Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Pedeciba Informática

Diseño Lógico de Data Warehouses a partir de Esquemas Conceptuales Multidimensionales

Verónika Peralta

Tesis de Maestría
Noviembre, 2001

Tutor: Dr. Raúl Ruggia.

Resumen

Un Data Warehouse (DW) es una base de datos que almacena información para la toma de decisiones. Las características de los DWs hacen que los modelos de datos y las estrategias de diseño utilizadas para bases de datos operacionales generalmente no sean aplicables para el diseño de un DW. Esto ha motivado el desarrollo de nuevas técnicas y estrategias de diseño.

Esta tesis trata los problemas de diseño lógico de un DW. Concretamente se propone un proceso de diseño para generar el esquema lógico relacional del DW a partir de un esquema conceptual multidimensional y una base de datos fuente integrada. La generación se lleva a cabo aplicando transformaciones de esquemas al esquema relacional de la base de datos fuente. Se propone un algoritmo que determina qué transformaciones se deben aplicar, basado en un conjunto de reglas de diseño. En la evaluación de las reglas intervienen el esquema conceptual, la base de datos fuente, correspondencias entre ellos y estrategias de diseño que indican restricciones de performance y almacenamiento.

El trabajo incluye la prototipación de una herramienta CASE que asiste al diseñador en la construcción de un DW relacional a partir de una especificación conceptual.

Agradecimientos

Quisiera agradecer a mi tutor, el Dr. Raúl Ruggia, quien me guió durante todo el proceso de investigación y escritura de esta tesis, y a los miembros del tribunal, Drs. Alejandro Gutierrez, Alvaro Tasistro y Alejandro Vaisman.

También quisiera agradecer a todos los integrantes del grupo CSI por sus invalorables aportes a este trabajo y por todo el apoyo que me brindaron.

Abstract

A Data Warehouse (DW) is a database that stores information oriented to satisfy decision-making requests. DW features are very different from conventional database ones, so that the data models and design strategies for the latter are generally not to be applicable for DW. These differences have motivated the development of new design techniques and strategies.

This work addresses DW logical design. Concretely, we propose a design process for generating the DW relational schema starting from a conceptual multidimensional schema and an integrated source database. The generation is carried out applying schema transformations to the relational shcema of the source database. We propose an algorithm to determine which transformations must be applied, which is based on a set of design rules. The rules involve the conceptual schema, the source database, correspondences between these schemes and design strategies related with performance and storage constraints.

This works includes the prototype of a CASE tool which assists the designer in the construction of a relational DW from a conceptual specification.

Índice

CAPÍTULO I - INTRODUCCIÓN 1

1. CONTEXTO 1
   - *1.1. Data Warehouses 1*
   - *1.2. Técnicas de Diseño 2*
2. MOTIVACIÓN 3
3. OBJETIVOS 3
4. SOLUCIÓN PROPUESTA 4
5. APORTES 6
6. ORGANIZACIÓN DEL DOCUMENTO 6

CAPÍTULO II - ESTADO DEL ARTE 7

1. INTRODUCCIÓN 7
2. DISEÑO DE DATA WAREHOUSES 7
   - *2.1. Diseño Conceptual 8*
   - *2.2. Diseño Lógico 9*
3. GENERACIÓN DEL ESQUEMA LÓGICO A PARTIR DEL ESQUEMA CONCEPTUAL 10
4. OTRAS TÉCNICAS DE DISEÑO 12
   - *4.1. Reglas de Diseño 12*
   - *4.2. Correspondencias entre esquemas 12*
5. CONCLUSIONES 13

CAPÍTULO III - DEFINICIÓN DE LINEAMIENTOS Y MAPEOS 15

1. INTRODUCCIÓN 15
2. DESCRIPCIÓN INFORMAL 17
3. NOTACIÓN 19
4. ESPECIFICACIÓN DEL ESQUEMA CONCEPTUAL 21
   - *4.1. Restricción a los tipos de datos del modelo CMDM 21*
   - *4.2. Otras restricciones sobre CMDM 23*
5. LINEAMIENTOS DE DISEÑO 24
   - *5.1. Materialización de Relaciones 24*
   - *5.2. Fragmentación Vertical de Dimensiones 25*
   - *5.3. Fragmentación Horizontal de Cubos 27*
   - *5.4. Definición de Lineamientos 29*
6. ESQUEMA INTERMEDIO 30
7. ESPECIFICACIÓN DE LA BASE DE DATOS FUENTE 31
8. MAPEOS ENTRE EL ESQUEMA INTERMEDIO Y LA BASE DE DATOS FUENTE 32
   - *8.1. Expresiones de Mapeo 32*
   - *8.2. Funciones de Mapeo 33*
   - *8.3. Algunas definiciones 35*
   - *8.4. Mapeo de Fragmentos 37*
   - *8.5. Mapeo de Cubos 38*
9. CONCLUSIONES 41

CAPÍTULO IV - GENERACIÓN DEL ESQUEMA LÓGICO DEL DW 43

1. INTRODUCCIÓN 43
2. DESCRIPCIÓN INFORMAL 43
3. PRIMITIVAS DE TRANSFORMACIÓN DE ESQUEMAS 48
4. REGLAS DE DISEÑO 51
   - *4.1. Descripción de las Reglas 52*
   - *4.2. Especificación de las Reglas 53*
5. ESPECIFICACIÓN DE UN ALGORITMO DE GENERACIÓN DEL ESQUEMA RELACIONAL 77
   - *5.1. Construcción de Tablas de Dimensión (para fragmentos de dimensiones) 78*
   - *5.2. Construcción de Tablas de Hechos (para cubos con mapeo base) 82*

5.3. Construcción de Tablas de Hechos (para cubos con mapeo recursivo) *85*
5.4. Construcción de Tablas de Hechos (para franjas de cubos) *87*
6. CONCLUSIONES 87

CAPÍTULO V - PROTOTIPACIÓN DE UNA HERRAMIENTA CASE 89

1. INTRODUCCIÓN 89
2. ARQUITECTURA DEL AMBIENTE CASE 90
3. DESCRIPCIÓN DE LA HERRAMIENTA 90
4. DESARROLLO DEL PROTOTIPO 95
5. CONCLUSIONES 96

CAPÍTULO VI - CONCLUSIONES 97

1. CONCLUSIONES 97
2. APORTES Y LIMITACIONES 99
3. TRABAJO EN CURSO Y FUTURO 100

ANEXOS 103

ANEXO 1. ESPECIFICACIÓN E INSTANCIACIÓN DE CMDM 103
1.1. Introducción *103*
1.2. Especificación de CMDM *103*
1.3. Cambios en la notación *104*
1.4. Instanciación para tipos estructurados *104*
1.5. Especificación resultante *108*
1.6. Conclusiones *109*
ANEXO 2. PARSING DEL ESQUEMA CONCEPTUAL 110
2.1. Introducción *110*
2.2. Componentes *110*
2.3. Orden de Niveles *111*
2.4. Claves de Niveles *112*
2.5. Conclusiones *112*
ANEXO 3. VINCULACIÓN DE TABLAS: LINKS Y ALIAS 113
3.1. Introducción *113*
3.2. Links *113*
3.3. Propiedades de los Links *114*
3.4. Conclusiones *116*
ANEXO 4. FUNCIONES AUXILIARES 117
4.1. Introducción *117*
4.2. Ítems del esquema intermedio *117*
4.3. Funciones de actualización de tablas *118*
4.4. Funciones de actualización de mapeos *119*
ANEXO 5. DESCRIPCIÓN DE LAS PRIMITIVAS 120
5.1. Introducción *120*
5.2. Descripción de las primitivas propuestas por Marotta *121*
5.3. Descripción de las nuevas primitivas *125*
5.4. Especificación de las nuevas primitivas *126*
5.5. Conclusiones *127*
ANEXO 6. DIAGRAMAS DE CLASES DE LA HERRAMIENTA 128
ANEXO 7. UN CASO DE ESTUDIO 132
7.1. Presentación del caso de estudio *132*
7.2. Lineamientos *135*
7.3. Mapeos *138*
7.4. Aplicación del algoritmo *142*
7.5. Conclusiones *148*

BIBLIOGRAFÍA 149

Índice de Figuras

FIGURA 1 – NIVELES DE DISEÑO....2
FIGURA 2 – PROCESO DE DISEÑO LÓGICO....4
FIGURA 3 – PROCESO DE DISEÑO PROPUESTO....5
FIGURA 4 – PROCESO DE DISEÑO DE UN DATA WAREHOUSE....8
FIGURA 5 – CLASIFICACIÓN DE DIFERENTES PROPUESTAS DE DISEÑO LÓGICO....14
FIGURA 6 – COMPARACIÓN DE DIFERENTES PROPUESTAS DE DISEÑO LÓGICO. 14
FIGURA 7 – PROCESO DE DISEÑO LÓGICO....16
FIGURA 8 – ESQUEMA CONCEPTUAL: A) DIMENSIÓN CLIENTES, B) DIMENSIÓN FECHAS Y C) RELACIÓN DIMENSIONAL ATENCIÓN....17
FIGURA 9 – DIFERENTES ESTRATEGIAS DE GENERACIÓN DE ESTRUCTURAS RELACIONALES....18
FIGURA 10 – BASE FUENTE 18
FIGURA 11 – NOTACIÓN GRÁFICA DE CMDM: A) DIMENSIONES Y B) RELACIONES DIMENSIONALES....21
FIGURA 12 – DEFINICIÓN DE NIVELES: A) COMPARTIENDO ÍTEMS, B) CON ÍTEMS DIFERENTES 23
FIGURA 13 – MATERIALIZACIÓN DE RELACIONES (CUBOS)....25
FIGURA 14 – FRAGMENTACIÓN VERTICAL DE DIMENSIONES 26
FIGURA 15 – FRAGMENTACIÓN HORIZONTAL DE CUBOS 28
FIGURA 16 – LINKS ENTRE TABLAS FUENTES. 31
FIGURA 17 – REPRESENTACIÓN GRÁFICA DE UNA FUNCIÓN DE MAPEO 34
FIGURA 18 – REPRESENTACIÓN GRÁFICA DE CONDICIONES DE MAPEO 35
FIGURA 19 – FRAGMENTACIÓN DE LA DIMENSIÓN PRODUCTOS....36
FIGURA 20 – REPRESENTACIÓN GRÁFICA DE UN MAPEO BASE DE CUBO....39
FIGURA 21 – REPRESENTACIÓN GRÁFICA DE UN MAPEO RECURSIVO DE CUBO 40
FIGURA 22 – LINEAMIENTOS DE DISEÑO....44
FIGURA 23 – MAPEO DE LOS FRAGMENTOS....44
FIGURA 24 – MAPEO BASE DEL CUBO DETALLE....45
FIGURA 25 – MAPEO RECURSIVO DEL CUBO RESUMEN 45
FIGURA 26 – REGLAS DE DISEÑO....46
FIGURA 27 – PRIMITIVAS DE TRANSFORMACIÓN DE ESQUEMAS 49
FIGURA 28 – NUEVAS PRIMITIVAS DE TRANSFORMACIÓN DE ESQUEMAS 49
FIGURA 29 – TRAZA DE APLICACIÓN DE LAS PRIMITIVAS....51
FIGURA 30 – FORMATO DE ESPECIFICACIÓN DE LAS REGLAS....54
FIGURA 31 – MAPEO DE LA DIMENSIÓN CLIENTES ANTES DE APLICAR JOIN 55
FIGURA 32 – MAPEO DE LA DIMENSIÓN CLIENTES DESPUÉS DE APLICAR JOIN....55
FIGURA 33 – MAPEO DE LA DIMENSIÓN CLIENTES: A) ANTES Y, B) DESPUÉS DE APLICAR RENAME 57
FIGURA 34 – MAPEO DE LA DIMENSIÓN FECHAS: A) ANTES Y, B) DESPUÉS DE APLICAR CALCULATE....59
FIGURA 35 – MAPEO DE LA DIMENSIÓN PRODUCTOS: A) ANTES Y, B) DESPUÉS DE APLICAR NEW ATTRIBUTE....62
FIGURA 36 – MAPEO DE LA DIMENSIÓN FECHAS: A) ANTES Y, B) DESPUÉS DE APLICAR GROUP 66
FIGURA 37 – MAPEO DEL CUBO VENTA-1 69
FIGURA 38 – MAPEO DEL CUBO VENTA-2 A PARTIR DEL CUBO VENTA-1, ANTES DE APLICAR DRILL-UP....70
FIGURA 39 – MAPEO DEL CUBO VENTA-2 DESPUÉS DE APLICAR DRILL-UP 70
FIGURA 40 – MAPEO DE LA DIMENSIÓN GEOGRAFÍA: A) ANTES Y, B) DESPUÉS DE APLICAR PRIMARY KEY73
FIGURA 41 – MAPEO DE LA DIMENSIÓN CLIENTES 75
FIGURA 42 – ARQUITECTURA DEL AMBIENTE CASE....90
FIGURA 43 – PANTALLA DE DEFINICIÓN DE LINEAMIENTOS 92
FIGURA 44 – PANTALLA DE DEFINICIÓN DE MAPEOS....92
FIGURA 45 – EJECUCIÓN PASO A PASO DEL ALGORITMO 93
FIGURA 46 – VISUALIZACIÓN DE LAS TABLAS DEL DW....93
FIGURA 47 – VISUALIZACIÓN DE FUNCIONES DE MAPEO....94
FIGURA 48 – VISUALIZACIÓN DE LA TRAZA DE DISEÑO....94
FIGURA 49 – ARQUITECTURA DE DWDESIGNER....95
FIGURA 50 – LINK ENTRE DOS TABLAS 114
FIGURA 51 – PROBLEMA DE CICLOS EN EL GRAFO DE LINKS....114
FIGURA 52 – PROBLEMA DE MÚLTIPLES LINKS ENTRE DOS TABLAS....115

FIGURA 53 – SOLUCIÓN DEL DOBLE VÍNCULO ENTRE TABLAS 115
FIGURA 54 – EJEMPLO DE AUTOMATIZACIÓN 120
FIGURA 55 – FRAME PRINCIPAL 128
FIGURA 56 – MÁQUINA VIRTUAL 129
FIGURA 57 – DIRECTORIOS: A) DE REGLAS DE DISEÑO Y, B) DE PASOS 129
FIGURA 58 – ESQUEMA INTERMEDIO Y ESQUEMA DE MAPEOS 130
FIGURA 59 – ESQUEMA INTERMEDIO 130
FIGURA 60 – ESQUEMA DE MAPEOS 131
FIGURA 61 – OBJETOS COLECCIONABLES 131
FIGURA 62 – RELACIONES DIMENSIONALES 132
FIGURA 63 – DIMENSIONES Y NIVELES 133
FIGURA 64 – LINKS ENTRE TABLAS DE LA BASE FUENTE 134
FIGURA 65 – ALIAS Y RE-DEFINICIÓN DE LOS LINKS (FRACCIÓN DE LOS LINKS) 135
FIGURA 66 – CUBOS DEFINIDOS 136
FIGURA 67 – FRANJAS DEFINIDAS 136
FIGURA 68 – FRAGMENTOS DEFINIDOS 137
FIGURA 69 – MAPEO DEL FRAGMENTO ROSA DE LA DIMENSIÓN CLIENTES 138
FIGURA 70 – MAPEO DEL FRAGMENTO CELESTE DE LA DIMENSIÓN CLIENTES 138
FIGURA 71 – MAPEO DEL FRAGMENTO ROSA DE LA DIMENSIÓN ARTÍCULOS 139
FIGURA 72 – MAPEO DEL FRAGMENTO AZUL DE LA DIMENSIÓN ARTÍCULOS 139
FIGURA 73 – MAPEO DEL FRAGMENTO VERDE DE LA DIMENSIÓN ARTÍCULOS 140
FIGURA 74 – MAPEO DEL FRAGMENTO DE LA DIMENSIÓN VENDEDORES 140
FIGURA 75 – MAPEO DEL FRAGMENTO DE LA DIMENSIÓN FECHAS 140
FIGURA 76 – MAPEO DEL CUBO VENTA-1 141
FIGURA 77 – MAPEO DEL CUBO VENTA-2 141
FIGURA 78 – MAPEO DEL CUBO VENTA-3 141
FIGURA 79 – ESQUEMA LÓGICO DEL DW 148

Índice de Definiciones

DEFINICIÓN 1 – CONCEPTUALSCHEMAS 22
DEFINICIÓN 2 – SCHCUBES 25
DEFINICIÓN 3 – SCHDFRAGMENTATION 26
DEFINICIÓN 4 – SCHCFRAGMENTATION 28
DEFINICIÓN 5 – INTERMEDIATESCHEMA 30
DEFINICIÓN 6 – LOGICALSCHEMA 31
DEFINICIÓN 7 – MAPEXPR 33
DEFINICIÓN 8 – MAPPINGS 33
DEFINICIÓN 9 – MAPATTRIBUTES 35
DEFINICIÓN 10 – MAPTABLES 36
DEFINICIÓN 11 – SCHFMAPPINGS 37
DEFINICIÓN 12 – DRILLUPS 38
DEFINICIÓN 13 – SCHCMAPPINGS 39
DEFINICIÓN 14 – LDIM 110
DEFINICIÓN 15 – ILEV 111
DEFINICIÓN 16 – LEVELFATHERS 111
DEFINICIÓN 17 – LEVELSONS 111
DEFINICIÓN 18 – TOPLEVELS 111
DEFINICIÓN 19 – BOTTOMLEVELS 112
DEFINICIÓN 20 – LEVELKEY 112
DEFINICIÓN 21 – OBJECTPARTITEMS 117
DEFINICIÓN 22 – OBJECTKEYITEMS 118
DEFINICIÓN 23 – OBJECTITEMS 118
DEFINICIÓN 24 – UPDATETABS 119
DEFINICIÓN 25 – UPDATEMAPS 119

Capítulo I - Introducción

1. Contexto

1.1. Data Warehouses

Un Data Warehouse (DW) es una base de datos que almacena información para la toma de decisiones. Dicha información es construida a partir de bases de datos que registran las transacciones de los negocios de la organización (bases operacionales[1]).

El objetivo de los DWs es consolidar información proveniente de diferentes bases de datos operacionales y hacerla disponible para la realización de análisis de datos de tipo gerencial. Los datos del DW son el resultado de transformaciones, chequeos de control de calidad e integración de los datos operacionales. Se incluyen también totalizaciones y datos pre-calculados en base a datos operaciones.

La prioridad en los DWs es el acceso interactivo e inmediato a información estratégica de un área de negocios. Los usuarios, en general con perfil gerencial, realizan sus propias consultas y cruzamientos de datos, utilizando herramientas especializadas con interfaces gráficas.

Por lo tanto, las operaciones preponderantes no son las transacciones, como en las bases de datos operacionales, sino consultas que involucran el cruzamiento de gran cantidad de datos, agrupaciones y sumarizaciones.

Las características de los DWs hacen que las estrategias de diseño para las bases de datos operacionales generalmente no sean aplicables para el diseño de DWs [Kim96], [Inm96].

Los modelos de datos para especificar los datos almacenados en el DW son diferentes. A nivel conceptual se proponen modelos multidimensionales [Ken96], [Agr97], [Car00], que representan la información como matrices multidimensionales o cuadros de múltiples entradas denominados *cubos*. A los ejes de la matriz se los llama *dimensiones* y representan los criterios de análisis, y a los datos almacenados en la matriz se los llama *medidas* y representan los indicadores o valores a analizar.

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

Para resolver los requerimientos planteados, los sistemas resultantes (Sistemas de Data Warehousing) combinan diferentes tecnologías.

[1] Se llama bases de datos operacionales a las diferentes bases de datos de una empresa u organización, asociadas a actividades o sistemas que permiten el ingreso confiable de información y el procesamiento eficiente de transacciones. A dichos sistemas se les llama OLTP: On Line Transaction Processing [Kim96].

1.2. Técnicas de Diseño

Las técnicas de diseño pueden clasificarse en cuatro niveles según el tipo de problemas que abordan. Se parte de técnicas que manipulan objetos de un modelo de datos sin aportar ningún criterio de diseño (*técnicas básicas*). A medida que se aumenta en el nivel, las técnicas correspondientes introducen elementos orientados a mejorar la productividad y calidad del diseño. Por esto, las técnicas de los niveles superiores se centran en tipos de sistemas de información o en contextos particulares de aplicación de sistemas de información. En [Bat92] se plantean estas ideas.

La Figura 1 muestra los niveles de diseño.

Figura 1 – Niveles de diseño

El nivel inferior corresponde a *técnicas básicas de diseño* para el modelo elegido, por ejemplo técnicas de diseño relacional para creación de estructuras del modelo (tablas, restricciones de integridad, etc.).

El siguiente nivel corresponde a *técnicas especializadas* para un determinado tipo de sistema de información, por ejemplo bases de datos centralizadas, federadas, distribuidas, etc. Cada sistema tiene sus propias técnicas especializadas de diseño, por ejemplo en bases de datos distribuidas existen técnicas para fragmentar tablas, tanto horizontal como verticalmente.

En un nivel superior se ubican las *estrategias de diseño*, orientadas a encarar globalmente un problema de diseño. Por ejemplo utilizar estrategias top-down o bottom-up para relevar requerimientos funcionales del sistema, o resolver la integración de esquemas en un ambiente federado con estrategias local-as-view o global-as-view. Las estrategias de diseño abstraen mecanismos para encarar problemas generales de diseño, y decidir qué técnicas conviene aplicar para la resolución de sub-problemas concretos.

En el nivel superior se ubican los *modelos de proceso* y *metodologías de diseño*. Los trabajos en este nivel resuelven la totalidad del problema, brindando metodologías, procesos o algoritmos que descomponen el problema en partes más pequeñas y muestran como atacar cada uno de los sub-problemas. Generalmente en este nivel es muy importante el orden en que se resuelven esos sub-problemas, mientras que las estrategias sólo se encargan de la resolución aislada de cada uno.

Por ejemplo: en [Spa94] se presenta un método para diseño conceptual basado en la integración de vistas conceptuales. Las técnicas básicas son las técnicas de modelado conceptual utilizadas para definir las vistas, y como técnicas especializadas presentan mecanismos de establecimiento de correspondencias entre vistas, y definen los conceptos de *link* y *path* de esas correspondencias. Luego presentan reglas de integración que atacan los problemas específicos de integración de elementos, links, paths y atributos. Estas reglas son estrategias para resolver la integración de componentes específicos. Por último presentan dos algoritmos que resuelven el problema general, presentando un orden de aplicación de las reglas.

2. Motivación

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

En el caso de diseño de DWs se debe tener en cuenta un componente adicional: las bases de datos fuentes. Un DW se construye con información extraída de un cierto conjunto de bases de datos fuentes. Durante el diseño lógico deben considerarse dichas bases y cómo se corresponden con el esquema conceptual. Por lo tanto es esencial poder relacionar los elementos del esquema conceptual con las tablas y atributos de las bases fuentes.

Para las bases de datos operacionales existen varias propuestas para construir un esquema relacional a partir de un esquema E/R [Mar89], [Teo86], [Jaj83]. Se consideran de gran importancia la existencia de técnicas para construir un esquema relacional de DW a partir de un esquema conceptual multidimensional.

De los trabajos existentes en diseño de DWs algunos proponen construir el esquema lógico a partir de requerimientos sin realizar un diseño conceptual, mientras que otros proponen la realización de un esquema conceptual y a partir de éste generar un esquema lógico basado en un tipo particular de diseño (generalmente estrella).

Estas carencias afectan la calidad del esquema resultado así como la productividad en el diseño. En la medida de que no hay una buena conexión entre la especificación conceptual y el esquema lógico diseñado pueden generarse diferencias respecto a la información que representan. El problema es mayor si no se especifica un esquema conceptual contra el cual validar el esquema lógico. También se pierde productividad en el desarrollo, ya que no se puede reutilizar el trabajo de análisis que normalmente corresponde a la etapa de diseño conceptual.

3. Objetivos

El objetivo de esta tesis es definir técnicas y estrategias de diseño para asistir al diseñador de un DW en la generación del esquema lógico tomando como entrada un esquema conceptual.

Las técnicas a proponer se ubican en los niveles de *Estrategias* y *Procesos* (ver sección 1.2), y se complementan con otros trabajos realizados en el laboratorio CSI[2]. Para definición de dichas técnicas es necesario contar con información sobre el esquema conceptual, las bases de datos fuentes, correspondencias entre ambos esquemas y lineamientos de diseño, los cuales también deben ser especificados.

Este trabajo es parte del proyecto DwDesigner[3] del laboratorio CSI, el cual incluye el desarrollo de un modelo conceptual multidimensional: CMDM [Car00] y técnicas especializadas de diseño relacional para el DW [Mar00]. En esta tesis se plantea vincular y consolidar dichos trabajos.

[2] CSI: Laboratorio Concepción de Sistemas de Información – Instituto de Computación – Facultad de Ingeniería – Universidad de la República. http://www.fing.edu.uy/~csi/

[3] El proyecto DwDesigner es un proyecto de investigación en las áreas de Diseño Conceptual y Lógico de Data Warehouses. http://www.fing.edu.uy/~csi/Proyectos/DwDesigner/

4. Solución Propuesta

En esta tesis se abordan los problemas de diseño lógico de DWs. La propuesta consiste en un proceso de diseño para construir un esquema lógico relacional tomando como entrada un esquema conceptual multidimensional y una base de datos fuente previamente integrada.

Según este enfoque, la tarea de diseñar un DW comienza con la construcción de un esquema conceptual multidimensional (*conceptual schema*). Dicho esquema conceptual se complementa con lineamientos de diseño (*guidelines*) que abstraen estrategias de diseño lógico de DWs y restricciones de performance y almacenamiento para el mismo. Como ejemplo de lineamientos, el diseñador puede indicar cómo particionar datos históricos o cuándo normalizar o denormalizar dimensiones. Los lineamientos indican cómo representar el esquema conceptual en el modelo relacional.

En este trabajo se propone un conjunto de lineamientos que permiten al diseñador indicar las estrategias a aplicar según los requerimientos de su problema concreto. Estos lineamientos constituyen un mecanismo sencillo y flexible para que el diseñador pueda expresar las propiedades que desea para el DW. Se proponen lineamientos por defecto que asisten al diseñador. El esquema conceptual más los lineamientos conforman el esquema intermedio (*intermediate schema*).

Se propone además, un mecanismo para especificar correspondencias o mapeos (*mappings*) entre el esquema intermedio y la base de datos fuente. El objetivo de los mapeos es indicar dónde se encuentra en la base de datos fuente, cada elemento del esquema intermedio. Dichos mapeos abstraen cálculos y funciones útiles para DWs.

La Figura 2 ilustra el proceso.

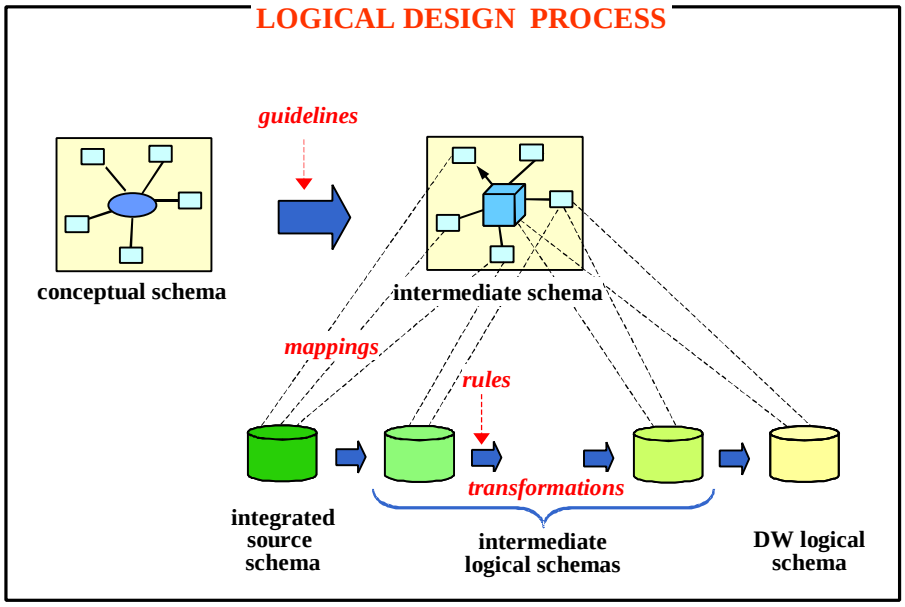

Figura 2 – Proceso de diseño lógico

En este trabajo nos basamos en la propuesta de técnicas especializadas para construir el esquema relacional de un DW presentada en [Mar00]. Dicha propuesta consiste en la aplicación de transformaciones de esquema (*transformations*), llamadas primitivas, a sub-esquemas[4] relacionales. Cada primitiva toma como entrada un sub-esquema y genera otro sub-esquema. Por ejemplo, existen primitivas que agregan atributos calculados a una tabla o que realizan totalizaciones a un conjunto de atributos.

[4] Se considera un sub-esquema como un conjunto de tablas que son parte de un esquema.

Estas primitivas se aplican a sub-esquemas relacionales, comenzando por los esquemas fuentes (*integrated source schema*), generando como resultado el esquema lógico del DW (*DW logical schema*). La aplicación de las primitivas deja como resultado una traza del diseño, útil para realizar la carga y actualización de los datos del DW. Si bien estas transformaciones ayudan al diseñador, es responsabilidad del mismo el lograr que el esquema lógico obtenido sea coherente con los requerimientos conceptuales, aplicando las primitivas adecuadas a su contexto.

Como forma de apoyar al diseñador en sus decisiones, en este trabajo se presenta un proceso de diseño que utiliza el esquema conceptual, los lineamientos y los mapeos como base para elegir qué primitivas se deben aplicar y en qué orden. Dicha elección se realiza en base a reglas de diseño (*rules*). Las reglas de diseño representan estrategias de diseño de DWs que sugieren al diseñador las transformaciones adecuadas para lograr un DW que se respete los requerimientos expresados a través del esquema conceptual y los lineamientos. Las reglas de diseño se encadenan mediante un algoritmo, generando el esquema relacional del DW.

El diseñador interviene en el proceso de diseño en: (1) la especificación del esquema conceptual, (2) la definición de lineamientos, y (3) la definición de mapeos. Para la generación del esquema lógico no es necesaria la intervención del diseñador por lo que el proceso es totalmente automatizable. Esto permite separar los aspectos semánticos, como estrategias e interpretación de los datos de la fuente, de los aspectos de implementación como el algoritmo de generación.

Según la clasificación de técnicas planteada anteriormente, nuestra propuesta se sitúa en los dos niveles superiores de diseño: estrategias y procesos. Concretamente, como estrategias se proponen un conjunto de reglas de diseño para construir el esquema relacional de un DW y como proceso se presenta un algoritmo que ordena la aplicación de dichas reglas.

Como técnicas básicas se utilizan las técnicas de diseño relacional, y como técnicas especializadas se utilizan las primitivas de transformación de esquemas propuestas en [Mar00].

La Figura 3 muestra los niveles de diseño abordados en esta propuesta. El algoritmo de generación del esquema lógico consiste en una serie de pasos que encadenan la aplicación de reglas de diseño. Cada regla consiste de la aplicación de una o más primitivas a un sub-esquema relacional.

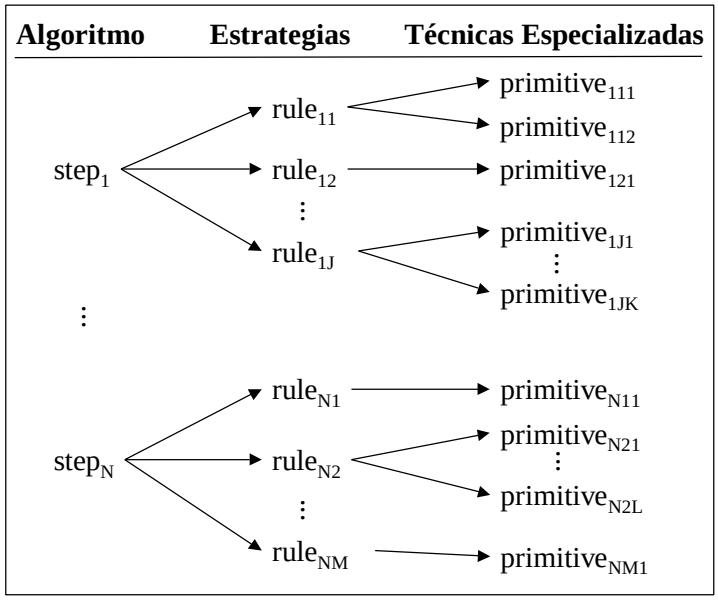

Figura 3 – Proceso de diseño propuesto

5. Aportes

En general, los trabajos existentes en diseño de DWs no conectan la etapa de diseño conceptual con la de diseño lógico. Los trabajos que plantean cierta conexión entre las etapas [Gol98], [Cab98] se centran en la especificación del esquema conceptual y presentan la generación de un esquema lógico fijo (generalmente estrella) a partir de éste. El objetivo de dicha generación es mostrar que los esquemas conceptuales pueden ser implementados en el modelo relacional, pero no profundizan en metodologías de diseño lógico, ni dan flexibilidad en cuanto a los esquemas generados (sólo estrella). Otros trabajos [Kim96] [Kor99] proponen construir el esquema lógico a partir de los requerimientos sin especificar un esquema conceptual. En general, estos trabajos consisten de un conjunto de técnicas de diseño poco estructuradas y conceptualizadas.

El principal aporte de este trabajo consiste en un proceso de diseño lógico que parte de un esquema conceptual junto con una base de datos fuente integrada. Este proceso se basa en reglas de diseño especializadas en DWs y un algoritmo que encadena la aplicación de las mismas. Tanto las reglas como el algoritmo propuesto constituyen un paso adelante en la sistematización del diseño de DWs.

Esta propuesta incluye:

- Definición de lineamientos de diseño que abstraen diferentes estrategias de diseño de DWs.
- Definición de mapeos que relacionan el esquema conceptual (enriquecido con lineamientos) y la base de datos fuente.
- Propuesta de reglas de diseño y el algoritmo que ordena su aplicación. El algoritmo puede automatizarse permitiendo que el diseñador se concentre en la especificación del esquema conceptual, lineamientos y mapeos.
- Extensiones a las primitivas de transformación de esquemas propuestas en [Mar00]. La utilización de primitivas en la generación del esquema lógico deja como resultado una traza de diseño, útil tanto para documentación y mantenimiento como para carga y actualización del DW.
- Implementación de un prototipo de una herramienta CASE que implementa este proceso de diseño.

6. Organización del documento

Esta tesis tiene 5 capítulos más. En el capítulo 2 se presenta una revisión de los trabajos en el área de diseño de DWs que fueron más relevantes para este trabajo.

En los capítulos 3 y 4 se presenta el proceso de diseño. En el capítulo 3 se define el proceso de diseño y los conceptos básicos utilizados. Se presentan los conceptos de lineamientos y mapeos y se describe el rol del diseñador dentro de dicho proceso.

En el capítulo 4 se presenta la generación del esquema lógico del DW. Se describen las reglas de diseño y el algoritmo automático de generación del esquema relacional.

En el capítulo 5 se describe brevemente la implementación de un prototipo de una herramienta CASE de diseño de DWs.

En el capítulo 6 se presentan las conclusiones, trabajo en curso y trabajo futuro. Por último se presentan los apéndices referenciados en el documento y la bibliografía consultada.

Capítulo II - Estado del Arte

1. Introducción

Este trabajo se sitúa en el área de diseño de Data Warehouses; específicamente propone técnicas para la construcción de un esquema lógico relacional de DW a partir de un esquema conceptual.

Las diferencias de los DWs con las bases de datos tradicionales, sobre todo en cuanto al tipo de consultas y performance esperada en las mismas, hacen que las estrategias de diseño y los modelos de datos utilizadas para el DW sean diferentes.

Los trabajos existentes sobre generación de esquemas relacionales a partir de esquemas conceptuales, en especial de esquemas E/R [Che76], pueden ser aplicados en algunos casos, pero las diferencias del modelo E/R con los modelos multidimensionales hacen necesaria la utilización de nuevas técnicas.

Algunos trabajos sobre reglas de diseño y establecimiento de correspondencias de diversas áreas de investigación son de utilidad para la definición de dichas técnicas.

En este capítulo se presenta una revisión del estado del arte en dichas áreas. En la sección 2 se presenta el proceso de diseño de un DW y se mencionan trabajos existentes en diseño conceptual y diseño lógico. En la sección 3 se analizan trabajos de generación de esquemas lógicos a partir de esquemas conceptuales, tanto para bases de datos operacionales como para DWs. En la sección 4 se citan algunos trabajos sobre reglas de diseño y correspondencias, y en la sección 5 se presentan las conclusiones.

2. Diseño de Data Warehouses

Los sistemas de Data Warehousing han sido objeto de variados trabajos de investigación en la última década. Los trabajos comprenden diferentes áreas y diferentes enfoques; algunos de los tópicos de interés pueden encontrarse en [Wid95], [Wu97] y [Cha97]. Sus marcadas diferencias con los sistemas operacionales provocaron el estudio de nuevas técnicas y metodologías de diseño.

Como en los sistemas de bases de datos tradicionales, el proceso de diseño del DW puede dividirse en tres etapas secuenciales: diseño conceptual, diseño lógico y diseño físico [Bat92].

En la Figura 4 se muestran las etapas con sus respectivas entradas y salidas de información.

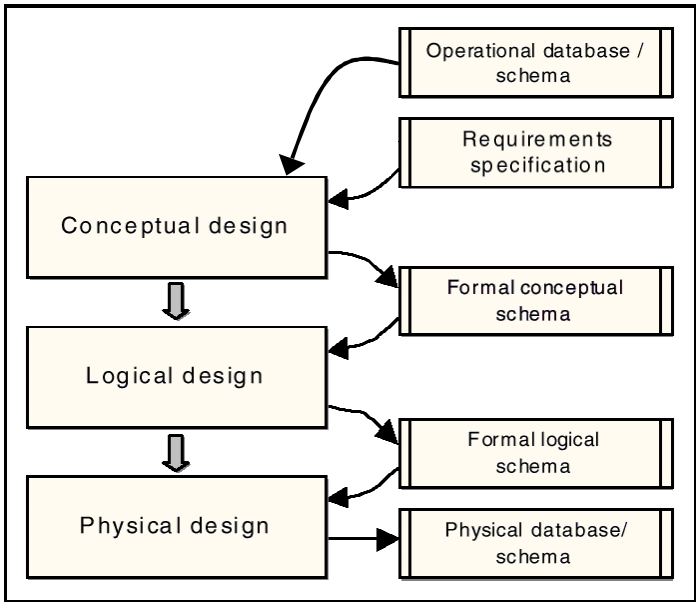

Figura 4 – Proceso de diseño de un Data Warehouse

En la etapa de diseño conceptual se construye un esquema conceptual de la realidad a partir de los requerimientos y/o bases fuentes. Dicho esquema conceptual es enriquecido con requerimientos de performance y almacenamiento durante la etapa de diseño lógico, y a partir de él se genera un esquema lógico, que es dependiente del tipo de modelo y tecnología de DBMS. Hay dos familias de esquemas lógicos: relacionales y multidimensionales, y actualmente se están considerando esquemas híbridos. Por último, en la etapa de diseño físico se implementa el esquema lógico en el manejador de bases de datos elegido, teniendo en cuenta técnicas de optimización física, como son: índices particiones, etc.

Las diferentes propuestas de diseño de DWs se enfocan en alguna de estas etapas; algunos trabajos proveen metodologías que involucran varias etapas. Algunos autores proponen metodologías con más etapas, que son, sub-etapas de las planteadas [Gol98], [Hus00], [Bal98].

En las siguientes secciones se resumen algunos de los trabajos en diseño conceptual y lógico.

2.1. Diseño Conceptual

El diseño conceptual tiene por objetivo la construcción de una descripción abstracta y completa del problema. Comienza con el análisis de requerimientos de los usuarios y de reglas de negocio, y finaliza con la construcción de un esquema conceptual expresado en términos de un modelo conceptual.

Los trabajos existentes en diseño conceptual para DWs corresponden fundamentalmente a modelos de datos. [Cab98], [Gol98a], [Leh98a], [Sap99a], [Fra99], [Car00], [Hus00]. Dichos modelos de datos son *modelos multidimensionales* que expresan la realidad en términos de dimensiones y medidas. En [Car00] (sección 2.1) y en [Bal98] (sección 6.4.1) se describen las estructuras básicas de los modelos multidimensionales. Una descripción más detallada puede encontrarse en [Ken96]. En [Car00] se comparan diferentes modelos.

En una primera fase se seleccionan los objetos relevantes para la toma de decisiones, y se especifica el propósito de utilizarlos como dimensiones y/o medidas. Para realizar dicha selección hay dos grandes enfoques que se basan respectivamente en un análisis de requerimientos [Car00], [Sap99a], [Mcg98], o en un análisis de las bases de datos fuente [Gol98a], [Cab98]. Hay enfoques intermedios [Hus00], [Bal98].

En el enfoque basado en requerimientos se analizan los requerimientos de los usuarios, y se identifican en ellos los hechos, dimensiones y medidas relevantes. La realidad se modela como un conjunto de cubos multidimensionales, que se obtienen a partir de los hechos, dimensiones y medidas identificados.

Para relevar los requerimientos se depende de la experiencia del diseñador y de entrevistas a los usuarios; es una fase poco automatizable. En [Bal98] se plantea una metodología para relevar requerimientos. En [Mcg98] se plantean algunos lineamientos para construir un modelo de negocios.

Los trabajos de diseño conceptual que siguen este enfoque parten de un relevamiento de requerimientos ya realizado y proponen modelos para formalizarlos.

En [Sap99a], Sapia, Blaschka, Höfling y Dinter presentan un modelo que extiende el modelo entidad-relación: E/R. Se enfatiza en la representación sencilla de las estructuras multidimensionales.

En [Car00], Carpani propone un modelo formal para representar los conceptos multidimensionales. Se enfatiza en lograr expresividad en los esquemas y se presenta un lenguaje para modelar restricciones de integridad [Car01].

En el enfoque basado en las bases fuentes se construyen cubos multidimensionales transformando un esquema conceptual de las bases fuentes. Como modelo conceptual de las fuentes, en general se utiliza el modelo E/R. Las diferentes metodologías comienzan por identificar en el esquema fuente los posibles hechos relevantes para la toma de decisiones. A partir de los hechos identificados navegan por las entidades y relaciones construyendo las jerarquías de las dimensiones.

En [Gol98a], Golfarelli, Maio y Rizzi construyen esquemas en forma de árboles, cuyas raíces son los hechos y las ramas conforman las dimensiones. Aplicando operaciones, que llaman poda e injerto, se descartan los atributos no relevantes. El modelo permite además representar algunas nociones de aditividad.

En [Cab98], Cabibbo y Torlone identifican los hechos y las dimensiones con los que construyen un esquema del modelo MD. Dicho modelo especifica formalmente las estructuras multidimensionales y permite manipular la dimensionalidad genérica. Muchos de los trabajos posteriores se basan en este modelo.

Algunos trabajos proponen encares intermedios. En [Hus00], Hüsemann, Lechtenbörger y Vossen proponen una etapa inicial de análisis y especificación de requerimientos, y luego construyen el esquema conceptual a partir de las bases fuentes, identificando en las mismas los conceptos ya relevados.

En la sección 7.4 de [Bal98], Ballard, Herreman, Schau, Bell, Kim y Valncic proponen identificar dimensiones, medidas y hechos en un E/R de las bases fuentes. Luego, se especifican semi-formalmente los requerimientos como cruzamientos de las dimensiones y medidas ya relevadas, y a partir de los requerimientos se obtienen esquemas estrella. Su modelo no es puramente conceptual ya que no es independiente de la implementación, pero presentan una metodología de diseño muy interesante, que puede ser reutilizable para diseño conceptual.

2.2. Diseño Lógico

La etapa de diseño lógico toma como entrada un esquema conceptual y genera un esquema lógico relacional o multidimensional. La dificultad principal es encontrar un esquema lógico que satisfaga no sólo los requerimientos funcionales de información, sino también requerimientos de performance en la realización de consultas complejas de análisis de datos. Esto tiene particular impacto en el caso de usarse bases relacionales, ya que las consultas de análisis de datos incluyen operaciones muy costosas para DBMS relacionales.

Algunas metodologías parten de un esquema conceptual y proveen mecanismos para generar un esquema lógico a partir de él [Bal98], [Cab98], [Gol98]; otras metodologías pasan por alto el modelado conceptual y construyen el esquema lógico a partir de los requerimientos y/o las bases fuentes [Kor99], [Kim98], [Sil97], [Mar00], [The99], [Lig99].

El resultado de esta etapa es la especificación formal de un esquema lógico para el DW. Los modelos propuestos incluyen materialización de vistas [Zhu95], [Lab96], [Hul96], [The99], [Lig99], modelos específicos basados en el modelo relacional y optimizados para consultas OLAP [Kim96], [Sil97], [Kor99], e implementaciones multidimensionales, en general propietarias de los manejadores [Vas99], [Hah00], [Ola98], [Ms00], [Stu00].

En el enfoque basado en vistas, el DW se ve como un conjunto de vistas materializadas de las bases fuentes. Estos trabajos no se centran en la representación de conceptos multidimensionales, como dimensiones y medidas, sino en materializar algunas vistas para lograr performance en un conjunto dado de consultas.

En el proyecto WHIPS se trabaja sobre la definición y mantenimiento del DW [Zhu95], [Lab96] y la consistencia de las vistas [Zhu97].

En el proyecto H2O proponen combinar vistas materializadas y virtuales centrándose en la integración de datos [Hul96], [Hul97].

En el proyecto DWQ, a partir de un conjunto de consultas sobre las fuentes, se resuelve qué vistas materializar de manera de maximizar la performance en las consultas y minimizar el espacio de disco utilizado. Utilizan funciones de costo que permiten automatizar la selección [The99], [The99a], [The99b], [Lig99].

Otro enfoque consiste en definir estructuras (dentro del modelo relacional) que optimicen las consultas que se realizarán al DW. Dichas consultas contienen gran número de joins y sumarizaciones que degradan la performance del sistema.

En [Kim96], Kimball propone el modelo estrella (star), que consiste de una gran tabla central conteniendo información sobre los hechos, y tablas más pequeñas (relacionadas a la tabla de hechos) con información sobre las dimensiones. Kimball propone lineamientos prácticos para construir un esquema estrella pero no presenta una metodología general. Varios trabajos de modelización conceptual generan esquemas estrella a partir de esquemas conceptuales [Gol98], [Cab98].

En [Kor99], Kortink y Moody analizan varios modelos (flat, terraced, star, constellation, galaxy, snowflake, star cluster) variantes del modelo estrella, y proponen una metodología para construirlos a partir de un esquema E/R de las fuentes.

Un enfoque diferente consiste en aplicar sucesivas transformaciones a una base fuente integrada hasta obtener el esquema lógico deseado para el DW. En [Mar99] y [Mar00], Marotta define un conjunto de transformaciones de esquemas que permite llegar a diferentes estilos de diseño (star, snowflake, …) según los requerimientos de performance y almacenamiento del problema. El trabajo incluye un conjunto de reglas y estrategias que facilitan la elección de las transformaciones a aplicar.

La aplicación de las transformaciones deja una traza de diseño que permite repercutir en el DW los cambios ocurridos en la fuente, y de esta manera manejar la evolución de la fuente.

3. Generación del Esquema Lógico a partir del Esquema Conceptual

La construcción de un esquema conceptual, y su posterior representación en un esquema lógico, es un problema clásico en el diseño de bases de datos.

En [Che76], Chen propone el modelo entidad-relación (E/R) para incorporar semántica a la modelización de bases de datos. Se analizan algunas ambigüedades de los modelos lógicos de redes, relacional y entity-set, y se analizan algunas alternativas de derivación del esquema E/R a cada uno de los modelos lógicos.

En [Teo86], Teorey, Yang y Fry proponen una metodología para transformar un esquema E/R en un esquema relacional basada en reglas de diseño. La aplicación de las reglas se ordena mediante una secuencia de pasos.

En [Jaj83], Jajodia, Ng y Springsteel proponen una metodología alternativa para transformar un esquema E/R a un esquema relacional.

En [Mar89], Markowitz y Shoshani proponen una metodología para representar un esquema E/R en el modelo relacional, cumpliendo un conjunto de condiciones que utilizan para probar la correctitud de su metodología. Los autores critican los trabajos anteriores presentando ejemplos en los que fallan dichas metodologías.

En diseño de DWs la generación del esquema lógico del DW a partir de un esquema conceptual no ha sido suficientemente atacada. Los trabajos existentes se centran en la especificación conceptual y mencionan como se sería su representación en un esquema estrella (o alguna de sus variantes).

Golfarelli, Maio y Ricci en [Gol98] proponen una metodología de diseño de DWs basada en su modelo conceptual DF (Dimensional Fact model) presentado en [Gol98a]. Los *esquemas de hechos* construidos con DF tienen una estructura central que representa al *hecho* (con las medidas), y ramificaciones a partir del *hecho* que representan las dimensiones (atributos unidos formado jerarquías).

Los autores proponen generar esquemas lógicos relacionales o multidimensionales a partir de los *esquemas de hechos*. No proponen ningún esquema en especial pero dan un ejemplo de como generar un esquema estrella. Para cada estructura (que representa un *hecho*) se construye una *fact table*, con atributos para las medidas y para los atributos de las dimensiones más próximos al hecho. Para cada rama (que representa una dimensión) se construye una *dimension table*, cuyos atributos son los atributos de la dimensión.

Luego proponen aplicar fragmentación vertical para las medidas que usualmente no se consultan juntas, y fragmentación horizontal para dividir la fact table en porciones más performantes. Las estrategias de fragmentación son tomadas de [Ozs91]. La fragmentación vertical de medidas es necesaria en este contexto ya que para construir los hechos no se tuvieron en cuenta los requerimientos, los cuales vinculan las medidas.

Los autores no se enfocan en una metodología de generación del esquema lógico, su punto central es la definición del modelo conceptual. La generación del esquema lógico se deja en manos del diseñador. Para esto debe decidir qué sub-esquema utilizar, qué resúmenes[5] desea mantener, y cómo realizar las fragmentaciones. Presentan un modelo de costos para apoyar en la decisión. La implementación de la generación también se deja en manos del diseñador, quien es responsable de mantener la conexión entre el esquema generado y los esquemas fuente, si lo desea.

A partir del modelo MD presentado por Cabibbo y Torlone en [Cab98] pueden generarse tanto a bases relacionales como arrays multidimensionales. Para el caso de bases relaciones plantean generar esquemas estrella, si bien se admiten variantes tipo snowflake.

El modelo está formado por f-tables y dimensiones. Las dimensiones tienen niveles, un orden parcial y funciones de roll-up; las f-tables tienen un conjunto de atributos asociados a niveles, y un nivel que oficia de medida. La generación del esquema estrella es trivial: se crea una fact table por cada f-table y una dimension table por cada dimensión. Los autores formalizan cómo generar el esquema y cómo poblar las instancias del esquema lógico a partir de las instancias del esquema MD.

La generación del esquema estrella es presentada formalmente, pero apunta a mostrar que los modelos MD se pueden implementar en el esquema relacional; los autores no se enfocan en brindar una metodología para realizar dicha generación.

[5] En el contexto de DW, el término *resumen* (aggregate) representa el resultado de agrupar los registros de una tabla, aplicando una operación de resumen o totalización a las medidas (por ejemplo: suma, promedio, máximo).

En [Bal98] Ballard et al. proponen una metodología de diseño que cubre todas las etapas, desde la especificación de requerimientos hasta la implementación y carga del DW. Si bien los requerimientos tienen un peso importante en el diseño, la creación del esquema lógico se hace a partir de las bases fuentes. Para construir el esquema lógico se utiliza el modelo estrella, aunque en algunos pasos de la metodología se lo toma como esquema conceptual.

La metodología no hace una diferenciación explícita entre diseño conceptual y lógico, por lo que es muy difícil separarlas, y no existe un momento de transición entre ambas etapas.

La metodología incluye algunos lineamientos de diseño interesantes para decidir los cruzamientos a implementar en tablas de hechos y la granularidad de las mismas, en base a los requerimientos.

En este trabajo se propone un algoritmo de generación automática del esquema lógico, en el que la intervención del diseñador sea fundamentalmente semántica indicando estrategias de diseño. Se quiere dar flexibilidad en cuanto al esquema elegido y dejar una traza de diseño de manera de automatizar también la posterior carga de datos.

4. Otras Técnicas de Diseño

4.1. Reglas de Diseño

La aplicación de reglas de diseño es una técnica clásica en el diseño de bases de datos. Diferentes trabajos proponen métodos (como secuencias de pasos), y en cada paso se aplican algunas reglas de diseño.

En [Teo86], Teorey, Yang y Fry proponen reglas para construir un esquema relacional a partir de un esquema E/R. La aplicación de las reglas se ordena mediante una secuencia de pasos.

En [Spa94], Spaccapietra y Parent plantean la integración de vistas utilizando reglas de integración. Proponen un algoritmo en el que cada paso consiste en la aplicación de una regla a las vistas que cumplan las precondiciones de la regla.

En [Ler90], Lerner y Habermann proponen transformaciones de esquemas que automatizan la evolución y reorganización de los esquemas.

En este trabajo se utilizan reglas de diseño para generar un esquema lógico relacional a partir de un esquema conceptual multidimensional. Se Propone un algoritmo que secuencia la aplicación de las reglas.

4.2. Correspondencias entre esquemas

El establecimiento de correspondencias entre diferentes esquemas es también una técnica clásica en el diseño de bases de datos. Diferentes metodologías de integración tienen como primer paso establecer correspondencias entre los esquemas a integrar, y luego a partir de ellas se realiza la integración.

En [Spa94], Spaccapietra y Parent establecen correspondencias de equivalencia entre elementos de las vistas a integrar.

En [Fan97], Fankhauser propone varios tipos de correspondencias, que luego simplifica a correspondencias de equivalencia para relacionar objetos de diferentes esquemas a integrar.

En [Vid01], Vidal, Lóscio y Salgado proponen correspondencias de equivalencia e inclusión para mapear[6] los objetos de fuentes XML con un esquema conceptual del mediador. En este caso, el mapeo se realiza entre objetos de diferente tipo.

En [Lar00], Larrañaga propone utilizar consultas SQL como mapeo entre las bases fuentes y un esquema estrella ya definido, con el objetivo de enriquecer el modelo estrella con datos descriptivos de las fuentes. La propuesta ataca el problema de drill-through hacia las fuentes, pero no es extensible a otro tipo de objetos.

En [Yan01], Yan, Miller, Haas y Fagin presentan una herramienta para deducir y refinar mapeos entre esquemas. Un mapeo asocia un valor a un atributo destino calculado a partir de un conjunto de valores de atributos fuentes. Cuando los atributos fuentes son de tablas diferentes utiliza caminos de join para relacionar dichas tablas; la integridad referencial es un punto de partida para la definición de los caminos. El mecanismo permite establecer múltiples caminos, y expresar caminos de outer join entre las tablas. Los mapeos se utilizan únicamente para bases relacionales. La herramienta: Clio, asiste al diseñador en la definición de los mapeos en dos modos: vistas de esquemas y vistas de datos.

En este trabajo se utilizan correspondencias para relacionar los elementos del esquema conceptual con una base fuente integrada. En el contexto de DWs no alcanza con correspondencias de equivalencia, se necesitan expresar cálculos y funciones sobre los elementos de la fuente.

5. Conclusiones

Las diferencias de los DWs con respecto a las bases de datos tradicionales provocaron el desarrollo de nuevas metodologías de diseño y la creación de nuevos modelos de datos. Los trabajos de generación de esquemas lógicos a partir de esquemas conceptuales no se pueden aplicar directamente pero sirven de base para nuevas propuestas.

El trabajo existente de diseño lógico de DWs a partir de esquemas conceptuales sólo resuelve la construcción de sub-esquemas específicos (estrella, snowflake), pero no resuelve casos generales. En este capítulo se estudiaron algunos de esos trabajos, pero en todos los casos carecen de generalidad y trazabilidad del proceso.

Otros trabajos proponen la construcción del esquema lógico sin un esquema conceptual previo, ya sea a partir de las bases fuentes, de consultas a las mismas, o de un análisis de requerimientos. Algunas de esas propuestas conducen a esquemas lógicos más generales, pero su obtención depende en gran medida de la intervención del diseñador.

En la tabla de la Figura 5 se clasifican algunos trabajos de diseño lógico de bases de datos tradicionales (BD) y de Data Warehouses (DW). Para cada trabajo se analiza la entrada a la etapa de diseño lógico, el modelo lógico elegido y la técnica por la cual se obtiene.

En la tabla de la figura Figura 6 se comparan algunos de esos trabajos, los que abordan diseño lógico de DWs y permiten ver la información en términos de dimensiones y medidas. Para cada propuesta se analiza:

- Si el esquema lógico se obtiene a partir de un esquema conceptual.
- El modelo lógico resultado.
- Si el diseñador puede elegir el estilo de diseño: estrella, snowflake, etc. Las propuestas que lo permiten, dan facilidades al diseñador, pero es éste quien, implementa dicho estilo.

[6] Un mapeo (mapping) es una regla de correspondencia entre conjuntos, que asocia cada elemento de un conjunto con un elemento de otro conjunto. El término mapear refiere a establecer un mapeo o correspondencia, para un elemento o para todo el conjunto.

- Si se mantiene una traza de operaciones realizadas durante el diseño, que permitan repetirlo, manejar cambios y facilitar la carga de datos. Algunos trabajos pueden lograrlo pero sólo a través de documentación.
- Si incluye indicadores de costos para las estructuras / operaciones elegidas. [Gol98] presenta un modelo de costos bastante completo, las demás propuestas sólo estiman el tamaño de las tablas resultado.
- Si se incluye la materialización de resúmenes o tablas de hechos derivadas.
- Si se presentan estrategias para fragmentar horizontalmente o verticalmente una tabla de hechos o un cubo. Algunas propuestas no lo indican explícitamente.

También se estudiaron diversas técnicas de transformación en base a reglas y correspondencias que se aplican directamente en este trabajo.

Área	Trabajo	Entrada	Modelo Lógico	Técnica Diseño Lógico
BD	[Che76]	Conceptual: E/R	Redes, relacional, entity-set	A partir del esq. conceptual
BD	[Teo86]	Conceptual: E/R	Relacional	A partir del esq. conceptual
BD	[Jaj83]	Conceptual: E/R	Relacional	A partir del esq. conceptual
BD	[Mar89]	Conceptual:EE/R	Relacional	A partir del esq. conceptual
DW	[Cab98]	Conceptual: MD	Estrella, MD	A partir del esq. conceptual
DW	[Gol98] [Gol98a]	Conceptual: MD	Estrella, snowflake, MD	A partir del esq. conceptual
DW	[Bal98]	Requerim + fuentes	Estrella	A partir de las bases fuentes
DW	[Zhu97]	Fuentes	Vistas	Materialización de vistas
DW	[Hul96] [Hul97]	Vistas	Vistas	Integración de datos
DW	[The99a] [Lig99]	Consultas	Vistas	Materialización de vistas
DW	[Kim96]	Requerimientos	Estrella	Patrones de diseño
DW	[Kor99] [Moo00]	Fuentes	Estrella, snowflake	A partir de las bases fuentes
DW	[Mar00]	Fuentes	Relacional	Transformación de esquemas

Figura 5 – Clasificación de diferentes propuestas de diseño lógico.

Trabajo	Parte del concep.	Modelo Lógico	Elección de estilo	Traza de diseño	Func. Costo	Mat. Resum.	Fragm. Horiz.	Fragm. Vert.
[Cab98]	Si	Estrella, MD	No	No	No	No	No	No
[Gol98] [Gol98a]	Diseñador	MD,estrella, snowflake	Si	No	Si	Si	Si	Si
[Bal98]	No	Estrella	No	No	Si	Si	No	No
[Kim96]	No	Estrella	No	No	Si	Si	No	No expl
[Kor99] [Moo00]	No	Estrella, snowflake	Si	No	No	No	No	No
[Mar00]	No	Relacional	Si	Si	No	Si	Si	Si

Figura 6 – Comparación de diferentes propuestas de diseño lógico.

Capítulo III - Definición de Lineamientos y Mapeos

1. Introducción

La construcción de un DW puede verse como una secuencia de tres etapas fundamentales: diseño conceptual, diseño lógico y diseño físico.

Durante la etapa de diseño conceptual se realiza una abstracción de la realidad basados en los objetos del negocio y los requerimientos de información. El resultado de esta etapa es un esquema conceptual que especifica el problema a resolver. El esquema lógico es una especificación más detallada que el esquema conceptual donde se modelan los datos según un modelo lógico de DBMS (por ejemplo el modelo relacional), definiéndose por lo tanto, estructuras más cercanas al nivel de almacenamiento. Durante la etapa de diseño lógico se construye el esquema lógico teniendo en cuenta no sólo el esquema conceptual, sino también estrategias para resolver los requerimientos de performance y almacenamiento [Bat92].

En el caso de diseño de DWs se debe tener en cuenta un componente adicional: las bases de datos fuentes. Un DW se construye con información extraída de un cierto conjunto de bases de datos fuentes. Durante el diseño lógico deben considerarse estas bases y cómo se corresponden con el esquema conceptual.

Durante la etapa de diseño físico se incorporan elementos específicos de almacenamiento y performance, como son la elección de índices, almacenamiento especializado, parámetros de sistemas, etc.

En este trabajo se propone un mecanismo para construir el esquema lógico de un DW, mediante transformaciones sucesivas aplicadas a las bases fuentes. Para la elección de las transformaciones adecuadas a cada caso se tienen en cuenta: el esquema conceptual, correspondencias entre el esquema conceptual y las fuentes, y decisiones de diseño (lineamientos).

A continuación se describe el proceso de diseño lógico de un DW seguido en este trabajo. El mismo consiste en: (1) definición de lineamientos, (2) definición de mapeos a la base fuente y (3) transformación del esquema fuente. La Figura 7 ilustra el proceso.

El proceso de diseño lógico tiene dos entradas: el esquema conceptual (conceptual schema) y el esquema lógico de la base fuente integrada (integrated source schema).

Primeramente el diseñador indica algunos lineamientos que complementan al esquema conceptual con estrategias de diseño lógico (*guidelines*), por ejemplo: cómo fragmentar datos históricos o qué datos almacenar juntos. El esquema conceptual más los lineamientos conforman el esquema intermedio (*intermediate schema*).

Luego el diseñador establece mapeos o correspondencias (*mappings*) entre el esquema intermedio y el esquema lógico de la fuente. Dichos mapeos indican dónde se encuentran en la fuente los diferentes elementos del esquema intermedio.

Finalmente se aplican transformaciones (*transformations*) al esquema lógico de la base fuente, refinándolo sucesivamente, hasta obtener el esquema lógico deseado para el DW (*DW logical schema*). Durante el proceso de transformación se tienen esquemas lógicos intermedios (*intermediate logical schemas*). El término *esquema lógico* representa el esquema que se tiene en alguna etapa de la transformación; al comienzo coincide con el esquema de la fuente, luego con algún esquema intermedio y al final con el esquema del DW.

En este trabajo se propone un algoritmo y un conjunto de reglas de diseño (*rules*) para llevar a cabo dicho refinamiento.

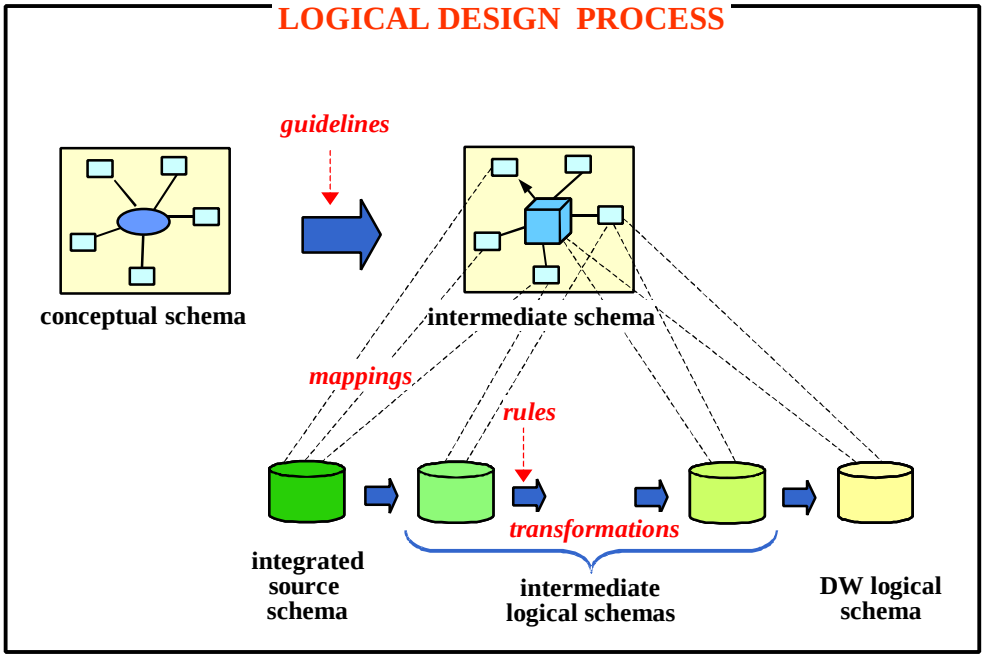

Figura 7 – Proceso de diseño lógico

El proceso de diseño lógico propuesto puede dividirse en dos grandes etapas: (1) definición de propiedades y mapeos a la fuente y (2) generación del esquema lógico.

En la primer etapa se definen los lineamientos y se establecen los mapeos a la base fuente. Esta etapa tiene una alta participación del diseñador, incorporando información semántica a través de propiedades y vínculos entre el esquema intermedio y la fuente. La tarea del diseñador no incluye ningún tipo de procesamiento ni transformación de los esquemas, sólo la definición de propiedades que éstos deben cumplir.

En la segunda etapa se lleva a cabo la generación del esquema lógico del DW a través de transformaciones aplicadas a la fuente. Se utilizan las definiciones realizadas en la primer etapa para elegir qué transformaciones aplicar al esquema de la base fuente y en qué orden hacerlo. Este proceso es automatizable y no requiere la participación del diseñador.

Esta forma de trabajo tiene como objetivo que el diseñador se concentre en aspectos semánticos (especificando propiedades y correspondencias) más que en la codificación del esquema resultante.

En este capítulo se presenta la primera etapa. En la sección 2 se presenta una descripción informal de la propuesta mediante un ejemplo, y en la sección 3 se define la notación utilizada en este documento.

En las siguientes secciones se estudian los distintos componentes del proceso: el esquema conceptual (sección 4), los lineamientos (sección 5), el esquema intermedio (sección 6), la base fuente integrada (sección 7) y los mapeos (sección 8).

En el siguiente capítulo (IV) se presenta la segunda etapa; se estudian las transformaciones, las reglas y el algoritmo propuesto.

2. Descripción Informal

En esta sección se presenta un ejemplo que ilustra la primera etapa del proceso de diseño lógico de un DW. Se ilustra tanto la definición de lineamientos como de mapeos.

El ejemplo trata de una empresa que brinda atención telefónica a su cartera de clientes y quiere contabilizar el tiempo invertido en atención de llamadas por mes y por cliente. Además interesan totales anuales y por ciudad y departamento de residencia del cliente.

En este ejemplo, se pueden identificar tres dimensiones: *clientes, fechas* y *llamadas* (medida). La dimensión clientes tiene 3 niveles: *departamento, ciudad* y *cliente*; y la dimensión fechas tiene 2 niveles: *año* y *mes*.

La Figura 8 muestra un esquema multidimensional para esta realidad especificado en CMDM [Car00]. Los diagramas a), b) y c) representan las dimensiones. Los cuadros de texto blancos son los niveles de la dimensión; sus nombres aparecen en negrita y debajo aparecen los atributos llamados *ítems*. Los ítems seguidos por un numeral (#) identifican al nivel. Las jerarquías de niveles se representan por flechas entre los niveles. La Figura 8d representa la relación dimensional *atención* que cruza las dimensiones *clientes* y *fechas* y tiene *llamados* como medida (nivel apuntado por una flecha).

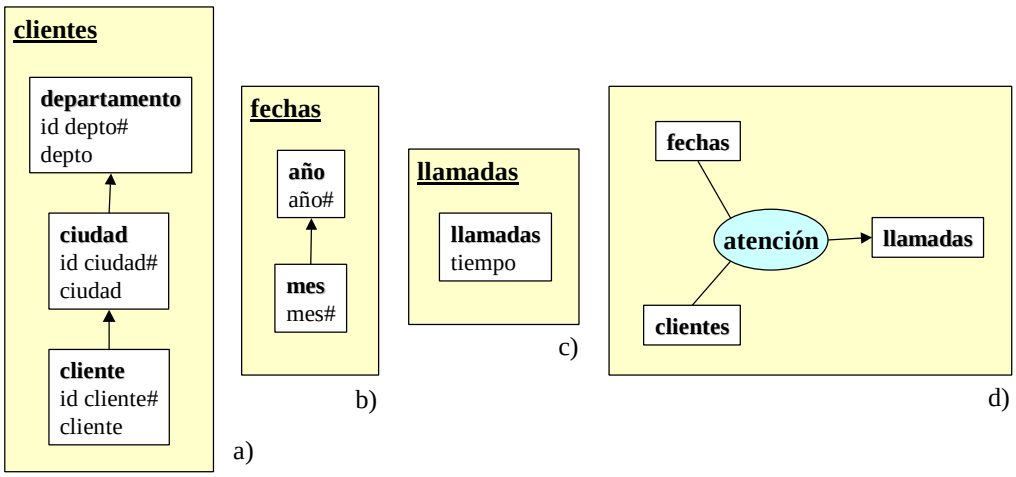

Figura 8 – Esquema conceptual: a) dimensión clientes, b) dimensión fechas y c) relación dimensional atención

Se quiere construir un esquema relacional a partir del esquema conceptual. Se tienen varias alternativas para definir tanto las tablas de dimensión (dimension tables), como las tablas de hechos (fact tables).

Con respecto a la dimensión *clientes*, se puede optar por almacenar cada nivel en una tabla, con los ítems del nivel y la clave del nivel superior en la jerarquía para poder realizar el join (Figura 9a). Otra opción sería mantener una única tabla para la dimensión que contenga todos los ítems (Figura 9b). También se pueden seguir estrategias intermedias, por ejemplo almacenar en una tabla el nivel *cliente* y en otra tabla los niveles *ciudad* y *departamento* (Figura 9c).

El diseñador debe decidir qué niveles almacenar juntos en una misma tabla, basándose en criterios de performance, redundancia y ocupación de disco.

La misma decisión debe tomarse con respecto a la dimensión *fechas*.

De la misma forma se debe definir qué tablas de hechos se van a implementar. La relación dimensional *atención* cruza dos dimensiones: *clientes* y *fechas*. El cruzamiento con más detalle es *meses* por *clientes* (Figura 9d) pero también se puede querer almacenar totales, por ejemplo, por *ciudad* y *año* (Figura 9e).

El diseñador debe decidir qué cruzamientos materializar, tratando de lograr un equilibrio entre performance y espacio de almacenamiento.

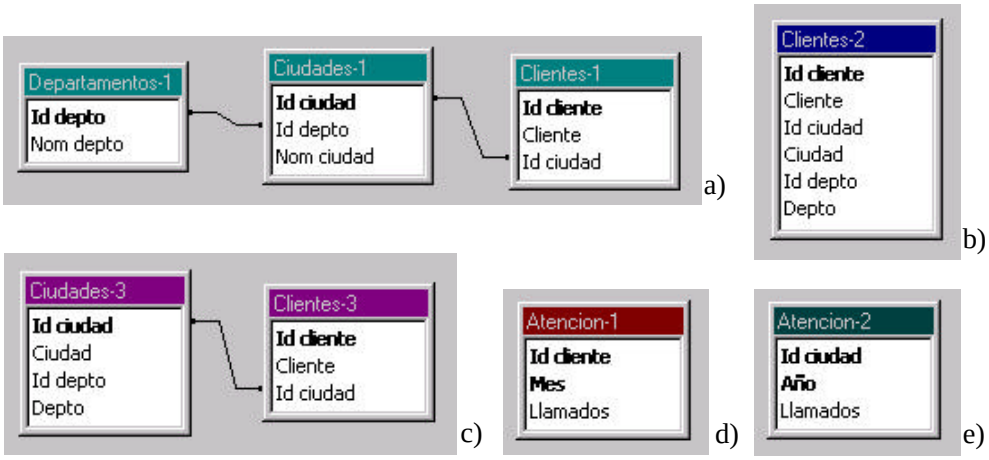

Figura 9 – Diferentes estrategias de generación de estructuras relacionales

Las tablas de hechos se pueden particionar horizontalmente de manera de mantener tablas con menos registros que mejoren la performance de las consultas de los usuarios. Se puede, por ejemplo, tener una tabla con los datos de los últimos dos años, y tener los datos históricos en otra tabla.

El diseñador debe decidir mediante qué criterios fragmentar horizontalmente las tablas de hechos.

A estos tipos de decisiones que debe tomar el diseñador se le denominan *lineamientos* (guidelines). Se trata de información que complementa al esquema conceptual con estrategias de diseño. Al definir los lineamientos el diseñador está dando pautas de alto nivel, de cómo debe ser el esquema lógico del DW.

Una vez definidos los lineamientos, se debe relacionar el esquema conceptual con la base de datos fuente integrada. La Figura 10 muestra una posible base de datos de la empresa con tres maestros: de clientes, ciudades y departamentos, y una tabla donde se registran los llamados de los clientes.

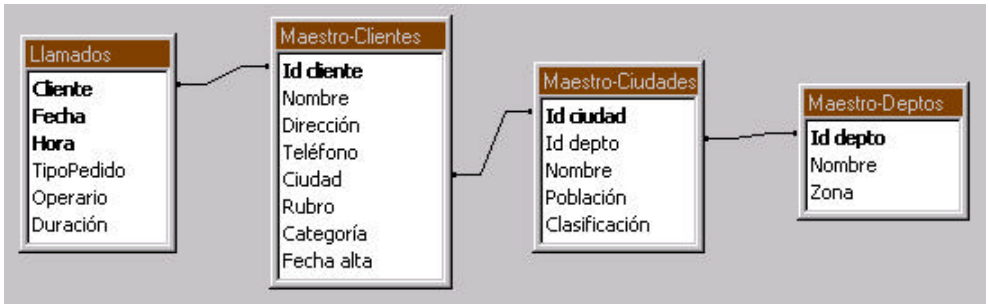

Figura 10 – Base fuente

El diseñador debe indicar de qué datos fuentes (tablas y atributos) se tomarán los datos para cargar el DW. Para ello indica como se corresponden los ítems del esquema conceptual con los atributos de la base fuente.

Por ejemplo, puede indicar que los ítems del nivel ciudad se mapean a los atributos *Id ciudad* y *Nombre* de la tabla *Maestro-Ciudades*, y los ítems *mes* y *año* se calculan a partir del atributo *Fecha* de la tabla *Llamados*.

En las siguientes secciones se formalizan estos conceptos.

3. Notación

Nombres de variables

Se utilizan variables en singular (A, L, X) para nombrar objetos y en plural (As, Ls, Xs) para nombrar conjuntos.

Se utiliza el prefijo Sch para definir los objetos o conjuntos de objetos principales que se utilizan a lo largo del documento.

Funciones y Conjuntos

As→ Bs representa el conjunto de las funciones que van del conjunto As en el conjunto Bs.

Set(As) representa el conjunto potencia del conjunto As (el conjunto de los subconjuntos finitos de As).

Seq(As) representa el conjunto de secuencias de elementos del conjunto As.

PartialOrders(As) representa el conjunto de los órdenes parciales sobre el conjunto As.

As denota la cardinalidad del conjunto As.

Expresiones Tipadas

Expressions(As) es el conjunto de expresiones tipadas definidas a partir de los elementos del conjunto As. Los elementos del conjunto As son parejas <variable, tipo>.

Sea BasicTypes el conjunto de los tipos básicos. Se consideran inicialmente los tipos *boolean, integer, float, string* y *date*. BasicTypes se define en la sección 1.4 del Anexo 1. Dado T ∈ BasicTypes, Instance(T) es el conjunto de valores del tipo T.

Dados T, U, V ∈ BasicTypes se define Operators(T,U,V) como el conjunto de operadores binarios que van de T x U en V, es decir:

Operators(T,U,V) ≡ T x U → V

Se define inductivamente el conjunto de expresiones válidas.

Si T ∈ BasicTypes ∧ E ∈ Instance(*T)* entonces <*E,T*> ∈ Expressions(As)
Si <*E,T*> ∈ As entonces <*E,T*> ∈ Expressions(As)
Si <*E,T*> ∈ Expressions(As) ∧ <*D,U*> ∈ Expressions(As) ∧ *q* ∈ Operators(T,U,V) ∧ V ∈ BasicTypes
entonces <*E q D, V*> ∈ Expressions(As)

Predicados

Predicates(As) es el conjunto de predicados definidos a partir de los elementos del conjunto As.

Se define inductivamente el conjunto de predicados válidos.

Si <*E,T*> ∈ Expressions(As) ∧ <*D,U*> ∈ Expressions(As) ∧ *q* ∈ Operators(T,U,boolean)
entonces *E q D* ∈ Predicates(As)

Expresiones de Resumen

RollUpExpressions(As) es el conjunto de expresiones tipadas definidas a partir de los elementos del conjunto As, que involucran operaciones de resumen (roll-up).

Como operaciones de roll-up se utilizan: count, min, max, sum y avg.

Se define inductivamente el conjunto:

Si $<E,T> \in$ Expressions(As) entonces <count(*E), integer*> $\in$ RollUpExpressions(As)
Si $<E,T> \in$ Expressions(As) $\wedge\ j \in$ {min, max} entonces $<j(E),T> \in$ RollUpExpressions(As)
Si $<E,T> \in$ Expressions(As) $\wedge\ T \in$ {integer, float} $\wedge\ j \in$ {sum, avg}
entonces $<j(E),T> \in$ RollUpExpressions(As)

Operador •

Se utiliza el operador • para agregar un prefijo al primer campo de una expresión simple tipada. El prefijo se toma del primer campo de una expresión.

Sea Es $\subseteq$ { <S,T> / S $\in$ Strings $\wedge$ T $\in$ BasicTypes }

Sea B=<P,…> / P $\in$ Strings

Se define: B•Es = { <Concat(Concat(B.P, "."), E.S), E.T> / E $\in$ Es }

Esta definición será de utilidad para concatenar nombres de tablas a atributos tipados, que se utilizarán como condiciones y funciones SQL.

Operador ¨

Se utiliza el operador ♦ para componer selección de campos a los elementos de un conjunto:

Sean: As = $Ds_1 \times Ds_2 \times \ldots$ Bs $\times \ldots Ds_N$. Sea Cs $\subseteq$ As

Se define: Cs♦Bs ≡ {C.Bs / C $\in$ Cs}

El operador también se utiliza para componer la aplicación de una función a los elementos de un conjunto:

Sea f: As → Bs. Sea Cs $\subseteq$ As

Se define: Cs♦f = {f(C) / C $\in$ Cs}

4. Especificación del Esquema Conceptual

Como especificación del esquema conceptual se utiliza el modelo conceptual multidimensional CMDM propuesto por Carpani en [Car00].

En CMDM se definen los conceptos de nivel, dimensión y relación dimensional y se presenta un lenguaje para especificar restricciones de integridad [Car01].

La Figura 11a muestra la definición gráfica de la dimensión *clientes* en CMDM. El nombre de la dimensión está en la esquina superior izquierda del recuadro externo. Los cuadros de texto son los niveles de la dimensión; sus nombres aparecen en negrita. Las jerarquías de niveles se representan por flechas entre los niveles[7], del nivel con más detalle (hijo[8]) al nivel con menos detalle (padre). La Figura 11b muestra la definición de la relación dimensional *venta* en CMDM. En la relación se cruzan las dimensiones *fechas, vendedores, clientes, artículos* y *cantidades*.

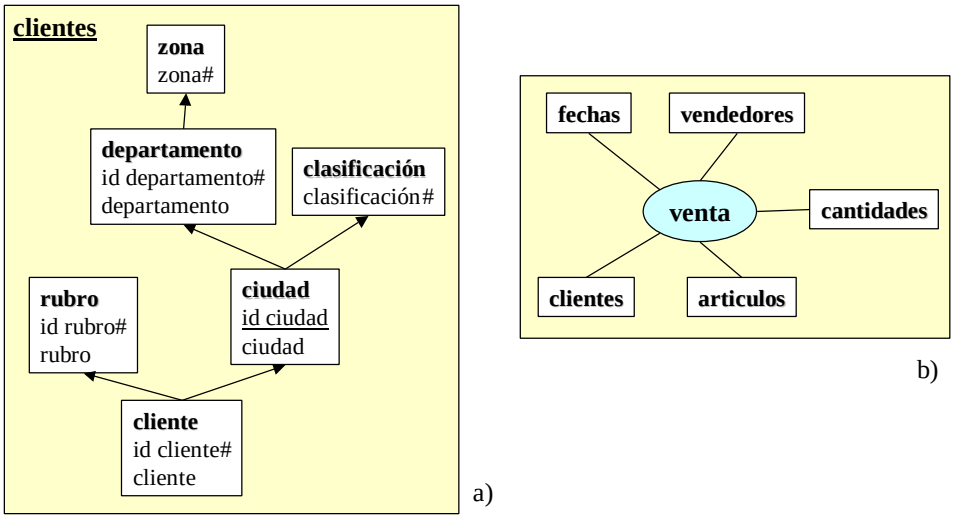

Figura 11 – Notación gráfica de CMDM: a) dimensiones y b) relaciones dimensionales

La descripción y especificación del modelo pueden encontrarse en [Car00]. En dicho trabajo no sólo se proponen las estructuras básicas del modelo, también se presenta un meta-lenguaje para instanciarlo a casos particulares.

4.1. Restricción a los tipos de datos del modelo CMDM

CMDM no restringe los tipos de datos los niveles, sin embargo, a la hora de asignarle estructuras de datos (en este caso representarlo en estructuras relacionales) es necesario acotar dichos tipos. En este trabajo se consideran niveles compuestos por uno o varios atributos, cada uno de ellos de un tipo simple. A los atributos de los niveles se les llama *ítems*.

[7] Las jerarquías de una dimensión pueden verse como un grafo dirigido y acíclico, en el que cada nodo es un nivel de la dimensión y cada arista indica que el nodo de entrada es un agrupamiento de elementos del nodo de salida. Si además de los niveles relacionados directamente (a través de una arista) se consideran las relaciones transitivas, se obtiene un orden parcial de los niveles.

[8] En este contexto se llama padre e hijo respectivamente a los nodos de entrada y salida de una arista.

Gráficamente los ítems aparecen debajo del nombre del nivel. Los ítems que identifican un nivel llevan un numeral (#) a la derecha (clave absoluta), y los que lo identifican sólo con respecto a su padre (clave débil o relativa) aparecen subrayados. En el ejemplo de la Figura 11a el nivel *ciudad* de la dimensión *clientes* está compuesto de 2 ítems: *id-ciudad* y *ciudad*. El ítem *id-ciudad* identifica a una ciudad dentro del departamento.

Utilizando las facilidades del propio modelo se propone una restricción a los tipos de datos de los niveles. En el Anexo 1 se presenta la especificación del modelo CMDM y de la restricción a los tipos.

A continuación se presenta la especificación de CMDM[9] luego de dicha restricción:

- CONCEPTUALSCHEMAS ≡ { <NAME, LEVS, DIMS, RELS, CONSTS > /
  NAME ∈ STRINGS ∧
  LEVS ∈ LEVELS ∧
  DIMS ⊆ DIMENSIONS(LEVS) ∧
  RELS ⊆ RELATIONS(DIMS) ∧
  CONSTS ⊆ FORM } [10]
- LEVELS ≡ { <LEVELNAME, IS, LCONSTS > /
  LEVELNAME ∈ STRINGS ∧
  IS ⊆ ITEMS ∧
  LCONSTS ⊆ FORM }
- DIMENSIONS (LEVS) ≡ { <DIMNAME, LS, PO, DCONSTS > /
  DIMNAME ∈ STRINGS ∧
  LS ⊆ LEVS ∧
  PO ∈ PARTIALORDERS(L) ∧
  DCONSTS ⊆ FORM } [11]
- RELATIONS (DIMS) ≡ { <RELNAME, DS, RCONSTS > /
  RELNAME ∈ STRINGS ∧
  DS ⊆ DIMS ∧
  RCONSTS ⊆ FORM }

Definición 1 – ConceptualSchemas

Un modelo conceptual está formado por un nombre, un conjunto de niveles, un conjunto de dimensiones que jerarquizan los niveles, un conjunto de relaciones que vinculan las dimensiones y un conjunto de restricciones.

Abreviaciones:

En lo sucesivo se trabaja con un único esquema conceptual: CONSCH ∈ CONCEPTUALSCHEMAS, por lo que se omite nombrarlo. Y se abrevia:

- SCHITEMS ≡ CONSCH.LEVS ♦ IS
- SCHLEVELS ≡ CONSCH.LEVS
- SCHDIMENSIONS ≡ CONSCH.DIMS
- SCHRELATIONS ≡ CONSCH.RELS ÿ

[9] Las estructuras de datos presentadas se construyen realizando un parsing de un esquema conceptual de CMDM.

[10] FORM es el conjunto de restricciones que se pueden definir en CMDM.

[11] PARTIALORDERS(Ls) es el conjunto de los órdenes parciales sobre el conjunto Ls.

4.2. Otras restricciones sobre CMDM

Además de la restricción a los tipos de datos de los niveles, en este trabajo se imponen algunas condiciones más al esquema conceptual para que se encuentre en las hipótesis de trabajo.

Unicidad de elementos

La definición de niveles contempla que un ítem pueda formar parte de varios niveles (ver Definición 1). Si bien la definición lo admite, en la práctica no es cómodo para manipularlos. Este problema es fácilmente solucionable definiendo ítems diferentes.

Por ejemplo, en la dimensión *productos* de la Figura 12a. Tanto el nivel *familia* como el nivel *producto* tienen un ítem *descripción*. La Figura 12b muestra la dimensión *productos* utilizando dos ítems diferentes para representar las descripciones: *desc-familia* y *desc-producto*.

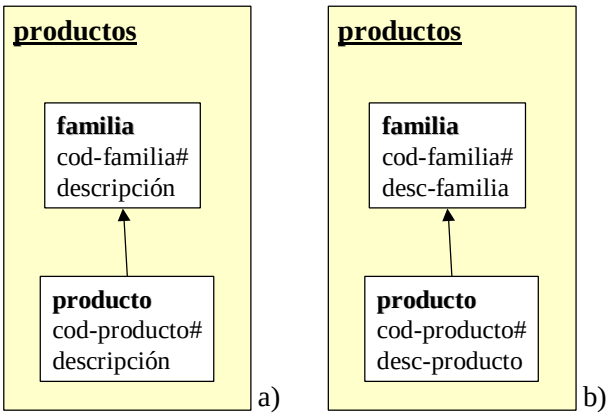

Figura 12 – Definición de niveles: a) compartiendo ítems, b) con ítems diferentes

Se exige que los conjuntos de ítems de los niveles sean disjuntos (los ítems son únicos por esquema). Análogamente se exige que los conjuntos de niveles de las dimensiones sean disjuntos (los niveles sean únicos por esquema).

El diseñador eventualmente tendrá que duplicar algunos elementos y renombrarlos para cumplir con esta condición.

Esta restricción nos permite saber a qué nivel pertenece un ítem dado, y a qué dimensión pertenece un nivel dado. En el Anexo 2 se definen algunas funciones que utilizan estas propiedades.

Claves de niveles

A través de las restricciones CMDM provee los mecanismos para definir claves a los niveles. Se pueden definir dos tipos de claves: relativas o absolutas, según identifiquen al nivel respecto a su padre en la jerarquía de la dimensión o lo identifiquen absolutamente. Esto último es análogo al concepto de entidad débil en el modelo E/R.

Para poder representar los elementos del esquema conceptual en estructuras relacionales se necesita identificar dichos elementos. Por tanto se exige que *todo nivel tenga una clave*, ya sea absoluta o relativa.

La clave en el esquema relacional, se obtendrá a partir de las claves absolutas de los niveles, o, para los niveles que no tienen clave absoluta, se puede construir una incorporando a la clave relativa los ítems que conforman la clave absoluta de los niveles padres. Para ello se define la función LevelKey que devuelve los ítems que identifican al nivel en forma absoluta. Su construcción se muestra en el Anexo 2.

- LEVELKEY : SCHLEVELS → SET (SCHITEMS)

5. Lineamientos de Diseño

Los lineamientos son información de diseño lógico que complementan al esquema conceptual y permiten al diseñador dar pautas sobre el esquema lógico deseado para el DW.

A través de los lineamientos, el diseñador define el estilo de diseño para el DW (snowflakes, estrella, mixto) e indica requerimientos de performance y almacenamiento (por ejemplo indicando que cubos implementar), etc.

En todos los casos se trata de información adicional ingresada por el diseñador en forma explícita.

A continuación se describen los lineamientos propuestos. El conjunto de lineamientos es extensible.

5.1. Materialización de Relaciones

En CMDM, una relación dimensional representa un espacio de cubos resultante de cruzar niveles de las dimensiones. Dicho espacio de cubos puede restringirse mediante las restricciones de integridad del propio modelo.

Las restricciones se construyen en base a predicados con cuantificadores (∀, ∃, ¬) para indicar que *"todos los cubos deben tener"*, o *"debe existir un cubo que tenga"* o *"ningún cubo debe tener"* dicha estructura. Por ejemplo: debe existir un cubo que cruce el nivel mes con el nivel producto.

Estas restricciones sugieren qué cubos sería interesante tener y cuáles no deberían existir. Sin embargo, la decisión de cuáles de esos cubos se deben materializar debe ser tomada en un momento posterior. En este contexto, materializar un cubo corresponde a precalcular los valores para los cruzamientos de las dimensiones y almacenarlos en una tabla. Luego se pueden obtener otros cruzamientos mediante operaciones efectuadas sobre éstos.

En CMDM, un cubo se define mediante una macro que indica los niveles y la medida del cubo [Car00]. Un cubo debe tener al menos un nivel de cada dimensión, uno de ellos en el rol de medida. Consideramos que esta característica es demasiado restrictiva y excluye algunos casos de interés y que se necesita una definición de cubos más libre o relajada, donde además de los cubos que se pueden especificar en CMDM se puedan representar:

- Cubos que no tengan medidas. Esto es interesante para representar sólo cruzamientos.
- Cubos que omitan algunas de las dimensiones de la relación dimensional. Esto tiene sentido cuando se quiere sumarizar totalmente la dimensión, es decir, no se quiere detalle a ningún nivel sino simplemente el total de la dimensión.

Para algunas dimensiones puede interesar mantener más de un nivel de detalle. Esto último tiene sentido si los niveles pertenecen a diferentes jerarquías, pero formalmente no se restringe.

Como lineamiento, el diseñador debe indicar el conjunto de cubos que serán implementados y almacenados físicamente en el DW. Para que la materialización se corresponda con el esquema conceptual, se debe indicar al menos un cubo que materialice cada relación dimensional.

La definición de qué cubos implementar se hace por extensión. A ese conjunto de cubos a materializar se le llama SchCubes.

A continuación se define la estructura que deben tener dichos cubos. Además del conjunto de niveles y la medida (opcional) con que se representan en CMDM, un cubo tiene un nombre y una referencia a la relación dimensional que materializa. Estos datos se agregaron para facilitar la manipulación de las estructuras.

- SCHCUBES ⊆ CUBES
- CUBES ≡ {<CUBENAME, R, LS, MEASURE> /
  CUBENAME ∈ STRINGS ∧
  R ∈ SCHRELATIONS ∧
  LS ⊆ R.DS♦LS ∧
  MEASURE ∈ (LS ∪ ⊥)}

Definición 2 – SchCubes

Un cubo está formado por un nombre, una relación a la cual materializa, un conjunto de niveles que conforman su nivel de detalle y opcionalmente un nivel que es elegido como medida. El conjunto SchCubes es definido por extensión por el diseñador.

Gráficamente se representan por un cubo unido a varios niveles (cuadros de texto) que conforman el nivel de detalle. La medida es el nivel marcado por una flecha. Dentro del cubo está su nombre y la relación que materializa (entre paréntesis). La Figura 13 muestra la materialización *venta-1* de la relación dimensional *venta* presentada en la Figura 11. Tiene por niveles: *mes*, *artículo*, *cliente*, *vendedor* y *cantidades*, y por medida: *cantidades*.

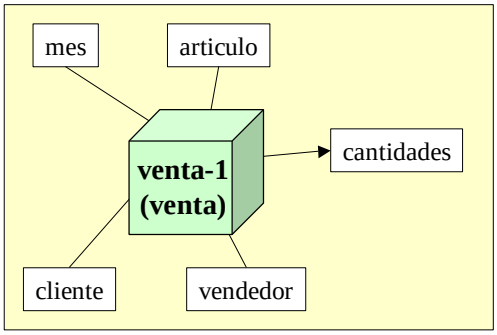

Figura 13 – Materialización de relaciones (Cubos)

5.2. Fragmentación Vertical de Dimensiones

El diseñador puede indicar el grado de normalización que quiere lograr al generar estructuras relacionales para cada dimensión. Por ejemplo, le puede interesar un esquema estrella, es decir, denormalizar todas las dimensiones y mantener fact tables. Por el contrario, le puede interesar un esquema de snowflakes, normalizando todas las dimensiones.

También le puede interesar tratar diferente cada dimensión, indicando para cada una si normaliza, denormaliza o efectúa una estrategia intermedia, indicando en este último caso, qué niveles quedan en la misma tabla.

Como lineamiento, el diseñador debe indicar para cada dimensión qué niveles desea almacenar juntos, conformando una fragmentación de los niveles de la dimensión.

Para ello define por extensión una función que, a cada dimensión le hace corresponder una fragmentación de sus niveles. A esa función se le llama SchDFragmentation.

Para que una fragmentación tenga sentido los niveles de cada fragmento deben estar relacionados jerárquicamente. Es decir, si se considera el grafo que representa a la jerarquía de la dimensión, el sub-grafo inducido por los niveles del fragmento debe ser conexo.

Si dos niveles de un mismo fragmento no están relacionados, ni directa ni transitivamente, entonces conforman un cruzamiento y se pierde la relación jerárquica de la dimensión. Por ejemplo, en la dimensión *clientes* de la Figura 11a no tiene sentido formar un fragmento con los niveles *rubro* y *ciudad* ya que no hay dependencia jerárquica entre ellos.

La fragmentación además debe ser completa, es decir que todos los niveles deben estar en al menos un fragmento para no perder información. Si no se quiere duplicar el almacenamiento de información de cada nivel, los fragmentos deben ser disjuntos, pero no es una exigencia. El diseñador decide cuando duplicar información de acuerdo a su estrategia de diseño.

A continuación se presenta la definición de fragmentación dimensional:

- SchDFragmentation ∈ {D / D ∈ SchDimensions} → CompleteFragmentSets(D) }
- CompleteFragmentSets (D) ≡ {Fs / Fs ⊆ Fragments (D) ∧ ∀L ∈ D.Ls . (
  ∃ F ∈ Fs . (L ∈ F)) }
- Fragments (D) ≡ {F / F ⊆ D.Ls ∧ ∀A,B ∈ F . (
  <A,B> ∈ D.PO ∨
  <B,A> ∈ D.PO ∨
  ∃ C ∈ F . (<A,C> ∈ D.PO ∧ <B,C> ∈ D.PO)) }

Definición 3 – SchDFragmentation

Un fragmento es un subconjunto de los niveles de la dimensión, que conforman un sub-grafo conexo de su jerarquía. Un conjunto completo de fragmentos cumple que cada nivel está al menos en uno de los fragmentos.

Una fragmentación dimensional es una función que a cada dimensión le asocia un conjunto completo de fragmentos. La función la define por extensión el diseñador.

Gráficamente se representa una fragmentación como una coloración de los niveles. Los niveles de un mismo fragmento se recuadran con el mismo color. Un nivel tiene varios colores si está en más de un fragmento. La fragmentación es completa si todos los niveles tienen color. En la Figura 14a se muestra una fragmentación de la dimensión *clientes* presentada en la Figura 11a. La fragmentación tiene 3 fragmentos disjuntos: *rubro y cliente* (celeste), *zona, departamento y ciudad* (verde) y *clasificación* (rosa). Los fragmentos de la Figura 14b no son disjuntos ya que el nivel *ciudad* pertenece al fragmento verde y al rosa.

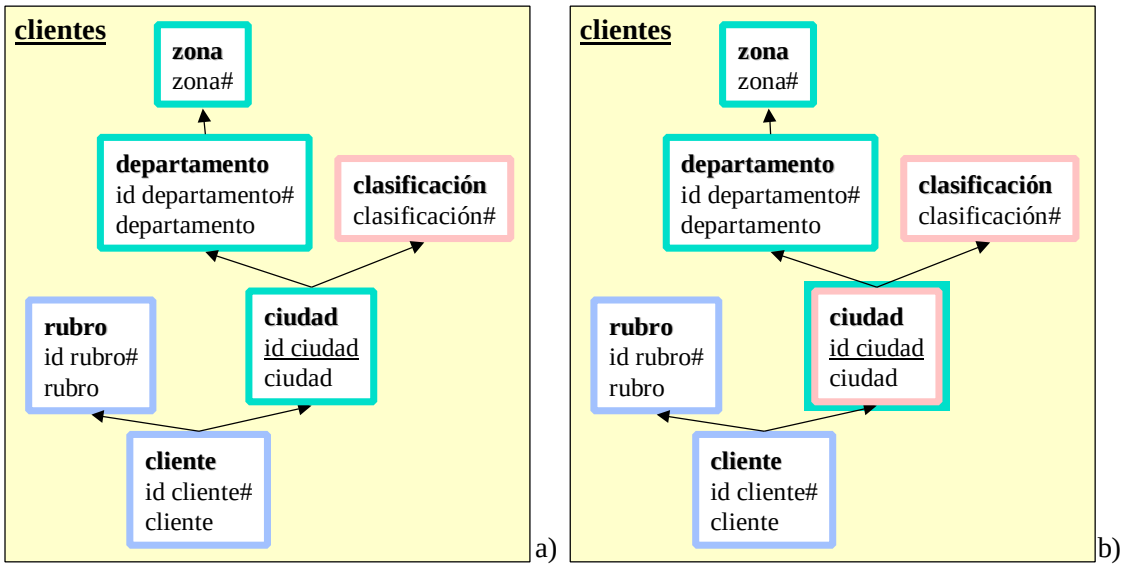

Figura 14 – Fragmentación vertical de dimensiones

Abreviaciones:

En lo sucesivo se llama SchFragments a todos los fragmentos de las dimensiones del esquema conceptual:

- SCHFRAGMENTS ≡ SCHDIMENSIONS ♦ SCHDFRAGMENTATION

5.3. Fragmentación Horizontal de Cubos

Representar un cubo en el modelo relacional puede dar como resultado una o varias tablas (fact tables) dependiendo del grado de fragmentación que se quiera lograr.

Fragmentar horizontalmente una tabla relacional corresponde a construir varias tablas con la misma estructura, y dividir las instancias entre ellas. Con esto se logra almacenar juntas las tuplas que son consultadas juntas y tener tablas más pequeñas, lo cual resulta en un aumento de la performance en las consultas.

Como ejemplo se considera el cubo venta-1 de la Figura 13, cuyas consultas más frecuentes corresponden a las ventas posteriores al 2000. Se puede fragmentar el cubo en dos fragmentos, uno para almacenar las tuplas correspondientes a ventas posteriores a ene-2000 y otro para las tuplas correspondientes a meses anteriores. Las tuplas de cada fragmento deben cumplir, respectivamente:

- mes ≥ ene-2000
- mes < ene-2000

La fragmentación horizontal para bases de datos distribuidas es estudiada por Özsu y Valduriez en [Ozs91]. En su trabajo proponen dos propiedades que debe cumplir la fragmentación horizontal: *completitud* y *disjuntez*. La propiedad de completitud exige que cada tupla esté en alguno de los fragmentos, y la propiedad de disjuntez exige que cada tupla que esté en a lo sumo un fragmento.

Como ejemplo se considera la siguiente fragmentación:

- mes ≥ ene-2000
- mes ≥ ene-1997 ∧ mes < ene-1999
- mes < ene-1998

La fragmentación no es completa, ya que las tuplas correspondientes a 1999 no pertenecen a ningún fragmento; y no es disjunta, ya que las tuplas correspondientes a 1997 pertenecen a las dos fragmentos.

En el contexto de bases de datos distribuidas, estas propiedades deben cumplirse para asegurar la completitud de los datos consultados, minimizando la redundancia. En el contexto de DW, es importante considerar estas propiedades, pero no es necesario que se cumplan. Si no se cumple la propiedad de disjuntez se obtiene redundancia, pero esto no es necesariamente un problema en un DW. Por ejemplo, es común mantener tablas de hechos con toda la historia, y tablas de hechos para los datos del último año. Respecto a la completitud, si bien interesa evitar la pérdida de información a nivel global, no siempre debe cumplirse localmente en la fragmentación de cada cubo.

Como ejemplo se consideran dos cubos que materializan la relación dimensional *venta* definida en la Figura 11, uno con detalle mensual y otro con detalle anual. Se realiza la siguiente fragmentación sobre los mismos:

cubo mensual	Cubo anual
– mes ≥ ene-2000	– mes ≥ ene-2000
– mes < ene-2000	

La fragmentación del cubo mensual es completa, pero la del cubo anual sólo tiene información de los últimos años. Sin embargo, las tuplas que no pertenecen al fragmento del cubo anual, pueden obtenerse a partir de los fragmentos del cubo mensual, por lo que no hay pérdida de información.

Al haber redundancia, la completitud de las fragmentaciones debe ser considerada globalmente. En este trabajo no se exigirá formalmente que se cumplan las propiedades, pero se sugiere tenerlas en cuenta durante la definición de la fragmentación.

Özsu y Valduriez definen un algoritmo que construye una fragmentación que cumple las propiedades de completitud y disjuntez [Ozs91]. El algoritmo construye un conjunto de predicados conjuntivos[12] que definen los fragmentos, a partir de un conjunto de predicados simples. En este trabajo se consideran predicados más generales (ver sección 3) llamados *franjas* (strips).

Para definir una fragmentación el diseñador debe indicar el conjunto de franjas a utilizar. Las franjas se expresan en términos de los ítems de los niveles del cubo, y se expresarán en términos de atributos de tablas en una etapa posterior. El diseñador puede utilizar franjas construidas con el algoritmo de [Ozs91] o construirlas con sus propios criterios.

A continuación se presenta la definición de fragmentación de cubos:

- SCHCFRAGMENTATION ∈ {C / C ∈ SCHCUBES} → {SET(PREDICATES(C.LS♦IS))}

Definición 4 – SchCFragmentation

Una fragmentación de cubos es una función que para cada cubo indica el conjunto de franjas en que se fragmenta.

Gráficamente la fragmentación se representa como un bloque de llamada a partir del cubo. Dentro del bloque se escriben las franjas. La Figura 15 muestra un conjunto de franjas asociado al cubo venta-1 definido en la Figura 13. Las dos primeras franjas determinan las ventas posteriores a enero de 2000 del vendedor Pérez y del resto de los vendedores (menos Pérez). La tercera franja determina las ventas anteriores a ene de 2000 de cualquier vendedor. La fragmentación es completa y disjunta.

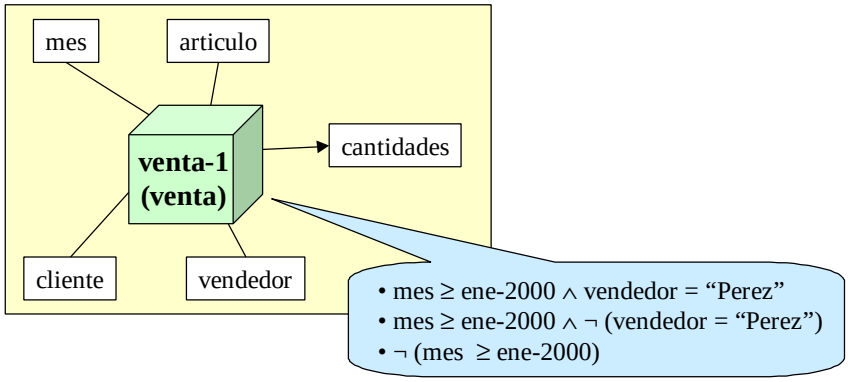

Figura 15 – Fragmentación horizontal de cubos

Abreviaciones:

En lo sucesivo se llama SchStrips a todas las franjas de todos cubos:

- SCHSTRIPS ≡ SCHCUBES♦SCHCFRAGMENTATION

[12] Los predicados conjuntivos, llamados *minterm* [Ozs91], son conjunciones de predicados simples y sus negaciones, de la forma: A_i *q value*, donde A_i es un atributo de la tabla, *value* es un valor del dominio de A_i y $q \in \{ =, <, \neq, \leq, >, \geq \}$

5.4. Definición de Lineamientos

El diseñador debe definir en forma explícita:

- El conjunto SchCubes, que indica qué cubos se desean implementar.
- La función SchDFragmentation, que indica cómo se fragmentan verticalmente las dimensiones.
- La función SchCFragmentation, que indica cómo se fragmentan horizontalmente los cubos.

Para definir los lineamientos el diseñador usualmente se basa en las restricciones de performance y almacenamiento de su sistema.

El conjunto SchCubes debe tener al menos un cubo por cada relación dimensional y con el máximo detalle para no perder información. Si se tienen limitaciones de espacio se puede implementar sólo un cubo de cada relación. Si en cambio el espacio físico no es un problema y se quiere priorizar la performance en las consultas, se puede materializar otros cubos para los cruzamientos más frecuentes o más exigentes. En general es inviable materializar todos los cruzamientos posibles, por lo tanto la elección de los cubos a materializar resulta en un compromiso entre costos de almacenamiento y tiempos de respuesta.

La forma de fragmentar las dimensiones tiene que ver con el estilo de diseño que se quiere dar al DW. Para lograr un esquema estrella se define un único fragmento por cada dimensión (con todos los niveles), en cambio, para un esquema snowflake se define un fragmento por cada nivel. La denormalización logra mejores tiempos de respuesta en las consultas ya que no es necesario realizar el join de varias tablas, pero tiene el máximo grado de redundancia. La redundancia es controlable si los datos de las dimensiones cambian lentamente, pero aumenta el tamaño de las tablas y por tanto degrada la performance cuando los datos son volátiles y se quieren conservar las distintas versiones. La normalización tiene los efectos contrarios. Las estrategias intermedias hacen un compromiso entre tiempos de respuesta y el control de redundancia.

En la fragmentación de los cubos intervienen dos factores: la cantidad de datos y los rangos de datos consultados juntos. Cuantas más franjas se tengan, cada una será de menor tamaño y por tanto se ganará en tiempos de respuesta. Por otro lado, si las consultas involucran datos de varias franjas, deben realizarse operaciones de drill-across (combinar datos de varios cubos) con efectos peores en los tiempos de respuesta. La decisión debe basarse en los requerimientos, observando qué datos son consultados juntos (afinidad), por ejemplo, qué rangos de fechas se consultan más frecuentemente. Las fragmentaciones más comunes se realizan por rangos de fechas o zonificaciones.

Se pueden definir estrategias por defecto que automaticen la definición de los lineamientos. Por ejemplo, una estrategia posible es construir siempre un esquema estrella con un único cubo por relación y una única franja por cubo. Otra estrategia podría consistir en fragmentar las dimensiones con más de 4 niveles, en fragmentos de 3 niveles cada uno, y crear franjas quinquenales para los cubos.

Podrían estudiarse funciones de costos y heurísticas que automaticen la definición de lineamientos de acuerdo a los factores mencionados: limitaciones de espacio físico, performance en las consultas, afinidad, cantidad de datos.

6. Esquema Intermedio

Al especificar los lineamientos el diseñador introduce elementos que complementan el esquema conceptual. En concreto, especifica qué cubos van a materializar las relaciones dimensionales y cómo se van a fragmentar las dimensiones y los cubos.

En esta sección se define una extensión al esquema conceptual, llamado esquema intermedio, para incorporar elementos de los lineamientos:

- INTERMEDIATESCHEMA ≡ < SCHITEMS, SCHLEVELS, SCHDIMENSIONS, SCHCUBES, SCHDFRAGMENTATION, SCHCFRAGMENTATION, CONSTS >

Definición 5 – IntermediateSchema

El esquema intermedio consiste de los ítems, niveles y dimensiones y restricciones del esquema conceptual, e incorpora los cubos, fragmentación de dimensiones y fragmentación de cubos definidos en los lineamientos.

Abreviaciones:

En lo sucesivo se utiliza el termino SchObjects (objetos del esquema intermedio) para referirse a ítems, niveles, dimensiones, fragmentos o cubos del esquema intermedio:

- SCHOBJECTS ≡ SCHITEMS ∪ SCHLEVELS ∪ SCHDIMENSIONS ∪ SCHFRAGMENTS ∪ SCHCUBES

7. Especificación de la Base de Datos Fuente

Se considera que la base de datos fuentes es una base relacional integrada, de la cual interesa representar las tablas, sus atributos (con sus tipos) y su clave primaria.

Dadas dos tablas, interesa reflejar cómo se vinculan, es decir, cómo se debe realizar el join entre ellas. A un vínculo de este tipo se le llama *link*. Cada link tiene asociado un predicado construido con los atributos de ambas tablas. Los links son relaciones simétricas.

Gráficamente la vinculación entre tablas puede expresarse mediante un grafo, donde los nodos son las tablas y las aristas son los links rotulados con los predicados.

Para los predicados de igualdad de atributos, que representan la mayoría de los casos, se omiten los rótulos en las aristas y se dibujan líneas que unen los atributos involucrados (tantas líneas como parejas de atributos a igualar). La Figura 16 muestra varias tablas fuentes y links definidos entre ellas con predicados de igualdad.

En el Anexo 3 se define el concepto de link y se presentan algunas ambigüedades y mecanismos para solucionarlas.

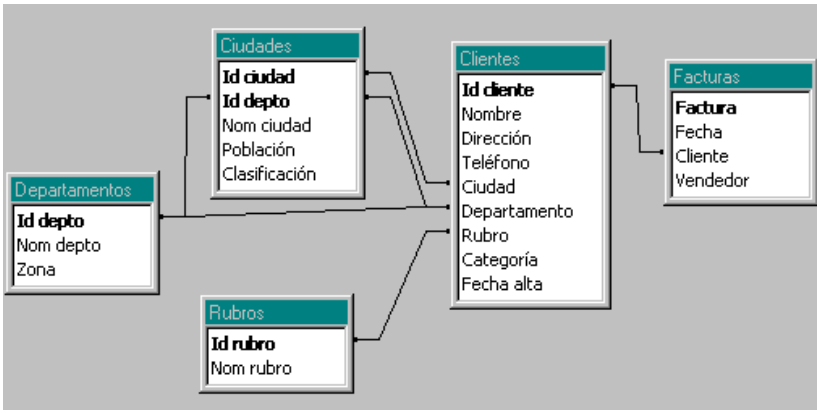

Figura 16 – Links entre tablas fuentes.

A continuación se presenta una abstracción de las bases fuentes:

- LogicalSchema ≡ < SchAttributes, SchTables, SchLinks >
- SchAttributes ⊆ { <AttName, Type > / AttName ∈ Strings ∧ Type ∈ BasicTypes }
- SchTables ⊆ { <TabName, As, PK> / TabName ∈ Strings ∧ As ⊆ SchAttributes ∧ PK ⊆ As }
- SchLinks ∈ {T_1/ T_1 ∈ SchTables} x {T_2 / T_2 ∈ SchTables} → Predicates (T_1 •As ∪ T_2 •As)[13]

Definición 6 – LogicalSchema

El esquema fuente está formado por un conjunto de atributos tipados, un conjunto de tablas que contienen a los atributos, algunos de los cuales son clave primaria de la tabla, y un conjunto de links entre las tablas.

[13] El operador • concatena el nombre de la tabla y del atributo, y mantiene el tipo del atributo (ver sección 3)

8. Mapeos entre el Esquema Intermedio y la Base de Datos Fuente

El esquema conceptual especifica la información que contendrá el DW, y a través de los lineamientos el diseñador indica las características que debe cumplir el esquema lógico. El esquema conceptual con el agregado de los lineamientos conforma el esquema intermedio.

Luego de construir el esquema intermedio, el siguiente paso es vincularlo con la base fuente. Para ello se establecen mapeos o correspondencias (mappings) que indican dónde se encuentran en el esquema lógico de la fuente los diferentes elementos del esquema intermedio.

Los mapeos son funciones que asocian a cada elemento del esquema intermedio una expresión construida en base a las tablas y atributos de la base de datos fuente. Son definidas por el diseñador en forma explícita.

Antes de definir los mapeos se presenta una caracterización de las expresiones involucradas.

8.1. Expresiones de Mapeo

En un mapeo, a cada ítem del esquema intermedio le corresponde una expresión construida en base a atributos de las tablas fuentes, a la que se llama **mapexpr** (expresión de mapeo).

Una expresión de mapeo puede ser un atributo de una tabla fuente (DirectME), o un cálculo que involucra varios atributos de una tupla (1calcME), o un resumen o totalización que involucra varios atributos de varias tuplas (NcalcME) o un valor externo a las fuentes como una constante, una estampa de tiempo o dígitos de versión (ExternME).

Una expresión se representa como una n-upla que tendrá diferentes campos dependiendo del tipo de expresión (DirectME, 1calcME, NcalcME, ExternME). Los campos utilizados son los siguientes (en plural representan conjuntos de elementos):

- Expr: es una expresión tipada (ver Expressions en la sección 3) construida en base a uno o más atributos de las tablas fuentes.
- Tab: es una tabla de la base fuente.
- Patt: es un atributo de una tabla de la base fuente con el nombre de la tabla como prefijo. En general se usa el operador • para concatenar el nombre de la tabla (ver sección 3).
- 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.

Las expresiones de mapeo son la unión de los distintos tipos de expresiones. Su definición es la siguiente:

- MAPEXPR ≡ DIRECTME ∪ 1CALCME ∪ NCALCME ∪ EXTERNME
- 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}

Definición 7 – MapExpr

8.2. Funciones de Mapeo

Definido MapExpr, se pueden definir funciones de mapeo:

- MAPPINGS(ITS) ≡ { F / F ∈ ITS → MAPEXPRS ∧ ∀ I ∈ ITS. (I.TYPE = F(I).EXPR.TYPE) }

Definición 8 – Mappings

Una función de mapeo hace corresponder a cada ítem, una expresión de mapeo, controlando que coincidan los tipos de ambos.

Las funciones de mapeo se utilizan en 2 contextos: vincular fragmentos de dimensiones a las tablas fuentes (mapeos de fragmentos), y vincular cubos a las tablas fuentes (mapeos de cubos). Esta vinculación se realiza definiendo una función de mapeo que haga corresponder los ítems del fragmento o cubo con las tablas fuentes.

Inicialmente se tiene una función de mapeo para cada fragmento de dimensión, y una función de mapeo para cada cubo. Al fragmentar los cubos (según SchCFragmentation) se obtiene una función para cada franja de cada cubo.

Gráficamente una función de mapeo se representa por flechas entre los ítems del esquema intermedio y los atributos de las tablas fuentes.

Cuando el mapeo es directo se representa con una línea corrida, cuando es un cálculo se representa con una línea cortada a cada atributo que interviene en el cálculo y se adjunta la definición del cálculo, y cuando es externo no se utilizan líneas pero se adjunta la expresión a la que mapea.

En la Figura 17 se muestra una función de mapeo para la dimensión *geografía*, con un único fragmento (rosa). El ítem *país* tiene un mapeo externo de tipo constante, el ítem *zona* tiene un mapeo calculado en base al atributo *zona* de la tabla *departamentos*. Los demás ítems tienen mapeos directos.

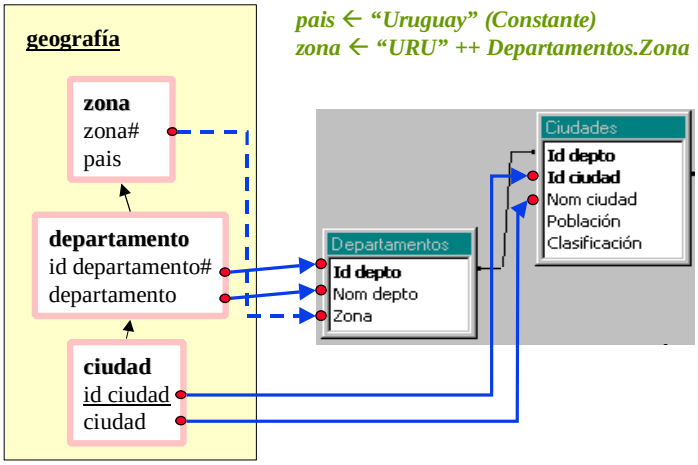

Figura 17 – Representación gráfica de una función de mapeo

Cada función de mapeo representa una vista o expresión SQL que tiene en el cabezal (SELECT) las expresiones de mapeo y obtiene los datos (FROM) de las tablas referenciadas en dichas expresiones, imponiendo los links entre las tablas como condición de join (WHERE). Las tablas involucradas deben tener links definidos, no tienen sentido los productos cartesianos.

En el ejemplo de la Figura 17, se asocia al fragmento de la dimensión *geografía* la siguiente vista SQL:

```
SELECT  "URU" ++ Departamentos.Zona as zona,
        "Uruguay" as pais,
        Departamentos.Id_depto as id_departamento,
        Departamentos.Nom_depto as departamento,
        Ciudades.Id_ciudad as id_ciudad,
        Ciudades.Nom_ciudad as ciudad
FROM Departamentos, Ciudades
WHERE Departamentos.Id_depto = Ciudades.Id_ciudad
```

Desde el punto de vista de las instancias, la instancia de la vista SQL debe coincidir con la instancia esperada para el fragmento o cubo que se quiere mapear (se mapea a sus ítems).

La vista SQL no se implementará, la generación del esquema lógico se hará a través de transformaciones de esquemas que conducirán al mismo resultado. Dichas transformaciones se estudian en el siguiente capítulo.

En la vinculación de un fragmento o un cubo pueden existir condiciones, por ejemplo que un atributo se encuentre en determinado rango. Estas condiciones pueden deberse a restricciones en el esquema conceptual o a restricciones que deseen aplicarse a las fuentes.

En el ejemplo de la Figura 17, el nivel *zona* puede tener una restricción indicando que sólo interesan las zonas menores a 10, que pueden ser por ejemplo, las zonas uruguayas. Esta es una restricción del esquema conceptual. También puede ocurrir que en la tabla *Ciudades* se almacenen ciudades necesarias en varios sistemas o secciones, y que para el DW sólo interesen las que tienen *Clasificación R*. Esta es una restricción respecto a las fuentes.

Ambas restricciones deben tenerse en cuenta al establecer los mapeos, e incorporar una condición tipo:

Departamentos.zona < 10 Ù Ciudades.Clasificación = "R"

Estas restricciones pueden verse como condiciones adicionales de la vista SQL. Para el ejemplo anterior se agregaría a la expresión:

```
AND Departamentos.Zona < 10
AND Ciudades.Clasificacion = "R"
```

Gráficamente las condiciones se representan como llamadas (callouts). La Figura 18 muestra la incorporación de las condiciones al mapeo de la Figura 17.

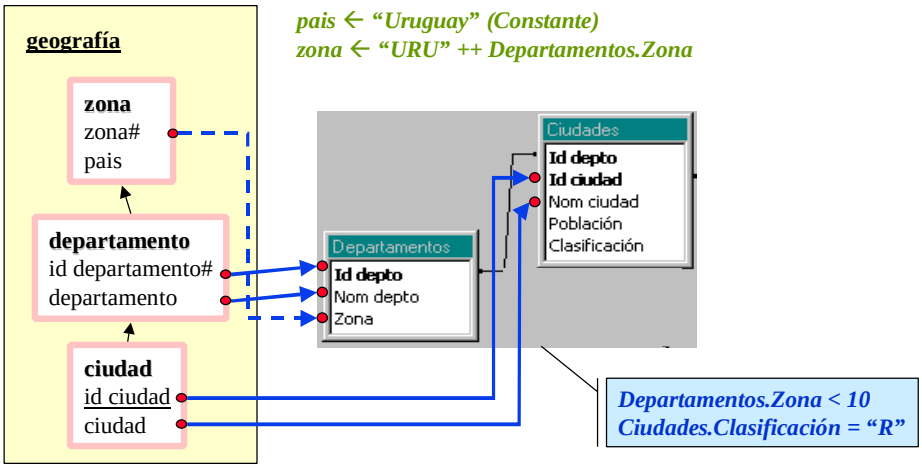

Figura 18 – Representación gráfica de condiciones de mapeo

Formalmente, las condiciones son predicados (ver sección 3) sobre atributos de las tablas fuentes. Como en una condición pueden intervenir atributos de varias tablas, se les pone como prefijo el nombre de la tabla para evitar ambigüedades.

Para definir las condiciones el diseñador debe tener en cuenta las restricciones del esquema conceptual y expresarlas en término de los atributos a los que mapean. Esta tarea se podría semi-automatizar tomando como entrada las restricciones del esquema conceptual. El diseñador además debe basarse en su conocimiento de las bases de datos fuentes.

8.3. Algunas definiciones

En las siguientes secciones se trabaja con expresiones de mapeo, en especial con mapeos directos (DirectME) y cálculos simples (1calcME). De ellas se quieren obtener elementos como los atributos a los que se mapea, y a qué tablas pertenecen.

Se utilizan algunas funciones auxiliares, definidas en forma de macros, que faciliten la manipulación de las expresiones. A continuación se listan dichas funciones:

Atributos mapeados

Dada una función de mapeo, se dice que un conjunto de ítems mapea a un atributo (o que el atributo es mapeado por el conjunto de ítems) cuando alguno de los ítems tiene un mapeo directo (DirectME) o cálculo simple (1calcME) con el atributo.

Se define una función que devuelve los atributos mapeados por un conjunto de ítems en una función de mapeo:

- 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))}

Definición 9 – MapAttributes

Tablas mapeadas

Dada una función de mapeo, se dice que un conjunto de ítems mapea a una tabla (o que la tabla es mapeada por el conjunto de ítems) si el conjunto de ítems mapea a uno de los atributos de la tabla.

Se define una función que devuelve las tablas mapeadas por un conjunto de ítems en una función de mapeo:

- 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) }

Definición 10 – MapTables

Ítems de los objetos

Para vincular un objeto del esquema intermedio (fragmento o cubo) a las bases fuentes, se definen funciones de mapeo que hacen corresponder los ítems "relevantes" del objeto con expresiones de mapeo.

Un fragmento debe mapear todos los ítems de los niveles que lo conforman, ya sean identificadores (pertenecen a la clave de un nivel) o descriptivos (no pertenecen a la clave). Notar que si un nivel tiene clave relativa, se debe mapear también a los ítems de niveles superiores que conforman su clave absoluta.

Una dimensión fragmentada debe poder reconstruirse sin pérdida de información. Para ello es necesario mantener la asociación entre los niveles, de manera de poder navegar en las jerarquías de la dimensión. Entonces, se debe mapear también las claves de los niveles padres en las jerarquías.

Por ejemplo, la Figura 19 muestra la fragmentación de la dimensión *productos* en dos fragmentos, con los niveles *familia* (rosa) y *producto* (azul).

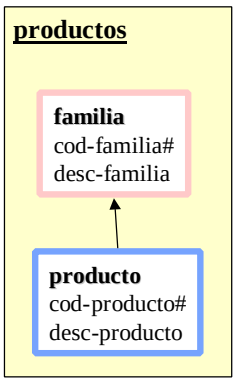

Figura 19 – Fragmentación de la dimensión productos

Se podría proponer funciones de mapeo solamente para los ítems de cada nivel. Se tendrían entonces las siguientes vistas SQL:

- *Familias* (*cod-familia*, *desc-familia*)
- *Productos* (*cod-producto*, *desc-producto*).

Pero con ese esquema no se puede reconstruir la jerarquía de la dimensión, es decir, se pierde información de a qué familia corresponde cada producto.

Una propuesta mejor consiste de las siguientes vistas:

- *Familias* (*cod-familia*, *desc-familia*)
- *Productos* (*cod-producto, desc-producto, cod-familia*).

Incorporar la clave del nivel *familia* a la vista *Productos* asegura la reconstrucción, propiedad indispensable en una fragmentación vertical [Ozs91].

Para un cubo, en cambio, se debe mapear los ítems necesarios para reconstruir el cruzamiento de los niveles que lo conforman. Para ello alcanza con mapear los ítems que identifican a los niveles. Notar que si un nivel tiene clave relativa, se debe mapear también a los ítems de niveles superiores que conforman su clave absoluta.

La función ObjectItems, definida en el Anexo 4, devuelve los ítems relevantes para el mapeo de los objetos. Estos son:

- Para los fragmentos: ítems de los niveles, ítems de las claves absolutas de los niveles, ítems de las claves absolutas de los niveles padres.
- Para los cubos: ítems de las claves absolutas de los niveles.

8.4. Mapeo de Fragmentos

Para mapear un fragmento debe definirse una función de mapeo para todos sus ítems, esto incluye algunos ítems de niveles superiores. Si corresponde, se debe definir una condición que deban cumplir los atributos de las fuentes.

Se define una función que a cada fragmento le hace corresponder una función de mapeo y una condición (opcional):

- SCHFMAPPINGS ∈ { F / F ∈ SCHFRAGMENTS} → {<MAP, COND> /
  MAP ∈ MAPPINGS (OBJECTITEMS(F))
  ∧ COND ∈ PREDICATES ({T•AS / T ∈ MAPTABLES(MAP, OBJECTITEMS(F))}) }

Definición 11 – SchFMappings

La función SCHFMAPPINGS asocia una función de mapeo a cada fragmento. Opcionalmente se puede asociar también una condición escrita en base a los atributos de las tablas a las que se mapea.

El diseñador debe definir por extensión las funciones de mapeo para cada fragmento, y especificar las condiciones en base a restricciones del modelo conceptual y su conocimiento de las bases de datos fuentes.

Cada función de mapeo de fragmento debe pensarse como una vista SQL. La instancia de esa vista debe coincidir con la instancia esperada para el fragmento. Si se piensa individualmente el mapeo de un ítem, pueden existir varios atributos a donde mapearlo. Por ejemplo: el ítem *id-ciudad* de la dimensión *geografía* de la Figura 18 puede mapearse al atributo *Id-ciudad* de la tabla *Ciudades* pero también puede mapearse al ítem *Ciudad* de la tabla *Clientes*. En la Figura 16 se muestran ambas tablas y su relación (link). Si bien ambos atributos contienen información sobre los identificadores de ciudades, la primera contiene a todas las ciudades y la segunda sólo a las ciudades donde vive algún cliente. La elección de a qué atributo mapear depende de qué ciudades deben poblar el DW: todas o en las que viven clientes.

Se presenta un problema si las claves foráneas no están bien definidas. Por ejemplo si hay ciudades en la tabla *Clientes* que no están en la tabla *Ciudades*. La elección del atributo adecuado en estos casos depende del conocimiento que tenga el diseñador sobre las fuentes.

En muchos casos puede elegirse arbitrariamente entre varios atributos sin cambiar el resultado. Por ejemplo el ítem *id-departamento* de la dimensión *geografía* de la Figura 18 puede mapearse al atributo *Id-depto* de la tabla *Departamentos* o al atributo *Id-depto* de la tabla *Ciudades* sin afectar el resultado ya que hay una condición de join por igualdad que los hace coincidir. Considerar el mapeo de fragmentos como una consulta SQL ayuda en la elección de los ítems adecuados.

8.5. Mapeo de Cubos

Análogamente, para mapear un cubo puede definirse una función de mapeo para sus ítems y una condición sobre las tablas mapeadas (mapeo base). Pero el mapeo de un cubo puede definirse recursivamente, en términos de un cubo ya mapeado que tenga más detalle (mapeo recursivo).

Para definir un cubo en términos de otro se debe especificar la secuencia de dimensiones a las que se quiere reducir el nivel de detalle (corresponde a una operación de drill-up) y las operaciones de roll-up asociadas a las medidas.

Para que un cubo Cr pueda ser mapeado recursivamente a partir de otro cubo Cb debe cumplirse:

- Cr.R = Cb.R */* materializan la misma relación dimensional */*
- Cr.Measure = Cb.Measure */* tienen las mismas medidas */*
- $\forall L \in Cr.Ls . ((L \in Cb.Ls) \vee (\exists B \in Cb.Ls . (<L,B> \in LDIM(L).PO))$[14] */* los niveles son los mismos o mayores en la jerarquía de una dimensión */*

Se utiliza la función de mapeo del cubo base para mapear los ítems de los niveles en común, pero la función se debe definir para los ítems en los que se diferencian. En particular, alcanza con mapear los ítems involucrados en la operación de drill-up de cada dimensión, es decir, los ítems del nivel a remplazar del cubo base y los ítems de los niveles que lo remplazan en el cubo que se quiere mapear recursivamente. Si este último no tiene detalle para la dimensión, porque se quiere la misma totalmente sumarizada, no es necesario mapear ningún ítem.

Por ejemplo: si un cubo Cb tiene como detalle *cliente* (de la dimensión *clientes*), y un cubo C2 se mapea en función del cubo Cb, con detalle *ciudad*, se debe indicar una función que mapee a los ítems clave de los niveles *cliente* y *ciudad*. Si un cubo C3 no tiene detalle para la dimensión *clientes* no es necesario mapear ningún ítem.

Un drill-up en una dimensión se define dando el nivel (del cubo base) a reemplazar, un conjunto de niveles con mayor jerarquía (del cubo a mapear recursivamente), un mapeo de los ítems clave de dichos niveles (sólo en caso querer detalle para la dimensión) y una operación de roll-up:

- DRILLUPS (CB, CR) ≡ DETAILDRILLUPS (DB,CR) ∪ NODETAILDRILLUPS (CB,CR)
- DETAILDRILLUPS (CB, CR) ≡ {<LBASE, LNEWS, MAP, RUP> / LBASE ∈ CB.LS
  ∧ LNEWS ⊆ CR.LS
  ∧ LNEWS ⊆ LDIM(LBASE).LS ∧ ∀L ∈ LNEWS . (<L, LBASE> ∈ LDIM(LBASE).PO)
  ∧ MAP ∈ MAPPINGS (OBJECTKEYITEMS(LBASE) ∪ LNEWS ♦ OBJECTKEYITEMS) [15]
  ∧ RUP ⊆ ROLLUPEXPRESSIONS(CR.MEASURE ♦ IS) }
- NODETAILDRILLUPS (CB, CR) ≡ {<LBASE, LNEWS, RUP> / LBASE ∈ CB.LS
  ∧ LNEWS ⊆ CR.LS
  ∧ LNEWS ⊆ LDIM(LBASE).LS ∧ ∀L ∈ LNEWS . (<L, LBASE> ∈ LDIM(LBASE).PO)
  ∧ RUP ⊆ ROLLUPEXPRESSIONS(CR.MEASURE ♦ IS) }

Definición 12 – DrillUps

[14] La función Ldim devuelve la dimensión a la que pertenece el nivel. Está descrita en el anexo 2.

[15] La función ObjectKeyItems devuelve los items clave del objeto. Está descrita en el anexo 4.

Entonces, un mapeo de cubos se define como una función que a cada cubo le hace corresponder, o bien una función de mapeo y una condición (mapeo base), o bien un cubo base, una secuencia de drill-ups y una función de mapeo base (mapeo recursivo):

- SCHCMAPPINGS ∈ {C / C ∈ SCHCUBES} → BASECMAPPINGS(C) ∪ RECURSIVECMAPPINGS(C)
- BASECMAPPINGS (C) ≡ {<MAP, COND, RUP> / MAP ∈ MAPPINGS (OBJECTITEMS(C)) ∧ RUP ⊆ ROLLUPEXPRESSIONS(C.MEASURE♦IS) ∧ COND ∈ PREDICATES ({T•AS / T ∈ MAPTABLES(MAP, OBJECTITEMS(C))}) }
- RECURSIVECMAPPINGS (C) ≡ {<CB, DUPS, MAP> / CB ∈ SCHCUBES ∧ DUPS ∈ SEQ(DRILLUPS(CB, C)) }[16] ∧ MAP ∈ MAPPINGS (OBJECTITEMS(CB))

Definición 13 – SchCMappings

La función SCHCMAPPING le asocia a cada cubo un mapeo base o un mapeo recursivo. Un mapeo base consiste de una función de mapeo con una condición y una función de roll-up por defecto; un mapeo recursivo consiste de un cubo base y una secuencia de expresiones de drill-up.

El diseñador debe definir por extensión el mapeo de cada cubo. Las funciones de mapeo para franjas se definirán automáticamente durante la etapa de generación del esquema lógico.

Gráficamente un mapeo base de cubo se representa en forma análoga a un mapeo de fragmento. En los niveles identificados por un único ítem pueden omitirse los ítems. En los niveles identificados por varios ítems deben detallarse todos. Dichos ítems pueden ser del nivel o identificadores de niveles superiores cuando el nivel tiene clave débil. Mediante un pentágono se especifican los predicados de roll-up para las medidas. La Figura 20 muestra una función de mapeo para el cubo venta-1.

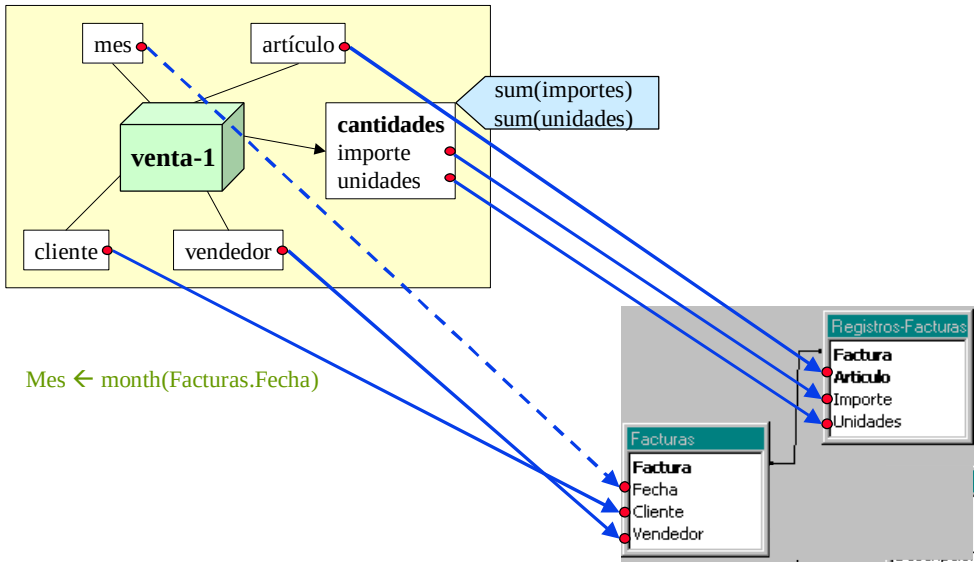

Figura 20 – Representación gráfica de un mapeo base de cubo

[16] Seq(A) es el conjunto de secuencias de elementos del conjunto A.

Un mapeo recursivo de cubo se representa uniendo los cubos por una flecha gruesa. Se expresa la función de mapeo entre los ítems de los niveles base y nuevos de manera análoga un mapeo base. Mediante un pentágono se especifican los predicados de roll-up para las medidas. La Figura 21 muestra el mapeo del cubo venta-2 en función del cubo venta-1, mediante un drill-up en la dimensión clientes (del nivel cliente, a los niveles ciudad y rubro).

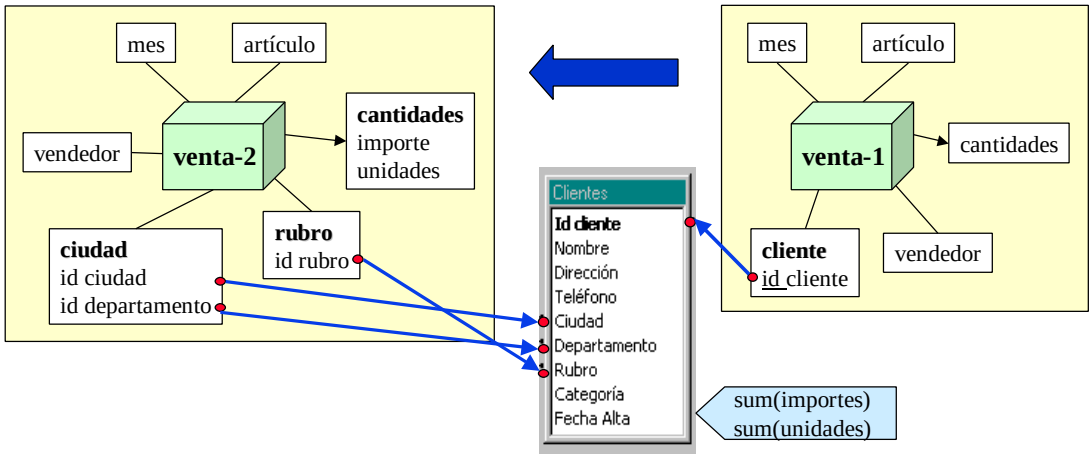

Figura 21 – Representación gráfica de un mapeo recursivo de cubo

Los mapeos base pueden pensarse como una consulta SQL al igual que los mapeos de fragmentos. Valen las mismas consideraciones de elección de atributos que para fragmentos, con la distinción de que se trata de realizar la menor cantidad posible de joins por razones de performance. Es decir, que siempre que se tengan definidas claves foráneas o se tenga confianza en la calidad de los datos, se puede evitar el join de algunas tablas. En el ejemplo de la Figura 21 se mapea el ítem *id-cliente* del nivel *cliente* al atributo *cliente* de la tabla *Facturas* en lugar de realizar un join con la tabla *Clientes* y mapearlo allí. La instancia de esa consulta SQL debe coincidir con la instancia esperada para el cubo.

Los mapeos recursivos simplifican la tarea de definición de mapeos. Alcanza con que el diseñador realice los mapeos de los drill-ups, en forma análoga a como mapea un fragmento. Los mapeos de los fragmentos de las dimensiones pueden ayudar a definir los mapeos de los drill-ups. Podría estudiarse un mecanismo semi-automático para deducirlos a partir de los mapeos de fragmentos.

9. Conclusiones

En este capítulo se presentó la primera etapa del proceso de diseño lógico. En esta etapa el diseñador tiene que realizar las siguientes tareas:

- Analizar el esquema conceptual (resultado del diseño conceptual) y si es necesario refinarlo para que sirva de entrada para el diseño lógico. En el refinamiento se deben definir tipos de datos adecuados para los niveles (concepto de ítem), asegurar la unicidad de ítems y niveles y definir claves para todos los niveles. Luego se puede proceder con el parsing del esquema conceptual.
- Definir lineamientos de diseño que incluyen: materialización de relaciones dimensionales (cubos), fragmentación de dimensiones (niveles que se desean almacenar juntos) y fragmentación de cubos (franjas).
- Analizar los esquemas fuentes. Dependiendo del manejador utilizado se debe complementar la información disponible en las fuentes de manera de tener definidas: tablas, atributos y links.
- Definir funciones de mapeos para fragmentos y cubos. En cada función se debe hacer corresponder una expresión de mapeo para cada ítem (del fragmento o cubo). El mapeo de un cubo se puede expresar recursivamente como drill-up de otro cubo ya mapeado. En el momento de definir las funciones se deben estudiar las restricciones del esquema conceptual y de las bases fuentes de manera de reflejarlas en los mapeos.

Capítulo IV - Generación del Esquema Lógico del DW

1. Introducción

En el capítulo anterior se presentó el proceso de diseño lógico de un DW, y se estudió la primera etapa del proceso.

Como resultado de la primer etapa se cuenta con:

- Un esquema intermedio, que es resultado de enriquecer el esquema conceptual con lineamientos.
- Una base fuente integrada.
- Mapeos entre el esquema intermedio y la base fuente integrada.

A partir del esquema intermedio, la base fuente y los mapeos, se construye el esquema lógico del DW mediante la aplicación de transformaciones sucesivas sobre esquemas relacionales, partiendo del esquema de la base de datos fuente. Las transformaciones aplicadas son las propuestas en [Mar00].

En este trabajo se propone un algoritmo y un conjunto de reglas para construir el esquema lógico en forma automática.

En esté capítulo se presenta la segunda etapa de proceso de diseño lógico. En la sección 2 se presenta una descripción informal de la propuesta mediante un ejemplo. En la sección 3 se presentan las transformaciones de esquema propuestas en [Mar00] con algunas extensiones. En la sección 4 se especifican las reglas de diseño y en la sección 5 se presenta el algoritmo. En la sección 6 se concluye.

2. Descripción Informal

En esta sección se presenta un ejemplo que ilustra la segunda etapa del proceso de diseño lógico de un DW. Se continúa el mismo con el ejemplo presentado en la sección 2 del capítulo anterior. En el mismo se ilustra la aplicación de reglas de diseño para construir un esquema relacional a partir del esquema intermedio.

El diseñador decide implementar un esquema estrella, denormalizando ambas dimensiones, y materializar dos cubos, uno con detalle por mes y cliente, y otro con totales por ciudad. Además al primer cubo decide fragmentarlo en dos franjas, una con los datos del año 2001 y otra con los datos anteriores. La Figura 22 muestra los lineamientos definidos.

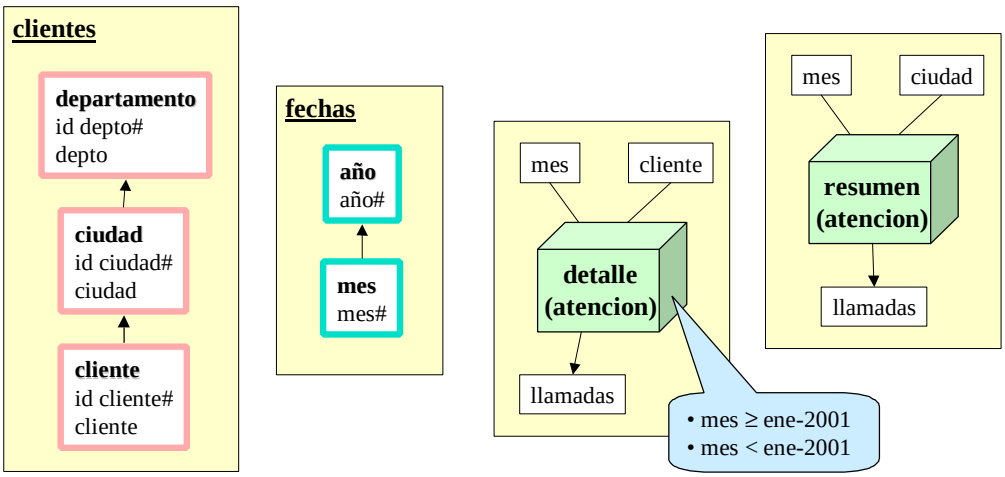

Figura 22 – Lineamientos de diseño

El diseñador establece los mapeos de los fragmentos como se muestra en la Figura 23.

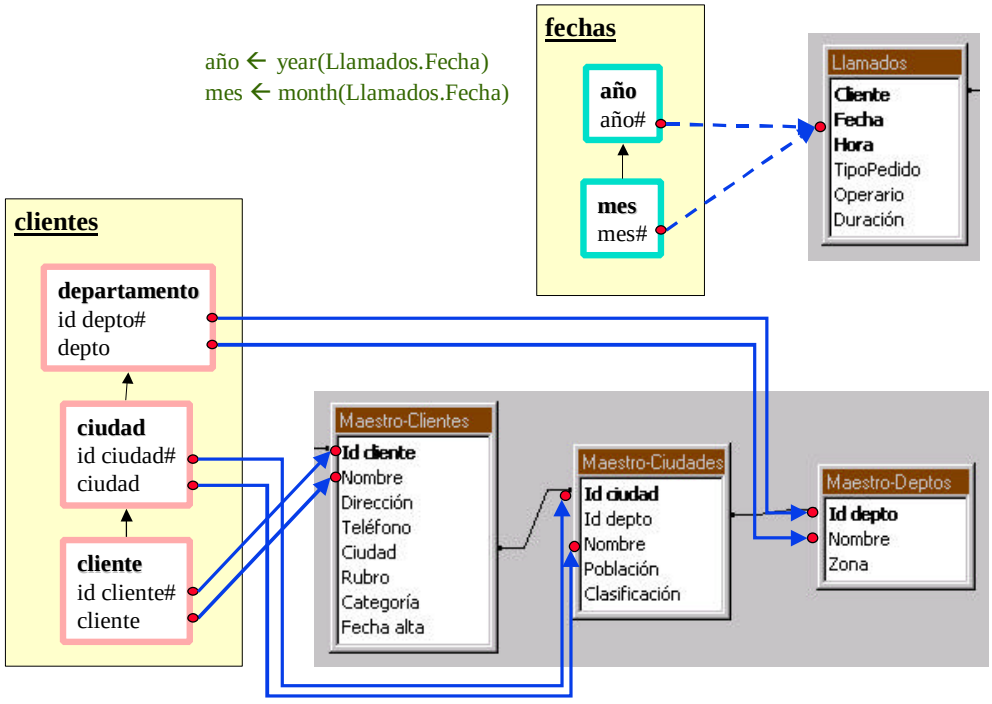

Figura 23 – Mapeo de los fragmentos

Para el cubo *detalle*, se establece un mapeo base (Figura 24), imponiendo como condición que las llamadas sean requerimientos (*TipoPedido* = *Req*). Se elige la función *sum* como roll-up para el ítem *tiempo* del nivel llamadas.

Para el cubo *resumen* se establece un mapeo recursivo, con un drill-up respecto a la dimensión *clientes* (Figura 25). También se elige *sum* como roll-up.

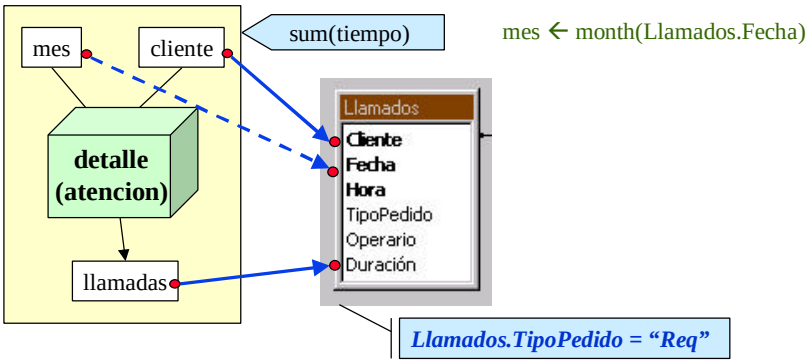

Figura 24 – Mapeo base del cubo detalle

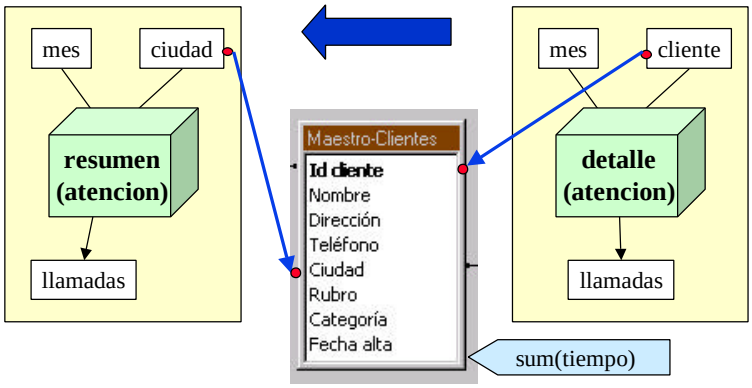

Figura 25 – Mapeo recursivo del cubo resumen

Si se quiere construir paso a paso el DW, el diseñador tomará una serie de decisiones, y en cada paso irá transformando el esquema relacional del DW. Para ello se basa en características que observa en los lineamientos y mapeos; es decir, cuando se cumple determinada característica o condición, realiza cierta transformación en el DW.

El objetivo de las reglas de diseño propuestas en este trabajo es automatizar esa secuencia de pasos, sustituyendo las decisiones del diseñador. Las reglas controlan que se verifiquen una serie de condiciones, y determinan que transformación debe aplicarse al esquema lógico.

La Figura 26 muestra una tabla con las reglas y un breve comentario de su utilidad. Las reglas marcadas con * representan familias de reglas orientadas a resolver un mismo problema.

	Regla	Condiciones de aplicación	Descripción
R1	Join	Un fragmento o cubo mapea a más de una tabla. Dos de esas tablas están relacionadas (tienen definido un link).	Combina las dos tablas generando una nueva tabla.
R2	Rename	Un fragmento o cubo mapea a una única tabla. Los ítems del objeto[17] que tienen mapeos directos (directME) y los atributos a los que mapean tienen nombres diferentes.	Renombra los atributos mapeados en forma directa, utilizando los nombres de los ítems.
R3	Calculate *	Un fragmento o cubo mapea a una única tabla. El objeto tiene un ítem con mapeo calculado (1calcME o NcalcME).	Agrega a la tabla un atributo que materializa el cálculo.
R4	Extern *	Un fragmento o cubo mapea a una única tabla. El objeto tiene un ítem con mapeo externo (externME).	Agrega a la tabla un atributo que materializa el valor externo.
R5	Group	Un fragmento o cubo mapea a una única tabla. La tabla tiene atributos que no están mapeados por los ítems del objeto.	Elimina de la tabla los atributos no mapeados, agrupando por los demás y realizando un resumen (roll-up) a las medidas.
R6	Drill Up *	Un cubo tiene un mapeo recursivo respecto a otro cubo. El segundo cubo mapea a una única tabla.	Reduce el nivel de detalle de la tabla, realizando un drill-up respecto a una dimensión.
R7	Primary Key	Un fragmento o cubo mapea a una única tabla. La clave del objeto no coincide con la clave de la tabla.	Modifica la clave primaria de la tabla para que coincida con la clave del objeto.
R8	Filter	Un fragmento o cubo mapea a una única tabla. La instancia del objeto debe verificar una condición. Esto puede deberse a que el objeto tenga condiciones en su mapeo, o que el objeto sea un cubo y tenga franjas definidas.	Elimina las tuplas de la tabla que no verifican la condición.

Figura 26 – Reglas de diseño

A continuación se muestra la aplicación de las reglas de diseño, continuando con el ejemplo.

Como se quiere denormalizar la dimensión *clientes*, se debe construir una única tabla que contenga todos los atributos que mapean a los ítems de la dimensión. La dimensión mapea a tres tablas: *Maestro-Clientes*, *Maestro-Ciudades* y *Maestro-Deptos.*

La regla R1 (Join) combina dos tablas que mapean a un mismo objeto. Se ejecuta la regla para combinar las tablas *Maestro-Clientes* y *Maestro-Ciudades*, obteniendo como resultado la tabla *DwClientes01* (Los atributos subrayados conforman la clave de la tabla):

```
DwClientes01 (Id-cliente, Nombre, Dirección, Teléfono, Ciudad, Rubro,
              Categoría, Fecha-alta, Id-ciudad, Id-depto, Nombre,
              Población, Clasificación)
```

Luego se ejecuta nuevamente la regla R1 para combinar las tablas *DwClientes01* y *Maestro-Deptos*, obteniendo como resultado la tabla:

```
DwClientes02 (Id-cliente, Nombre, Dirección, Teléfono, Ciudad, Rubro,
              Categoría, Fecha-alta, Id-ciudad, Id-depto, Nombre,
              Población, Clasificación, Id-depto2, Nombre2, Zona)
```

[17] Se refiere a un objeto del esquema intermedio, que puede ser un fragmento o un cubo.

Los nombres de los ítems *id-cliente*, *cliente*, *id-ciudad*, *ciudad*, *id-depto y depto* no coinciden con los nombres de los atributos a los que mapean. La regla R2 (Rename) renombra esos atributos. Se obtiene como resultado:

```
DwClientes03 (id-cliente, cliente, Dirección, Teléfono, Ciudad, Rubro,
              Categoría, Fecha alta, id-ciudad, Id depto, ciudad,
              Población, Clasificación, id-depto, depto, Zona)
```

El fragmento de la dimensión *clientes* mapea a los atributos *id-cliente*, *cliente*, *id-ciudad*, *ciudad*, *id-depto* y *depto* por lo que se deben eliminar los demás atributos. Para ello se aplica la regla R5 (Group), que elimina los atributos que no mapean y si es necesario agrupa por los demás para eliminar repetidos. Se obtiene como resultado:

```
DwClientes04 (id-cliente, cliente, id-ciudad, ciudad, id-depto, depto)
```

La clave de la tabla no coincide con la clave del fragmento (la clave del nivel más bajo en la jerarquía: *id_cliente*). Se ejecuta la regla R7 (PrimaryKey) para ajustar la clave y se obtiene como resultado:

```
DwClientes05 (id-cliente, cliente, id-ciudad, ciudad, id-depto, depto)
```

Se quiere denormalizar también la dimensión fechas, la cual mapea a una única tabla: *Llamados*, por lo tanto no se cumple la condición de la regla R1. En cambio, si se cumplen las hipótesis de la regla R3, ya que los ítems *mes* y *año* tienen mapeos calculados. Se ejecuta la regla R3 (Calculate) para incorporar el atributo *año* a la tabla. Se obtiene:

```
DwFechas01   (Cliente, Fecha, Hora, TipoPedido, Operario, Duración, año)
```

Luego se ejecuta nuevamente la regla R3 para agregar el atributo *mes*, obteniendo como resultado:

```
DwFechas02   (Cliente, Fecha, Hora, TipoPedido, Operario, Duración, año, mes)
```

Al igual que para la dimensión clientes se deben eliminar los atributos sobrantes agrupando por los que tienen mapeo. Se aplica la regla R5 y se obtiene:

```
DwFechas03   (año, mes)
```

Luego se ajusta la clave aplicando la regla R7. Como resultado se obtiene:

```
DwFechas04   (año, mes)
```

El cubo *detalle* mapea a una única tabla: *Llamados*. Es necesario renombrar algunos atributos y agregar el atributo calculado *mes*. Análogamente se ejecutan las reglas R2 y R3 y se obtienen respectivamente:

```
DwDetalle01  (id-cliente, Fecha, Hora, TipoPedido, Operario, llamadas)
DwDetalle02  (id-cliente, Fecha, Hora, TipoPedido, Operario, llamadas, mes)
```

La función de mapeo tiene una condición, por lo que deben eliminarse las tuplas que no la verifican. La regla R8 (Filter) elimina las tuplas que no cumplen la condición. Se obtiene como resultado una tabla con el mismo esquema pero con su instancia filtrada. Esto registra en una traza de diseño la información necesaria para implementar los programas de carga del DW.

```
DwDetalle03  (id-cliente, Fecha, Hora, TipoPedido, Operario, llamadas, mes)
```

El cubo mapea a los atributos *id-cliente*, *llamadas* y *mes*. Se deben eliminar los demás atributos. Para ello se aplica la regla R5 y se obtiene como resultado:

```
DwDetalle04  (id-cliente, llamadas, mes)
```

Luego se ajusta la clave ejecutando la regla R7. Se obtiene:

```
DwDetalle05  (id-cliente, llamadas, mes)
```

El cubo *resumen* está mapeado recursivamente respecto al cubo *detalle*, mediante un drill-up por la dimensión *clientes*. Se debe aplicar un drill-up a partir de la tabla *DwDetalle05* que mapea al cubo *detalle*. Se utiliza la tabla *Maestro-Clientes* para vincular los niveles *cliente* y *ciudad*. Se aplica la regla R6 (Drill-Up) que ejecuta el drill-up y aplica roll-up a la medida. Se obtiene como resultado:

```
DwResumen01  (id-ciudad, llamadas, mes)
```

Para fragmentar en franjas el cubo *detalle* se aplica la regla R8 (Filter) a la tabla *DwDetalle05* que mapea al cubo. La regla se aplica una vez para cada franja, obteniendo:

```
DwDetalleFranja01 (id-cliente, llamadas, mes)
DwDetalleFranja02 (id-cliente, llamadas, mes)
```

En las siguientes secciones se describen las reglas y se presenta un algoritmo que indica el orden de aplicación de las mismas.

3. Primitivas de Transformación de Esquemas

En esta sección se presenta el conjunto de primitivas de transformación de esquemas propuesto en [Mar00] y cómo contribuyen al diseño del esquema lógico del DW.

En este enfoque, el diseño de un DW es un proceso que comienza con el esquema fuente y termina con el esquema resultado, el cual corresponde al esquema del DW. Este se genera por la aplicación de sucesivas transformaciones al esquema fuente y a los sub-esquemas intermedios que son generados durante el proceso. Entonces, todos los elementos que constituyen el esquema final: tablas y atributos, son el resultado de aplicar las primitivas al esquema fuente.

Las primitivas son transformaciones básicas que toman como entrada un sub-esquema y su salida es otro sub-esquema.

Los tipos de transformaciones cubiertas por las primitivas son: particiones de tablas, combinación (merge) de tablas, agregado de atributos, borrado de atributos y cambios en claves primarias y foráneas.

La Figura 27 muestra una tabla con las primitivas de transformación de esquemas. Las primitivas marcadas con * representan familias de primitivas orientadas a resolver un mismo problema.

	Transformación	Descripción
P1	Identity	Dada una tabla, genera otra tabla igual a la tabla origen.
P2	Data Filter	Dada una tabla origen, genera otra donde sólo algunos de los atributos se preservan. Su objetivo es eliminar atributos puramente operacionales.
P3	Temporalization	Agrega un elemento de tiempo al conjunto de atributos de una tabla.
P4	Key Generalisation *	Generaliza la clave primaria de una tabla de dimensión.
P5	Foreign Key Update	A través de esta primitiva, una clave foránea y su referencia pueden ser cambiadas en una tabla. Es muy útil cuando varía la clave primaria de una tabla.
P6	DD-Adding *	Agrega a una tabla un atributo que se deriva de otros.
P7	Attribute Adding	Agrega un atributo a una tabla de dimensión. Es muy útil para realizar un versionamiento de tuplas en una tabla.
P8	Hierarchy Roll Up	Realiza un roll-up a través de uno de los atributos de una tabla, siguiendo una jerarquía. A su vez, puede generar otra tabla de jerarquía con el correspondiente nivel de detalle.
P9	Aggregate Generation	Dada una tabla de medida, genera otra tabla de medida, donde los datos están resumidos o agrupados por un conjunto de atributos.
P10	Data Array Creation	Dada una tabla que contiene un atributo de medida, y un atributo que representa un conjunto predeterminado de valores, genera una tabla con estructura de matriz.
P11	Partition by Stability *	Particiona una tabla de manera de organizar sus datos en forma histórica. Particiones horizontales y verticales pueden aplicarse según el criterio del diseñador
P12	Hierarchy Generation *	Genera tablas de jerarquía, tomando como entrada tablas que contienen la totalidad o parte de una jerarquía.
P13	Minidimension Break off	Elimina un conjunto de atributos de una tabla de dimensión, construyendo una nueva tabla con ellos.
P14	New Dimension Crossing	Permite materializar el cruzamiento de un conjunto de tablas de dimensión, en una nueva tabla de dimensión. Es útil en el proceso de denormalización de tablas, lo que aumenta la performance de las consultas.

Figura 27 – Primitivas de transformación de esquemas

Las primitivas fueron pensadas para una aplicación interactiva, en la que el diseñador elige las transformaciones adecuadas para su problema. Para resolver casos generales se necesitan transformaciones sencillas que complementen el conjunto. Además, las primitivas fueron pensadas para transformaciones de esquemas únicamente, hace falta alguna transformación para manejar instancias.

En este trabajo se definen nuevas primitivas que complementan a las anteriores. La Figura 28 muestra una tabla con las primitivas incorporadas. En el Anexo 5 se presenta la descripción y especificación de las mismas siguiendo el modelo de Marotta. Se presenta también un ejemplo de la necesidad de nuevas primitivas.

	Transformación	Descripción
Q1	Relation Join	Realiza el join de dos tablas generando una nueva.
Q2	Attribute Renaming	Dada una tabla, genera otra renombrando algunos atributos.
Q3	Instance Filter	Dada una tabla, genera otra que es estructuralmente igual pero eliminando las tuplas que no satisfacen una condición.
Q4	Primary Key Modification	Dada una tabla, modifica su clave primaria.

Figura 28 – Nuevas primitivas de transformación de esquemas

El conjunto de primitivas no pretende ser completo en el sentido de permitir construir cualquier esquema relacional, pero intenta permitir la construcción del DW. Las primitivas abstraen y generalizan técnicas de diseño de DW y permiten generar esquemas de DW según diferentes sub-modelos o combinaciones entre ellos.

La intención es aplicar las primitivas a un esquema fuente para hacerlo más adecuado a las consultas. A continuación se presentan algunos ejemplos tomados de [Mar99].

> *Muchas tablas en los sistemas operacionales no mantienen una noción temporal. Por ejemplo, las tablas de stock suelen tener los datos del stock actual, actualizándolo con cada movimiento de los productos. Sin embargo, en el DW la mayoría de las tablas necesitan incluir un elemento temporal, de manera que puedan mantener información histórica. Para este propósito existe una primitiva llamada* ***Temporalization*** *que agrega un elemento de tiempo al conjunto de atributos de una tabla.*
>
> *En los sistemas de producción, usualmente, los datos se calculan a partir de otros datos en el momento de las consultas, a pesar de la complejidad de algunas funciones de cálculo, para prevenir cualquier clase de redundancia. Por ejemplo, los precios de productos expresados en dólares son calculados a partir de los precios expresados en otra moneda y una tabla conteniendo el valor del dólar. En un sistema de DW, a veces es conveniente mantener esta clase de datos calculados, por razones de performance. Existe un grupo de primitivas, cuyo nombre es* ***DD-Adding****, que agrega a una tabla un atributo derivado a partir de otros.*

Las primitivas son transformaciones de alto nivel y encapsulan conocimiento de diseño de DWs pero dando flexibilidad en la aplicación de diferentes estrategias (por ej. estrella, snowflake). Las mismas también incluyen la especificación de la semántica de las tablas resultantes a través de operaciones de transformación en un lenguaje tipo SQL.

En este enfoque, el DW es el resultado de aplicar sucesivas transformaciones a las bases fuentes. Cada relación del DW se obtiene como composición de la aplicación de varias primitivas.

La aplicación subsiguiente de las primitivas genera una *traza* de las transformaciones realizadas. La traza provee información de las primitivas aplicadas, sus argumentos y sus resultados.

Gráficamente, la traza se representa mediante un grafo dirigido y acíclico (DAG).

Los nodos pueden ser:

- Tablas (nodos rectangulares), tanto del esquema fuente (*source* relations) como del DW (*DW* relations)
- Aplicaciones de primitivas (nodos circulares).

Los arcos pueden unir:

- Una tabla del esquema fuente y una primitiva, lo cual representa que la tabla es parámetro de la primitiva.
- Dos primitivas, representando que parte del resultado de la aplicación de una primitiva se utiliza como parámetro para la otra primitiva.
- Una primitiva y una tabla del DW, que representa que la tabla es parte del resultado de la primitiva.

En la Figura 29 se muestra una traza de aplicación.

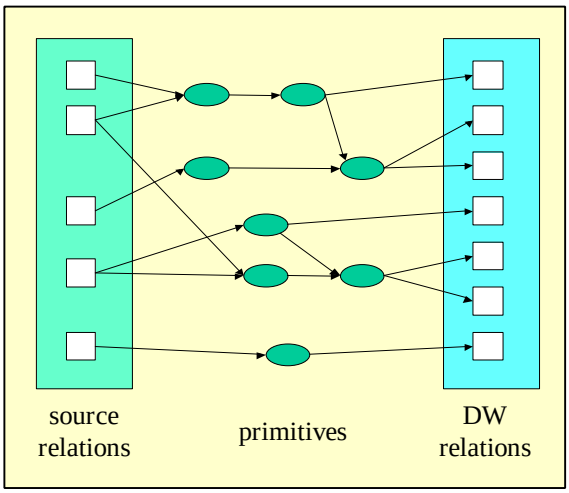

Figura 29 – Traza de aplicación de las primitivas

La traza se utiliza para facilitar la generación de código de carga y actualización del DW así como para la gestión de la evolución del DW frente a cambios en el esquema fuente.

Las primitivas no inducen una estrategia o metodología en particular. Es más, su aplicación sin criterios de diseño bien definidos puede llevar a resultados no deseados, a veces, con problemas de consistencia. Para ayudar al diseñador en este aspecto se provee: (i) invariantes del esquema del DW, (ii) estrategias para resolver los problemas típicos que aparecen en el diseño de un DW y (iii) reglas de consistencia para la aplicación de las primitivas.

En la siguiente sección se presenta un conjunto de reglas de diseño que determinan la secuencia de primitivas a aplicar. Algunas reglas se basan en las estrategias propuestas por Marotta en [Mar00].

4. Reglas de Diseño

En esta sección se presenta un mecanismo para construir el esquema lógico del DW partiendo del esquema intermedio, la base fuente y las correspondencias entre ellos. El objetivo es automatizar el proceso de diseño teniendo en cuenta las estrategias de diseño especificadas mediante los lineamientos.

Se propone un algoritmo para diseñar el esquema lógico del DW basado en la aplicación de reglas de diseño. Dichas reglas se aplican a objetos del esquema intermedio (fragmentos de dimensiones, cubos, etc.) que cumplen determinadas condiciones y dan como resultado la aplicación de primitivas al esquema lógico del DW.

El objetivo de las reglas es determinar cuándo puede aplicarse una determinada transformación, según si los objetos del esquema intermedio cumplen una serie de premisas o condiciones. Las reglas utilizan y modifican las tablas y las funciones de mapeo asociadas.

Debe observarse que las primitivas de transformación de esquema propuestas en [Mar00] deben ser aplicadas con criterios de diseño bien definidos para obtener un DW adecuado a los requerimientos. Las reglas de diseño y el algoritmo asociado asisten al diseñador en el proceso de diseño lógico, determinando las transformaciones a aplicar de acuerdo a los criterios de diseño especificados en los lineamientos.

En la sección 4.1 se describen las reglas y se especifican en la sección 4.2. En la sección 5 se presenta el algoritmo.

4.1. Descripción de las Reglas

A continuación se presenta una breve descripción de cada regla con la intención de mostrar la utilidad y el comportamiento de las mismas.

Rule 1. JOIN

Un objeto del esquema intermedio (un fragmento o un cubo) puede mapear a una o varias tablas fuentes. En el último caso se debe construir una nueva tabla tomando atributos de varias tablas fuentes, combinándolas adecuadamente teniendo en cuenta los links entre ellas.

La regla R1 se aplica cuando un objeto mapea a más de una tabla.

Su objetivo es combinar dos de las tablas que mapean al objeto (relacionadas entre si por un link), generando una nueva tabla. Se aplica la primitiva Q1 – Relation Join.

Rule 2. RENAME

En las bases fuentes puede haberse utilizado cualquier tipo de nomenclatura para los atributos, no necesariamente mnemotécnicos. Es razonable renombrar los atributos de las tablas fuentes para asignarles nombres adecuados. Se utilizan los nombres de los ítems del esquema intermedio.

La regla R2 se aplica cuando un objeto mapea a una única tabla, y los nombres de algunos de sus ítems, con mapeo directo (directME), difieren de los nombres de los atributos a los que mapean.

Su objetivo es renombrar los atributos de la tabla utilizando los nombres de los ítems. Se aplica la primitiva Q2 – Attribute Renaming.

Rule 3. CALCULATE

Algunos ítems no mapean directamente con atributos de la base fuente, sino que mapean con expresiones o cálculos sobre ellos.

Para los que el mapeo es simple (1calcME) se debe calcular el nuevo atributo. Para los que el mapeo involucra varias tuplas (NcalcME) se debe calcular el resumen.

La regla R3 se aplica cuando un objeto mapea a una única tabla, y tiene un ítem con mapeo calculado.

Su objetivo es agregar a la tabla un atributo que materialice el cálculo. Se aplica una primitiva de la familia P6 – DD-Adding.

Rule 4. EXTERN

Algunos ítems tienen mapeos externos (externME), ya sean constantes, marcas de tiempo o dígitos de versión. Se deben agregar atributos que correspondan a esos ítems.

La regla R4 se aplica cuando un objeto mapea a una única tabla, y tiene un ítem con mapeo externo.

Su objetivo es agregar a la tabla un atributo que materialice el valor externo. Se aplican las primitivas P7 – Attribute Adding, P3 – Temporalization o P4.1 – Version Digits.

Rule 5. GROUP

Muchas veces la tabla que mapea a un objeto tiene atributos que no son mapeados por ningún ítem. En dichos casos deben eliminarse los atributos no mapeados y agruparse por los restantes, de manera de reducir el tamaño de la tabla y eliminar tuplas repetidas. Para el caso de un cubo, se aplican operaciones de resumen a las medidas (roll-up).

La regla R5 se aplica cuando un objeto mapea a una única tabla, y dicha tabla tiene atributos que no mapean a ningún ítem del objeto.

Su objetivo es eliminar de la tabla los atributos no mapeados, agrupando por los restantes atributos. Se aplica la primitiva P9 – Aggregate Generation.

Rule 6. DRILL-UP

El mapeo de un cubo puede ser recursivo, es decir, puede estar expresado en términos de otro cubo, que tiene las mismas medidas pero más detalle para algunas dimensiones. En estos casos se debe realizar un drill-up, es decir, generar una nueva tabla que resuma a la tabla que mapea al cubo base y aplique operaciones de resumen a las medidas.

Hay dos casos: cuando interesa menos detalle para una dimensión, o cuando no interesa ningún detalle para esa dimensión. En el primer caso deben calcularse totales parciales para las medidas agrupando por el nuevo nivel de detalle de la dimensión y los demás atributos. En el segundo caso no se tienen atributos para la dimensión y se debe agrupar sólo por atributos de otras dimensiones.

La regla R6 se aplica cuando un cubo tiene un mapeo recursivo respecto a otro cubo (base), y éste último mapea a una única tabla.

Su objetivo es reducir el nivel de detalle de la tabla que mapea al cubo base, aplicando un drill-up respecto a una dimensión, agrupando por los restantes atributos y aplicando una función de roll-up a los atributos que mapean a las medidas. Se aplican las primitivas P8 – Hierarchy Roll Up o P9 – Aggregate Generation.

Rule 7. PRIMARY KEY

Las claves de las tablas que mapean a los objetos pueden no coincidir con las claves de los propios objetos. En esos casos se necesita actualizar dichas claves.

La regla R7 se aplica cuando un objeto mapea a una única tabla, y la clave de dicha tabla no coincide con la clave del objeto.

Su objetivo es modificar la clave de la tabla para que coincida con la clave del objeto. Se aplica la primitiva Q4 – Primary Key Update.

Rule 8. FILTER

En los mapeos se pueden especificar condiciones que deben cumplir las instancias, en base a restricciones del esquema conceptual o de las bases fuentes. El lineamiento de fragmentación de cubos también especifica condiciones que deben cumplir las franjas.

La regla R8 se aplica cuando un objeto mapea a una única tabla, y su función de mapeo tiene una condición. Se aplica también cuando un cubo tiene definidas franjas, cada una con un predicado o condición.

Su objetivo es verificar que las tuplas de la tabla que mapea al objeto cumplan con una condición y descartar las tuplas que no la cumplen. Se aplica la primitiva Q3 – Instance Filter.

4.2. Especificación de las Reglas

Las siguientes especificaciones constan de 5 secciones: *Description, Objects, Input, Conditions* y *Result*. Description es una explicación en lenguaje natural del comportamiento de la regla. Objects enumera los objetos del esquema intermedio a los que se puede aplicar la regla. Input enumera los parámetros de entrada a la regla, estos son: funciones de mapeo de los objetos y tablas del esquema lógico. Conditions es un conjunto de predicados que los objetos y las entradas deben cumplir para poder aplicar la regla. Finalmente, Result es la salida de la regla que consta de tablas resultado de aplicar transformaciones a las tablas de entrada, y funciones de mapeo actualizadas para reflejar la aplicación de la regla.

La Figura 30 muestra un diagrama con el formato de especificación.

RULE *rulenumber* : *RULE NAME*

Description:
Describe en lenguaje natural el comportamiento de la regla.

Objects:
Elementos del esquema intermedio a los que se aplica la regla.

Input:
Es la entrada a la regla. Está compuesta por:

- **Maps:** Funciones de mapeo definidas para los objetos.
- **Tables:** Tablas que mapean a los objetos.

Conditions:
Son condiciones que deben cumplir los objetos y las tablas y funciones de mapeo que conforman la entrada.

- **Let:** Definiciones de variables para ser usadas en el alcance de la regla.

Result:
- **Tables:** Nuevas tablas, resultado de aplicar una primitiva.
- **Maps:** Nuevas funciones de mapeo, para referenciar a las nuevas tablas.

Figura 30 – Formato de especificación de las reglas

La ejecución de una regla tiene como resultado la aplicación de primitivas. Esto implica que se generan una o más tablas que se agregan al esquema lógico. Estas tablas pueden ser tablas a las que se aplicarán más transformaciones, o pueden ser tablas definitivas para el DW.

También deben definirse links entre las nuevas tablas y las tablas existentes. Las tablas generadas son transformaciones de las tablas de entrada a la regla, por lo que es común que se relacionen con éstas por las claves y que hereden los links a otras tablas.

Por ejemplo: se tiene una tabla *clientes* con clave primaria *código-cliente*, y se aplica la primitiva P6 – DD-Adding para agregar el atributo calculado *edad*, generando la tabla *clientes2*. Las tablas *clientes* y *clientes2* tienen la misma clave y tienen un link por ella. Además la tabla *clientes2* tendrá links a las mismas tablas que *clientes* y con los mismos predicados.

En el Anexo 4 se define el procedimiento *UpdateTabs* que mantiene actualizado el esquema lógico, agregando la nueva tabla y generando links para ella.

Durante de la aplicación de la regla se define una nueva función de mapeo para el objeto. Se deben actualizar los mapeos a fragmentos o cubos (el que corresponda) para incorporar la nueva función.

En el Anexo 4 se define el procedimiento *UpdateLinks* que mantiene actualizados los mapeos de cubos y fragmentos.

Ambos procedimientos (UpdateTabs y UpdateLinks) son utilizados en la especificación de las reglas.

Rule R1. Join

Objetivo:

El objetivo de esta regla es combinar dos tablas que mapean a un objeto, generando una nueva tabla. Se aplica la primitiva Q1 – Relation Join.

Esta regla se utiliza para construir los esqueletos[18], tanto de fragmentos de dimensiones como de cubos.

Ejemplo:

La Figura 31 muestra la dimensión *clientes* y su mapeo a las tablas *Departamentos*, *Ciudades* y *Clientes*.

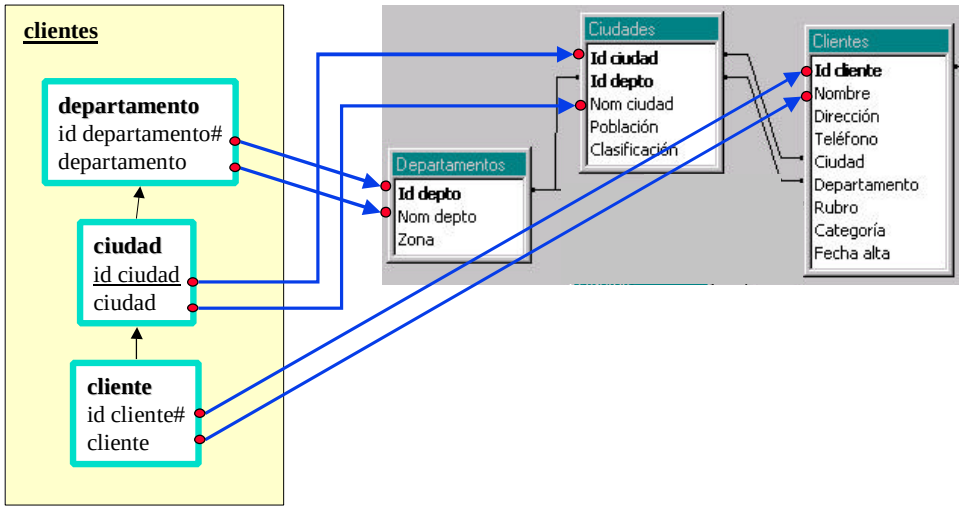

Figura 31 – Mapeo de la dimensión clientes antes de aplicar Join

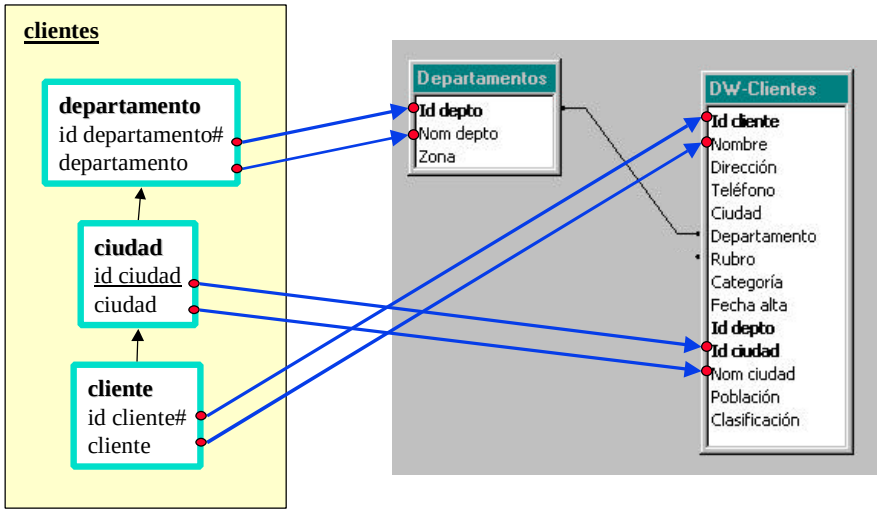

Figura 32 – Mapeo de la dimensión clientes después de aplicar Join

La estrategia elegida es denormalizar, por lo que se definió un único fragmento de la dimensión con todos los niveles.

[18] Se llama esqueleto a una única tabla que mapea a un objeto, y tiene todos los atributos mapeados por los items del objeto. Esta tabla es producto de combinar varias tablas mediante la operación de join. Luego se aplicarán más transformaciones al esqueleto.

Se debe combinar (join) todas las tablas a las que se mapea. La regla se aplica a dos tablas, por lo que en este ejemplo debe aplicarse la regla dos veces. Primero se aplica a las tablas *Clientes* y *Ciudades*, esto es, se construye una nueva tabla, de nombre *DW-Clientes*, con los atributos de las dos anteriores.

Esto actualiza la función de mapeo para que referencie a los atributos de la nueva tabla cuando corresponda. La Figura 32 muestra la función de mapeo actualizada.

De la misma forma se aplica la regla a las tablas *Departamentos* y *DW-Clientes*, obteniendo que la función de mapeo referencie a una sola tabla (esqueleto).

Especificación:

RULE R1 - JOIN

Description:

Dado un objeto del esquema intermedio, su función de mapeo y dos tablas relacionadas[19] que mapean al objeto, se genera una nueva tabla con los atributos de ambas siguiendo el link entre ellas.

Objects:

- OS ∈ SchFragments ∪ SchCubes */* un fragmento o un cubo */*

Input:

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

Conditions:

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

Result:

- **Tables:**
  - 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 **
- **Maps:**
  - F' = ReplaceTable (F,T1,T')
  - F'' = ReplaceTable (F',T2, T')
  - UpdateMaps (F'', OS)

[19] En este trabajo, se dice que dos tablas están relacionadas si tienen definido un link entre ellas.

Rule R2. Rename

Objetivo:

El objetivo de esta regla es renombrar atributos que mapean en forma directa (directME) a ítems, tanto de fragmentos como de cubos, cuyos nombres difieren de los nombres de dichos ítems. Se aplica la primitiva Q2 – Attribute Renaming.

Consideraciones:

- Esta regla se debe aplicar después de la regla R1 (Join), que construye los esqueletos de los objetos. Esto garantiza que los atributos con mapeo directo estén en el esqueleto.
- Se presenta un conflicto cuando un atributo de una tabla mapea con más de un ítem de la misma dimensión. Esto es un problema, ya que sólo puede darse un nombre al atributo.

  Este tipo de conflictos debe solucionarse definiendo uno de los mapeos como DirectME, y el otro cómo 1calcME (copia del atributo).

Ejemplo:

La Figura 33a muestra la dimensión *Clientes* y el mapeo de un fragmento a la tabla *Departamentos*.

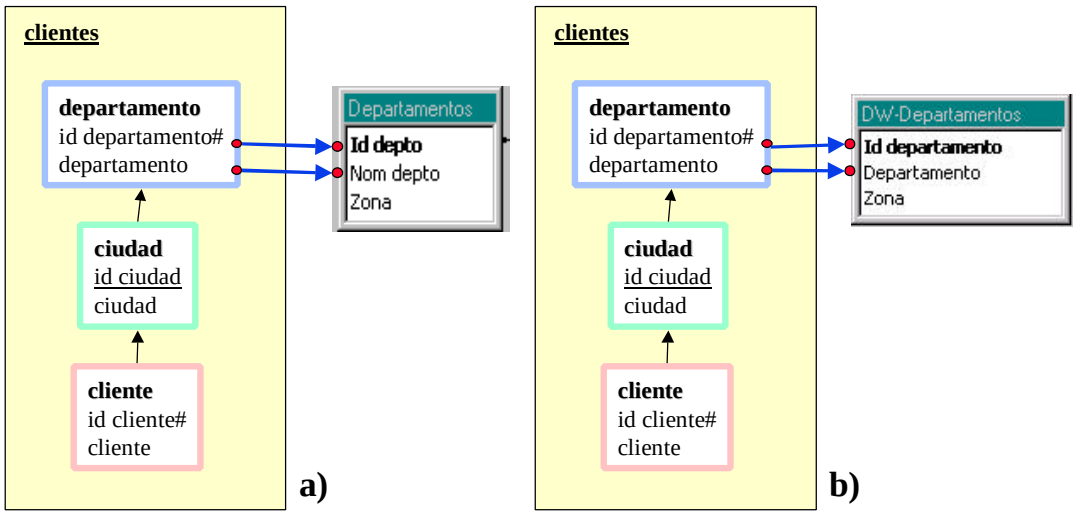

Figura 33 – Mapeo de la dimensión clientes: a) antes y, b) después de aplicar Rename

Se renombran los atributos de la tabla *Departamentos*, esto es, se construye una nueva tabla de nombre *DW-Departamentos*, renombrando los atributos *Id-depto* y *Nom-depto* por *Id-departamento* y *Departamento* respectivamente.

Esto actualiza la función de mapeo, para que referencie a los atributos de la nueva tabla cuando corresponda. La Figura 33b muestra la función de mapeo actualizada.

Especificación:

RULE R2 - RENAME

Description:

Dado un objeto del esquema intermedio, su función de mapeo y una tabla que lo mapea, se genera una nueva tabla renombrando sus atributos según los nombres de los ítems.

Objects:

- OS ∈ SchFragments ∪ SchCubes */* un fragmento o un cubo */*

Input:

- **Maps:** F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
- **Tables:** T ∈ MapTables(F, ObjectItems(OS)) */* tabla que mapea al objeto */*

Conditions:

Let:

- 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 sóla tabla */*

Result:

- **Tables:**
  - 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 */*
- **Maps:**
  - F' = ReplaceAttributes (F, List)
  - F'' = ReplaceTable (F',T, T')
  - UpdateMaps (F'', OS)

Rule R3. Calculate

Objetivo:

El objetivo de esta regla es agregar a la tabla que mapea a un objeto, atributos que materialicen los cálculos. Se aplica una primitiva de la familia P6 – DD-Adding.

Consideraciones:

- Esta regla se debe aplicar después de la regla R1 (Join), que construye los esqueletos de los objetos. Esto garantiza que los atributos con mapeo directo o cálculo simple estén en el esqueleto.
- Para los cálculos simples (1calcME), como ya se construyeron los esqueletos, los atributos de los que depende el cálculo se encuentran en una única tabla.
- Para los cálculos de resumen (NcalcME), los atributos de los que depende el cálculo se encuentran en otra tabla, diferente del esqueleto, pero que tiene links con ésta.

Variantes:

Se tienen dos variantes dependiendo del tipo de mapeo:

- *Simple-Calculate*: Resuelve cálculos simples, con mapeo 1calcME.
- *Aggregate-Calculate*: Resuelve cálculos de resumen, con mapeo NcalcME.

Ejemplo:

La Figura 34a muestra la dimensión *Fechas* y su mapeo a la tabla *Facturas*. Los mapeos de los ítems *mes* y *año* son calculados.

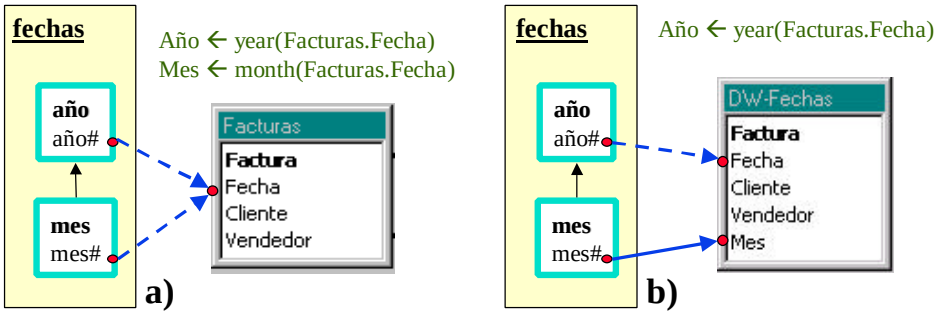

Figura 34 – Mapeo de la dimensión fechas: a) antes y, b) después de aplicar Calculate

Se calcula un nuevo atributo, de nombre *Mes*, como *month(Facturas.Fecha)*. Esto es, se construye una nueva tabla, de nombre *DW-Fechas*, agregando dicho atributo.

Esto actualiza la función de mapeo para que referencie a los atributos de la nueva tabla cuando corresponda. La Figura 34b muestra la función de mapeo actualizada.

De la misma forma se calcula un atributo para el ítem *año*.

Especificación:

RULE R3 - CALCULATE

SUBRULE R3.1 – SIMPLE CALCULATE

Description:

Dado un objeto del esquema intermedio, uno de sus ítems, su función de mapeo (con mapeo 1calcME para el ítem) y una tabla que lo mapea, se genera una nueva tabla agregando el atributo calculado.

Objects:

- OS ∈ SchFragments ∪ SchCubes */* un fragmento o un cubo */*
- I ∈ ObjectItems(OS) */* un ítem */*

Input:

- **Maps:** F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
- **Tables:** T ∈ MapTables(F, ObjectItems(OS)) */* tabla que mapea al objeto */*

Conditions:

- F(I) ∈ 1calcME */* cálculo simple */*
- # MapTables(F, ObjectItems(OS)) = 1 */* el objeto es mapeado por una sóla tabla */*

Result:

- **Tables:**
  - 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 */*
- **Maps:**
  - F' = ReplaceMap (F, F(I), <I, T', T'•I, F(I).Cond)}) */* ahora con mapeo DirectME*/*
  - F'' = ReplaceTable (F',T, T')
  - UpdateMaps (F'', OS)

RULE R3 – CALCULATE

SUBRULE R3.2 – AGGREGATE CALCULATE

Description:

Dado un objeto del esquema intermedio, uno de sus ítems, su función de mapeo (con mapeo NcalcME para el ítem) y una tabla que lo mapea, se genera una nueva tabla agregando el atributo calculado.

Objects:

- OS ∈ SchFragments ∪ SchCubes */* un fragmento o un cubo */*
- I ∈ ObjectItems(OS) */* un ítem */*

Input:

- **Maps:** F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
- **Tables:** T ∈ MapTables(F, ObjectItems(OS)) */* tabla que mapea al objeto */*

Conditions:

- F(I) ∈ NcalcME */* cálculo de resumen */*
- # MapTables(F, ObjectItems(OS)) = 1 */* el objeto es mapeado por una sóla tabla */*

Result:

- **Tables:**
  - 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 */*
- **Maps:**
  - F' = ReplaceMap (F, F(I), <I, T', T'•I, F(I).Cond)}) */* ahora con mapeo DirectME*/*
  - F'' = ReplaceTable (F',T, T')
- UpdateMaps (F'', OS)

Rule R4. Extern

Objetivo:

El objetivo de esta regla es agregar atributos que materialicen valores externos, correspondientes a mapeos externos (externME). Se aplican las primitivas P7 – Attribute Adding, P3 – Temporalization o P4.1 – Version Digits.

Consideraciones:

- Esta regla se debe aplicar después de la regla R1 (Join), que construye los esqueletos de los objetos. Esto garantiza que los atributos mapeados estén en el esqueleto.

Variantes:

Se tienen tres variantes dependiendo del tipo de mapeo:

- *Constant-Extern-Value*: Resuelve mapeos ExternME de tipo constante.
- *Timestamp-Extern-Value*: Resuelve mapeos ExternME de tipo marca de tiempo.
- *Version-Extern-Value*: Resuelve mapeos ExternME de tipo dígitos de versión.

Ejemplo:

La Figura 35a muestra la dimensión *Productos* y su mapeo a la tabla *MaestroProductos*. El mapeo del ítem *version* es externo, de tipo dígitos de versión.

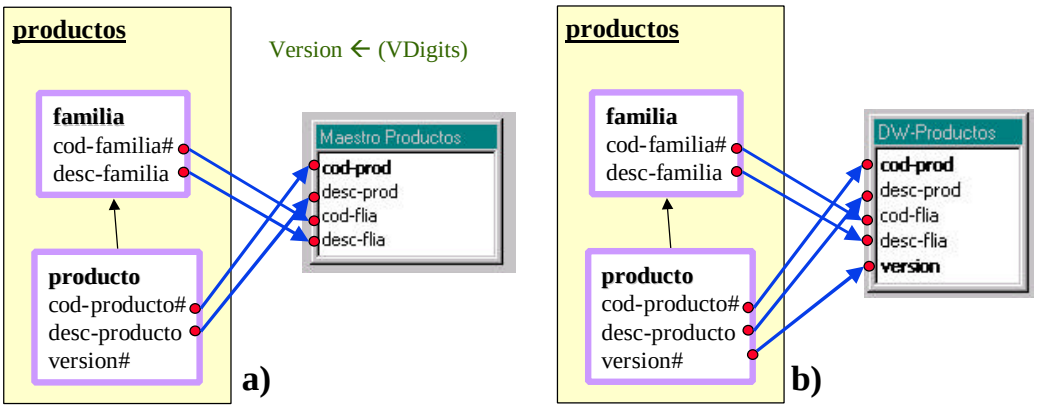

Figura 35 – Mapeo de la dimensión productos: a) antes y, b) después de aplicar New Attribute

Se calcula un nuevo atributo, de nombre *version* que se calcula como dígitos de versión. Esto es, se construye una nueva tabla, de nombre *DW-Productos*, agregando dicho atributo.

Esto actualiza la función de mapeo, para que referencie a los atributos de la nueva tabla cuando corresponda. La Figura 35b muestra la función de mapeo actualizada.

Especificación:

RULE R4 - EXTERN

SUBRULE R4.1 – CONSTANT EXTERN VALUE

Description:

Dado un objeto del esquema intermedio, uno de sus ítems, su función de mapeo (con mapeo ExternME de tipo constante para el ítem) y una tabla que lo mapea, se genera una nueva tabla agregando el atributo correspondiente al ítem.

Objects:

- OS ∈ SchFragments ∪ SchCubes */* un fragmento o un cubo */*
- I ∈ ObjectItems(OS) */* un ítem */*

Input:

- **Maps:** F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
- **Tables:** T ∈ MapTables(F, ObjectItems(OS)) */* tabla que mapea al objeto */*

Conditions:

- F(I) ∈ ExternME ∧ F(I).Ind = Constant */* mapeo constante */*
- # MapTables(F, ObjectItems(OS)) = 1 */* el objeto es mapeado por una sóla tabla */*

Result:

- **Tables:**
  - T' = PrimitiveP7 (*/* Attribute Adding */*
    {T}, */* esquema fuente */*
    I.ItemName, */* atributo a agregar */*
    {Instance(T)} */* instancias */*
)
  - UpdateTabs (T', {T}) */* con links a T por su clave, y hereda los links de T */*
- **Maps:**
  - F' = ReplaceMap (F, F(I), <I, T', T'•I, F(I).Cond)}) */* ahora con mapeo DirectME*/*
  - F'' = ReplaceTable (F',T, T')
  - UpdateMaps (F'', OS)

RULE R4 – EXTERN

SUBRULE R4.2 – TIMESTAMP EXTERN VALUE

Description:

Dado un objeto del esquema intermedio, uno de sus ítems, su función de mapeo (con mapeo ExternME de tipo marca de tiempo para el ítem) y una tabla que lo mapea, se genera una nueva tabla agregando el atributo correspondiente al ítem.

Objects:

- OS ∈ SchFragments ∪ SchCubes */* un fragmento o un cubo */*
- I ∈ ObjectItems(OS) */* un ítem */*

Input:

- **Maps:** F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
- **Tables:** T ∈ MapTables(F, ObjectItems(OS)) */* tabla que mapea al objeto */*

Conditions:

- F(I) ∈ ExternME ∧ F(I).Ind = Timestamp */* mapeo por estampa de tiempo */*
- # MapTables(F, ObjectItems(OS)) = 1 */* el objeto es mapeado por una sóla tabla */*

Result:

- **Tables:**
  - 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 */*
- **Maps:**
  - F' = ReplaceMap (F, F(I), <I, T', T'•I, F(I).Cond)}) */* ahora con mapeo DirectME*/*
  - F'' = ReplaceTable (F',T, T')
  - UpdateMaps (F'', OS)

RULE R4 – EXTERN

SUBRULE R4.3 – VERSION EXTERN VALUE

Description:

Dado un objeto del esquema intermedio, uno de sus ítems, su función de mapeo (con mapeo ExternME de tipo dígito de versión para el ítem) y una tabla que lo mapea, se genera una nueva tabla agregando el atributo correspondiente al ítem.

Objects:

- OS ∈ SchFragments ∪ SchCubes */* un fragmento o un cubo */*
- I ∈ ObjectItems(OS) */* un ítem */*

Input:

- **Maps:** F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
- **Tables:** T ∈ MapTables(F, ObjectItems(OS)) */* tabla que mapea al objeto */*

Conditions:

- F(I) ∈ ExternME ∧ F(I).Ind = Version */* mapeo por dígitos de versión */*
- # MapTables(F, ObjectItems(OS)) = 1 */* el objeto es mapeado por una sóla tabla */*

Result:

- **Tables:**
  - T' = PrimitiveP4.1 (*/* Version Digits */*
    {T}, */* esquema fuente */*
    I.ItemName, */* atributo de versión */*
    {Instance(T)} */* instancias */*
)
  - UpdateTabs (T', {T}) */* con links a T por su clave, y hereda los links de T */*
- **Maps:**
  - F' = ReplaceMap (F, F(I), <I, T', T'•I, F(I).Cond)}) */* ahora con mapeo DirectME*/*
  - F'' = ReplaceTable (F',T, T')
  - UpdateMaps (F'', OS)

Rule R5. Group

Objetivo:

El objetivo de esta regla es eliminar los atributos de una tabla que no mapean a ningún ítem, agrupando por los restantes atributos. Se aplica la primitiva P9 – Aggregate Generation.

La regla se utiliza para eliminar atributos que no están mapeados por los ítems de un objeto, tanto fragmentos de dimensiones como cubos. En estos últimos debe aplicarse roll-up a las medidas.

Consideraciones:

- Esta regla se debe aplicar después de la regla R1 (Join), que construye los esqueletos de los objetos. Esto garantiza que los atributos mapeados estén en el esqueleto.

Variantes:

Se tienen dos variantes dependiendo del tipo de objetos:

- *Fragment-Group*: Elimina atributos de tablas que mapean a fragmentos.
- *Cube-Group*: Elimina atributos de tablas que mapean a cubos.

Ejemplo:

La Figura 36a muestra la dimensión *Fechas* y su mapeo a la tabla *DW-Fechas-2*, de la que sólo se tienen mapeos a los atributos *Mes* y *Año*.

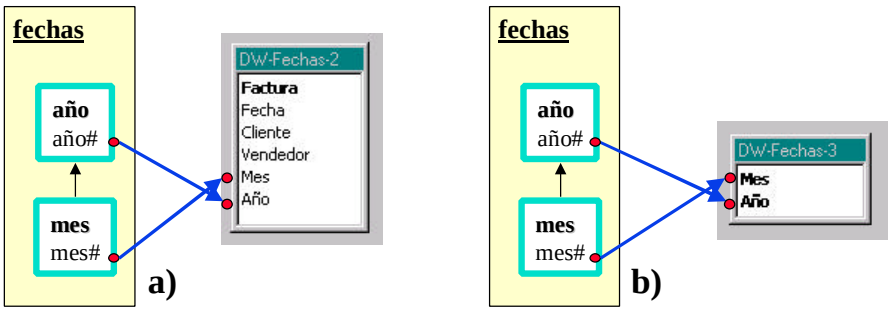

Figura 36 – Mapeo de la dimensión fechas: a) antes y, b) después de aplicar Group

La granularidad de la tabla *DW-Fechas-2* es una factura, y debe llevarse a un mes-año. Se construye una nueva tabla, de nombre *DW-Fechas-3*, sólo con los atributos deseados, agrupando por éstos.

Esto actualiza la función de mapeo, para que referencie a los atributos de la nueva tabla cuando corresponda. La Figura 36b muestra la función de mapeo actualizada.

Especificación:

RULE R5 - GROUP

SUBRULE R5.1 – FRAGMENT GROUP

Description:

Dado un fragmento, su función de mapeo y una tabla que lo mapea, se genera una nueva tabla eliminando los atributos no mapeados y agrupando por los demás.

Objects:

- OS ∈ SchFragments */* un fragmento */*

Input:

- **Maps:** F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
- **Tables:** T ∈ MapTables(F, ObjectItems(OS)) */* tabla que mapea al objeto */*

Conditions:

- T.As – MapAttributes(F, ObjectItems(OS)) ≠ ∅ */* hay atributos para eliminar */*
- # MapTables(F, ObjectItems(OS)) = 1 */* el objeto es mapeado por una sóla tabla */*

Result:

- **Tables:**
  - 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 */*
- **Maps:**
  - F' = ReplaceTable (F,T, T')
  - UpdateMaps (F', OS)

RULE R5 – GROUP

SUBRULE R5.2 – CUBE GROUP

Description:

Dado un cubo, su función de mapeo explícita y una tabla que lo mapea, se genera una nueva tabla eliminando los atributos no mapeados, agrupando por los demás y aplicando roll-up a los atributos que mapean a las medidas.

Objects:

- OS ∈ SchCubes */* un cubo */*
- Rup ⊆ RollUpExpressions(OS.Measure♦Is) */* predicados de roll-up para las medidas */*

Input:

- **Maps:** F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
- **Tables:** T ∈ MapTables(F, ObjectItems(OS)) */* tabla que mapea al objeto */*

Conditions:

- T.As – MapAttributes(F, ObjectItems(OS)) ≠ ∅ */* hay atributos para eliminar */*
- # MapTables(F, ObjectItems(OS)) = 1 */* el objeto es mapeado por una sóla tabla */*

Result:

- **Tables:**
  - 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 */*
- **Maps:**
  - F' = ReplaceTable (F,T, T')
  - UpdateMaps (F', OS)

Rule R6. Drill-Up

Objetivo:

El objetivo de esta regla es construir la tabla de hechos asociada a un cubo con mapeo recursivo. El mapeo del cubo se hace en base a otro cubo, (cubo base) que tiene más detalle con respecto a algunas dimensiones.

La regla se utiliza para construir una tabla de hechos con menos detalle que la tabla que mapea al cubo base. Para ello se aplica un drill-up por alguna de las dimensiones del cubo, se agrupa por los restantes atributos y se aplica una función de roll-up a los atributos que mapean a las medidas. Se aplican las primitivas P8 – Hierarchy Roll Up o P9 – Aggregate Generation.

Consideraciones:

- Esta regla se debe aplicar después de la regla R1 (Join), que construye los esqueletos de los objetos. Esto garantiza que los atributos mapeados estén en el esqueleto.
- Cuando no interesa información de detalle por una dimensión, se aplica la primitiva P9.
- Cuando interesa reducir el nivel de detalle de una dimensión, se aplica la primitiva P8. Para ello es necesario aplicar previamente Join al mapeo de la jerarquía, de manera de construir un esqueleto para ésta.

Variantes:

Se tienen dos variantes dependiendo de los nuevos niveles del drill-up:

- *Hierarchy-Drill-Up*: Resuelve el drill-up cuando interesa reducir el nivel de detalle de una jerarquía de dimensión.
- *Total-Drill-Up*: Resuelve el drill-up cuando no interesa tener detalle de una dimensión.

Ejemplo:

La Figura 37 muestra el cubo *venta-1* y su mapeo a la tabla *DW-Venta-1*. La Figura 38 muestra el cubo *venta-2* y su mapeo a partir del cubo *venta-1*, como drill-up a partir de la dimensión *clientes*. Los ítems de la dimensión *clientes* de ambos cubos mapean a la tabla *Clientes*.

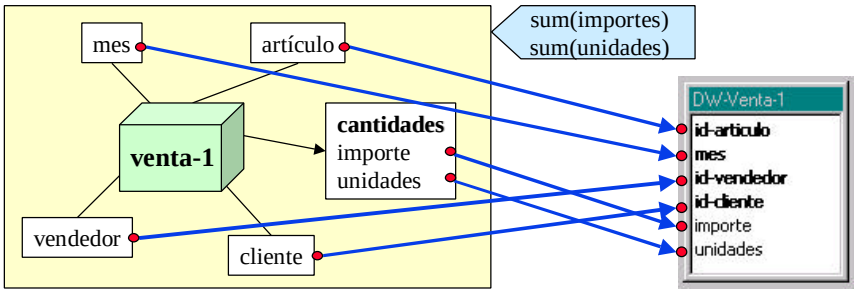

Figura 37 – Mapeo del cubo venta-1

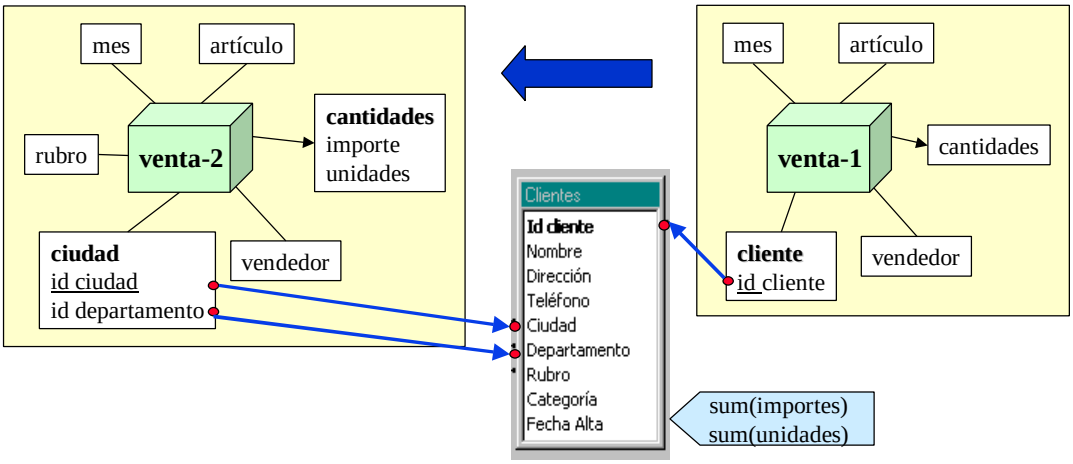

Figura 38 – Mapeo del cubo venta-2 a partir del cubo venta-1, antes de aplicar drill-up

Se construye una nueva tabla, de nombre *DW-Venta-2*, haciendo un drill-up por la dimensión clientes a la tabla *DW-Venta-1*.

Esto actualiza la función de mapeo, para que referencie a los atributos de la nueva tabla cuando corresponda. La Figura 39 muestra la función de mapeo actualizada.

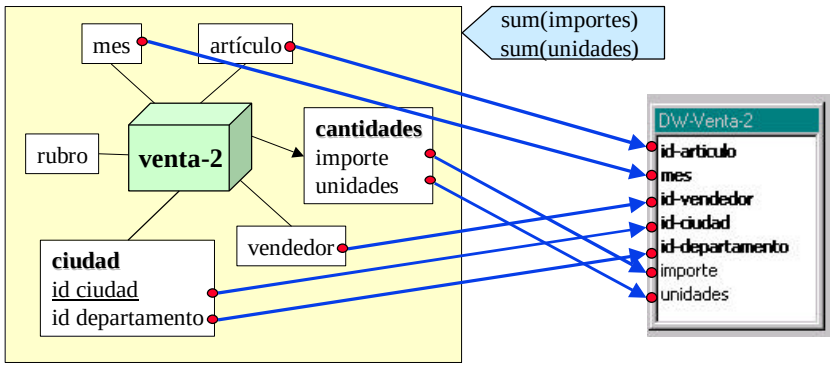

Figura 39 – Mapeo del cubo venta-2 después de aplicar drill-up

Especificación:

RULE R6 - DRILL-UP

SUBRULE R6.1 – HIERARCHY-DRILL-UP

Description:

Dados dos cubos (base y recursivo), la función de mapeo y la tabla que mapea al cubo base, y un drill-up respecto a una de las dimensiones y la función de mapeo de su jerarquía, se genera una nueva tabla con el nuevo nivel de detalle para la dimensión aplicando roll-up a las medidas.

Objects:

- CB, CR ∈ SchCubes */* los cubos base y recursivo */*
- Dup ∈ DrillUps(CB, CR) */* un drill-up */*

Input:

- **Maps:** F ∈ Mappings(ObjectItems(CB)) */* función de mapeo del cubo base */*
  Fd = Dup.Map */* función de mapeo de la jerarquía del drill-up */*
- **Tables:** 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 */*

Conditions:

- Dup.Lnews ≠ ∅ */* sino es eliminar completamente la dimensión */*
- # MapTables(F, ObjectItems(CB)) = 1 */* el cubo base es mapeado por una sóla tabla */*
- # MapTables(Dup.Map,ObjectKeyItems(Dup.Lbase) ∪ Dup.Lnews♦ObjectKeyItems)
  / los involucrados en el roll-up mapean a una sóla tabla */*

Result:

- **Tables:**
  - 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 */*
- **Maps:**
  - F’ = ReplaceTable (F,T, T’)
  - UpdateMaps (F’, CR)

RULE R6 – DRILL-UP

SUBRULE R6.2 – TOTAL-DRILL-UP

Description:

Dados dos cubos (base y recursivo), la función de mapeo y la tabla que mapea al 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.

Objects:

- CB, CR ∈ SchCubes */* los cubos base y recursivo */*
- Dup ∈ DrillUps(CB, CR) */* un drill-up */*

Input:

- **Maps:** F ∈ Mappings(ObjectItems(CB)) */* función de mapeo del cubo base */*
- **Tables:** T ∈ MapTables(F, ObjectItems(CB)) */* tabla que mapea al cubo base */*

Conditions:

- Dup.Lnews = ∅ */* se elimina la dimensión */*
- # MapTables(F, ObjectItems(CB)) = 1 */* el cubo base es mapeado por una sóla tabla */*

Result:

- **Tables:**
  - 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 */*
- **Maps:**
  - F' = ReplaceTable (F,T, T')
  - UpdateMaps (F', CR)

Rule R7. Primary Key

Objetivo:

El objetivo de esta regla es modificar las claves de las tablas cuando no coinciden con las claves de los objetos a los que mapean. Se aplica la primitiva Q4 – Primary Key Update.

Consideraciones:

- Esta regla se debe aplicar después de la regla R1 (Join), que construye los esqueletos de los objetos. Esto garantiza que los atributos mapeados estén en el esqueleto.

Ejemplo:

La Figura 40a muestra la dimensión *Geografía*, la cual se decidió normalizar. El fragmento rosa (con el nivel *ciudad*) mapea a la tabla *Ciudades*.

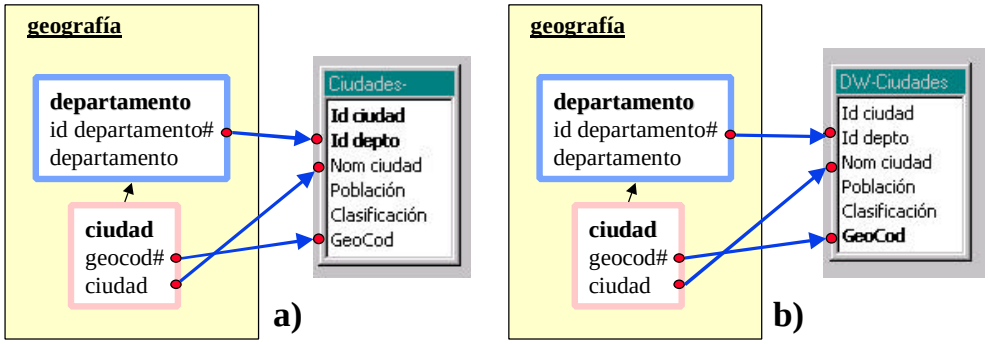

Figura 40 – Mapeo de la dimensión geografía: a) antes y, b) después de aplicar Primary Key

El ítem *geocod* es clave del nivel ciudad. El ítem *geocod* de la tabla *Ciudades* se construyó concatenando los atributos *id-ciudad* e *id-depto*, por lo tanto no figura en la clave, ya que es un atributo calculado.

Pero *geocod* es otra clave de la tabla *Ciudades* y se quiere que sea la clave primaria. Se construye una nueva tabla con idéntica estructura pero con clave primaria *geocod*.

Esto actualiza la función de mapeo, para que referencie a los atributos de la nueva tabla cuando corresponda. La Figura 40b muestra la función de mapeo actualizada.

Especificación:

RULE R7 - PRIMARY KEY

Description:

Dado un objeto del esquema intermedio, su función de mapeo y una tabla que lo mapea, se genera una nueva tabla modificando su clave primaria.

Objects:

- OS ∈ SchFragments ∪ SchCubes */* un fragmento o un cubo */*

Input:

- **Maps:** F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
- **Tables:** T ∈ MapTables(F, ObjectItems(OS)) */* tabla que mapea al objeto */*

Conditions:

- MapAttributes(F, ObjectKeyItems(OS)) ≠ T.PK */* las claves no coinciden */*
- # MapTables(F, ObjectItems(OS)) = 1 */* el objeto es mapeado por una sóla tabla */*

Result:

- **Tables:**
  - 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 */*
- **Maps:**
  - F' = ReplaceTable (F,T, T')
  - UpdateMaps (F', OS)

Rule R8. Filter

Objetivo:

El objetivo de esta regla es especificar que las tuplas de la tabla que mapean a un objeto deben cumplir con una condición dada, y descartar las tuplas que no la cumplen.

Se aplica la primitiva Q3 – Instance Filter. La aplicación de dicha primitiva genera información en la traza de diseño (ver sección 3) que es de utilidad para generar los programas de carga y actualización del DW. En este punto no se genera una restricción de integridad, ni código que controle la condición, esto se hará más adelante durante la generación de los programas de carga.

La regla se utiliza para aplicar las condiciones indicadas en los mapeos, y para particionar horizontalmente un cubo utilizando las condiciones de las franjas.

Consideraciones:

- Esta regla se debe aplicar después de la regla R1 (Join), que construye los esqueletos de los objetos. Esto garantiza que los atributos mapeados estén en el esqueleto.

Variantes:

Se tienen dos variantes:

- *Instance-Filter*: Resuelve el filtrado de condiciones especificadas en los mapeos.
- *Strip-Filter*: Resuelve la fragmentación horizontal de cubos aplicando las condiciones de las franjas.

Ejemplo:

La Figura 41 muestra la dimensión *Clientes*, la cual se quiere normalizar. El fragmento que contiene al nivel *cliente* (incluye la clave absoluta del nivel padre) mapea a la tabla *Clientes*.

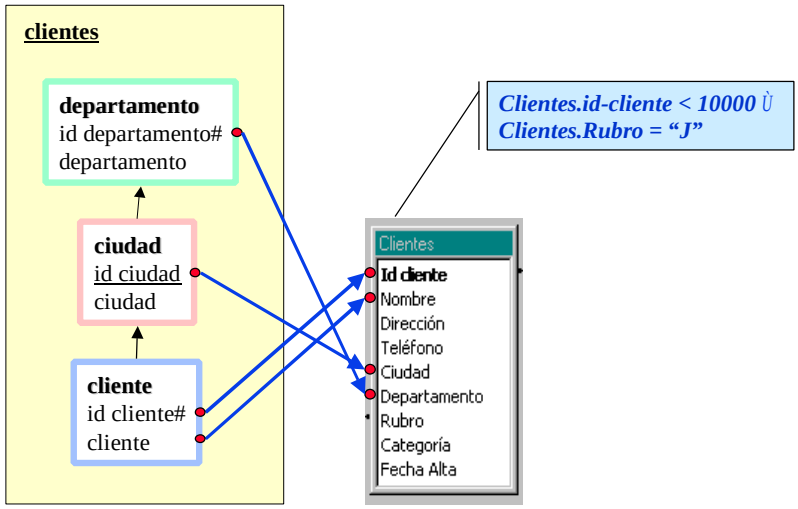

Figura 41 – Mapeo de la dimensión clientes

Sólo interesan los clientes uruguayos, que son los que tienen identificador (*id cliente*) menor a 10000, restricción proveniente del modelo conceptual.

En la tabla *Clientes* se almacenen clientes de varios sistemas o secciones, y para el DW sólo interesan los clientes de *Rubro J*. Es una restricción respecto a la fuente.

Ambas restricciones se tuvieron en cuenta al establecer los mapeos, por lo que el mapeo del fragmento tiene como condición:

Clientes.id-cliente < 10000 Ù Clientes.Rubro = "J"

Se construye una nueva tabla con la misma estructura pero cuyas tuplas cumplan dicha condición. Gráficamente la función de mapeo queda igual.

Especificación:

RULE R8 - FILTER

Description:

Dado un objeto del esquema intermedio, su función de mapeo, una tabla que lo mapea, y una condición que deben cumplir las instancias, se genera una nueva tabla filtrando las tuplas que no cumplen la condición.

Objects:

- OS ∈ SchFragments ∪ SchCubes ∪ SchStrips */* un fragmento, un cubo o una franja */*
- Cond ∈ Predicates({T •As / T ∈ SchTables}) */* una condición */*

Input:

- **Maps:** F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
- **Tables:** T ∈ MapTables(F, ObjectItems(OS)) */* tabla que mapea al objeto */*

Conditions:

- Cond ≠ ⊥ */* la condición está bien definida */*
- # MapTables(F, ObjectItems(OS)) = 1 */* el objeto es mapeado por una sóla tabla */*

Result:

- **Tables:**
  - 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 */*
- **Maps:**
  - F' = ReplaceTable (F,T, T')
  - UpdateMaps (F', OS)

5. Especificación de un Algoritmo de Generación del Esquema Relacional

En esta sección se presenta un algoritmo para construir el esquema lógico de un DW. Dicho algoritmo propone un orden de aplicación para las reglas descritas en la sección anterior.

El algoritmo se divide en dos partes: (1) construcción de las tablas de dimensión y (2) construcción de las tablas de hechos. La última parte, a su vez, se divide en tres sub-partes: (a) construcción de tablas de hechos para cubos con mapeo base, (b) construcción de tablas de hechos para cubos con mapeo recursivo, y (c) construcción de tablas de hechos para franjas de cubos.

PARTE 1:

Para construir las tablas de dimensión se ejecutan los siguientes pasos:

Step 1: Construir los esqueletos.
Step 2: Renombrar atributos para ítems con mapeo directo.
Step 3: Generar atributos para ítems con mapeo calculado o externo.
Step 4: Aplicar filtros.
Step 5: Eliminar atributos no mapeados.
Step 6: Ajustar las claves.

PARTE 2:

a) Para construir las tablas de hechos para cubos con mapeo base se ejecutan los siguientes pasos:

Step 7: Construir los esqueletos.
Step 8: Renombrar atributos para ítems con mapeo directo.
Step 9: Generar atributos para ítems con mapeo calculado o externo.
Step 10: Aplicar filtros.
Step 11: Eliminar atributos no mapeados.
Step 12: Ajustar las claves.

b) Para construir las tablas de hechos para cubos con mapeo recursivo se ejecutan los siguientes pasos:

Step 13: Construir las tablas de las jerarquías.
Step 14: Aplicar los drill-ups.

c) Para construir las tablas de hechos (para las franjas de todos los cubos) se ejecuta un único paso:

Step 15: Construir las tablas de las franjas.

La construcción de tablas de dimensión puede ser independiente de la construcción de tablas de hechos. Sin embargo en muchos casos interesa definir claves foráneas (de las tablas de hechos a las tablas de dimensiones) o realizar chequeos de integridad durante la carga, por ejemplo controlar que los valores de una tabla de hechos correspondientes a una dimensión sean instancias de las tablas de dimensión. Para esto es más cómodo generar primero las tablas de dimensión.

En cuanto a las tablas de hecho, hay que generar primero las tablas para cubos con mapeo base, ya que éstas se utilizan en la generación de las tablas para cubos con mapeo recursivo. Luego de construidas las tablas de hecho (para cubos con ambos tipos de mapeos) se las fragmenta de acuerdo a las franjas.

El algoritmo puede automatizarse completamente. También podría ejecutarse paso a paso intercalando transformaciones sugeridas por el diseñador.

En las siguientes secciones se especifica cada paso.

5.1. Construcción de Tablas de Dimensión (para fragmentos de dimensiones)

Se construye una tabla de dimensión (dimension table) para cada fragmento de dimensión. Se utiliza el lineamiento de fragmentación de dimensiones (SchDFragmentation) para identificar qué fragmentos quieren almacenarse separadamente.

Durante la ejecución se utiliza (y se va modificando) la función de mapeo de cada fragmento.

Como primer paso se construye el esqueleto de cada fragmento. Un esqueleto es una tabla única que mapea al fragmento, la cual contiene todos atributos mapeados en forma simple (mapeos directME o 1calcME) por los ítems del fragmento. El esqueleto se obtiene de combinar varias tablas fuentes mediante la operación de join, y se va transformando en los sucesivos pasos del algoritmo.

Los ítems de los fragmentos pueden mapear a diferentes tablas, por lo que debe aplicar *join* a éstas hasta obtener una única tabla. Esa tabla es el esqueleto inicial. En los sucesivos pasos se utiliza esta propiedad (el fragmento mapea a una única tabla), la cual es condición necesaria para la mayoría de las reglas.

Los atributos que tienen mapeos directos ya están en el esqueleto pero pueden tener nombres no mnemotécnicos, por lo que se los renombra utilizando los nombres de los ítems que los mapean. Con esto se resuelven los mapeos directos.

Luego se resuelven los restantes tipos de mapeos: se generan atributos para ítems que mapean a una expresión, y para ítems que tienen mapeos externos. En este punto todos los ítems tienen mapeos directos (a los atributos renombrados, o a los nuevos atributos generados) y los atributos a los que mapean están en el esqueleto.

Se deben aplicar filtros a las tuplas del esqueleto de acuerdo a las condiciones impuestas en los mapeos. El filtrado se difiere hasta este momento, en que todos los ítems tienen sus mapeos al esqueleto y no se han eliminado los atributos sobrantes, ya que las condiciones pueden involucrar a cualquiera de los atributos, incluso los que no mapean a ningún ítem.

Hasta ahora sólo se agregaron atributos al esqueleto, no se eliminó ninguno. Es necesario eliminar los atributos que no son mapeados. Al eliminarlos, se pueden borrar atributos que determinaban la granularidad e incluso la clave primaria de la tabla, por lo que es conveniente agrupar por los atributos restantes. La eliminación no puede realizarse antes para no borrar atributos que puedan ser necesarios para realizar cálculos o joins con tablas necesarias para los cálculos o para aplicar filtros.

Luego se ajusta la clave primaria de acuerdo a la clave del fragmento. Debe ser el último paso, para asegurar que estén todos los atributos, y que no sobre ninguno, lo que cambiaría la granularidad pudiendo incluso invalidar la clave.

A continuación se detallan los pasos en la construcción de las tablas de los fragmentos:

STEP 1 - CONSTRUIR LOS ESQUELETOS

Para cada fragmento que mapee a más de una tabla, aplicar la regla R1 (Join), a dos de las tablas a las que mapea y que están relacionadas (tienen definido un link). Se itera hasta que el fragmento mapee 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*

STEP 2 - RENOMBRAR ATRIBUTOS PARA ÍTEMS CON MAPEO DIRECTO

Para cada fragmento, que tenga ítems con mapeo directo, cuyos nombres difieran de los nombres de los atributos que los mapean, 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*

STEP 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 mapeo calculado, y la regla R4 (Extern, en alguna de sus versiones) para cada ítem con mapeo externo.

∀ 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*

STEP 4 - APLICAR FILTROS

Para cada fragmento, que tenga mapeo con 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*

STEP 5 - ELIMINAR ATRIBUTOS NO MAPEADOS

Para cada fragmento, cuyo esqueleto tenga atributos que no mapean 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*

STEP 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*

5.2. Construcción de Tablas de Hechos (para cubos con mapeo base)

Se construye una tabla de hechos (fact table) para cada cubo con mapeo base. Se utiliza el lineamiento de materialización de relaciones (SchCubes) para identificar qué cubos quieren materializarse, y la función de mapeo de cubos (SchCMappings) para determinar cuáles tienen mapeo base.

En un primer momento se trabaja con los cubos que tienen asociada una función de mapeo base, y luego se resuelven los mapeos recursivos. Durante la ejecución se utiliza (y se va modificando) la función de mapeo de cada cubo.

Como primer paso se construye el esqueleto de cada cubo. De forma análoga que para fragmentos, un esqueleto es una tabla única que mapea al cubo, la cual contiene todos los atributos mapeados en forma simple (directME o 1calcME). El esqueleto se va transformando en los sucesivos pasos del algoritmo.

Los ítems de los cubos pueden mapear a diferentes tablas, por lo que debe aplicarse *join a* éstas hasta obtener una única tabla. Esa tabla es el esqueleto inicial. En los sucesivos pasos se utiliza esta propiedad (el cubo mapea a una única tabla), la cual es condición necesaria para la mayoría de las reglas.

Análogamente se deben renombrar los atributos con mapeos directos, y agregar atributos para los ítems con mapeos calculados y externos. En este punto todos los ítems tienen mapeos directos y los atributos a los que mapean están en el esqueleto.

Se deben aplicar filtros a las tuplas del esqueleto de acuerdo a las condiciones impuestas en los mapeos. Luego es necesario eliminar los atributos que no mapean, agrupando por los restantes y aplicando roll-up a las medidas. Luego se ajusta la clave primaria de acuerdo a la clave del cubo. La justificación del orden de los pasos es análoga que para los fragmentos.

A continuación se detallan los pasos en la construcción de las tablas de hechos de los cubos con mapeo base:

STEP 7 - CONSTRUIR LOS ESQUELETOS

Para cada cubo con mapeo base, que mapee a más de una tabla, aplicar la regla R1 (Join), a dos de las tablas que lo mapean y están relacionadas (tienen definido un link). Iterar hasta que el cubo mapee 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 */*

STEP 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 */*

Ejecutar Aplicar (C, Map, T) */* genera una nueva tabla, y actualiza la función de mapeo */*

Fin */* para todo cubo */*

STEP 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 mapeo calculado, y la regla R4 (Extern, en alguna de sus versiones) para cada ítem con mapeo externo.

∀ 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 */*

STEP 10 - APLICAR FILTROS

Para cada cubo con mapeo base, que tenga mapeo con 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 */*

STEP 11 - ELIMINAR ATRIBUTOS NO MAPEADOS

Para cada cubo con mapeo base, cuyo esqueleto tenga atributos que no mapean 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 */*

STEP 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 */*

5.3. Construcción de Tablas de Hechos (para cubos con mapeo recursivo)

Se construye una tabla de hechos (fact table) para cada cubo con mapeo recursivo. Se utiliza el lineamiento de materialización de relaciones (SchCubes) para identificar qué cubos quieren materializarse, y la función de mapeo de cubos (SchCMappings) para determinar cuáles tienen mapeo recursivo.

En este punto se trabaja con los cubos que tienen mapeo recursivo, es decir, que su mapeo está definido en términos de otro cubo, al cual se le debe aplicar una secuencia de drill-ups. Se utilizan las funciones de roll-up asociadas a las medidas, y funciones que mapean a las jerarquías de las dimensiones a las que se aplica drill-up (si es necesario).

Para cada dimensión a la que se aplica drill-up se tiene una función de mapeo para los ítems de esa dimensión en ambos cubos.

En un primer paso se deben procesar las funciones de mapeo de las jerarquías, como que fueran funciones de mapeo de fragmentos: armando el esqueleto, ajustando los atributos, filtrando valores, eliminando sobrantes y ajustando la clave. Esto es necesario para asegurar que el posterior drill-up se haga en un sólo paso, y tenga en ese esqueleto todos los atributos necesarios para ejecutarse.

Luego se efectúa el drill-up.

En este momento el cubo pasa a tener mapeo base, ya que se construyó un esqueleto para él.

A continuación se detallan los pasos en la construcción de las tablas de hechos de los cubos con mapeo recursivo:

STEP 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 y 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 */*

STEP 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 */*

5.4. Construcción de Tablas de Hechos (para franjas de cubos)

Se construye una tabla de hechos para cada franja de cubo. Se utiliza el lineamiento de materialización de relaciones (SchCubes) para identificar qué cubos quieren materializarse, y el lineamiento de fragmentación de cubos (SchCFragmentation) para identificar que franjas deben almacenarse por separado.

Se deben aplicar filtros a las tuplas del esqueleto de cada cubo, de acuerdo a las condiciones de las franjas. Este paso se deja para el final, momento en el cual todos los cubos tienen su tabla de hechos.

A continuación se detallan los pasos en la separación de las franjas de los cubos:

STEP 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 */*

6. Conclusiones

En este capítulo se presentó la segunda etapa del proceso de diseño lógico: la generación del esquema lógico del DW.

Se presentaron un conjunto de reglas de diseño que son aplicadas a objetos del esquema intermedio que cumplen una serie condiciones y dan como resultado la aplicación de una transformación al esquema lógico del DW. Estas reglas permiten construir los casos más frecuentes de DWs pero no pretenden ser completas en el sentido de guiar a cualquier estilo de diseño.

Por último se formuló un algoritmo que especifica el orden en que se deben aplicar las reglas para construir un esquema lógico que siga los lineamientos.

Capítulo V - Prototipación de una Herramienta CASE

1. Introducción

Este trabajo es parte del proyecto DwDesigner[20] del Laboratorio CSI[21], un proyecto de investigación en las áreas de diseño conceptual y lógico de Data Warehouses. El objetivo del proyecto es proveer al diseñador de un DW de herramientas que lo asistan automatizando tareas de diseño y ambientes gráficos que faciliten el proceso de diseño.

El proyecto incluye el desarrollo de un Modelo Conceptual Multidimensional: CMDM [Car00], [Pic00], técnicas para diseño lógico basado en transformaciones de esquemas [Mar00], [Gar00], [Per00], la repercusión de la evolución del esquema fuente en el DW [Mar00], [Alc01], la generación del esquema lógico a partir del esquema conceptual [Per00a], [Per01], y la persistencia de los objetos diseñados [Arz00].

En este trabajo se desarrolla el prototipo de una herramienta CASE que soporta la generación del esquema relacional del DW a partir del esquema conceptual. El prototipo incluye una interfaz para la definición de lineamientos, una interfaz para la definición de mapeos, y la aplicación automática del algoritmo de generación del esquema lógico.

En las siguientes secciones se describe brevemente la herramienta. En la sección 2 se presenta la arquitectura del ambiente CASE al que pertenece la herramienta y en la sección 3 se describe DwDesigner. En la sección 4 se presentan las generalidades de diseño e implementación, que son complementadas con los diagramas de clases que se presentan en el Anexo 6. En la sección 5 se concluye.

[20] El proyecto DwDesigner es un proyecto de investigación en las áreas de Diseño Conceptual y Lógico de Data Warehouses. http://www.fing.edu.uy/~csi/Proyectos/DwDesigner/

[21] CSI: Laboratorio Concepción de Sistemas de Información – Instituto de Computación – Facultad de Ingeniería – Universidad de la República. http://www.fing.edu.uy/~csi/

2. Arquitectura del Ambiente CASE

El proyecto DwDesigner propone el desarrollo de un ambiente basado en herramientas CASE para diseño de DWs, diseñadas y programadas en forma totalmente independiente y que se comunican a través de una interfaz CORBA. La Figura 42 muestra una arquitectura simplificada del ambiente.

En el nivel más bajo se cuenta con una *interfaz CORBA* para intercomunicación de objetos. Las herramientas de diseño se comunican a través de la interfaz CORBA, y guardan los objetos diseñados e información de control en un *repositorio de diseños*, dialogando con un *manejador de repositorio* [Arz00].

En un nivel más alto están las herramientas de diseño. Actualmente se cuenta con el prototipo de herramienta de diseño conceptual: *EdCMDM*, que es un editor gráfico de CMDM [Pic00] y el prototipo de una herramienta de diseño lógico: DwDesigner, que provee una interfaz gráfica para la generación del esquema relacional del DW aplicando transformaciones de esquema a una base de datos fuente integrada [Gar00], [Per01] e incluye manejo de evolución de esquemas [Alc01].

Estos prototipos fueron desarrollados en el marco de proyectos de grado y tesis de maestrías.

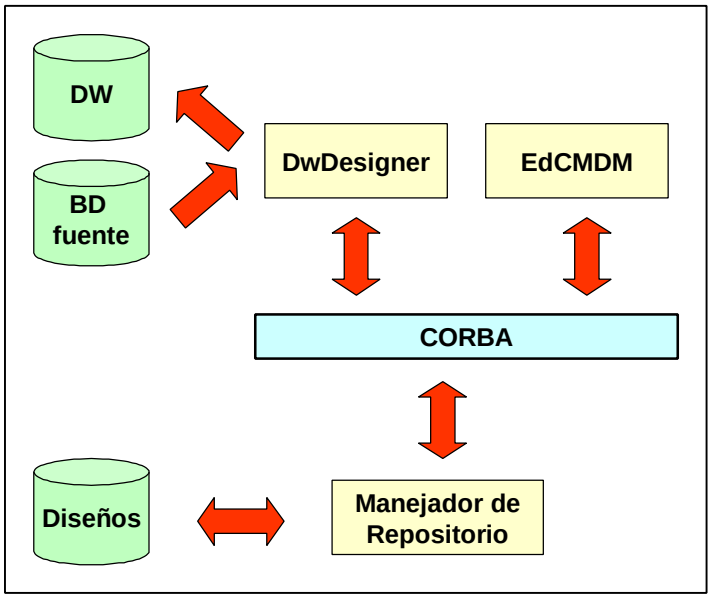

Figura 42 – Arquitectura del ambiente CASE

3. Descripción de la Herramienta

El objetivo de la herramienta DwDesigner es asistir al diseñador en la construcción de un DW relacional.

En este trabajo se desarrolla la tercer versión de la herramienta. La primer versión resuelve la aplicación interactiva de transformaciones de esquemas (primitivas) a una base fuente integrada [Gar00], y el chequeo de reglas de consistencia e invariantes del esquema [Mar00]. La segunda versión incorpora el manejo de evolución de los esquemas fuentes y su repercusión en el DW [Alc01]. Esta tercer versión incorpora a DwDesigner la definición interactiva de lineamientos y mapeos a la base fuente y la generación automática del esquema lógico relacional del DW a partir de un esquema conceptual de CMDM.

Dentro de las extensiones de esta tercer versión se tienen las siguientes funcionalidades:

Edición de lineamientos:

La herramienta presenta el esquema conceptual, con una interfaz similar a la de EdCMDM, y permite definir lineamientos interactivamente. Por ejemplo, presenta la definición de una dimensión, mostrando las jerarquías de niveles y permite fragmentarla seleccionando gráficamente qué niveles conforman cada fragmento (Figura 43). Se implementan también algunos algoritmos para definir lineamientos por defecto y la posibilidad de navegar y modificar los lineamientos definidos. La herramienta advierte al diseñador cuando sus especificaciones son incorrectas o incompletas, permitiéndole modificar los lineamientos definidos; por ejemplo, si al fragmentar una dimensión algún nivel no pertenece a ningún fragmento se advierte al diseñador.

Se utiliza un esquema conceptual diseñado con la herramienta EdCMDM, y almacenado en el repositorio de diseños.

Edición de mapeos:

La herramienta presenta el esquema intermedio y permite especificar mapeos a una base fuente integrada. Para ello, el diseñador asocia a cada ítem de un objeto (fragmento de dimensión o cubo) una expresión de mapeo construida a partir de la base fuente. La herramienta permite construir las expresiones de mapeo definidas en esta tesis, y tiene prevista su extensión a otros tipos de expresiones. También se asiste al diseñador en la especificación de condiciones de mapeo y funciones de roll-up (para el mapeo de cubos). La herramienta permite visualizar y modificar gráficamente las funciones de mapeo (Figura 44). Análogamente a la edición de lineamientos, se advierte al diseñador cuando sus especificaciones son incorrectas o incompletas, permitiéndole realizar modificaciones.

Generación automática del esquema relacional:

La generación del esquema relacional se realiza automáticamente por lo que tiene muy poca interfaz hacia el diseñador, el cual sólo debe elegir una opción de un menú).

La generación consiste en la aplicación del algoritmo y las reglas de diseño propuestas en el Capítulo IV - secciones 4 y 5. La ejecución de cada regla implica la evaluación de sus precondiciones a partir de lineamientos y mapeos, la construcción de los parámetros necesarios para la aplicación de las primitivas correspondientes, y la ejecución de dichas primitivas. Como resultado de la ejecución de una regla se obtienen nuevas tablas en el DW, actualizaciones a las funciones de mapeo, y se refleja la aplicación de las primitivas en la traza de diseño.

Se incluye también la posibilidad de ejecutar paso a paso el algoritmo (Figura 45) de manera de monitorear la ejecución, pudiendo visualizar las relaciones generadas (Figura 46), las funciones de mapeo (Figura 47), y la traza de diseño (Figura 48).

Las funcionalidades de edición de lineamientos [Cal01] y mapeos [Bar01] están siendo desarrolladas en el marco de proyectos de estudiantes de ingeniería de software[22], propuestos y supervisados por el autor de esta tesis, en el rol de usuario del sistema [Per01].

En esta tesis se desarrollan las funcionalidades de generación automática del esquema lógico, resolviendo la interconexión con las primitivas de diseño. Para ello se prototipó con una interfaz sencilla, el acceso al esquema conceptual, lineamientos y mapeos, funcionalidades que serán sustituidas por las desarrolladas por los proyectos de ingeniería de software al final del semestre.

[22] Proyectos de la asignatura *Taller de Ingeniería de Software*, asignatura del octavo semestre de la carrera de Ingeniero en Computación, de la Facultad de Ingeniería. En esta asignatura los estudiantes (en grupos de aproximadamente 10 estudiantes) participan en un proyecto de ingeniería de software, guiado y controlado, sometido a diversos tipos de restricciones análogas a las que son comunes en la industria.

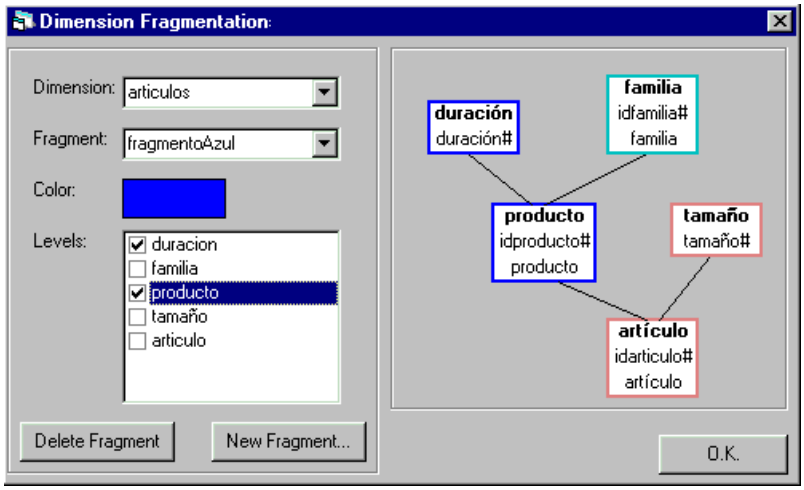

Figura 43 – Pantalla de definición de lineamientos

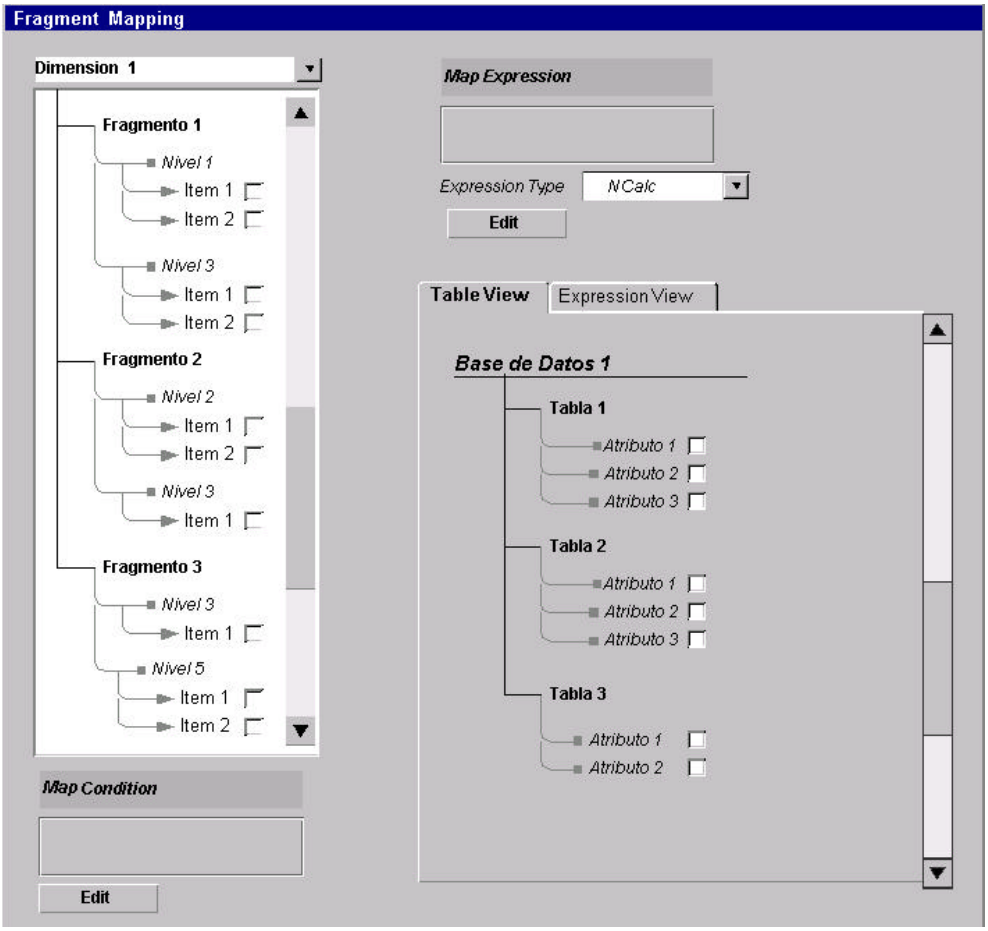

Figura 44 – Pantalla de definición de mapeos

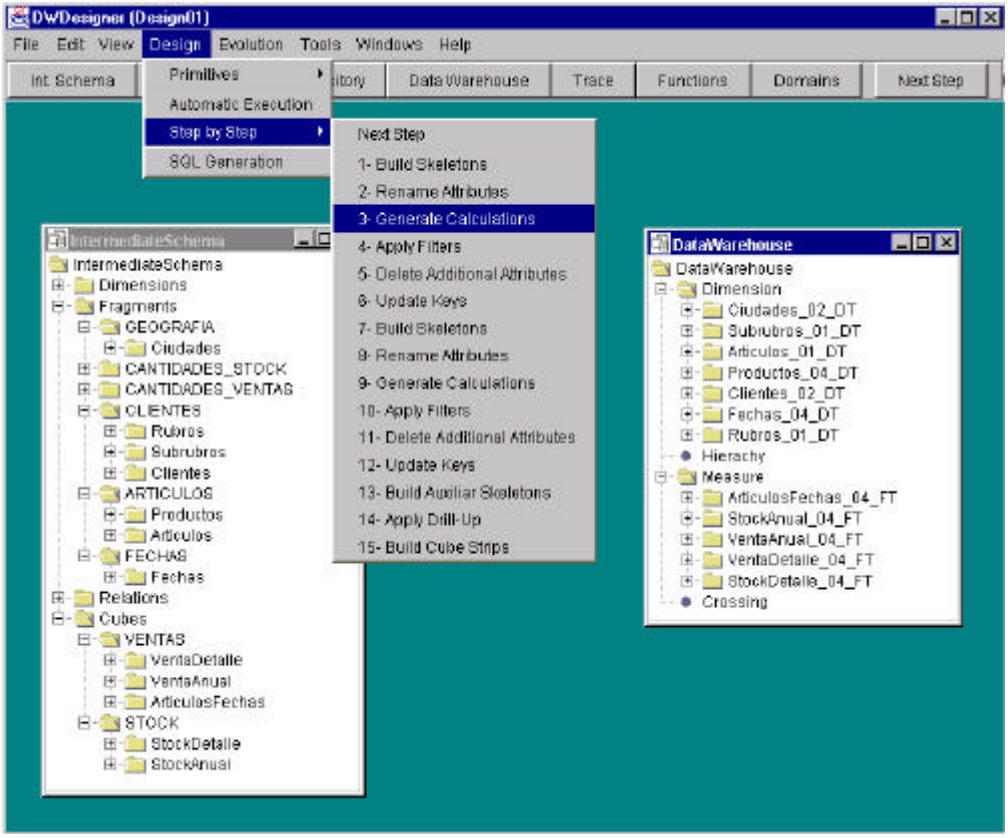

Figura 45 – Ejecución paso a paso del algoritmo

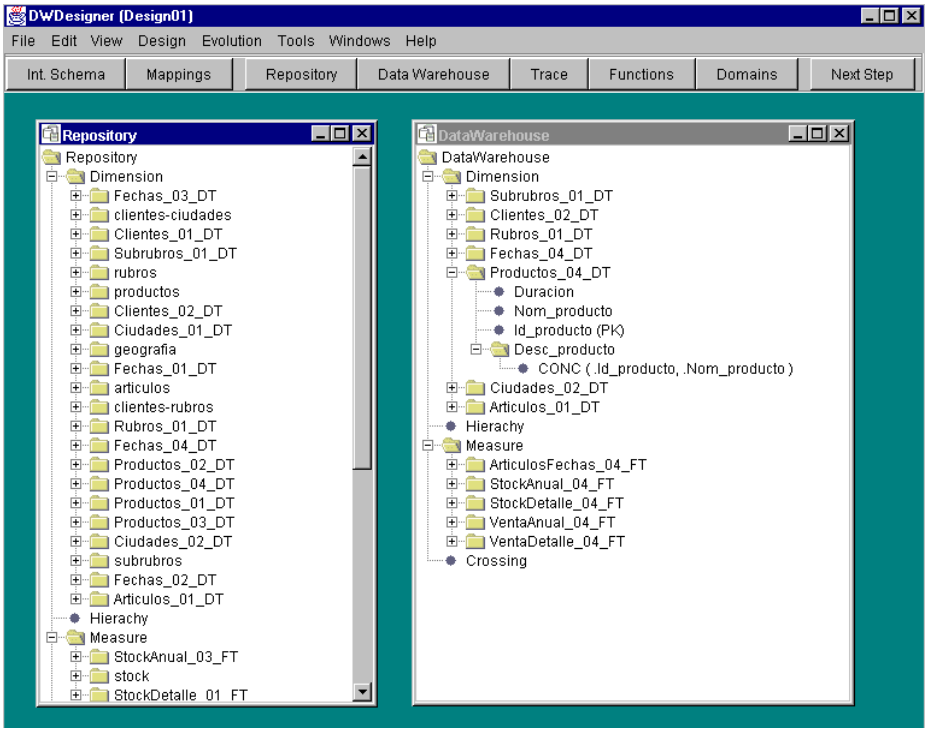

Figura 46 – Visualización de las tablas del DW

La Figura 46 muestra el DW (a la derecha) con sus tablas de dimensión (Dimension) y sus tablas de hechos (Measure). Se pueden ver los atributos de la tabla *Productos_04_DT*, uno de ellos: *Desc_productos* es un atributos calculado, como concatenación de los atributos *Id_producto* y *Nom_producto*. En el repositorio (a la izquierda) están todas las tablas, tanto las del DW como las tablas intermedias.

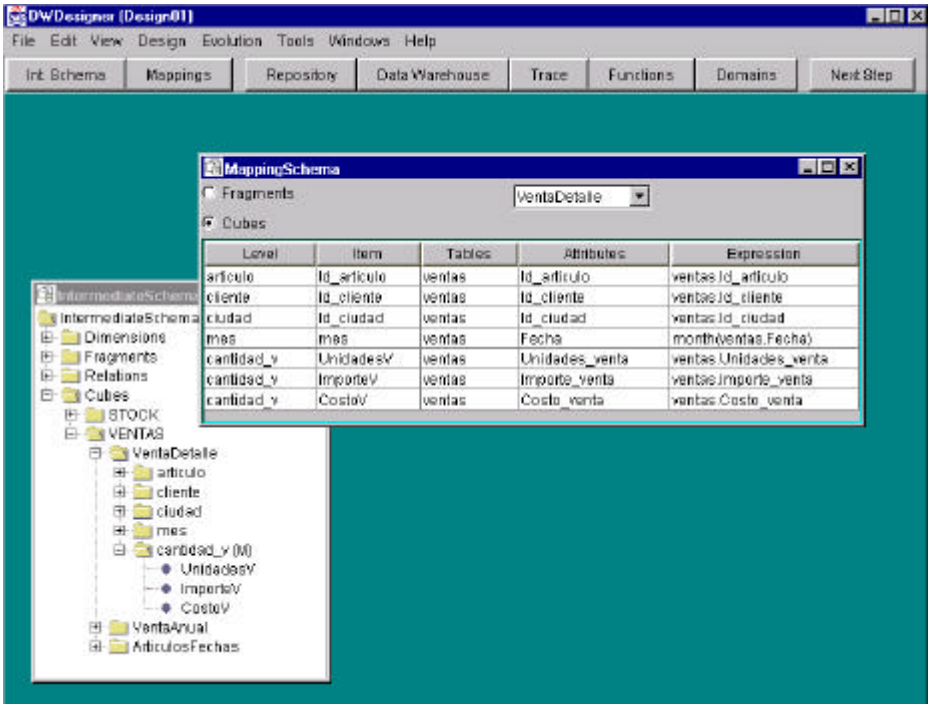

Figura 47 – Visualización de funciones de mapeo

La Figura 47 muestra el esquema intermedio (abajo, a la izquierda), y en particular, muestra la definición del cubo *VentaDetalle* que tiene por niveles: *artículo, cliente, ciudad, mes y cantidad_v*, el último nivel usado como medida. Se muestra también la función de mapeo del cubo (arriba a la derecha), que asocia a cada ítem de cada nivel, una expresión de mapeo (se muestran tablas y atributos mapeados y la expresión).

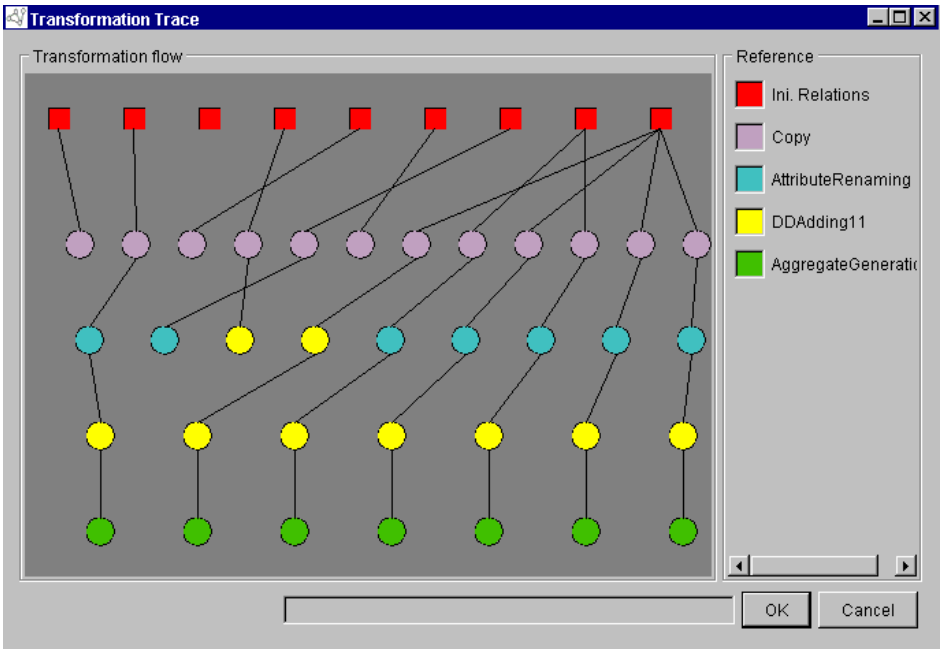

Figura 48 – Visualización de la traza de diseño

La Figura 48 muestra la traza de diseño. Los rectángulos (parte superior) representan las tablas de la base de datos fuente, y los círculos representan la aplicación de una primitiva. Las líneas indican que el nodo superior (una tabla fuente o una tabla intermedia resultado de la aplicación de una primitiva) es parámetro de la primitiva (nodo inferior).

4. Desarrollo del Prototipo

Para el diseño de la herramienta se utilizaron técnicas de orientación a objetos, y se utilizó UML como lenguaje de especificación. Los diagramas de clases se muestran en el Anexo 6.

La arquitectura de la herramienta se muestra en la Figura 49. Consta de tres niveles: *persistencia*, *aplicación* y *presentación.*

El nivel de persistencia provee funciones básicas para guardar y recuperar diseños (interfaz repositorio), tanto en disco local como en el repositorio de diseños. En dicho repositorio se almacenan el esquema conceptual, los lineamientos, los mapeos, el esquema lógico y la traza de diseño. Se proveen también funciones para conectarse a la base de datos fuente (interfaz conexión a datos) tanto para extraer el esquema fuente como para generar los scripts de creación de las tablas del DW.

El nivel de aplicación es que maneja la lógica de la herramienta. Contiene a la máquina virtual, la cual comprende las estructuras de datos para representar tanto al esquema intermedio como las tablas y atributos del esquema lógico. La máquina virtual proporciona los servicios necesarios para ejecutar las funcionalidades básicas, como aplicación de primitivas y reglas de diseño y mantenimiento de la traza.

El nivel de presentación permite la interacción entre el usuario y la máquina virtual. Esta interacción se lleva a cabo a través de interfaces gráficas para aplicación interactiva de primitivas (interfaz primitivas), repercusión de cambios del esquema fuente en el DW (interfaz evolución), definición de lineamientos (interfaz lineam.), definición de mapeos (interfaz mapeos) e interfaz (mínima) de generación automática del esquema relacional (interfaz generador).

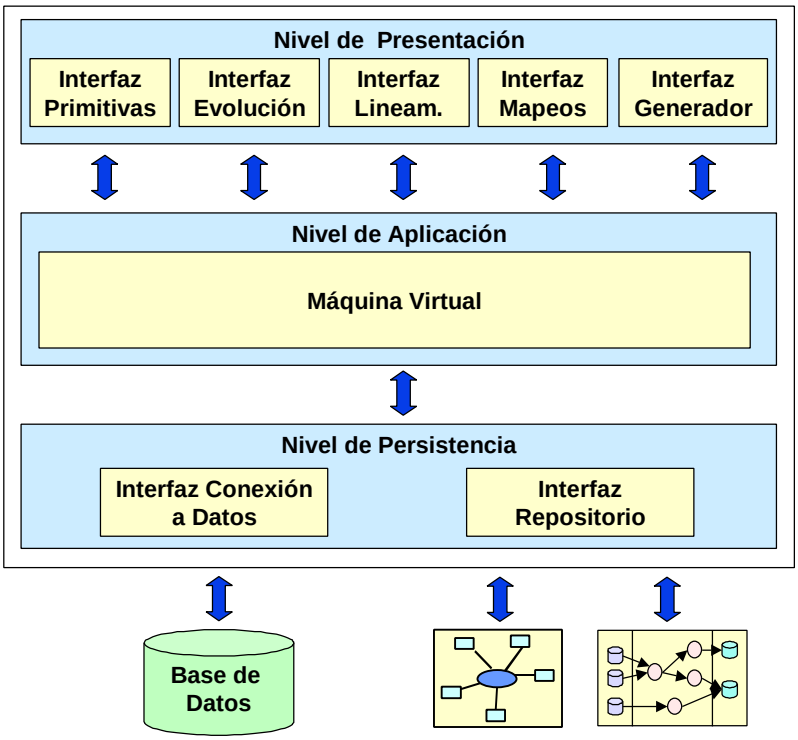

Figura 49 – Arquitectura de DwDesigner

El prototipo se implementó en Java. Se utilizó el compilador Java Development Kit versión 1.2.2 (jdk1.2.2) y la plataforma de desarrollo de Borland Jbuilder versión 2.

Los detalles de implementación de las versiones anteriores pueden encontrarse en [Gar00], [Mar00] y [Alc01]. La implementación de las funcionalidades de definición de lineamientos y mapeos se encuentran en [Cal01] y [Bar01] respectivamente. En esta sección se explican brevemente las generalidades de la implementación del algoritmo y las reglas de diseño.

Para la implementación de las reglas de diseño se utilizó una clase abstracta *TransformationRule* y sub-clases para cada una de las reglas. La clase *TransfRuleDirectory* es una secuencia, que contiene las reglas de diseño existentes y se inicializa al principio, a través de un archivo de configuración. Este mecanismo, análogo al de las primitivas, permite la incorporación de nuevas reglas sin necesidad de modificar ni recompilar el código existente.

Cada regla tiene una referencia a la primitiva que debe aplicar si se cumplen las precondiciones. El método *apply* de cada regla controla las precondiciones, arma la lista de parámetros que necesita la primitiva, ejecuta dicha primitiva y actualiza las funciones de mapeo.

El algoritmo se implementa como una iteración en una secuencia de pasos. Para la implementación de los pasos se utilizó una clase abstracta *Step* y sub-clases para cada una de los pasos. Análogamente se implementó la clase *StepDirectory*.

Cada paso tiene referencias a una o más reglas de diseño, que pueden ser aplicadas en dicho paso, dependiendo de las precondiciones de las reglas. El método *apply* de cada paso itera por los objetos del esquema intermedio (fragmentos, cubos o franjas, dependiendo del paso), y para cada objeto, determina si corresponde aplicar alguna regla, y en caso afirmativo invoca al método *apply* de la regla pasando como parámetro el objeto, su función de mapeo y las tablas a las que mapea.

En el menú *Design*, en la opción *Step by Step*, puede seleccionarse un paso, lo que desencadena la invocación al método *apply* del paso correspondiente, el cual se accede a través del *StepDirectory*. La opción *Automatic Execution* del mismo menú, itera por los pasos del *StepDirectory* invocando el método *apply* de cada uno.

En la clase *Core*, que implementa la máquina virtual de la herramienta (ver Anexo 6) se incorporaron las secuencias *StepDirectory* y *TransfRuleDirectory* que se inicializan al comenzar un nuevo diseño (menú *File* opción *New*). Se incorporaron también a la clase *Core* las clases *IntermediateSchema* y *MappingSchema* que implementan el esquema intermedio y las funciones de mapeo respectivamente. Dichas clases y todas las clases relacionadas, como componentes del esquema intermedio (*Dimension*, *Level*, *Cube*, etc.), funciones de mapeo (*Mapping*, *MapExpr*, etc.), editores gráficos (*IntExplorer*, *MapExplorer*, etc.), etc. se prototiparon para poder ejecutar el algoritmo, pero serán reemplazadas por las que se están desarrollando en los proyectos [Cal01] y [Bar01].

5. Conclusiones

DwDesigner es el prototipo de una herramienta CASE de diseño de DWs. En este trabajo se desarrolló la tercer versión del prototipo, desarrollando algunas funcionalidades de la herramienta y supervisando el desarrollo de otras.

La herramienta implementa la construcción de un esquema lógico relacional a partir de un esquema conceptual multidimensional diseñado con EdCMDM. Brinda interfaces gráficas y algoritmos por defecto que asisten al diseñador en la definición de lineamientos y mapeos a la base de datos fuente. La herramienta cuenta con un algoritmo automático de generación del esquema relacional, que según las precondiciones de las reglas de diseño, ejecuta las primitivas de transformación de esquemas.

La herramienta es parte de un ambiente de herramientas CASE de diseño de DWs que incluye: diseño conceptual, diseño lógico, evolución del esquema lógico y persistencia.

Capítulo VI - Conclusiones

1. Conclusiones

Esta tesis presenta un proceso de diseño para construir el esquema lógico de un DW relacional tomando como entrada el esquema conceptual y una base de datos fuente integrada. El proceso se divide en dos grandes etapas, la primera de definición de propiedades y correspondencias, es realizada por el diseñador, y la segunda de transformación del esquema fuente se realiza automáticamente.

Los resultados obtenidos consisten en:

- La especificación formal de lineamientos y mapeos, que complementan al esquema conceptual y lo relacionan con la base de datos fuente.
- La especificación de nuevas primitivas de transformación de esquema que extienden la propuesta de [Mar00].
- La definición de reglas de diseño que sugieren las primitivas adecuadas para aplicar según el esquema conceptual, los lineamientos y mapeos.
- Un algoritmo que encadena la aplicación de las reglas.
- Un prototipo de una herramienta CASE para diseño de DWs, que implementa los resultados anteriores.

Especificación de Lineamientos y Mapeos

La intervención del diseñador se hace al comienzo de la etapa de diseño lógico, a través de la definición de un conjunto de lineamientos que complementan al esquema conceptual con propiedades y estrategias de diseño lógico. No se induce a ninguna estrategia o estilo de diseño específico, sino que a través de los lineamientos el diseñador puede indicar sus requerimientos particulares. Por otro lado, los lineamientos podrían generarse automáticamente para estilos de diseño específicos (por ejemplo: un esquema estrella).

Los lineamientos hacen transparente la elección de estructuras relacionales adecuadas para el esquema del DW y permiten diferir el estudio de la base fuente. De esta manera, el diseñador se ocupa de definir propiedades que debe cumplir el esquema del DW, y no de cómo obtenerlo transformando la fuente.

El diseñador también relaciona los objetos del esquema intermedio (esquema conceptual más lineamientos) con los atributos de la base fuente. Para ello se proveen expresiones y funciones de mapeo que abstraen las propiedades relevantes de la fuente, útiles para la aplicación de las transformaciones.

Las funciones de mapeo se complementan con condiciones o restricciones que deben cumplir las instancias de la fuente. Estas condiciones se obtienen de restricciones del esquema conceptual y del conocimiento de la base fuente.

Especificación de Nuevas Primitivas

Las primitivas propuestas en [Mar00] fueron pensadas para una aplicación interactiva, en la que el diseñador elige las transformaciones adecuadas para su problema concreto.

Para brindar generalidad al proceso, en el sentido de conducir a un buen diseño del DW a partir de diferentes diseños de la fuente, se definieron nuevas primitivas de transformación para tratar casos más generales. Estas primitivas permiten generalizar condiciones de join entre tablas, independizarse de los nombres de los atributos, manejar más flexiblemente las claves y manejar restricciones de instancias.

Definición de Reglas de Diseño

Para la construcción del esquema lógico se proveen un conjunto de reglas de diseño que encapsulan los criterios de aplicación de las transformaciones dependiendo de las características del esquema intermedio. Las reglas se aplican a objetos del esquema intermedio que mapean a la base fuente y cumplen determinadas condiciones.

Se utilizan las primitivas de transformación de esquemas como unidades en el procesamiento. Esto permite la trazabilidad del diseño, lo cual es beneficioso como documentación para el mantenimiento y evolución del DW, pero también para relacionar el diseño con otras actividades como son la carga y actualización de los datos.

Definición del Algoritmo de Generación del Esquema Lógico

La aplicación de transformaciones sin un orden o estrategia definida puede conducir a un DW que no es adecuado a los requerimientos. Para ayudar al diseñador en la elección de las transformaciones adecuadas se presenta un algoritmo que ordena la aplicación de las reglas.

Este algoritmo puede ejecutarse sin la intervención del diseñador, pero puede también personalizarse para permitir al diseñador realizar sus propias transformaciones.

Implementación de una Herramienta CASE

Se prototipó una herramienta CASE que implementa el proceso de diseño propuesto en esta tesis y realiza la generación de un esquema relacional para un DW a partir de un esquema conceptual multidimensional y una base de datos fuente integrada.

La herramienta toma como entrada la especificación de un esquema conceptual y una base de datos fuente. El diseñador define lineamientos y mapeos, y luego ejecuta el algoritmo de generación del esquema lógico. Durante la ejecución del algoritmo se interactúa con la herramienta de aplicación de las primitivas [Gar00], [Alc01] y se genera la traza de diseño.

La herramienta puede conectarse con otros módulos que la complementan, incluyendo diseño conceptual [Pic00], y persistencia del diseño [Arz00].

Todos los módulos conforman un ambiente de herramientas CASE para diseño de DWs que cubre: diseño conceptual, diseño lógico, evolución del esquema lógico y persistencia.

2. Aportes y Limitaciones

La investigación realizada en este trabajo constituye un paso más en el desarrollo de técnicas y herramientas para diseño de DWs.

Los principales aportes consisten en:

- Especificación de la entrada al algoritmo de diseño: esquema conceptual, lineamientos, bases de datos fuentes y mapeos. La mayoría de las propuestas de diseño no lo incluyen o son limitadas.
- Especificación de reglas de diseño y un algoritmo que las ordena. Estas reglas conceptualizan criterios de diseño de DWs, sobre lo cual no se han identificado trabajos para DW.
- Implementación de una herramienta CASE para diseño lógico de DWs.

Dado un esquema conceptual especificado con CMDM, una base de datos fuente y un conjunto de mapeos entre ellos, el algoritmo genera un esquema lógico relacional para el DW que cumpla con un conjunto de lineamientos.

Sin embargo, la construcción ad-hoc de un esquema lógico para el DW (sin aplicar el algoritmo) puede obtener esquemas relacionales diferentes. Dado un esquema lógico cualquiera no se asegura que ese esquema pueda ser construido aplicando el algoritmo.

En particular, en esta tesis no se resuelven:

- Casos de diseño relacional muy fragmentado:

  En la fragmentación de dimensiones se tomó como unidad de fragmentación al nivel, un esquema lógico que divida los atributos correspondientes a un nivel en diferentes tablas (fragmentación vertical de los niveles) no puede ser construido mediante la aplicación del algoritmo.

  Incorporando lineamientos que permitan realizar operaciones de más bajo nivel pueden abarcarse más estilos de diseño.

  El estudio de nuevos lineamientos es una vía interesante para complementar al algoritmo. Esto podría implicar la definición de nuevas primitivas.

- Optimización de casos particulares:

  El algoritmo propuesto trata de resolver casos generales, por lo que muchas de las primitivas que apuntan a casos particulares no fueron utilizadas.

  Se puede estudiar la posibilidad de aplicar esas primitivas para casos especiales, lo que implica una caracterización de casos que serán resueltos de manera alternativa. Por ejemplo, sería interesante aplicar las primitivas Minidimension Break Off y Data Array Creation, sólo aplicables en casos muy especiales.

  La definición de casos implicará la definición de nuevos lineamientos, y probablemente de nuevas reglas de diseño.

En general la generación de un esquema lógico a partir de un esquema conceptual es un problema complejo de resolver, en el sentido de que no se puede abarcar todos los casos particulares, y es posible encontrar contraejemplos de esquemas lógicos que no puedan ser generados [Mar89]. El algoritmo propuesto en esta tesis intenta cubrir un subconjunto amplio de casos y es flexible para su extensión, ya sea definiendo nuevos pasos, reglas de diseño, lineamientos o primitivas.

3. Trabajo en Curso y Futuro

Actualmente se están implementando dos herramientas de asistencia al diseñador en la definición de lineamientos y mapeos respectivamente, como parte del trabajo de un taller de ingeniería de software [Per01], [Cal01], [Bar01]. Estas herramientas son editores independientes, en el sentido de que no interactúan con el resto de las herramientas del ambiente CASE. Se planea programar dicha interacción, de manera de incluir sus funcionalidades dentro de DwDesigner, brindando una interfaz gráfica para la primera etapa del diseño lógico.

Paralelamente se está trabajando en una interfaz web para DwDesigner, que permita diseñar el esquema lógico del DW a través de un navegador (browser). Actualmente se está desarrollando un prototipo que asiste en la definición de lineamientos y mapeos, realizando consultas a un servidor web y ejecutando *applets*, con el objetivo de extenderlo para que abarque todas las funcionalidades de DwDesigner (en una cuarta versión). Estas funcionalidades se están desarrollando en el marco de un taller de ingeniería de software [Per01], [Bit01].

Se está trabajando también en mejorar la persistencia de los objetos diseñados utilizando metadata (se está experimentando con CWM) en el marco de un proyecto de grado [Mur01].

En el contexto de tesis de maestrías del laboratorio CSI, se está estudiando la generación de programas de carga y actualización de DWs basándose en un modelo tipo workflow y paralelizando la ejecución [Lar01a], [Bou99], [Lab00]. El diseño de los programas de carga es un elemento importante en la etapa de diseño lógico, ya que no alcanza con modelar el esquema, hay que modelar también el comportamiento de las instancias. Una posibilidad interesante es la utilización de la traza de transformaciones como entrada para los programas de carga y actualización y adaptar la especificación del manejo de instancias de las primitivas. [Lar01], [Mar00].

En el contexto de una tesis de doctorado del laboratorio CSI, se está trabajando en definir una arquitectura que modele el flujo de información desde la Web hasta el DW. En dicho trabajo se estudia la evolución de los esquemas fuentes (páginas Web) y cómo propagar los cambios al DW [Mar01].

Como trabajo de investigación futuro se propone:

- Aplicar el algoritmo y las reglas de diseño en casos reales.

  Este trabajo comenzó con el estudio de casos de proyectos reales. En todos los casos se trata de proyectos finalizados donde el diseño del DW ya había sido realizado, y sólo en algunos se contaba con un esquema conceptual. Esos proyectos contribuyeron en la identificación de problemas, búsqueda de técnicas para solucionarlos, y en la elección de las primitivas más convenientes para llegar al esquema elegido por el diseñador del DW.

  Sería bueno experimentar con nuevos casos de estudio, ya no para obtener ideas, sino para evaluar la calidad del proceso y mejorarlo.

  Las primitivas de transformación de esquemas ya se han testeado en casos reales.

- Formalizar y demostrar propiedades.

  Un trabajo interesante sería caracterizar y razonar sobre los tipos de DWs construibles siguiendo el proceso de diseño de manera de probar propiedades sobre él.

  Una propiedad interesante a probar es que en el proceso de generación no se pierde información.

  Las instancias de la base fuente son equivalentes a las del esquema conceptual, esto lo manifiesta el diseñador al establecer los mapeos. Falta demostrar que en el proceso de transformación no se pierde información respecto a los sub-esquemas fuentes expresados en los mapeos.

  Otra propiedad interesante para demostrar es que el esquema resultante satisface las restricciones del problema.

- Extender el tratamiento del esquema conceptual para tener en cuenta restricciones de integridad.

  Actualmente se tienen en cuenta sólo algunas restricciones de integridad de las expresables en CMDM. Las restantes restricciones las tiene en mente el diseñador y las especifica como restricciones en las funciones de mapeo.

  Sería importante contemplar otros tipos de restricciones, y que las mismas se interpreten automáticamente durante la aplicación de las reglas. Como ejemplo concreto, los cubos definidos en CMDM pueden ser una primer idea para la definición de cubos a materializar. Esto ayuda en la definición de los lineamientos.

- Incorporación de información de carga y actualización en los lineamientos.

  La definición de lineamientos es el momento en el cual el diseñador indica su estrategia de diseño. Dicha estrategia tiene una influencia crucial para la posterior carga y actualización del DW.

  Se puede incorporar en los lineamientos información que facilite o elimine ambigüedades en el momento de programar dichos procesos. Por ejemplo, se puede incorporar nociones de frecuencia de actualización, orden y paralelización de las diferentes etapas y manejo de errores.

- Criterios de calidad en los esquemas.

  En esta tesis se mencionaron algunos criterios de calidad en la definición de los lineamientos, pero en última instancia se deja la elección al diseñador. Por ejemplo, se sugiere que no haya solapamiento de los fragmentos de una dimensión, o que las franjas de un cubo sean disjuntas, y que se implemente un cubo de cada relación dimensional con el máximo detalle.

  Se podría profundizar en la definición de criterios de calidad, y funciones de costo que actúen de cotas de cuándo conviene o no respetar esos criterios (performance vs. calidad).

- Estudio de la evolución.

  En [Mar00] se plantea la evolución de la base fuente. Esa evolución debe ser propagada a los mapeos si se quiere tener actualizada la información sobre la generación realizada.

  Pero hay otra forma de evolución y es la ocurrencia de cambios en los requerimientos, y por consiguiente en el esquema conceptual y en los lineamientos. Se podría estudiar una taxonomía de cambios para este contexto y ver como repercute en los mapeos y las transformaciones aplicadas.

También se proponen algunas mejoras para el prototipo:

- Interconectar la herramienta con el editor de CMDM.

  Actualmente la herramienta toma como entrada un esquema conceptual preprocesado, donde se tienen en cuenta la estructura de los objetos y sólo algunas de las restricciones. Las demás restricciones las tiene en mente el diseñador para reflejarlas en los mapeos.

  Sería muy útil incorporar un módulo de parsing del esquema conceptual (cuyas características están especificadas en el Anexo 2) para acceder directamente a un esquema conceptual.

  Para poder utilizar todas las restricciones expresables en CMDM es necesario que el parser interprete lógicas de alto orden. Sin embargo, un parser más modesto que busque determinadas restricciones, expresadas mediante macros fijas definidas para tales fines, ayudaría en gran medida en la definición de los mapeos. Para esto será necesario relevar el conjunto de restricciones básico a aplicar en CMDM.

Anexos

Anexo 1. Especificación e Instanciación de CMDM

1.1. Introducción

CMDM no restringe los tipos de datos de los niveles, sin embargo, a la hora de asignarle estructuras de datos, es necesario acotar dichos tipos. Utilizando las facilidades para instanciar el modelo se hace una restricción al tipo de datos de los niveles.

En la sección 1.2 se presenta la especificación original de CMDM, y en la sección 1.3 se introducen algunos cambios en la notación. En la sección 1.4 se realiza la instanciación de tipos y en la sección 1.5 se presenta la especificación resultante.

1.2. Especificación de CMDM

A continuación se presenta un fragmento de la especificación de CMDM (sección 4.5 la tesis [Car00]):

- DIMSCHC [23] ≡ { <NAME, DECLS, DIMS, RELS, CONSTS > /
  NAME ∈ STRINGS ∧
  DECLS ∈ DECLARATIONSC ∧
  DIMS ∈ SET(DIMENSIONSC(DECLS)) ∧
  RELS ∈ SET(RELATIONSC(DIMS)) ∧
  CONSTS ∈ SET(FORM) } [24] [25]
- DECLARATIONSC ≡ { <LEVEL_NAMES, TYPE, CONSTS > /
  LEVEL_NAMES ∈ SET(STRINGS) ∧
  TYPE ∈ LEVEL_NAMES → ETYPE ∧
  CONSTS ∈ LEVEL_NAMES → SET(FORM) }

[23] Se utiliza una C como sufijo para distinguir las definiciones originales.

[24] Set(A) representa al conjunto de los subconjuntos finitos del conjunto A

[25] FORM es el lenguaje de las Fórmulas para el lenguaje de Restricciones

- DIMENSIONSC (DECLS) ≡ { <DIMNAME, L, PO, CONSTS > /
  DIMNAME ∈ STRINGS ∧
  L ∈ SET(DECLS.LEVEL_NAMES) ∧
  PO ∈ POS(L) ∧
  CONSTS ∈ SET(FORM) }[26]
- RELATIONSC (DIMS) ≡ { <RELNAME, D, CONSTS > /
  RELNAME ∈ STRINGS ∧
  D ∈ SET(DIMS) ∧
  CONSTS ∈ SET(FORM) }

1.3. Cambios en la notación

Para uniformizar la notación utilizada en la especificación del modelo conceptual con la utilizada en la especificación de otros elementos, se proponen algunos cambios meramente sintácticos que facilitan la lectura de las expresiones.

Para tener mayor claridad en la lectura se cambian algunos nombres en la definición. Los criterios son:

- Nombres en singular para elementos. (por ejemplo: A).
- Nombres en plural para conjuntos de elementos. (por ejemplo: As).
- Nombres mnemotécnicos para constantes o funciones predefinidas. (por ejemplo se cambia Pos por ParcialOrders).

En la siguiente lista, las expresiones a la izquierda de los dos puntos (:) representan el contexto en el cual, las expresiones entre los dos puntos y la flecha (➔) se cambian por las expresiones a la derecha de la flecha.

- Dimensions: Pos ➔ ParcialOrders
- Dimensions: L ➔ Ls
- Relations: D ➔ Ds

En toda la especificación se cambian:

- A ∈ Set(B) ➔ A ⊆ B

1.4. Instanciación para tipos estructurados

Fundamentos

CMDM provee soporte para múltiples tipos de datos. (Sección 4.4.4.3 del documento [Car00]).

El modelo trabaja con los nombres de los niveles (identificadores) y una función que a partir de un nombre de nivel devuelve su tipo, permitiendo niveles de cualquier tipo. Ese tipos pertenecen a ETYPE, que representa el conjunto de las expresiones de tipo.

Instanciar el modelo para un tipo determinado significa indicar como construir ETYPE.

Para dar semántica a una expresión de tipo, se utiliza la función Etype2Set, que interpreta un tipo, o sea un elemento de ETYPE, y devuelve el conjunto de valores válidos para el tipo representado por la expresión.

[26] Pos(L) representa al conjunto de los órdenes parciales sobre el conjunto L.

A continuación se presenta la definición de ETYPE para soportar solamente el tipo conjunto de ítems (noción de record sin posición):

Sea ItemNames el conjunto de los nombres de ítems.

Sea BasicTypes el conjunto de los tipos básicos. Se considera inicialmente a los tipos *boolean, integer, float, string* y *date*. Este conjunto podría extenderse para incorporar nociones de tamaño y precisión, e inclusive incorporar nuevos tipos, por ejemplo sub-rangos.

- Boolean ∈ BasicTypes.
- Integer ∈ BasicTypes.
- Float ∈ BasicTypes.
- String ∈ BasicTypes.
- Date ∈ BasicTypes.

A partir de ItemNames y BasicTypes se define ETYPE.

Si:

- $b_1 \in$ BasicTypes, $b_2 \in$ BasicTypes, … $b_n \in$ BasicTypes, con $n \geq 1$
- $a_1 \in$ ItemNames, $a_2 \in$ ItemNames, … $a_n \in$ ItemNames, con $a_i \neq a_j$, $1 \leq i < j \leq n$.

Entonces: $\{<a_1, b_1>, <a_2, b_2>, \ldots <a_n, b_n>\} \in$ ETYPE.

Definido ETYPE se puede definir la función EType2Set:

Sea EType2SetB la función de interpretación para la expresiones de tipo BasicTypes.

Si:

- $b_1 \in$ BasicTypes, $b_2 \in$ BasicTypes, … $b_n \in$ BasicTypes, con $n \geq 1$
- $a_1 \in$ ItemNames, $a_2 \in$ ItemNames, … $a_n \in$ ItemNames, con $a_i \neq a_j$, $1 \leq i < j \leq n$.
- v_1= EType2SetB (b_1), v_2= EType2SetB (b_2), … v_n= EType2SetB (b_n), con $n \geq 1$.

Se define: EType2Set ($\{<a_1, b_1>, <a_2, b_2>, \ldots <a_n, b_n>\}$) = $\{a_1 : v_1, a_2 : v_2, \ldots a_n : v_n\}$

Definición de Ítems

Instanciado Etype se puede formalizar el concepto de ítem:

Si:

- $b_1 \in$ BasicTypes
- $a_1 \in$ ItemNames

Entonces: $<a_1, b_1> \in$ Items.

De acuerdo a esta definición se cumple: ETYPE = Set(Items).

De esta forma se puede trabajar con los ítems como componentes de los niveles.

Definición de Niveles

El conjunto DECLARATIONSC definido en [Car00] es útil para especificar CMDM, pero a la hora de instanciarlo, resulta más cómodo trabajar en forma explícita con los ítems de un nivel, y no hacerlo a través de las función TYPE (componente de DECLARATIONSC).

En la especificación de CMDM se sustituye el conjunto DECLARATIONSC por el conjunto LEVELS, por razones meramente operativas. Para ello se debe probar que se puede traducir un conjunto en otro sin pérdida de información. En particular, dado un elemento de DECLARATIONSC, se construir su correspondiente en LEVELS.

Se define el conjunto LEVELS de la siguiente forma:

- LEVELS ≡ { <LEVELNAME, IS, LCONSTS > /
  LEVELNAME ∈ STRINGS ∧
  IS ∈ ETYPE ∧
  LCONSTS ⊆ FORM }

Se definen 2 funciones que traducen de DECLS ∈ DECLARATIONSC a LEVS ⊆ LEVELS.

DECLS2LEVS toma un elemento de DECLARATIONSC y devuelve un subconjunto de LEVELS.

DECLS2LEVS : DECLARATIONSC → SET (LEVELS)

DECLS2LEVS (DECLS) = { <LEVELNAME, IS, LCONSTS> /
LEVELNAME ∈ DECLS.LEVEL_NAMES ∧
IS = DECLS.TYPE (LEVELNAME),
LCONST = DECLS.CONSTS (LEVELNAME) > }

LEVS2DECLS toma un subconjunto LEVELS y devuelve un elemento de DECLARATIONSC.

LEVS2DECLS : DECLARATIONSC → SET (LEVELS)

LEVS2DECLS (LEVS) = <LEVEL_NAMES, TYPE, CONSTS> /
LEVEL_NAMES = {L.LEVELNAME / L∈ LEVS} ∧
TYPE ∈ LEVEL_NAMES → ETYPE ∧
∀ L ∈ LEVS . TYPE(L.LEVELNAME)= L.IS ∧
CONSTS ∈ LEVEL_NAMES → SET(FORM) ∧
∀ L ∈ LEVS . CONSTS(L.LEVELNAME)= L.LCONSTS

A continuación se prueba que estas funciones conservan la información. Para ello se muestra que dado un elemento DECLS ∈ DECLARATIONSC, si se aplica DECLS2LEVS y luego LEVS2DECLS se obtiene el elemento original.

Lema:

Dado D ∈ DECLARATIONSC se cumple que LEVS2DECLS(DECLS2LEVS(D))=D.

Demostración:

Sean Ls ⊆ LEVELS, LS = DECLS2LEVS (D)

E ∈ DECLARATIONSC, E = LEVS2DECLS (Ls)

Se debe probar que D=E.

Aplicando la definición de LEVS2DECLS, se tiene que:

E = LEVS2DECLS (LS) = <LEVEL_NAMES, TYPE, CONSTS> /
LEVEL_NAMES = {L.LEVELNAME / L∈ LS} ∧
TYPE ∈ LEVEL_NAMES → ETYPE ∧
∀ L ∈ LS . TYPE(L.LEVELNAME)= L.IS ∧
CONSTS ∈ LEVEL_NAMES → SET(FORM) ∧
∀ L ∈ LS . CONSTS(L.LEVELNAME)= L.LCONSTS

Esto es:

- E.LEVEL_NAMES = {L.LEVELNAME / L∈ LS} (i)
- E.TYPE cumple ∀ L ∈ LS . E.TYPE(L.LEVELNAME)= L.IS (ii)
- E.CONSTS cumple ∀ L ∈ LS . E.CONSTS(L.LEVELNAME)= L.LCONSTS (iii)

Aplicando la definición de DECLS2LEVS, se tiene que:

LS = DECLS2LEVS (D) = { <LEVELNAME, IS, LCONSTS> /
LEVELNAME ∈ D.LEVEL_NAMES ∧
IS = D.TYPE (LEVELNAME),
LCONST = D.CONSTS (LEVELNAME) > }

Esto es:

- {L.LEVELNAME / L∈ LS} = D.LEVEL_NAMES (iv)
- ∀ L ∈ LS . L.IS = D.TYPE (L.LEVELNAME) (v)
- ∀ L ∈ LS . L.LCONSTS = D.CONSTS (L.LEVELNAME) (vi)

Por lo tanto:

E.LEVEL_NAMES = {L.LEVELNAME / L∈ LS} por (i)
= D.LEVEL_NAMES por (iv)

E.TYPE cumple ∀ L ∈ LS . E.TYPE(L.LEVELNAME)= L.IS por (ii)
y cumple ∀ L ∈ LS . D.TYPE(L.LEVELNAME)= L.IS por (v)
por tanto E.TYPE = D.TYPE

E.CONSTS cumple ∀ L ∈ LS . E.CONSTS(L.LEVELNAME)= L.LCONSTS por (iii)
y cumple ∀ L ∈ LS . D.CONSTS(L.LEVELNAME)= L.LCONSTS por (vi)
por tanto E.CONSTS = D.CONSTS

Y por igualdad de campos, se deduce que E = D. ÿ

Definición de Dimensiones

En forma análoga se define el conjunto DIMENSIONS, el cual tiene una componente de tipo LEVELS en lugar de nombres de niveles. Dicho conjunto sustituye al conjunto DIMENSIONSC.

Las funciones DIMSC2DIMS y DIMS2DIMSC realizan la transformación entre los conjuntos. La demostración de isomorfismo es análoga.

Dado LEVS ⊆ LEVELS se define el conjunto DIMENSIONS de la siguiente forma:

- DIMENSIONS (LEVS) ≡ { <DIMNAME, LS, PO, DCONSTS > /
DIMNAME ∈ STRINGS ∧
LS ⊆ LEVS ∧
PO ∈ PARTIALORDERS(L) ∧
DCONSTS ⊆ FORM }

Definición de Relaciones

De forma análoga, se define el conjunto RELATIONS, el cual tiene una componente de tipo DIMENSIONS en lugar de DIMENSIONSC. Dicho conjunto sustituye al conjunto RELATIONSC.

Las funciones RELSC2RELS y RELS2RELSC realizan la transformación entre los conjuntos. La demostración de isomorfismo es directa.

Dado DIMS ⊆ DIMENSIONS se define el conjunto RELATIONS de la siguiente forma:

- RELATIONS (DIMS) ≡ { <RELNAME, DS, RCONSTS > /
  RELNAME ∈ STRINGS ∧
  DS ⊆ DIMS ∧
  RCONSTS ⊆ FORM }

1.5. Especificación resultante

Dado DS= < NAME, DECLS, DIMS, RELS, CONSTS > ∈ DIMSCHC se construye CS= <NAME, LEVS, DIMS, RELS, CONSTS > ∈ CONCEPTUALSCHEMA donde:

- CS.NAME= DS.NAME
- CS.LEVS= DECLS2LEVS(DS.DECLS)
- CS.DIMS= DIMSC2DIMS(DS.DIMS)
- CS.RELS= RELSC2RELS(DS.RELS)
- CS.CONSTS= DS.CONSTS

A continuación se presenta la especificación del esquema conceptual, construido a partir del esquema CMDM.

- CONCEPTUALSCHEMAS ≡ { <NAME, LEVS, DIMS, RELS, CONSTS > /
  NAME ∈ STRINGS ∧
  LEVS ∈ LEVELS ∧
  DIMS ⊆ DIMENSIONS(LEVS) ∧
  RELS ⊆ RELATIONS(DIMS) ∧
  CONSTS ⊆ FORM }
- LEVELS ≡ { <LEVELNAME, IS, LCONSTS > /
  LEVELNAME ∈ STRINGS ∧
  IS ⊆ ITEMS ∧
  LCONSTS ⊆ FORM }
- DIMENSIONS (LEVS) ≡ { <DIMNAME, LS, PO, DCONSTS > /
  DIMNAME ∈ STRINGS ∧
  LS ⊆ LEVS ∧
  PO ∈ PARTIALORDERS(L) ∧
  DCONSTS ⊆ FORM }
- RELATIONS (DIMS) ≡ { <RELNAME, DS, RCONSTS > /
  RELNAME ∈ STRINGS ∧
  DS ⊆ DIMS ∧
  RCONSTS ⊆ FORM }

1.6. Conclusiones

En este anexo se presentó una restricción a los tipos de datos de los niveles de CMDM a través de una instanciación del modelo.

La restricción de tipos permite niveles de tipo conjunto de ítems, siendo un ítem una pareja: <nombre, tipo_simple>.

Se definió una estructura de datos alternativa para niveles (Levels), y funciones de transformación desde y hacia la estructura original (Declarations). Se demostró que las funciones de transformación preservan la información. Análogamente se definieron estructuras alternativas para dimensiones y relaciones a partir de niveles.

Dichas definiciones permiten re-definir un esquema conceptual en términos de ítems y manipularlo en forma más cómoda.

Anexo 2. Parsing del Esquema Conceptual

2.1. Introducción

En el Anexo 1, en la sección 1.2 se presenta la especificación de CMDM.

Dado un esquema especificado en CMDM, es posible que el diseñador tenga que completarlo o modificarlo mínimamente para cumplir con algunas restricciones necesarias para generar un esquema relacional a partir de él. Estas restricciones se detallan en la sección 4 del capítulo 3, y son:

- Los niveles están formados por un conjunto de ítems. El diseñador debe cambiar la función *type* del componente *Decls* del esquema, de manera que asocie un conjunto de ítems a cada nivel.
- Un ítem pertenece a un único nivel. Eventualmente se tienen que renombrar los ítems de manera que cada uno esté en un único nivel.
- Un nivel pertenece a una única dimensión. Eventualmente se deben renombrar los niveles y los ítems que los conforman para que cada nivel esté en una única dimensión.
- Todos los niveles tienen una clave, ya sea relativa o absoluta. El diseñador debe agregar restricciones de claves a los niveles que no las tienen.

A partir del esquema con las modificaciones que agregue el diseñador, se construye el esquema ConSch de la Definición 1. La construcción se explica en la sección 1.5 del Anexo 1.

A partir del esquema ConSch y sus restricciones pueden construirse algunas funciones que faciliten la manipulación del modelo, por ejemplo una función que dado un nivel nos devuelva su clave relativa.

Para construir estas funciones es necesario recorrer el esquema interpretando el lenguaje de restricciones. La forma en que se implemente depende de las macros que se utilicen para definir las restricciones. Lo más razonable es trabajar con un conjunto finito de macros. En este trabajo no se pretende resolver dicho parsing.

A continuación se definen algunas funciones que permiten manipular más cómodamente al esquema.

2.2. Componentes

Cada ítem pertenece a un único nivel y cada nivel a una única dimensión. Por lo tanto puede saberse a qué objetos pertenecen a partir del esquema conceptual.

Se define una función que dado un nivel devuelve la dimensión a la que pertenece.

- LDIM : SCHLEVELS → SCHDIMENSIONS

  LDIM(L) = D / D ∈ SCHDIMENSIONS ∧ L ∈ D.LS

Definición 14 – LDim

Análogamente, se define una función que dado un ítem devuelve el nivel al que pertenece.

- ILEV : SCHITEMS → SCHLEVELS

  ILEV(I) = I / D ∈ SCHLEVELS ∧ I ∈ L.IS

Definición 15 – ILev

2.3. Orden de Niveles

Los niveles de una dimensión deben cumplir ser un orden parcial, conformando jerarquías. No sólo es necesario saber qué niveles están por encima de otros en una jerarquía, sino que se necesita saber qué niveles son los inmediatamente superiores (padres), o inmediatamente inferiores (hijos), o quienes están en la cima (no tienen padre) o por debajo de todo (no tienen hijos). Todas estas relaciones pueden obtenerse a partir de los órdenes parciales de las dimensiones del esquema.

Se define una función que dado un nivel devuelve los niveles que están inmediatamente encima en la jerarquía de la dimensión, es decir, los niveles padres.

- LEVELFATHERS : SCHLEVELS → SET (SCHLEVELS)

  LEVELFATHERS(L) = { P / P ∈ LDIM(L).LS ∧ !∃M∈ LDIM(L).LS . (<P,M> ∈ LDIM(L).PO ∧ <M,L> ∈ LDIM(L).PO) }

Definición 16 – LevelFathers

Análogamente se define una función que dado un nivel devuelve los niveles que están inmediatamente debajo en la jerarquía de la dimensión, es decir, los niveles hijos.

- LEVELSONS : SCHLEVELS → SET (SCHLEVELS)

  LEVELSONS(L) = { H / H ∈ LDIM(L).LS ∧ L ∈ LEVELFATHERS(H) }

Definición 17 – LevelSons

Utilizando las funciones anteriores se define una función que dada una dimensión devuelve los niveles que están por encima en las jerarquías, es decir, los niveles que no tienen padres.

- TOPLEVELS : SCHDIMESIONS → SET (SCHLEVELS)

  TOPLEVELS(D) = { L / L ∈ D.LS ∧ !∃P∈ D.LS . (<P,L> ∈ D.PO) }

Definición 18 – TopLevels

Análogamente se define una función que dada una dimensión devuelve los niveles que están por debajo en las jerarquías, es decir, los niveles que no tienen hijos.

- BOTTOMLEVELS : SCHDIMESIONS → SET (SCHLEVELS)

  BOTTOMLEVELS(D) = { L / L ∈ D.LS ∧ !∃P∈ D.LS . (<L,P> ∈ D.PO) }

Definición 19 – BottomLevels

2.4. Claves de Niveles

Para poder identificar los diferentes elementos de un nivel se exige que *todo nivel tenga una clave*, ya sea absoluta o relativa.

Se consideran las funciones RELATIVEKEY y ABSOLUTEKEY que dado un nivel nos devuelven los ítems que lo identifican, relativa o absolutamente. Estas funciones pueden definirse por extensión parseando las restricciones de los niveles del esquema.

- RELATIVEKEY : SCHLEVELS → SET (SCHITEMS)
- ABSOLUTEKEY : SCHLEVELS → SET (SCHITEMS)

Notar que no son funciones totales ya que algunos niveles pueden no tener clave absoluta y otros no tener clave relativa. Pero dado un nivel, alguna de las dos funciones estará definida para él ya que se exige que todo nivel tenga una clave.

Para los niveles que no tienen clave absoluta se puede construir una incorporando a la clave relativa los ítems que conforman la clave absoluta de los niveles padres.

Para los niveles que están en la cima de una jerarquía (no tienen padre) la clave relativa es también absoluta.

Se puede entonces definir inductivamente una función que calcule una clave absoluta para cada nivel.

- LEVELKEY : SCHLEVELS → SET (SCHITEMS)

  if ABSOLUTEKEY(L) ≠ ⊥ **then** LEVELKEY (L) = ABSOLUTEKEY(L)
  else if L ∈ TOPLEVELS(LDIM(L)) **then** LEVELKEY (L) = RELATIVEKEY(L)
  else LEVELKEY (L) = RELATIVEKEY(L) ∪ {ABSOLUTEKEY(P) / P ∈ LEVELFATHERS(L)}

Definición 20 – LevelKey

2.5. Conclusiones

En este anexo se describió como completar el esquema conceptual para que sea posible generar un esquema relacional a partir de él. Para ello se necesita restringir los tipos de los niveles, duplicar elementos compartidos y definir claves para todos los niveles.

También se definieron algunas funciones útiles para manipular el esquema.

Anexo 3. Vinculación de Tablas: Links y Alias

3.1. Introducción

Para la construcción del modelo lógico del DW es necesario aplicar transformaciones que utilizan atributos de varias tablas, realizando el join de dichas tablas. Por qué atributos realizar el join y que condiciones se deben imponer es algo que depende exclusivamente de la fuente.

La decisión de la condición de join a utilizar puede tomarse en el momento de aplicar cada transformación, pero esto implica la intervención continua del diseñador. Si se quiere automatizar la aplicación de las transformaciones, la vinculación de las tablas debe hacerse junto con la definición de la base fuente.

Se debe indicar por qué atributos aplicar el join, eventualmente imponiendo condiciones. A nivel conceptual, estos atributos representan relaciones entre entidades del esquema conceptual de la fuente (por ejemplo un esquema E/R). A nivel lógico, generalmente están determinados por las claves foráneas, o nombres iguales en atributos. En todos los casos depende de la forma en que se diseñe la base fuente y de las técnicas utilizadas por el diseñador de las mismas.

Para la construcción de una primer versión de la vinculación entre las tablas pueden usarse las claves foráneas, y luego refinar ese resultado.

3.2. Links

Se llama link a una condición de join para dos tablas. Dadas dos tablas, la forma de vincularlas es a través de un link, y es lo que se usará para aplicar transformaciones a la fuente.

Intuitivamente, un link corresponde a la cláusula where de una sentencia SQL que relaciona dos tablas. Un link se define como un predicado a partir de los atributos de ambas tablas.

Se utiliza una función que dadas dos tablas nos devuelve el link entre ellas, si lo hay. En el Capítulo III - sección 7, se la definió de la siguiente manera:

- SCHLINKS $\in \{T_1 / T_1 \in$ SCHTABLES$\}$ X $\{T_2 / T_2 \in$ SCHTABLES$\} \rightarrow$ PREDICATES $(T_1 \bullet$AS $\cup$ $T_2 \bullet$AS$)$

En la Figura 50 se muestra un link entre las tablas *Clientes* y *Facturas*, a través de los atributos *id-cliente* y *cliente* respectivamente (operación de igualdad). El vínculo se dedujo ya que *Facturas.cliente* es clave foránea de *Clientes.id-cliente*.

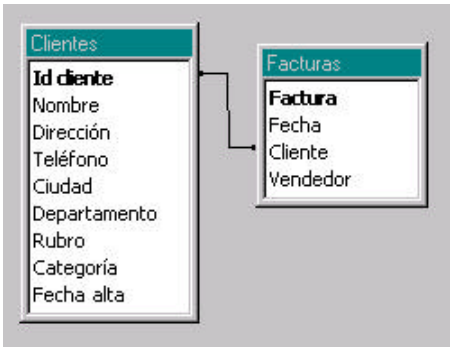

Figura 50 – Link entre dos tablas

El link entre ambas tablas se expresa con el siguiente predicado:

Facturas.cliente = Clientes.id-cliente.

Una consulta a los atributos de dichas tablas, utilizando el link tendría la forma:

```
SELECT   *
  FROM   Facturas, Clientes
 WHERE   Facturas.cliente = Clientes.Id-cliente
```

3.3. Propiedades de los Links

El conjunto de links entre las tablas de la base de datos fuente determina un grafo, cuyos nodos son las tablas, y los arcos son los atributos.

Es deseable que ese grafo cumpla ciertas propiedades:

1- El grafo es conexo:

Si el grafo no es conexo, habrá tablas a las que no se puede realizar join, a menos de un producto cartesiano entre las mismas.

2- El grafo no contiene ciclos:

Si el grafo contiene ciclos, para construir una consulta puede haber más de un camino de joins.

En la Figura 51, para realizar una consulta con atributos de las tablas *Registros-Facturas* y *Productos* se debe incluir también a la tabla *Artículos* o a la tabla *Articulos V*. Cuando el grafo tiene ciclos, una herramienta automática no sabría que links utilizar para construir la consulta.

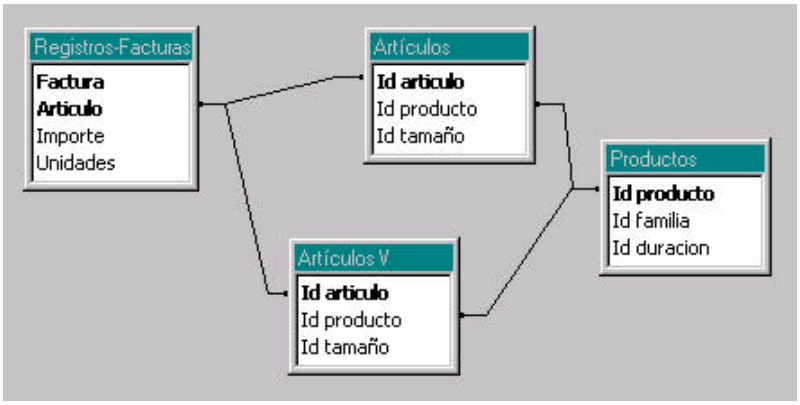

Figura 51 – Problema de ciclos en el grafo de links

3- Dos tablas tienen un único link:

(aunque la expresión de vinculación involucre a varios atributos)

La condición de join puede contener cálculos e involucrar a varios atributos de ambas tablas, pero esa condición debe ser evaluada como un todo, es decir, todas las sub-condiciones deben coincidir.

Lo que no se quiere que ocurra, es tener varias expresiones, y que a veces se cumpla una, y otras veces otra.

En la Figura 52 se muestra una doble vinculación entre las tablas *Códigos* y *Artículos*. La tabla códigos brinda las descripciones para los identificadores de artículos (atributo tipo vale ART) y para los identificadores de tamaños (atributo tipo vale TAM).

En un buen diseño relacional, la tabla *Códigos* debería sustituirse por dos o más tablas, cada una con las descripciones de un objeto conceptual (artículo, producto, tamaño, etc.).

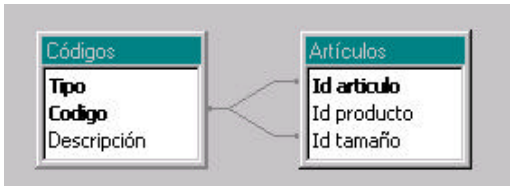

Figura 52 – Problema de múltiples links entre dos tablas

Para hacer una consulta que involucre las dos tablas, no se quiere imponer las dos condiciones, porque es probable que de vacía o muy pocos valores. Si se necesitan las dos descripciones en una misma consulta entonces se realiza el join con dos instancias de la tabla códigos. Por ejemplo:

```
SELECT  id_articulo, T2.descripcion as descripcion_articulo, id_tamaño, T3.descripcion as desc_tamaño
  FROM  articulos T1, codigos T2, codigos T3
 WHERE  T1.id_articulo = T2.codigo
   AND  T2.tipo = 'ART'
   AND  T1.id_tamaño = T3.id_tamaño
   AND  T3.tipo = 'TAM'
```

La solución a este problema consiste en definir dos vistas de la misma tabla, y asignar un link a cada una de ellas. A esas vistas se les llama Alias.

La Figura 53.muestra la utilización de dos alias para la tabla *Códigos*: *Codigos-I* y *Codigos-II*.

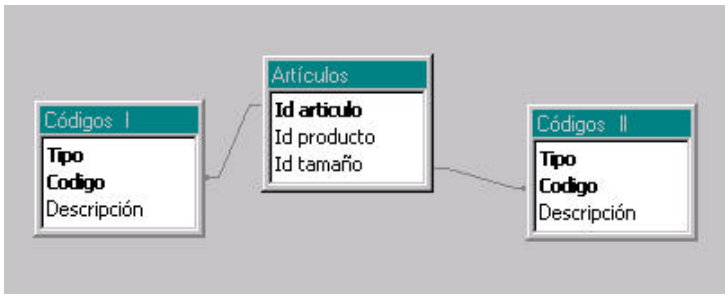

Figura 53 – Solución del doble vínculo entre tablas

La definición de alias no siempre es la mejor solución, se necesitará la opinión del diseñador para tomar este tipo de decisiones.

Si no se cumplen estas propiedades se encontrarán ambigüedades durante la construcción del esquema lógico del DW.

3.4. Conclusiones

En este anexo se presentó una forma de vincular dos tablas por medio de una condición de join entre las mismas (link).

Los links entre las tablas de la base fuente determinan un grafo que debe cumplir: ser conexo, sin ciclos y mono-arista. Cuando es necesario tener varios links entre dos tablas, pueden definirse diferentes vistas de una de las tablas (alias) y asignar un link para cada una de ellas.

Tanto la definición de links como de alias es una tarea totalmente manual que debe realizar el diseñador según su conocimiento de la fuente. Las claves foráneas y la repetición de nombres en los atributos son un buen punto de partida.

Anexo 4. Funciones auxiliares

4.1. Introducción

En este anexo se presenta la definición de algunas funciones utilizadas en la especificación.

4.2. Ítems del esquema intermedio

Los objetos del esquema intermedio están formados, directa o indirectamente, por ítems. Se define una función para obtener los ítems que componen a los objetos. Esta función se utiliza en las siguientes secciones:

- OBJECTPARTITEMS : SCHOBJECTS → SCHITEMS

  Para cada caso la función se define como:

  - Dado I ∈ SCHITEMS, OBJECTPARTITEMS (I) = {I}
  - Dado L ∈ SCHLEVELS, OBJECTPARTITEMS (L) = L.IS
  - Dado D ∈ SCHDIMENSIONS, OBJECTPARTÍTEMS (D) = D.LS♦IS
  - Dado F ∈ SCHFRAGMENTS, OBJECTPARTÍTEMS (F) = F♦IS
  - Dado C ∈ SCHCUBES, OBJECTPARTÍTEMS (C) = C.LS♦IS

Definición 21 – ObjectPartItems

Para identificar en forma única la instancia de un objeto hacen falta algunos ítems del objeto y en algunos casos ítems de otros objetos. Por ejemplo, la clave de un nivel puede ser débil respecto a niveles superiores por lo que para identificarlo hacen falta ítems de esos niveles superiores.

Se define una función para obtener los ítems que componen la clave de los objetos. Esta función se utiliza en las siguientes secciones:

- ObjectKeyItems : SchObjects → SchItems

  Para cada caso la función se define como:

  - Dado I ∈ SchItems, ObjectKeyItems (I) = {I}
  - Dado L ∈ SchLevels, ObjectKeyItems (L) = LevelKey(L)
  - Dado D ∈ SchDimensions, ObjectKeyItems(D)=BottomLevels(D)♦LevelKey[27]
  - Dado F ∈ SchFragments, ObjectKeyItems (F) = BottomLevels(F)♦LevelKey
  - Dado C ∈ SchCubes, ObjectKeyItems (C) = C.Ls♦LevelKey

Definición 22 – ObjectKeyItems

Se define una función para obtener los ítems relevantes del objeto, según el caso los que lo componen o que lo identifican. Los niveles y fragmentos necesitan además referenciar a los niveles inmediatamente superiores en las jerarquías de la dimensión (padres) para mantener la asociación y permitir navegar por dichas jerarquías.

- ObjectItems : SchObjects → SchItems

  Para cada caso la función se define como:

  - Dado I ∈ SchItems, ObjectItems (I) = ObjectPartItems(I)
  - Dado L ∈ SchLevels, ObjectItems (L) = ObjectPartItems(L) ∪ ObjectKeyItems(L) ∪ FatherLevels(L)♦LevelKey[28]
  - Dado D ∈ SchDimensions, ObjectItems (D)= ObjectPartItems(L)
  - Dado F ∈ SchFragments, ObjectItems (F) = ObjectPartItems(L) ∪ ObjectKeyItems(F) ∪ F♦FatherLevels♦LevelKey
  - Dado C ∈ SchCubes, ObjectItems (C) = ObjectKeyItems(C)

Definición 23 – ObjectItems

4.3. Funciones de actualización de tablas

La ejecución de una regla tiene como resultado la aplicación de primitivas. Esto implica que se generan una o más tablas que se agregan al esquema lógico. Estas tablas pueden ser tablas intermedias o temporales, a las que se aplicarán más transformaciones, o pueden ser tablas definitivas para el DW.

También deben definirse links entre las nuevas tablas y las tablas existentes. Las tablas generadas son transformaciones de las tablas de entrada a la regla, por lo que es común que se relacionen con éstas por las claves y que hereden los links a otras tablas.

[27] La función BottomLevels devuelve el conjunto de niveles que están por debajo en las jerarquías de la dimensión. Está descrita en el anexo 2.

[28] La función FatherLevels devuelve el conjunto de niveles que son padres en las jerarquías de la dimensión de alguno de los niveles dados. Está descrita en el anexo 2.

Por ejemplo: se tiene una tabla *clientes* con clave primaria *código-cliente*, y se aplica la primitiva P6 – DD-Adding para agregar el atributo calculado *edad*, generando la tabla *clientes2*. Las tablas *clientes* y *clientes2* tienen la misma clave y tienen un link por ella. Además la tabla *clientes2* tendrá links a las mismas tablas que *clientes* y con los mismos predicados.

Se define un procedimiento que se encarga de mantener actualizado el esquema lógico.

- UPDATETABS : SCHTABLES X SET(SCHTABLES)

  UPDATETABS (N, TS)

  - Agregar N a SCHTABLES
  - ∀T ∈ TS se define: SCHLINKS (N,T) = EQUAL (T•(T.PK), N•(T.PK)
  - ∀T ∈ TS, ∀U ∈ SCHTABLES se define: SCHLINKS (N,U) = REPLACETABLE(SCHLINK(T,U),T,N)

Definición 24 – UpdateTabs

El procedimiento UpdateTabs agrega la tabla (primer parámetro) al conjunto de tablas del esquema lógico. Luego establece links por claves primarias a las tablas del conjunto dado, y copia los links de éstas con otras tablas.

4.4. Funciones de actualización de mapeos

Durante de la aplicación de la regla se define una nueva función de mapeo para el objeto. Se deben actualizar los mapeos a fragmentos o cubos (el que corresponda) para incorporar la nueva función.

Se define un procedimiento que se encarga de mantener actualizados los mapeos de cubos y fragmentos.

- UPDATEMAPS : MAPPINGS(ITS) X SCHOBJECTS con ITS ⊆ SCHITEMS

  UPDATEMAPS (T, OS)

  - **Si** OS ∈ SCHFRAGMENTS **entonces** SCHFMAPPINGS(OS).MAP = F

    Else SCHCMAPPINGS(OS).MAP = F

Definición 25 – UpdateMaps

Anexo 5. Descripción de las Primitivas

5.1. Introducción

En este anexo se presenta una descripción de las primitivas de transformación de esquemas propuestas por Marotta en [Mar00] y se proponen nuevas primitivas para realizar otras operaciones.

Las primitivas de [Mar00] fueron pensadas para utilizarse interactivamente de manera de dar varias opciones para resolver un mismo problema, lo cual es beneficioso en muchos casos particulares. Sin embargo, al intentar generalizar su aplicación, muchas de ellas no se utilizan ya que resuelven casos muy particulares, y otras si bien pueden aplicarse no resulta lo más natural.

Por ejemplo, se puede querer agregar un atributo de una tabla a otra como redundancia. En la Figura 54 se presentan 3 tablas: *MaestroClientes*, *Ciudades* y *Departamentos*. El diseñador quiere agregar el atributo *Zona* de la tabla *Departamentos* a la tabla *MaestroClientes*. La forma de hacerlo es como un atributo calculado a través de la primitiva DD-Adding N-1. Sin embargo, esto no se puede realizar directamente ya que las tablas no tienen atributos en común.

Para este caso en particular, el diseñador decide agregar primero el atributo *Zona* a la tabla *Ciudades*, generando una tabla *CiudadesAux* y luego agregarlo de esta a la tabla *MaestroClientes*. Si bien esto resuelve el problema correctamente es muy difícil generalizar esta situación, ya que en muchos casos habrá muchas tablas involucradas, con condiciones de join más complejas, y se querrán cálculos complejos.

La estrategia propuesta para este caso es realizar el join de todas las tablas involucradas, y luego eliminar los atributos que sobren. Para eso se necesita una primitiva que pueda realizar el join de dos tablas aplicando cualquier condición de join.

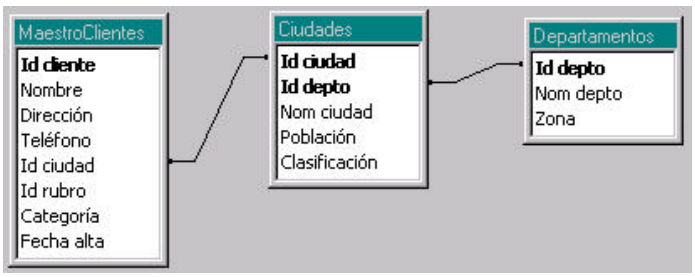

Figura 54 – Ejemplo de automatización

Si bien las primitivas pueden aplicarse para resolver muchos problemas de diseño de DW, para resolver problemas generales se necesita complementarlas con nuevas primitivas.

Se propone un conjunto de primitivas que complementa al original con nuevas transformaciones. Para diferenciarlas de las propuestas por Marotta se utiliza la letra Q en lugar de P en la numeración (primitivas Q1 a Q4).

En la sección 5.2 se presenta la descripción del conjunto original de primitivas tomada de [Mar00], y en la sección 5.3 se describen las nuevas primitivas. En la sección 5.4 se presenta una especificación de las nuevas primitivas siguiendo la notación que utiliza Marotta, y en la sección 5.5 se concluye.

Para continuar con la propuesta de Marotta tanto las descripciones como la especificación de las primitivas está escrita en inglés.

5.2. Descripción de las primitivas propuestas por Marotta

Primitive P1. IDENTITY

This primitive is useful when we want to generate in the DW, a relation that is exactly the same as another one. The original relation may be one existing in the source database or one that is an intermediate result (the result of the application of a primitive).

It gives as result, a copy of the relation given as input.

Primitive P2. DATA FILTER

In operational databases, there are some attributes that are of interest for the DW system, but there are some others that correspond to data that is purely operational and that is not useful for the kind of analysis that is made with the DW.

The goal of this primitive is to preserve only the useful attributes, removing the other ones.

Primitive P3. TEMPORALIZATION

Many relations in operational systems do not maintain a temporal notion. For example, stock relations use to have the current stock data, updating it with each product movement.

In DWs, many relations need to include a temporal element, so that they can maintain historical information.

This primitive adds an element of time to the set of attributes of a relation.

Primitive P4. KEY GENERALIZATION

The real world subjects represented in dimensions usually evolve through time. For example, a client may change his address, a product may change its description or package_size.

In some cases it is enough to maintain only the last value, but in other ones it is necessary to store all versions of the element, so that history is maintained.

The goal of this group of primitives is to generalize the primary key of a dimension relation, so that more than one tuple of each subject represented in the relation, can be stored.

Two alternatives are provided to do this generalization, through the primitives: Version Digits and Key Extension.

Primitive P4.1. VERSION DIGITS

To generalize the key, version digits are added to each value of the attribute.

Primitive P4.2. KEY EXTENSION

The key is extended; new attributes of the relation are included in it.

Primitive P5. FOREIGN KEY UPDATE

When the key of a relation is changed, it is necessary to make the same changes in all the foreign keys that reference to it from other relations. For example, if an attribute is added to a key, it must be added also to the foreign keys of the referencing relations.

This primitive is useful for updating a foreign key in a relation when its corresponding primary key is modified.

Primitive P6. DD-ADDING

In production systems, usually, data is calculated from other data at the moment of the queries, in spite of the complexity of some calculation functions, in order to prevent any kind of redundancy.

For example, the product prices expressed in dollars are calculated from the product prices expressed in some other currency and a table containing the dollar values.

In a DW system, sometimes it is convenient to maintain these kind of data calculated, for performance reasons.

The primitives of this group add an attribute that is derived from others to a relation. They never cause changes to the grain of the relation.

Primitive P6.1. DD-ADDING 1-1

In this case, the calculations are made over only one relation and one tuple. For example, the total import of a sale is calculated from the quantity sold and the unit-price, which are all in the same relation.

Primitive P6.2. DD-ADDING N-1

In this case two relations are used. A calculated attribute is added to one of the relations. This attribute is derived from some attributes from the same relation and others from the other relation. This is the case of the example mentioned for the group of primitives (product prices).

The calculation function works over only one tuple of the relations. This tuple must be obtained uniquely through a join of the two relations.

Primitive P6.3. DD-ADDING N-N

This is the more complex case. Two relations and n tuples are used for the attribute calculation. Consider the following example in a bank. There exists a relation with client data and another relation with account data. If we want to add to the former the total amount of all the accounts for each client, the amounts contained in the second relation must be summed for each client.

The calculation function works over a set of tuples of one of the relations. These tuples must be obtained through a join of the two relations.

Primitive P7. ATTRIBUTE ADDING

The real world subjects represented in dimensions usually evolve through time. For example, a client may change his address, a product may change its description or package_size.

Sometimes it is required to maintain the history of these changes in the DW. In some cases, only a fixed number of values of certain attribute should be stored. For example, it could be useful to maintain the current value of an attribute and the last one before it, or the current value of an attribute and the original one.

In these cases, empty attributes are reserved in a dimension relation, for future changes. Suppose, for instance, that when a client changes his address we want to store the new and the old addresses. With this primitive an attribute is added to the relation, initially with a null value, to be filled in case the client moves out.

Primitive P8. HIERARCHY ROLL UP

In operational databases the information in the relations are stored at the highest level of detail that is possible. For example, the measure relations use to have all the movements. Usually, in these relations there is an attribute that has a hierarchy associated.

Often, when these relations are used in a DW, they are summarized by an attribute following some hierarchy (doing a "roll-up"), for example, if data is in a daily level and monthly totals are required. In this case we are doing a roll-up in a hierarchy of time.

This primitive does the roll up by one of the attributes of a relation following a hierarchy. Besides, it can generate another hierarchy relation with the corresponding level of detail.

Primitive P9. AGGREGATE GENERATION

In operational systems data is managed as crossings of many dimensions. In general, many DW relations are constructed from these crossings, and data is grouped by some of the dimensions. Other dimensions are removed as a consequence of this information grouping.

For example, for a salary system, may be of great importance which employee has made certain sale. However, for analyzing the sales at a global level in the DW, it is required resumed data and not that information in particular.

This primitive removes a set of attributes from a measure relation, summarizing the measures. This operation has the effect of decreasing the number of tuples of the relation.

Primitive P10. DATA ARRAY CREATION

In a relation where measures are maintained on a month-by-month basis, it can be useful, instead of having an attribute for the month and another one for the measure, to have 12 attributes for the measures of the 12 months respectively. With this structure comparative reports can be done more easily and with better performance, since annual totals are calculated at a tuple level. Besides, the number of tuples decreases.

This multiple attributes schema (data array) is useful not only for months, in fact it can be used for any attribute whose associated set of values is finite and known (so that an attribute can be assigned to each value).

Given a relation that contains an attribute that represents a pre-determined set of values, this primitive generates a relation with a data array structure.

Primitive P11. PARTITION BY STABILITY

In some cases it is recommended to partition a relation, distributing its data into different relations. This can be useful, for example, for maintaining the most recent data more accessible than the rest of the data. It also allows organizing data according to its propensity for change.

These primitives partition a relation, in order to organize its data storage. The first (Vertical Partition) or the second primitive (Horizontal Partition) of this family can be applied, depending on the design criterion used.

Primitive P11.1. VERTICAL PARTITION

This primitive applies a vertical partition to a dimension relation, giving several relations as result. It distributes the attributes, so that they are grouped according to their propensity for change.

Primitive P11.2. HORIZONTAL PARTITION

Two relations, one for more current data and the other for historical information, are generated from an original one. Each resulting relation contains the same attributes as the source one.

Primitive P12. HIERARCHY GENERATION

This is a family of primitives that generate hierarchy relations, having as input relations that include a hierarchy or a part of one.

In addition, they transform the original relations, so that they do not include the hierarchy any more. Instead of this, they reference the new hierarchy relation or relations, through a foreign key.

The three primitives that compose this family implement three different design alternatives for the generated hierarchy.

Primitive P12.1. DE-NORMALIZED

This primitive generates only one relation for the hierarchy.

Primitive P12.2. SNOWFLAKE

This primitive generates several relations for the hierarchy, representing it in a normalized form.

Primitive P12.3. FREE DECOMPOSITION

This primitive generates several relations for the hierarchy. The form (distribution of attributes) of these relations is decided by the designer.

Primitive P13. MINIDIMENSION BREAK OFF

Often, in a dimension there is a set of attributes that have a limited number of possible values.

The idea is to code the various combinations of values of these attributes (only the combinations that really occur) and store them in a separate relation, so that they can be referenced from other relations. Storage space is saved using this structure.

This primitive generates two dimension relations. One is the result of eliminating a set of attributes from a dimension relation. The other is a relation that contains only this set of attributes. Besides, it defines a foreign key between the two relations.

Primitive P14. NEW DIMENSION CROSSING

In many cases, we need to materialize a dimension crossing in a new relation. This can be done through a join of some relations. For example, there is a measure relation where the product dimension is crossed with other dimensions, and another relation where supplier is determined by product. The supplier dimension can be added to the measure relation and the product can be removed, obtaining a crossing between supplier and the other dimensions existing in the measure relation.

5.3. Descripción de las nuevas primitivas

Primitive Q1. RELATION JOIN

This primitive is useful when we want to generate in the DW, a relation that is the join of two existent relations. For example, when we want to de-normalize a dimension we store together all the attributes of the dimension. Another application is when we want to perform a complex calculus that requires attributes from many tables, it is easier if all the attributes are stored together in the same relation.

Primitive Q2. ATTRIBUTE RENAMING

In many cases the attributes names do not express its correct mining. This primitive is useful to rename these attributes.

Besides, some primitives need attributes in different tables have the same name to perform the transformation, and this not always occurs in source databases.

Primitive Q3. INSTANCE FILTERING

Sometimes a relation contains data that is not useful for some queries. For example, a relation with information about customers may contain customers of all regions of the country. In some query we may want to list only customers who live in Montevideo. With these primitive we can generate a relation that contains only a subset of another relation instance.

Primitive Q4. PRIMARY KEY MODIFICATION

When some attributes are added or deleted from a relation, or when we group or filter its instance, the key of the relation may change. This primitive is useful for updating a primary key in a relation.

5.4. Especificación de las nuevas primitivas

Primitiva Q1:

Primitive Q1: RELATION JOIN

Description:

Given two relations, it generates a new one joining the giving relations.

Input:

- Source schema: 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}, the join function.
- $Y_1 \subseteq$ {A_1, …, A_n}, the set of attributes of R_1 to be excluded.
- $Y_2 \subseteq$ {B_1, …, B_m}, the set of attributes of R_2 to be excluded.
- Source instance: r_1, r_2.

Resulting schema:

- R' (D_1, ..., D_r) $\in$ *Rel* / {D_1, ..., D_r} = ({A_1, ..., A_n} - Y_1) $\cup$ ({B_1, ..., B_m} - Y_2) $\wedge$ Att_K (R') $\subseteq$ Att_K (R') $\cup$ Att_K (R')

Generated instance:

- r' = select (Att(R_1) - Y_1) $\cup$ (Att(R_2) - Y_2)
  from r_1, r_2
  where f (C_1, ..., C_k)

Primitiva Q2:

Primitive Q2: ATTRIBUTE RENAMING

Description:

Given a relation, it generates another by renaming some of the attributes.

Input:

- source schema : R(A_1, ..., A_n) $\in$ *Rel*
- { A_{i1}, ..., A_{im} } $\subseteq$ { A_1, ..., A_n }, a set of attributes of R
- { B_1, ..., B_m }, the new names for the given attributes
- source instance : r

Resulting schema:

- R' $\in$ *Rel* / R' = R but with another attribute names.

Generated instance:

- r' = r

Primitiva Q3:

Primitive Q3: INSTANCE FILTERING

Description:

Given a relation, it generates another that is structurally the same, but eliminates the tuples that not satisfy a condition.

Input:

- Source schema : $R(A_1, ..., A_n) \in Rel$
- $f (C_1, ..., C_k) / \{ C_1, ..., C_k \} \subseteq \{ A_1, ..., A_n \}$, where f is a user-defined function that returns boolean.
- Source instance : r

Resulting schema:

- $R' \in Rel$ / $R' = R$

Generated instance:

- $r' \subseteq r$
  r' = select * from r where $f(C_1, ..., C_k)$;

Primitiva Q4:

Primitive Q4: PRIMARY KEY MODIFICATION

Description:

Given a relation, it changes its primary key.

Input:

- Source schema : $R(A_1, ..., A_n) \in Rel$
- $\{ C_1, ..., C_k \} \subseteq \{ A_1, ..., A_n \}$, the new primary key attributes.
- source instance : r

Resulting schema:

- $R' \in Rel$ / $R' = R$ but with another primary key.

Generated instance:

- $r' = r$

5.5. Conclusiones

En este anexo se presentó una descripción del conjunto de primitivas propuestas en [Mar00] y se lo extendió con nuevas primitivas.

Las nuevas primitivas realizan operaciones para casos más generales, en los cuales es difícil optar entre varias primitivas. Además se agregan primitivas para trabajar con instancias, no sólo transformar esquemas.

Se incorporaron las primitivas necesarias para realizar este trabajo, pueden ser necesarias otras primitivas para otro tipo de trabajos, por ejemplo para resolver la carga incremental del DW.

Anexo 6. Diagramas de Clases de la Herramienta

En este apéndice se presentan los diagramas de clases de alto nivel de la herramienta, y los diagramas de clases de bajo nivel de las clases agregadas o modificadas en el contexto de esta tesis. El resto de las clases pueden encontrarse en [Gar00] y [Mar00].

Las clases coloreadas representan cambios respecto a la versión anterior, y las clases en negrita son nuevas clases.

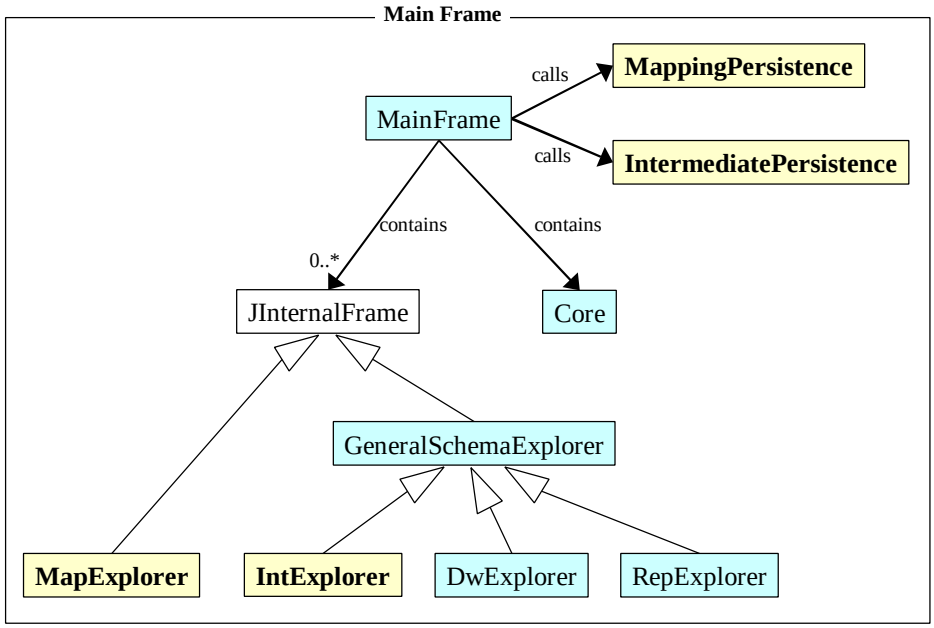

Figura 55 – Frame principal

La Figura 55 muestra el diagrama de clases de alto nivel de la herramienta. La clase *MainFrame* es la ventana principal de la aplicación, la cual contiene los menús, barras de herramientas y sub-ventanas para desplegar y navegar por la estructura de los objetos (*MapExplorer*, *IntExplorer*, *DwExplorer* y *RepExplorer*). La clase invoca a las clases de manejo de persistencia (*IntermediatePersistence* e *MappingPersistence*) responsables del guardado y recuperación de objetos y crea la máquina virtual (*Core*).

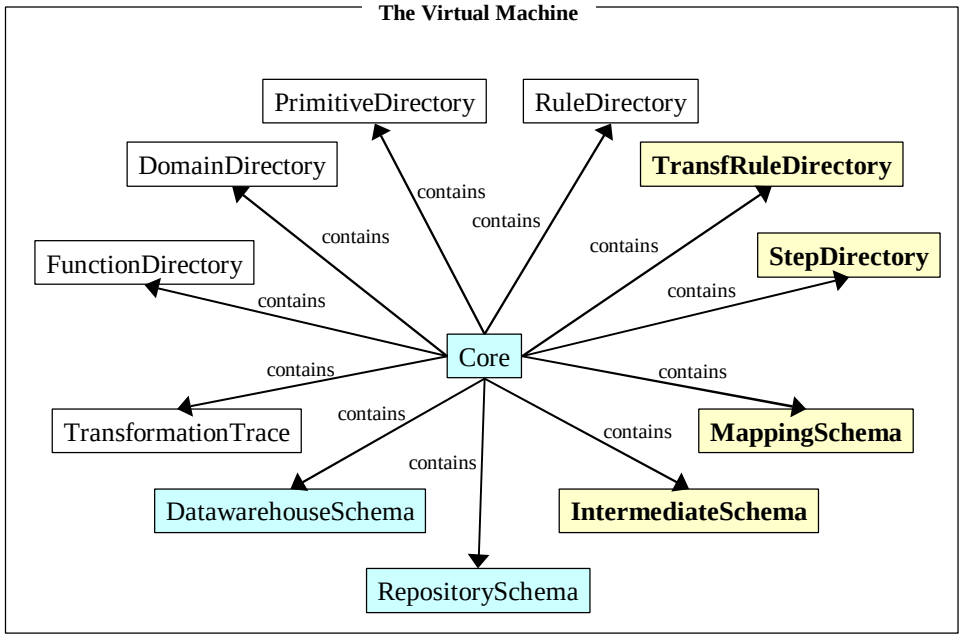

Figura 56 – Máquina virtual

La Figura 56 muestra el diagrama de clases de la máquina virtual (*Core*). La máquina virtual contiene directorios de objetos (*FunctionDirectory*, *DomainDirectory*, *PrimitiveDirectory*, *RuleDirectory*, *TransfRuleDirectory* y *StepDirectory*). También contiene los esquemas (*DatawarehouseSchema*, *RepositorySchema*, *IntermediateSchema* y *MappingSchema*) y la traza de diseño (*TransformationTrace*).

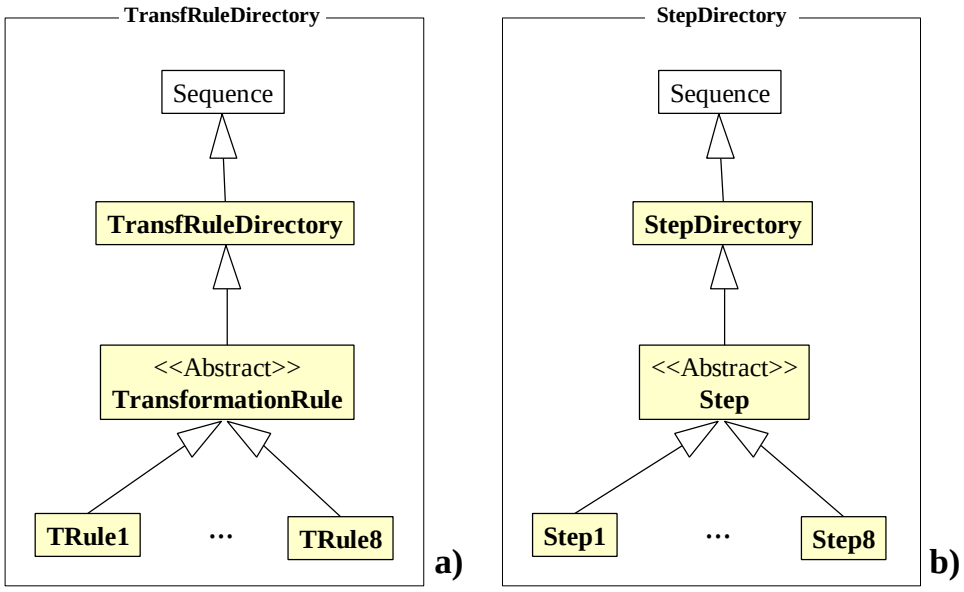

Figura 57 – Directorios: a) de reglas de diseño y, b) de pasos

La Figura 57 muestra la jerarquía de clases de los directorios, y la utilización de clases abstractas para facilitar la incorporación de nuevos objetos (reglas de diseño y pasos).

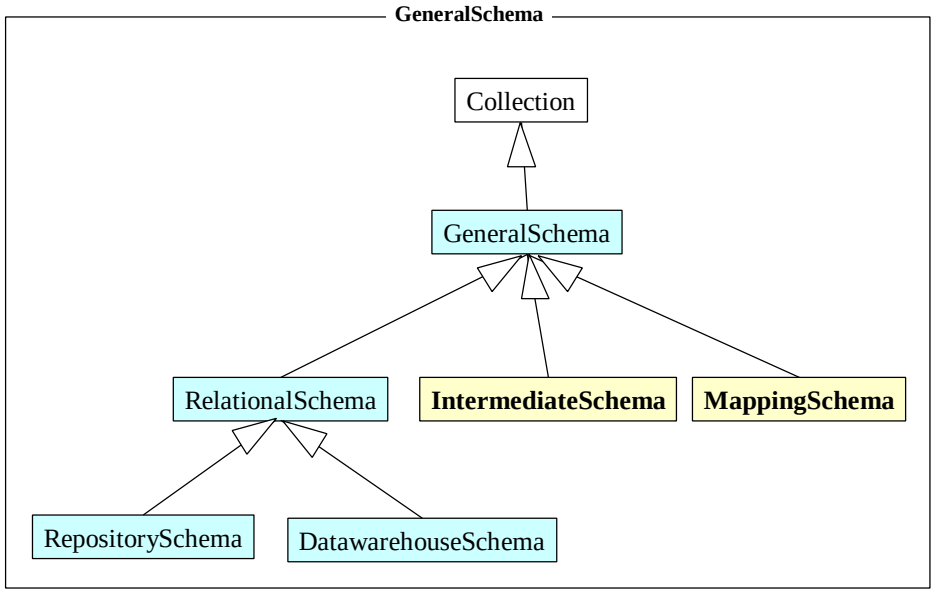

Figura 58 – Esquema intermedio y esquema de mapeos

La Figura 58 muestra la jerarquía de clases de los esquemas.

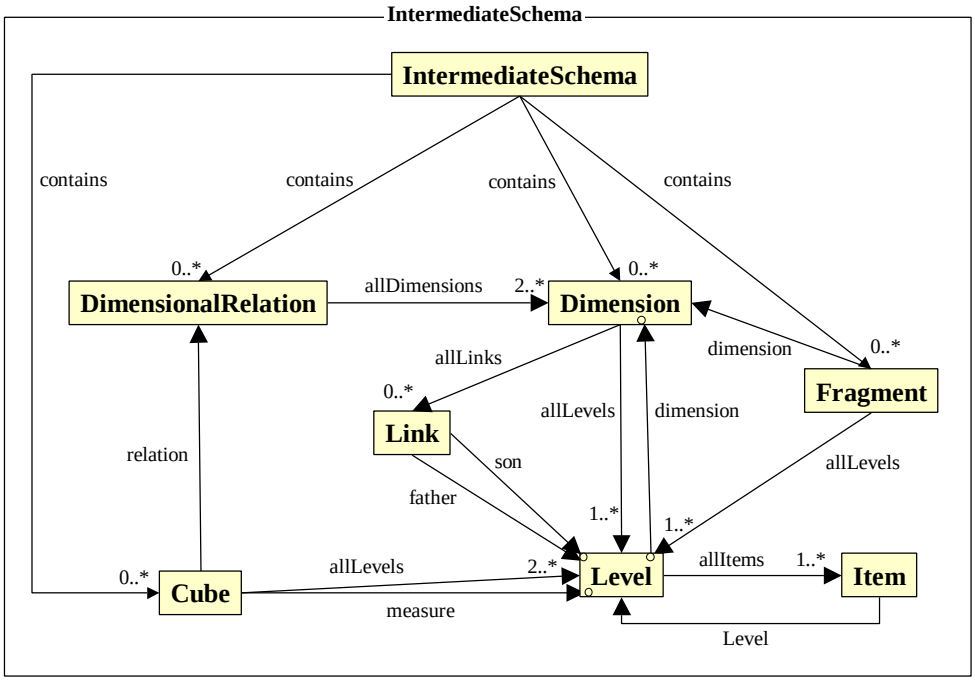

Figura 59 – Esquema intermedio

La Figura 59 muestra el esquema intermedio y sus componentes: relaciones dimensionales, dimensiones, cubos y fragmentos.

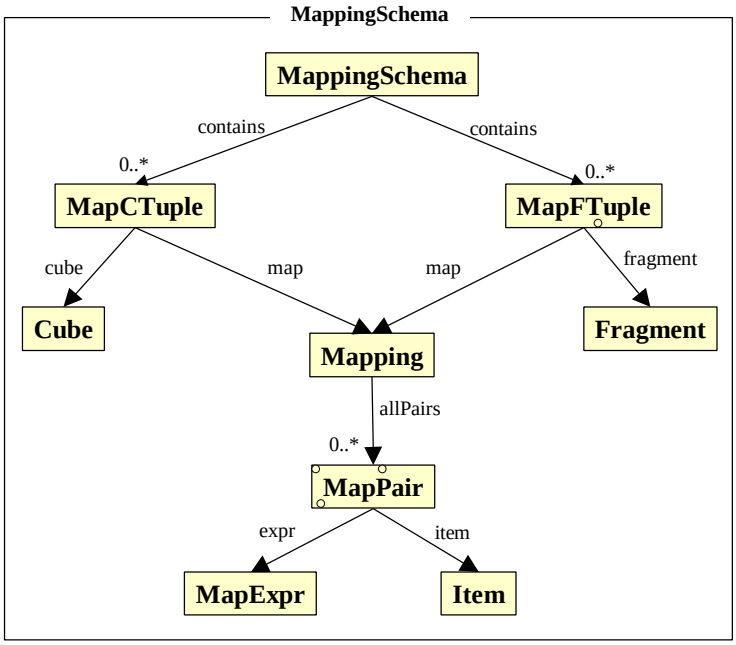

Figura 60 – Esquema de mapeos

La Figura 60 muestra el esquema de mapeos y sus componentes: *MapFTuple* y *MapCTuple*, que representan los mapeos de fragmentos y cubos respectivamente.

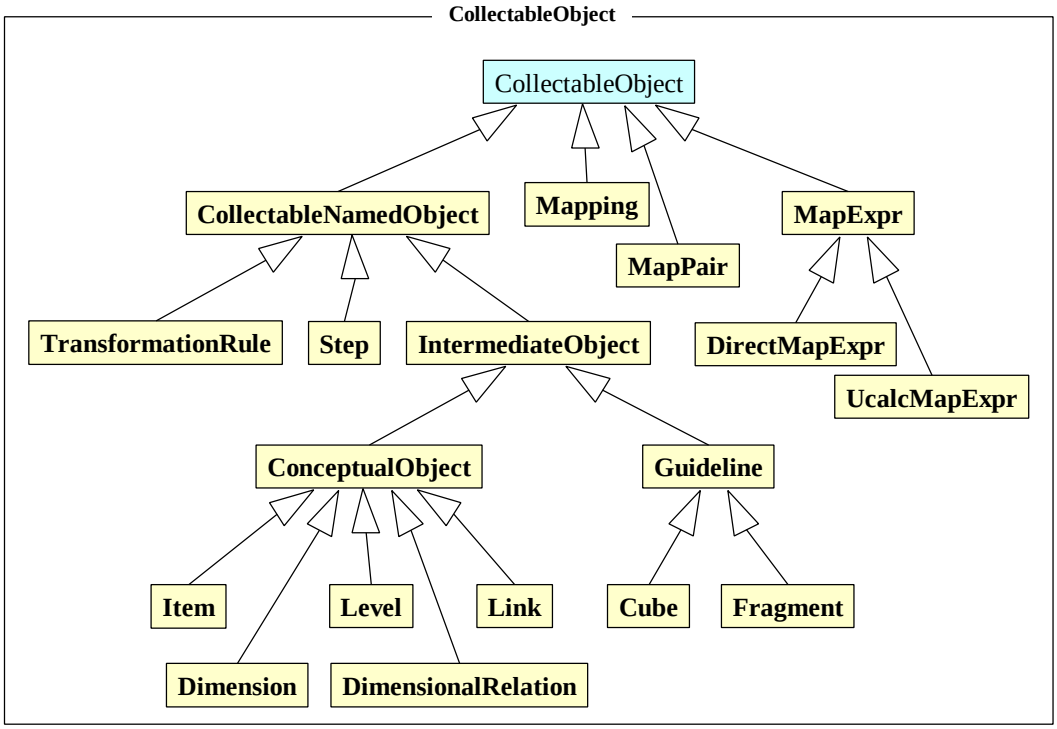

Figura 61 – Objetos coleccionables

La Figura 61 muestra la jerarquía de clases de los objetos coleccionables, tanto reglas de diseño y pasos como objetos del esquema intermedio y mapeos. Se incorporó la clase *CollectableNamedObject* para abstraer la propiedad nombre común en todas las clases.

Anexo 7. Un caso de Estudio

En este anexo se presenta el proceso de diseño a través de un caso de estudio. En el mismo se muestra tanto la definición de lineamientos y mapeos como la aplicación del algoritmo.

Una versión más extensa del caso de estudio puede encontrarse en [Per01a].

Las siguientes secciones siguen paso a paso el proceso de diseño del DW. En la sección 7.1 se introduce el caso de estudio, tanto el esquema conceptual como la base fuente. En la sección 7.2 se definen los lineamientos y en la sección 7.3 se definen los mapeos entre el esquema intermedio y la fuente. En la sección 7.4 se muestra la aplicación del algoritmo y en la sección 7.5 se concluye.

7.1. Presentación del caso de estudio

En esta sección se presenta un caso de estudio sobre una empresa que distribuye productos agrícolas. Se presenta el esquema conceptual y la base fuente. La narrativa del caso de estudio y los requerimientos pueden encontrarse en [Per01a].

Esquema conceptual

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

Se relevaron 4 dimensiones: artículos, clientes, vendedores y fechas. La dimensión cantidades representa a las medidas, pero es tratada como una dimensión más ya que CMDM trabaja con dimensionalidad genérica [Car00]. La Figura 63 muestra la representación gráfica de las dimensiones.

Se relevó una única relación dimensional: venta, que vincula todas las dimensiones relevadas. La Figura 62 muestra la representación gráfica de la relación dimensional.

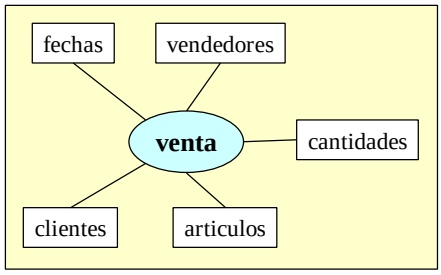

Figura 62 – Relaciones Dimensionales

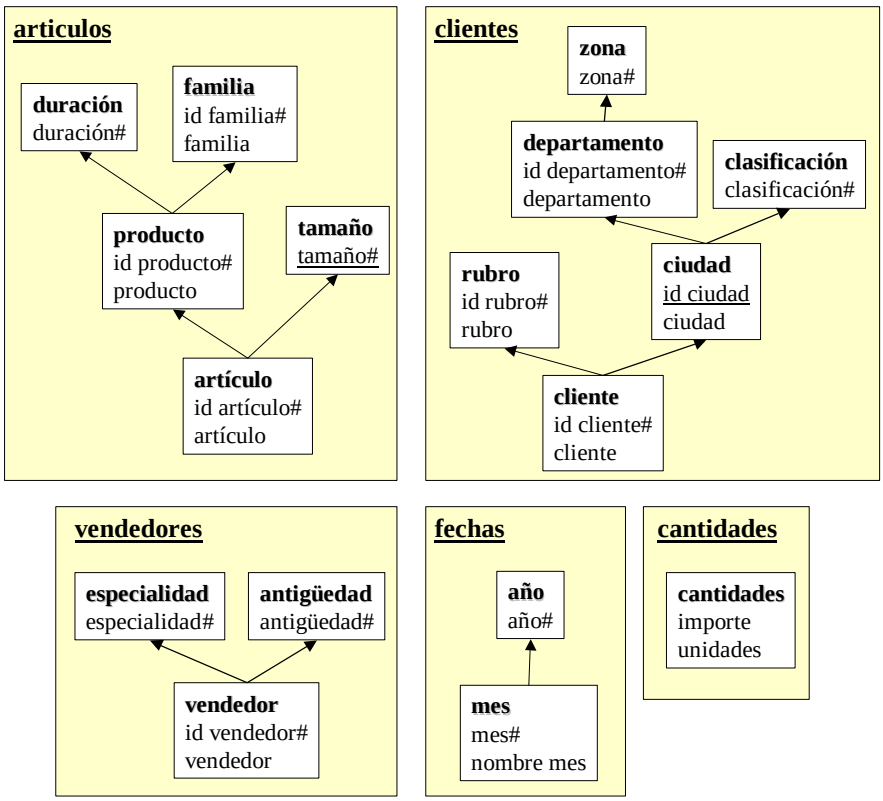

Figura 63 – Dimensiones y Niveles

Base 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.

La Figura 64 bosqueja los links entre las tablas.

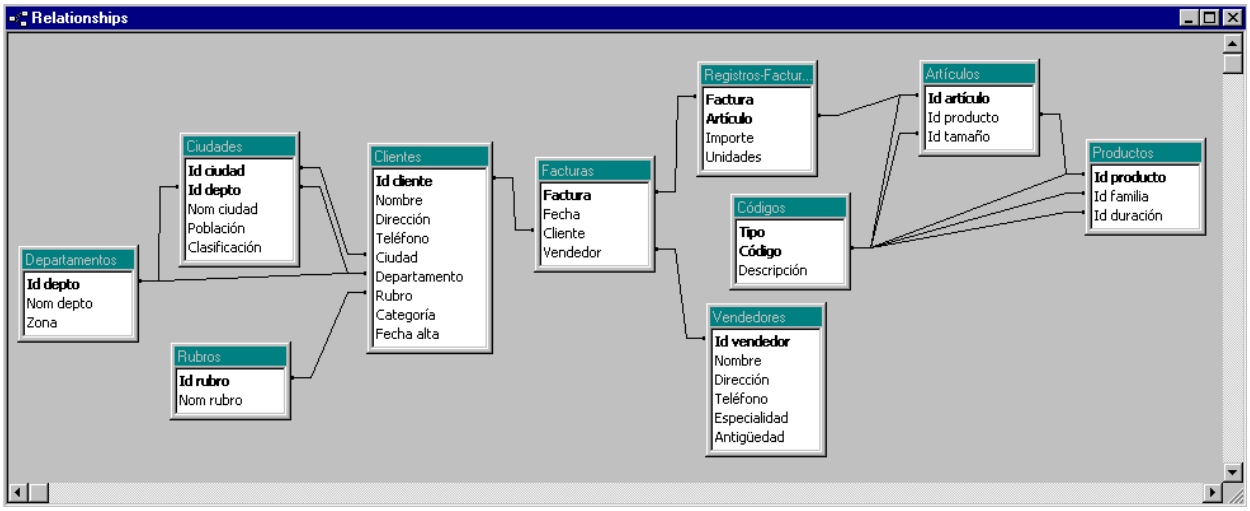

Figura 64 – Links entre tablas de la base fuente

Se presenta un problema con las múltiples líneas entre las tablas *Artículos* y *Códigos*, ya que la intención definir links para dos instancias de la tabla *Códigos*, no cumplir ambas condiciones. Para ello se definen alias de las tablas.

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

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

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

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

Una descripción del manejo de alias puede encontrarse en el Anexo 3.

Los atributos de las tablas *Artículos* y *Productos* referencian a los respectivos alias (por ejemplo: se define un link entre el atributo *Id articulo* de la tabla *Artículos* con el atributo *Id articulo* de la tabla *Codigos-id articulo*). La Figura 65 muestra la definición de links para las nuevas tablas.

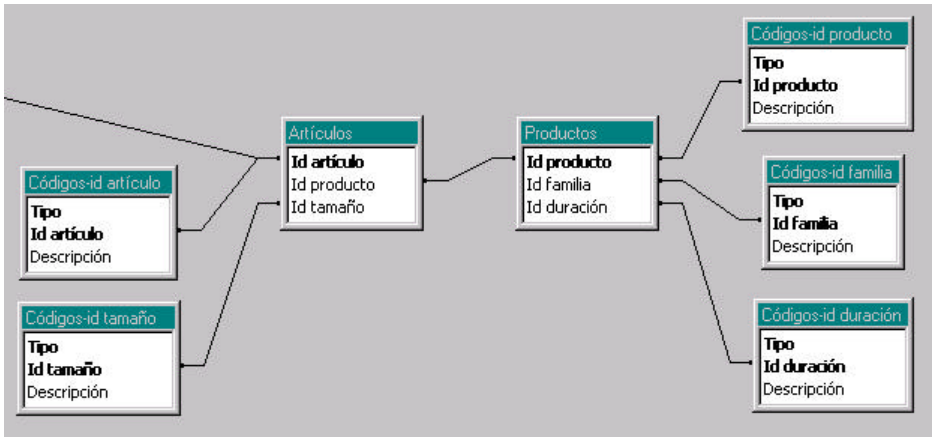

Figura 65 – Alias y Re-definición de los Links (fracción de los links)

7.2. Lineamientos

A continuación se presentan los lineamientos definidos para el ejemplo.

Materialización de Relaciones

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

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

La Figura 66 muestra la representación gráfica de los cubos.

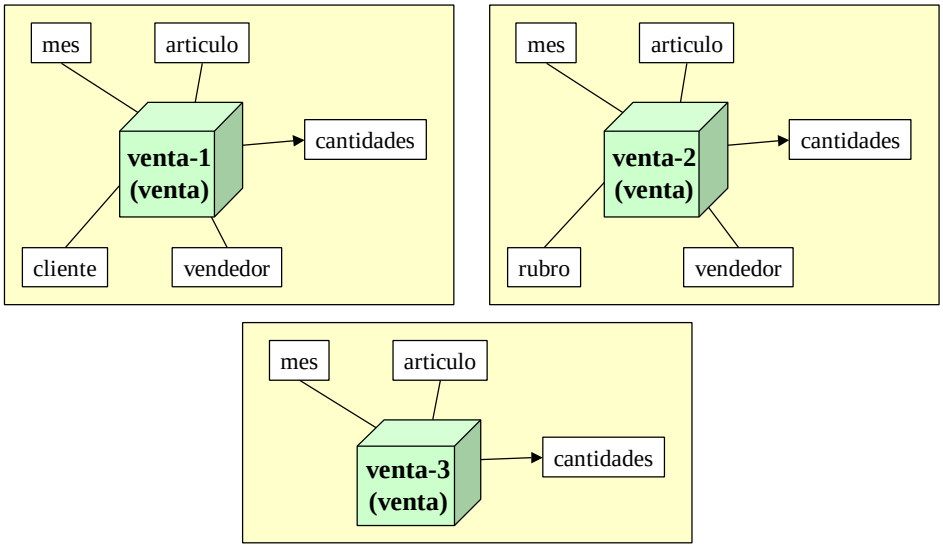

Figura 66 – Cubos definidos

Fragmentación de Cubos

Se decide fragmentar los cubos de la siguiente manera:

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

La Figura 67 muestra la representación gráfica de las franjas definidas.

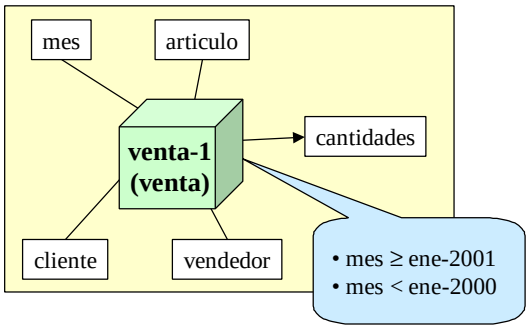

Figura 67 – Franjas definidas

Fragmentación de Dimensiones

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

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

La Figura 68 muestra la representación gráfica de los fragmentos.

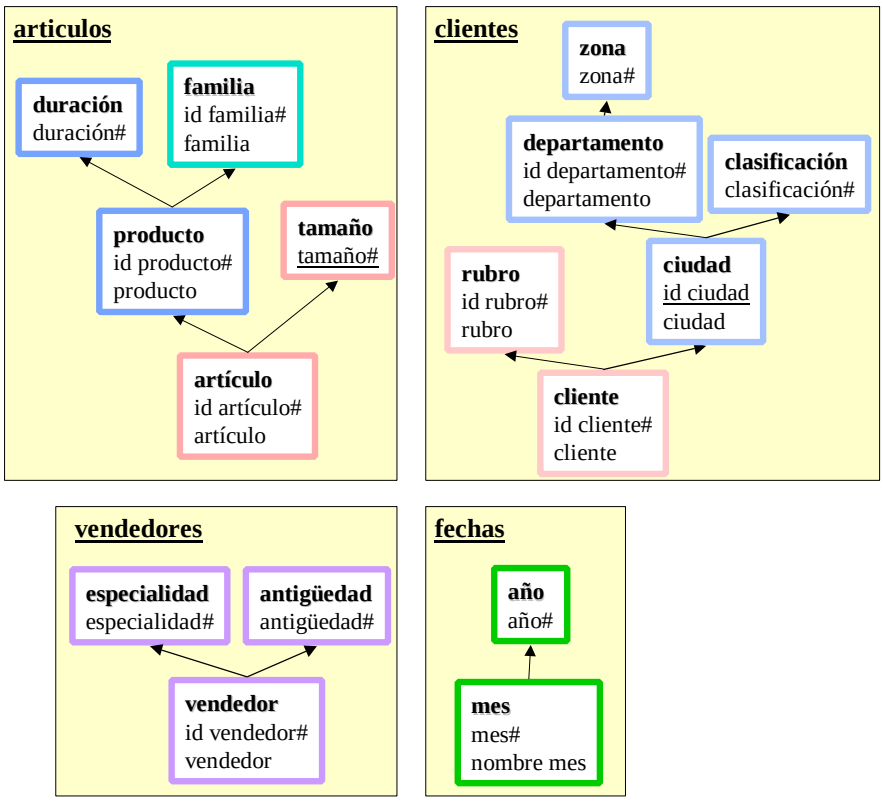

Figura 68 – Fragmentos definidos

7.3. Mapeos

A continuación se presentan los mapeos definidos para los fragmentos de las dimensiones y para los cubos. En algunos casos se definen condiciones de mapeo.

Mapeos de Fragmentos

La Figura 69 y la Figura 70 muestran los mapeos de los fragmentos de la dimensión clientes. La Figura 71, la Figura 72 y la Figura 73 muestran los mapeos de los fragmentos de la dimensión artículos. La Figura 74 muestra el mapeo del único fragmento de la dimensión vendedores, y la Figura 75 muestra el de la dimensión fechas.

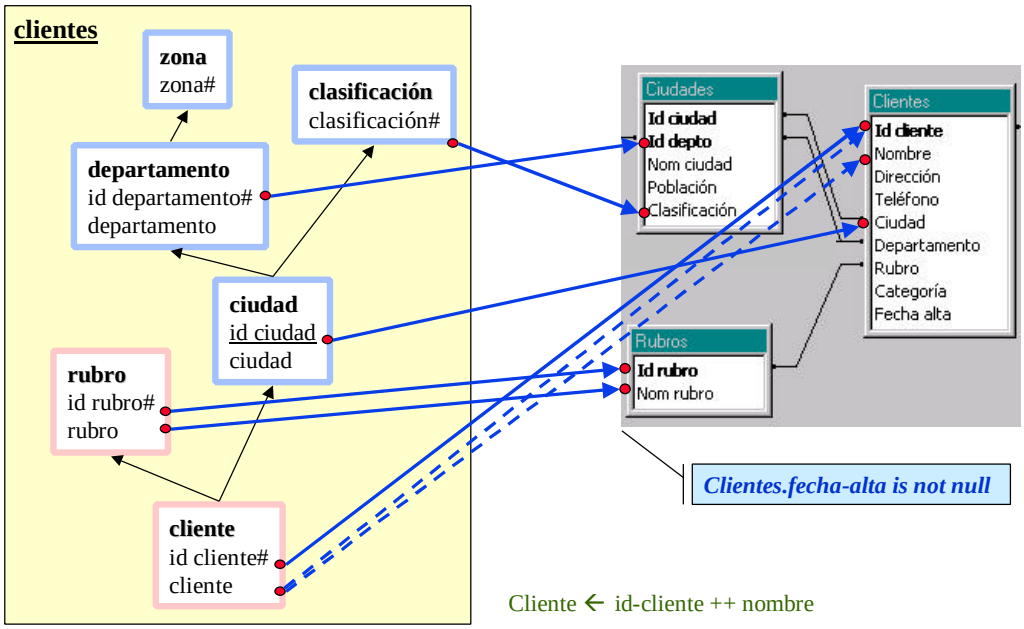

Figura 69 – Mapeo del fragmento rosa de la dimensión clientes

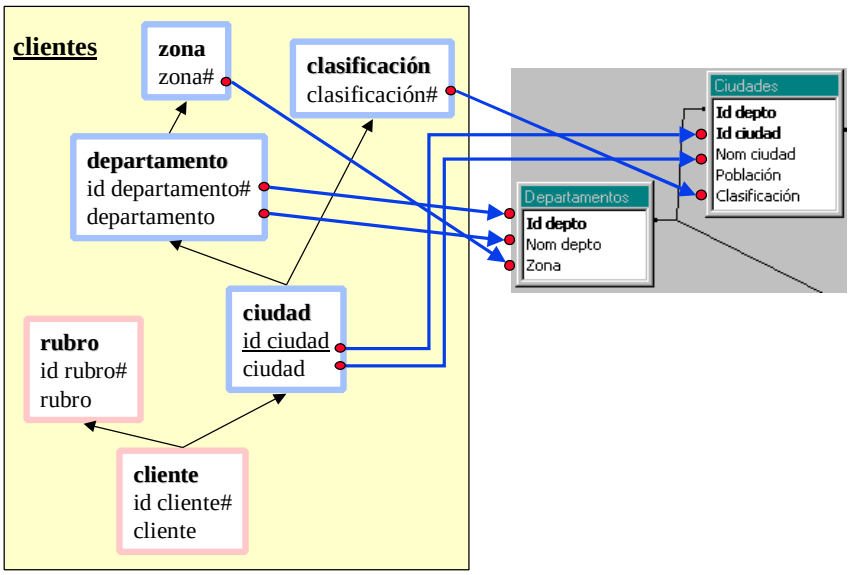

Figura 70 – Mapeo del fragmento celeste de la dimensión clientes

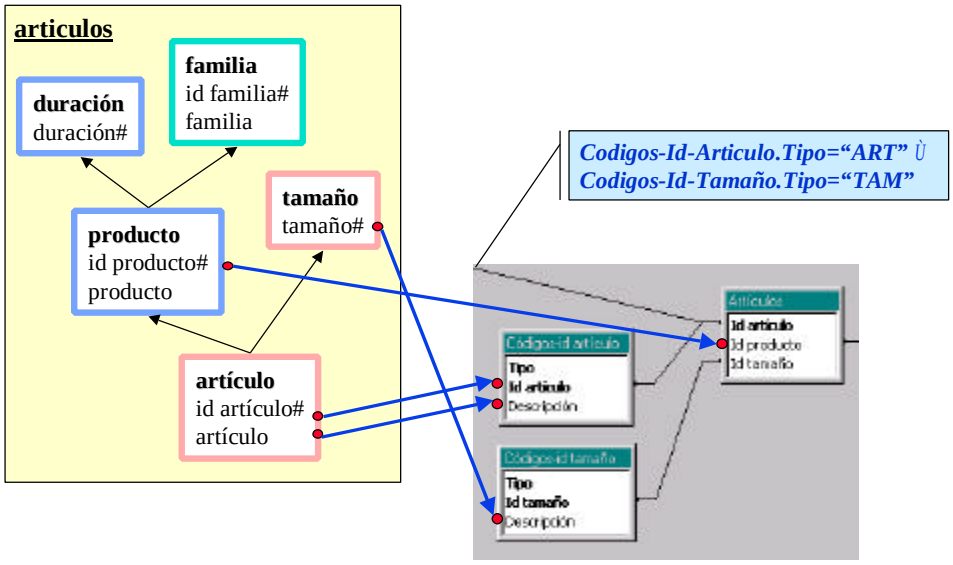

Figura 71 – Mapeo del fragmento rosa de la dimensión artículos

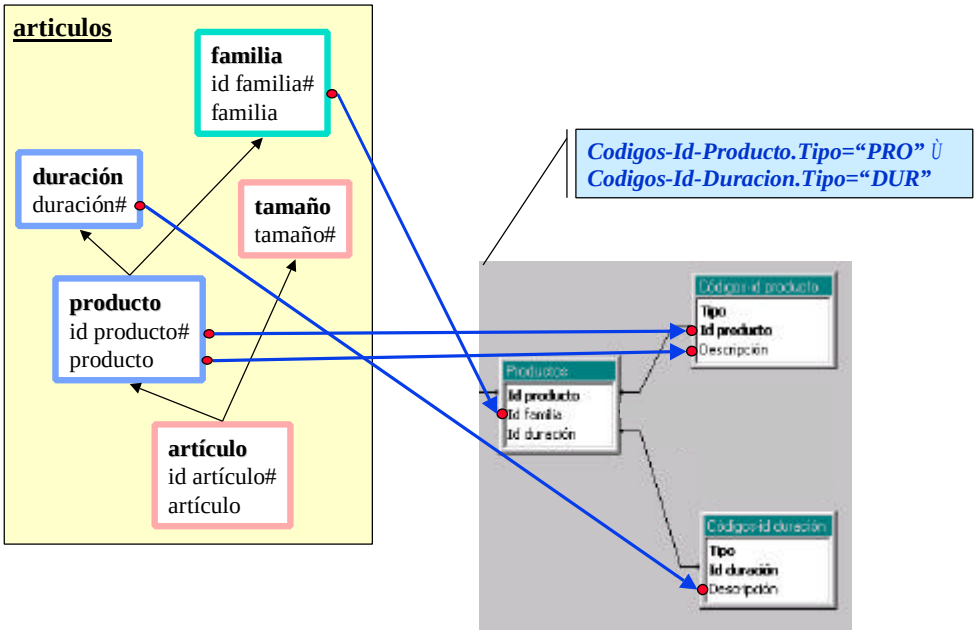

Figura 72 – Mapeo del fragmento azul de la dimensión artículos

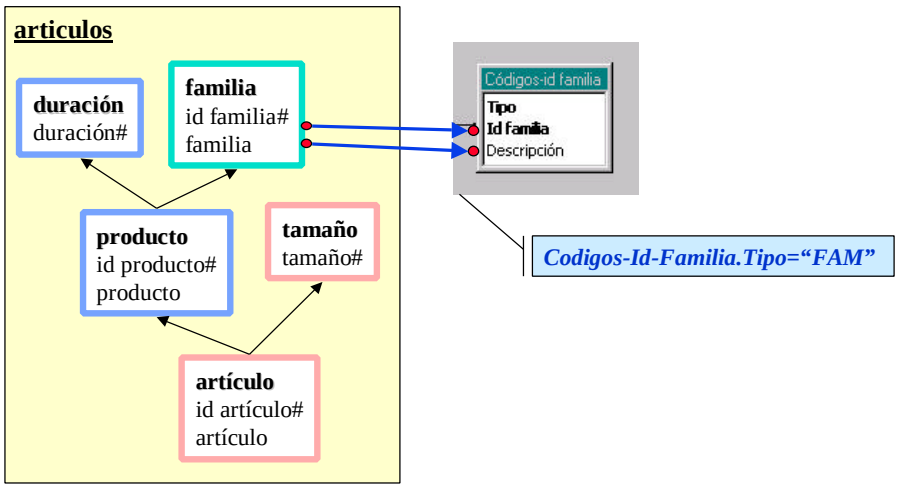

Figura 73 – Mapeo del fragmento verde de la dimensión artículos

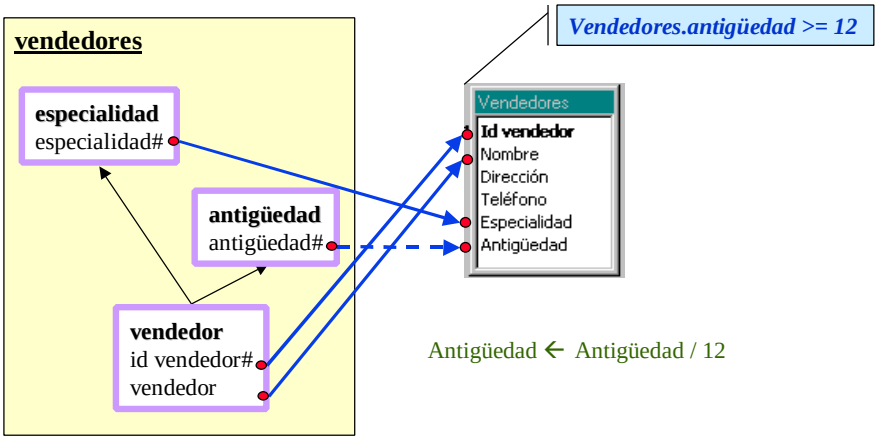

Figura 74 – Mapeo del fragmento de la dimensión vendedores

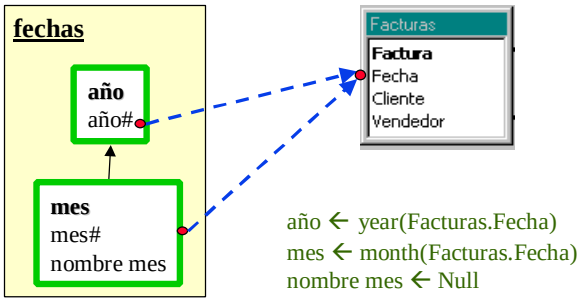

Figura 75 – Mapeo del fragmento de la dimensión fechas

Mapeos de Cubos

La Figura 76 muestra el mapeo del cubo venta-1. La Figura 77 muestra el mapeo del cubo venta-2 como drill-up del cubo venta-1 por la dimensión clientes. La Figura 78 muestra el mapeo del cubo venta-3 como drill-up del cubo venta-1 eliminando dimensiones.

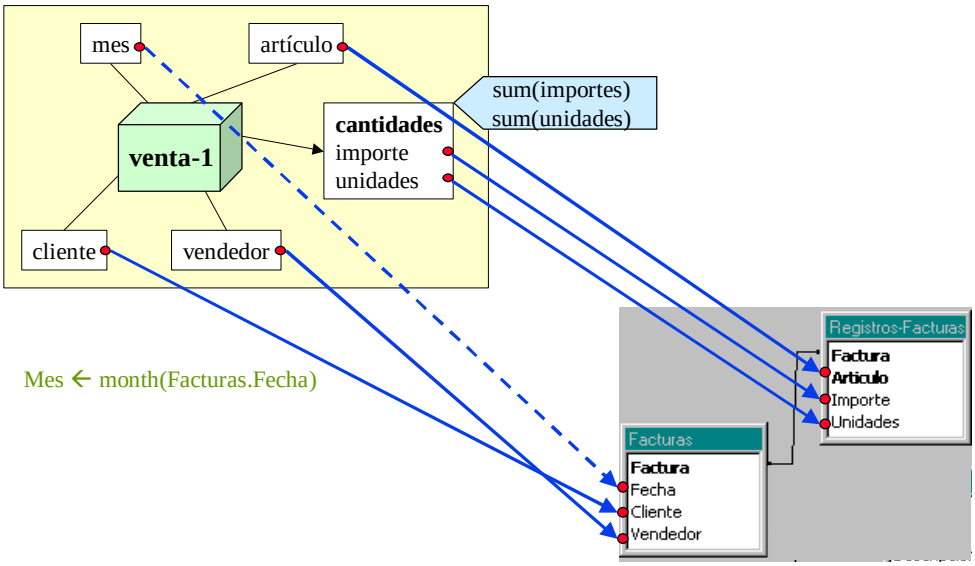

Figura 76 – Mapeo del cubo venta-1

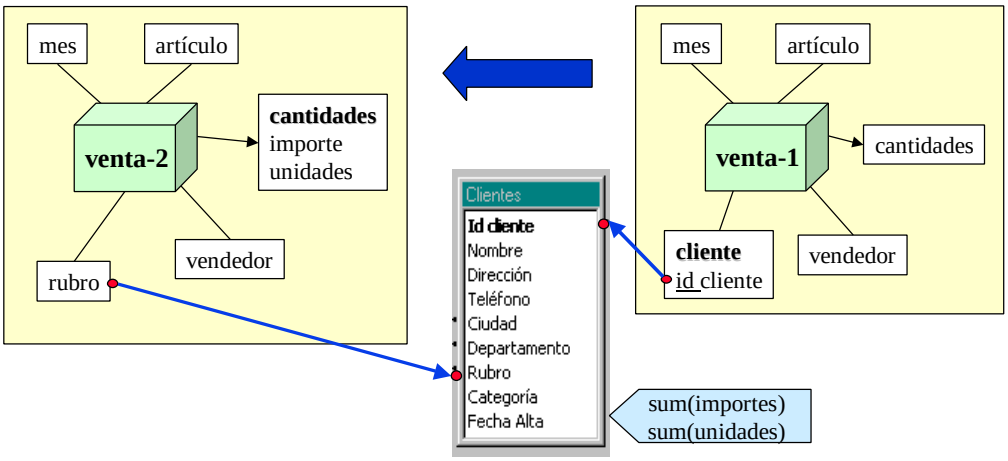

Figura 77 – Mapeo del cubo venta-2

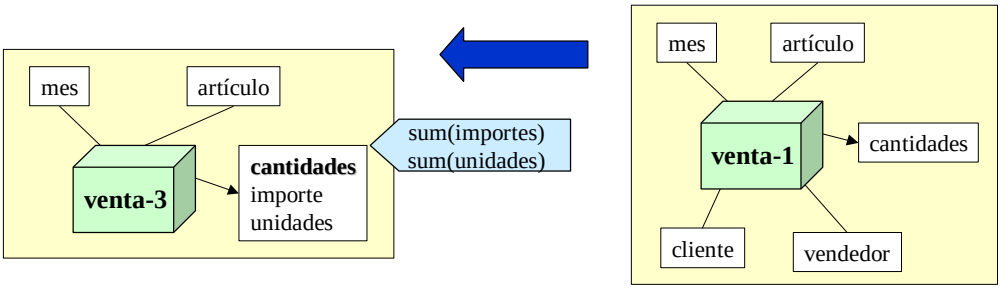

Figura 78 – Mapeo del cubo venta-3

7.4. Aplicación del algoritmo

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

CONSTRUCCIÓN DE TABLAS DE DIMENSIÓN

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

STEP 1 - CONSTRUIR EL ESQUELETO

El fragmento rosa de la dimensión *clientes* (se le llamará DwClientes) mapea a tres tablas fuentes: *Clientes*, *Ciudades* y *Rubros*. Se itera aplicando la regla R1 (*Join*) de a dos tablas. Sea M = SchFMapping(DwClientes).Map, la función de mapeo del fragmento.

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

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

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

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

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

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

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

STEP 2 - RENOMBRAR ATRIBUTOS PARA ÍTEMS CON MAPEO DIRECTO

Los ítems *rubro* e *id-departamento* del fragmento DwClientes tienen mapeos directos a los atributos *nom-rubro* y *id-depto* respectivamente, cuyos nombres difieren de los nombres de los ítems. Se aplica la regla R2 (*Rename*). Sea M = SchFMapping(DwClientes).Map, la función de mapeo del fragmento.

- Ejecutar Rename (DwClientes, M, DwClientes02).

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

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

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

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

STEP 3 - GENERAR ATRIBUTOS PARA ÍTEMS CON MAPEO CALCULADO O EXTERNO

El ítem *cliente* del fragmento DwClientes tiene mapeo de cálculo simple (1calcME) a los atributos *id-cliente* y *nombre*. Se aplica la regla R3.1 (*Simple-Calculate*).

Sea M = SchFMapping(DwClientes).Map, la función de mapeo del fragmento.

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

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

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

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

El ítem *nombre-mes* del fragmento DwFechas tiene mapeo externo de tipo constante. Se aplica la regla R4.1 (*Constant-Extern-Value*). Sea M = SchFMapping(DwClientes).Map, la función de mapeo del fragmento.

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

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

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

STEP 4 - APLICAR FILTROS

El fragmento DwClientes tiene una condición en su función de mapeo: *Clientes.fecha-alta is not null*. Se aplica la regla R8 (*Filter*). Sea M = SchFMapping(DwClientes).Map, la función de mapeo del fragmento.

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

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

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

- DwArticulos04 (tipo1, id-articulo, articulo, id-articulo2, id-producto, id-tamaño1, tipo2, id-tamaño2, tamaño)

- DwProductos04 (id-producto1, id-familia, id-duracion1, tipo1, id-producto, producto, tipo2, id-duracion2, duracion)
- DwFamilias02 (tipo, id-familia, familia)
- DwVendedores03 (id-vendedor, vendedor, direccion, telefono, especialidad, antiguedad1, antiguedad)

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

STEP 5 - ELIMINAR ATRIBUTOS SOBRANTES

Los atributos *nombre*, *direccion*, *telefono*, *ciudad*, *departamento*, *rubro1*, *categoria*, *fecha-alta* y *nom-ciudad* no están mapeados al fragmento DwClientes. Se aplica la regla R5.1 (*Fragment-Group*). Sea M = SchFMapping(DwClientes).Map, la función de mapeo del fragmento.

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

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

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

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

STEP 6 - AJUSTAR LAS CLAVES

La clave del fragmento DwClientes es *id-cliente* (clave del nivel inferior en la jerarquía), y la clave de la tabla que lo mapea (*DwClientes06)* está formada por los atributos *id-cliente*, *id-rubro*, *id-departamento* e *id-ciudad*. Se aplica la regla R7 (*Primary-Key*).

Sea M = SchFMapping(DwClientes).Map, la función de mapeo del fragmento.

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

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

Análogamente se aplica la regla al fragmento DwFechas. Se obtiene como resultado la tabla:

- DwFechas04 (año, mes, nombre-mes)

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

Se tienen como resultados finales las tablas:

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

CONSTRUCCIÓN DE TABLAS DE HECHOS PARA CUBOS CON MAPEO BASE

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

STEP 7 - CONSTRUIR EL ESQUELETO

El cubo venta-1 mapea a dos tablas fuentes: *Facturas* y *Registros-Facturas*. Se aplica la regla R1 (*Join*). Sea M = SchCMapping(venta-1).Map, la función de mapeo del cubo.

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

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

STEP 8 - RENOMBRAR ATRIBUTOS PARA ÍTEMS CON MAPEO DIRECTO

Los ítems *id-cliente*, *id-vendedor* e *id-articulo* del cubo venta-1 tienen mapeos directos a los atributos *cliente*, *vendedor* y *articulo* respectivamente, cuyos nombres difieren de los nombres de los ítems. Se aplica la regla R2 (*Rename*). Sea M = SchCMapping(venta-1).Map, la función de mapeo del cubo.

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

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

STEP 9 - GENERAR ATRIBUTOS PARA ÍTEMS CON MAPEO CALCULADO O EXTERNO

El ítem *mes* del cubo venta-1 tiene mapeo de cálculo simple (1calcME) al atributo *fecha*. Se aplica la regla R3.1 (*Simple-Calculate*). Sea M = SchCMapping(venta-1).Map, la función de mapeo del cubo.

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

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

STEP 10 - APLICAR FILTROS

El cubo no tiene condiciones en el mapeo por lo que no se aplica la regla R8 (*Filter*).

STEP 11 - ELIMINAR ATRIBUTOS SOBRANTES

Los atributos *factura1*, *fecha* y *factura-2* no están mapeados al cubo venta-1. Se aplica la regla R5.2 (*Cube-Group*). Sea M = SchCMapping(venta-1).Map, la función de mapeo del cubo.

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

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

STEP 12 - AJUSTAR LAS CLAVES

La clave del cubo venta-1 está formada por los ítems *id-cliente*, *id-vendedor*, *id-artículo* y *mes* (clave de los niveles que no son medida), y la clave de la tabla que lo mapea (*DwVenta104)* es *id-articulo*. Se aplica la regla R7 (*Primary-Key*). Sea M = SchCMapping(venta-1).Map, la función de mapeo del cubo.

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

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

Se tienen como resultado final la tabla:

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

CONSTRUCCIÓN DE TABLAS DE HECHOS PARA CUBOS CON MAPEO RECURSIVO

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

STEP 13 - ARMAR LA TABLA DE LA JERARQUIA

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

STEP 1 – CONSTRUIR EL ESQUELETO

El fragmento ClientesAuxiliar mapea a una única tabla por lo que no verifica la precondición de la regla R1 (*Join*). La regla no se aplica.

STEP 2 – RENOMBRAR ATRIBUTOS PARA ITEMS CON MAPEO DIRECTO.

El ítem *id-rubro* del fragmento ClientesAuxiliar tiene un mapeo directo al atributo *rubro*, cuyo nombre difiere del nombre del ítem. Se aplica la regla R2 (*Rename*).

Sea M = SchFMapping(ClientesAuxiliar).Map, la función de mapeo del fragmento.

- Ejecutar Rename (ClientesAuxiliar, M3, Clientes).

  Resultado DwClientesAuxiliar01 (id-cliente, nombre, direccion, telefono, ciudad, departamento, id-rubro, categoria, fecha-alta)

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

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

STEP 4 – APLICAR FILTROS.

El fragmento ClientesAuxiliar no tiene condiciones en su función de mapeo por lo que no se aplica la regla R8 (*Filter*).

STEP 5 – ELIMINAR ATRIBUTOS SOBRANTES.

Los atributos *nombre*, *direccion*, *telefono*, *ciudad*, *departamento*, *categoria* y *fecha-alta* no están mapeados al fragmento ClientesAuxiliar. Se aplica la regla R5.1 (*Fragment-Group*).

Sea M = SchFMapping(ClientesAuxiliar).Map, la función de mapeo del fragmento.

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

  Resultado DwClientesAuxiliar02 (id-cliente, id-rubro)

STEP 6 – AJUSTAR LAS CLAVES.

No es necesario ajustar la clave del fragmento ClientesAuxiliar por lo que no se aplica la regla R7 (*PrimaryKey*).

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

STEP 14 - APLICAR EL DRILL-UP

El cubo venta-2 tiene un mapeo recursivo respecto al cubo venta-1, con un drill-up por la dimensión *clientes* para reducir el detalle de la dimensión. Se aplica la regla R6.1 (*Hierarchy-Drill-Up*).

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

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

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

El cubo venta-3 tiene un mapeo recursivo respecto al cubo venta-1, con dos drill-ups por las dimensiones *clientes* y *vendedores* para eliminar el detalle de las dimensiones. Se aplica la regla R6.2 (*Total-Drill-Up*).

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

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

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

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

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

Se tienen como resultados finales las tablas:

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

CONSTRUCCIÓN DE TABLAS DE HECHOS PARA FRANJAS DE CUBOS

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

STEP 15 - ARMAR LA TABLA DE CADA FRANJA

El cubo venta-1 tiene definidas dos bandas: *mes >= ene-2001* (a la que se llamará banda1) y *mes < ene-2001* (a la que se llamará banda2). Se aplica la regla R8 (*Filter*) para cada banda. Sea M = SchCMapping(venta-1).Map, la función de mapeo del cubo.

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

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

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

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

Se tienen como resultados finales las tablas:

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

Se obtiene como resultado el esquema lógico que se muestra en la Figura 79:

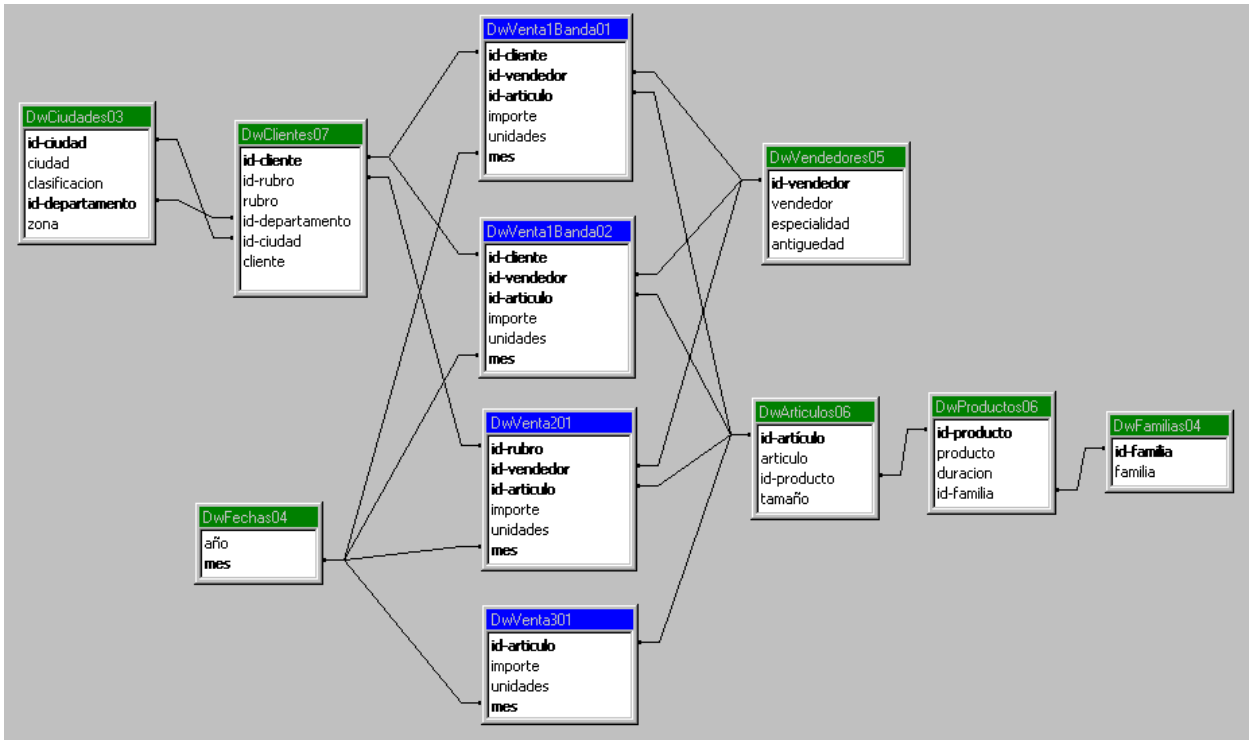

Figura 79 – Esquema lógico del DW

7.5. Conclusiones

En este anexo se presenta un caso de estudio en diseño de DWs. En particular se muestra la aplicación del proceso de diseño para generar el esquema relacional del DW a partir de su esquema conceptual.

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

Bibliografía

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

[Agr97] Agrawal, R. Gupta, A. Sarawagi, S.: "Modelling Multidimensional Databases", ICDE’97, UK, 1997.

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

[Arz00] G. Arzúa, G. Gil, S. Sharoian. “Manejador de Repositorio para Ambiente CASE”. Undergraduate Project. Advisor: Raúl Ruggia. InCo, Universidad de la República, Uruguay, 2000.

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

[Bar97] Baralis, E. Paraboschi, S. Teniente, E.: “Materialized view selection in a multidimensional database”. VLDB’97, Greece, 1997.

[Bar01] Barcelo, L. Bartesaghi, M. Bettosini, F. Fagalde, B. Lezue, S. Michelena, L. Miranda, C. Polero, M. Rouiller, M. Saccone, M. Zubia, F.: “Implementación de herramientas CASE que asistan en el Diseño de Data Warehouses (Definición de Mapeos)”. Software engineering project. Advisor: Jorge Triñanes. Users: Verónika Peralta, Raúl Ruggia. InCo, Universidad de la República, Uruguay. On going work.

[Bat92] Batini, C. Ceri, S. Navathe, S.: “Conceptual Database Design. An Entity-Relationship Approach”. Benjamin/Cummings Publishing. 1992.

[Bit01] Bittencourt, E. Borghi, D. Bravo, A. Camino, L. Giménez, H. González, M. Madera, M. Ravinovich, J. Roldós, G. Romano, X. Serra, F. Viera, M.: “Implementación de herramientas CASE que asistan en el Diseño de Data Warehouses (Interfaz Web)”. Software engineering project. Advisor: Andrés Vignaga. Users: Verónika Peralta, Raúl Ruggia. InCo, Universidad de la República, Uruguay. On going work.

[Boe99] Boehnlein, M. Ulbrich-vom Ende, A.:”Deriving the Initial Data Warehouse Structures from the Conceptual Data Models of the Underlying Operational Information Systems.", DOLAP’99, USA, 1999.

[Boh97] Bohn, K.: “Converting Data for Warehouses”. DBMS magazine, June 1997.

[Bou99] Bouzeghoub, M. Fabret, F. Matulovic-Broqué, M. “Modeling Data Warehouse Refreshment Process as a Workflow Application”. DMDW’99, Germany, 1999.

[Bro97] Brooks, P.: “March of the Data Marts”. DBMS magazine, March 1997.

[Cab97] Cabibbo, L. Torlone, R.:”Querying Multidimensional Databases", DBPL’97, 1997.

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

[Cab99] Cabibbo, L. Torlone, R.:”Data Independence in OLAP Systems", SEBD’99, 1999.

[Cal98] Calvanese, D. De Giacomo, G. Lenzerini, M. Nardi, D. Rosati, R.: “Source integration in data warehousing”. Technical Report. 1998.

[Cal99] Calvanese, D. De Giacomo, G. Lenzerini, M. Nardi, D. Rosati, R.: "A Principled Approach to Data Integration and Reconciliation in Data Warehousing". DMDW '99, Germany. 1999.

[Cal01] Calegari, D. Copette, D. Fajardo, F. Ferré, J. Kristic, G. López, R. López, P. Pizzorno, E. Sierra, N. Tobler, F. Viña, C.: "Implementación de herramientas CASE que asistan en el Diseño de Data Warehouses (Definición de Lineamientos)". Software engineering project. Advisor: Jorge Triñanes. Users: Verónika Peralta, Raúl Ruggia. InCo, Universidad de la República, Uruguay. On going work.

[Car97] Carpani, F.Goyoaga, J. Fernández, L.: "Modelos Multidimensionales". JIIO'97. Uruguay, 1997.

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

[Car01] Carpani, F. Ruggia, R.: "An Integrity Constraints Language for a Conceptual Multidimensional Data Model". SEKE'01, Argentina, 2001.

[Cha97] Chaudhuri, S. Dayal, U.: "An overview of Data Warehousing and OLAP Technology". SIGMOD Record 26(1). 1997.

[Cha99] Chan, M. Leong, H. Si, A.: "Incremental Update to Aggregated Information for Data Warehouses over Internet". DOLAP '00, USA. 2000.

[Che76] Chen, P.: "The entity-relationship model- towards a unified view of data", ACM Trans. on Database Systems 1,1. March 1976.

[Doc00] Do Carmo, A.: *"Aplicando Integración de Esquemas en un contexto DW-Web"*. Master's Thesis. Advisor: Regina Motz. Pedeciba, Universidad de la República, Uruguay, 2000.

[Fan95] Frankhauser, P. Motz, R. Huck, G.: "*Schema Integration Methodology"*. Deliverable D4-4/1, IRO-DB(P8629). 1995.

[Fan97] Fankhauser, P.: "A Methodology for Knowledge-Based Schema Integration". PhD-Thesis, Technical University of Vienna, 1997.

[Fra99] Franconi, E. Sattler, U.:"A Data Warehouse Conceptual Data Model for Multidimensional Aggregation. ", DMDW'99, Germany, 1999.

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

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

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

[Gol99] Golfarelli, M. Rizzi, S.:"Designing the Data Warehouse: Key Steps and Crucial Issues", Journal of Computer Science and Information Management. Vol. 2. N. 3, 1999.

[Gol00] Golfarelli, M. Maio, D. Rizzi, S.:" Applying Vertical Fragmentation Techniques in Logical Design of Multidimensional Databases". DAWAK'00, UK, 2000.

[Gup95] Gupta, A. Mumick, I.: "Maintenance of Materialized Views: Problems, Techniques, and Applications". Data Engineering Bulletin, June 1995.

[Gup97] Gupta, H.: "Selection of Views to Materialize in a Data Warehouse". Conf. on Database Theory, 1997.

[Gut99] Gutiérrez, A. Motz, R. Ruggia, R. "Diseño y mantenimiento dinámico de Data Warehouses – Aplicación en el contexto de la Web". JIIO'99, Uruguay, 1999.

[Gut00] Gutiérrez, A. Motz, R. Viera, D.: "Building Databases with Information Extracted from Web Documents". SCCC'00. Chile, 2000.

[Gys97] Gyssens, M. Lakshmanan, L.:"A Foundation for Multi-Dimensional Databases", VLDB'97, Greece, 1997.

[Hah00] Hahn, K. Sapia, C. Blaschka, M.: ”Automatically Generating OLAP Schemata from Conceptual Graphical Models", DOLAP’00, USA, 2000.

[Hai91] Hainaut, J.: ”Entity-Generating schema transformations for Entity-Relationship models”. ER’91, USA, 1991.

[Ham95] Hammer, J. Garcia-Molina, H. Widom, J. Labio, W. Zhuge, Y.: “The Stanford Data Warehousing Project”. Data Eng. Bulletin, 18(2), June 1995.

[Hul96] Hull, R. Zhou, G.: “A Framework for Supporting Data Integration Using the Materialized and Virtual Approaches”. ACM SIGMOD, Canada, 1996.

[Hul97] Hull, R.: “Managing Semantic Heterogeneity in Databases: A Theorical Perspective”. PODS, 1997.

[Hüs00] Hüsemann, B. Lechtenbörger, J. Vossen, G.:"Conceptual Data Warehouse Design". DMDW’00, Sweden, 2000.

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

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

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

[Jar97] Jarke, M. Vassiliou, Y.: “Data Warehouse Quality Design: A Review of the DWQ Project”. Conf on Information Quality, UK, 1997.

[Jar99] Jarke, M. Lenzerini, M. Vassiliou, Y. Vassiliadis, P.: "Fundamentals of Data Warehouses”. Springer-Verlag, 1999.

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

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

[Kim96a] Kimball, R.: “Mastering Data Extraction”. DBMS magazine, June 1996.

[Kim98] Kimball, R. Reeves, L. Ross, M. Thornthwaite, W.:"The Datawarehouse Lifecycle Toolkit". John Wiley & Son, Inc., 1998.

[Kor99] Kortnik, M. Moody, D.: “From Entities to Stars, Snowflakes, Clusters, Constellations and Galaxies: A Methodology for Data Warehouse Design”. ER’99 Industrial Track, France, 1999.

[Lab96] Labio, W. Garcia-Molina, H. Gorelik, V.: Efficient Snapshot Differential Algorithms for Data Warehousing. VLDB’96, Bombay, 1996.

[Lab00] Labio, W. Wiener, J. Garcia-Molina, H. Gorelik, V.: Efficient Resumption of Interrupted Warehouse Loads. SIGMOD’00, USA, 2000.

[Lar00] Larrañaga, I.: “Mapeamiento entre Modelo Estrella y Modelo Relacional”. Internal Report. InCo, Universidad de la República, Uruguay, 2000.

[Lar01] Larrañaga, I.: “Especificación de Primitivas en SQL”. Internal Report. InCo, Universidad de la República, Uruguay, 2001.

[Lar01a] Larrañaga, I.: “Aplicación de PVM a herramientas ETL”. Internal Report. InCo, Universidad de la República, Uruguay, 2001.

[Leh98] Lehner, W. Albrecht, J. Wedekind, H.:”Normal Forms for Multidimensional Databases.", SSDBM’98, Italy, 1998.

[Leh98a] Lehner, W.:"Modeling Large Scale Olap Scenarios.", EDBT’98, Spain, 1998.

[Len97] Lenz, H. Shoshani, A.: "Summarizability in OLAP and Statistical Databases”. Conf. on Statistical and Scientific Databases, 1997.

[Ler90] Lerner, B. Habermann, A.: "Beyond Schema Evolution to Database Reorganization”. OOPSLA/ECOOP, 1990.

[Lev96] Levy, A. Rajaraman, A. Ordille, J.: "Querying Heterogeneous Information Sources Using Source Descriptions". VLDB'96, India, 1996.

[Li96] Li, C. Wang, X.S.:"A Data Model for Supporting On-Line Analytical Processing", CIKM'96, USA, 1996.

[Lig99] Ligouditianos, S. Sellis, T. Theodoratos, D. Vassiliou, Y.: "Heuristic Algorithms for Designing a Data Warehouse with SPJ Views". DaWaK '99, Italy, 1999.

[Lou92] Loucopoulos, P. Zicari, R.: "Conceptual Modeling, databases, and Case". J. Wiley & Sons, Inc. 1992.

[Mar89] Markowitz, V. Shoshani, A.: "On the Correctness of Representing Extended Entity-Relationship Structures in the Relational Model". SIGMOD'89, USA, 1989.

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

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

[Mar01] Marotta, A. Motz, R. Ruggia, R.: "Managing Source Schema Evolution in Web Warehouses". WIIW '01, Brazil, 2001.

[Mat96] Mattison, R.:"Data Warehousing: strategies, technologies and techniques". Mc. Graw Hill, 1996.

[Mcg98] Mc Guff, F.: "Data Modelling for Data Warehouses". http://members.aol.com/fmcguff/dwmodel. 1998.

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

[Mot98] Motz, R.: "Propagation of Structural Modifications to Integrated Schema". ADBIS'98, Poland, 1998.

[Mot98a] Motz, R., Fankhauser, P.: "Propagation of Semantic Modifications to an Integrated Schema". COOPIS'98, USA, 1998.

[Ms00] Microsoft Corp. "OLEDB for OLAP". http://www.microsoft.com/data/oledb/olap/. 2000.

[Mur01] Murialdo, C. Delfino, E. Mussini, C.: "Aplicación para Integración de Herramientas CASE con JAVA y CORBA". Undergraduate project. Advisors: Raúl Ruggia, Fernando Rodriguez. InCo, Universidad de la República, Uruguay. On going work.

[Ola98] OLAP Council. "The APB-1 Benchmark. Release II". http://www.olapcouncil.org/research/bmarkly.htm. 1998.

[Ozs91] Ozsu, M.T. Valduriez, P.: "Principles of Distributed Database Systems". Prentice-Hall Int. Editors. 1991.

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

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

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

[Per01] Peralta, V. Ruggia, R.: "Implementación de herramientas CASE que asistan en el Diseño de Data Warehouses". Technical Report. InCo, Universidad de la República, Uruguay, 2000.

[Per01a] Peralta, V.: "Un caso de estudio sobre diseño lógico de Data Warehouses". Technical Report. InCo, Universidad de la República, Uruguay, 2001.

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

[Rou99] Roussopoulos, N.: "Materialized Views and Data Warehouses". Sigmod Record, Volume 27, 1. 1998.

[Sap99] Sapia, C. Blaschka, M. Höfling, G.: "An Overview of Multidimensional Data Models for OLAP". Technical Report, 1999.

[Sap99a] Sapia, C. Blaschka, M. Höfling, G. Dinter, B.: "Extending the E/R Model for the Multidimensional Paradigm". DWDM'98, Singapure, 1998.

[Sil97] Silverston, L. Inmon, W. Graziano, K.: "The Data Model Resource Book". John Wiley & Sons, Inc. 1997.

[Spa94] Spaccapietra, S. Parent, C.: "View integration: A step forward in solving structural conflicts". TKDE'94, Vol 6, No. 2, 1994.

[Sri99] Srivastava, J. Chen, P.: "Warehouse Creation -A Potential Roadblock to Data Warehousing". IEEE, TKDE, Vol. 11, No. 1, 1999.

[Sta96] Staudt, M. Jarke, M.: "Incremental Maintenance of Extenally Materialized Views". VLDB'96, India, 1996.

[Stu00] Sturm, J.: "Data Warehousing with Microsoft SQL Server". ISBN 0-7356-0859-8. 2000.

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

[The99] Theodoratos, D. Sellis, T.: "Designing Data Warehouses". DKE'99, 1999.

[The99a] Theodoratos, D. Ligoudistianos, S. Sellis, T.: "Designing the Global Data Warehouse with SPJ Views". CAISE'99, Germany, 1999.

[The99b] Theodoratos, D. Sellis, T.: "Dynamic Data Warehouse Design". DaWaK'99, Italy, 1999.

[Tho97] Thomsen, E.:"OLAP Solutions. Building Multidimensional Information". John Wiley & Sons, Inc. 1997.

[Ull88] Ullman.: "Principles of Database and Knowledge-Base Systems", vol 1. Computer Science Press. 1988.

[Vas99] Vassiliadis, P. Sellis, T.: "A Survey on Logical Models for OLAP Databases". Technical Report, DWQ, NTUA, Greece, 1999.

[Vid01] Vidal, V. Lóscio, B. Salgado, A.: "Using correspondence assertions for specifying the semantics of XML-based mediators". WIIW'2001, Brazil, 2001.

[Wid95] Widom, J.: "Research Problems in Data Warehousing". CIKM'95, USA, 1995.

[Wil97] Williams, J.: "Tools for Traveling Data". DBMS magazine, June 1997.

[Wu97] Wu, M. Buchmann, A.: "Research Issues in Data Warehousing". BTW'97, Germany, 1997.

[Yan97] Yang, J. Karlapalem, K. Li, Q.: "Algorithms for materialized view design in data warehousing environment". VLDB'97, Greece, 1997.

[Yan01] Yang, L. Miller, R. Haas, L. Fagin, R.: "Data-Driven Understanding and Refinement of Schema Mappings". ACM SIGMOD'01, USA, 2001.

[Zhu95] Zhuge, Y. Garcia-Molina, H. Hammer, J. Widom, J.: "View Maintenance in a Warehousing Environment". ACM SIGMOD'95, USA, 1995.

[Zhu97] Zhuge, Y. Garcia-Molina, H. Wiener, J. : "Consistency Algorithms for Multi-Source Warehouse View Maintenance". Journal of Distributed and Parallel Databases, July 1997.