Capítulo III - Definición del Esquema Lógico del DW

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 incorporan nociones de almacenamiento y estructuración de los datos. 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.

Hay un componente adicional a tener en cuenta: las bases fuentes. Un DW no es una base de datos para construir desde cero, sino que debe construirse con información extraída de un cierto conjunto de bases fuentes. Durante el diseño lógico deben considerarse las fuentes 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 herramientas, etc.

En este trabajo proponemos un mecanismo para construir el esquema lógico de un DW, mediante transformaciones sucesivas aplicadas a las bases fuentes. En la elección de las transformaciones adecuadas a cada caso intervienen el esquema conceptual, correspondencias entre el esquema conceptual y las fuentes, y lineamientos del diseñador.

La Figura 3 muestra los pasos en el diseño lógico del DW.

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).

En un primer paso el diseñador indica algunos lineamientos que complementan al esquema conceptual con estrategias de diseño lógico (guidelines), por ejemplo: como particionar datos históricos o que 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.

A partir de allí, se va transformando (transformations) el esquema lógico de la fuente, refinándolo sucesivamente (intermediate logical schema), hasta obtener el esquema lógico deseado para el DW (DW logical schema). Se propone un algoritmo y un conjunto de reglas de transformación (rules) para llevar a cabo dicho refinamiento.

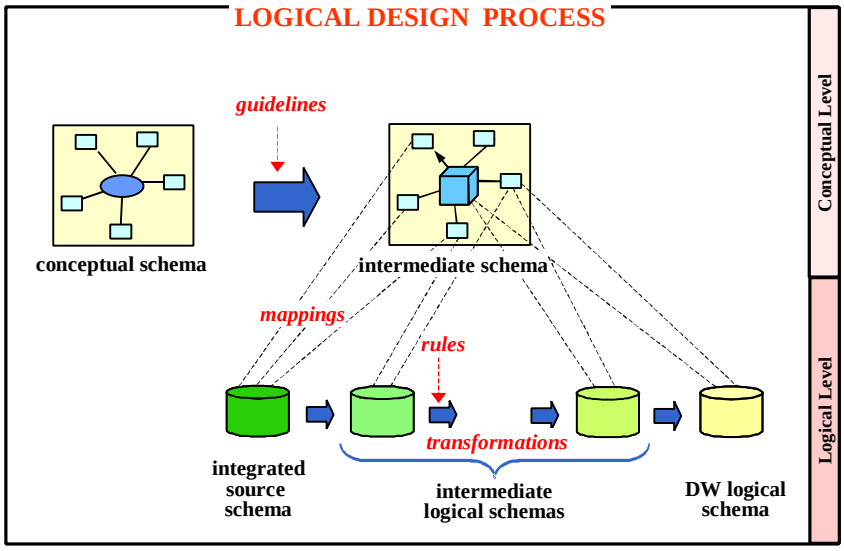

Figura 3 – Proceso de diseño lógico

El diseño lógico puede dividirse en dos etapas, la primera de definición del esquema lógico y la segunda de construcción del mismo.

En la primer etapa se definen los lineamientos y se establecen los mapeos a la base fuente. 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 propiedades que éstos deben cumplir.

En la segunda etapa es lleva a cabo la construcción del esquema lógico del DW a través de transformaciones aplicadas a la fuente. Se utilizan las definiciones realizadas en la primer etapa para elegir que transformaciones aplicar a los datos y en que 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) y no en aspectos operativos o de implementación.

En este capítulo se presenta la primera etapa. Se estudian los distintos componentes del proceso: el esquema conceptual (sección 3), los lineamientos (sección 4), el esquema intermedio (sección 5), la base fuente integrada (sección 6) y los mapeos (sección 7). En el siguiente capítulo se presenta la segunda etapa; se estudian las transformaciones, las reglas y el método propuesto.

2. Notación

Nombres de variables

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

Utilizaremos el prefijo Sch para definir los objetos / conjuntos de objetos principales que se manipularán 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).

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

As denota la cardinalidad del conjunto As.

Expresiones Tipadas

Expresions(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 considera inicialmente los tipos *boolean, integer, float, string* y *date*. BasicTypes se define en el Anexo 1, en la sección 1.4.1. 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*> ∈ Expresions(As)
Si <*E,T*> ∈ As entonces <*E,T*> ∈ Expresions(As)
Si <*E,T*> ∈ Expresions(As) ∧ <*D,U*> ∈ Expresions(As) ∧ *q* ∈ Operators(T,U,V) ∧ V ∈ BasicTypes
entonces <*E q D, V*> ∈ Expresions(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*> ∈ Expresions(As) ∧ <*D,U*> ∈ Expresions(As) ∧ *q* ∈ Operators(T,U,boolean)
entonces *E q D* ∈ Predicates(As)

Expresiones de agregación

GroupExpresions(As) es el conjunto de expresiones tipadas definidas a partir de los elementos del conjunto As, que involucran la agregación de varios elementos.

Como operadores de agregación se utilizan: count, min, max, sum y avg.

Se define inductivamente el conjunto:

Si <*E,T*> ∈ Expresions(As) entonces <*E,T*> ∈ GroupExpresions(As)
Si <*E,T*> ∈ Expresions(As) ∧ entonces <count(*E), integer*> ∈ GroupExpresions(As)
Si <*E,T*> ∈ Expresions(As) ∧ *j* ∈ {min, max} entonces <*j* (*E),T*> ∈ GroupExpresions(As)
Si <*E,T*> ∈ Expresions(As) ∧ *T* ∈ {integer, float} ∧ *j* ∈ {sum, avg}
entonces <*j* (*E),T*> ∈ GroupExpresions(As)

Expresiones de roll-up

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

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

Se define inductivamente el conjunto:

Si <*E,T*> ∈ Expresions(As) ∧ entonces <count(*E), integer*> ∈ RollUpExpresions(As)
Si <*E,T*> ∈ Expresions(As) ∧ *j* ∈ {min, max} entonces <*j* (*E),T*> ∈ RollUpExpresions(As)
Si <*E,T*> ∈ Expresions(As) ∧ *T* ∈ {integer, float} ∧ *j* ∈ {sum, avg}
entonces <*j* (*E),T*> ∈ RollUpExpresions(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 $\equiv$ {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 $\rightarrow$ Bs. Sea Cs $\subseteq$ As

Se define: Cs♦f = {f(c) / c $\in$ Cs}

3. 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 4a muestra la definición gráfica de la dimensión *clientes* en CMDM. El nombre de la dimensión está en la esquina superior izquierda del recuadro amarillo. Los cuadros de texto blanca son los niveles de la dimensión; sus nombres aparecen en negrita. Las jerarquías de niveles se representan por flechas entre los niveles, del nivel con menos agregación (hijo) al nivel con más agregación (padre). La Figura 4b muestra la definición de la relación dimensional *venta* en CMDM. En la relación se cruzan las dimensiones *fechas, vendedores, clientes, artículos* y *cantidades*.

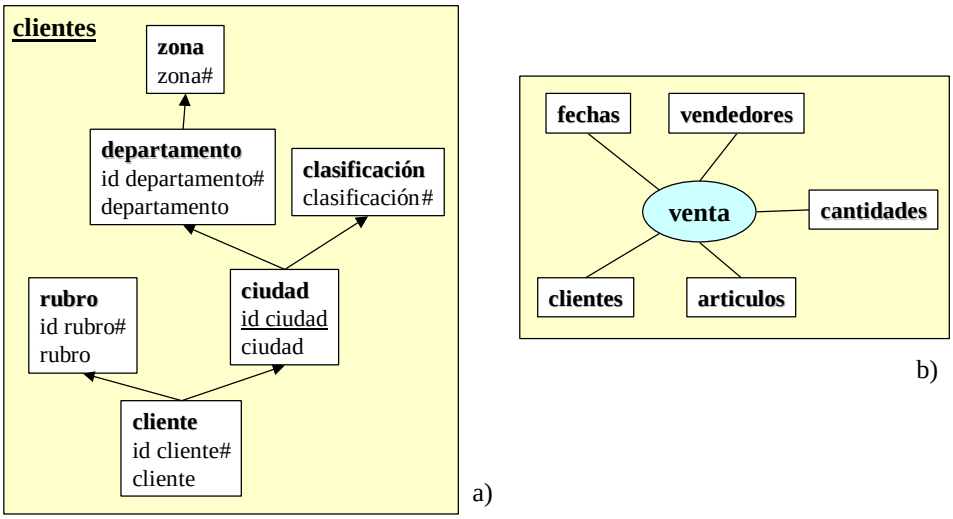

Figura 4 – Notación gráfica de CMDM: a) dimensiones y b) relaciones dimensionales

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 instanciar el modelo a casos particulares.

3.1. Restricción de tipos

CMDM no restringe los tipos que pueden tener los niveles, sin embargo, a la hora de asignarle estructuras de datos (en nuestro caso mapearlo a estructuras relacionales), es necesario acotar dichos tipos. Trabajaremos con niveles compuestos por uno o varios atributos, cada uno de ellos de un tipo simple. A los atributos de los niveles los llamaremos *ítems*.

Gráficamente, los ítems aparecen debajo del nombre del nivel. Los ítems que identifican a 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 4a 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 para instanciar el modelo haremos una restricción al tipo de datos de los niveles. En el Anexo 1 se presenta la especificación del modelo CMDM y nuestra restricción de tipos.

A continuación presentamos la especificación de CMDM[3] luego de dicha restricción:

- CONCEPTUALSCHEMAS ≡ { <NAME, LEVS, DIMS, RELS, CONSTS > /
  NAME ∈ STRINGS ∧
  LEVS ∈ LEVELS ∧
  DIMS ⊆ DIMENSIONS(LEVS) ∧
  RELS ⊆ RELATIONS(DIMS) ∧
  CONSTS ⊆ FORM } [4]
- 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 } [5]
- 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

[3] Las estructuras de datos presentadas se construyen parseando la especificación de un esquema conceptual de CMDM.

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

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

3.2. Otras restricciones

Además de la restricción en el tipo de los niveles, impondremos algunas condiciones más al esquema conceptual para que se encuentre en nuestras hipótesis de trabajo. Eventualmente será necesario que el diseñador redefina algunas de sus estructuras para cumplir con nuestras condiciones.

Disjuntez 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 no es cómodo para manipularlos. Este problema es fácilmente solucionable definiendo ítems diferentes.

Por ejemplo, en la dimensión *productos* de la Figura 5a. Tanto el nivel *familia* como el nivel *producto* tienen un ítem *descripción*. La Figura 5b muestra la dimensión *productos* utilizando dos ítems diferentes para representar las descripciones: *desc-familia* y *desc-producto*.

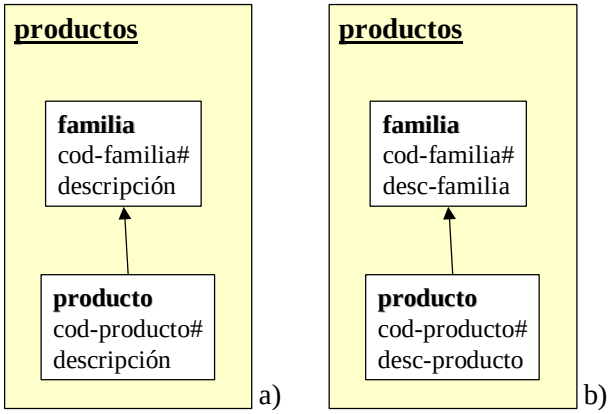

Figura 5 – Definición de niveles: a) compartiendo ítems, b) con ítems diferentes

Se exige que cada ítems forme parte de a lo sumo un nivel. Análogamente, se exige que un nivel esté en a lo sumo una dimensión.

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

Esta restricción nos permitirá 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 es análogo al concepto de entidad débil en el modelo E/R [Che76].

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

El diseñador eventualmente tendrá que rever el esquema conceptual y definir las claves faltantes.

Para los niveles que no tienen clave absoluta, podemos 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)

Definición 2 – LevelKey

4. Lineamientos

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.

Los lineamientos permiten elegir el estilo de diseño para el DW (snowflakes, estrella, mixto), indicar requerimientos de performance y almacenamiento (indicando que cubos implementar), etc.

En todos los casos son información adicional ingresada por el diseñador en forma explícita.

A continuación se describen algunos lineamientos propuestos, el conjunto de lineamientos puede extenderse para incorporar más información de diseño.

4.1. Materialización de Relaciones

En el esquema conceptual una relación dimensional representa un espacio de cubos resultado de cruzar algunos niveles de las dimensiones. Dicho espacio de cubos puede restringirse con las restricciones de integridad del propio modelo. Estas restricciones sugieren qué cruzamientos sería interesantes tener, y cuáles no debieran existir.

Sin embargo, la decisión de cuáles de esos cubos se materializan debe ser tomada en un momento posterior. Entendemos por materializar, al hecho de precalcular los valores para los cruzamientos de ese cubo, y almacenarlos en una relación. Luego se podrán obtener otros cruzamientos mediante operaciones efectuadas sobre éstos.

En CMDM los cubos son restricciones a las relaciones dimensionales, expresadas en el lenguaje de restricciones del modelo. Un cubo se define mediante una macro que indica los niveles y la medida del cubo ([Car00]). Los cubos se usan en predicados con cuantificadores ($\forall$, $\exists$, $\neg$) para indicar que *"todos los cubos tienen"*, o *"debe existir un cubo que tenga"* o *"ningún cubo debe tener"* dicha estructura.

Un cubo debe tener al menos un nivel de cada dimensión. De algunas dimensiones podría quererse más de un nivel de detalle. Esto último tiene sentido si los niveles pertenecen a diferentes jerarquías, pero no se restringe.

Un esquema conceptual que no tenga cubos es válido en CMDM, sin embargo, a nivel lógico el diseñador tendrá que materializar de alguna forma las relaciones dimensionales del esquema. Es por ello que necesitamos una definición de cubos más 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.

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

La definición de que cubos implementar se hará por extensión. A ese conjunto de cubos a materializar le llamaremos SchCubes.

A continuación describimos la estructura que deberán tener los dichos cubos. Además del conjunto de niveles y la medida (opcional) con que se representan en CMDM daremos un nombre al cubo, y mantendremos una referencia a la relación dimensional que materializan. Estos agregados son por motivos puramente operacionales.

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

Definición 3 – 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 los representamos con un cubo unido a varios niveles (rectángulos blancos) que conforman el nivel de detalle. Las medidas son los niveles marcados por una flecha. Dentro del cubo está su nombre y la relación que materializa entre paréntesis. La Figura 6 muestra la materialización *venta-1* de la relación dimensional *venta* presentada en la Figura 4. Tiene por niveles: *mes, artículo, cliente, vendedor* y *cantidades*, y por medida: *cantidades*.

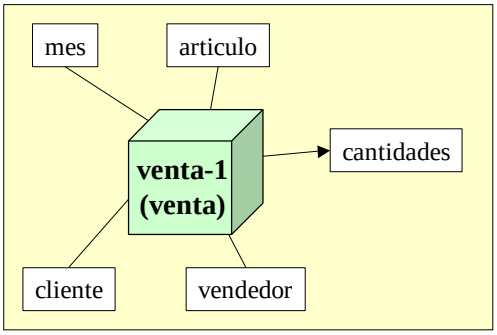

Figura 6 – Materialización de relaciones (Cubos)

4.2. Fragmentación de Dimensiones

El diseñador puede indicar el grado de normalización que quiere lograr al traducir cada dimensión al modelo relacional. Por ejemplo, puede querer un esquema estrella, es decir, denormalizar todas las dimensiones y mantener fact tables. Por el contrario, puede querer un esquema de snowflakes, normalizando todas las dimensiones.

Puede querer tratar diferente cada dimensión, indicando para cada una si normaliza, denormaliza o efectúa una estrategia intermedia, indicando en este último caso, qué niveles se guardarán juntos.

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

Para ello definirá por extensión una función que a cada dimensión le haga corresponder una fragmentación de sus niveles. A esa función le llamaremos SchDFragmentation.

Para que una fragmentación tenga sentido, los niveles de cada fragmento deben estar relacionados jerárquicamente. Es decir, las relaciones padre-hijo de los niveles de un fragmento deben conformar un grafo 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 4a, no tiene sentido almacenar juntos los niveles *rubro* y *ciudad*, ya que no hay dependencias 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.

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 4 – SchDFragmentation

Un fragmento es un subconjunto de los niveles de la dimensión, unidos por relaciones padre-hijo del orden parcial de la dimensión. 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.

Si no se quiere duplicar el almacenamiento de información de cada nivel los fragmentos deben ser disjuntos, pero no es una exigencia. El diseñador decidirá cuando duplicar información de acuerdo a su estrategia de diseño.

Gráficamente representamos una fragmentación como una coloración de los niveles. Los niveles de un mismo fragmento se recuadran con el mismo color. Un nivel tendrá varios colores si está en más de un fragmento. La fragmentación es completa si todos los niveles tienen color. En la Figura 7a se muestra una fragmentación de la dimensión *clientes*, presentada en la Figura 4a. La fragmentación tiene 3 fragmentos disjuntos: *rubro y cliente* (celeste), *zona, departamento y ciudad* (verde) y *clasificación* (rosa). Los fragmentos de la Figura 7b no son disjuntos ya que el nivel *ciudad* pertenece al fragmento verde y al rosa.

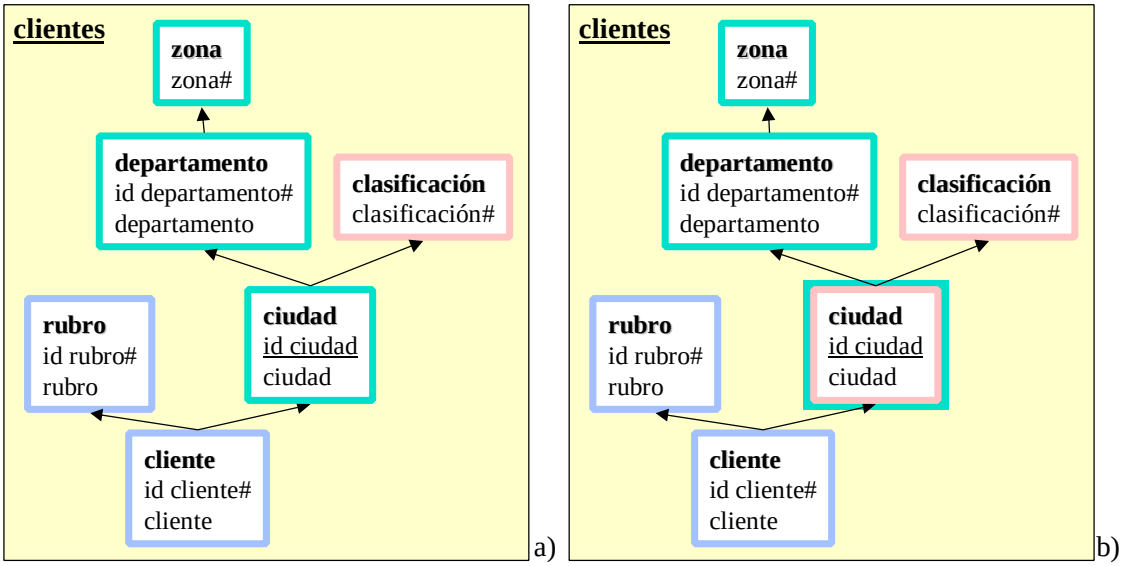

Figura 7 – Fragmentación de dimensiones

Abreviaciones:

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

- SCHFRAGMENTS ≡ SCHDIMENSIONS ♦ SCHDFRAGMENTATION

4.3. Fragmentación de Cubos

El traducir un cubo al 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 consideremos el cubo venta-1 de la Figura 6, y supongamos que las consultas más frecuentes son para los dos últimos años. Se pueden almacenar en un fragmento las tuplas correspondientes a ventas posteriores a ene-2000, y en otro fragmento las que corresponden a meses anteriores.

La fragmentación horizontal es estudiada por Özsu y Valduriez en [Ozs91]; utilizaremos varios de sus resultados.

Cada fragmento contiene un subconjunto de las tuplas de la tabla. Se debe asegurar que cada tupla esté en algún fragmento (condición de completitud). También es interesante pedir que cada tupla esté en un único fragmento (condición de disjuntez) pero esto no siempre es deseable, ya que a veces se quiere mantener redundancia para ganar eficiencia.

Para llevar a cabo la fragmentación se consideran predicados sencillos 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 $q \in \{ =, <, \neq, \leq, >, \geq \}$. Esto podría extenderse a predicados más complejos.

Özsu y Valduriez definen otros predicados, que llamaremos bandas[6], que son la conjunción de predicados simples o sus negaciones, y definen un método que las construye a partir de los predicados simples. Las bandas construidas con su método cumplen las condiciones de completitud y disjuntez.

Para definir una fragmentación el diseñador debe indicar el conjunto de bandas a utilizar. Las bandas estarán expresadas en términos de los ítems de los niveles del cubo, y serán traducidas a atributos de tablas en una etapa posterior. El diseñador puede utilizar bandas construidas con el método de [Ozs91] o construirlas con sus propios criterios. En el último caso debe garantizar que al menos se cumple la propiedad de completitud, para no perder ningún elemento de la instancia del cubo.

En lo que sigue se asumirá que BANDS(IS) es el conjunto de bandas que se pueden definir sobre el conjunto de ítems IS y que la función BANDCOMPLETE(BS) indica si el conjunto de bandas BS es completo.

A continuación se presenta la definición de fragmentación de cubos:

- SCHCFRAGMENTATION $\in \{C / C \in$ SCHCUBES$\} \rightarrow$ BANDSETS(C) $\}$
- BANDSETS(C) $\equiv \{$BS / BS $\subseteq$ BANDS (C.LS$\blacklozenge$IS) $\wedge$ BANDCOMPLETE(BS) $\}$

Definición 5 – SchCFragmentation

BandSets son los posibles conjuntos de bandas completas que se construyen con los ítems de los niveles del cubo. Una fragmentación de cubos es una función que para cada cubo indica el conjunto de bandas en que se fragmenta.

Gráficamente la fragmentación se representa como una como un bloque de llamada a partir del cubo. Dentro del bloque se escriben las bandas. La Figura 8 muestra un conjunto de bandas asociado al cubo venta-1, que fue definido en la Figura 6. Las dos primeras bandas determinan las ventas posteriores a enero de 2000 del vendedor Pérez y del resto de los vendedores (menos Pérez). La tercera banda determina las ventas anteriores a ene de 2000 de cualquier vendedor.

[6] En [Ozs91] le llaman minterm predicates.

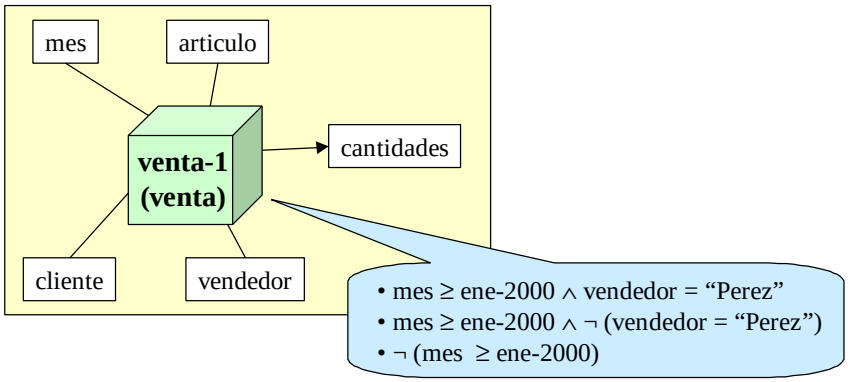

Figura 8 – Fragmentación de cubos

Abreviaciones:

En lo sucesivo se llama SchBands a todas las bandas de todos cubos:

- SCHBANDS ≡ SCHCUBES ♦ SCHCFRAGMENTATION

5. Esquema Intermedio

Al especificar las estrategias, el diseñador introduce elementos que complementan el esquema conceptual. En concreto, especifica que 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, al que llamaremos esquema intermedio, para incorporar elementos de las estrategias:

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

Definición 6 – IntermediateSchema

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

Se utiliza el término SchObjects (objetos del esquema intermedio) para referirse a ítems, niveles, dimensiones, fragmentos o cubos del esquema intermedio.

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

Definición 7 – SchObjects

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 será de utilidad 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 8 – 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 será de utilidad 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[7]
  - Dado F ∈ SCHFRAGMENTS, OBJECTKEYITEMS (F) = BOTTOMLEVELS(F)♦LEVELKEY
  - Dado C ∈ SCHCUBES, OBJECTKEYITEMS (C) = C.LS♦LEVELKEY

Definición 9 – ObjectKeyItems

Se define una función para obtener los ítems relevantes del objeto, según el caso los que lo componen o que lo identifican:

- 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)
  - Dado D ∈ SCHDIMENSIONS, OBJECTITEMS (D)= OBJECTPARTITEMS(L)
  - Dado F ∈ SCHFRAGMENTS, OBJECTITEMS (F) = OBJECTPARTITEMS(L) ∪ OBJECTKEYITEMS(F)
  - Dado C ∈ SCHCUBES, OBJECTITEMS (C) = OBJECTKEYITEMS(C)

Definición 10 – ObjectItems

[7] La función BottomLevels devuelve el conjunto de niveles que están por debajo en las jerarquías de la dimensión. Está descrita en el anexo 2.

6. Bases fuentes

Se trabajará con una base relacional integrada. Interesa abstraer las tablas, sus atributos (con sus tipos) y su clave primaria.

Dadas dos tablas, interesa reflejar como se vinculan, es decir, como se debe realizar el join entre ellas. A un vínculo de este tipo lo llamamos *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 9 muestra varias tablas fuentes y links definidos entre ellas con predicados de igualdad.

En el Anexo 3 se estudia la definición de links, se presentan algunas ambigüedades y mecanismos para solucionarlas.

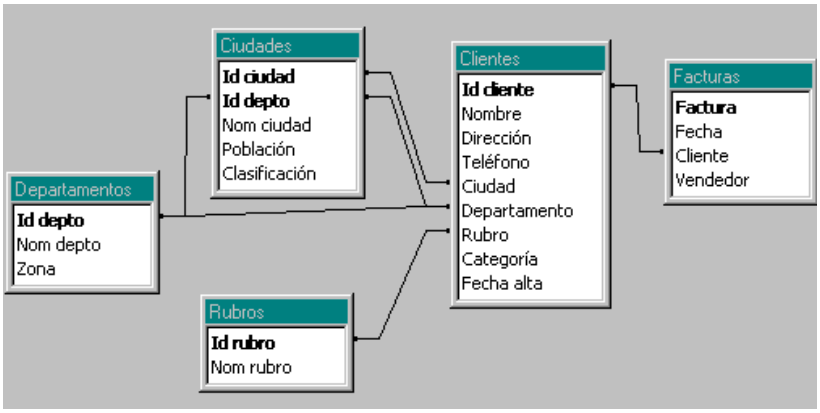

Figura 9 – Links entre tablas fuentes.

A continuación se presenta una abstracción de las bases fuentes:

- SOURCESCHEMA ≡ < SCHATTRIBUTES, SCHTABLES, SCHLINKS >
- SCHATTRIBUTES ⊆ { <ATTNAME, TYPE > / ATTNAME ∈ STRINGS ∧ TYPE ∈ BASICTYPES }
- SCHTABLES ⊆ { <TABNAME, AS, PK > / TABNAME ∈ STRINGS ∧ AS ⊆ SCHATTRIBUTES ∧ PK ⊆ AS }
- SCHLINKS ∈ $\{T_1 / T_1 \in \text{SCHTABLES}\} \times \{T_2 / T_2 \in \text{SCHTABLES}\} \rightarrow$ PREDICATES ($T_1 \bullet$AS $\cup$ $T_2 \bullet$AS) [8]

Definición 11 – SourceSchema

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 relación, y un conjunto de links entre las tablas.

[8] El operador • concatena el nombre de la tabla y del atributo, y mantiene el tipo del atributo (ver sección 2)

7. Mapeos

El esquema conceptual especifica la información que contendrá el DW, y a través de los lineamientos, el diseñador explicitó las características que debe cumplir el esquema lógico; y con eso se construyó el esquema intermedio. El siguiente paso es vincular el esquema intermedio 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 fuente. Estas funciones son definidas por el diseñador en forma explícita.

Antes de definir los mapeos se presentará una caracterización de las expresiones a las que se mapea.

7.1. Expresiones de Mapeo

En un mapeo, a cada ítem del esquema intermedio le corresponderá una expresión construida en base a atributos de las tablas fuentes, a la que llamaremos **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 una totalización que involucra varios atributos de varias tuplas (NcalcME) o algo 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 sección 2) construida en base a uno o más atributos de las tablas fuente.
- Tab: es una tabla fuente.
- Patt: es un atributo de una tabla fuente con el nombre de la tabla como prefijo. En general se usa el operador • para concatenar el nombre de la tabla.
- Ind: es un indicador de como se llenarán los valores (para ExternME), pudiendo ser con una constante, una marca de tiempo o dígitos de versión.

La definición de una expresión de mapeo 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 ∈ EXPRESIONS(PATTS) }
- NCALCME ≡ { <EXPR, TAB, PATTS > / TAB ∈ SCHTABLES ∧ PATTS ⊆ TAB•AS ∧ EXPR ∈ GROUPEXPRESIONS(PATTS) }
- EXTERNME ≡ { <EXPR, IND> / EXPR ∈ EXPRESIONS(∅) ∧ IND ∈ {CONSTANT, TIMESTAMP, VERSION}

Definición 12 – MapExpr

Las expresiones de mapeo son la unión de los distintos tipos de expresiones.

7.2. Funciones de Mapeo

Habiendo definido MapExpr, se pueden definir funciones de mapeo:

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

Definición 13 – 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 para 2 fines: vincular un fragmento de una dimensión a las tablas fuentes que poblarán dicho fragmento (mapeos de fragmentos), y vincular los cubos a las tablas fuentes (mapeos de cubos).

Representamos gráficamente una función de mapeo, como vínculos (líneas) entre los ítems del modelo conceptual 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 10 se muestra una función de mapeo para la dimensión *geografía* (con un único fragmento). 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.

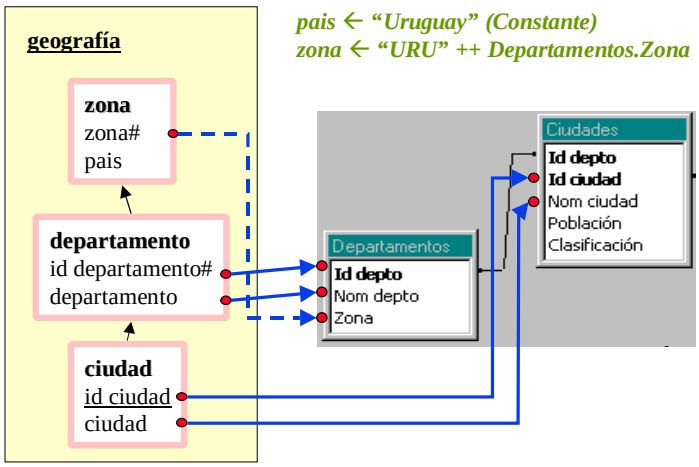

Figura 10 – Representación gráfica de una función de mapeo

Inicialmente se tendrá una función de mapeo para cada fragmento de dimensión, y una función de mapeo para cada cubo. Al ir fragmentando los cubos (según SchCFragmentation) se tendrá una función para cada banda de cubo.

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 10, el nivel *zona* podría tener una restricción indicando que sólo interesan las zonas menores a 10, que podrían 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"

Gráficamente representamos las condiciones como llamadas (callouts). La Figura 11 muestra la incorporación de las condiciones al mapeo de la Figura 10.

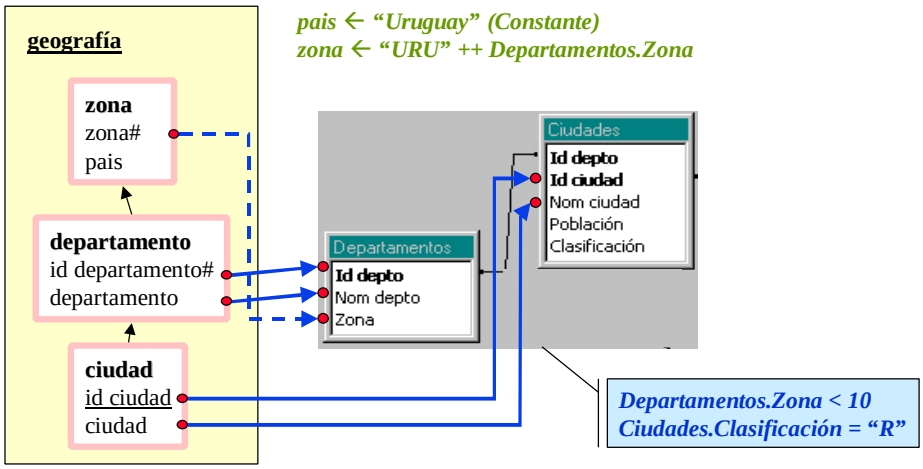

Figura 11 – Representación gráfica de condiciones de mapeo

Formalmente, las condiciones son predicados (ver sección 2) 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 modelo conceptual y traducirlas en término de los atributos a los que mapean. Un parser de restricciones del modelo conceptual puede ayudar a realizar esta tarea. El diseñador además debe basarse en su conocimiento de las fuentes.

7.3. 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 que 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:

Mapeo a atributos

Dada una función de mapeo, se dice que un conjunto de ítems mapea a un atributo, 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 $F \in$ MAPPINGS (ITEMS,TABLES), $Is \subseteq$ ITEMS

$$\text{MAPATTRIBUTES}(F, Is) = \{A / \exists I \in Is . (F(I) \in \text{DIRECTME} \wedge F(I).\text{TAB}\bullet A = F(I).\text{PATT})\} \cup \{A / \exists I \in Is . (F(I) \in \text{1CALCME} \wedge \exists T \in F(I).\text{TAB} . (T\bullet A \in F(I).\text{PATT}))\}$$

Definición 14 – MapAttributes

Mapeo a tablas

Dada una función de mapeo, se dice que un conjunto de ítems mapea a una tabla si mapea a uno de sus atributos.

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 15 – MapTables

7.4. Mapeo de Fragmentos

Para mapear un fragmento, debe definirse una función de mapeo para todos sus ítems, esto incluye algunos ítems de niveles superiores que formen parte de la clave de los niveles del fragmento. 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 ∈ SCHDIMENSIONS♦SCHDFRAGMENTATIONS}→
  {<MAP, COND> / MAP ∈ MAPPINGS (F♦IS)
  ∧ COND ∈ PREDICATES ({T•AS / T ∈ MAPTABLES(MAP, F♦IS)}) }

Definición 16 – SchFMappings

La función SCHFMAPPING 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 fuentes.

Cada mapeo de fragmento debe pensarse como una consulta SQL que selecciona los atributos mapeados de las tablas correspondientes imponiendo el predicado como condición de filtrado y los links entre las tablas como condición de join. Las tablas involucradas deben joinear entre sí, no tienen sentido los productos cartesianos. La instancia de esa consulta SQL 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 11 puede mapearse al atributo *Id-ciudad* de la tabla *Ciudades* pero también puede mapearse al ítem *Id-ciudad* de la tabla *Clientes*. En la Figura 9 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 que atributo mapear dependerá de que ciudades deben poblar el DW: todas o en las que viven clientes.

Un problema peor ocurre 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 11 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. El considerar el mapeo de fragmentos como una consulta SQL ayuda en la elección de los ítems adecuados.

7.5. Mapeo de Cubos

Análogamente, para mapear un fragmento puede definirse una función de mapeo para sus ítems y una condición sobre las tablas mapeadas. Pero el mapeo de un cubo puede hacerse también en términos de un cubo ya definido aplicando drill-up en una o más dimensiones.

Para definir un cubo en términos de otros se debe especificar la lista de dimensiones por las que se hace drill-up y las operaciones de roll-up asociadas a las medidas.

Para que un cubo C1 pueda ser expresado como drill-up de otro cubo C2 debe cumplirse:

- C2.Measure = C1.Measure */* tienen las mismas medidas */*
- $\forall L \in C2.Ls . ((L \in C1.Ls) \vee (\exists B \in C1.Ls . (<L,B> \in \text{LEVDIMENSION}(L).PO))$ */* los niveles son los mismos o mayores en la jerarquía de una dimensión */*

Se utilizará la función de mapeo del cubo original para los ítems de los niveles en común, pero la función se debe definir para los ítems en los que diferencian. En particular alcanza con mapear el drill-up por cada dimensión, es decir, dar una función que mapee los ítems del nivel original en los ítems del nuevo nivel.

Por ejemplo: si un cubo C1 tiene como detalle *cliente*, y un cubo C2 se mapea en función del cubo C1 con detalle *ciudad*, se debe indicar una función que mapee a los ítems clave de los niveles *cliente* y *ciudad*.

Un drill-up se define como una cuádrupla consistente de un nivel a reemplazar, un conjunto de niveles con mayor jerarquía, un mapeo de los ítems clave de dichos niveles y una operación de roll-up:

- $\text{DRILLUPS}(\text{CO}, \text{CN}) \equiv \{<\text{LOLD}, \text{LNEWS}, \text{MAP}, \text{RUP}> / \text{LOLD} \in \text{CO.LS} \wedge \text{LNEWS} \subseteq \text{CN.LS}$
  $\wedge \text{LNEWS} \subseteq \text{LEVDIMENSION}(\text{LOLD}).\text{LS}$
  $\wedge \forall L \in \text{LNEWS} . (<L, \text{LOLD}> \in \text{LEVDIMENSION}(\text{LOLD}).\text{PO}$
  $\wedge \text{MAP} \in \text{MAPPINGS}(\text{OBJECTKEYITEMS}(\text{LOLD}) \cup \text{LNEWS} \blacklozenge \text{OBJECTKEYITEMS})$
  $\wedge \text{RUP} \in \text{ROLLUPEXPRESIONS}(\text{CN.MEASURE}) \}$

Definición 17 – DrillUps

Se define una función que a cada cubo le hace corresponder o bien una función de mapeo y una condición, o bien un conjunto de drill-ups:

- $\text{SCHCMAPPINGS} \in \{C / C \in \text{SCHCUBES}\} \rightarrow \{<\text{MAP}, \text{COND}, \text{RUP}> /$
  $\text{MAP} \in \text{MAPPINGS} (\text{C.LS} \blacklozenge \text{IS}) \wedge \text{RUP} \in \text{ROLLUPEXPRESIONS}(\text{C.MEASURE})$
  $\wedge \text{COND} \in \text{PREDICATES} (\{\text{T} \bullet \text{AS} / \text{T} \in \text{MAPTABLES}(\text{MAP}, \text{C.LS} \blacklozenge \text{IS})\}) \}$
  $\cup \{ <\text{CUB}, \text{DUPS}> / \text{CUB} \in \text{MAPPINGS} \wedge \text{DUPS} \subseteq \text{DRILLUPS}(\text{C}, \text{CUB}) \}$

Definición 18 – SchCMappings

Si el mapeo de un cubo se hace explícito, la función SCHCMAPPING le asocia una función de mapeo con una condición y una función de roll-up por defecto. Si se hace en términos de otro cubo, le asocia un cubo y un conjunto de expresiones de drill-up.

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

Gráficamente un mapeo de cubo por extensión se representa como 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 15 muestra una función de mapeo para el cubo venta-1.

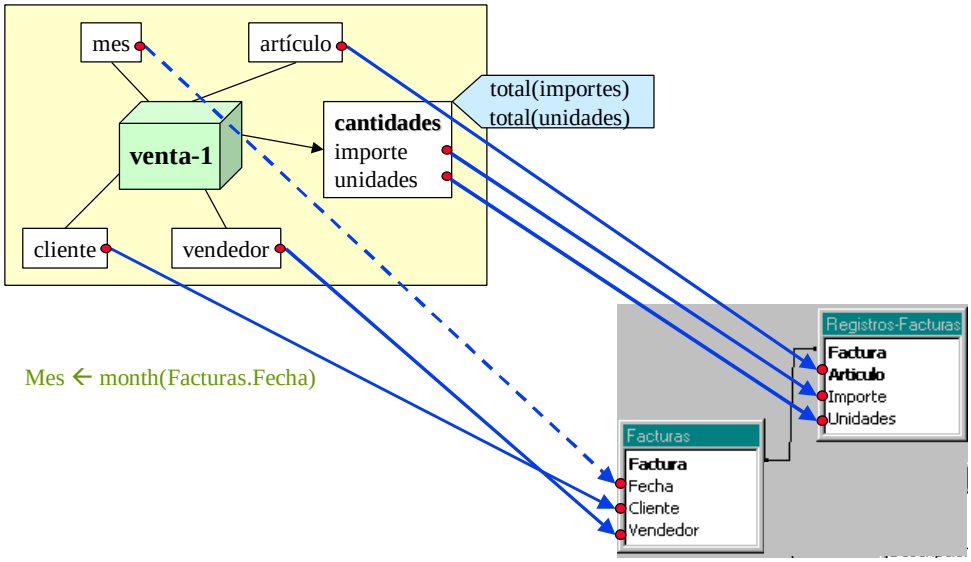

Figura 12 – Representación gráfica de un mapeo explícito de cubo

Un mapeo en términos de otro cubo se representa uniendo los cubos por una flecha gruesa. Se expresa la función de mapeo entre los ítems viejos y nuevos, de manera similar a un mapeo explícito. Mediante un pentágono se especifican los predicados de roll-up para las medidas. La Figura 13 muestra el mapeo del cubo venta-2 en función del cubo venta-1.

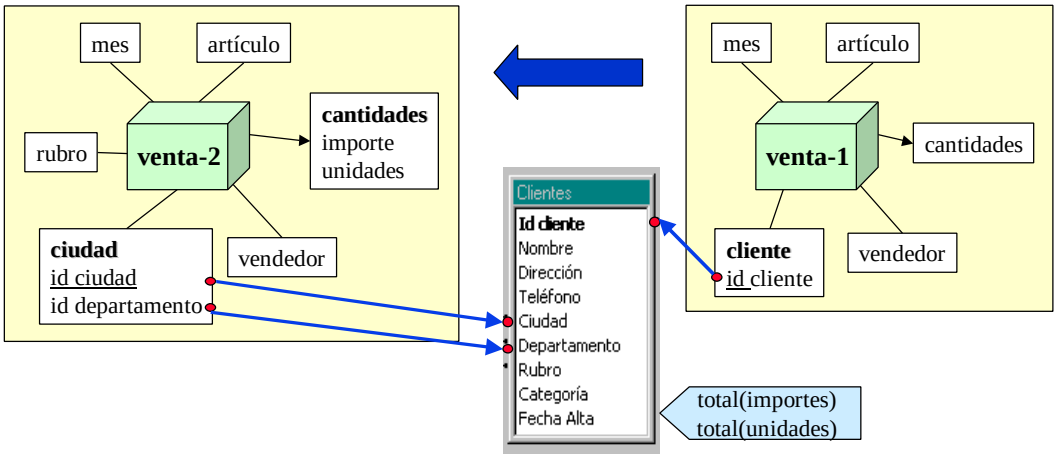

Figura 13 – Representación gráfica de un mapeo de cubo en función de otro cubo

El diseñador debe definir por extensión el mapeo de cada cubo.

Los mapeos explícitos 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 joinear la menor cantidad posible de tablas por performance. Es decir, que siempre que se tengan definidas claves foráneas, o se tenga confianza en la calidad de los datos, se puede evitar joinear algunas tablas. En el ejemplo de la Figura 13 se mapea el ítem *id-cliente* del nivel *cliente* al atributo *cliente* de la tabla *Facturas* en lugar de joinear con la tabla *Clientes* y mapearlo allí. La instancia de esa consulta SQL debe coincidir con la instancia esperada para el cubo.

8. Conclusiones

En este capítulo se presentó la primera etapa del proceso de diseño lógico: la definición del esquema lógico.

En esta etapa el diseñador tiene que realizar las siguientes tareas:

- Analizar el esquema conceptual (resultado del diseño conceptual) y si es necesario adaptarlo para que sirva de entrada para el diseño lógico. En la adaptación se deben definir tipos adecuados para los niveles (concepto de ítem), asegurar que los niveles no compartan ítems y las dimensiones no compartan niveles (disjuntez de elementos) 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 (bandas).
- Analizar las bases fuentes y parsearlas. Dependiendo del manejador utilizado, se deberá 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 / cubo). El mapeo de un cubo se puede expresar como drill-up de otro cubo ya mapeado. En el momento de definir las funciones se estudian las restricciones del esquema conceptual y de las bases fuentes de manera de reflejarlas en los mapeos.