Modelización del Pasaje del Esquema Conceptual al Esquema Lógico de Data Warehouses

Verónika Peralta

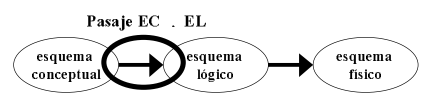
Junio 2001

Contenido

- Introducción.
- Definiciones.
- Reglas.
- Método.
- Conclusiones.

Introducción

Motivación

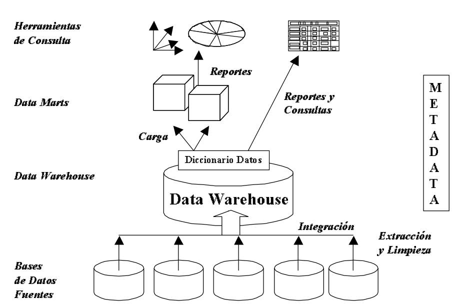


- **Diferencias con BD tradicionales:**
  - Prioridad:
    - » Performance vs. redundancia.
  - Elementos importantes:
    - » Esquema conceptual.
    - » Bases de datos fuentes.
    - » Correspondencias entre ellos.

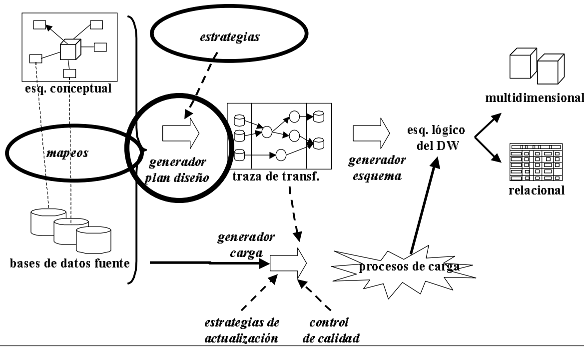
Objetivo

- **Definir un método (¿o metodología?)**
  - De pasaje entre el esquema conceptual y el esquema lógico de un DW.
  - Debe permitir:
    - » Que el diseñador aplique cualquier estrategia de diseño.
    - » Transformar los esquemas y las instancias.
    - » Dar facilidades para la posterior carga de los datos.
  - Se quiere un procedimiento semi-automático.
- **Prototipar una herramienta CASE.**

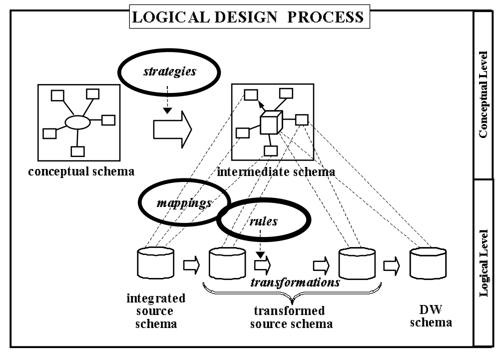
En la arquitectura...



Proceso Global de Diseño

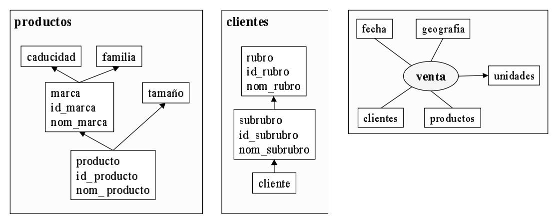


Proceso de Diseño Lógico



Definiciones

Esquema Conceptual



- Se usa CMDM:
  - » Niveles, jerarquías, dimensiones, relaciones dimensionales, medidas.
  - » Restricciones.

Esquema Conceptual

- ◆ **CMDM es “general”:**
  - – Al pensar el esquema lógico debe sacrificarse la generalidad.
    - » Se restringe CMDM.
- ◆ **Restricciones a CMDM:**
  - – Restricciones de Integridad:
    - » Lenguaje de restricciones muy expresivo.
    - » Difícil parsear todas las restricciones posibles.
    - » Se usan algunas restricciones “frecuentes”.
      - ◆ Ej: Claves, Restricciones de instancias (edad > 18).

Esquema Conceptual

- ◆ **Restricciones a CMDM:**
  - – Tipos de los niveles:
    - » Restringimos a producto cartesiano de tipos simples.
      - ◆ Item: componente de un nivel.
  - – Jerarquías de niveles:
    - » Exigimos que haya un único nivel inferior.
  - – Claves de niveles:
    - » CMDM permite:
      - ◆ Claves que lo identifican en toda la dimensión.
      - ◆ Claves que lo identifican respecto al padre (débil).
      - ◆ Eventualmente sin clave.
    - » Exigimos la existencia de claves.

Esquema Conceptual

- **Funciones:**
  - Definimos algunas funciones:
    - » Clave de un nivel: id nivel más id niveles superiores.
    - » Nivel más bajo.
    - » Niveles inferiores.

Esquema Conceptual

Items .{ <ItemName, Type>/ ItemName . Strings . Type .simpletypes}

ConceptualSchema . <SchItems,SchLevels,SchDimensions,SchRelations>

SchItems . Items
SchLevels . { <LevelName, Is, Consts > / LevelName . Strings .
Is . SchItems . Consts . form}
SchDimensions . { <DimName, Ls, Po, Consts > / DimName . Strings .
Ls . SchLevels . Po . PartialOrders(Ls) . Consts . form }
SchRelations . { <RelName, Ds, Consts > / RelName . Strings .
Ds . SchDimensions . Consts . form }

Estrategias (ex Lineamientos)

- **Información adicional al esquema conceptual**
  - Lo complementan con conceptos de diseño lógico.
- **Objetivos:**
  - Elegir el estilo de diseño: snowflake, estrella, etc.
  - Indicar requerimientos de performance y almacenamiento.
  - Indicar estrategias: mantener versiones, temporalizar.

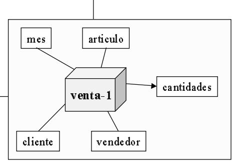
Estrategias

- **Materialización de Relaciones:**
  - Cubos: son restricciones a las relaciones.
    - » Se puede indicar la "intención" de tener algún cubo.
    - » Se puede sólo indicar:
      - cosas que no se quieren (¬),
      - cosas que se deben cumplir siempre ( )
      - o alguna vez ( ).
    - » Puede no indicarse nada.
  - En el diseño lógico tiene que hacerse explícito.
    - » Se debe especificar que cubos existirán.

Estrategias

- **Materialización de Relaciones:**
  - Qué cubos se van a implementar.
    - » Qué niveles y que medidas se eligen.

SchCubes . { <CubeName, R, Ls, Measure> /
CubeName . Strings .
R . SchRelations .
Ls . R.Ds .Ls .
Measure . (Ls . .)}



Estrategias

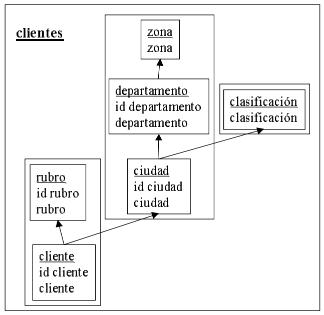
- **Particiones de Dimensiones:**
  - Qué niveles se almacenan juntos.
    - » Son particiones de las dimensiones.
    - » Normalizar, denormalizar, estrategia intermedia.

SchDimensionPartition . {D / D . SchDimensions}→LevelParts(D)

LevelParts (D) . {Ps / Ps . LinkedLevels (D.Ls) . Ps . DisjuntPartition (D.Ls) }
LinkedLevels (D) . {Ls / Ls . D.Ls . A,B . Ls (
(<A,B> . D.PO . . C . D.Ls / C . Ls . <A,C> . D.PO . <C,B> . D.PO) .
(<B,A> . D.PO . . C . D.Ls / C . Ls . <B,C> . D.PO . <C,A> . D.PO))}

Estrategias

- **Particiones de Dimensiones:**



Estrategias

- **Particiones de Cubos:**
  - Cómo particionar cubos.
    - » Particiones horizontales.

SchCubePartition . {C / C . SchCubes} → BandSets(C) }

BandSets(C) . {Bands / Bands . Conditions (C.Ls .Is) . OR(Bands) = TRUE}

Ejemplo

fecha . '01-01-2000'
fecha < '01-01-2000'

Estrategias

- **Conservación de la Historia:**
  - Combinación de diferentes estrategias:
    - » Agregar dígitos de versión a items.
    - » Agregar items de marcas de tiempo.
    - » Agregar items.
    - » Agregar items a la clave del nivel.

SchHistorization . <DigItems, TimeITems, NewItems, KeyItems> /
DigItems . Levs → Set(Items) .
TimeItems . Levs → Set(Items) .
NewItems . Levs → Set(Items) .
KeyItems . Levs → Set(Items)

Esquema Intermedio

- **Se extiende el modelo conceptual agregando:**
  - Nuevos items para mantener la historia.
  - Cubos que materialicen las relaciones.
  - Particiones en las dimensiones.
  - Particiones en los cubos.

Esquema Intermedio

IntermediateSchema . < SchItems, SchLevels, SchDimensions, SchCubes, SchDimensionPartition, SchCubePartitions, SchHistorization, Consts >

SchItems = SchItems . HistItems (SchLevels)
SchLevels = SchLevels . UpdateLevel
UpdateLevel(L) = <L.LevelName, L.Is . HistItems ({L}), L.Const>

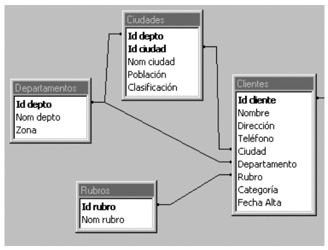
Esquema Intermedio

ObjectItems: (SchItems . SchLevels . SchDimensions . SchCubes)
→ SchItems

-Dado I . SchItems, ObjectItems (I)= {I}
-Dado L . SchLevels, ObjectItems (L)= L.Is
-Dado D . SchDimensions, ObjectItems (D)= D.Ls .Is
-Dado C . SchCubes, ObjectItems (C)= C.Ls .Is

Base Fuente

- **Se trabaja con una única fuente:**
  - Base relacional.
  - Integrada.
- **Interesan:**
  - Atributos.
  - Tablas.
  - Claves primarias.
  - Links.



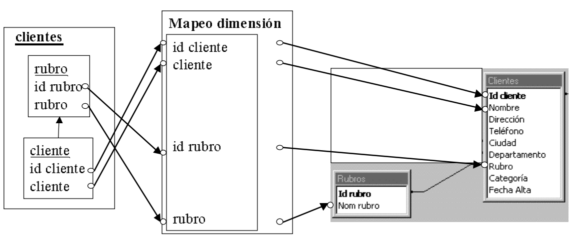
Base Fuente

SourceSchema . <SchAttributes, SchTables, SchLinks >

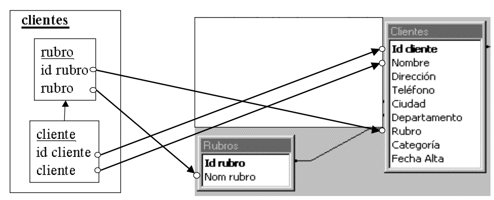
SchAttributes . { <AttName, Type > /
AttName . Strings .
Type . SIMPLETYPES }
SchTables . { <TabName, As, PK > /
TabName . Strings .
As . SchAttributes .
PK . As }
SchLinks . $\{T_1 / T_1$. SchTables} x $\{T_2 / T_2$. SchTables} →
Conditions (T_1.As . T_2.As) . .

Mapeos

- Correspondencias entre items del esquema intermedio y las fuentes.



Mapeos



Mapeos

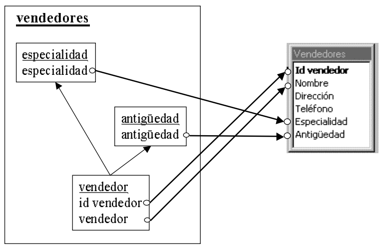
- **Son funciones entre los items del esquema intermedio y la fuente:**
  - A un item le asocia una expresión de mapeo.
- **Expresiones de mapeos:**
  - Puede ser:
    - » Un atributo de la fuente: DirectME.
    - » Un cálculo sobre algunos atributos: 1calcME.
    - » Una totalización sobre algunos atributos: NcalcME.
    - » Una constante: ConstantME.

Mapeos

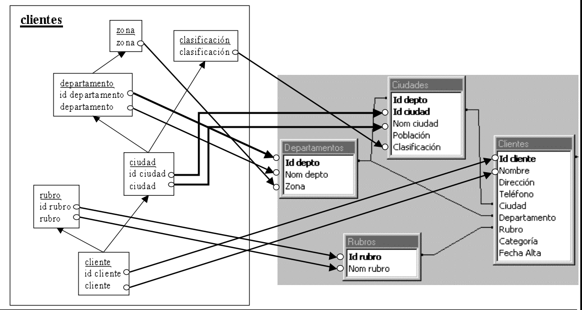
Mappings (Its) . { f / f . Its → MapExprs .
. I . Its. (I.Type = exprtype(f(I))) }

SchDimensionMappings . {D / D . SchDimensions} →
Mappings (D.Ls .Is)
SchCubeMappings . {C / C . SchCubes} →
(Mappings (C.Ls .Is) . SchCubes)

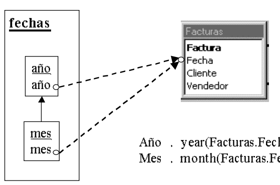
Mapeos Dimensiones



Mapeos Dimensiones

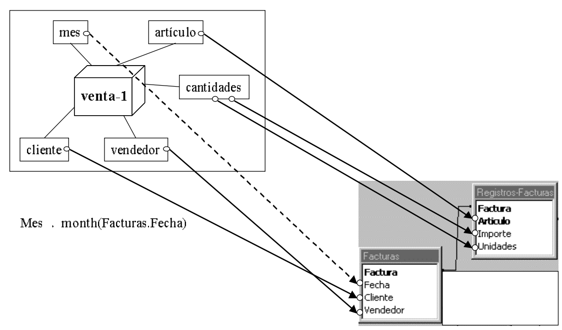


Mapeos Dimensiones



Año . year(Facturas.Fecha)
Mes . month(Facturas.Fecha)

Mapeos Cubos



Mes . month(Facturas.Fecha)

Reglas

Reglas

- **Son reglas de transformación.**
- **Se aplican a objetos del ECS.**
  - Que cumplen determinadas condiciones.
- **Estados:**
  - Tablas del esquema.
  - Funciones de mapeo.
- **Cada regla transformará el estado.**
  - Estado inicial. (input)
  - Estado final. (result)
  - Transformación.

Reglas

RULE *rulenumber* : *RULE NAME*

Description:
Describe el escenario y los objetivos para la aplicación de la regla.

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

Input:
- **Maps:** Funciones de mapeo definidas para los objetos.
- **Tables:** Tablas asociadas a la regla.

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

Conditions:
Condiciones que deben cumplir los objetos para que se aplique la regla.

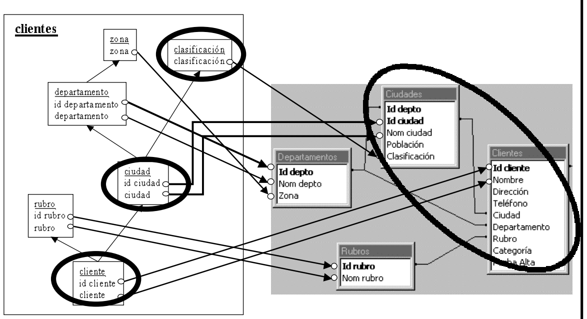
Result:
- **Maps:** Funciones de mapeo que cambiaron por aplicar la regla.
- **Tables:** Tablas que cambiaron por aplicar la regla.

Transformation:
Indica la primitiva a aplicar (en caso de cumplirse las condiciones), detallando los parámetr entrada.

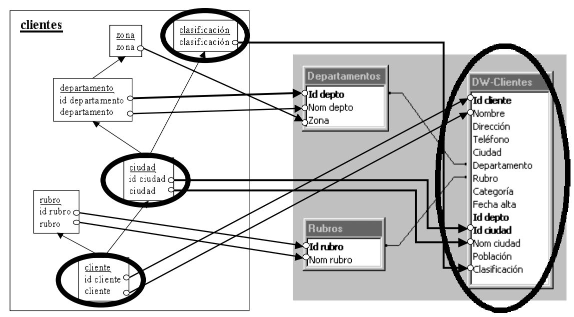
Rule 1: Join

- Motivación
  - Se definieron estrategias:
    - » Particiones de Dimensiones:
      - Indica que niveles se desean almacenar juntos.
    - » Cubos:
      - Indica que cruzamientos se desean almacenar juntos.
  - Se definieron mapeos.
    - » Pueden referenciar a varias tablas.
  - Se quiere construir una tabla:
    - » que contenga todos los items involucrados,
    - » exceptuando a los agregados.

Rule 1: Join



Rule 1: Join



Rule 1: Join

RULE 1 - JOIN

Description:
Dado un conjunto de items, una función de mapeo sobre ellos y dos tablas que joinean, y map dichos items, se genera una nueva tabla con los atributos de ambas, siguiendo el camino natural de j

Objects:
- OS $\in$ SchDimensionPartitions ♦ SchDimensions $\cup$ SchCubes /* un grupo de niveles o un cubo */

Input:
- **Maps:** F $\in$ Mappings(Its) / Its $\subseteq$ SchItems $\wedge$ OS ♦ ObjectItems $\subseteq$ Its /* una función de mapeo */
- **Tables:** T1, T2 $\in$ MapTables(F, OS ♦ ObjectItems), T1 $\neq$ T2, SchLinks(T1,T2) $\neq \perp$ /* table mapean a los items */

Result:
- **Tables:** T' /* con links a T1 y T2 por sus claves, y hereda los links de T1 y T2 *
- **Maps:** F' result of:
  - In F: ReplaceTable (T1,T')
  - In F: ReplaceTable (T2,T')

Rule 1: Join

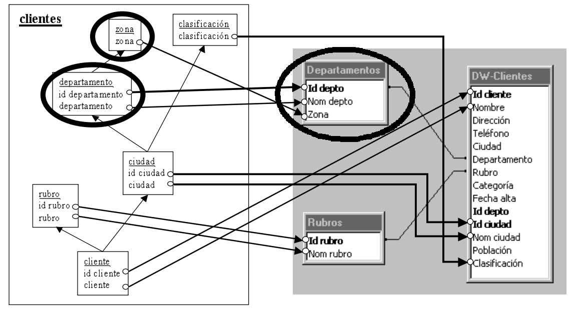
Transformation:
Apply: Primitive Q5 – Relation Merge

Parameters:
- {T1, T2}, /* esquema fuente */
- SchLinks(T1,T2), /* función de join */
- $\emptyset$, /* atributos de la 1er relación a eliminar */
- $\emptyset$, /* atributos de la 2da relación a eliminar */
- {T1.Instance, T2.Instance} /* instancias */

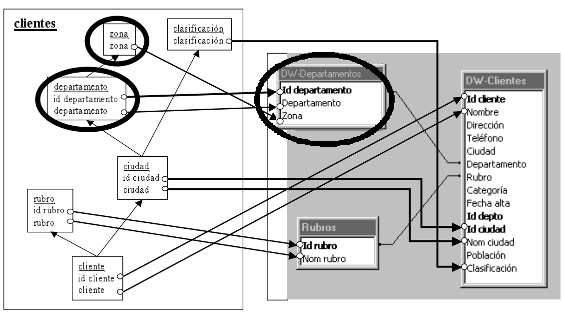
Rule 2: Rename

- **Motivación**
  - En las bases fuentes se puede haber utilizado cualquier nomenclatura para los atributos.
  - En el esquema conceptual se le asignaron nombres adecuados a los items.
  - Se quiere renombrar los atributos de las fuentes con los nombres de los items.
    - » Se utilizarán los mapeos directME.
- **Problema:**
  - Un atributo mapea con varios items.
  - Solución: Un mapeo directME, los otros 1calcME.

Rule 2: Rename



Rule 2: Rename



Rule 2: Rename

RULE 1 - RENAME

Description:
Dado un conjunto de items, una función de mapeo sobre ellos y una tabla a la que se mapea, se g una nueva tabla renombrando sus atributos según los mapeos.

Objects:
- OS ∈ SchDimensions ∪ SchCubes /* una dimensión o un cubo */
- Its ⊆ SchItems /* un conjunto de items */

Input:
- **Maps:** F ∈ Mappings(Its) / Its = ObjectItems(OS) /* una función de mapeo */
- **Tables:** T ∈ MapTables(F,Its) /* tabla que mapea a los items */

Let:
- List = {<I,A> / I ∈ Its ∧ MapType(F(I)) = directME ∧ A ∈ MapAttributes(F, {I})
  ∧ A.AttrName ≠ I.ItemName} /* los mapeos directos a un atributo de la tabla, que no se no igual que el item */

Conditions:
- List ≠ ∅

Rule 2: Rename

Result:
- **Tables:** T' /* con links a T por sus claves, y hereda los links de T */
- **Maps:** F' result of:
  - For each <I,A> ∈ List, In F(I): ReplaceAttribute (A, I)
  - In F: ReplaceTable (T, T')

Transformation:
Apply: Primitive Q2 – Attribute Renaming

Parameters:
- {T.TabName}, /* esquema fuente */
- {<A.AttrName, I.ItemName> / <I,A> ∈ List}, /* <atributos a renombrar, nuevos nombres> */
- {T.Instance} /* instancias */