de tipo DirectME, 1CalcME y ExternME de tipo Versión para decidir el conjunto de tablas con correspondencia (ver apéndice D, sección D.4).

Las cardinalidades de los links (5) se requieren para detectar el problema de múltiple conteo presentado en la sección 4.3.3. Las extensiones necesarias para poder contemplarlas se encuentran en el apéndice D, sección D.5.

Al mismo tiempo que se está recorriendo el grafo formado por las relaciones y links, se está calculando la máxima cardinalidad (6) involucrada en los links utilizados (1 o N). Ésta será la cardinalidad del esqueleto ($card_{Esqueleto}$), usada posteriormente en el caso de los cubos para saber si existe un problema de múltiple conteo o no.

Finalmente en (8), la función GranularitySetsOfMeasures(O, Ts) da los conjuntos con misma granularidad de las medidas, es decir, divide las medidas en niveles, donde cada nivel contiene medidas que se alcanzan por links de cardinalidad 1. Si hay más de un conjunto significa que hay medidas a distintos niveles de granularidad y por tanto un problema de múltiple conteo.

4.4.3 Tratamiento de Correspondencias Directas

```
  DirectMappings:
    ∀ I ∈ { I ∈ ObjectItems(O) / Map(I) ∈ DirectME }
      If V_select ≠ "select "
        V_select = V_select || ", "

      If O ∈ SchCubes ∧ I ∈ MeasureItems
(1)     V_select = V_select || Rup(I) || " as " || I.ItemName
      Else
        If Map(I).Patt.AttrName ≠ I.ItemName
(2)       V_select = V_select || Map(I).Patt || " as " || I.ItemName
        Else
(3)       V_select = V_select || Map(I).Patt
      End-If
```

```
(4)     If O ∉ SchCubes ∨ I ∉ MeasureItems
          If V_groupby ≠ "group by "
            V_groupby = V_groupby || ", "

(5)       V_groupby = V_groupby || Map(I).Patt
        End-If
      End-∀
```

El código anterior coloca los atributos correspondientes a los ítems con correspondencia directa en la cláusula SELECT (1, 2 y 3) En la misma recorrida también se agregan los atributos en la cláusula GROUP BY (5), en caso que corresponda (ver a comentario de línea 4). Hay que notar que en el ∀ sólo se consideran los ítems con correspondencia directa (Map(I) ∈ DirectME).

En la recorrida se debe prestar atención a que el ítem sea o no una medida, porque de ser medida corresponde aplicar la función de resumen (ver línea 1 y sección 4.2.1). Además puede ser necesario corregir Rup(I) para agregar los nombres de las relaciones de las cuales provienen los atributos[19], eliminando así posibles ambigüedades. En el algoritmo esto no se expresa para simplificar la presentación.

Observar que sólo se está cambiando el nombre a los atributos para los cuales es necesario hacerlo (ver sección 3.2.2). Esto da origen a las alternativas presentadas en (2) y (3). Tener en cuenta también que por definición Map(I).Patt contiene el nombre de la relación de la BDF como prefijo (ver apéndice D, sección D.4), por esto no se hace la nota sobre agregar el nombre de la relación.

Por último observar también que cada atributo con correspondencia directa pertenece al agrupamiento, siempre y cuando no se trate de una medida (ver secciones 4.1.5 y 4.2.1).

4.4.4 Tratamiento de Marcas de Tiempo

```
TimeMappings:
  ∀ I ∈ { I ∈ ObjectItems(O) / Map(I) ∈ ExternME ∧
    Map(I).Ind = Timestamp }
    If V_select ≠ "select "
      V_select = V_select || ", "

(1)   V_select = V_select || Map(I).Expr || " as " || I.ItemName

    If V_groupby ≠ "group by "
      V_groupby = V_groupby || ", "

(2)   V_groupby = V_groupby || Map(I).Expr
  End-∀
```

El código anterior agrega a la cláusula SELECT (1) y a la cláusula GROUP BY (2) la expresión asociada a las marca de tiempo (ver el Ejemplo 5 para un caso concreto). Como antes hay que observar que sólo se consideran en el ∀ los ítems con correspondencia de tipo marca de tiempo. Tomar en cuenta también que sería un error de diseño definir un Timestamp como medida, dado que una medida procede de un valor numérico.

[19] El nombre de la relación se encuentra en la definición de la correspondencia.

4.4.5 Tratamiento de Dígito de Versión

```
VersionMappings:
  ∀ I ∈ { I ∈ ObjectItems(O) / Map(I) ∈ ExternME ∧
    Map(I).Ind = Version }
    If Vselect ≠ "select "
      Vselect = Vselect || ", "

(1)   Vselect = Vselect || " (" || Map(I).Expr || " || " ||
          Map(I).Patt || ") as " || I.ItemName

    If Vgroupby ≠ "group by "
      Vgroupby = Vgroupby || ", "

(2)   Vgroupby = Vgroupby || Map(I).Expr || " || " || Map(I).Patt
  End-∀
```

El código anterior agrega a la cláusula SELECT (1) y a la cláusula GROUP BY (2) la expresión asociada al manejo de versión de un atributo (ver el Ejemplo 6 para un caso concreto de este tipo de correspondencia).

Notar que Map(I).Expr representa la posibilidad de utilizar cualquier función que permita versionar el atributo (ver sección 3.2.3.2). Recordar que se alteró la definición de las correspondencias de tipo dígito de versión para contener la tabla y expresión asociada (ver apéndice D, sección D.4).

Nuevamente observar que sería un error definir una medida que proviene de una correspondencia de tipo dígito de versión, entre otras cosas porque la naturaleza de la expresión utilizada (concatenación de cadenas de texto) hace que el valor no sea numérico.

4.4.6 Tratamiento de Constantes

```
ConstantMappings:
  ∀ I ∈ { I ∈ ObjectItems(O) / Map(I) ∈ ExternME ∧
    Map(I).Ind = Constant }
    If Vselect ≠ "select "
      Vselect = Vselect || ", "

    If O ∈ SchCubes ∧ I ∈ MeasureItems ∧ cardEsqueleto = 1
(1)   Vselect = Vselect || Rup(I) || " as " || I.ItemName
    Else
(2)   Vselect = Vselect || Map(I).Expr || " as " || I.ItemName

    If O ∈ SchCubes ∧ I ∈ MeasureItems ∧ cardEsqueleto ≠ 1
(3)   Warning "Asumiento " || Map(I).Expr || " como valor final
        agregado de la medida, por el problema de múltiple conteo".
  End-∀
```

El código anterior agrega a la cláusula SELECT la expresión correspondiente a la constante, pero para esto debe tomar en cuenta si se trata o no de una medida. En el caso de tratarse de una medida se agrega la expresión de agregación asociada a ésta (1), siempre y cuando la cardinalidad del esqueleto ($card_{Esqueleto}$)[20] sea igual a 1. Cuando

[20] Ver la sección 4.4.2 por más información sobre la cardinalidad del esqueleto.

no se trata de una medida o el esqueleto no tiene cardinalidad 1, se coloca la expresión asociada a la constante (2).

Tratándose de una medida, si la cardinalidad del esqueleto no es 1 y se utilizara la función de agregación, se induciría un problema de múltiple conteo, ya que no estaría claro cuantas veces se consideraría el valor de esta constante (ver sección 4.3.3). Para evitar este problema se considera que el valor de la constante ya es el resultado de la agregación y no el valor a agregar (línea 2). En la línea (3) se genera una advertencia previniendo al usuario precisamente de esta decisión.

4.4.7 Tratamiento de Cálculos Simples

```
   SimpleMappings:
     ∀ I ∈ { I ∈ ObjectItems(O) / Map(I) ∈ 1CalcME }
       If V_select ≠ "select "
         V_select = V_select || ", "

       If O ∈ SchCubes ∧ I ∈ MeasureItems ∧ card_Esqueleto ≠ 1
         Error "La medida " || I || " de " || O || " no puede
           ser calculada correctamente si el esqueleto no tiene
           cardinalidad 1".

       If O ∈ SchCubes ∧ I ∈ MeasureItems
(1)      V_select = V_select || Rup(I) || " as " || I.ItemName
       Else
(2)      V_select = V_select || Map(I).Expr || " as " || I.ItemName

       If O ∉ SchCubes ∨ I ∉ MeasureItems
         If V_groupby ≠ "group by "
           V_groupby = V_groupby || ", "

(3)      V_groupby = V_groupby || Map(I).Expr
       End-If
     End-∀
```

Este código siguiendo la línea de los casos anteriores agrega a la cláusula SELECT (1, 2) y a la cláusula GROUP BY (3) la expresión asociada al cálculo simple. Como en la sección anterior hay que notar que si se trata de una medida y la cardinalidad del esqueleto no es 1 se genera un problema de múltiple conteo, ya que no estaría claro cuantas veces se consideraría el valor de la expresión (ver sección 4.3.3).

4.4.8 Tratamiento de Cálculos de Resumen

```
   AggregateMappings:
     ∀ I ∈ { I ∈ ObjectItems(O) / Map(I) ∈ {NCalcME, NCalcME'} }
(1)    Let <T,L> ∈ { <t,l> / t∈ Ts ∧ l ∈ SchLinks(Map(I).Tab, t) ≠ ⊥ }

       If V_select ≠ "select "
         V_select = V_select || ", "

(2)    V'_name = "V_" || I.ItemName

(3)    DifferentGranularity =
         Att_T(L) −(≡) MapAttributes(O, ObjectKeyItems(O))♦AttrName ≠ ∅
```

```
        If (O ∈ SchFragments ∧ I ∉ ObjectKeyItems(O)
             ∧ DifferentGranularity)
(4)        Error "Se ha producido un error de clave primaria en el
             item " || I || " del fragmnto " || O
        End-If
        Aggregate = True
        If (O ∈ SchCubes ∧ I ∈ MeasureItems ∧ NoAsoc(Rup(I)
          ∧ Rup(I) = Map(I).Expr)
          If (DifferentGranularity)
(5)         Error "Se ha producido un error de asociatividad en la
              medida " || I || " del cubo " || O
          Else
(6)         Aggregate = False
          End-If
        End-If

        If O ∈ SchCubes ∧ I ∈ MeasureItems ∧ Aggregate
(7)       V_select = V_select || Rup(I) || " as " || I.ItemName
        Else
(8)       V_select = V_select || V'_name || "." || I.ItemName

        If V_from ≠ "from "
          V_from = V_from || ", "

(9)     V' = "select " || Att_Map(I).Tab(L) || ", " || Map(I).Expr ||
             " as " || I.ItemName || " from " || Map(I).Tab ||
             " group by " || Att_Map(I).Tab(L)

(10)    V_from = V_from || "(" || V' || ") as " || V'_name

        If V_where ≠ "where "
          V_where = V_where || " and "

(11)    V_where = V_where || L

        If O ∉ SchCubes ∨ I ∉ MeasureItems ∨ Aggregate = False
          If V_groupby ≠ "group by "
            V_groupby = V_groupby || ", "

(12)      V_groupby = V_groupby || V'_name || "." || I.ItemName
        End-If
      End-∀
```

El código anterior agrega a la cláusula FROM (10) las subconsultas necesarias para el cálculo de los atributos resumidos. Simultáneamente es agregado a la cláusula SELECT (7 o 8) el atributo de la subconsulta que alberga el resultado. Se opera de la misma forma con la cláusula GROUP BY (12) si corresponde. Las condiciones de join entre la subconsulta y el esqueleto también son introducidas en la cláusula WHERE (11).

La complejidad adicional presentada por este código está relacionada con la detección de los problemas de asociatividad (5) y clave primaria (4). Para ambos es necesario conocer si la granularidad del NCalcME es mayor que la del fragmento o cubo (3). Con este fin se comprueba si existen atributos utilizados en el join, que no pertenezcan a la

clave del fragmento (esto es la resta de atributos planteada)[21]. Por último es necesario aclarar que $Att_R(P)$, es una función que permite obtener los atributos de la relación R utilizados en el predicado P.

Además de lo anterior es necesario notar que:

- En (1) se está tomando alguno de los links con el esqueleto para resolver el NCalcME. Tal y como se explicó en la sección 3.2.3.5 el link debe existir y si hubieran varias posibilidades serían equivalentes.
- V'_{name} (2) es el nombre temporal que se le dará a la subconsulta V'. La estrategia para designar V'_{name} resulta conveniente pues los nombres de los atributos no pueden repetirse.
- En (6) si los niveles de granularidad son iguales no hay problema de asociatividad, pero no se debe volver a agregar.
- Es necesario ajustar la expresión Rup(I) (7) para que refiera al atributo obtenido por la subconsulta V'.
- En (9) se arma la expresión de cálculo de V'. Notar que se utilizan del link L (1) los atributos en función de los cuales se expresa el predicado ($Att_{Map(I).Tab}(L)$) y la expresión (Map(I).Expr) y relación a resumir.

4.4.9 Tratamiento de Condiciones

```
Conditions:
   If Cond ≠ ⊥
      If V_where ≠ "where "
         V_where = V_where || " and "

      V_where = V_where || "(" || Cond || ")"
   End-If

   If F ≠ ⊥
      If V_where ≠ "where "
         V_where = V_where || " and "

(1)   V_where = V_where || "(" || F.Cond || ")"
   End-If
```

Tener presente que la condición asociada a una franja de cubo (1) está escrita en función de los ítems, por lo cual es necesario sustituirlos por su correspondiente expresión como se mencionó en la sección 4.2.3.

[21] Prestar atención a que se utiliza la diferencia de los conjuntos de atributos según las clases de equivalencia definidas por los links ($-_{(\equiv)}$), ver sección 4.3.3. De esta forma se pueden restar atributos de distintas tablas que representan lo mismo. Según la resta tradicional los atributos serían distintos por pertenecer a tablas distintas.

4.4.10 Armado de la Vista

ViewAssembling:

(1) V = "create view " || viewname || " as " || V_{select}

(2) If V_{from} ≠ "from "
V = V || " " || V_{from}

If V_{where} ≠ "where "
V = V || " " || V_{where}

If $V_{groupby}$ ≠ "group by "
$V_{groupby}$ = V || " " || $V_{groupby}$

Notar que en (1) el nombre de la vista a crear es el nombre del fragmento si se trata de un fragmento, o el nombre del cubo o franja, dependiendo si el cubo se dividió en franjas o no.

En (2) una situación como la del Ejemplo 7 produce la necesidad de considerar que pueda no haber V_{from}, V_{where} y $V_{groupby}$. No es necesario tomar precauciones respecto a V_{select}, dado que no aún en el caso de la traza vacía (presentado en la sección 3.4), existen atributos con correspondencia, por lo cual éstos se agregarán al V_{select}.

4.5 Análisis de Costos

Uno de los objetivos para tomar una sola vista representando la transformación en lugar de una secuencia de éstas, es que la solución con una sola vista ofrece un mejor desempeño al ser ejecutada en un RDBMS. No es muy difícil observar que la mayoría de las deducciones de este capítulo no hacen que una alternativa tenga mejor desempeño que la otra, ya que fácilmente pueden ser deducidas por un optimizador de sentencias SQL. Si en la secuencia de vistas originada por el enfoque Naive, se analiza el plan de consulta para resolver la última de éstas, el optimizador aplicará la mayoría de las observaciones realizadas en esta sección.

Sin embargo hay una transformación utilizada en la sección 4.1.3 referida a los mapeos NCalcME que no puede ser deducida por un optimizador. Ésta propone utilizar:

$$\Pi_{Att(R),e}(R \otimes_{cond(R,R')} \gamma_{CR',E\rightarrow e}(R'))$$

en lugar de cómo se planteó originalmente en la definición de la primitiva:

$$\gamma_{Att(R),E\rightarrow e}\ (R \otimes_{cond(R,R')} R')$$

Para evidenciar la mejora se analizará exclusivamente este caso. Sea entonces el siguiente caso referido a construcción de cubos.

Ejemplo 24: Ejemplo de análisis de costos.

Sean las siguientes relaciones:

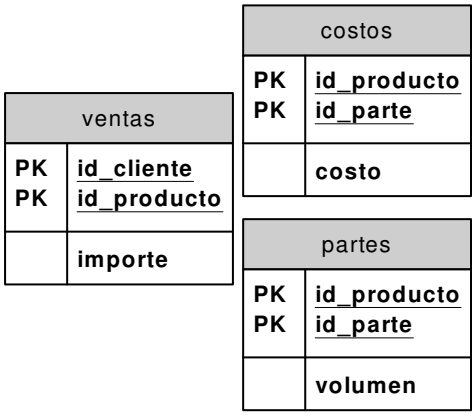


Figura 32: Ejemplo de relaciones para análisis de costos.

Sea una tabla de ventas, donde se registra que se vendió un producto a un cliente cobrando por esto determinado importe.

```
create table ventas (
   id_cliente number(8),
   id_producto int,
   importe number(8,2),
   constraint ventas_pk primary key (id_cliente, id_producto));
```

Sea una tabla costos donde se guardan los costos de cada parte que conforma el producto venido.

```
create table costos (
   id_producto int,
   id_parte int,
   costo int,
   constraint costos_pk primary key (id_producto, id_parte));
```

Y por último sea una tabla partes donde se encuentra el volumen de cada una de las partes.

```
create table partes (
   id_producto int,
   id_parte int,
   volumen int,
   constraint partes_pk primary key (id_producto, id_parte));
```

Supongamos que en el DW se desea un cubo como el que muestra la Figura 33, donde además del importe de cada venta, también estén los costos y el volumen. Dadas las relaciones planteadas anteriormente, una forma de llegar al resultado es definir dos correspondencias NCalcME, donde se resuman los datos de las relaciones costos y partes [(22)].

[22] Notar que con correspondencias directas aparece el problema de múltiple conteo, por eso no se utilizaron.

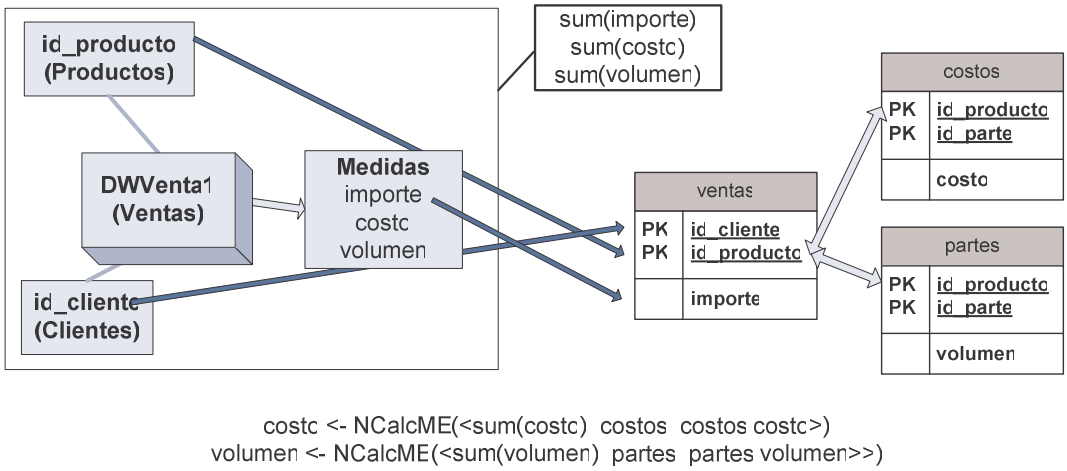


Figura 33: Cubo para el análisis de ventas.

Se plantean en primera instancia las vistas (A, B) que se producirían en el paso 3 del algoritmo para manejar las correspondencias NCalcME, según el enfoque Naive:

```
create view A as
      select p.id_cliente, p.id_producto,
            sum(c.costo) as costo
      from ventas p, costos c
      where p.id_producto = c.id_producto
      group by p.id_cliente, p.id_producto;

create view B as
      select p.id_cliente, p.id_producto, p.costo,
            sum(v.volumen) as volumen
      from A p, partes v
      where p.id_producto = v.id_producto
      group by p.id_cliente, p.id_producto, p.costo;
```

Y por otro lado, la vista que se propone para resumir ambas en la nueva propuesta de carga (véase como ambos agrupamientos son llevados a dos sub-consultas en el FROM):

```
create view C as
      select p.id_cliente, p.id_producto, c.costo, v.volumen
      from ventas p,
            (select id_producto, sum(costo) as costo
             from costos group by id_producto) c,
            (select id_producto, sum(volumen) as volumen
             from partes group by id_producto) v
      where p.id_producto = v.id_producto and
            v.id_producto = c.id_producto;
```

Para estimar los costos se recurrió a un Oracle versión 9.2.0.1.0 en el cual se crearon las relaciones y vistas antes mencionadas. Como muestra de datos se pobló la relación ventas con 65536 registros generados a partir de las siguientes sentencias:

```
create sequence seq_ventas start with 1 increment by 1 nocache
nocycle;
```

```
insert into ventas values (seq_ventas.nextval, 1, 1000);
insert into ventas values (seq_ventas.nextval, 2, 2000);
insert into ventas values (seq_ventas.nextval, 3, 3000);
insert into ventas values (seq_ventas.nextval, 4, 4000);

insert into ventas
  select seq_ventas.nextval, id_producto, importe from ventas;
```

Esta última sentencia se repitió hasta obtener el número de tuplas indicado. A su vez las relaciones costos y partes se poblaron de forma similar contando cada una con 64 registros. A modo de ejemplo sólo se presenta una de ellas:

```
create sequence seq_costo start with 1 increment by 1 nocache nocycle;

insert into costos values (1, 0, 1);
insert into costos values (2, 0, 1);
insert into costos values (3, 0, 1);
insert into costos values (4, 0, 1);

insert into costos
  select id_producto, seq_costo.nextval, costo from costos;
```

Nuevamente esta última sentencia se repitió hasta obtener la cantidad de tuplas deseadas.

Al pedir los planes de ejecución para B y C considerando retornar todas las tuplas, Oracle retorno lo siguiente:

Explain (Cost: 76860) | Compact View | Statistics

Execution Step	Order	Est Cost	Est Row Count	Est Byte Count
SELECT STATEMENT	11	76860	185364	5004828
SORT (GROUP BY)	10	76860	185364	5004828
TABLE ACCESS (BY INDEX ROWID) OF 'IGNACIO.PARTES'	9		16	96
NESTED LOOPS	8	148	2965824	80077248
VIEW 'IGNACIO.A'	6	148	185364	3892644
SORT (GROUP BY)	5	148	185364	2595096
TABLE ACCESS (BY INDEX ROWID) OF 'IGNACIO.COSTOS'	4		8	48
NESTED LOOPS	3	148	524288	7340032
INDEX (FULL SCAN) OF 'IGNACIO.VENTAS_PK' (UNIQUE)	1	148	65536	524288
INDEX (RANGE SCAN) OF 'IGNACIO.COSTOS_PK' (UNIQUE)	2		8	
INDEX (RANGE SCAN) OF 'IGNACIO.PARTES_PK' (UNIQUE)	7		16	

Figura 34: Plan de ejecución y costos Vista Original (B).

Explain (Cost: 21) | Compact View | Statistics

Execution Step	Order	Est Cost	Est Row Count	Est Byte Count
SELECT STATEMENT	12	21	65536	3932160
HASH JOIN	11	21	65536	3932160
VIEW	4	1	4	104
SORT (GROUP BY)	3	1	4	24
TABLE ACCESS (BY INDEX ROWID) OF 'IGNACIO.COSTOS'	2	1	32	192
INDEX (FULL SCAN) OF 'IGNACIO.COSTOS_PK' (UNIQUE)	1		32	
HASH JOIN	10	18	65536	2228224
VIEW	8	1	4	104
SORT (GROUP BY)	7	1	4	24
TABLE ACCESS (BY INDEX ROWID) OF 'IGNACIO.PARTES'	6	1	64	384
INDEX (FULL SCAN) OF 'IGNACIO.PARTES_PK' (UNIQUE)	5		64	
INDEX (FAST FULL SCAN) OF 'IGNACIO.VENTAS_PK' (UNIQUE)	9	16	65536	524288

Figura 35: Plan de ejecución y costos Vista Propuesta (C).

La diferencia de costos estimados como se observa en las figuras es significativa: mientras que la vista B tiene un costo estimado de 78860, la vista propuesta (C) tiene un costo estimado de 21 (aproximadamente 3755 veces menos). El principal problema como se puede ver en la Figura 34 es que el agrupamiento final se calcula sobre la tabla

de hechos, mientras que en la Figura 35 esto no ocurre, los agrupamientos son en las tablas de resumen. Al mismo tiempo se puede ver que los conjuntos de tuplas involucrados en la Figura 35 son del orden de la cardinalidad de ventas, mientras que en la Figura 34 se llegan a manejar hasta 44 veces más tuplas (paso 8).

La misma prueba se realizó sobre PostgreSQL obteniendo para la vista B el siguiente plan y estimación de costos:

```
HashAggregate  (cost=109107.47..109189.39 rows=6554 width=24)
  -> Hash Join  (cost=3.50..108897.75 rows=20972 width=24)
     Hash Cond: ("outer".id_producto = "inner".id_producto)
     -> GroupAggregate  (cost=1.70..105245.59 rows=65536 width=16)
        -> Nested Loop  (cost=1.70..96562.07 rows=1048576 width=16)
           Join Filter: ("outer".id_producto = "inner".id_producto)
           -> Index Scan using ventas_pk on ventas p  (cost=0.00..2188.52
rows=65536 width=12)
           -> Materialize  (cost=1.70..2.34 rows=64 width=8)
              -> Seq Scan on costos c  (cost=0.00..1.64 rows=64 width=8)
     -> Hash  (cost=1.64..1.64 rows=64 width=8)
        -> Seq Scan on partes v  (cost=0.00..1.64 rows=64 width=8)
```

Para el caso de la vista C los resultados son:

```
Hash Join  (cost=4.18..2124.58 rows=65536 width=28)
  Hash Cond: ("outer".id_producto = "inner".id_producto)
  -> Seq Scan on ventas p  (cost=0.00..1137.36 rows=65536 width=12)
  -> Hash  (cost=4.17..4.17 rows=4 width=24)
     -> Hash Join  (cost=4.02..4.17 rows=4 width=24)
        Hash Cond: ("outer".id_producto = "inner".id_producto)
        -> HashAggregate  (cost=1.96..2.01 rows=4 width=8)
           -> Seq Scan on costos  (cost=0.00..1.64 rows=64 width=8)
        -> Hash  (cost=2.05..2.05 rows=4 width=12)
           -> Subquery Scan v  (cost=1.96..2.05 rows=4 width=12)
              -> HashAggregate  (cost=1.96..2.01 rows=4 width=8)
                 -> Seq Scan on partes  (cost=0.00..1.64 rows=64 width=8)
```

Los resultados sobre PostgreSQL reafirman las conclusiones obtenidas para el caso de Oracle, los valores asignados a los costos no son comparables pero los resultados demuestran la conveniencia de usar la vista C.

♦

El Ejemplo 24 muestra que existe una diferencia de costos significativa a favor de la nueva propuesta frente a lo obtenido por el enfoque Naive. Además del análisis anterior, observando los cambios realizados por la nueva propuesta, existe otro punto relevante a comentar. Éste es el de los cubos recursivos tratado en la sección 4.2.2. Éste no se analiza más allá de unos comentarios, porque ya nos es suficiente con lo anterior para justificar una mejora de rendimiento.

Además del Ejemplo 24, existe la posibilidad de pensar en una mejora de rendimiento a raíz de las consideraciones de la sección 4.2.2, dado que ésta propone eliminar dos pasos del algoritmo (13 y 14) y en su lugar reorganizar las correspondencias antes de comenzar, para tratar todos los cubos con los pasos 7 al 12. La oportunidad de mejora aparece por el lado de que, un cubo recursivo, es una secuencia de primitivas más larga en el enfoque Naive, que la de un cubo base, por la simple razón de que como prefijo tiene la cadena de primitivas que construye el cubo base (ver sección 3.3.2).

Al tratarse de una cadena más larga y de intereses contrapuestos, dado que como son cubos distintos cada uno puja por obtener un conjunto de datos diferente, algunas de las operaciones tienden a anularse. Un Drill-Up total, que elimina una dimensión del cubo base tratando de llegar al cubo recursivo, significa que todos los esfuerzos para agregar esta dimensión al cubo base (join's, agregación de más tuplas, etc.) fueron en vano, pues finalmente no va a ser utilizada.

Estas anulaciones no son más que simplificaciones al analizar el plan de ejecución de la vista final, sin embargo no está claro que un optimizador pueda realizarlas, por la complejidad de las vistas obtenidas (secuencia de vistas SPJG). Recordemos que los argumentos a este respecto que se utilizaron para realizar la nueva propuesta no consideraban obtener vistas equivalentes con uno u otro mecanismo, sino que se basaban en el hecho de que trabajando puramente a nivel de correspondencias, el resultado debía ser equivalente (ver sección 4.2.2).

5 Conclusiones

Esta tesis presenta una solución para la carga y actualización de un DW desarrollado con el enfoque de [Per01]. En dicho trabajo el problema de carga no se aborda, por lo cual, una vez obtenido el esquema del DW el diseñador se enfrenta al desafío de cargar los datos en éste. El problema en el que se concentra esta tesis es la transformación de los datos, desde la BDF hasta el DW, pero existen una serie de problemas relacionados que no son tratados: correctitud de la especificación, calidad de los datos, optimización del proceso, posibilidad de ejecución paralela, etc.

El capítulo 2 analiza los diversos enfoques sobre transformación de datos encontrados en la literatura. De éste podemos concluir que existen propuestas orientadas a SQL, orientadas a correspondencias y otras que definen un conjunto de operaciones para representar la transformación. Las soluciones basadas en SQL presentan ventajas relacionadas con la literatura disponible y el conocimiento que en general se tiene sobre esta área, sin embargo pueden necesitar ser extendidas para realizar operaciones de transformación complejas. Las soluciones basadas en operaciones plantean construcciones típicas de la tarea en la cual se concentran, por lo cual puede considerarse que expresan mejor la transformación realizada. Finalmente las orientadas a correspondencias expresan muy poco sobre cómo se lleva a cabo la transformación; las operaciones o el SQL podrían utilizarse como la especificación de la transformación a realizar por las correspondencias[23].

En este trabajo se considera que es mejor contar con una solución de carga basada en SQL [TLS99], por las ventajas que ofrece, porque las primitivas pueden ser transformadas a SQL y además por considerar que una solución propietaria no ofrece ventajas tangibles. Si este trabajo hubiera seguido la línea de elaborar una solución propietaria, tendría que haber abierto una serie de líneas de trabajo (ejecución, optimización, etc.) con pocas garantías de obtener una solución interesante. Hay que recordar que la naturaleza y propósito de las primitivas es la de representar las tareas típicas de construcción de un DW desde el punto de vista lógico, que mucho dista del problema de cargar los datos.

El capítulo 3 analiza la alternativa considerada más simple para resolver el problema de carga, el enfoque Naive. Este enfoque considera la sentencia SQL relacionada con cada primitiva y se limita a proponer como solución la secuencia de vistas asociada a la secuencia de primitivas. El capítulo muestra cómo se desarrolla la aplicación del algoritmo y qué trazas se generan para cada caso. La Figura 24 muestra la secuencia de vistas y la Tabla 3 resume la asociación entre el paso del algoritmo, la primitiva y la vista relacionada. De la tabla y la figura se observa que no todas las primitivas de [Mar00] son utilizadas y que [Per01] introduce nuevas. También a lo largo del capítulo se observan casos de borde (Traza Vacía) y una serie de posibles errores, no atribuibles al diseño sino a la forma de ejecución de la carga que propone el enfoque Naive.

---

[23] Notar que este enfoque es el utilizado por [Per01]. Las correspondencias (en su caso DirectME, 1CalcME, etc.) son una definición de más alto nivel sobre el resultado deseado. Las primitivas son la descripción concreta sobre la transformación a realizar.

El capítulo 4 toma el análisis del capítulo 3 y establece la necesidad de hacer una nueva propuesta. El enfoque Naive resulta poco claro con respecto a la transformación que realiza (cantidad y complejidad de las vistas generadas), e incurre en ciertos errores por la forma en la que se manipulan los datos, no haciendo nada al respecto. Además hay que notar que también es posible optimizar el proceso de formas que no están al alcance de un optimizador SQL. La nueva solución analizada en este capítulo propone realizar una vista unificada para cada fragmento de dimensión o franja de cubo, que resume todas las transformaciones de la traza. Para esto opera a nivel de álgebra relacional con las vistas del enfoque Naive y luego mediante simplificaciones y transformaciones se obtiene la vista unificada.

La solución propuesta por esta tesis queda definida por las vistas unificadas. Este trabajo utiliza el enfoque propuesto por Theodoratos [TLS99], donde el problema de construcción del DW consiste en la definición de dichas vistas. La implementación concreta del DW está fuera del alcance de este trabajo. Una posible estrategia de implementación puede ser la materialización de las vistas.

El proceso de análisis llevado a cabo al comienzo del capítulo 4 permite identificar qué aporte realiza cada primitiva en la vista unificada. De esta forma es posible en la sección 4.4 reescribir un algoritmo de construcción de la vista unificada, pero planteado ahora en función de las correspondencias y lineamientos, en lugar de procesar la traza de primitivas (enfoque Naive). Esto lleva a afirmar que es posible definir la solución de carga sin el uso de la traza de primitivas, lo cual implica que la vista unificada está definida únicamente por los datos de entrada que fueron usados en [Per01] para generar el esquema.

En el capítulo 4 también se demuestra que la propuesta no incurrirá en los problemas encontrados en el enfoque Naive, pues se toman acciones al respecto, incluyendo el pedido de que el problema sea replanteado si no hay alternativas. También se demuestra que la vista unificada presenta un mejor desempeño que las vistas del enfoque Naive, pues en condiciones en las cuales un optimizador no podría hacer nada al respecto, un replanteo en la vista asociada a las correspondencias NCalcME realizado por la nueva propuesta desemboca en una importante mejora de desempeño.

En consonancia con lo antes presentado, podemos decir que la propuesta de carga de este trabajo ofrece una solución abierta y estándar, porque se basa en SQL, fácil de utilizar, porque no exige más información que la ya brindada, robusta y con mejor desempeño que una solución Naive.

La aplicación de la propuesta de carga puede ser observada en el apéndice E. En el marco de esta tesis también se desarrolló un prototipo con el que se obtuvieron las vistas presentadas en dicho apéndice.

5.1 Trabajos Futuros

Durante la elaboración de esta tesis se identificaron una serie de líneas de trabajo que no pudieron ser analizadas, pero que pueden resultar interesantes para futuros trabajos en el área.

Consistencia de la especificación. Este trabajo dividió los errores en dos categorías, los considerados errores de diseño (claves primarias incorrectas, claves foráneas que no se cumplen, diferencias de tipos, valores nulos, etc.) y los errores introducidos por el proceso de carga. En particular este trabajo identificó y trató los problemas introducidos

por el proceso de carga, pero resultaría valioso poder validar y tratar los errores en el diseño. Recordar que la transformación está gobernada por las correspondencias y no es sencillo establecer que dicha transformación genera esquemas e instancias válidas.

Actualización del DW. En esta tesis se analiza principalmente el problema de la carga inicial del DW, manejando un enfoque de vistas materializadas como soporte para la actualización y mantenimiento de éste. Como ya hemos mencionado existen propuestas que describen esta solución como muy simplista para el problema de actualización [BFM99][VSS02a][VGD99]. Resulta interesante analizar si con la información disponible es posible utilizar alguno de los trabajos en el área para dar una mejor alternativa a este problema.

Retoma de procesos de carga interrumpidos. Si una carga falla, un enfoque posible es re-hacer toda la carga desde el comienzo. Tomando en cuenta los volúmenes de información que se manejan en el contexto del Data Warehousing, una mejor alternativa es retomar la carga desde el punto donde ésta se interrumpió. Partiendo del trabajo de Garcia-Molina sobre este problema [LWGG00], un desafío interesante es aplicarlo a la solución presentada en este trabajo.

Criterios de calidad de la solución. En [Per01] se menciona el problema de la calidad del esquema obtenido haciendo referencia a aspectos que son propios del diseño de éste. El mismo análisis se puede realizar en referencia a la calidad de los datos, manejando aspectos como la frescura de la información, la calidad de las transformaciones aplicadas, performance, consistencia de los datos, etc. [TB99][JV97]. Esta información podría incluso ser analizada sobre la traza, tal y como se desarrolló este trabajo, arrojando cotas o intervalos para los valores de calidad antes de tener que calcular la vista.

Evolución del proceso de carga. Marotta, en [Mar00], trata el problema de la evolución del esquema fuente y el impacto que éste crea en el diseño del DW. En este sentido valora la utilidad de haber realizado el diseño en función de primitivas de transformación. Sería interesante analizar cómo dicha evolución del esquema fuente puede ser acompañada desde el aspecto de la carga.

Versionado y manejo de la historia. Dos problemas mencionados pero no atacados en esta tesis son el del versionado y el manejo de la historia. Si bien se utilizaron primitivas que trabajan sobre este aspecto (P3 en la sección 3.2.3.1 y P4.1 en 3.2.3.2) se entiende que el tema merece ser ampliado. Este trabajo deja en las funciones asociadas a las correspondencias la responsabilidad del versionado y el manejo de la historia, lo cual trata de no comprometer ninguna posible decisión y deja abierta la puerta a futuros trabajos en esta línea.

Interconectar la solución con el DWDesigner. En el contexto de esta tesis se desarrolló un prototipo con el fin de demostrar la factibilidad de la propuesta. No se planteó como objetivo incorporarlo a la herramienta DWDesigner, sin embargo resultaría interesante hacerlo en el futuro.

Bibliografía

A continuación se presenta la bibliografía utilizada para redactar este trabajo. En todos los casos en que es posible se adjunta a la referencia un link, con el propósito de facilitar su localización.

[ASS03] Alberto Abelló, José Samos, Félix Saltor: "*Implementing Operations to Navigate Semantic Star Schemas*". Proceedings of the 6th ACM international workshop on Data warehousing and OLAP (DOLAP), 2003, pages 56 - 62. (http://portal.acm.org/citation.cfm?id=956071)

[BFM99] Mokrane Bouzeghoub, Françoise Fabret, Maja Matulovic-Broqué: "*Modeling the Data Warehouse Refreshment Process as a Workflow Application*". Proceedings of the Intl. Workshop on Design and Management of Data Warehouses, DMDW'99, Heidelberg, Germany, June 14-15, 1999. (http://www.informatik.uni-trier.de/~ley/db/conf/dmdw/dmdw1999.html)

[Car00] Fernando Carpani: *"CMDM: Un Modelo Conceptual para la Especificación de Bases Multidimensionales"*. Tesis de Maestría, Universidad de la Republica, Uruguay, 2000. (http://www.fing.edu.uy/inco/grupos/csi/)

[CD97] Surajit Chaudhuri, Umeshwar Dayal: "*An Overview of Data Warehousing and OLAP Technology*". ACM SIGMOD Record, vol. 26 (1), pages 65-74, 1997 (http://www.informatik.uni-trier.de/~ley/db/journals/sigmod/ChaudhuriD97.html)

[Cod93] E. F. Codd.: "*Providing OLAP to user-analysts: An IT mandate*". E.F. Codd and Associates, 1993. (http://dev.hyperion.com/resource_library/white_papers/)

[CR02] Kajal T. Claypool, Elke A. Rundensteiner: "*Sangam: A Transformation Modeling Framework*". Technical Report TR02-8, Computer Science Department, UMass-Lowell. (http://www.cs.uml.edu/~kajal/research/pubs/sangam-tr02-8/index.html)

[CR03] Kajal T. Claypool, Elke A. Rundensteiner: "*Sangam: A Transformation Modeling Framework*". Eighth International Conference on Database Systems for Advanced Applications (DASFAA '03), March 26-28, 2003, Kyoto, Japan. (http://citeseer.ist.psu.edu/606751.html)

[EN97] Ramez Elmasri, Shamkant B. Navathe: "*Sistemas de Bases de Datos, Conceptos Fundamentales*". ISBN: 0-201-65370-2, Addison-Wesley Iberoamericana. 1997.

[Gar98] Stephen R. Gardner: "*Building the Data Warehouse*". Communications of the ACM 41, 9 (1998), 52-60. (http://www.informatik.uni-trier.de/~ley/db/journals/cacm/Gardner98.html)

[GCBL97] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, Hamid Pirahesh: "*Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals*". Journal of Data Mining and Knowledge Discovery, pages 29--53. 1997. (http://citeseer.csail.mit.edu/gray97data.html)

[GFSS00] Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon: "*Declaratively cleaning your data using AJAX*". In: Journ. Bases de Donn ees Avanc ees, Oct. 2000. (http://citeseer.ist.psu.edu/314874.html)

[GFSS00a] Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon: "*An Extensible Framework for Data Cleaning*". In Proceedings of the International Conference on Data Engineering (ICDE), San Diego, CA, page 312, 2000. (http://citeseer.ist.psu.edu/galhardas00extensible.html)

[GMR98] Matteo Golfarelli, Dario Maio, Stefano Rizzi: "*Conceptual Design of Data Warehouses from E/R Schemes*". In Proc. Hawaii Int. Conf. on System Sciences, vol. VII, Kona, Hawaii, pp. 334-343, 1998. (http://citeseer.ist.psu.edu/golfarelli98conceptual.html)

[GR00] Matteo Golgarelli, Stefano Rizzi: "*View Materialization for Nested GPSJ Queries*". Proceedings of the International Workshop on Design and Management of Data Warehouses, DMDW'2000, Stockholm, Sweden, June 2000. (http://www.informatik.uni-trier.de/~ley/db/conf/dmdw/dmdw2000.html)

[Gup97] Himanshu Gupta: "*Selection of Views to Materialize in a Data Warehouse*". In Proc. Sixth ICDT, pages 98--112, Delphi, Jan. 1997. (http://citeseer.ist.psu.edu/gupta97selection.html)

[GUW00] Hector Garcia-Molina; Jeff Ullman; Jennifer Widom: *"Database System Implementation"*. Prentice Hall, ISBN: 0-13-040264-8.

[HLV00] Bodo Husemann, Jens Lechtenborger, Gottfried Vossen: "*Conceptual Data Warehouse Design*". In Proceedings of International Workshop on Design and Management of Data Warehouses, Stockholm, 2000. (http://citeseer.csail.mit.edu/husemann00conceptual.html)

[HRU96] Venky Harinarayan, Anand Rajaraman, Jeffrey D. Ullman: "*Implementing data cubes efficiently*". In Proc. ACM SIGMOD '96, pages 205--216, Montreal, June 1996. (http://citeseer.ist.psu.edu/harinarayan96implementing.html)

[Inm96] William H. Inmon: "*Building the Data Warehouse*", second edition. ISBN: 0471141615, John Wiley & Sons, 1996.

[JV97] Matthias Jarke, Yannis Vassiliou: "*Data Warehouse Quality: A Review of the DWQ Project*". In Proc. of the 2nd Intl. Conf. on Information Quality Cambridge, Mass., pages 98--112, 1997. (http://citeseer.ist.psu.edu/jarke97data.html)

[Kim96] Ralph Kimbal: "*The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*". ISBN: 0471153370 , John Wiley & Sons, 1996.

[LS97] Hans-J. Lenz, Arie Shoshani: "*Summarizability in OLAP and Statistical Data Bases*". In 9th Int. Conf. on Scientific and Statistical Database Management (SSDBM), pages 132-143. IEEE Press, 1997. (http://citeseer.ist.psu.edu/lenz97summarizability.html)

[LSS96] Laks V. S. Lakshmanan, Fereidoon Sadri, Iyer N. Subramanian: "*SchemaSQL A Language for Interoperability in Relational Multi-database Systems*". Proc. 22nd VLDB, pp. 239-250, Mumbai, India, 1996. (http://citeseer.ist.psu.edu/lakshmanan96schemasql.html)

[LWGG00] Wilburt Juan Labio, Janet Weiner, Hector Garcia-Molina, Vlad Gorelik: "*Efficient resumption of interrupted warehouse loads*". In Proc. of the ACM SIGMOD International Conference on Management of Data, pages 46--57, Dallas, Texas, May 2000. (http://citeseer.ist.psu.edu/107124.html)

[Mar00] Adriana Marotta: *"Data Warehouse Design and Maintenance through Schema Transformations"*, Tesis de Maestría, Universidad de la Republica, Uruguay, 2000. (http://www.fing.edu.uy/inco/grupos/csi/)

[MR02] Adriana Marotta, Raul Ruggia: "*Data Warehouse Design: A schema-transformation approach*", International Conference of the Chilean Computing Society, SCCC 2002, Chile, November 2002. (http://ftp.informatik.uni-trier.de/~ley/db/conf/sccc/sccc2002.html)

[Per01] Veronika Peralta: *"Diseño Lógico de Data Warehouses a partir de Esquemas Conceptuales Multidimensionales"*. Tesis de Maestría, Universidad de la Republica, Uruguay, 2001. (http://www.fing.edu.uy/~vperalta)

[RA05] Oscar Romero, Alberto Abelló: "*Improving automatic SQL translation for ROLAP tools*". In Proceedings of Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2005). Granada (Spain), September 2005. Pages 123-130. Thomson Editores, ISBN 84-9732-434-X. (http://www.lsi.upc.es/~aabello/publications/home.html)

[RD00] Erhard Rahm, Hong Hai Do: "*Data Cleaning: Problems and Current Approaches*". IEEE Bulletin of the Technical Committee on Data Engineering, Vol 23 No. 4, December 2000. (http://lips.informatik.uni-leipzig.de/pub/2000-45/en)

[RH01] Vijayshankar Raman, Joseph M. Hellerstein: "*Potter's Wheel: An Interactive Data Cleaning System*". The VLDB Journal, pages 381-390, 2001. (http://citeseer.ist.psu.edu/raman01potters.html)

[Sho97] Arie Shoshani: "*OLAP and Statistical Databases: Similarities and Differences*". In Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pages 185-196, Tuscon, Arizona, May 1997. (http://citeseer.ist.psu.edu/shoshani97olap.html)

[TB99] Dimitri Theodoratos, Mokrane Bouzeghoub: "*Data Currency Quality Factors in Data Warehouse Design*". Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'99), Heidelberg, Germany, June 1999. (http://citeseer.ist.psu.edu/theodoratos99data.html)

[TLS99] Dimitri Theodoratos, Spyros Ligoudistianos, Timos K. Sellis: "*Designing the Global Data Warehouse with SPJ Views*". In Proc. of the 11th Conf. on Advanced Information Systems Engineering (CAiSE'99), 1999. (http://citeseer.ist.psu.edu/theodoratos99designing.html)

[TS99] Dimitri Theodoratos, Timos K. Sellis: *"Designing Data Warehouses"*. Journal Data Knowledge Engineering, volumen 31, número 3, páginas 279-301, 1999. (http://citeseer.ist.psu.edu/265105.html)

[Vas00] Panos Vassiliadis: "*Gulliver in the land of data warehousing: practical experiences and observations of a researcher*". In Proc. 2 nd Intl. Workshop on Design and Management of Data Warehouses (DMDW), Sweden, 2000. (http://www.informatik.uni-trier.de/~ley/db/conf/dmdw/dmdw2000.html)

[VGD99] Athanasios Vavouras, Stella Gatziu, Klaus R. Dittrich: "*Modeling and Executing the Data Warehouse Refreshment Process*". International Symposium on Database Applications in Non-Traditional Environments (DANTE'99), p. 66, 1999. (http://doi.ieeecomputersociety.org/10.1109/DANTE.1999.844943)

[VGD99a] Athanasios Vavouras, Stella Gatziu, Klaus R. Dittrich: "*The SIRIUS Approach for Refreshing Data Warehouses Incrementally*". Datenbanksysteme in Buro, Technik und Wissenschaft (BTW), GI-Fachtagung, Freiburg, pages 80-96, 1999. (http://www.informatik.uni-trier.de/~ley/db/conf/btw/btw99.html)

[VGD00] Athanasios Vavouras, Stella Gatziu, Klaus R. Dittrich: "*Modeling and Executing the Data Warehouse Refreshment Process*" (Technical Report). IFI-2000.01. 2000. (http://citeseer.ist.psu.edu/vavouras00modeling.html, http://www.ifi.unizh.ch/techreports/TR_2000.html)

[VSS02] Panos Vassiliadis, Alkis Simitsis, Spiros Skiadopoulos: *"Modeling ETL Activities as Graphs"*. Proceedings of the 4th Intl. Workshop DMDW'2002, Toronto, Canada, May 2002. (http://www.sigmod.org/dblp/db/conf/dmdw/dmdw2002.html#VassiliadisSS02)

[VSS02a] Panos Vassiliadis, Alkis Simitsis, Spiros Skiadopoulos: *"Conceptual modeling for ETL processes"*, Proceedings of DOLAP, 2002. (http://citeseer.ist.psu.edu/vassiliadis02conceptual.html)

[VVSKS00] Panos Vassiliadis, Zografoula Vagena, Spiros Skiadopoulos, Nikos Karayannidis, Timos Sellis: "*ARKTOS: A Tool For Data Cleaning and Transformation in Data Warehouse Environments*". Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, vol. 23, no. 4, pp. 42-47, December 2000. (http://www.informatik.uni-trier.de/~ley/db/journals/debu/debu23.html)

[Wid95] Jennifer Widom: "*Research Problems in Data Warehousing*". 4th International Conference on Information and Knowledge Management, Baltimore, Maryland, pages 25-30, 1995. (http://citeseer.ist.psu.edu/widom95research.html)

[WB97] Ming-Chuan Wu, Alejandro P. Buchmann: "Research Issues in Data Warehousing". Datenbanksysteme in Buro, Technik und Wissenschaft (BTW), pages 61-82, 1997. (http://citeseer.ist.psu.edu/wu97research.html)

[YL94] Weipeng P. Yan and Per-Åke Larson: *"Performing Group-By before Join"*. Proceedings of the 10th IEEE International Conference on Data Engineering, pages 89-100, Huston, Texas, Feb. 1994. (http://citeseer.ist.psu.edu/yan94performing.html)

[YL95] Weipeng P. Yan and Per-Åke Larson: *"Eager Aggregation and Lazy Aggregation"*. The VLDB Journal, pages 345-357, 1995. (http://citeseer.ist.psu.edu/yan95eager.html)

A Algoritmo

Este apéndice contiene el algoritmo presentado en [Per01] dado que es analizado en profundidad tanto en el capítulo 3 para elaborar la solución Naive, como en el capítulo 4 para la nueva propuesta.

A.1 Paso 1 - Construir los esqueletos

Para cada fragmento que posea correspondencias a más de una tabla, aplicar la regla R1 (Join). Se seleccionan dos de las tablas a las que el fragmento tenga correspondencias y que estén relacionadas (tienen definido un link). Se itera hasta que el fragmento tenga correspondencias a una única tabla (esqueleto).

```
∀ F ∈ SchFragments /* para cada fragmento */
   Sea <Map,Cond> = SchFMappings(F) /* el mapeo del fragmento */
   Sea Ts = MapTables(Map, ObjectItems(F)) /* tablas que mapean al fragmento */
   Mientras #Ts > 1
      Sean T1, T2 ∈ Ts, T1 ≠ T2, SchLinks(T1,T2) ≠ ⊥ /* dos tablas relacionadas */
      Aplicar Join (F, Map, T1, T2) /* genera una nueva tabla, y actualiza la función de
      mapeo */
      Sea Ts = MapTables(Map, ObjectItems(F)) /* recalcula tablas que mapean al
      fragmento */
   Fin /* mientras */
Fin /* para todo fragmento */
```

A.2 Paso 2 - Renombrar atributos para ítems con mapeo directo

Para cada fragmento, que tenga ítems con correspondencia directa (DirectME), cuyos nombres difieran de los nombres de los atributos que los tienen correspondencia, aplicar la regla R2 (Rename).

```
∀ F ∈ SchFragments /* para cada fragmento */
   Sea <Map,Cond> = SchFMappings(F) /* el mapeo del fragmento */
   Sea T ∈ MapTables(Map, ObjectItems(F)) /* única tabla que mapea al fragmento */
   Aplicar Rename (F, Map, T) /* genera una nueva tabla, y actualiza la función de mapeo */
Fin /* para todo fragmento*/
```

A.3 Paso 3 - Generar atributos para ítems con mapeo calculado o externo

Para cada fragmento, aplicar: la regla R3 (Calculate, en alguna de sus versiones) para cada ítem con correspondencia calculada, y la regla R4 (Extern, en alguna de sus versiones) para cada ítem con correspondencia externa.

```
∀ F ∈ SchFragments /* para cada fragmento */
   Sea <Map,Cond> = SchFMappings(F) /* el mapeo del fragmento */
   ∀ I ∈ ObjectItems(F)
      Sea T ∈ MapTables(Map, ObjectItems(F)) /* única tabla que mapea al fragmento
      */
      Si Map(I) ∈ 1calcME /* el ítem tiene mapeo de cálculo simple */
       Aplicar Simple-Calculate (F, I, Map, T)
      Si Map(I) ∈ NcalcME /* el ítem tiene mapeo de cálculo de resumen */
       Aplicar Aggregate-Calculate (F, I, Map, T)
      Si Map(I) ∈ ExternME ∧Map(I).Ind = Constant /* el ítem tiene mapeo constante
      */
       Aplicar Constant-Extern-Value (F, I, Map, T)
      Si Map(I) ∈ ExternME ∧Map(I).Ind = Timestamp /* el ítem tiene mapeo de
      marca de tiempo*/
       Aplicar Timestamp-Extern-Value (F, I, Map, T)
      Si Map(I) ∈ ExternME ∧Map(I).Ind = Version /* el ítem tiene mapeo de dígitos
      de versión */
       Aplicar Version-Extern-Value (F, I, Map, T)
   Fin /* para todo ítem */
Fin /* para todo fragmento */
```

A.4 Paso 4 - Aplicar filtros

Para cada fragmento, que tenga condiciones, aplicar la regla R8 (Filter).

```
∀ F ∈ SchFragments /* para cada fragmento */
   Sea <Map,Cond> = SchFMappings(F) /* el mapeo del fragmento */
   Sea T ∈ MapTables(Map, ObjectItems(F)) /* única tabla que mapea al fragmento */
   Aplicar Filter (F, Cond, Map, T)
Fin /* para todo fragmento */
```

A.5 Paso 5 - Eliminar atributos sin correspondencia

Para cada fragmento, cuyo esqueleto tenga atributos que no poseen correspondencia a ningún ítem, aplicar la regla R5 (Group).

```
∀ F ∈ SchFragments /* para cada fragmento */
   Sea <Map,Cond> = SchFMappings(F) /* el mapeo del fragmento */
   Sea T ∈ MapTables(Map, ObjectItems(F)) /* única tabla que mapea al fragmento */
   Aplicar Fragment-Group (F, Map, T) /* genera una nueva tabla, y actualiza la función
   de mapeo */
Fin /* para todo fragmento */
```

A.6 Paso 6 - Ajustar las claves

Para cada fragmento, cuyo clave difiera de la clave de su esqueleto, aplicar la regla R7 (Primary-Key).

```
∀ F ∈ SchFragments /* para cada fragmento */
   Sea <Map,Cond> = SchFMappings(F) /* el mapeo del fragmento */
   Sea T ∈ MapTables(Map, ObjectItems(F)) /* única tabla que mapea al fragmento */
   Aplicar Primary-Key (F, Map, T) /* genera una nueva tabla, y actualiza la función de
   mapeo */
Fin /* para todo fragmento */
```

A.7 Paso 7 - Construir los esqueletos

Para cada cubo con mapeo base, que tenga correspondencias a más de una tabla, aplicar la regla R1 (Join), a dos de las tablas a las cuales tiene correspondencias y están relacionadas (tienen definido un link). Iterar hasta que el cubo tenga correspondencias a una única tabla (esqueleto).

```
∀ C ∈ SchCubes / SchCMappings(C) ∈ BaseCMappings /* para cada cubo con mapeo
base */
   Sea <Map,Cond,Rup> = SchCMappings(C) /* el mapeo del cubo */
   Sea Ts = MapTables(Map, ObjectItems(C)) /* tablas que mapean al cubo */
   Mientras #Ts > 1
      Sean T1, T2 ∈ Ts, T1 ≠ T2, SchLinks(T1,T2) ≠ ⊥ /* dos tablas relacionadas */
      Aplicar Join (C, Map, T1, T2) /* genera una nueva tabla, y actualiza la función de
      mapeo */
      Sea Ts = MapTables(Map, ObjectItems(C)) /* recalcula tablas que mapean al
      cubo */
   Fin /* mientras */
Fin /* para todo cubo */
```

A.8 Paso 8 - Renombrar atributos para ítems con mapeo directo

Para cada cubo con mapeo base, que tenga ítems con mapeo directo, cuyos nombres difieran de los nombres de los atributos que los mapean, aplicar la regla R2 (Rename).

```
∀ C ∈ SchCubes / SchCMappings(C) ∈ BaseCMappings /* para cada cubo con mapeo
base */
   Sea <Map,Cond,Rup> = SchCMappings(C) /* el mapeo del cubo */
   Sea T ∈ MapTables(Map, ObjectItems(C)) /* única tabla que mapea al cubo */
   Aplicar Rename (C, Map, T) /* genera una nueva tabla, y actualiza la función de mapeo */
Fin /* para todo cubo */
```

A.9 Paso 9 - Generar atributos para ítems con mapeo calculado o externo

Para cada cubo con mapeo base, aplicar: la regla R3 (Calculate, en alguna de sus versiones) para cada ítem con correspondencia calculada, y la regla R4 (Extern, en alguna de sus versiones) para cada ítem con correspondencia externa.

```
∀ C ∈ SchCubes / SchCMappings(C) ∈ BaseCMappings /* para cada cubo con mapeo
base */
   Sea <Map,Cond,Rup> = SchCMappings(C) /* el mapeo del cubo */
   ∀ I ∈ ObjectItems(C)
      Sea T ∈ MapTables(Map, ObjectItems(C)) /* única tabla que mapea al cubo */
      Si Map(I) ∈ 1calcME /* el ítem tiene mapeo de cálculo simple */
        Aplicar Simple-Calculate (C, I, Map, T)
      Si Map(I) ∈ NcalcME /* el ítem tiene mapeo de cálculo de resumen */
        Aplicar Aggregate-Calculate (C, I, Map, T)
      Si Map(I) ∈ ExternME ∧Map(I).Ind = Constant /* el ítem tiene mapeo constante
      */
        Aplicar Constant-Extern-Value (C, I, Map, T)
      Si Map(I) ∈ ExternME ∧Map(I).Ind = Timestamp /* el ítem tiene mapeo de
      marca de tiempo */
        Aplicar Timestamp-Extern-Value (C, I, Map, T)
      Si Map(I) ∈ ExternME ∧Map(I).Ind = Version /* el ítem tiene mapeo de dígitos
      de versión */
        Aplicar Version-Extern-Value (C, I, Map, T)
   Fin /* para todo ítem */
Fin /* para todo cubo */
```

A.10 Paso 10 - Aplicar filtros

Para cada cubo con mapeo base, que tenga condiciones, aplicar la regla R8 (Filter).

```
∀ C ∈ SchCubes / SchCMappings(C) ∈ BaseCMappings /* para cada cubo con mapeo
base */
   Sea <Map,Cond,Rup> = SchCMappings(C) /* el mapeo del cubo */
   Sea T ∈ MapTables(Map, ObjectItems(C)) /* única tabla que mapea al cubo */
   Aplicar Filter (C, Cond, Map, T)
Fin /* para todo cubo */
```

A.11 Paso 11 - Eliminar atributos sin correspondencia

Para cada cubo con mapeo base, cuyo esqueleto tenga atributos que no tienen correspondencia a ningún ítem, aplicar la regla R5 (Group).

```
∀ C ∈ SchCubes / SchCMappings(C) ∈ BaseCMappings /* para cada cubo con mapeo
base */
   Sea <Map,Cond,Rup> = SchCMappings(C) /* el mapeo del cubo */
   Sea T ∈ MapTables(Map, ObjectItems(C)) /* única tabla que mapea al cubo */
   Aplicar Cube-Group (C, Rup, Map, T) /* genera una nueva tabla, y actualiza la
   función de mapeo */
Fin /* para todo cubo */
```

A.12 Paso 12 - Ajustar las claves

Para cada cubo con mapeo base, cuyo clave difiera de la clave de su esqueleto, aplicar la regla R7 (Primary-Key).

```
∀ C ∈ SchCubes / SchCMappings(C) ∈ BaseCMappings /* para cada cubo con mapeo
base */
   Sea <Map,Cond,Rup> = SchCMappings(C) /* el mapeo del cubo */
   Sea T ∈ MapTables(Map, ObjectItems(C)) /* única tabla que mapea al cubo */
   Aplicar Primary-Key (C, Map, T) /* genera una nueva tabla, y actualiza la func. de
   mapeo */
Fin /* para todo cubo */
```

A.13 Paso 13 - Armar la tabla de la Jerarquía

Para cada Drill-Up de cada cubo con mapeo recursivo, que implique reducción de detalle de una dimensión, definir un fragmento ficticio con los niveles del Drill-Up, aplicarle los pasos 1 a 6.

```
∀ C ∈ SchCubes / SchCMappings(C) ∈ RecursiveCMappings /* para c/ cubo con mapeo
recursivo */
   Sea <Cub,Dups,Map> = SchCMappings(C) /* el mapeo del cubo */
   ∀ Dup ∈ Dups /* para cada drill-up del mapeo del cubo */
      Si Dup ∈ DetailDrillUps(Cbase,C) /* si tiene detalle para la dimensión */
         /*Se construye un fragmento ficticio que tenga los niveles Lbase y Lnews */
         Sea FR = {Dup.Lbase} ∪ Dup.Lnews /* fragmento ficticio con los niveles
         involucrados */
         Se define SchFMappings(FR) = <Dup.Map, ⊥ > /* se asocia la función de
         mapeo al fragmento ficticio */
         Ejecutar Step 1 a Step 6 para FR. /* se construye el esqueleto del fragmento
         ficticio */
         Dup.Map = SchFMappings(FR) /* se actualiza el mapeo del drill-up */
      Fin /* si */
   Fin /* para todo drill-up */
Fin /* para todo cubo */
```

A.14 Paso 14 - Aplicar los Drill-Ups

Para cada Drill-Up de cada cubo con mapeo recursivo, aplicar la regla R6 (Drill-Up).

```
Mientras ∃ C ∈ SchCubes . (SchCMappings(C) ∈ RecursiveCMappings
∧ SchCMappings(C).Cub ∈ BaseCMappings
   /* un cubo con mapeo recursivo, en función de un cubo con mapeo base */
   Sea <Cbase,Dups,Map> = SchCMappings(C) /* el mapeo del cubo */
   Si SchCMappings(Cbase) ∈ BaseCMapping(Cbase) /* el cubo base tiene mapeo base */
      Sea T ∈ MapTables(Map, ObjectItems(Cbase)) /* única tabla que mapea al cubo
      base */
      ∀ Dup ∈ Dups /* para cada drill-up del mapeo del cubo */
         Si Dup ∈ NoDetailDrillUps(Cbase,C)
            /* si se elimina la dimensión del detalle del cubo */
            Aplicar Total-Drill-Up (Cbase,C, Dup, Map, T)
```

```
        Sino /* si tiene detalle para la dimensión */
           Sea TF ∈ MapTables(Dup.Map, ObjectItems(Dup.Lbase
           ∪ Dup.Lnews))
           /* única tabla que mapea a los ítems del drill up */
           Aplicar Hierarchy-Drill-Up (Cbase, C, Dup, Map, Dup.Map, T,
           TF)
        Fin /* si */
      Fin /* para todo */
      /* Ahora el cubo tiene un mapeo base */
      SchCMapping(C) ≡ <Map, ⊥, ⊥ > /* Sólo interesa la func. de mapeo para procesar las
      franjas */
   Fin /* si */
Fin /* mientras */
```

A.15 Paso 15 - Armar las tablas de las Franjas

Para cada cubo, que tenga franjas definidas, aplicar la regla R8 (Filter), con la condición de cada franja.

```
∀ C ∈ SchCubes /* para cada cubo */
   Sea Map = SchCMappings(C).Map /* el mapeo del cubo */
   ∀ F ∈ SchCFragmentation(C) /* una franja */
      Sea T ∈ MapTables(Map, ObjectItems(C)) /* única tabla que mapea al cubo */
      Aplicar Filter (C, F, Map, T)
   Fin /* para toda franja */
Fin /* para todo cubo */
```

B Reglas

En este apéndice se presentan las reglas de [Per01]. Dado que son referidas desde el algoritmo de generación del esquema (apéndice A) son necesarias para el entendimiento del documento. Por otro lado algunas han tenido que ser levemente modificadas. Cada modificación que se realiza a la versión original de las reglas se comenta puntualmente.

B.1 Regla R1 - JOIN

Dado un objeto del esquema intermedio, su función de correspondencia y dos tablas relacionadas que tienen correspondencias al objeto, se genera una nueva tabla con los atributos de ambas siguiendo el link entre ellas.

Objetos:
OS ∈ SchFragments ∪ SchCubes */* un fragmento o un cubo */*

Entrada:
Mapeos: F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
Tablas: T1, T2 ∈ MapTables(F, ObjectItems(OS)), T1 ≠ T2 */* tablas que mapean al objeto */*

Condiciones:
SchLinks(T1,T2) ≠ ⊥ */* las tablas están relacionadas */*
MapTables(F, ObjectItems(OS)) > 1 */* el objeto es mapeado por más de una tabla */*

Resultado:
Tablas:
T' = PrimitiveQ1 (*/* Relation Join */*
{T1, T2}, */* esquema fuente */*
SchLinks(T1,T2), */* función de join */*
∅ , */* atributos de la 1er tabla a eliminar */*
∅ , */* atributos de la 2da tabla a eliminar */*
{Instance(T1), Instance(T2)} */* instancias */*
)
UpdateTabs (T', {T1,T2}) */* con links a T1 y T2 por sus claves, y hereda los links de T1 y T2 */*

Mapeos:
F' = ReplaceTable (F,T1,T')
F'' = ReplaceTable (F',T2, T')
UpdateMaps (F'', OS)

B.2 Regla R2 - RENAME

Dado un objeto del esquema intermedio, su función de correspondencia y una tabla que a la que tiene correspondencia, se genera una nueva tabla renombrando sus atributos según los nombres de los ítems.

Objetos:
OS ∈ SchFragments ∪ SchCubes */* un fragmento o un cubo */*

Entrada:
Mapeos: F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
Tablas: T ∈ MapTables(F, ObjectItems(OS)) */* tabla que mapea al objeto */*

Condiciones:
Sea:
List = {<I,A> / I ∈ ObjectItems(OS) ∧ F(I) ∈ DirectME ∧ A = F(I).Expr ∧ A.AttrName ≠ I.ItemName} */* los mapeos directos a un atributo de la tabla, que no se nombran igual que el ítem */*
List ≠ ∅
MapTables(F, ObjectItems(OS)) = 1 */* el objeto es mapeado por una sola tabla */*

Resultado:
Tablas:
T' = PrimitiveQ2 (*/* Attribute Renaming */*
{T}, /* esquema fuente */
{<A.AttrName, I.ItemName> / <I,A> ∈ List}, /* <*atributos, nuevos nombres*> */
{Instance(T)} */* instancias */*
)
UpdateTabs (T', {T}) */* con links a T por su clave, y hereda los links de T */*

Mapeos:
F' = ReplaceAttributes (F, List)
F'' = ReplaceTable (F',T, T')
UpdateMaps (F'', OS)

B.3 Regla R3 - CALCULATE

B.3.1 Sub-Regla R3.1 - SIMPLE CALCULATE

Dado un objeto del esquema intermedio, uno de sus ítems, su función de correspondencia (con correspondencia 1CalcME para el ítem) y una tabla a la que tiene correspondencia, se genera una nueva tabla agregando el atributo calculado.

Objetos:
OS ∈ SchFragments ∪ SchCubes */* un fragmento o un cubo */*
I ∈ ObjectItems(OS) */* un ítem */*

Entrada:
Mapeos: F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
Tablas: T ∈ MapTables(F, ObjectItems(OS)) */* tabla que mapea al objeto */*

Condiciones:
F(I) ∈ 1CalcME */* cálculo simple */*
MapTables(F, ObjectItems(OS)) = 1 */* el objeto es mapeado por una sola tabla */*

Resultado:
Tablas:
T' = PrimitiveP6.1 (*/* DD-Adding 1-1 */*
{T}, */* esquema fuente */*
I.ItemName = F(I).Expr, */* función de cálculo */*
{Instance(T)} */* instancias */*
)
UpdateTabs (T', {T}) */* con links a T por su clave, y hereda los links de T */*

Mapeos:
F' = ReplaceMap (F, F(I), <I, T', T'•I, F(I).Cond))) */* ahora con mapeo DirectME*/*
F'' = ReplaceTable (F',T, T')
UpdateMaps (F'', OS)

B.3.2 Sub-Regla R3.2 - AGGREGATE CALCULATE

Dado un objeto del esquema intermedio, uno de sus ítems, su función de correspondencia (con correspondencia NCalcME para el ítem) y una tabla a la que tiene correspondencia, se genera una nueva tabla agregando el atributo calculado.

Objetos:
OS ∈ SchFragments ∪ SchCubes */* un fragmento o un cubo */*
I ∈ ObjectItems(OS) */* un ítem */*

Entrada:
Mapeos: F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
Tablas: T ∈ MapTables(F, ObjectItems(OS)) */* tabla que mapea al objeto */*

Condiciones:
F(I) ∈ NCalcME */* cálculo de resumen */*
MapTables(F, ObjectItems(OS)) = 1 */* el objeto es mapeado por una sola tabla */*

Resultado:
Tablas:
T' = PrimitiveP6.3 (*/* DD-Adding N-N */*
{T, F(I).Tab}, */* esquema fuente */*
I.ItemName = F(I).Expr, */* función de cálculo */*
{}, */* atributos adicionales de agrupamiento */*
TabLink (T, F(I).Tab).Cond, */* función de join */*
{Instance(T), Instance(F(I).Tab)} */* instancias */*
)

UpdateTabs (T', {T}) */* con links a T por su clave, y hereda los links de T */*

Mapeos:

F' = ReplaceMap (F, F(I), <I, T', T'•I, F(I).Cond)}) */* ahora con mapeo DirectME*/*
F'' = ReplaceTable (F',T, T')
UpdateMaps (F'', OS)

B.4 Regla R4 - EXTERN

B.4.1 Sub-Regla R4.1 - CONSTANT EXTERN VALUE

Dado un objeto del esquema intermedio, uno de sus ítems, su función de correspondencia (con correspondencia ExternME de tipo constante para el ítem) y una tabla a la que tiene correspondencia, se genera una nueva tabla agregando el atributo correspondiente al ítem.

Objetos:
OS ∈ SchFragments ∪ SchCubes */* un fragmento o un cubo */*
I ∈ ObjectItems(OS) */* un ítem */*

Entrada:
Mapeos: F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
Tablas: T ∈ MapTables(F, ObjectItems(OS)) */* tabla que mapea al objeto */*

Condiciones:
F(I) ∈ ExternME ∧ F(I).Ind = Constant */* mapeo constante */*
MapTables(F, ObjectItems(OS)) = 1 */* el objeto es mapeado por una sola tabla */*

Resultado:
Tablas:
T' = PrimitiveP7 (*/* Attribute Adding */*
{T}, */* esquema fuente */*
F(I).Expr, */* valor de la constante */*
I.ItemName, */* atributo a agregar */*
{Instance(T)} */* instancias */*
)
UpdateTabs (T', {T}) */* con links a T por su clave, y hereda los links de T */*

Mapeos:

F' = ReplaceMap (F, F(I), <I, T', T'•I, F(I).Cond)}) */* ahora con mapeo DirectME*/*
F'' = ReplaceTable (F',T, T')
UpdateMaps (F'', OS)

Nota: Esta última regla fue alterada con el fin de contemplar las modificaciones realizadas a la primitiva P7 (ver la sección C.6 de apéndice C) y las funciones de mapeo (ver la sección D.4 del apéndice D). Esto es el agregado de las funciones de cálculo para los nuevos valores o lo que es lo mismo el valor constante que cada atributo tomará. Estas funciones (o constantes) se obtienen a partir de los mapeos (F(I).Expr) como se puede ver en el planteo de la regla.

B.4.2 Sub-Regla R4.2 - TIMESTAMP EXTERN VALUE

Dado un objeto del esquema intermedio, uno de sus ítems, su función de correspondencia (con correspondencia ExternME de tipo marca de tiempo para el ítem) y una tabla a la que tiene correspondencia, se genera una nueva tabla agregando el atributo correspondiente al ítem.

Objetos:
OS ∈ SchFragments ∪ SchCubes */* un fragmento o un cubo */*
I ∈ ObjectItems(OS) */* un ítem */*

Entrada:
Mapeos: F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
Tablas: T ∈ MapTables(F, ObjectItems(OS)) */* tabla que mapea al objeto */*

Condiciones:
F(I) ∈ ExternME ∧ F(I).Ind = Timestamp */* mapeo por estampa de tiempo */*
MapTables(F, ObjectItems(OS)) = 1 */* el objeto es mapeado por una sola tabla */*

Resultados:
Tablas:
T' = PrimitiveP3 (*/* Temporalization */*
{T}, */* esquema fuente */*
I.ItemName, */* atributo de tiempo */*
true, */* será parte de la clave */*
{Instance(T)} */* instancias */*
)
UpdateTabs (T', {T}) */* con links a T por su clave, y hereda los links de T */*

Mapeos:
F' = ReplaceMap (F, F(I), <I, T', T'•I, F(I).Cond)}) */* ahora con mapeo DirectME*/*
F'' = ReplaceTable (F',T, T')
UpdateMaps (F'', OS)

B.4.3 Sub-Regla R4.3 - VERSION EXTERN VALUE

Dado un objeto del esquema intermedio, uno de sus ítems, su función de correspondencia (con correspondencia ExternME de tipo dígito de versión para el ítem) y una tabla a la que tiene correspondencia, se genera una nueva tabla agregando el atributo correspondiente al ítem.

Objetos:
OS ∈ SchFragments ∪ SchCubes */* un fragmento o un cubo */*
I ∈ ObjectItems(OS) */* un ítem */*

Entrada:
Mapeos: F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
Tablas: T ∈ MapTables(F, ObjectItems(OS)) */* tabla que mapea al objeto */*

Condiciones:
F(I) ∈ ExternME ∧ F(I).Ind = Version */* *mapeo por dígitos de versión* */*
MapTables(F, ObjectItems(OS)) = 1 */* *el objeto es mapeado por una sola tabla* */*

Resultado:
Tablas:
T' = PrimitiveP4.1 (*/* *Version Digits* */*
{T}, /* esquema fuente */
F(I).Expr.AttrName, */* *atributo versionado /*
I.ItemName, */* *atributo de versión* */*
{Instance(T)} */* *instancias* */*
)
UpdateTabs (T', {T}) */* *con links a T por su clave, y hereda los links de T* */*

Mapeos:
F' = ReplaceMap (F, F(I), <I, T', T'•I, F(I).Cond)}) */* *ahora con mapeo DirectME*/*
F'' = ReplaceTable (F',T, T')
UpdateMaps (F'', OS)

Nota: Esta última regla fue alterada con el fin de contemplar las modificaciones realizadas a la primitiva P4.1 (ver la sección C.3 del apéndice C) y las funciones de mapeo (ver la sección D.4 del apéndice D). Esto es indicar el atributo que será versionado (F(I).Expr.AttrName).

B.5 Regla R5 - GROUP

B.5.1 Sub-Regla R5.1 – FRAGMENT GROUP

Dado un fragmento, su función de correspondencia y una tabla a la que tiene correspondencia, se genera una nueva tabla eliminando los atributos sin correspondencia y agrupando por los demás.

Objetos:
OS ∈ SchFragments */* *un fragmento* */*

Entrada:
Mapeos: F ∈ Mappings(ObjectItems(OS)) */* *función de mapeo del objeto* */*
Tablas: T ∈ MapTables(F, ObjectItems(OS)) */* *tabla que mapea al objeto* */*

Condiciones:
T.As – MapAttributes(F, ObjectItems(OS)) ≠ ∅ */* *hay atributos para eliminar* */*
MapTables(F, ObjectItems(OS)) = 1 */* *el objeto es mapeado por una sola tabla* */*

Resultado:
Tablas:
T' = PrimitiveP9 (*/* *Aggregate Generation* */*
{T}, */* *esquema fuente* */*
{ }, */* *medidas* */*
{ }, */* *funciones de resumen* */*

T.As♦AttrName - MapAttributes(F, ObjectItems(OS))♦AttrName */*atr. a eliminar */*
{Instance(T)} */* instancias */*
)
UpdateTabs (T', {T}) */* con links a T por su clave, y hereda los links de T */*

Mapeos:
F' = ReplaceTable (F,T, T')
UpdateMaps (F', OS)

B.5.2 Sub-Regla R5.2 – CUBE GROUP

Dado un cubo, su función de correspondencia explícita y una tabla a la que tiene correspondencia, se genera una nueva tabla eliminando los atributos sin correspondencia, agrupando por los demás y aplicando roll-up a los atributos que corresponden a las medidas.

Objetos:
OS ∈ SchCubes */* un cubo */*
Rup ⊆ RollUpExpressions(OS.Measure♦Is) */* predicados de roll-up para las medidas */*

Entrada:
Mapeos: F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
Tablas: T ∈ MapTables(F, ObjectItems(OS)) */* tabla que mapea al objeto */*

Condiciones:
T.As – MapAttributes(F, ObjectItems(OS)) ≠ ∅ */* hay atributos para eliminar */*
MapTables(F, ObjectItems(OS)) = 1 */* el objeto es mapeado por una sola tabla */*

Resultados:
Tablas:
T' = PrimitiveP9 (*/* Aggregate Generation */*
{T}, */* esquema fuente */*
MapAttributes(F, ObjectItems(OS.Measure))♦AttrName, */* medidas */*
Rup, */* funciones de resumen */*
T.As♦AttrName – MapAttributes(F, ObjectItems(OS))♦AttrName */*atributos a eliminar */*
{Instance(T)} */* instancias */*
)
UpdateTabs (T', {T}) */* con links a T por su clave, y hereda los links de T */*

Mapeos:
F' = ReplaceTable (F,T, T')
UpdateMaps (F', OS)

B.6 Regla R6 - DRILL-UP

B.6.1 Sub-Regla R6.1 - HIERARCHY-DRILL-UP

Dados dos cubos (base y recursivo), la función de correspondencia, la tabla a la que tiene correspondencia el cubo base, un drill-up respecto a una de las dimensiones y la función de correspondencia de su jerarquía, se genera una nueva tabla con el nuevo nivel de detalle para la dimensión aplicando roll-up a las medidas.

Objetos:
CB, CR ∈ SchCubes */* los cubos base y recursivo */*
Dup ∈ DrillUps(CB, CR) */* un drill-up */*

Entrada:
Mapeos:
F ∈ Mappings(ObjectItems(CB)) */* función de mapeo del cubo base */*
Fd = Dup.Map */* función de mapeo de la jerarquía del drill-up */*
Tablas:
T ∈ MapTables(F, ObjectItems(CB)) */* tabla que mapea al cubo base */*
Td ∈ MapTables(Fd,ObjectKeyItems(Dup.Lbase) ∪
Dup.Lnews♦ObjectKeyItems) */* tabla que mapea a los involucrados en el roll-up */*

Condiciones:
Dup.Lnews ≠ ∅ */* sino es eliminar completamente la dimensión */*
MapTables(F, ObjectItems(CB)) = 1 */* el cubo base es mapeado por una sola tabla */*
MapTables(Dup.Map, ObjectKeyItems(Dup.Lbase) ∪ Dup.Lnews♦ObjectKeyItems)
/ los involucrados en el roll-up mapean a una sola tabla */*

Resultados:
Tablas:
T' = PrimitiveP8 (*/* Hierarchy Roll Up */*
{T, Td}, */* esquema fuente */*
MapAttributes(F, ObjectItems(CB.Measure))♦AttrName, */* medidas */*
MapAttributes(Dup.Map, Dup.Lnews♦ObjectKeyItems)♦AttrName,
*/*granularidad */*
Dup.Rup, */* funciones de resumen */*
{}, */* atributos a eliminar de la tabla de medidas, no incluye granularidades inferiores */*
{}, */* atributos a eliminar de la tabla de dimensión, incluir granularidades inf. si se quiere */*
false, */* no generar la nueva jerarquía */*
{Instance(T)} */* instancias */*
)
UpdateTabs (T', {T}) */* con links a T por su clave, y hereda los links de T */*

Mapeos:
F' = ReplaceTable (F,T, T')
UpdateMaps (F', CR)

B.6.2 Sub-Regla R6.2 - TOTAL-DRILL-UP

Dados dos cubos (base y recursivo), la función de correspondencia, la tabla a la que tiene correspondencia el cubo base, y un Drill-Up respecto a una de las dimensiones, se genera una nueva tabla sin nivel de detalle para la dimensión aplicando Roll-Up a las medidas.

Objetos:
CB, CR ∈ SchCubes */* los cubos base y recursivo*/*
Dup ∈ DrillUps(CB, CR) */* un drill-up */*

Entrada:
Mapeos: F ∈ Mappings(ObjectItems(CB)) */* función de mapeo del cubo base */*
Tablas: T ∈ MapTables(F, ObjectItems(CB)) */* tabla que mapea al cubo base */*

Condiciones:
Dup.Lnews = ∅ */* se elimina la dimensión */*
# MapTables(F, ObjectItems(CB)) = 1 */* el cubo base es mapeado por una sola tabla */*

Resultados:
Tablas:
T' = PrimitiveP9 (*/* Aggregate Generation */*
{T}, */* esquema fuente */*
MapAttributes(F, ObjectItems(CB.Measure))♦AttrName, */* medidas */*
Dup.Rup, */* funciones de resumen */*
MapAttributes(F, ObjectKeyItems(Dup.Lbase)) ♦AttrName */*atributos a eliminar */*
{Instance(T)} */* instancias */*
)
UpdateTabs (T', {T}) */* con links a T por su clave, y hereda los links de T */*

Mapeos:
F' = ReplaceTable (F,T, T')
UpdateMaps (F', CR)

B.7 Regla R7 - PRIMARY KEY

Dado un objeto del esquema intermedio, su función de correspondencia y una tabla que a la que tiene correspondencia, se genera una nueva tabla modificando su clave primaria.

Objetos:
OS ∈ SchFragments ∪ SchCubes */* un fragmento o un cubo */*

Entrada:
Mapeos: F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
Tablas: T ∈ MapTables(F, ObjectItems(OS)) */* tabla que mapea al objeto */*

Condiciones:
MapAttributes(F, ObjectKeyItems(OS)) ≠ T.PK */* las claves no coinciden */*
MapTables(F, ObjectItems(OS)) = 1 */* el objeto es mapeado por una sola tabla */*

Resultados:
Tablas:
T' = PrimitiveQ4 (*/* Primary Key Update */*
{T}, */* esquema fuente */*
MapAttributes(F, KeyItems(OS)) ♦ AttrName */*atr. de la nueva clave */*
{Instance(T)} */* instancias */*
)
UpdateTabs (T', {T}) */* con links a T por su clave, y hereda los links de T */*

Mapeos:
F' = ReplaceTable (F,T, T')
UpdateMaps (F', OS)

B.8 Regla R8 - FILTER

Dado un objeto del esquema intermedio, su función de correspondencia, una tabla a la que tiene correspondencia, y una condición que deben cumplir las instancias, se genera una nueva tabla filtrando las tuplas que no cumplen la condición.

Objetos:
OS ∈ SchFragments ∪ SchCubes ∪ SchStrips */* un fragmento, un cubo o una franja */*
Cond ∈ Predicates({T •As / T ∈ SchTables}) */* una condición */*

Entrada:
Mapeos: F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
Tablas: T ∈ MapTables(F, ObjectItems(OS)) */* tabla que mapea al objeto */*

Condiciones:
Cond ≠ ⊥ */* la condición está bien definida */*
MapTables(F, ObjectItems(OS)) = 1 */* el objeto es mapeado por una sola tabla */*

Resultados:
Tablas:
T' = PrimitiveQ3 (*/* Instance Filter */*
{T}, */* esquema fuente */*
Cond, */* condición de filtrado */*
{Instance(T)} */* instancias */*
)
UpdateTabs (T', {T}) */* con links a T por su clave, y hereda los links de T */*

Mapeos:
F' = ReplaceTable (F,T, T')
UpdateMaps (F', OS)

C Primitivas

Este apéndice presenta y analiza las primitivas mencionadas en el documento, haciendo hincapié en las sentencias SQL que generan la instancia de datos asociada a cada una. Una razón importante para incluir la presentación de las primitivas es que algunas de éstas fueron modificadas, ya sea porque no disponían originalmente de una sentencia SQL asociada (Q2 y Q4), o porque no la presentaban de una forma útil a este contexto (P4.1 y P7).

Cabe resaltar que no todas las primitivas presentadas en [Mar00] están incluidas en este apéndice, ya que no todas son utilizadas por el algoritmo de [Per01] (sección 3.4) y tampoco se mencionaron en alguna otra parte de este documento. También es necesario indicar que [Per01] introduce cuatro primitivas en su trabajo (Q1, Q2, Q3 y Q4) que sí se incluyen en este apéndice.

C.1 Primitiva P1 - Identity

Dada una relación, ésta genera otra que es exactamente la misma de la que se partió.

Entrada:
- Esquema Origen: $R \in Rel$

Instancia Generada:

```
r' = select * from R
```

C.2 Primitiva P3 - Temporization

Esta primitiva agrega un elemento de tiempo al conjunto de atributos de una relación.

Entrada:
- Esquema Origen: $R(A_1, ..., A_n) \in Rel / \exists X \subset \{A_1, ..., A_n\} \wedge X \in Att_K(R)$
- T atributo de tiempo / $DOM(T) = \{t_0, .., t_k\}$ conjunto de medidas de tiempo $\vee$ $DOM(T) = \{c / c \subseteq \{t_0, .., t_k\}$ conjunto de medidas de tiempo$\}$.
- Key, un argumento booleano. Este dice si T será parto de la clave de R o no.

Instancia Generada:
Se supone que existe t() la función que da el atributo de tiempo al que se hace mención.

```
r' = select A1, .., An, t() as T from R
```

Nota: Sobre esta última primitiva se debe hacer la precisión de que la función t() que sea elegida para la implementación debe ser determinista.

C.3 Primitiva P4.1 - Version Digits

Para generalizar la clave, dígitos de versión son agregados a cada valor del atributo.

Entrada:

- Esquema Origen: $R(A_1, ..., A_n) \in Rel$ / $A_j \in Att_K(R)$

Instancia Generada [24]:
Se supone que existe f(), la función que da el numero de versión que se agregará.

```
r' = select A1, ..., f() || Aj, ..., An from R
```

Nota: Sobre esta última se observa que originalmente se consideraba A_1 como el atributo a versionar, sin embargo en un intento por enfatizar que éste no es siempre el primero se introdujo A_j. También es de notar que la función f() a utilizar en una implementación concreta debe ser determinista.

C.4 Primitiva P6.1 - DD-Adding 1-1

Dada una relación, esta primitiva agrega un atributo que es derivado de otros de la misma relación.

Entrada:

- Esquema origen: $R(A_1, ..., A_n) \in Rel$
- $f(A_{k1}, ..., A_{km})$ / $\{A_{k1}, ..., A_{km}\} \subseteq \{A_1, ..., A_n\}$

Instancia Generada:

```
r' = select A1, ..., An, f(Ak1, ..., Akm) from R
```

C.5 Primitiva P6.3 - DD-Adding N-N

Esta primitiva agrega a una relación un atributo que es derivado de un atributo de otra relación. En este caso la función de cálculo trabaja sobre un conjunto de tuplas de la otra relación. Este conjunto es obtenido mediante una operación de join entre las dos relaciones.

Entrada:

- Esquemas Origen: $R_1(A_1, ..., A_n)$, $R_2(B_1, ..., B_m) \in Rel$
- e(B) / $B \in \{B_1, ..., B_m\}$, la expresión de agregación
- X / $X \subset Att_D(R_2)$, atributos de agrupamiento adicionales
- A / $A \in \{A_1, ..., A_n\} \wedge A \in \{B_1, ..., B_m\}$, atributo de join

[24] El símbolo "||" hace referencia al operador de concatenación.

Instancia Generada:

```
r' = select A1, ..., An, e(B)
   from R1, R2
   where R1.A = R2.A
   group by A1, ..., An, X
```

C.6 Primitiva P7 - Attribute Adding

Dada una relación de dimensión, esta primitiva agrega uno o más atributos a ésta.

Entrada:

- Esquema origen: $R(A_1, ..., A_n) \in Rel_D$
- $\{B_1, ..., B_m\}$, conjunto de atributos a agregar

Instancia Generada:
Se supone la existencia de las funciones: $\{f_1(A_1, ..., A_n), ..., f_m(A_1, ..., A_n)\}$, tal que conforman un conjunto de funciones con las que se obtienen los valores de los atributos.

```
r' = select A1, ..., An, f1(A1, ..., An) as B1, ..., fm(A1, ..., An) as Bm
   from R
```

Nota: Sobre esta última se observa que originalmente por cada atributo constante se agregaba un valor NULL a la construcción. Es decir la construcción era de la forma:

```
r' = select A1, ..., An, NULL, ..., NULL from R
```

Esto claramente no es práctico a los efectos de la carga. Para corregir este aspecto se plateó la extensión agregando el conjunto de funciones f a la primitiva.

C.7 Primitiva P8 - Hierarchy Roll Up

Dada una relación de medida R_1 y una relación de jerarquía R_2, esta primitiva realiza una operación de Roll-Up a R_1 por uno de sus atributos siguiendo la jerarquía dada por R_2. Además ésta puede generar otra relación de jerarquía con la granularidad correspondiente.

Entrada:

- Esquemas origen:
  $R_1(A_1, ..., A_n) \in Rel_M / \exists A \subset \{A_1, ..., A_n\} \wedge A \subset Att_{FK}(R_1, R_2)$
  $R_2(B_1, ..., B_{n'}) \in Rel_J / A \subset \{B_1, ..., B_{n'}\} \wedge A \subset Att_K(R_2)$
- Atributos de medida: $Z = \{Z_1, \ldots, Z_k\} = Att_M(R_1), Z \subset \{A_1, ..., A_n\}$
- Agregaciones de los atributos: $\{e_1(Z_1), \ldots, e_k(Z_k)\}$
- Nivel de la jerarquía: $B / B \subset \{B_1, ..., B_{n'}\} \wedge B \subset Att_D(R_2)$
- Atributos que por su granularidad salen de R_1: $X / X \subset \{A_1, ..., A_n\} \wedge X \subset (Att_D(R_1) \cup Att_M(R_1))$
- Atributos que por su granularidad salen de R_2: $Y / Y \subset \{B_1, ..., B_{n'}\}$
- Indica si genera nueva jerarquía: agg_h ∈ Bolean

Instancia Generada:
Se supone:

$V = \{V_1, \ldots, V_m\} / V = ((((\{A_1, ..., A_n\} - A) \cup B) - X) - Z$
$V' = \{V'_1, \ldots, V'_{m'}\} / V' = \{B_1, ..., B_{n'}\} - Y$

```
r' = select V1, …, Vm, e1(Z1), …, ek(Zk)
   from R1, R2
   where R1.A = R2.A
   group by V1, …, Vm
```

Si se indica agg_h:

```
r'' = select distinct V'1, …, V'm, from R2
```

C.8 Primitiva P9 - Aggregate Generation

Dada una relación de medidas, esta primitiva genera otra relación de medida, donde la información es resumida (o agrupada) dado un conjunto de atributos.

Entrada:

- Esquemas origen: $R(A_1, ..., A_n) \in Rel_M$
- $Z = \{Z_1, .., Z_k\}$, un conjunto de atributos de medida
- $\{e_1(Z_1), \ldots, e_k(Z_k)\}$, expresiones de agregación
- $Y / Y \subset \{A_1, ..., A_n\} \wedge Y \subset (Att_D(R) \cup Att_M(R))$, los atributos a remover

Instancia Generada:
Sea $X = \{X_1, .., X_m\} = \{A_1, \ldots, A_n\} - Z$

```
r' = select X1, .., Xm, e1(Z1), …, ek(Zk) from R group by X1, .., Xm
```

C.9 Primitiva Q1 - Relation Join

Dadas dos relaciones esta primitiva genera una nueva con el join de ambas.

Entrada:

- Esquema Origen: $R_1(A_1, .., A_n), R_2(B_1, .., B_m) \in Rel$
- $f(C_1, .., C_k) / \{C_1, .., C_k\} \subseteq \{A_1, .., A_n\} \cup \{B_1, .., B_m\}$, la función de join.
- $Y_1 \subseteq \{A_1, .., A_n\}$, el conjunto de atributos de R_1 a ser excluidos.
- $Y_2 \subseteq \{B_1, .., B_m\}$, el conjunto de atributos de R_2 a ser excluidos.

Instancia Generada:

```
r' = select (Att(R1) - Y1) ∪ (Att(R2) - Y2)
   from R1, R2
   where f(C1, .., Ck)
```

C.10 Primitiva Q2 - Attribute Renaming

Dada una relación esta primitiva genera otra renombrando algunos de los atributos.

Entrada:
- Esquema Origen: R (A_1, .., A_n) ∈ *Rel*
- {A_{i1}, .., A_{im}} ⊆ {A_1, .., A_n}, un conjunto de atributos de R.
- {B_1, .., B_m}, los nuevos nombres para los atributos.

Instancia Generada:
Se supone que f es una función que nos da el nombre de los atributos según la primitiva, es decir:

$\forall\ 1 \le j \le n$
$f(A_j) = A_j$ si $A_j \notin \{A_{i1}, .., A_{im}\}$
$f(A_j) = B_k$ si $A_j = A_{ik}$

```
r' = select A1 as f(A1)(25), .., An as f(An) from R
```

Nota: Sobre esta última se observa que originalmente la instancia fue planteada como r'=r sin embargo se ha adaptado la definición con el fin de explicitar el cambio de nombres de atributos.

C.11 Primitiva Q3 - Instance Filtering

Dada una relación, esta primitiva genera otra que es estructuralmente la misma, pero elimina algunas tuplas que no satisfacen la condición.

Entrada:
- Esquema Origen: R(A_1, ..., A_n) ∈ *Rel*
- f(C_1, ..., C_k) / {C_1, ..., C_k} ⊆ {A_1, ..., A_n}

Instancia Generada:

```
r' = select A1, ..., An from R where f(C1, ..., Ck)
```

C.12 Primitiva Q4 - Primary Key Modification

Dada una relación, esta primitiva cambia su clave primaria.

Entrada:
- Esquema Origen: R(A_1, ..., A_n) ∈ *Rel*
- {C_1, ..., C_k} ⊆ {A_1, ..., A_n}, los nuevos atributos de la clave

[25] Claramente esta sentencia no es válida en el lenguaje SQL si esperamos contar con la función en el momento de creación de la vista. Sin embargo la función f fue utilizada sólo con fines demostrativos para presentar el resultado. La función f no tiene que ser utilizada para un caso específico dado que directamente se pueden generar las cláusulas "as" donde fueran necesarias.

Instancia Generada:

```
r' = select A1, ..., An from R
```

Nota: Sobre esta última se observa que originalmente la expresión asociada a la instancia fue r'=r sin embargo tomamos una expresión SQL equivalente con el objetivo de mantener todas las definiciones de esta forma.

D Extensiones y Lemas

A lo largo de este trabajo surgen un conjunto de propiedades y extensiones al AR necesarias para completar el análisis. A continuación son introducidas respetando el orden de aparición.

D.1 Extensión del Operador de Proyección

En [GUW00] se define un operador de proyección extendido entre otros aspectos con la capacidad de introducir expresiones en la proyección. Estas expresiones sólo pueden involucrar atributos de la relación proyectada, constantes, operadores aritméticos y operadores de cadena.

Al tratar de expresar con este operador la primitiva P3 existe un problema, ya que la función que da la marca de tiempo no está definida en estos términos, ya que ésta por ejemplo puede ser del estilo "*getDate()*".

El problema que se presenta al utilizar un conjunto más amplio de expresiones que incluyan funciones disponibles por el manejador de base de datos, es que las funciones pueden no ser determinísticas. Una función es determinística cuando cualquier invocación que se realice con los mismos parámetros dará siempre el mismo resultado. La función mencionada anteriormente "*getDate()*" no es determinística ya que retornará la fecha del momento en que sea invocada.

Dado que está fuera del alcance de este trabajo no se permitirá el uso de funciones no determinísticas. Esto quiere decir que la implementación de la primitiva P3 será por ejemplo con un valor constante o aunque no lo sea la función deberá ser siempre determinística.

D.2 Extensión del Operador de Agrupamiento y Agregación

En [GUW00] se define un operador de agrupamiento y agregación no presente entre los operadores del álgebra relacional básica. A los efectos de este trabajo es necesario contar con un operador de agrupamiento y agregación que tenga la capacidad de contener expresiones y soportar cambios de nombres en los atributos tal y como lo hace el operador de proyección. En particular también se necesita que las expresiones involucradas sean las definidas en la sección D.1.

Con este fin se escribirá el operador de γ con estas hipótesis tomando el siguiente recaudo. Si γ contiene expresiones que no son permitidas en el operador γ original se utilizará la siguiente equivalencia como definición del nuevo operador:

$$\gamma_{f1 \to B1, \ldots, fm \to Bm, e1 \to C1, \ldots, ek \to Ck}(R) \equiv \Pi_{f1 \to B1, \ldots, fm \to Bm, C1, \ldots, Ck}(\gamma_{A1, \ldots, An, e1 \to C1, \ldots, ek \to Ck}(R))$$

Donde:

- f_1, .., f_m son funciones tales como las que se pueden usar en las proyecciones, por ejemplo f_i podría ser $f_i = A_1 + A_n$ o también $f_i = A_1$, es decir directamente un atributo. Éstas no son funciones de agregación.
- A_1, ..., A_n son los atributos de R por los cuales se agrupa. Éstos son todos los atributos que son utilizados por las funciones f_1, .., f_m.
- e_1, .., e_k son expresiones de agregación tales como se pueden usar en el operador γ original

En definitiva las expresiones se resuelven en la proyección y se agrupa por los atributos que sean necesarios para resolver f_1, .., f_m. Con esto se quiso exponer que el nuevo operador γ puede ser definido en función de los anteriores.

En el contexto de este trabajo además interesa saber que esta expresión del AR puede ser escrita con una única vista. Para nuestro nuevo operador la vista correspondiente a $\gamma_{f1 \to B1, .., fm \to Bm, e1 \to C1, .., ek \to Ck}$ (R) será de la forma:

```
create view V as select f1 as B1, .., fm as Bm, e1 as C1, .., ek as Ck
from R group by A1, …, An
```

Traduciendo la equivalencia se tendrá que la vista anterior debe ser equivalente a:

```
create view V' as select A1, …, An, e1 as C1, .., ek as Ck from R group
by A1, …, An
create view V'' as select f1 as B1, .., fm as Bm, C1, .., Ck from V'
```

Para validar la equivalencia se debe probar que toda tupla en V también está en V'' y viceversa. A continuación se esbozará cómo será la prueba.

V → V'': Sea $v_1 = <b_1, .., b_m, c_1, .., c_k>$ perteneciente a V, es claro que existe un conjunto no vacio de tuplas en R que contribuye a que v_1 pertenezca al resultado.
Este conjunto sería dado por $\{<a_1, \ldots, a_n, a_{n+1}, ..., a_{n+k}>\}$, donde $a_1, \ldots, a_n$ son los valores de los atributos por los cuales se agrupa y $a_{n+1}, ..., a_{n+k}$ son los que se resumen con las funciones de agrupamiento $(e_1, \ldots, e_k)$.
Entonces existe una tupla $<a_1, \ldots, a_n, c_1, .., c_k>$ en V', considerando que $c_1, .., c_k$ son los resultados de las funciones de agrupamiento $(e_1, \ldots, e_k)$ que no pueden ser diferentes ya que se aplican sobre las mismas tuplas que se hubieran aplicado en V y claramente las funciones de agrupamiento son determinísticas.
Como V'' no descarta tuplas de V' debe existir una tupla $<b'_1, \ldots, b'_m, c_1, .., c_k>$ en V'' donde $b'_1, \ldots, b'_n$ son los resultados de $f_1, \ldots, f_m$.
Notar que las funciones f_1, .., f_m no pueden depender de otros atributos que no sean A_1, …, A_n, dado que son los únicos que participan en el agrupamiento original planteado en V; si no fuera así la consulta estaría planteada erróneamente.
A su vez como las funciones f_1, .., f_m se aplican sobre los mismos parámetros (porque los parámetros o son constantes o son los atributos $A_1, \ldots, A_n$) y son determinísticas, se puede deducir que los resultados deben ser los mismos que se obtuvieron en V, o sea: $b_1 = b'_1, \ldots b_m = b'_m$.
Con esto último es posible concluir que si v_1 está en V estará también en V''.

En el caso V'' → V los argumentos son similares.

D.3 Equivalencia de Instancia asociada a la primitiva P6.3

En esta sección se demostrará que la propuesta realizada para representar la instancia de datos generada por la aplicación de la primitiva P6.3 es equivalente a la original. Dicha equivalencia puede resumirse utilizando álgebra relacional como sigue:

$$\gamma_{Att(R),E}(R \otimes_{cond(R,R')} R') \equiv \prod_{Att(R),e}(R \otimes_{cond(R,R')} \gamma_{CR',E\to e}(R'))$$

Ecuación 1: Equivalencia que plantea la alternativa a la expresión de la primitiva P6.3.

Donde *Att*(R) es el conjunto de todos los atributos de R, E es una expresión de agregación que sólo involucra atributos de R' (para ser consistentes con la primitiva P6.3) y $C_{R'}$ son los atributos de R' que aparecen en cond(R, R'). Además se tomará como hipótesis para esta demostración que $\delta(R) = R$, o sea que no hay repetidos en R.

Para demostrar la equivalencia se debe probar que toda tupla presente en un lado de la equivalencia también lo está en el otro. Aquí se realiza una justificación informal, sin embargo en [YL94] se encontrará una demostración formal.

Para contextualizar [YL94] es necesario considerar que la equivalencia principal ahí presentada (Ecuación 2) es como se puede apreciar en la Ecuación 3 (simplificaciones mediante). Por otro lado, la Ecuación 3 no es más que otra forma de escribir la equivalencia tratada en este trabajo (Ecuación 1).

$$E1: F[AA]\pi_A[GA_1,GA_2,AA]\zeta[GA_1,GA_2]\sigma[C_1 \wedge C_0 \wedge C_2](R_1 x R_2)$$
$$E2: \pi_A[GA_1,GA_2,FAA]\sigma[C_0](F[AA]\pi_A[GA_1+,AA]\zeta[GA_1+]\sigma[C_1]R_1 \ x \ \pi_A[GA_2+]\sigma[C_2]R_2)$$

Ecuación 2: Equivalencia principal de [YL94].

$$F[AA]=E,\ FAA=e,\ AA=Att(e)\ \ GA_1=\emptyset,\ GA_1+=C_{R'},\ GA_2=Att(R),\ GA_2+=\ Att(R)\cup C_R,$$
$$C_1=\emptyset,\ C_2=\emptyset,\ C_0=cond(R,R'),\ R_1=R',R_2=R$$

$$E1: E\pi_A[Att(R),Att(e)]\zeta[Att(R)]\sigma[cond(R,R')](R' x R)$$
$$E2: \pi_A[Att(R),e]\sigma[cond(R,R')](E\pi_A[C_{R'},Att(e)]\zeta[C_{R'}]R' \ x \ R)$$

Ecuación 3: Equivalencia principal sustituida.

Las dependencias funcionales que son utilizadas para comprobar el cumplimiento de la equivalencia principal (Ecuación 4) [YL94], se verifican al ser nuevamente expresadas en términos de este trabajo (Ecuación 5), lo cual demuestra que la equivalencia de este trabajo es correcta. Recordemos que por cond(R,R'), sabemos que $C_R \rightarrow C_{R'}$, pues la condición es una equivalencia de estas variables, además $Att(R) \supseteq C_R$, por ende $Att(R) \rightarrow C_{R'}$.

$$FD_1: (GA_1,GA_2) \rightarrow GA_1+$$
$$FD_2: (GA_1+,GA_2) \rightarrow RowID(R_2)$$

Ecuación 4: Dependencias funcionales para probar la equivalencia principal.

$$FD_1: Att(R) \rightarrow C_{R'}$$
$$FD_2: (C_{R'}, Att(R)) \rightarrow RowID(R)$$

Ecuación 5: Dependencias funcionales contextualizadas.

D.3.1 Justificación Informal

Suponiendo que existe una tupla $t = <r_1, .., r_n, e>$ del lado izquierdo de la igualdad, dado que se está utilizando una operación de agregación se deduce que existe un conjunto de tuplas $\{<r_1, .., r_n, e_0>, .., <r_1, .., r_n, e_i>\}$ donde los primeros atributos son comunes y al resumir los valores $e_0, .., e_i$ se obtiene e.

A partir de esto es posible deducir que por lo menos existe una tupla $<r_1, .., r_n>$ en R que contribuye al join, porque de no estar $<r_1, .., r_n, e>$ no podría aparecer en el resultado de la operación de join. Notar además que no puede existir más de una tupla contribuyendo, dado que se están proyectando todos los atributos de R y por hipótesis se sabe que no hay tuplas repetidas en R.

De forma similar es posible deducir que existe un conjunto de tuplas $\{<r'_1, .., e_0, .., r'_n,>, .., <r'_1, .., e_i, .., r'_n>\}$ en R', dado que por hipótesis se sabe que los valores $e_0, ..,e_i$ sólo pueden provenir de un atributo de R'. Para que cada uno de los valores $e_0, ..,e_i$ aparezca en la operación de join, como sólo hay una tupla del lado de R, debe haber exactamente i del lado de R' que satisfagan la condición. Notar que no altera el resultado que las tuplas existentes estén o no repetidas, sólo se establece que serán i.

A continuación se analiza la parte derecha de la equivalencia, considerando $<r_1, .., r_n>$ en R y el conjunto de tuplas mencionado anteriormente: $\{<r'_1, .., e_0, .., r'_n,>, .., <r'_1, .., e_i, .., r'_n>\}$ en R', ya que se estableció que en los párrafos anteriores que éstos existirán.

En la parte derecha es posible deducir que al agregar las tuplas de R' por los atributos $C_{R'}$ sólo se produce una tupla $<r'_{CR'1}, .., r'_{CR'n}, e>$. Para esto se debe observar que se estableció que $<r_1, .., r_n>$ existe en R y se sabe que no hay tuplas repetidas. Por otro lado en la condición de join se planteará igualar los atributos de ambas relaciones. Esto indica que los atributos en $C_{R'}$ sólo pueden tomar los valores de los atributos de R, y cada uno de éstos sólo puede tomar un valor (fijada la tupla de R). Por ende al agrupar por ellos es claro que éstos sólo pueden producir una tupla. Finalmente es sabido que al evaluar E con $e_0, .., e_i$ se obtiene e.

A partir de lo anterior se deduce que $<r_1, .., r_n, e>$ está en el lado derecho de la equivalencia, ya que $<r_1, .., r_n>$ está en R, $<r'_{CR'1}, .., r'_{CR'n}, e>$ está en R' y claramente la condición de join se satisface. Además como sólo hay dos tuplas involucradas se produce una única tupla como resultado. La proyección final elimina los atributos sobrantes.

Finalmente se ha llegado a que si $<r_1, .., r_n, e>$ está del lado izquierdo, lo estará también del lado derecho. Para el recíproco, suponiendo que existe una tupla $t = <r_1, .., r_n, e>$ del lado derecho de la igualdad, al considerar la operación de join y utilizando el hecho de que no hay repetidos en R se vuelve a deducir que exactamente una tupla $<r_1, .., r_n>$ está en R. Tomando nuevamente la operación de join se deduce que existe $<r'_{CR'1}, .., r'_{CR'n}, e>$ y al considerar el agrupamiento se deduce que $\{<r'_1, .., e_0, .., r'_n,>, .., <r'_1, .., e_i, .., r'_n>\}$ están en R', siendo como antes que e es el resultado de evaluar la agregación con los $e_0, .., e_i$.

Analizando la parte izquierda de la ecuación se observa que la operación de join producirá $\{<r_1, .., r_n, r'_1, .., e_0, .., r'_n,>, .., <r_1, .., r_n, r'_1, .., e_i, .., r'_n>\}$. Finalmente si se considera el agrupamiento, es simple deducir que $<r_1, .., r_n, e>$ estará en el lado izquierdo.

D.4 Extensiones a las Correspondencias

Con el fin de adaptar el trabajo de [Per01] a la generación de un proceso de carga es necesario modificar la definición de correspondencias presentada por dicho trabajo. Las modificaciones son consecuencia de los cambios realizados en las primitivas P4.1 y P7, que afectan directamente a las "*expresiones de mapeo*"[Per01] (expresiones de correspondencia en el contexto de este trabajo).

Según [Per01] las expresiones de correspondencia están definidas de la siguiente forma:

```
MapExpr ≡ DirectME ∪ 1CalcME ∪ NCalcME ∪ ExternME
```

Donde:

```
DirectME ≡ { <Expr, Tab, Patt> / Tab ∈ SchTables ∧ Expr ∈ Tab.As ∧
Patt = Tab♦{Expr} }
1CalcME ≡ { <Expr, Tabs, Patts> / Tabs ⊆ SchTables ∧ Patts ⊆ { T♦As /
T ∈ Tabs} ∧ Expr ∈ Expressions(Patts) }
NCalcME ≡ { <Expr, Tab, Patts> / Tab ∈ SchTables ∧ Patts ⊆ Tab♦As ∧
Expr ∈ RollUpExpressions(Patts) }
ExternME ≡ { <Expr, Ind> / Expr ∈ Expressions(∅) ∧ Ind ∈ {Constant,
Timestamp, Version} }
```

Recordar que:

- **Expr**: es una expresión tipada constituida a base de uno o más atributos de las tablas fuente.
- **Tab**: es una tabla de la base fuente.
- **Patt**: es un atributo de una de las tablas de la base fuente con el nombre de la tabla como prefijo.
- **Ind**: es un indicador de cómo se llenan los valores (para ExternME), pudiendo ser con una constante, una marca de tiempo o dígitos de versión.

Un problema es que se han extendido las primitivas P4.1 [26] (aceptando cualquier atributo para versionar) y P7 [27] (introduciendo funciones / expresiones para dar el valor de la instancia de los nuevos atributos), lo cual requiere modificar las correspondencias de tipo ExternME de forma de poder utilizar estas nuevas capacidades.

El cambio consiste en agregar un atributo a la definición de los ExternME con Ind ∈ {Versión} de forma de poder indicar cual atributo será versionado. Para el caso de la primitiva P7 no es necesario realizar un cambio en la definición porque ya se disponía de una expresión que será utilizada para contener el valor constante a utilizar. Como previamente la primitiva no necesitaba dicha expresión ésta no se utilizaba, sin embargo ahora sí será pertinente.

[26] Las modificaciones a la primitiva P4.1 pueden verse en la sección C.3.
[27] Las modificaciones a la primitiva P7 pueden verse en la sección C.6.

La nueva definición para ExternME será entonces:

```
CTExternME ≡ { <Expr, Ind> / Expr ∈ Expressions(∅) ∧ Ind ∈ {Constant,
Timestamp} }
VExternME ≡ { <Expr, Tab, Patt, Ind> / Tab ∈ SchTables ∧ Expr ∈
Expressions(Tab♦As) ∧ Patt = Tab♦As ∧ Ind ∈ {Version} }
ExternME ≡ CTExternME ∪ VExternME
```

Otro aspecto que se vio modificado por esta misma razón es la definición de las funciones MapAttributes (que indica el conjunto de ítems que tienen correspondencia a un atributo) y MapTables (que indica las tablas que son mencionadas en las correspondencias de un conjunto de ítems):

```
MapAttributes : Mappings x Set(SchItems) → Set(SchAttributes)
Sea F ∈ Mappings (Items, Tables), IS ⊆ Items
MapAttributes (F, IS) =
   {A / ∃ I ∈ IS . (F(I) ∈ DirectME ∧ F(I).Tab♦A = F(I).Patt)} ∪
   {A / ∃ I ∈ IS . (F(I) ∈ 1CalcME ∧ ∃ T ∈ F(I).Tab .(T♦A ∈ F(I).Patt))}

MapTables : Mappings x Set(SchItems) → Set(SchTables)
Sea F ∈ Mappings (Items,Tables), IS ⊆ Items
MapTables (F, IS) =
   { T / ∃ I ∈ IS . (F(I) ∈ DirectME ∧ T = F(I).Tab) } ∪
   { T / ∃ I ∈ IS . (F(I) ∈ 1CalcME ∧ T ∈ F(I).Tabs) }
```

Siendo las siguientes las nuevas definiciones:

```
MapAttributes : Mappings X Set(SchItems) → Set(SchAttributes)
Sea F ∈ Mappings (Items, Tables), Is ⊆ Items
MapAttributes (F, Is) =
   {A / ∃ I ∈ Is . (F(I) ∈ DirectME ∧ F(I).Tab♦A = F(I).Patt)} ∪
   {A / ∃ I ∈ Is . (F(I) ∈ 1CalcME ∧ ∃ T ∈ F(I).Tab .(T♦A ∈ F(I).Patt))} ∪
   {A / ∃ I ∈ Is . (F(I) ∈ ExternME ∧ F(I).Ind ∈ {Version} ∧ F(I).Tab♦A =
   F(I).Patt)}

MapTables : Mappings X Set(SchItems) → Set(SchTables)
Sea F ∈ Mappings (Items, Tables), Is ⊆ Items
MapTables (F, Is) =
   { T / ∃ I ∈ Is . (F(I) ∈ DirectME ∧ T = F(I).Tab) } ∪
   { T / ∃ I ∈ Is . (F(I) ∈ 1CalcME ∧ T ∈ F(I).Tabs) } ∪
   { T / ∃ I ∈ Is . (F(I) ∈ ExternME ∧ F(I).Ind ∈ {Version} ∧
     T = F(I).Tab) }
```

Los cambios anteriores simplemente reflejan que ahora las correspondencias para agregar dígitos de versión también refieren atributos de relaciones de la BDF. Resta mencionar que en el apéndice Reglas se pueden observar los cambios efectuados a la definición de las reglas que trabajan con estas correspondencias.

Además de la modificación a las correspondencias externas de tipo dígito de versión también fue necesario para la resolución de los problemas de múltiple conteo extender la definición de las correspondencias de tipo NCalcME. Con éste propósito se creó el tipo de correspondencia NCalcME', que a diferencia de la original permite manejar N relaciones. La nueva correspondencia se definirá de la siguiente forma:

```
NCalcME' ≡ { <Expr, Tabs, Patts> / Tabs ⊆ SchTables ∧ Patts ⊆ Tab♦As ∧
Expr ∈ RollUpExpressions(Patts) }
```

Esta correspondencia sólo es utilizada internamente en un contexto muy acotado que es el algoritmo que genera la vista unificada (sección 4.4), por lo cual no afecta el trabajo de [Per01] ni los aspectos anteriores a la presentación del algoritmo de cálculo de la vista unificada.

D.5 Extensiones a los Links

En este trabajo fue necesario agregar a la definición de SchLinks información sobre las cardinalidades de los links definidos en [Per01]. La información es necesaria para determinar posibles problemas en el algoritmo de carga, asociados a los problemas de múltiple conteo (ver sección 4.3.3). La definición brindada por [Per01] de SchLinks es la siguiente:

```
SchLinks ∈ {T1 / T1 ∈ SchTables} x {T2 / T2 ∈ SchTables} → Predicates
(T1♦As ∪ T2♦As)
```

Este trabajo no modifica la definición anterior pero sí agrega la siguiente:

```
SchLinksCards ∈ {T1/ T1 ∈ SchTables} X {T2 / T2 ∈ SchTables} → {1,N} x
{1,N}
```

Se agrega la definición de cardinalidad del link, definida como una función de las relaciones conectadas por el link, en las cardinalidades asociadas a éstas. Como nota se puede acotar que no se cambió la definición de SchLinks para minimizar el impacto en lo ya definido en función de ésta.

E Caso de Estudio

En esta sección se presenta un caso de estudio inspirado en el presentado en [Per01]. El caso trata sobre una empresa que distribuye productos agrícolas. Uno de los objetivos de tomar como base el caso de [Per01] es permitir fácilmente comparar los resultados de la generación del esquema y la carga; por otro lado éste se extendió de forma de permitir apreciar los aportes realizados por este trabajo. Se presenta el esquema conceptual, la BDF, los lineamientos de diseño y los resultados obtenidos por algoritmo de generación de esquema. Luego el esquema conceptual, la BDF, los lineamientos de diseño se vuelven a utilizar en el algoritmo de carga y se presentan los resultados de éste.

E.1 Esquema Conceptual

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

El caso de estudio plantea 4 dimensiones: Artículos, Clientes, Vendedores y Fechas. La dimensión Cantidades así como la dimensión Cantidad de Facturas representan a las medidas, pero son tratadas como una dimensión más ya que CMDM trabaja con dimensionalidad genérica [Car00]. La Figura 37 muestra la representación gráfica de las dimensiones.

El caso presenta dos relaciones dimensionales: Ventas, que vincula casi todas las dimensiones relevadas y Facturas, que utiliza otro subconjunto de las dimensiones. La Figura 36 muestra la representación gráfica de cada relación dimensional.

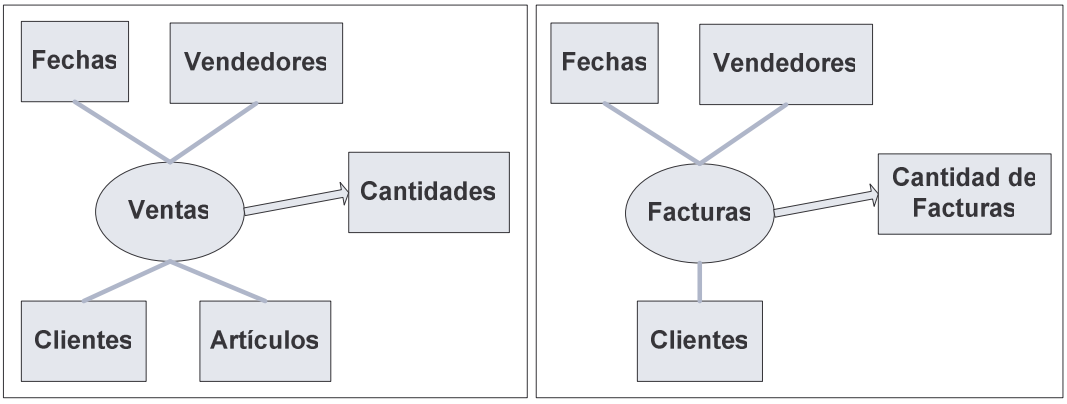


Figura 36: Relaciones dimensionales del caso de estudio.

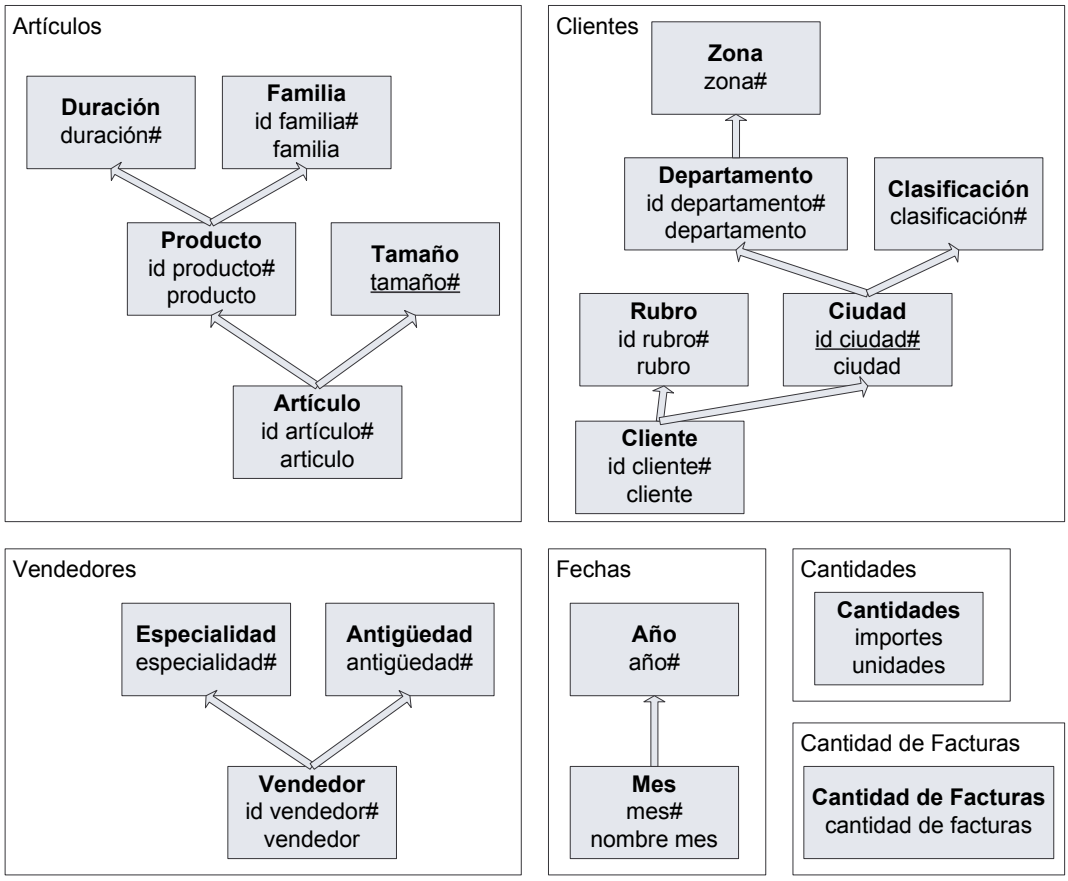


Figura 37: Dimensiones y niveles del caso de estudio.

E.2 Base de Datos Fuente

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 desglose 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 perecedero).

- **Códigos** (tipo, código, descripción)

Contiene descripciones de códigos utilizadas por el sistema. El campo tipo indica a qué se refiere el código (se encuentran códigos 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.

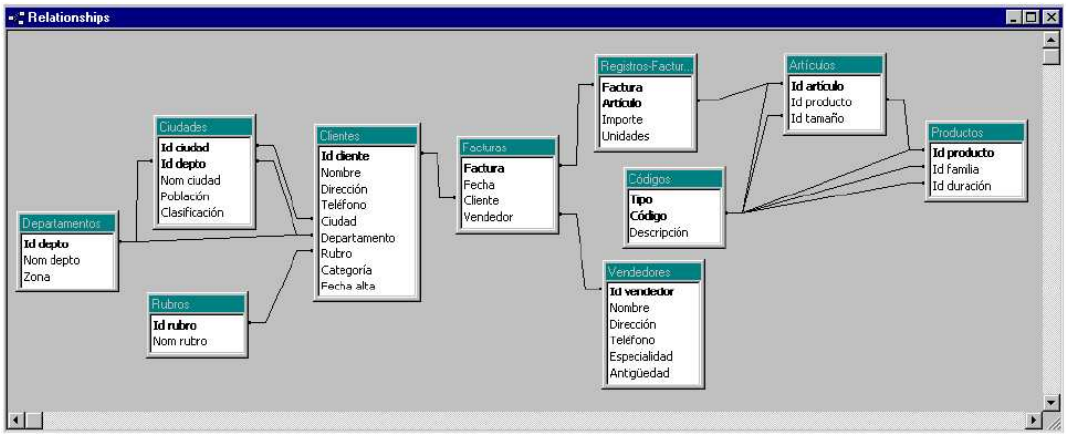


Figura 38: Links utilizados en la BDF.

En el ejemplo se necesita utilizar alias en dos ocasiones. La primera dado que el atributo Código de la tabla Códigos tiene links con dos atributos de la tabla Artículos (observar la parte superior derecha de la Figura 38). La segunda porque el atributo Código de la tabla Códigos tiene links con tres atributos de la tabla Productos.

Se resuelven las ambigüedades generando cinco alias de la tabla Códigos:

Códigos_Id_Artículo (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)

Estos alias se implementaran utilizando vistas, por lo cual se crearán las siguientes vistas en la base de datos:

```
create view Codigos_Id_Articulo as select tipo, codigo as id_articulo,
descripcion from Codigos;
create view Codigos_Id_Tamaño as select tipo, codigo as id_tamaño,
descripcion from Codigos;
create view Codigos_Id_Producto as select tipo, codigo as id_producto,
descripcion from Codigos;
create view Codigos_Id_Familia as select tipo, codigo as id_familia,
descripcion from Codigos;
create view Codigos_Id_Duracion as select tipo, codigo as id_duracion,
descripcion from Codigos;
```

Los nuevos links utilizando los alias definidos son:

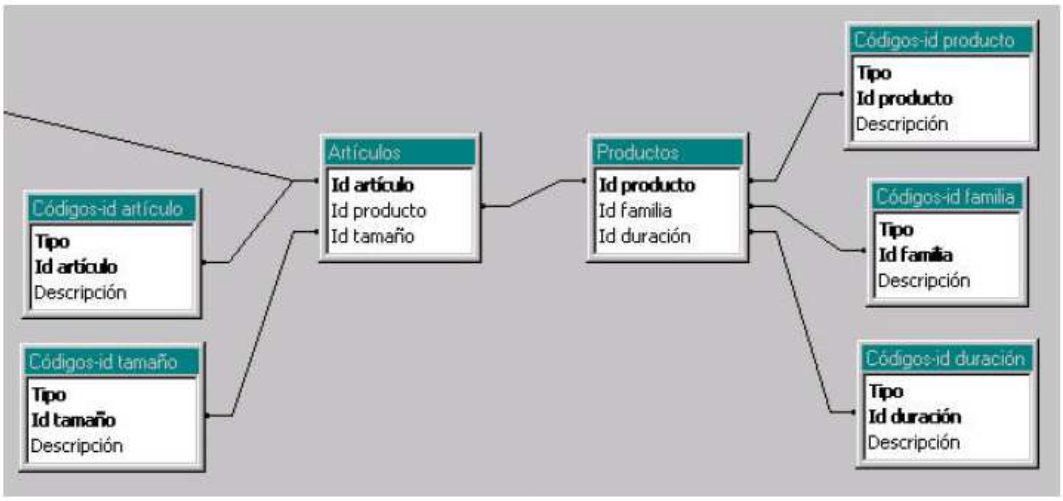


Figura 39: Alias definidos para el caso de estudio.

E.3 Lineamientos

En el caso de estudio se utilizarán los siguientes lineamientos:

Para la relación dimensional Venta se elige materializar tres cubos:

- DWVenta1: Con detalle de artículos, clientes, vendedores y meses.
- DWVenta2: Con detalle de artículos, rubros, vendedores y meses.
- DWVenta3: Con detalle de artículos y meses.
- DWVenta4: Con detalle de clientes, vendedores y meses.

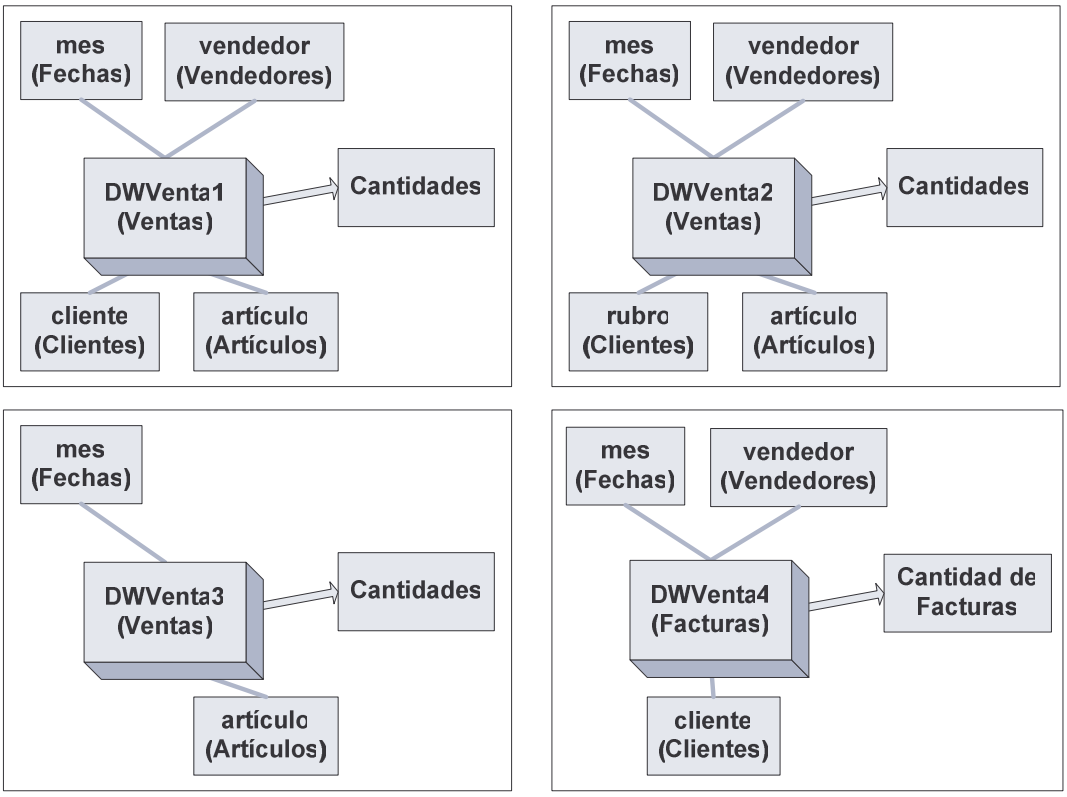


Figura 40: Cubos definidos para el caso de estudio.

Se decide utilizar las siguientes franjas para los cubos antes definidos:

- DWVenta1: Una franja para las ventas del año actual (DWVenta1Banda01) y otra con el resto de la historia (DWVenta1Banda02).
- DWVenta2: Una única franja (DWVenta201).
- DWVenta3: Una única franja (DWVenta301).
- DWVenta4: Una única franja (DWVenta401).

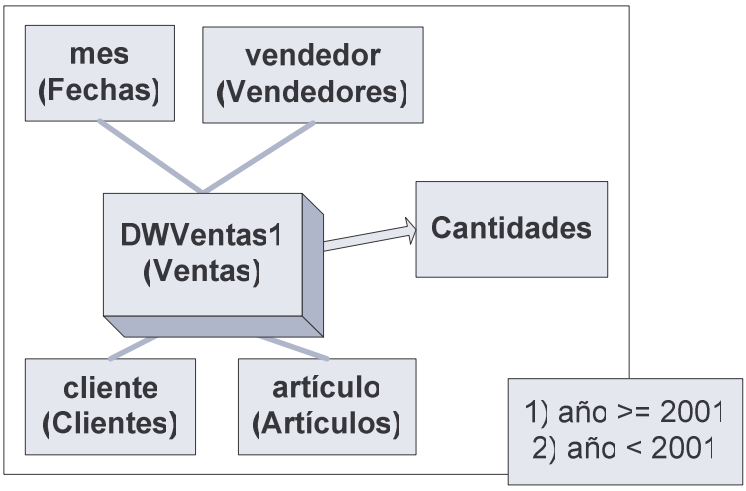


Figura 41: Única franja definida para el caso de estudio.

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

- Clientes: 2 fragmentos, uno con cliente y rubro (DWClientes), y el otro con los restantes (DWCiudades).
- Artículos: 3 fragmentos, uno con artículo y tamaño (DWArticulos), otro con producto y duración (DWProductos), y el otro con familia (DWFamilias).
- Vendedores: una única relación (DWVendedores).
- Fechas: una única relación (DWFechas).
- Cantidades: No se implementará, será utilizada como medida.
- Cantidad de Facturas: No se implementará, será utilizada como medida.

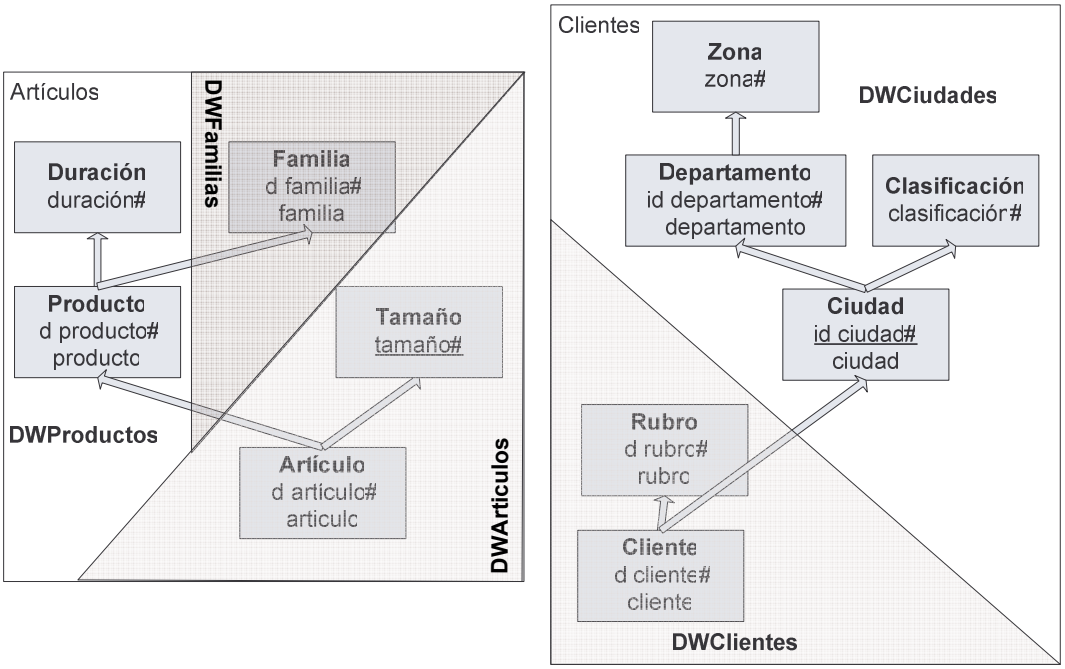


Figura 42: Fragmentos de Artículos y Clientes definidos para el caso de estudio.

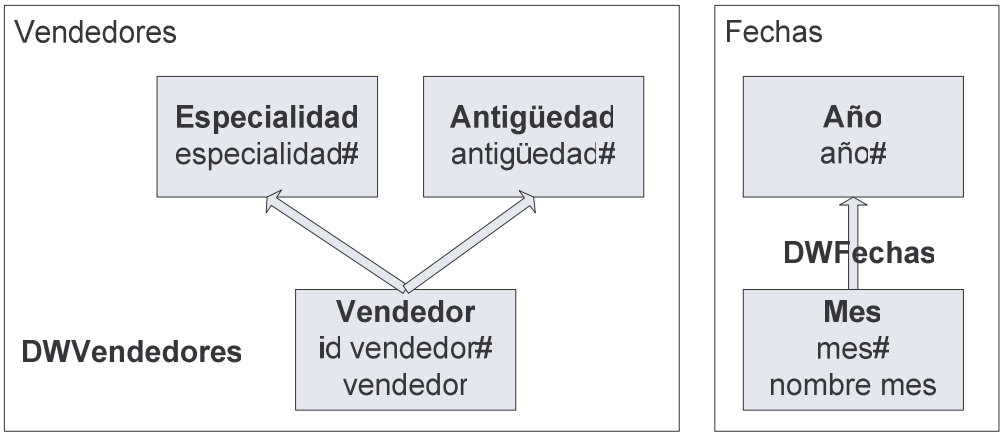


Figura 43: Fragmentos de Vendedores y Fechas definidos para el caso de estudio.

E.4 Correspondencias

E.4.1 Correspondencias de Fragmentos

La Figura 44 y la Figura 45 muestran las correspondencias de los fragmentos de la dimensión Clientes. La Figura 46, la Figura 47 y la Figura 48 muestran las correspondencias de los fragmentos de la dimensión artículos. La Figura 49 muestra las correspondencias del único fragmento de la dimensión vendedores y la Figura 50 muestra el de la dimensión fechas.

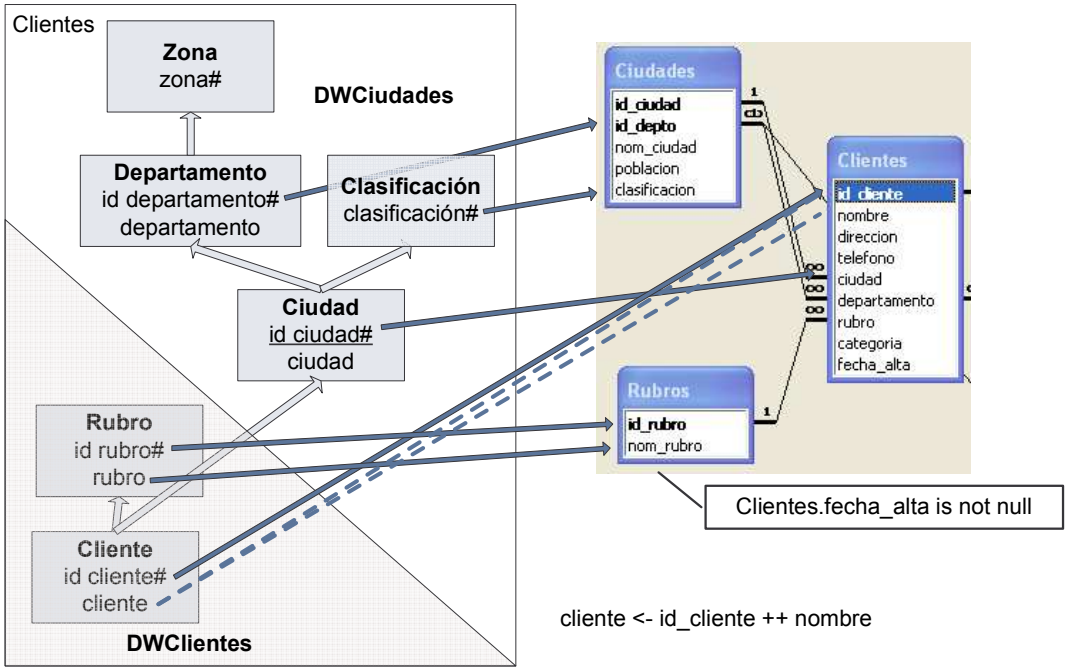


Figura 44: Correspondencias del fragmento DWClientes.

Notar que es necesario establecer correspondencias a niveles de DWCiudades para poder completar la clave de la dimensión. Notar además que las correspondencias y los links están ideados para permitir las operaciones de Join necesarias.

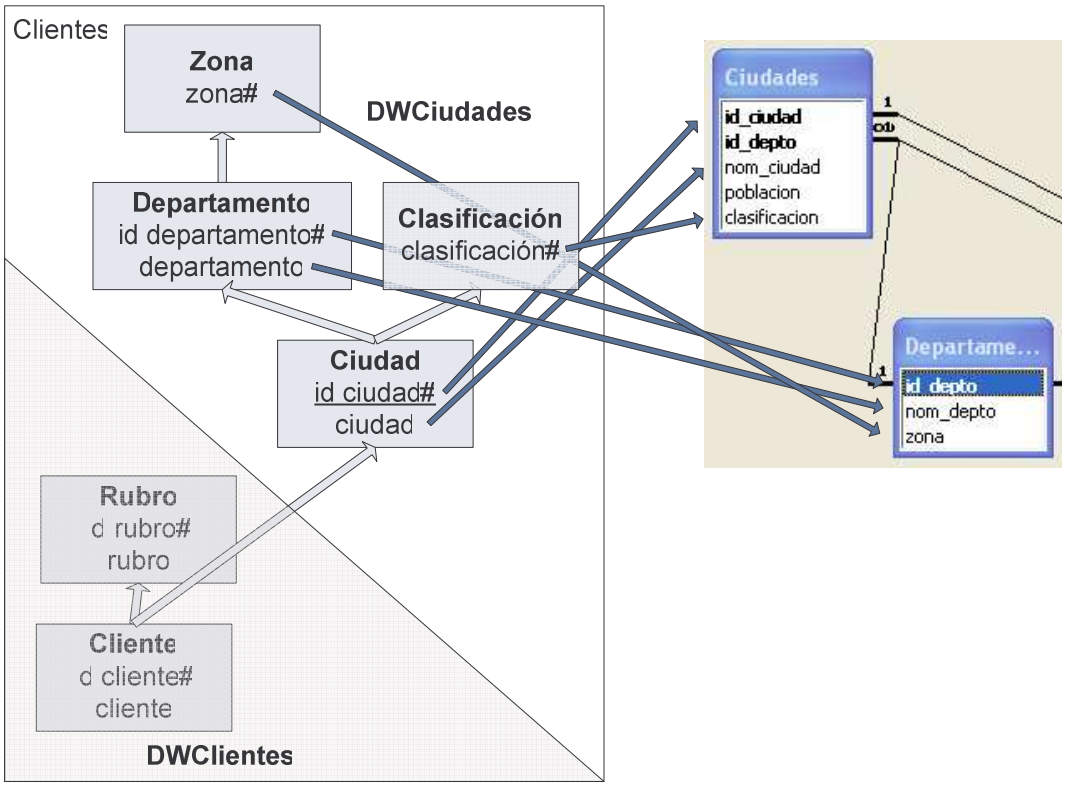


Figura 45: Correspondencias del fragmento DWCiudades.

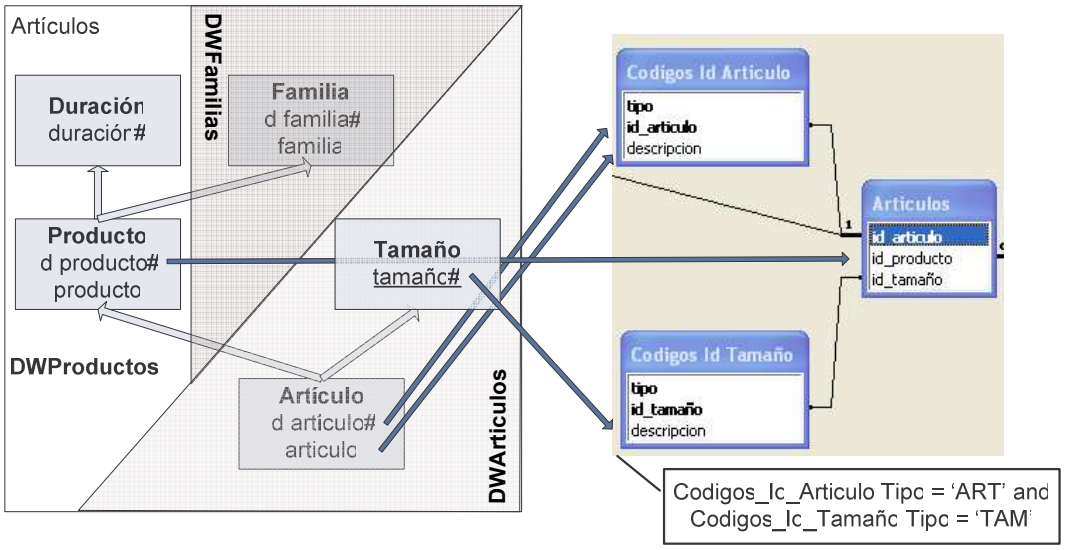


Figura 46: Correspondencias del primer fragmento DWArticulos.

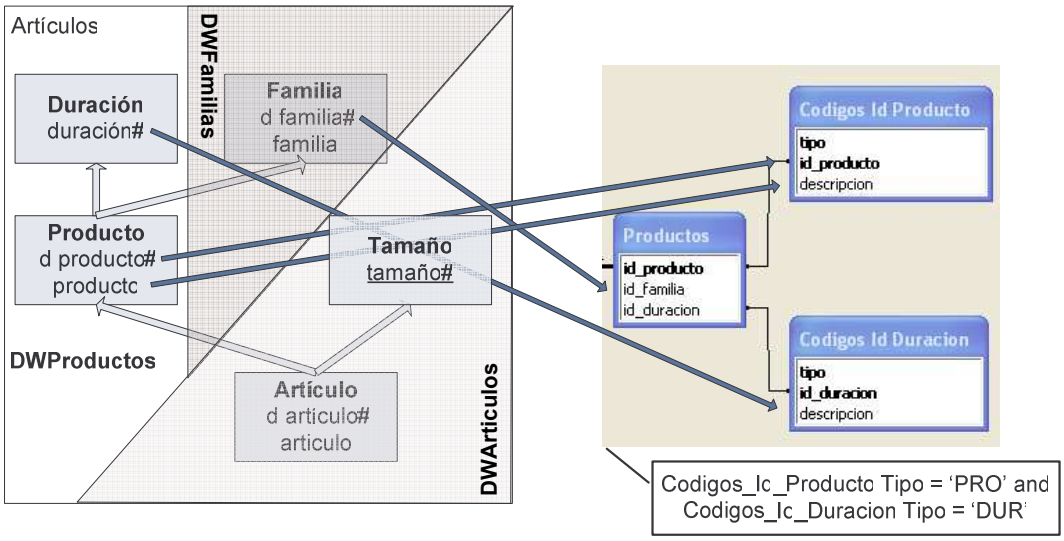


Figura 47: Correspondencias para el fragmento DWProductos.

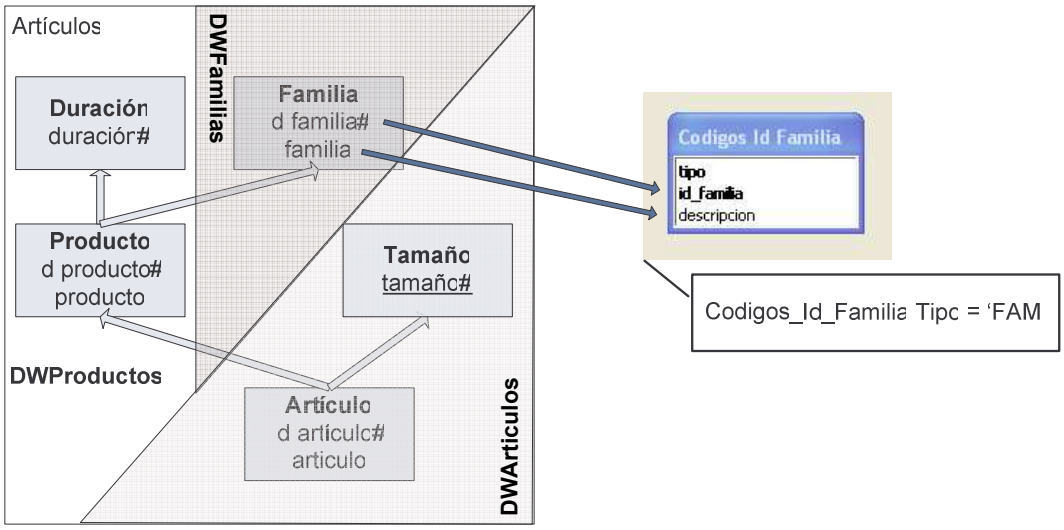


Figura 48: Correspondencias del fragmento DWFamilias.

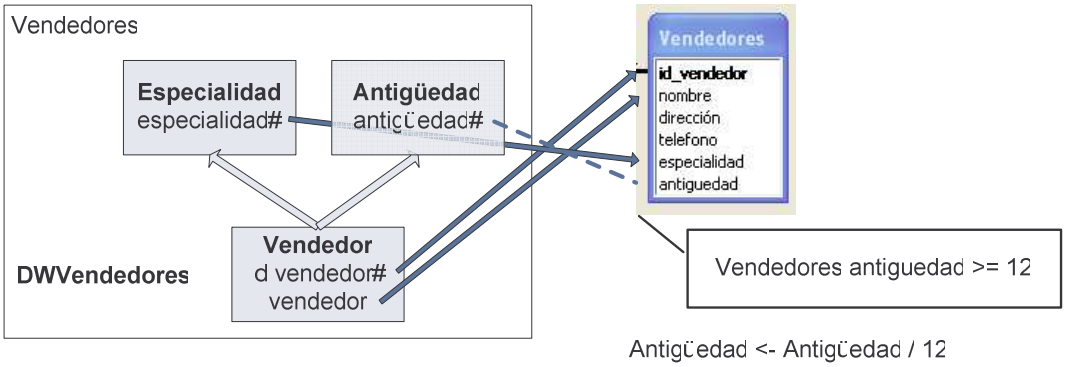


Figura 49: Correspondencias del fragmento DWVendedores.

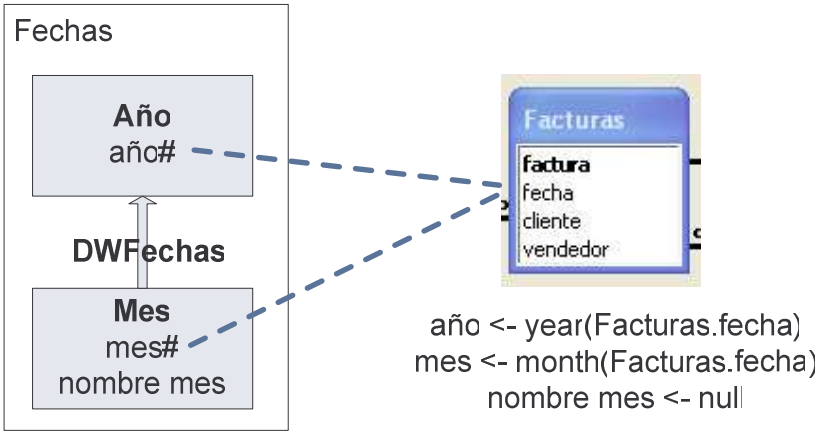


Figura 50: Correspondencias del fragmento DWFechas.

E.4.2 Correspondencias de Cubos

La Figura 51 muestra las correspondencias del cubo DWVenta1. La Figura 52 muestra las correspondencias del cubo DWVenta2 como Drill-Up del cubo DWVenta1 por la dimensión Clientes. La Figura 53 muestra las correspondencias del cubo DWVenta3 obtenidas del cubo DWVenta1 eliminando dimensiones.

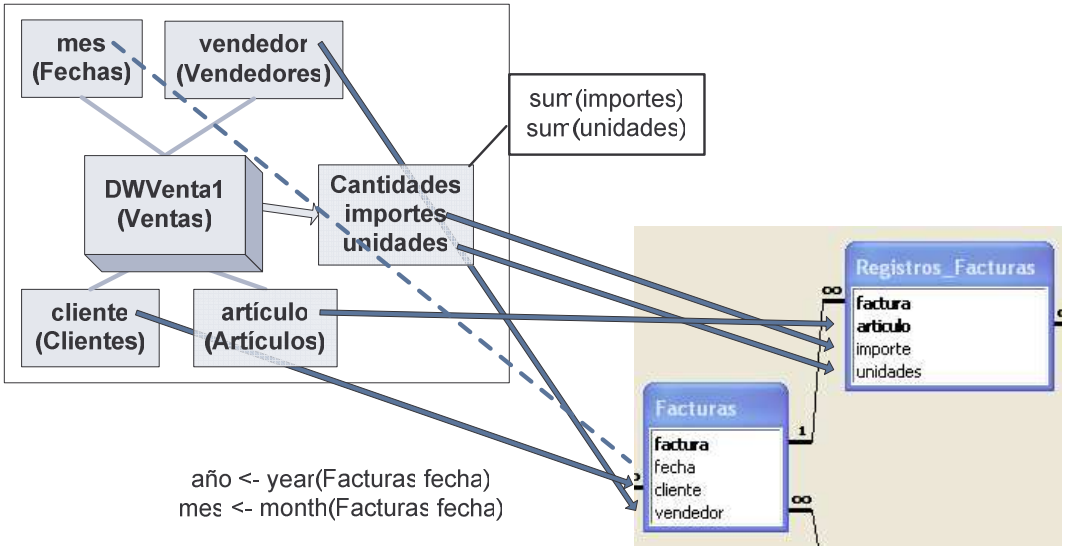


Figura 51: Correspondencias del cubo DWVenta1 de la relación dimensional Ventas.

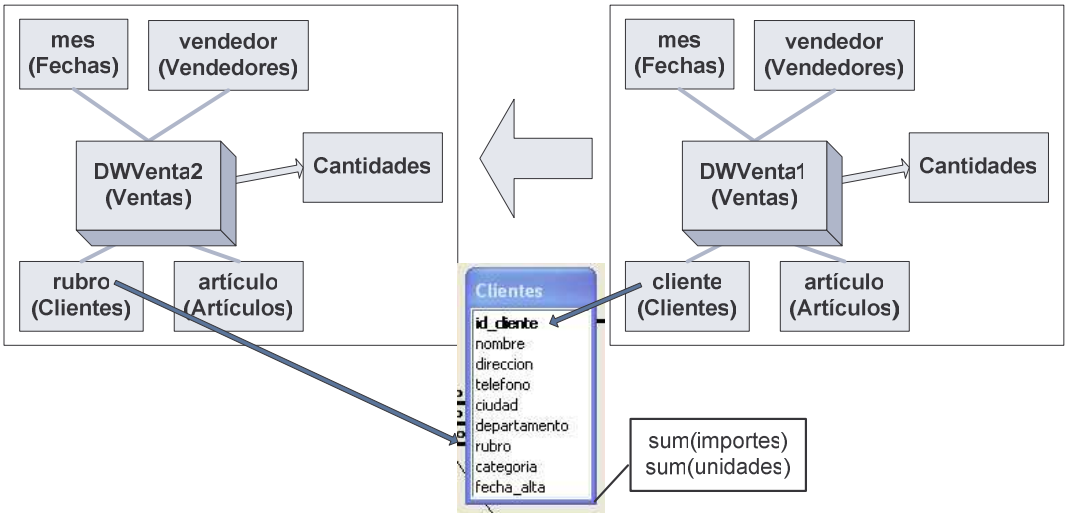


Figura 52: Correspondencias para el cubo DWVenta2 de la relación dimensional Ventas.

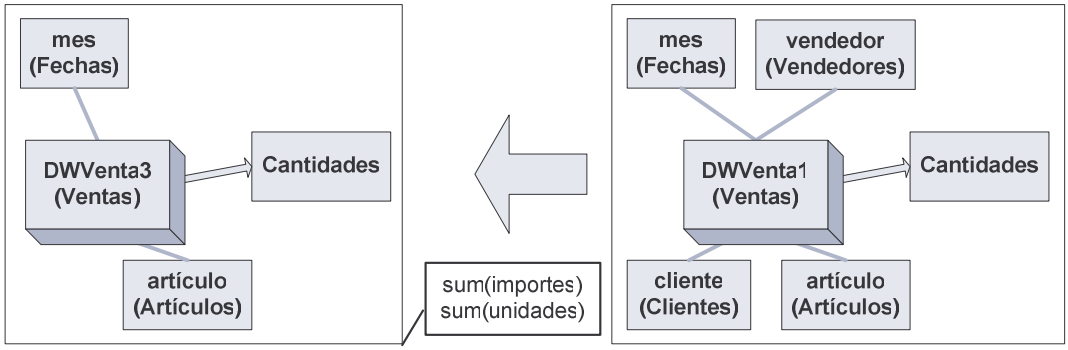


Figura 53: Correspondencias para el cubo DWVenta3 de la relación dimensional Ventas.

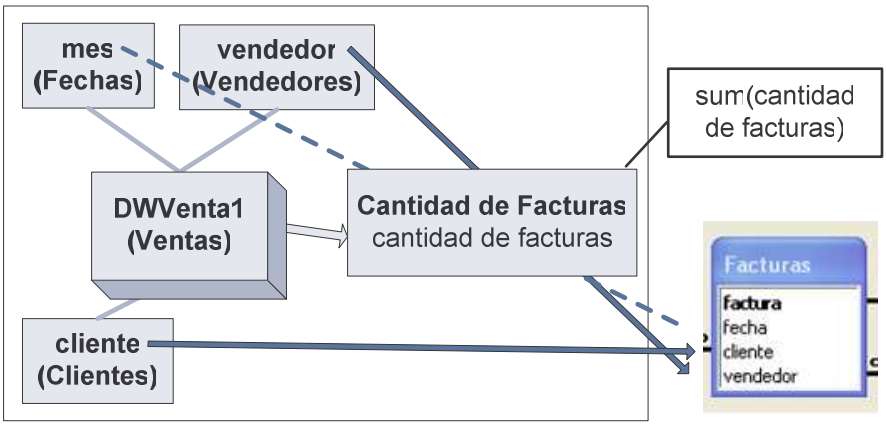


Figura 54: Correspondencias para el cubo DWVenta4 de la relación dimensional Facturas.

E.5 Aplicación del Algoritmo

A continuación se muestran los resultados obtenidos tras la aplicación del algoritmo, puntualizando las consideraciones realizadas en cada caso. El ejemplo se implementó sobre Microsoft Access ™, por lo cual las sentencias están generadas para esta plataforma.

E.5.1 Fragmentos

Para el caso del fragmento DWClientes la vista obtenida tras la aplicación del algoritmo es:

```
create view DWClientes as
   select Ciudades.clasificacion, Ciudades.id_depto as id_departamento,
     Clientes.ciudad as id_ciudad, Clientes.id_cliente,
     Rubros.id_rubro, Rubros.nom_rubro as rubro,
     Cstr(id_cliente) ++ ' - ' ++ nombre as cliente
   from Rubros, Clientes, Ciudades
   where Rubros.id_rubro = Clientes.rubro and
     Ciudades.id_depto = Clientes.departamento and
     Ciudades.id_ciudad = Clientes.ciudad and
     (Clientes.fecha_alta is not null)
   group by Ciudades.clasificacion, Ciudades.id_depto, Clientes.ciudad,
     Clientes.id_cliente, Rubros.id_rubro, Rubros.nom_rubro,
     Cstr(id_cliente) ++ ' - ' ++ nombre;
```

En el inicio de la ejecución del algoritmo, más precisamente en el armado del esqueleto, podemos notar que se decide colocar las relaciones Rubros, Clientes y Ciudades, dado que existen ítems con correspondencias a estas relaciones. Conjuntamente se colocan también las condiciones de join que permiten mantener la unidad del esqueleto.

En la siguiente parte del algoritmo, para cada una de las correspondencias directas se agrega el atributo correspondiente a la cláusula SELECT. Si el nombre del ítem difiere del nombre del atributo también se agrega la cláusula as para hacer la corrección. Además de lo anterior cada uno de los atributos también es colocado en la cláusula GROUP BY. Para el caso del ítem cliente que se calcula mediante una expresión, ésta también se agrega tanto a la cláusula SELECT como a la GROUP BY.

En la sección de procesamiento de condiciones del algoritmo se agrega la condición sobre la fecha de alta a la cláusula WHERE, que ya contenía las condiciones de join para las relaciones del esqueleto. Finalmente en base a todas estas consideraciones es que se forma la vista para el fragmento DWClientes.

Por último observemos que no puede haber un problema de clave primaria ya que no hay ningún cálculo de resumen involucrado. La relación propuesta por [Per01] para dicho fragmento es:

```
DWClientes07 (id_cliente, id_rubro, rubro, id_departamento, id_ciudad,
cliente)
```

Se puede observar que la única diferencia sustancial (obviando diferencias en el orden de los atributos), es que el atributo clasificacion no está presente en la relación, pero esto se debe a que por error Peralta lo elimina al formar DWClientes06 en [Per01], siendo que no debía hacerlo (el párrafo que explica cómo se forma DWClientes06 deja

claro que no debe quitarse). Una diferencia no sustancial es el nombre pero esto se debe a que Peralta agrega un índice a las distintas relaciones que va construyendo el algoritmo.

Una observación que cabe realizar es que más allá del algoritmo utilizado para llegar a la vista, el resultado obtenido es consistente con los requerimientos del fragmento. Para el caso del fragmento DWCiudades la vista propuesta por el algoritmo es:

```
create view DWCiudades as
   select Ciudades.id_ciudad, Ciudades.nom_ciudad as ciudad,
     Departamentos.id_depto as id_departamento,
     Departamentos.nom_depto as departamento, Ciudades.clasificacion,
     Departamentos.zona
   from Departamentos, Ciudades
   where Departamentos.id_depto = Ciudades.id_depto
   group by Ciudades.id_ciudad, Ciudades.nom_ciudad,
     Departamentos.id_depto, Departamentos.nom_depto,
     Ciudades.clasificacion, Departamentos.zona;
```

Las consideraciones para la vista anterior son prácticamente las mismas que se aplicó con DWClientes, por lo cual no las reiteraremos. La relación presentada en [Per01] es:

```
DWCiudades03 (id_ciudad, ciudad, clasificacion, id_departamento,
departamento, zona)
```

Aquí no hay diferencias entre ambas propuestas y como en el caso anterior la propuesta expresada por la vista es adecuada para el problema del fragmento en cuestión. Para el caso de DWArticulos la vista propuesta es:

```
create view DWArticulos as
   select Articulos.id_producto, Codigos_Id_Articulo.id_articulo,
     Codigos_Id_Articulo.descripcion as articulo,
     Codigos_Id_Tamaño.descripcion as tamaño
   from Codigos_Id_Articulo, Articulos, Codigos_Id_Tamaño
   where Codigos_Id_Articulo.id_articulo = Articulos.id_articulo and
     Codigos_Id_Tamaño.id_tamaño = Articulos.id_tamaño and
     (Codigos_Id_Articulo.Tipo='ART' and Codigos_Id_Tamaño.Tipo='TAM')
   group by Articulos.id_producto, Codigos_Id_Articulo.id_articulo,
     Codigos_Id_Articulo.descripcion, Codigos_Id_Tamaño.descripcion;
```

La vista anterior presenta la particularidad de que Codigos_Id_Articulo y Codigos_Id_Tamaño son vistas y no relaciones, dado que son producidas por el mecanismo de alias, exceptuando este aspecto las consideraciones realizadas por el algoritmo son las mismas que en los casos anteriores. La relación presentada en [Per01] para este caso es:

```
DWArticulos05 (id_articulo, articulo, id_producto, tamaño)
```

Nuevamente no hay diferencias entre la relación y la vista propuesta, además la vista propuesta es consistente con los requerimientos del fragmento. En el caso de DWProductos la vista propuesta es:

```
create view DWProductos as
   select Productos.id_familia, Codigos_Id_Producto.id_producto,
     Codigos_Id_Producto.descripcion as producto,
     Codigos_Id_Duracion.descripcion as duracion
   from Productos, Codigos_Id_Duracion, Codigos_Id_Producto
   where Codigos_Id_Producto.id_producto = Productos.id_producto and
     Codigos_Id_Duracion.id_duracion = Productos.id_duracion and
     (Codigos_Id_Producto.Tipo='PRO' and
       Codigos_Id_Duracion.Tipo='DUR')
   group by Productos.id_familia, Codigos_Id_Producto.id_producto,
     Codigos_Id_Producto.descripcion, Codigos_Id_Duracion.descripcion;
```

La relación para DWProductos es:

```
DWProductos05 (id_familia, id_producto, producto, duracion)
```

No hay diferencias entre la vista y la relación esperada, además de que la vista propuesta es consistente con los requerimientos del fragmento. Para el caso de DWFamilias la vista propuesta es:

```
create view DWFamilias as
   select Codigos_Id_Familia.id_familia,
     Codigos_Id_Familia.descripcion as familia
   from Codigos_Id_Familia
   where (Codigos_Id_Familia.Tipo='FAM')
   group by Codigos_Id_Familia.id_familia,
     Codigos_Id_Familia.descripcion;
```

La relación para DWFamilias es:

```
DWFamilias03 (id_familia, familia)
```

Tampoco hay diferencias, en el caso de DWVendedores la vista es:

```
create view DWVendedores as
   select Vendedores.id_vendedor, Vendedores.nombre as vendedor,
     Vendedores.especialidad, Vendedores.antiguedad / 12 as antigüedad
   from Vendedores
   where (Vendedores.antiguedad >=12)
   group by Vendedores.id_vendedor, Vendedores.nombre,
     Vendedores.especialidad, Vendedores.antiguedad / 12;
```

La relación para el caso es:

```
DWVendedores04 (id_vendedor, vendedor, especialidad, antiguedad)
```

Tampoco en esta ocasión hay observaciones particulares. En el caso de DWFechas por otro lado la vista propuesta es:

```
create view DWFechas as
   select null as nombre_mes, month(Facturas.fecha) as mes,
     year(Facturas.fecha) as año
   from Facturas
   group by month(Facturas.fecha), year(Facturas.fecha);
```

En la vista anterior es pertinente observar que nombre_mes se consideró con correspondencia de tipo constante y valor null, por eso el algoritmo no lo incluyó en la cláusula GROUP BY. La relación propuesta para este caso es:

```
DWFechas04 (año, mes, nombre_mes)
```

Hemos de notar que en lo anterior debemos realizar un cambio con respecto a la propuesta original, el cambio es que la clave debe incluir también el año. En primera instancia al observar las correspondencias inducimos que mes representaba únicamente el número de mes (mes <- month(Facturas.Fecha)), sin embargo posteriormente encontramos otras referencias que no concordaban y en su lugar sugerían que mes era una combinación del mes y el año (en DWVenta1Banda01 por ejemplo mes >= ene-2001). Para decidir esta disyuntiva optamos por la opción de que representara solamente el número de mes e hicimos cambios como el de la clave de DWFechas04 para ser coherentes. En definitiva la relación que tomaremos para esta vista es:

```
DWFechas04 (año, mes, nombre_mes)
```

El cambio antes mencionado, de la misma forma que otros que serán presentados a continuación, ya fueron considerados en la especificación que se presentó en las secciones E.1, E.2, E.3 y E.4. Se mencionan en esta instancia con el fin de justificar las diferencias con las relaciones propuestas en [Per01].

E.5.2 Cubos

Pasando ahora a las vistas asociadas a los cubos, primeramente tenemos las vistas asociadas ambas bandas de DWVenta1, para el caso de DWVenta1Banda01:

```
create view DWVenta1Banda01 as
  select Facturas.cliente as id_cliente,
    Facturas.vendedor as id_vendedor,
    Registros_Facturas.articulo as id_articulo,
    sum(Registros_Facturas.importe) as importes,
    sum(Registros_Facturas.unidades) as unidades,
    year(Facturas.fecha) as año, month(Facturas.fecha) as mes
  from Registros_Facturas, Facturas
  where Facturas.factura = Registros_Facturas.factura and
    ((year(Facturas.fecha)) >= 2001)
  group by Facturas.cliente, Facturas.vendedor,
    Registros_Facturas.articulo, year(Facturas.fecha),
    month(Facturas.fecha);
```

Y para el caso de DWVenta1Banda02, que como se puede observar sólo difiere en la condición asociada a la franja:

```
create view DWVenta1Banda02 as
   select Facturas.cliente as id_cliente,
      Facturas.vendedor as id_vendedor,
      Registros_Facturas.articulo as id_articulo,
      sum(Registros_Facturas.importe) as importes,
      sum(Registros_Facturas.unidades) as unidades,
      year(Facturas.fecha) as año,
      month(Facturas.fecha) as mes
   from Registros_Facturas, Facturas
   where Facturas.factura = Registros_Facturas.factura and
      ((year(Facturas.fecha)) < 2001)
   group by Facturas.cliente, Facturas.vendedor,
      Registros_Facturas.articulo, year(Facturas.fecha),
      month(Facturas.fecha);
```

En lo anterior debemos aclarar qué, conforme a la observación realizada para el fragmento DWFechas, acerca del tipo de atributo mes, se ajustaron las condiciones de las franjas para que fueran compatibles con el tipo del atributo. La condición mes >= ene-2001 que se planteó originalmente, se cambió por año >= 2001, que es equivalente y no presenta el problema antes mencionado. De igual forma mes < ene-2001 se reemplazó por año < 2001. Otro efecto del cambio de DWFechas fue la inclusión de año en los cubos que manejaban sólo el mes, dado que cambia la clave de la dimensión. Para realizar la modificación fue necesario agregar la correspondencia de para el nuevo ítem.

Sobre el procesamiento realizado por el algoritmo debemos notar que a diferencia de los casos de fragmentos ahora sí fue necesario considerar si se trataba de una medida al momento de agregar atributos a la cláusula SELECT y a la cláusula GROUP BY. Los casos en los que no se trataba de una medida se conducían como en el caso de los fragmentos, los casos en los que sí era una medida se agregaban al SELECT pero utilizando la función de agregación asociada a la medida y no se agregaban al GROUP BY.

La otra diferencia que presenta el procesamiento de esta vista, con respecto al procesamiento que su hubiera realizado en el caso que fuera un fragmento, es la inclusión de la condición asociada a la franja.

Debemos notar que a pesar de haber una relación 1-N entre las relaciones del esqueleto, el hecho de que todas las medidas estén en la relación Registros_Facturas hace que no exista un problema de múltiple conteo. Claramente tampoco puede ocurrir un problema de asociación en las medidas, dado que no hay ninguna correspondencia de resumen. Las relaciones propuestas por [Per01] para estos dos cubos son las siguientes:

```
DWVenta1Banda01   (id_cliente,   id_vendedor,   id_articulo,   importes,
unidades, mes)
DWVenta1Banda02   (id_cliente,   id_vendedor,   id_articulo,   importes,
unidades, mes)
```

Considerando la observación sobre la clave de DWFechas estas son:

```
DWVenta1Banda01   (id_cliente,   id_vendedor,   id_articulo,   importes,
unidades, año, mes)
DWVenta1Banda02   (id_cliente,   id_vendedor,   id_articulo,   importes,
unidades, año, mes)
```

Tomando esta última consideración notamos que no hay diferencias entre la relación planteada y la vista obtenida. Para el caso de DWVenta2 y su única franja DWVenta201 la vista obtenida por el algoritmo es:

```
create view DWVenta201 as
   select Facturas.vendedor as id_vendedor,
     Registros_Facturas.articulo as id_articulo,
     Clientes.rubro as id_rubro,
     sum(Registros_Facturas.importe) as importes,
     sum(Registros_Facturas.unidades) as unidades,
     year(Facturas.fecha) as año, month(Facturas.fecha) as mes
   from Registros_Facturas, Facturas, Clientes
   where Facturas.factura = Registros_Facturas.factura and
     Clientes.id_cliente = Facturas.cliente
   group by Facturas.vendedor, Registros_Facturas.articulo,
     Clientes.rubro, year(Facturas.fecha), month(Facturas.fecha);
```

Sobre la ejecución del algoritmo debemos puntualizar que al ejecutar la transformación de correspondencias recursivas a base, se elimina la correspondencia del ítem id_cliente y se agrega la del ítem id_rubro, más allá de esto el algoritmo transcurre como en el caso anterior. Por otro lado, la relación propuesta para este cubo, ya considerando el cambio de la dimensión DWFechas es:

```
DWVenta201 (id_rubro, id_vendedor, id_articulo, importes, unidades,
año, mes)
```

Como en los casos anteriores coinciden la relación y la vista obtenida. Para el caso de la vista DWVenta301 el resultado es el siguiente:

```
create view DWVenta301 as
   select Registros_Facturas.articulo as id_articulo,
     sum(Registros_Facturas.importe) as importes,
     sum(Registros_Facturas.unidades) as unidades,
     year(Facturas.fecha) as año, month(Facturas.fecha) as mes
   from Registros_Facturas, Facturas
   where Facturas.factura = Registros_Facturas.factura
   group by Registros_Facturas.articulo, year(Facturas.fecha),
     month(Facturas.fecha);
```

En el caso de esta vista el algoritmo elimina las correspondencias de los ítems id_cliente e id_vendedor al ejecutar la transformación de correspondencias recursivas en base, como antes más allá de esto el algoritmo transcurre como los casos anteriores. La relación propuesta para este cubo, considerando el cambio de la dimensión DWFechas es:

```
DWVenta302 (id_articulo, importe, unidades, año, mes)
```

Como en los casos anteriores no hay diferencias entre la relación propuesta y la vista obtenida por el algoritmo a nivel de estructura. Como último caso de cubo se encuentra DWVenta4, en este no tenemos un resultado en [Per01]. El resultado obtenido por el algoritmo es:

```
create view DWVenta401 as
   select Facturas.cliente as id_cliente,
      Facturas.vendedor as id_vendedor,
      year(Facturas.fecha) as año,
      month(Facturas.fecha) as mes,
      sum(V_cantidad_de_facturas.cantidad_de_facturas)
         as cantidad_de_facturas
   from Facturas,
      (select factura, count(factura) as cantidad_de_facturas
      from Facturas
      group by factura) as V_cantidad_de_facturas
   where V_cantidad_de_facturas.factura = Facturas.factura
   group by Facturas.cliente, Facturas.vendedor, year(Facturas.fecha),
      month(Facturas.fecha);
```

En la ejecución del algoritmo para la vista anterior debemos observar que al llegar al cálculo de resumen se introduce en la cláusula FROM la vista originada por el NCalcME (V_cantindad_de_facturas). La vista del resumen se basa en que la relación a resumir es Facturas y que la operación de resumen es contar las facturas. Una vez construida la vista sólo resta utilizar la función de agregación de la medida para agregar el atributo V_cantidad_de_facturas.cantindad_de_facturas a la cláusula SELECT. Para unir el resultado del resumen con el esqueleto se utiliza el hecho de que la relación resumida ya está en el esqueleto, por lo cual se fabrica un link por la clave primaria de esta relación (Facturas.factura).

Notemos que como la operación de resumen del NCalcME y la operación de agregación de la medida son distintas no se produce un problema de asociatividad. Tampoco se produce un problema de múltiple conteo dado que el nivel de granularidad del resumen y el del esqueleto son el mismo (por factura).

E.6 Caso de Estudio en XML

Esta sección presenta el caso de estudio en términos del XML que utiliza el prototipo para representar la información sobre la base de datos fuente, CMDM, lineamientos, etc. Este planteo no difiere de lo presentado en las secciones anteriores pero tiene como cualidad el estilo de presentación XML.

```
<metadata>
  <database>
    <relation name="Departamentos">
      <attribute name="id_depto" type="int" key="true"/>
      <attribute name="nom_depto" type="varchar(50)" key="false"/>
      <attribute name="zona" type="varchar(50)" key="false"/>
    </relation>
    <relation name="Ciudades">
      <attribute name="id_ciudad" type="int" key="true"/>
      <attribute name="id_depto" type="int" key="true"/>
      <attribute name="nom_ciudad" type="varchar(50)" key="false"/>
      <attribute name="poblacion" type="long" key="false"/>
      <attribute name="clasificacion" type="varchar(50)" key="false"/>
    </relation>
    <relation name="Rubros">
      <attribute name="id_rubro" type="int" key="true"/>
      <attribute name="nom_rubro" type="varchar(50)" key="false"/>
    </relation>
    <relation name="Clientes">
      <attribute name="id_cliente" type="int" key="true"/>
      <attribute name="nombre" type="varchar(50)" key="false"/>
      <attribute name="direccion" type="varchar(50)" key="false"/>
      <attribute name="telefono" type="varchar(50)" key="false"/>
      <attribute name="ciudad" type="int" key="false"/>
```

```
    <attribute name="departamento" type="int" key="false"/>
    <attribute name="rubro" type="int" key="false"/>
    <attribute name="categoria" type="varchar(50)" key="false"/>
    <attribute name="fecha_alta" type="date" key="false"/>
  </relation>
  <relation name="Facturas">
    <attribute name="factura" type="long" key="true"/>
    <attribute name="fecha" type="date" key="false"/>
    <attribute name="cliente" type="int" key="false"/>
    <attribute name="vendedor" type="int" key="false"/>
  </relation>
  <relation name="Registros_Facturas">
    <attribute name="factura" type="long" key="true"/>
    <attribute name="articulo" type="int" key="true"/>
    <attribute name="importe" type="double" key="false"/>
    <attribute name="unidades" type="long" key="false"/>
  </relation>
  <relation name="Articulos">
    <attribute name="id_articulo" type="int" key="true"/>
    <attribute name="id_producto" type="int" key="false"/>
    <attribute name="id_tamaño" type="int" key="false"/>
  </relation>
  <relation name="Productos">
    <attribute name="id_producto" type="int" key="true"/>
    <attribute name="id_familia" type="int" key="false"/>
    <attribute name="id_duracion" type="int" key="false"/>
  </relation>
  <relation name="Codigos">
    <attribute name="tipo" type="varchar(50)" key="true"/>
    <attribute name="codigo" type="int" key="true"/>
    <attribute name="descripcion" type="varchar(50)" key="false"/>
  </relation>
  <relation name="Vendedores">
    <attribute name="id_vendedor" type="int" key="true"/>
    <attribute name="nombre" type="varchar(50)" key="false"/>
    <attribute name="dirección" type="varchar(50)" key="false"/>
    <attribute name="telefono" type="varchar(50)" key="false"/>
    <attribute name="especialidad" type="varchar(50)" key="false"/>
    <attribute name="antiguedad" type="int" key="false"/>
  </relation>
  <alias name="Codigos_Id_Articulo" relation="Codigos">
    <attribute name="tipo" relation-attribute="tipo"/>
    <attribute name="id_articulo" relation-attribute="codigo"/>
    <attribute name="descripcion" relation-attribute="descripcion"/>
  </alias>
  <alias name="Codigos_Id_Tamaño" relation="Codigos">
    <attribute name="tipo" relation-attribute="tipo"/>
    <attribute name="id_tamaño" relation-attribute="codigo"/>
    <attribute name="descripcion" relation-attribute="descripcion"/>
  </alias>
  <alias name="Codigos_Id_Producto" relation="Codigos">
    <attribute name="tipo" relation-attribute="tipo"/>
    <attribute name="id_producto" relation-attribute="codigo"/>
    <attribute name="descripcion" relation-attribute="descripcion"/>
  </alias>
  <alias name="Codigos_Id_Familia" relation="Codigos">
    <attribute name="tipo" relation-attribute="tipo"/>
    <attribute name="id_familia" relation-attribute="codigo"/>
    <attribute name="descripcion" relation-attribute="descripcion"/>
  </alias>
  <alias name="Codigos_Id_Duracion" relation="Codigos">
    <attribute name="tipo" relation-attribute="tipo"/>
    <attribute name="id_duracion" relation-attribute="codigo"/>
    <attribute name="descripcion" relation-attribute="descripcion"/>
  </alias>
  <link relation-1="Codigos_Id_Articulo" cardinality-1="1" relation-2="Articulos"
cardinality-2="1">
    <linked-attributes attribute-1="id_articulo" attribute-2="id_articulo"/>
  </link>
  <link relation-1="Codigos_Id_Tamaño" cardinality-1="1" relation-2="Articulos"
cardinality-2="N">
    <linked-attributes attribute-1="id_tamaño" attribute-2="id_tamaño"/>
  </link>
  <link relation-1="Codigos_Id_Producto" cardinality-1="1" relation-2="Productos"
cardinality-2="1">
    <linked-attributes attribute-1="id_producto" attribute-2="id_producto"/>
  </link>
```

```
   <link relation-1="Codigos_Id_Familia" cardinality-1="1" relation-2="Productos"
cardinality-2="N">
     <linked-attributes attribute-1="id_familia" attribute-2="id_familia"/>
   </link>
   <link relation-1="Codigos_Id_Duracion" cardinality-1="1" relation-2="Productos"
cardinality-2="N">
     <linked-attributes attribute-1="id_duracion" attribute-2="id_duracion"/>
   </link>
   <link relation-1="Departamentos" cardinality-1="1" relation-2="Ciudades" cardinality-
2="N">
     <linked-attributes attribute-1="id_depto" attribute-2="id_depto"/>
   </link>
   <link relation-1="Departamentos" cardinality-1="1" relation-2="Clientes" cardinality-
2="N">
     <linked-attributes attribute-1="id_depto" attribute-2="departamento"/>
   </link>
   <link relation-1="Rubros" cardinality-1="1" relation-2="Clientes" cardinality-2="N">
     <linked-attributes attribute-1="id_rubro" attribute-2="rubro"/>
   </link>
   <link relation-1="Ciudades" cardinality-1="1" relation-2="Clientes" cardinality-
2="N">
     <linked-attributes attribute-1="id_ciudad" attribute-2="ciudad"/>
     <linked-attributes attribute-1="id_depto" attribute-2="departamento"/>
   </link>
   <link relation-1="Clientes" cardinality-1="1" relation-2="Facturas" cardinality-
2="N">
     <linked-attributes attribute-1="id_cliente" attribute-2="cliente"/>
   </link>
   <link relation-1="Vendedores" cardinality-1="1" relation-2="Facturas" cardinality-
2="N">
     <linked-attributes attribute-1="id_vendedor" attribute-2="vendedor"/>
   </link>
   <link relation-1="Facturas" cardinality-1="1" relation-2="Registros_Facturas"
cardinality-2="N">
     <linked-attributes attribute-1="factura" attribute-2="factura"/>
   </link>
   <link relation-1="Articulos" cardinality-1="1" relation-2="Registros_Facturas"
cardinality-2="N">
     <linked-attributes attribute-1="id_articulo" attribute-2="articulo"/>
   </link>
   <link relation-1="Productos" cardinality-1="1" relation-2="Articulos" cardinality-
2="N">
     <linked-attributes attribute-1="id_producto" attribute-2="id_producto"/>
   </link>
  </database>
  <cmdm>
    <dimension name="clientes">
      <level name="cliente" key-type="absolute">
        <item name="id_cliente" key="true"/>
        <item name="cliente" key="false"/>
      </level>
      <level name="rubro" key-type="absolute">
        <item name="id_rubro" key="true"/>
        <item name="rubro" key="false"/>
      </level>
      <level name="ciudad" key-type="relative">
        <item name="id_ciudad" key="true"/>
        <item name="ciudad" key="false"/>
      </level>
      <level name="clasificacion" key-type="absolute">
        <item name="clasificacion" key="true"/>
      </level>
      <level name="departamento" key-type="absolute">
        <item name="id_departamento" key="true"/>
        <item name="departamento" key="false"/>
      </level>
      <level name="zona" key-type="absolute">
        <item name="zona" key="true"/>
      </level>
      <level-order level-source="cliente" level-target="rubro"/>
      <level-order level-source="cliente" level-target="ciudad"/>
      <level-order level-source="ciudad" level-target="clasificacion"/>
      <level-order level-source="ciudad" level-target="departamento" used-for-week-
key="true"/>
      <level-order level-source="departamento" level-target="zona"/>
    </dimension>
    <dimension name="articulos">
```

```
    <level name="articulo" key-type="absolute">
      <item name="id_articulo" key="true"/>
      <item name="articulo" key="false"/>
    </level>
    <level name="tamaño" key-type="absolute">
      <item name="tamaño" key="true"/>
    </level>
    <level name="producto" key-type="absolute">
      <item name="id_producto" key="true"/>
      <item name="producto" key="false"/>
    </level>
    <level name="familia" key-type="absolute">
      <item name="id_familia" key="true"/>
      <item name="familia" key="false"/>
    </level>
    <level name="duracion" key-type="absolute">
      <item name="duracion" key="true"/>
    </level>
    <level-order level-source="articulo" level-target="tamaño"/>
    <level-order level-source="articulo" level-target="producto"/>
    <level-order level-source="producto" level-target="familia"/>
    <level-order level-source="producto" level-target="duracion"/>
  </dimension>
  <dimension name="vendedores">
    <level name="vendedor" key-type="absolute">
      <item name="id_vendedor" key="true"/>
      <item name="vendedor" key="false"/>
    </level>
    <level name="especialidad" key-type="absolute">
      <item name="especialidad" key="true"/>
    </level>
    <level name="antiguedad" key-type="absolute">
      <item name="antiguedad" key="true"/>
    </level>
    <level-order level-source="vendedor" level-target="especialidad"/>
    <level-order level-source="vendedor" level-target="antiguedad"/>
  </dimension>
  <dimension name="fechas">
    <level name="año" key-type="absolute">
      <item name="año" key="true"/>
    </level>
    <level name="mes" key-type="relative">
      <item name="mes" key="true"/>
      <item name="nombre_mes" key="false"/>
    </level>
    <level-order level-source="mes" level-target="año"/>
  </dimension>
  <dimension name="cantidades">
    <level name="cantidades" key-type="absolute">
      <item name="importes" key="true"/>
      <item name="unidades" key="true"/>
    </level>
  </dimension>
  <dimension name="cantidad_de_facturas">
    <level name="cantidad_de_facturas" key-type="absolute">
      <item name="cantidad_de_facturas" key="true"/>
    </level>
  </dimension>
  <dimensional-relation name="ventas">
    <dimension-reference dimension="fechas"/>
    <dimension-reference dimension="vendedores"/>
    <dimension-reference dimension="clientes"/>
    <dimension-reference dimension="articulos"/>
    <dimension-reference dimension="cantidades"/>
  </dimensional-relation>
  <dimensional-relation name="facturas">
    <dimension-reference dimension="fechas"/>
    <dimension-reference dimension="vendedores"/>
    <dimension-reference dimension="clientes"/>
    <dimension-reference dimension="cantidad_de_facturas"/>
  </dimensional-relation>
</cmdm>
<design-lines>
  <fragment name="DWClientes" dimension="clientes">
    <level-reference level="cliente"/>
    <level-reference level="rubro"/>
  </fragment>
```

```
    <fragment name="DWCiudades" dimension="clientes">
      <level-reference level="ciudad"/>
      <level-reference level="clasificacion"/>
      <level-reference level="departamento"/>
      <level-reference level="zona"/>
    </fragment>
    <fragment name="DWArticulos" dimension="articulos">
      <level-reference level="articulo"/>
      <level-reference level="tamaño"/>
    </fragment>
    <fragment name="DWProductos" dimension="articulos">
      <level-reference level="producto"/>
      <level-reference level="duracion"/>
    </fragment>
    <fragment name="DWFamilias" dimension="articulos">
      <level-reference level="familia"/>
    </fragment>
    <fragment name="DWVendedores" dimension="vendedores">
      <level-reference level="vendedor"/>
      <level-reference level="antiguedad"/>
      <level-reference level="especialidad"/>
    </fragment>
    <fragment name="DWFechas" dimension="fechas">
      <level-reference level="año"/>
      <level-reference level="mes"/>
    </fragment>
    <cube name="DWVenta1" dimensional-relation="ventas">
      <level-reference dimension="fechas" level="mes" measure="false"/>
      <level-reference dimension="articulos" level="articulo" measure="false"/>
      <level-reference dimension="vendedores" level="vendedor" measure="false"/>
      <level-reference dimension="clientes" level="cliente" measure="false"/>
      <level-reference dimension="cantidades" level="cantidades" measure="true"/>
      <strip name="DWVenta1Banda01">
        <strip-condition><![CDATA[año >= 2001]]></strip-condition>
        <strip-item>año</strip-item>
      </strip>
      <strip name="DWVenta1Banda02">
        <strip-condition><![CDATA[año < 2001]]></strip-condition>
        <strip-item>año</strip-item>
      </strip>
    </cube>
    <cube name="DWVenta201" dimensional-relation="ventas">
      <level-reference dimension="fechas" level="mes" measure="false"/>
      <level-reference dimension="articulos" level="articulo" measure="false"/>
      <level-reference dimension="cantidades" level="cantidades" measure="true"/>
      <level-reference dimension="vendedores" level="vendedor" measure="false"/>
      <level-reference dimension="clientes" level="rubro" measure="false"/>
    </cube>
    <cube name="DWVenta301" dimensional-relation="ventas">
      <level-reference dimension="fechas" level="mes" measure="false"/>
      <level-reference dimension="articulos" level="articulo" measure="false"/>
      <level-reference dimension="cantidades" level="cantidades" measure="true"/>
    </cube>
    <cube name="DWVenta401" dimensional-relation="facturas">
      <level-reference dimension="fechas" level="mes" measure="false"/>
      <level-reference dimension="vendedores" level="vendedor" measure="false"/>
      <level-reference dimension="clientes" level="cliente" measure="false"/>
      <level-reference dimension="cantidad_de_facturas" level="cantidad_de_facturas"
measure="true"/>
    </cube>
  </design-lines>
  <mappings>
    <fragment-mapping dimension="clientes" fragment="DWClientes">
      <level-mapping level="clasificacion">
        <item-mapping item="clasificacion">
          <direct-me tab="Ciudades">
            <expr>clasificacion</expr>
            <patt tab="Ciudades" attribute="clasificacion"/>
          </direct-me>
        </item-mapping>
      </level-mapping>
      <level-mapping level="departamento">
        <item-mapping item="id_departamento">
          <direct-me tab="Ciudades">
            <expr>id_depto</expr>
            <patt tab="Ciudades" attribute="id_depto"/>
          </direct-me>
```

```
    </item-mapping>
  </level-mapping>
  <level-mapping level="ciudad">
    <item-mapping item="id_ciudad">
      <direct-me tab="Clientes">
        <expr>ciudad</expr>
        <patt tab="Clientes" attribute="ciudad"/>
      </direct-me>
    </item-mapping>
  </level-mapping>
  <level-mapping level="cliente">
    <item-mapping item="id_cliente">
      <direct-me tab="Clientes">
        <expr>id_cliente</expr>
        <patt tab="Clientes" attribute="id_cliente"/>
      </direct-me>
    </item-mapping>
    <item-mapping item="cliente">
      <one-calc-me>
        <expr>Cstr(id_cliente) ++ ' - ' ++ nombre</expr>
        <tab>Clientes</tab>
        <patt tab="Clientes" attribute="id_cliente"/>
        <patt tab="Clientes" attribute="nombre"/>
      </one-calc-me>
    </item-mapping>
  </level-mapping>
  <level-mapping level="rubro">
    <item-mapping item="id_rubro">
      <direct-me tab="Rubros">
        <expr>id_rubro</expr>
        <patt tab="Rubros" attribute="id_rubro"/>
      </direct-me>
    </item-mapping>
    <item-mapping item="rubro">
      <direct-me tab="Rubros">
        <expr>nom_rubro</expr>
        <patt tab="Rubros" attribute="nom_rubro"/>
      </direct-me>
    </item-mapping>
  </level-mapping>
  <restrictions>Clientes.fecha_alta is not null</restrictions>
</fragment-mapping>
<fragment-mapping dimension="clientes" fragment="DWCiudades">
  <level-mapping level="ciudad">
    <item-mapping item="id_ciudad">
      <direct-me tab="Ciudades">
        <expr>id_ciudad</expr>
        <patt tab="Ciudades" attribute="id_ciudad"/>
      </direct-me>
    </item-mapping>
    <item-mapping item="ciudad">
      <direct-me tab="Ciudades">
        <expr>nom_ciudad</expr>
        <patt tab="Ciudades" attribute="nom_ciudad"/>
      </direct-me>
    </item-mapping>
  </level-mapping>
  <level-mapping level="departamento">
    <item-mapping item="id_departamento">
      <direct-me tab="Departamentos">
        <expr>id_depto</expr>
        <patt tab="Departamentos" attribute="id_depto"/>
      </direct-me>
    </item-mapping>
    <item-mapping item="departamento">
      <direct-me tab="Departamentos">
        <expr>nom_depto</expr>
        <patt tab="Departamentos" attribute="nom_depto"/>
      </direct-me>
    </item-mapping>
  </level-mapping>
  <level-mapping level="clasificacion">
    <item-mapping item="clasificacion">
      <direct-me tab="Ciudades">
        <expr>clasificacion</expr>
        <patt tab="Ciudades" attribute="clasificacion"/>
      </direct-me>
```

```
      </item-mapping>
     </level-mapping>
     <level-mapping level="zona">
      <item-mapping item="zona">
       <direct-me tab="Departamentos">
        <expr>zona</expr>
        <patt tab="Departamentos" attribute="zona"/>
       </direct-me>
      </item-mapping>
     </level-mapping>
    </fragment-mapping>
    <fragment-mapping dimension="articulos" fragment="DWArticulos">
     <level-mapping level="producto">
      <item-mapping item="id_producto">
       <direct-me tab="Articulos">
        <expr>id_producto</expr>
        <patt tab="Articulos" attribute="id_producto"/>
       </direct-me>
      </item-mapping>
     </level-mapping>
     <level-mapping level="articulo">
      <item-mapping item="id_articulo">
       <direct-me tab="Codigos_Id_Articulo">
        <expr>id_articulo</expr>
        <patt tab="Codigos_Id_Articulo" attribute="id_articulo"/>
       </direct-me>
      </item-mapping>
      <item-mapping item="articulo">
       <direct-me tab="Codigos_Id_Articulo">
        <expr>descripcion</expr>
        <patt tab="Codigos_Id_Articulo" attribute="descripcion"/>
       </direct-me>
      </item-mapping>
     </level-mapping>
     <level-mapping level="tamaño">
      <item-mapping item="tamaño">
       <direct-me tab="Codigos_Id_Tamaño">
        <expr>descripcion</expr>
        <patt tab="Codigos_Id_Tamaño" attribute="descripcion"/>
       </direct-me>
      </item-mapping>
     </level-mapping>
     <restrictions>Codigos_Id_Articulo.Tipo='ART' and
Codigos_Id_Tamaño.Tipo='TAM'</restrictions>
    </fragment-mapping>
    <fragment-mapping dimension="articulos" fragment="DWProductos">
     <level-mapping level="familia">
      <item-mapping item="id_familia">
       <direct-me tab="Productos">
        <expr>id_familia</expr>
        <patt tab="Productos" attribute="id_familia"/>
       </direct-me>
      </item-mapping>
     </level-mapping>
     <level-mapping level="producto">
      <item-mapping item="id_producto">
       <direct-me tab="Codigos_Id_Producto">
        <expr>id_producto</expr>
        <patt tab="Codigos_Id_Producto" attribute="id_producto"/>
       </direct-me>
      </item-mapping>
      <item-mapping item="producto">
       <direct-me tab="Codigos_Id_Producto">
        <expr>descripcion</expr>
        <patt tab="Codigos_Id_Producto" attribute="descripcion"/>
       </direct-me>
      </item-mapping>
     </level-mapping>
     <level-mapping level="duracion">
      <item-mapping item="duracion">
       <direct-me tab="Codigos_Id_Duracion">
        <expr>descripcion</expr>
        <patt tab="Codigos_Id_Duracion" attribute="descripcion"/>
       </direct-me>
      </item-mapping>
     </level-mapping>
```

```
    <restrictions>Codigos_Id_Producto.Tipo='PRO' and
Codigos_Id_Duracion.Tipo='DUR'</restrictions>
   </fragment-mapping>
   <fragment-mapping dimension="articulos" fragment="DWFamilias">
    <level-mapping level="familia">
     <item-mapping item="id_familia">
      <direct-me tab="Codigos_Id_Familia">
       <expr>id_familia</expr>
       <patt tab="Codigos_Id_Familia" attribute="id_familia"/>
      </direct-me>
     </item-mapping>
     <item-mapping item="familia">
      <direct-me tab="Codigos_Id_Familia">
       <expr>descripcion</expr>
       <patt tab="Codigos_Id_Familia" attribute="descripcion"/>
      </direct-me>
     </item-mapping>
    </level-mapping>
    <restrictions>Codigos_Id_Familia.Tipo='FAM'</restrictions>
   </fragment-mapping>
   <fragment-mapping dimension="vendedores" fragment="DWVendedores">
    <level-mapping level="vendedor">
     <item-mapping item="id_vendedor">
      <direct-me tab="Vendedores">
       <expr>id_vendedor</expr>
       <patt tab="Vendedores" attribute="id_vendedor"/>
      </direct-me>
     </item-mapping>
     <item-mapping item="vendedor">
      <direct-me tab="Vendedores">
       <expr>nombre</expr>
       <patt tab="Vendedores" attribute="nombre"/>
      </direct-me>
     </item-mapping>
    </level-mapping>
    <level-mapping level="antiguedad">
     <item-mapping item="antiguedad">
      <one-calc-me>
       <expr>Vendedores.antiguedad / 12</expr>
       <tab>Vendedores</tab>
       <patt tab="Vendedores" attribute="antiguedad"/>
      </one-calc-me>
     </item-mapping>
    </level-mapping>
    <level-mapping level="especialidad">
     <item-mapping item="especialidad">
      <direct-me tab="Vendedores">
       <expr>especialidad</expr>
       <patt tab="Vendedores" attribute="especialidad"/>
      </direct-me>
     </item-mapping>
    </level-mapping>
    <restrictions>Vendedores.antiguedad &gt;=12</restrictions>
   </fragment-mapping>
   <fragment-mapping dimension="fechas" fragment="DWFechas">
    <level-mapping level="mes">
     <item-mapping item="mes">
      <one-calc-me>
       <expr>month(Facturas.fecha)</expr>
       <tab>Facturas</tab>
       <patt tab="Facturas" attribute="fecha"/>
      </one-calc-me>
     </item-mapping>
     <item-mapping item="nombre_mes">
      <extern-me>
       <constant ind="Constant">
        <expr>null</expr>
       </constant>
      </extern-me>
     </item-mapping>
    </level-mapping>
    <level-mapping level="año">
     <item-mapping item="año">
      <one-calc-me>
       <expr>year(Facturas.fecha)</expr>
       <tab>Facturas</tab>
       <patt tab="Facturas" attribute="fecha"/>
```

```
      </one-calc-me>
    </item-mapping>
  </level-mapping>
</fragment-mapping>
<base-cube-mapping dimensional-relation="ventas" cube="DWVenta1">
  <level-mapping dimension="fechas" level="año">
    <item-mapping item="año">
      <one-calc-me>
        <expr>year(Facturas.fecha)</expr>
        <tab>Facturas</tab>
        <patt tab="Facturas" attribute="fecha"/>
      </one-calc-me>
    </item-mapping>
  </level-mapping>
  <level-mapping dimension="fechas" level="mes">
    <item-mapping item="mes">
      <one-calc-me>
        <expr>month(Facturas.fecha)</expr>
        <tab>Facturas</tab>
        <patt tab="Facturas" attribute="fecha"/>
      </one-calc-me>
    </item-mapping>
  </level-mapping>
  <level-mapping dimension="clientes" level="cliente">
    <item-mapping item="id_cliente">
      <direct-me tab="Facturas">
        <expr>cliente</expr>
        <patt tab="Facturas" attribute="cliente"/>
      </direct-me>
    </item-mapping>
  </level-mapping>
  <level-mapping dimension="vendedores" level="vendedor">
    <item-mapping item="id_vendedor">
      <direct-me tab="Facturas">
        <expr>vendedor</expr>
        <patt tab="Facturas" attribute="vendedor"/>
      </direct-me>
    </item-mapping>
  </level-mapping>
  <level-mapping dimension="articulos" level="articulo">
    <item-mapping item="id_articulo">
      <direct-me tab="Registros_Facturas">
        <expr>articulo</expr>
        <patt tab="Registros_Facturas" attribute="articulo"/>
      </direct-me>
    </item-mapping>
  </level-mapping>
  <measure-level-mapping dimension="cantidades" level="cantidades">
    <item-mapping item="importes">
      <direct-me tab="Registros_Facturas">
        <expr>importe</expr>
        <patt tab="Registros_Facturas" attribute="importe"/>
      </direct-me>
      <rollup-expression>sum(importes)</rollup-expression>
    </item-mapping>
    <item-mapping item="unidades">
      <direct-me tab="Registros_Facturas">
        <expr>unidades</expr>
        <patt tab="Registros_Facturas" attribute="unidades"/>
      </direct-me>
      <rollup-expression>sum(unidades)</rollup-expression>
    </item-mapping>
  </measure-level-mapping>
</base-cube-mapping>
<base-cube-mapping dimensional-relation="facturas" cube="DWVenta401">
  <level-mapping dimension="fechas" level="año">
    <item-mapping item="año">
      <one-calc-me>
        <expr>year(Facturas.fecha)</expr>
        <tab>Facturas</tab>
        <patt tab="Facturas" attribute="fecha"/>
      </one-calc-me>
    </item-mapping>
  </level-mapping>
  <level-mapping dimension="fechas" level="mes">
    <item-mapping item="mes">
      <one-calc-me>
```

```
        <expr>month(Facturas.fecha)</expr>
        <tab>Facturas</tab>
        <patt tab="Facturas" attribute="fecha"/>
       </one-calc-me>
      </item-mapping>
     </level-mapping>
     <level-mapping dimension="clientes" level="cliente">
      <item-mapping item="id_cliente">
       <direct-me tab="Facturas">
        <expr>cliente</expr>
        <patt tab="Facturas" attribute="cliente"/>
       </direct-me>
      </item-mapping>
     </level-mapping>
     <level-mapping dimension="vendedores" level="vendedor">
      <item-mapping item="id_vendedor">
       <direct-me tab="Facturas">
        <expr>vendedor</expr>
        <patt tab="Facturas" attribute="vendedor"/>
       </direct-me>
      </item-mapping>
     </level-mapping>
     <measure-level-mapping dimension="cantidad_de_facturas"
level="cantidad_de_facturas">
      <item-mapping item="cantidad_de_facturas">
       <n-calc-me tab="Facturas">
        <expr>count(factura)</expr>
        <patt tab="Facturas" attribute="factura"/>
       </n-calc-me>
       <rollup-expression>sum(cantidad_de_facturas)</rollup-expression>
      </item-mapping>
     </measure-level-mapping>
   </base-cube-mapping>
   <recursive-cube-mapping dimensional-relation="ventas" cube="DWVenta201" base-
cube="DWVenta1">
     <partial-drill-up dimension="clientes">
      <source-level-mapping level="cliente">
       <item-mapping item="id_cliente">
        <direct-me tab="Clientes">
         <expr>id_cliente</expr>
         <patt tab="Clientes" attribute="id_cliente"/>
        </direct-me>
       </item-mapping>
      </source-level-mapping>
      <target-level-mapping level="rubro">
       <item-mapping item="id_rubro">
        <direct-me tab="Clientes">
         <expr>rubro</expr>
         <patt tab="Clientes" attribute="rubro"/>
        </direct-me>
       </item-mapping>
      </target-level-mapping>
     </partial-drill-up>
     <measure-level-rollup dimension="cantidades" level="cantidades">
      <measure-item-rollup item="importes" rollup-expression="sum(importes)"/>
      <measure-item-rollup item="unidades" rollup-expression="sum(unidades)"/>
     </measure-level-rollup>
   </recursive-cube-mapping>
   <recursive-cube-mapping dimensional-relation="ventas" cube="DWVenta301" base-
cube="DWVenta1">
     <total-drill-up dimension="clientes"/>
     <total-drill-up dimension="vendedores"/>
     <measure-level-rollup dimension="cantidades" level="cantidades">
      <measure-item-rollup item="importes" rollup-expression="sum(importes)"/>
      <measure-item-rollup item="unidades" rollup-expression="sum(unidades)"/>
     </measure-level-rollup>
   </recursive-cube-mapping>
  </mappings>
</metadata>
```

Índice

Glosario

Base de Datos Multidimensional: Una base de datos multidimensional es tal que en lugar de almacenar la información en múltiples tablas bidimensionales (como lo hacen las bases de datos relacionales), representa las claves de las entidades relacionadas como diferentes dimensiones. 2

Bases de Datos Operacionales: Son las diferentes BD de una empresa u organización, asociadas a las actividades o sistemas que permiten el ingreso confiable de información y el procesamiento eficiente de transacciones. 1

BD: Abreviatura de Base de Datos. 1

BDF: Abreviatura de Base de Datos Fuente. 3

BDM: Abreviatura de Base de Datos Multidimensional. 2

BDO: Abreviatura de Bases de Datos Operacionales..... 1

CMDM: Modelo multidimensional definido por [Car00]..... 3

CSI: Grupo de Concepción de Sistemas de Información. 3

Cubo Base: Cubo definido directamente en función de la BDF, a diferencia de los Cubos Recursivos..... 34

Cubos Recursivos: Son cubos definidos en función de otros cubos, a diferencia de los cubos base que lo hacen refiriendo a la BDF. 34

Data Warehouse: Es una colección de datos orientadas a un dominio, integrado, no volátil y variable en el tiempo que ayuda a la toma de decisiones de la empresa u organización..... 2

Data Warehousing: Es el término generalmente usado para definir la tecnología que provee la infraestructura de software utilizada para sistemas de soporte a la toma de decisiones y aplicaciones OLAP..... 1

DM: Abreviatura de Data Mart. 2

Eager Aggregation: Estrategia para bajar los group-by y de esta forma optimizar sentencias..... 53

Expresiones de correspondencia: Las expresiones de correspondencia son, las expresiones, con las que se definen las correspondencias (DirectME, 1CalcME, etc.). 125

Fragmentación de Cubos: Posibilidad de materializar cubos en múltiples relaciones, *Véase* Franjas del Cubo. 34

Fragmento Ficticio: Fragmento que no es parte de la definición sino que se crea temporalmente para solucionar aspectos internos del algoritmo de generación de esquema. 34

Franjas del Cubo: Divisiones físicas (distintas relaciones) de los datos contenidos en el cubo. Ej.: Cubo Ventas, Franja: Datos Históricos (año < año_actual), Franja: Datos Actuales (año = año_actual). 35

MapAttributes: Es una función que devuelve los atributos que tienen correspondencia a ítems de un conjunto dado. 78

MapTables: Es una función que devuelve el conjunto de tablas que tienen correspondencias con un conjunto de ítems dados. 78

Metadata: La Metadata es información estructurada y codificada que describe características de la información. 2

Múltiple Conteo: Problema asociado con contar o agregar tuplas de forma incorrecta. 44

ObjectItems: Es una función que devuelve los ítems asociados al objeto indicado. 78

ObjectKeyItems: Es una función que devuelve los ítems que identifican al objeto dado. 78

OLTP: Sistema que permite el ingreso confiable de información y el procesamiento eficiente de transacciones. Abreviación por sus siglas en ingles (On Line Transaction Processing) 1

PODS: Conferencia PODS (Principles of Database Systems), http://www.sigmod.org/pods. 2

Predicado: Los predicados son utilizados en [Per01] para por ejemplo denotar condiciones de join en la construcción SchLinks. 85

Primitivas: Operaciones de transformación de esquemas y datos definidas en [Mar00]. 7

RDBMS: Abreviatura del inglés: Relational Database Management System (Sistema Administrador de Base de Datos Relacionales). 4

SchCFragmentantion: Es una función que a cada cubo le hace corresponder un conjunto de franjas en las cuales se fragmenta el cubo. 78

SchCMappings: Es una función que a cada cubo le hace corresponder un mapeo base o un mapeo recursivo de cubo. 78

SchCubes: Conjunto de todos los cubos del esquema conceptual definidos por el diseñador. 77

SchFMappings: Es una función que a cada fragmento le hace corresponder una función de correspondencia y opcionalmente una condición. 77

SchFragments: Abreviatura que representa el conjunto todos los fragmentos de dimensión del esquema conceptual. 77

SchLinks: Es uno de los componentes de la definición de la base de datos fuente y representa el conjunto de links entre tablas que define como se realizan los joins. ... 77

SIGMOD: Conferencia SIGMOD (Special Interest Group on Management of Data), http://www.sigmod.org. 2

Sistemas Legados: Un sistema legado es un sistema o aplicación que continua siendo utilizada porque el usuario (típicamente una organización) no quiere reemplazarlo o rediseñarlo. Mucha gente habla de éstos como sistemas anticuados. 2

Traza de Diseño: Secuencia de Primitivas que describe como éstas se conectan para realizar la transformación. Éstas se describen en [Mar00]. 7

Vista unificada: Vista que resume todos los pasos del enfoque Naive para obtener los datos de un fragmento de dimensión o franja de cubo. 47