UNIVERSIDAD DE LA EMPRESA

TEORÍA DEL DISEÑO PARA BASES DE DATOS RELACIONALES

BASES DE DATOS I

5.1 INTRODUCCIÓN

En esta parte se tratarán temas referentes al diseño de bases de datos relacionales.

En general, el objetivo del diseño de una base de datos relacional es generar un conjunto de esquemas de relaciones que permitan almacenar la información con un mínimo de redundancia, pero que a la vez faciliten la recuperación de la información. Una de las técnicas para lograrlo consiste en diseñar esquemas que tengan una *forma normal* adecuada.

Cuando se está diseñando una base de datos relacional, a menudo se elige una alternativa de varios conjuntos de esquemas. Algunas opciones son más convenientes que otras por varias razones. Se estudiarán algunas propiedades deseables de los esquemas de relaciones y se considerarán varios algoritmos para obtener un esquema de la base de datos, con buenas propiedades.

La idea central en el diseño de esquemas de bases de datos es el concepto de *dependencia de datos,* esto es, una restricción sobre las instancias de relaciones posibles. Por ejemplo, si un atributo determina en forma única a otro atributo (como aparentemente es el caso en la relación *HABITACIONES* de la base de datos del hotel, donde el atributo *NUM-HABITACION* determina el *PRECIO*), decimos que hay una *dependencia funcional.* En el caso de *HABITACIONES, PRECIO* depende funcionalmente de *NUM-HABITACION.* También se considerarán dependencias funcionales más complejas, llamadas *dependencias multivaluadas,* donde algunos atributos determinan un conjunto de valores correspondientes a otros atributos, en vez de determinar un único valor.

5.1.1 Problemas de un mal diseño

Antes de estudiar detalladamente cómo diseñar un buen esquema de base de datos, veamos un caso práctico de lo que es un mal diseño:

Problema Considérese el esquema de una base de datos, que contiene una única relación, cuyo esquema se ajusta a la siguiente tabla.

PROVEEDOR	DOMICILIO	ARTICULO	PRECIO
Acme Corp.	*Cuareim 1522*	*Monitor VGA*	*450*
Acme Corp.	*Cuareim 1522*	*Monitor Herc*	*120*
Computronix	*Yaguarón 9786*	*Monitor VGA*	*470*

Podemos observar varios problemas en este diseño:

1. Redundancia. El domicilio del proveedor se repite una vez por cada artículo provisto.

2. Inconsistencia potencial (anomalías de actualización). Como consecuencia de la redundancia, podría darse el caso de que actualicemos el domicilio de un proveedor en una tupla y olvidarnos de hacerlo en las demás tuplas de dicho proveedor. De esta forma no tendríamos un único domicilio para cada proveedor como intuitivamente creemos que debería ser.

3. Anomalía de inserción. No podemos registrar el domicilio de un proveedor si dicho proveedor no provee al menos un artículo. Deberemos poner valores nulos de artículo y precio en la tupla de dicho proveedor, pero entonces, cuando ingresemos un artículo de dicho proveedor, ¿nos acordaremos de borrar la tupla con valores nulos? Peor aún, artículo y proveedor son la clave para acceder al precio (ya que el precio queda determinado funcionalmente por el artículo y su proveedor) por lo que puede ser complicado o imposible buscar tuplas que tengan valores nulos en tales atributos.

4. Anomalías de borrado. Es el problema inverso al anterior: en el caso de que borremos todos los items de un mismo proveedor, perdemos la información del domicilio de dicho proveedor.

En este ejemplo, todos los problemas mencionados desaparecen si en el esquema de la base de datos tenemos dos esquemas de relaciones, que se ajusten a las siguientes tablas:

Tabla PROVEEDORES

PROVEEDOR	**DOMICILIO**
Acme Corp.	*Cuareim 1522*
Computronix	*Yaguarón 9786*

Tabla PRECIOS

PROVEEDOR	**ARTICULO**	**PRECIO**
Acme Corp.	*Monitor VGA*	*450*
Acme Corp.	*Monitor Herc*	*120*
Computronix	*Monitor VGA*	*470*

La tabla *PROVEEDORES* registra los domicilios de cada proveedor exactamente una vez, por lo que no tiene redundancia de información. Adicionalmente, podemos insertar datos de nuevos proveedores aún cuando momentáneamente dicho proveedor no esté proveyendo artículo alguno. La tabla *PRECIOS* registra a los proveedores, los artículos que proveen, y el precio que cada proveedor cobra por cada artículo que provee.

Es importante destacar que aún permanecen ciertas dudas. Por ejemplo, la descomposición mostrada tiene una desventaja: para listar la tabla de precios junto con los domicilios de cada proveedor, debemos primero hacer alguna operación que junte la información de ambas tablas.

5.1.2 Dependencias y redundancia

La relación entre las dependencias y la redundancia es bastante estrecha. En general, una *dependencia* es simplemente una *restricción de integridad* que establece que solamente un cierto subconjunto de todas las instancias posibles, son "legales", es decir, solamente ciertas relaciones reflejan un estado posible del mundo real. La forma de la redundancia es obvia cuando las dependencias son funcionales. Cuando vemos la primera tupla de la tabla original, sabemos que "Cuareim 1522" es el domicilio de "Acme Corp.". Si sabemos que el proveedor determina funcionalmente al domicilio en dicha relación, y vemos que en la segunda tupla el proveedor es otra vez "Acme Corp.", no tendríamos necesidad de observar el domicilio en la segunda tupla, puesto que sabemos que deberá ser "Cuareim 1522". Por lo tanto es información redundante.

En el caso de tipos de dependencias más generales, la forma que adopta la redundancia puede no ser tan clara. Sin embargo, en todos los casos las dependencias funcionales causan la redundancia y a la vez sirven para descomponer de forma tal que no la haya.

5.2 DEPENDENCIAS FUNCIONALES

Definición: [Dependencia Funcional]

Sea RS = (ATR,DMN,dom,M,SC)

Sean X , Y conjuntos de atributos tomados de ATR.
Una dependencia **X → Y** sobre PRS es una restricción que es satisfecha por una posible instancia de relación prs si y solo si para todas tuplas t1 y t2 de prs tales que t1[X] = t2[X] implica que t1[Y] = t2[Y]

X → Y se lee como X determina a Y ó Y es determinado por X.

5.2.1 Implicación lógica de dependencias

Definición: [Implicación de Dependencias]

Sea un esquema de relación RS=(ATR,DMN,dom,M,SC) donde:

SC contiene dependencias funcionales y sea
X → Y una dependencia funcional.

Decimos que SC |= {**X → Y**}

SC implica lógicamente a **X → Y** si cada instancia ***r*** de RS que satisface las dependencias funcionales de SC también satisface **X → Y**.

Definición: [Implicación lógica de restricciones]

Sea PRS=(ATR,DMN,dom) un esquema de relación primitivo y sea SC(PRS) el conjunto de todas las restricciones (o dependencias) que puedan definirse sobre PRS.

Sean SC1 y SC2 subconjuntos de SC(PRS)

- SC1 |= SC2 si cada instancia posible de PRS que satisface a SC1 también satisface a SC2.
- SC1 es equivalente a SC2, denotado SC1 <=> SC2 , si SC1 |= SC2 y SC2 |= SC1

Definición: [Conjunto clausura de restricciones]

Sea SC* el conjunto de todas las dependencias funcionales implicadas por SC, es decir:

SC* = { X → Y | SC |= X → Y }

Denominamos SC* como el **conjunto clausura** de SC.

Teorema Sea *RS = (ATR, DMN, dom, M, SC)* un esquema de relación y sea *X --> Y* una fd en *SC.* Sea *r* una instancia de la relación *RS.* Entonces $r = \pi(r,XY) * \pi(r,X(ATR - Y))$.
■

Observaciones: Informalmente el teorema anterior dice que siempre que una dependencia funcional se cumpla en una relación, esta relación puede ser descompuesta de tal forma que la relación original puede siempre ser reconstruida realizando un join.
■

Teorema Sea *RS = (ATR, DMN, dom, M, SC)* un esquema de relación en el cual *SC* es un conjunto de fds. Sean *X,Y* conjuntos de atributos de *ATR* y supongamos que para

cualquier instancia *r* de *RS* tenemos que $r = \pi(r, XY) * \pi(r, X(ATR - Y))$.
Entonces $X \rightarrow Y \in SC^*$ o $X \rightarrow (ATR - Y) \in SC^*$.

En general, para implicar lógicamente dependencias:

- Se utilizan los **Axiomas de Armstrong** para implicar lógicamente dependencias funcionales.

- Con las tres primeras reglas se pueden hacer todas las inferencias válidas.

- Las demás reglas se pueden deducir de las tres primeras, por lo que se consideran reglas adicionales.

Dependencias triviales:

- $X \rightarrow Y$ siempre que $Y \subset X$
- $X \rightarrow \varnothing$

5.2.2 Axiomas de Armstrong

- *Regla 1* - **REFLEXIVA**

  $\{\} \models X \rightarrow Y$, si $Y \subset X$

- *Regla 2* - **AUMENTACIÓN**

  $\{ X \rightarrow Y \} \models XZ \rightarrow YZ$

- *Regla 3* – **TRANSITIVA**

  $\{ X \rightarrow Y , Y \rightarrow Z \} \models X \rightarrow Z$

- *Regla 4* – **UNION**

  $\{ X \rightarrow Y , X \rightarrow Z \} \models X \rightarrow YZ$

- *Regla 5* - **INTERSECCION**

  $\{ X \rightarrow Y , X \rightarrow Z \} \models X \rightarrow Y \cap Z$

- *Regla 6* - **REDUCCION**

  $\{ X \rightarrow Y \} \models X \rightarrow Y - X$

- *Regla 7* - **AUMENTACIÓN GENERALIZADA**

  $\{ X \rightarrow Y \} \models W \rightarrow V$, si $X \subset W$ y $V \subset XY$

- *Regla 8* - **FRAGMENTACIÓN**

  $\{ X \rightarrow Y \} \models X \rightarrow A$, si $A \subset Y$

- *Regla 9* - **TRANSITIVA GENERALIZADA**

  $\{ X \rightarrow Y, U \rightarrow V \} \models W \rightarrow Z$, si $U \subset XY$, $X \subset W$ y $Z \subset VW$

5.2.3 Determinantes (claves)

Definición: [Determinante]

Sea RS un esquema de relación con: ATR = {A1,A2, ... ,A3 } y SC el conjunto de restricciones que contiene dependencias funcionales.

Decimos que el conjunto de atributos X de ATR es determinante o clave si:

1) $X \rightarrow$ ATR está en SC*
2) Para ningún $Y \subseteq X$ ocurre que $Y \rightarrow$ ATR está en SC*
   esta es la *condición de minimalidad*

Un mismo esquema de relación puede tener varias claves. Se utiliza el término ***clave candidata*** para denominar un conjunto mínimo de atributos que determinan funcionalmente a todos los atributos del esquema.

Frecuentemente se designa una de las claves candidatas como ***clave primaria***, que podrá utilizarse como clave de archivo cuando la relación sea implementada en un DBMS.

El término ***superclave*** se utiliza para denominar a cualquier superconjunto propio de una clave. Observar que toda clave es una superclave.

Para determinar claves se necesita calcular el conjunto clausura de dependencias funcionales SC* o al menos, dado el conjunto SC necesitamos testear si $X \rightarrow Y$ pertenece a SC*.

Lema:
Si se deduce $X \rightarrow Y$ a partir de SC usando los axiomas de Armstrong, entonces $X \rightarrow Y$ se cumple para cualquier instancia ***r*** posible del RS, en la cual las dependencias de SC son verdaderas.

5.2.4 Clausura de atributos

Definición: [Clausura de Atributos X+]

Sea SC un conjunto de dependencias funcionales sobre el conjunto de atributos ATR y sea X un conjunto de atributos de ATR.
Se define conjunto clausura de X con respecto a SC , denotado X+, como al conjunto de atributos A tales que
SC |= $X \rightarrow A$, utilizando las reglas de inferencia.

Teniendo en cuenta lo anterior podemos saber si una dependencia $X \rightarrow Y$ pertenece a SC* si y solo si $Y \subseteq X+$

Algoritmo de Cálculo de X+:

ENTRADA : Conjunto de atributos X y conjunto de dependencias SC.
SALIDA : X+ (Clausura de X respecto a SC)

```
Resultado := X
Mientras (Resultado cambió)
        Para cada dependencia Y → Z en SC
                Si Y ⊂ Resultado
                        Resultado := Resultado ∪ Z
                FinSi
        FinPara
FinMientras
```

Cómo determinar si un conjunto de atributos es clave?

Un conjunto de atributos X es superclave si su clausura X+ contiene a todos los atributos del esquema.

Cuando se está buscando una clave, se toma como posible clave al conjunto de atributos tal que, sus miembros no aparecen en la parte derecha de las dependencias funcionales. Es decir los atributos que no son determinados por otros.

Este conjunto de atributos pertenece a la clave.

5.2.5 Cubrimientos

Definición: [Equivalencia de conjuntos de fds] Sean F y G conjuntos de fds. Decimos que F es equivalente a G si F* = G*. Se suele decir que F cubre a G (y G cubre a F).
■

Algoritmo
ENTRADA: F y G (conjuntos de dfs)
SALIDA : VERDADERO si cada fd de F está en G*
PARA CADA Y-->Z en F
  Calcular Y+ según G
  SI Z no es subconjunto de Y+
    RETORNAR (FALSO)
  FINSI
FINPARA
RETORNAR (VERDADERO)
■

Observaciones: Para probar que *F** = *G** hay que aplicar 2 veces el algoritmo anterior: una vez para probar que cada fd de *F* está en *G** y otra vez para probar que cada fd de *G* está en *F*.*

Una cuestión interesante es poder hallar un conjunto cubrimiento, con la menor cantidad de fds posible, y tales que las mismas sean lo más sencillas posibles.
■

Lema Cada conjunto *F* de fds es cubierto por un conjunto *G* en el cual todos los conjuntos de atributos a la derecha de cada fd contienen un único atributo.
■

Definición: [Cubrimiento minimal] Un conjunto *F* de fds es *minimal* si:

1. El lado derecho de cada fd es un único atributo,

2. para ninguna *X-->Y* en *F* se cumple que *F - {X-->Y}* es equivalente a *F,*

3. para ninguna *X-->Y* en *F* y siendo *Z* < *X* se cumple que *F - {X-->Y} U {Z-->Y}* es equivalente a *F.*

■

Observaciones: Una forma práctica de conseguir un conjunto minimal consiste en convertir todas las fds a fds con un único atributo en la parte derecha. Luego, para cada *X --> Y* ver si *F - {X --> Y}* equivale a *F;* si esto es cierto, eliminar *X --> Y* de *F.* Ahora para cada fd que quedó, tratar de eliminar atributos redundantes en la parte izquierda.

■

Teorema Todo conjunto *SC* de fds es equivalente a un conjunto *SC'* minimal.

■

5.2.6 Descomposición de esquemas

Definición: [Descomposición de un esquema] Sea un esquema de relación *RS = (ATR, DMN, dom, M, SC).* Un esquema de base de datos *DS = (PDS, DM, SDC)* con *PDS = { RSi = (ATRi, DMNi, domi, Mi, SCi) | i=1..k }* es llamado *descomposición (de join sin pérdida)* si:

1. ATRi $\subseteq$ *ATR, i=1..k.*

2. ATR1 U ATR2 U ... U ATRk = ATR.

3. DM expresa que *DS* es una representación de la relación *RS* con el significado *M.*

4. SDC = Ø.

5. Para cada instancia *r* de *RS,* $\pi(r,ATR1)^* \pi(r,ATR2)^*...^* \pi(r,ATRk) = r$

Problema Sea un esquema *RS = (ATR, DMN, dom, M, SC)* donde *ATR = {PROVEEDOR, DOMICILIO, ARTICULO, PRECIO}* y *SC = {PROVEEDOR --> DOMICILIO, NOMBRE ARTICULO --> PRECIO}.* Supongamos la siguiente instancia:

PROVEEDOR	DOMICILIO	ARTICULO	PRECIO
Acme Corp.	*Cuareim 1522*	*Monitor VGA*	*450*
Acme Corp.	*Cuareim 1522*	*Monitor Herc*	*120*
Computronix	*Yaguarón 9786*	*Monitor VGA*	*470*
Computronix	*Yaguarón 9786*	*Monitor Color*	*520*

Supóngase una descomposición de *RS* en los dos subesquemas *RS1* y *RS2*, que tienen respectivamente a los conjuntos de atributos *{PROVEEDOR, DOMICILIO, ARTICULO}* y *{ARTICULO, PRECIO}:*

PROVEEDOR	DOMICILIO	ARTICULO
Acme Corp.	*Cuareim 1522*	*Monitor VGA*
Acme Corp.	*Cuareim 1522*	*Monitor Herc*
Computronix	*Yaguarón 9786*	*Monitor VGA*
Computronix	*Yaguarón 9786*	*Monitor Color*

ARTICULO	PRECIO
Monitor VGA	*450*
Monitor Herc	*120*
Monitor VGA	*470*
Monitor Color	*520*

Veamos ahora qué es lo que ocurre si hacemos el join de ambas tablas: *R1*R2.* El resultado es el siguiente:

PROVEEDOR	**DOMICILIO**	**ARTICULO**	**PRECIO**
Acme Corp.	*Cuareim 1522*	*Monitor VGA*	*450*
Acme Corp.	*Cuareim 1522*	*Monitor VGA*	*470*
Acme Corp.	*Cuareim 1522*	*Monitor Herc*	*120*
Computronix	*Yaguarón 9786*	*Monitor VGA*	*450*
Computronix	*Yaguarón 9786*	*Monitor VGA*	*470*
Computronix	*Yaguarón 9786*	*Monitor Color*	*520*

Observemos que *no* hemos reconstruido la tabla original. Han aparecido dos tuplas erróneas (segunda y cuarta). Es decir, la descomposición del esquema original en dos subesquemas es una descomposición de join con pérdida.
■

El siguiente método práctico puede aplicarse para testear si una descomposición *d = {RS1,...,RSk}* de un esquema *RS = ({A1,A2,...,An}, DMN, dom, M, SC)* es de join sin pérdida: primero se construye una tabla con *n* columnas y *k* renglones. La columna *j* corresponde al atributo *Aj* y el renglón *i* corresponde al esquema *RSi.* En la posición *(i,j)* colocamos el simbolo "a_i" si *Aj* está en *ATRi.* En caso contrario colocamos el simbolo "b_ij". Ahora consideramos repetidamente cada una de las fds *X-->Y* en *SC* hasta que no se puedan hacer más cambios a la tabla. Cada vez que consideramos una fd *X-->Y* buscamos los renglones que coinciden en todas las columnas de los atributos *X.* Si encontramos dos renglones con tales características, igualamos en ellos a los atributos correspondientes a los atributos *Y.* Al igualar dos símbolos, si uno es un "a_j", hacemos que el otro sea "a_j". Si son simbolos "b" entonces igualamos arbitrariamente por alguno de ellos. Si en algún momento descubrimos que algún renglón tiene la forma "a_1 a_2 ... a_k" entonces la descomposición es de join sin pérdida. Si en un momento dado la tabla permanece incambiada sin que haya algún renglón lleno de "a", la descomposición es con pérdida.

Teorema Sea *{RS1,RS2}* una descomposición del esquema *RS = (ATR, DMN, dom, M, SC).* La descomposición es de join sin pérdida con respecto a *SC* si y sólo si, o bien se cumple *R1 ∩ R2 --> R1 - R2,* o bien se cumple *R1 ∩ R2 --> R2 - R1.* Observar que al decir "se cumple" queremos decir "están en SC*."

5.2.7 Preservación de dependencias

Hemos visto que una cualidad deseable en una descomposición es que sea de join sin pérdida. Ahora veremos que también es deseable que una descomposición preserve todas las dependencias funcionales que se cumplen.

Definición: [Proyección de fds] Sea *RS = (ATR, DMN, dom, M, SC)* un esquema de relación donde *SC* es un conjunto de fds. Sea *ATRi ⊆ ATR.* Una *proyección del conjunto de fds SC sobre ATRi* es un conjunto de fds *SCi* que es lógicamente equivalente a *{ X-->Y | X-->Y ∈ SC* & XY ⊆ ATRi }. La notación es πfd(SC,ATRi).*
■

Ejemplo: Sea el esquema de relación *RS = ({A,B,C,D}, DMN, dom, M, {A-->B, B-->C, AC-->D}.* Fácilmente se puede verificar que una proyección de *SC* sobre *ABD* es *{A-->B, A-->D}.*
■

Definición: [Preservación de dependencias] Sea un esquema RS = (ATR, DMN, dom, M, SC) y una descomposición d = {RS1,...,RSk}. Decimos que la descomposición preserva las dependencias de SC si la unión de todas las
dependencias en πfd(SC,Ri) donde i⊆k, implica lógicamente a todas las fds de SC.
■

Algoritmo [Test de preservación]

```
ENTRADA:  RS = (ATR,DMN,dom,M,SC) y {RS1,...,RSk} (descomposición de RS)
SALIDA :  VERDADERO si la descomposición preserva fds
PARA CADA  X-->Y en SC
  Z := X
  MIENTRAS  Z cambió
  /* Agregar a Z los atributos comunes de RSi con la clausura de los atributos
        comunes de Z y RSi (i=1..k) */
    Z := Z U ( (Z ∩ RSi)+  ∩ RSi )  ,i=1..k
  FINMIENTRAS
  SI Y >= Z  /* ¿Y no es subconjunto propio de Z? */
```


```
    RETORNAR (FALSO)
  FINSI
FINPARA
RETORNAR (VERDADERO)
■
```

5.3 DEPENDENCIAS MULTIVALUADAS

Sea una instancia *r* de un esquema de relación *RS = (ATR, DMN, dom, M, SC)* . Se dice que *X→→Y* (y se lee *"X multidetermina a Y"*) cuando el hecho de que existan en *r* dos tuplas *t1* y *t2* tales que *t1(X)=t2(X)* implica que existen en *r* otras dos tuplas *t3* y *t4* tales que:

1) *t1(X) = t2(X) = t3(X) = t4(X)*
2) *t3(Y) = t1(Y)* $\wedge$ *t3(ATR-X-Y) = t2(ATR-X-Y)*
3) *t4(Y) = t2(Y)* $\wedge$ *t4(ATR-X-Y) = t1(ATR-X-Y)*

Gráficamente:

X	**Y**	**ATR-X-Y**	
x1	y1	z1	t1
x1	y2	z2	t2
...	...	...	
x1	y2	z1	t4
x1	y1	z2	t3

Intuición primera: se pueden intercambiar los valores de Y para t1 y t2 y armar dos nuevas tuplas que también deben estar en r.

Intuición segunda: dado un valor para X hay un *conjunto* de valores de Y asociado a X y no asociado a ATR-X-Y.

Ejemplo:

CURSO	**PROFESOR**	**HORARIO**	**AULA**	**ESTUDIANTE**	**NOTA**
C1	Milani	Lunes 9	A	Juan	8
C1	Milani	Miércoles 9	B	Juan	8
C1	Milani	Viernes 9	C	Juan	8
C1	Milani	Lunes 9	A	María	9
C1	Milani	Miércoles 9	B	María	9
C1	Milani	Viernes 9	C	María	9
C1	Milani	Lunes 9	A	Pedro	10

C1	Milani	Miércoles 9	B	Pedro	10
C1	Milani	Viernes 9	C	Pedro	10

Un curso se dicta en algunos horarios y salones fijos. Cada estudiante tiene una tupla por cada dictado del curso (cada par horario-aula) con la nota repetida en cada aparición.

Es lógico pensar que Curso →→ Horario Aula porque seguramente hay un conjunto de pares horario-aula asociados con cada curso y disociado del resto de los atributos.

Por el contrario, no es lógico pensar que Curso →→ Horario. Si eso fuera cierto, deberíamos encontrar al menos la tupla siguiente en r (y en realidad no está):

C1	Milani	Lunes 9	B	Juan	8

Lema Si una instancia *r* de un esquema de relación *RS = (ATR, DMN, dom, M, SC)* satisface *X->->Y*, entonces *r* satisface la mvd *X ->-> ATR - XY.*
■

La definición de mvd exige que *X* y *Y* sean disjuntos en *X ->-> Y.* Supongamos que eliminamos esta restricción de la definición. Sea *r* una instancia de *RS* que satisface *X ->-> Y* (bajo la definición modificada) y sea *Y' = Y - X.* Bajo cualquiera de las definiciones *r* satisface *X ->-> Y'.*

Ejemplo: Sea un esquema *RS = ({A,B,C,D}, DMN, dom, M, SC)* y sea *r* una instancia de *RS:*

A	B	C	D
a	*b*	*c*	*d*
a	*b*	*c'*	*d'*
a	*b*	*c*	*d'*
a	*b*	*c'*	*d*
a	*b'*	*c'*	*d*
a'	*b*	*c*	*d'*

Esta instancia satisface *AB->->BC,* entonces también satisface *AB->->C.*
■

5.3.1 Casos especiales

Veremos cuál es el significado de los casos especiales *Ø ->-> Y* y *X ->-> Ø* para una instancia *r* de un esquema *RS.* Consideremos *Ø ->-> Y* y sea *Z = ATR - Y.* Para cualquier par de tuplas *t1* y *t2* en *r* se cumple que *t1[Ø] = t2[Ø].* Si *r* satisface *Ø ->-> Y,* entonces debe existir una tupla *t3* perteneciente a *r,* tal que: *t3[Y] = t1[Y]* y *t3[Z] = t2[Z].* O sea, *r* debe ser el producto cartesiano de las proyecciones $\pi(r,Y)$ y $\pi(r,Z)$.

La mvd *X ->-> Ø* se satisface trivialmente en cualquier instancia de un esquema que contenga a *X* entre sus atributos.

5.3.2 Propiedades de las mvds

Una vez dada la definición de mvd, veremos cómo se relaciona con la propiedad de descomposición con join sin pérdida.

Teorema Sea *r* una instancia del esquema *RS,* con *X, Y, Z* subconjuntos de *ATR* tales que *Z = ATR - XY.* La instancia *r* satisface la mvd *X ->-> Y* si se descompone con join sin pérdida en los esquemas de relación *RS1 = ({X,Y}, DMN1, dom1, M1, SC1)* y *RS2 = ({X,Z}, DMN2, dom2, M2, SC2).*
■

Corolario Sea *r* una instancia de un esquema *RS* y sean $X \in Y$ subconjuntos de *ATR.* Si *r* satisface *X --> Y,* entonces satisface la mvd *X ->-> Y.*
■

Supongamos que tenemos un esquema de relacion *RS* y un conjunto *SC* de dependencias funcionales sobre *ATR.* Queremos conocer el conjunto de mvds que se cumplen en una instancia *r* de *RS* que satisface *SC.* Por el último corolario sabemos que si *X --> Y* pertenece a *SC*,* entonces *r* satisface *X ->-> Y,* y por un lema anterior sabemos que *r*

satisface *X ->-> ATR - XY.* La pregunta que se nos plantea ahora es si existen otras mvds que se satisfagan siempre en *r* y que no correspondan directamente a fds?. La respuesta es "no".

Teorema Sea *SC* un conjunto de fds definidas sobre *ATR,* el conjunto de atributos de un esquema *RS.* Sean *X,Y,Z* subconjuntos de *ATR,* con *Z = ATR - XY.* Sea *X+* la clausura de *X* bajo *SC.* Si *Y* no está contenida en *X+* y *Z* no está contenida en *X+,* entonces existe una instancia *r* del esquema *RS* que satisface *SC* y que no satisface la mvd *X ->-> Y.*
■

Por el teorema podemos ver que las unicas mvds implicadas por un conjunto de fds son aquellas de la forma *X ->-> Y,* donde *Y* está contenido en *X+* o *ATR - XY* está contenido en *X+.*

Ejemplo: Sea *RS = ({A,B,C,D,E,I}, DMN, dom, M, SC)* y *SC = { A --> BC, C --> D}.* Entonces *SC* implica *A ->-> BCD* y *A ->-> C* pero *SC* no implica *A ->-> DE.*
■

5.3.3 Reglas de inferencia para mvds

Como fue el caso para las fds podemos calcular SC* desde SC usando las definiciones formales de dependencias. Sin embargo es más fácil trabajar sobre un conjunto de dependencias usando un sistema de reglas de inferencia.

La siguiente lista de reglas de inferencia para mvds es válida y completa. Al decir "válida" queremos decir que las reglas no generan dependencias que no son lógicamente implicadas por SC. Al decir "completa" queremos decir que las reglas nos permiten generar todas las dependencias en SC*.

■M1. AUMENTACION MULTIVALUADA:
{X ->-> Y} |= WX ->-> VY, si V $\subseteq$ W

■M2. TRANSITIVA MULTIVALUADA:
{X ->-> Y, Y ->-> Z} |= X ->-> Z--Y

■M3. UNION MULTIVALUADA:

{X ->-> Y, X ->-> Z} |= X ->-> YZ

■M4. INTERSECCION MULTIVALUADA:
{X ->-> Y, X ->-> Z} |= X ->-> Y ∩ Z

■M5. DIFERENCIA MULTIVALUADA:
{X ->-> Y, X ->-> Z} |= X ->-> Y - Z, X ->-> Z - Y

■M6. COMPLEMENTACION:
{X ->-> Y} |= X ->-> ATR - Y

■FM1. REPLICACION:
{X --> Y} |= X ->-> Y

■FM2. COALESCENCIA:
{X ->-> Y, W --> Z} |= X --> Z
si Z ⊆ Y, y existe W tal que W ∩ Y = Ø

■FM3. CUASITRANSITIVA:
{X ->-> Y, Y --> Z} |= X --> Z - Y

Ejemplo: Sea RS = ({A,B,C,G,H,I}, DMN, dom, M, SC) donde SC = {A ->-> B, B ->-> HI, CG ->-> H }. Se pueden deducir:

- ■A ->-> CGHI (aplicando M6 a A ->-> B)
- ■A ->-> HI (M2 a A ->-> B y B ->-> HI)
- ■B-->H (FM2 a B ->-> HI y CG-->H y CG ∩ HI = Ø)
- ■A ->-> CG (M5 a A ->-> (CGHI - HI)).

■

Ejemplo: Considerar un esquema de relación RS = (ATR, DMN, dom, M, SC) donde ATR = {A, B, C, D, E, F} y SC = {A ->-> BC, A ->-> CD, BC ->-> CE, B-->C, CD-->DF}. Entre otras, las siguientes restricciones son también satisfechas para las instancias de la relación RS:

A ->-> DEF	A ->-> ADEF
A ->-> C	A ->-> B
A ->-> D	A ->-> BCD
A ->-> E	AEF ->-> BCE
B ->-> C	A-->F

■

5.3.4 Cálculo de la clausura de un conjunto de mvds

El cálculo de *SC*,* cuando contiene fds y mvds, presenta las mismas dificultades (o peores) que habíamos comentado cuando el cálculo de la clausura de un conjunto de fds. En la práctica nos vamos a limitar a testear si una mvd está en el conjunto clausura, sin calcularlo. La forma práctica para ver si X ->-> Y $\in$ SC* es:

1. A partir de SC, construir un conjunto M que contiene a todas las mvds de SC, y por cada fd *Z-->(A1 A2 ... Ak)* $\in$ *SC,* sustituirla por Z->->Ai, i=1..k.

2. Calcular la *base de dependencias de X respecto al conjunto M, B(M,X).* Este cálculo se realiza con un algoritmo que se verá en breve.

3. Ver si *Y - X = C1 U ... U Cn, Ci* $\in$ *B(M,X).*

Algoritmo [Cálculo de B(M,X), base de dependencias de X respecto a M]
ENTRADA: un esquema *RS, M* conjunto de mvds obtenido de *SC, X*$\subseteq$*ATR.*
SALIDA : *B(M,X)*

```
BMX := ATR - X

MIENTRAS (BMX cambió) HACER
  PARA CADA (V ->->W ∈ M) y (Y∈BMX) y (Y∩W != Ø) y (Y∩V = Ø) HACER
    BMX := (BMX - Y) U (Y ∩ W) U (Y - W)
  FINPARA
FINMIENTRAS
```

RETURN (BMX)
■

Ejemplo: Se calculará B(M,{A B}) para un esquema primitivo de relacion PRS con:

- ATR = {A B C D E F G}
- SC = {AB ->-> CD, C ->-> F, C-->E}

El algoritmo arranca con M = {AB ->-> CD, C ->-> F, C ->-> E}. El conjunto BMX de conjuntos disjuntos de atributos es: BMX = {C D E F G}. En pasos sucesivos, este conjunto BMX se transforma en: {CD EFG}, luego en {CD F EG} y finalmente en {CD F E G}.

5.3.5 Joins sin pérdida y preservación de mvds

Teorema Sea *RS = (ATR, DMN, dom, M, SC)* un esquema de relación, donde *SC* es un conjunto de fds y mvds. Sea una descomposición de *RS* en dos subesquemas *RS1 = (ATR1, DMN1, dom1, M1, SC1)* y *RS2 = (ATR2, DMN2, dom2, M2, SC2).* Entonces está descomposición es de join sin pérdida si y sólo se cumple *al menos una* de las siguientes implicaciones: *1. ATR1 ∩ ATR2 ->-> ATR1 2. ATR1 ∩ ATR2 ->-> ATR2*

■

El problema de la conservación de dependencias cuando se tienen mvds no es tan sencillo como en el caso de las fds. Sea *RS = (ATR, DMN, dom, M, SC)* un esquema de relación y una descomposición *D = { RSi = (ATRi, DMNi, domi, Mi, SCi)* | *i=1..n }* de *RS. SC* es un conjunto de fds y mvds. Sea un conjunto *SC'i* que consiste de:

- Las fds en *SC** que incluyan únicamente atributos de *ATRi.*
- Las mvds *X->->(Y ∩ ATRi)* donde *X⊆ATRi* y *X->->Y* está en *SC*.*

Definición: [Descomposición preservadora de mvds] La descomposición *{RS1, RS2, ..., RSn}* es una *descomposición preservadora de dependencias* si para cada conjunto de instancias *r1, r2, ..., rn* respectivamente de los esquemas *RS1, RS2, ..., RSn,* tales que cada *ri* satisface a *SC'i,* existe una instancia *r* de *RS* que satisface *SC* y para la cual *ri = π(r, ATRi),* para *i=1..n.*
■

5.3.6 Dependencias join

Dada la importancia del concepto del join sin pérdida, es útil poder limitar el conjunto de instancias legales de un esquema *RS* a aquellas para las cuales una descomposición dada sea de join sin pérdida. En esta sección vamos a definir esa restricción, llamada *dependencia join.*

Problema

Existen situaciones en las que un esquema puede ser descompuesto en tres subesquemas pero no en dos.

Ejemplo: Usaremos el clásico ejemplo de los bebedores, cervezas y bares. El esquema tiene una restricción que puede ser representada por la mvd *CERVEZA ->-> BEBEDOR* (dada una cerveza se determina al conjunto de bebedores de la misma) o por la mvd *CERVEZA ->-> BAR* (dada una cerveza se determina al conjunto de bares que la sirven).

Consideremos ahora un nuevo esquema *RS'* modificando levemente la restricción de integridad: Si un bebedor que beba una cerveza es tal que ese bebedor frecuenta un bar en el cual esa cerveza es servida, entonces ese bebedor bebe esa cerveza en ese bar. Veamos una instancia de *RS':*

BEBEDOR	CERVEZA	BAR
Juan	*Budweisser*	*Loro Azul*
Juan	*Budweisser*	*Heaven*
Juan	*Heineken*	*Loro azul*
Luis	*Budweisser*	*Loro Azul*

Se verifica que *no se cumplen las mvds CERVEZA ->-> BEBEDOR, BEBEDOR ->-> BAR* ni *BAR ->-> CERVEZA.* No se puede entonces descomponer en dos esquemas sin perder información, pero sí se puede en tres subesquemas:

R1 = ({BEBEDOR, CERVEZA}, DMN1, dom1, M1, SC1)
R2 = ({CERVEZA, BAR}, DMN2, dom2, M2, SC2)
R3 = ({BEBEDOR, BAR}, DMN3, dom3, M3, SC3)

Definición: [Dependencia join, join dependence, jd] Sea *ATR = {A1, ..., An}* y *X1,...,Xm* subconjuntos de *{A1,...,An}.* Se dice que existe una *dependencia join (join dependence, jd) *(X1,...,Xm)* si y sólo si la descomposición del esquema en *m* subesquemas con conjuntos de atributos *X1,...,Xm* respectivamente es de join sin pérdida. O sea que para toda instancia *r* del esquema se cumple que $r = \pi(r,X1) * ... \pi(r,Xm)$.
■

Observaciones: Las mvds son casos particulares de las jds, ya que toda instancia *r* de *RS = ({X,Y,Z}, DMN, dom, M, SC)* que satisfaga la *X ->-> Y* satisface la jd *(XY,XZ).* La forma práctica de deducir una jd a partir de un conjunto SC se hace con el mismo método utilizado para testear join sin pérdida.
■

Observaciones: En el caso de las fds y de las mvds, fue posible dar un sistema de reglas de inferencia. Desafortunadamente no se conoce un conjunto de reglas equivalente para las dependencias join. Al parecer es necesario considerar clases de dependencias más generales que las jds para construir un conjunto de reglas de inferencia válido y completo.
■

Sea *RS = (ATR, DMN, dom, M, SC)* un esquema de relación y sea una descomposición *DRS = { RSi = (ATRi, DMNi, domi, Mi, SCi) | i=1..n }.* Se utiliza la jd *(ATR1, ATR2, ..., ATRn)* para restringir el conjunto instancias válidas, a aquellas para las cuales *DRS* sea una descomposición de join sin pérdida. La instancia *r* de *RS* satisface la jd *(ATR1, ATR2, ..., ATRn)* si ocurre que $r = \pi(r, ATR1) * \pi(r, ATR2) * ... * \pi(r, ATRn)$. La jd es trivial si alguno de los *ATRi* es *ATR.*

Consideremos la jd *(ATR1, ATR2)* en el esquema *RS.* Esta dependencia requiere que $r = \pi(r,ATR1) * \pi(r,ATR2)$. Sea *r* una instancia que contiene las dos tuplas *t1* y *t2,* cuya definición es:

$t1[ATR1 - ATR2] = \langle a1,a2,...,ai \rangle$
$t1[ATR1 \cap ATR2] = \langle ai+1,...,aj \rangle$
$t1[ATR2 - ATR1] = \langle aj+1,...,an \rangle$

$t2[ATR1 - ATR2] = \langle b1,b2,...,bi \rangle$
$t2[ATR1 \cap ATR2] = \langle ai+1,...,aj \rangle$
$t2[ATR2 - ATR1] = \langle bj+1,...,bn \rangle$

Así, $t1[ATR1 \cap ATR2] = t2[ATR1 \cap ATR2]$, pero *t1* y *t2* tienen diferentes valores en todos

los demás atributos. A continuación se calculará *π(r,ATR1)*π(r,ATR2):*

	ATR1 - ATR2	ATR1 ∩ ATR2
π(t1,ATR1)	a1 ... ai	ai+1 ... aj
π(t2,ATR2)	b1 ... bi	bi+1 ... bj

	ATR1 ∩ ATR2	ATR2 - ATR1
π(t1,ATR2)	ai+1 ... aj	aj+1 ... an
π(t2,ATR1)	ai+1 ... aj	bj+1 ... bn

Estas dos figuras muestran a *π(r,ATR1)* y *π(r,ATR2).* Cuando se calcula el join se obtienen dos tuplas adicionales *t3* y *t4:*

	R1 - R2	R1 ∩ R2	R2 - R1
t1	a1 ... ai	ai+1 ... aj	aj+1 ... an
t2	b1 ... bi	ai+1 ... aj	bj+1 ... bn
t3	a1 ... ai	ai+1 ... aj	bj+1 ... bn
t4	b1 ... bi	ai+1 ... aj	aj+1 ... an

Si se cumple **(ATR1,ATR2)* siempre que se tengan las tuplas *t1* y *t2* será necesario tener las tuplas *t3* y *t4.* La figura anterior muestra una representación tabular de la jd **(ATR1,ATR2).* Compárese con la figura que representaba tabularmente a la mvd *V->->W.* Si *V = ATR1 ∩ ATR2* y *W = ATR1,* puede verse que las dos representaciones tabulares son la misma. De hecho, **(ATR1,ATR2)* no es más que otra forma de expresar que *ATR1 ∩ ATR2 ->-> ATR1.* Si se emplean las reglas de complementación y aumentación para mvds, puede verse que *ATR1 ∩ ATR2 ->-> ATR1* implica que *ATR1 ∩ ATR2 ->-> ATR2.* Por tanto, **(ATR1,ATR2)* es equivalente a *ATR1 ∩ ATR2 ->-> ATR2.*

En consecuencia, toda jd de la forma **(ATR1,ATR2)* es equivalente a una mvd. Sin embargo existen otras jds que no son equivalentes a una mvd.

Ejemplo: Sea un esquema de relación *RS = ({A,B,C}, DMN, dom, M, SC).* La jd **({A,B}, {B,C}, {A,C})* no es equivalente a ningún conjunto de mvds. Veamos una representación tabular de esta dependencia:

A	B	C
a1	*b1*	*c2*
a2	*b1*	*c1*
a1	*b2*	*c1*
a1	*b1*	*c1*

Para comprobar que ningún conjunto de mvds implica lógicamente a la jd, considérese la tabla previa como una instancia de relación *r.* La instancia *r* satisface a la jd, como puede comprobarse si se calcula $\pi(r,\{A,B\}) * \pi(r, \{B,C\}) * \pi(r,\{A, C\})$. Se verá que el resultado es precisamente *r.* Sin embargo, *r* no satisface ninguna mvd que no sea trivial. Para comprobarlo, verifíquese que *r* no satisface ninguna de las mvds *A->->B, A->->C, B->->A, B->->C, C->->A o C->->B.*

■

5.4 NORMALIZACION SEGUN FDS

5.4.1 Formas normales 1NF, 2NF, 3NF y BCNF

Las *formas normales (normal forms, NFs)* son ciertas propiedades deseables para esquemas de relación en los que se han definido dependencias funcionales. Tales propiedades garantizan que la mayoría de los problemas de redundancia y anomalías no ocurran.

Definición: [Forma normal de Boyce-Codd, BCNF] Un esquema de relación *RS* con un conjunto *SCR* que contiene fds, cumple la *forma normal de Boyce-Codd (BCNF)* si para cada $X \text{-->} Y \in SC^*$ tal que *Y* no pertenece a *X,* tenemos que *X* es una superclave de *RS* (es decir, X es una clave o contiene a una clave). Un esquema de base de datos está en BCNF si todos sus esquemas de relación están en BCNF.

■

Definición: [Atributo primo] Un atributo *A* en el conjunto *ATR* de un esquema *RS* se denomina *primo* si es miembro de alguna clave de RS. En caso contrario se denomina *no primo.*
■

Definición: [3ra. forma normal, 3NF] Un esquema de relación *RS* con un conjunto *SCR* que contiene fds, cumple la *tercera forma normal (3NF)* si para cada $X \rightarrow Y \in SCR^*$ tal que *Y* no pertenece a *X,* tenemos que o bien *X* es una superclave de *RS,* o bien *Y* es primo. Un esquema de base de datos está en *3NF* si todos sus esquemas de relación están en *3NF.*
■

Definición: [2da. forma normal, 2NF] Un esquema de relación *RS* cumple la *segunda forma normal (2NF)* cuando para cada $X \rightarrow Y \in SC^*$ tal que *X* esté propiamente contenida en alguna clave de *RS* y *Y* no esté incluída en *X,* ocurre que *Y* es primo. Un esquema de base de datos está en *2NF* si todos sus esquemas de relaciones están en *2NF.*
■

Observaciones: Si alguna fd $X \rightarrow Y$ viola la 3NF, es debido a uno de los siguientes casos:

■*X es un subconjunto propio de alguna clave.*
En este caso decimos que $X \rightarrow Y$ es una *dependencia parcial.*

■*X no es subconjunto propio de clave alguna.*
En este caso decimos que $X \rightarrow Y$ es una *dependencia transitiva.*

La *3NF* implica que las restricciones $X \rightarrow Y$, donde *X* es una clave y *Y* no es una superclave, y $Y \rightarrow A$, donde *A* es no primo y no está en *XY,* no pueden ocurrir simultáneamente. Esta frase usualmente se resume diciendo que la *dependencia transitiva* $X \rightarrow Y \rightarrow A$ no puede ocurrir. Entonces concluímos que *una relación está en 3NF si no tiene dependencias transitivas.*

La *2NF* dice que no hay tales dependencias transitivas en las que $Y \subseteq X$. Estas transitivas especiales se denominan *dependencias parciales.* Entonces, *una relación está en 2NF si no hay dependencias parciales.* Por lo dicho, *si una esquema de relación o de base de datos está en 3NF, entonces está también en 2NF.* El contrario no se verifica necesariamente.

También, si un esquema de relación o un esquema de base de datos está en BCNF, entonces está también en 3NF (y por supuesto en 2NF). El contrario no se verifica necesariamente.

■

Problema Sea el esquema de relación *VISITANTES = (ATRV, DMNV, domV, MV, SCV),* donde *ATRV = {NUMERO, NOMBRE, DIRECCION, CIUDAD, PAIS}* y *SCV = {NUMERO --> NOMBRE DIRECCION CIUDAD PAIS, CIUDAD --> PAIS}.*

Obviamente, *NUMERO* es una clave. Entonces cualquier conjunto de atributos que contenga a esta clave, es una superclave. Por definición, *NUMERO* es un atributo primo. Debido a que *NUMERO* es claramente la única clave, todos los demás atributos son no primos.

En el esquema, se cumple la dependencia transitiva *NUMERO-->CIUDAD CIUDAD-->PAIS.* ¿Cuál podrá ser la motivación para que se cumpla la *3NF,* o en otras palabras, qué hay de malo en tener una relación que no cumpla la *3NF?* Si un esquema de relación no cumple *3NF,* sabemos que existe una dependencia transitiva *X-->Y-->A.* Si una instancia de *VISITANTES* contiene, por ejemplo, 100 visitantes que provienen de la misma ciudad, entonces la información referente al país al que la ciudad pertenece también se repite 100 veces. Esta redundancia puede causar anomalías de actualización. En el improbable, pero no imposible caso de una corrección de fronteras geográficas, será necesario cambiar el país de 100 tuplas (y olvidar cambiar alguna tupla causará una inconsistencia en la base de datos).

Otro problema con esta dependencia transitiva es que es imposible almacenar la ciudad de la que proviene un visitante, si no se sabe a qué país dicha ciudad pertenece. El problema no ocurriría si los tres atributos no aparecieran todos en el mismo esquema. Es decir, las dependencias transitivas pueden también causar anomalías de inserción y de borrado.

■

Definición: [1ra. forma normal, 1NF] Un esquema de relación *RS* cumple la *primera forma normal (1NF)* cuando el dominio de cada atributo del esquema consiste en valores indivisibles o atómicos, no conjuntos de tuplas o valores de otros dominios más elementales. Es decir, los elementos del conjunto *DMN* del esquema, son conjuntos cuyos elementos son elementales.

■

5.4.2 Descomposición a BCNF y 3NF

A continuación veremos como descomponer a BCNF un esquema de relación de forma tal que la descomposición sea de join sin pérdida. Cualquier esquema puede descomponerse de esta manera. Adicionalmente, cualquier esquema se puede descomponer a 3NF de forma tal que la descomposición sea de join sin pérdida y que además preserve las dependencias. Sin embargo pueden haber descomposiciones a BCNF tales que sean de join sin pérdida pero que no preservan las dependencias.

Algoritmo [Descomposición a 3NF con join sin pérdida]
ENTRADA: RS=(ATR,DMN,dom,M,SC), donde SC es un conjunto de fds.
SALIDA: Una descomposición DS=(PDS,DM,Ø) de RS, en 3NF.

DS := ({RS},DM,Ø) /* DM como aparece en la definición de descomposición */
PARA CADA RSi(ATRi,DMNi,domi,Mi,SCi) en PDS y una X-->Y-->A en RSi
  Reemplazar RSi por π(RSi,YA) y π(RSi,ATR - A)
  en los que los conjuntos de restricciones son reemplazados
  por una proyección de SCi sobre YA y ATR - A respectivamente.
FINPARA
■

Ejemplo: Usando el esquema del ejemplo previo, claramente *A* es la única clave y por ello *C* es un atributo no primo. Como consecuencia, *A-->B-->C* es una dependencia transitiva. Entonces descomponemos *RS* en:

■*RS1 = ({B,C}, DMN1, dom1, M1, {B-->C})*

■*RS2 = ({A,B,D}, DMN2, dom2, M2, {A-->B, A-->D})*

Debido a que ambos esquemas están en 3NF, el algoritmo termina.

■

Cuando definimos la 3NF, nuestro propósito fue eliminar redundancia y anomalías de actualización que pudieran ser causadas por la presencia de fds. Esta redundancia fue removida por descomposición de la base de datos de acuerdo a algunas fds. Queremos ahora contestar a la siguiente pregunta: ¿Qué fds, en general, podemos usar para

descomponer una relación en una forma que tenga sentido? Primero que nada, solicitamos al lector a convencerse de que es suficiente con considerar las fds de la forma *Y-->A* donde *A* es un atributo único. De tales fds *Y-->A* no consideramos, por supuesto a aquellas en las que $A \subseteq Y$ (por ser triviales). Aún más, tampoco tiene sentido usar una fd *Y-->A* donde *Y* es una superclave. Debido a que en este caso todos los atributos del esquema son determinados por *Y,* la descomposición no quitará redundancia alguna. Estas consideraciones nos llevan a la descomposición a BCNF.

Algoritmo [Descomposición a BCNF con join sin pérdida]
ENTRADA: RS=(ATR,DMN,dom,M,SC), donde SC es un conjunto de fds.
SALIDA: Una descomposición DS=(PDS,DM,Ø) de RS, en BCNF.
NOTAS: ¡No necesariamente preserva dependencias!
DS := ({RS},DM,Ø) /* DM como aparece en la definición de descomposición */
PARA CADA RSi(ATRi,DMNi,domi,Mi,SCi) en PDS
y una fd no trivial Y-->A $\in$ SC* tal que Y no es superclave:

> Reemplazar RSi por π(RSi,YA) y π(RSi,ATR - A) en los que los conjuntos de restricciones son reemplazados por una proyección de SCi sobre YA y ATR - A respectivamente.

FINPARA
■

Ejemplo: Considérese el esquema de relación *RS = ({A, B, C, D}, DMN, dom, M, SC)* donde *SC = {A-->B, B-->C, CD-->A, AC-->D}.* Este esquema no está en BCNF porque en *B-->C, B* no es superclave. Entonces descomponemos *RS* en *RS1 = ({B,C}, DMN1, dom1, M1, {B-->C})* y *RS2 = ({A,B,D}, DMN2, dom2, M2, {A-->B, A-->D, BD-->A})*

Debido a que ambos esquemas están en BCNF, el algoritmo termina.
■

Observaciones: Tanto el algorimo que descompone a 3NF como el que descompone a BCNF, usualmente llevan un tiempo exponencial. Podemos preguntarnos por qué hemos discutido la 3NF, ya que la BCNF obviamente es una meta de normalización más deseable. La respuesta es que, desafortunadamente, decidir si un esquema está en BCNF o no, es un problema NP-completo.

Veremos que la descomposición a 3NF puede hacerse en tiempo polinomial. Esto puede hacernos optar por descomponer siempre a 3NF, pero además hay otros motivos que veremos, referentes a la preservación de fds.

■

5.4.3 Preservación de dependencias

Cuando llevamos un esquema a una determinada forma normal, por supuesto que queremos que el resultado normalizado mantenga la información del original. Por ello es que queremos que nuestras descomposiciones no tengan pérdida. Las descomposiciones de join sin pérdida nos permiten reconstruir la instancia del esquema original a partir de la instancia del esquema normalizado. Sin embargo, una base de datos no consiste únicamente de datos, sino también de restricciones, de las cuales también queremos mantener su información durante la normalización, debido a que restricciones más débiles automáticamente implican menos chequeos de integridad rigurosos. Definimos entonces:

Definición: [Representación preservadora de dependencias] Sea *RS = (ATR, DMN, dom, M, SC)* un esquema de relación. Un esquema de base de datos *DS = (PDS, DM, SDC)* con *PDS = { RSi = (ATRi, DMNi, domi, Mi, SCi) | i=1..k }* se denomina *representación preservadora de dependencias de RS* si:

1. $ATRi \subseteq ATR$, *i=1..k.*
2. *ATR1 U ATR2 U ... U ATRk = ATR.*
3. $SDC = \emptyset$
4. *SC* = (SC1 U SC2 U ... U SCk)**

Si además *DS* es una descomposición (de join sin pérdida) de *RS, DS* se denomina *descomposición preservadora de dependencias de RS.*
■

Problema Reconsideremos el esquema *VISITANTES.* Podemos descomponerlo a 3NF de forma tal que la descomposición obtenida es preservadora de las dependencias. Sin embargo hay casos en los que el algoritmo de descomposición a 3NF resulta en una descomposición que *no* preserva dependencias.

Si también reconsideramos el esquema *RS = ({A,B,C,D}, DMN, dom, M, SC)* donde *SC = {A-->B, B-->C, CD-->A, AC-->D},* aplicando el algoritmo de descomposición a BCNF obtuvimos *RS1 = ({B,C}, DMN1, dom1, M1, SC1)* donde *SC1 = {B-->C},* y *RS2 = ({A,B,D},DMN2,dom2,M2,SC2)* donde *SC2 = {A-->B, A-->D, BD-->A }.* Esta descomposición en BCNF no preserva dependencias, ya que es imposible obtener *CD-->A* a partir de *SC1* y *SC2.*

■

Desafortunadamente hay esquemas de relación que no pueden descomponerse a BCNF preservando dependencias. Es decir, si deseamos la BCNF, que es una meta loable, deberemos sacrificar algunas dependencias. Sin embargo hay una alternativa si no se desea hacer el sacrificio. Es posible obtener una descomposición preservadora de dependencias que cumpla la 3NF. El siguiente algoritmo cumple dicho objetivo:

Algoritmo[Descomposición a 3NF preservadora de dependencias]
ENTRADA: RS=(ATR,DMN,dom,M,SC), donde SC es un conjunto de fds.
SALIDA: Una descomposición DS=(PDS,DM,Ø) de RS, en 3NF con preservación de dependencias.

1. Calcular SC'={Yi-->Ai | i=1..m}, un cubrimiento minimal de SC.
2. Buscar una clave arbitraria X de RS.
3. Sea RS0 la proyección de RS según X en la que el conjunto
   de restricciones es reemplazado por la proyección de SC' según X.
4. Construir DS = (PDS,DM,Ø) con:

    ■PDS = {RS0} U {RSi / i=1..m} donde para todo i=1..m, RSi es la proyección de SC' sobre YiAi

    ■DM como en la definición de descomposición

REPETIR
  SI existen i=0..m, j=0..m tales con i<>j tales que
    el esquema RSi está contenido en el esquema RSj ENTONCES
    Eliminar RSi de PDS
  FINSI
HASTA QUE no sea posible hacer más eliminaciones
■

Ejemplo: Sea el esquema *RS=({A,B,C,D},DMN,dom,M,SC)* donde *SC = {A-->B, B-->C, CD-->A, AC-->D}.* Un cubrimiento minimal de *SC* puede ser *SC' = {A-->B, B-->C, CD-->A, A-->D}.* Una clave de *RS* es *A.* Aplicando el algoritmo previo se genera una descomposición consistente en

■*RS0 = ({A}, DMN0, dom0, M0, Ø)*
■*RS1 = ({A,B}, DMN1, dom1, M1, {A-->B})*

- *RS2 = ({B,C}, DMN2, dom2, M2, {B-->C})*
- *RS3 = ({A,C,D}, DMN3, dom3, M3, {CD-->A, A-->C, A-->D})*
- *RS4 = ({A,D}, DMN4, dom4, M4, {A-->D})*

Claramente, *RS0* y *RS4* pueden eliminarse. Por lo tanto nuestro esquema de base de datos final consiste en *RS1, RS2 y RS3.* Fácilmente puede verificarse que representa una descomposición de *RS* en 3NF y con preservación de dependencias.
■

Observaciones: a razón de la inclusión de RS0 en la descomposición generada por el algoritmo es que el cubrimiento minimal SC' (asi como SC) puede no involucrar a todos los atributos en ATR. Claramente, cada atributo de RS debe aparecer en algún esquema de la descomposición. Un atributo que no aparezca en fd alguna de los conjuntos de restricciones dados (SC o SC', no importa) sin embargo está contenido en cada clave de RS. Entonces incluyendo la proyección de RS según una clave arbitraria garantiza que cada atributo aparezca en la descomposición. A pesar de que el último paso del algoritmo elimina alguna redundancia, no se garantiza la ausencia de esquemas de relación redundantes, según puede verse en el siguiente ejemplo.
■

Ejemplo: Sea el esquema *RS = ({A,B,C,D,E}, DMN, dom, M, SC)* donde

SC = {AD-->B, AB-->C, C-->E, BE-->C}.

Claramente *SC* es un cubrimiento minimal. Usando los pasos 1-4 del algoritmo tenemos que *RS* puede descomponerse a 3NF con preservación de dependencias, consistiendo en las proyecciones de *RS* sobre *ADB, ABC, CD y BDC,* en los que los conjuntos de restricciones son reemplazados por las proyecciones correspondientes de *SC.*

El último paso del algoritmo elimina el tercer esquema, dejando las proyecciones de *RS* sobre *ADB, ABC, BDC.* Nótese que el primero y el último esquema son suficientes para una descomposición sin pérdida (la presencia de *BD-->C* en *SC* garantiza que *RS* puede descomponerse en sus proyecciones sobre *BDC* y *ABD*). Por transitividad, *AB-->D* $\in$ *SC*.* Entonces *AB-->D* está en la proyección de *SC* según *ADB. AB-->D* y *BD-->C* que está por supuesto en la proyección de *SC* sobre *BDC* implican a *AB-->C.* Entonces *AB-->C* es también redundante para la preservación de las dependencias. Se deduce que las proyecciones de *RS* sobre *ADB, BCD* llevan a una descomposición preservadora de dependencias de *RS,* en 3NF y con menos esquemas de los que generó el algoritmo.
■

Teorema Cada esquema de relación tiene una descomposición preservadora de dependencias a 3NF, que puede obtenerse con el algoritmo.
■

Corolario Cada esquema de base de datos tiene una descomposición preservadora de dependencias a 3NF, que puede obtenerse con el algoritmo.
■

5.5 NORMALIZACION SEGUN MVDS Y JDS

5.5.1 Forma normal 4NF

Definición: [4ta. forma normal, 4NF] Un esquema de relación *RS = (ATR, DMN, dom, M, SC)* está en *cuarta forma normal (4NF)* si para cada mvd no trivial *X->->Y* con *SC |= X ->-> Y, X* es una superclave en *RS.* En otras palabras, decimos que *RS* está en *4NF* si para todas las mvds *X->->Y* $\in$ *SC*,* donde *Y* es no vacío y *XY* no incluye a todos los atributos de *RS,* se cumple que *X* es superclave de RS.

Un esquema de base de datos está en 4NF si cada uno de sus esquemas de relación está en 4NF.
■

Observaciones: La definición de 4NF sólo difiere de la de BCNF en que se utilizan mvds en vez de fds. Todo esquema en 4NF está en BCNF (para comprobarlo, recordemos la regla de replicación. Si un esquema no está en BCNF, entonces existe una fd no trivial *X-->Y,* donde *X* no es una clave. Puesto que *X-->Y* implica que *X->->Y* (por la regla de replicación), entonces el esquema no puede estar en 4NF.
■

5.5.2 Descomposición a 4NF

La analogía entre 4NF y BCNF se aplica también al algoritmo para descomponer un esquema a 4NF. Este algoritmo es idéntico al de descomposición a BCNF, con la excepción de que se emplean mvds en vez de fds. Por la naturaleza de las mvds y de los joins sin pérdida, el algoritmo genera solamente descomposiciones de join sin pérdida.

Algoritmo de descomposición en 4NF con join sin pérdida

Descomp := (ATR)
Mientras Descomp cambie
  Para cada Ri de Descomp
    Buscar en π*mvd(SC,ATRi)* alguna X→→ Y violación de 4NF en ATRi
      Si existe, reemplazar ATRi por XY y Ri-Y

Devolver Descomp

Cómputo de *πmvd(SC,ATR)*

1) Computar SC*
2) Para cada X→Y en SC*, si X⊆S, agregar X→(Y∩S) a *πmvd(SC,ATR)*
3) Para cada X→→Y en SC*, si X⊆S, agregar X→→(Y∩S) a *πmvd(SC,ATR)*

5.5.3 Forma normal 5NF

Existen casos de esquemas cuyas instancias tienen redundancia, a pesar de estar en 4FN y que admiten una descomposición en más de 2 subesquemas con join sin pérdida que elimina esta redundancia. La restricción que representa este hecho es la *dependencia join (join dependence, jd).*

Definición: [Quinta forma normal, 5NF, PJNF] Sea *RS = (ATR, DMN, dom, M, SC)* un esquema de relación donde *SC* es un conjunto que contiene jds. Decimos que *RS* está en *quinta forma normal (5NF)* o en la *forma normal de proyección dependencia (PJNF)* con respecto a *SC* si para toda *jd *(RS1, ..., RSp)* implicada por *SC* que se aplica en *RS,* o bien la jd es trivial o los todo *ATRi* de cada *RSi* es superclave de RS.

Un esquema de base de datos está en 5NF si cada uno de sus esquemas de relación está en 5NF.
■

Observaciones: Puesto que toda mvd es también una jd, es fácil comprobar que cualquier esquema en PJNF está también en 4NF. Así, en general, puede no ser posible encontrar una descomposición a PJNF que preserve las dependencias.
■

5.6 FORMAS NORMALES GENERALES

Hasta ahora la normalización se ha enfocado a la definición de algún tipo de restricción (fd, mvd o jd), aplicada a un esquema *RS = (ATR, DMN, dom, M, SC),* para después emplear dicha restricción para definir una forma normal. La *forma normal de dominio-clave* está basada en tres conceptos:

1. Declaración de dominio. Sea $A \in ATR$ y sea $D \in DMN$. Sea $(A,D) \in dom$. La declaración de dominio requiere que para toda tupla t, $t[A] \in D$.

2. Declaración de clave. Sea $K \in ATR$. La declaración de clave requiere que K sea una superclave del esquema, es decir que $K \rightarrow ATR$. Nótese que todas las declaraciones de clave son fds, pero no viceversa.

3. Restricción general. La restricción general es un predicado en el conjunto de todas las instancias de un esquema dado, y generalmente se expresa en una forma previamente establecida, por ejemplo, lógica de primer orden. Las dependencias que se han estudiado son ejemplos de restricciones generales.

A continuación se dará un ejemplo de una restricción general que no sea una dependencia funcional.

Ejemplo: En una base de datos bancaria, sea un esquema *RS* con un conjunto de atributos *ATR = {SUCURSAL, NROCUENTA, SALDO}.* Supóngase que todas las cuentas corrientes cuyo número comienza con "9" son cuentas especiales de alta tasa de interés, con un saldo mínimo de $2.500. En este caso se incluiría como restricción general: *"si el primer dígito de t[NROCUENTA] es '9', entonces* $t[SALDO] \geq 2500$*."*
■

Observaciones: Las declaraciones de dominio y las de clave pueden probarse fácilmente en un DBMS real. Sin embargo, es posible que sea muy costoso (en tiempo y espacio) probar una restricción general. El objetivo de un diseño de base de datos en forma normal de dominio-clave es permitir la prueba de restricciones generales empleando solamente las de dominio y clave.
■

Definición: [Forma normal de dominio-clave, domain-key normal form, DKNF] Sea *D* un conjunto de restricciones de dominio y *K* uno de restricciones de clave para un esquema *RS.* Las restricciones generales para *RS* se denotarán por medio de *G.* El esquema *RS* está en la *forma normal de dominio-clave (DKNF),* si *D U K* implica lógicamente a *G.*
■

Ejemplo: Volviendo a la restricción general que se dio arriba para las cuentas corrientes, la restricción implica que, tal como está diseñada, la base de datos no está en DKNF. Para crear un diseño DKNF, se requieren dos subesquemas:

- *RS1 = ({SUCURSAL, CUENTA, SALDO}, DMN1, dom1, M1, SC1)*
- *RS2 = ({SUCURSAL, CUENTA, SALDO}, DMN2, dom2, M2, SC2)*

Se conservan todas las dependencias que se tenían en RS como restricciones generales. Las restricciones de dominio de RS2 exigen que el número de cuenta comience con 9 y el saldo sea mayor de 2500. Las limitantes de dominio de RS1 exigen que el número de cuenta no comience con 9.
■

A continuación se comparará la DKNF con las demás formas normales que se han estudiado. En las otras formas normales no se tomaron en cuenta restricciones de dominio. Se supuso (implícitamente) que el dominio de cada atributo era algún dominio apropiado. Para cada forma normal, se permitió una forma restringida de restricción general (un conjunto de fds, mvds o de jds). Por tanto, es posible volver a escribir las definiciones de PJNF, 4NF, BCNF y 3NF de manera tal que se vean como casos especiales de DKNF.

Sea *RS = ({A1,A2,...,An}, DMN, dom, M, G)* un esquema de relación. Sea *dom(Ai)* el dominio del atributo *Ai,* siendo todos estos dominios infinitos. Todas las restricciones de dominio *D* son entonces de la forma $Ai \subseteq dom(Ai)$. Sean las restricciones generales un conjunto *G* de fds, mvds o jds. Si *F* es el conjunto de fds en *G,* sea *K* el conjunto de restricciones de clave de todas las fds no triviales en F* de la forma *X-->ATR.* El esquema *RS* está en PJNF si y sólo si está en DKNF con respecto a *D, K, G.*

Observaciones: Una consecuencia de DKNF es que se eliminan todas las anomalías de inserción y borrado. La DKNF representa una forma normal "extrema", porque permite restricciones arbitrarias, no sólo dependencias, y sin embargo ayuda a probar estas restricciones en forma eficiente. Por supuesto, si un esquema no está en DKNF, puede pasarse a DKNF por medio de una descomposición, aunque, como ya se ha visto, tales

descomposiciones no siempre preservan las dependencias.
■

5.7 ESQUEMAS NFNF

Aunque no se ha hecho mayor énfasis en la 1NF, se ha supuesto su cumplimiento desde que se introdujo el modelo relacional. Un esquema está en 1NF si los dominios de y todos los atributos son atómicos. Un dominio es atómico si se considera a los elementos del dominio como unidades indivisibles. La cuestión importante en la 1NF no es el dominio en sí, sino la forma como se emplean los elementos del dominio en la base de datos. Por ejemplo, el dominio de todos los enteros no sería atómico si se considera a cada número entero como una lista ordenada de dígitos.

Los últimos avances en la teoría y práctica han puesto en duda la legitimidad de suponer que se cumple la 1NF, creándose esquemas que se denominan *no 1NF (non-first normal form, NFNF).* Vamos a estudiar un caso:

Problema Considérese un sistema de recuperación de documentos. Para cada documento, se almacena la siguiente información: título del documento, lista de autores, fecha y lista de palabras clave.

Es evidente que si se define un esquema de relación para la información antes mencionada, varios dominios serán no atómicos:

- **Autores.** Un documento puede tener varios autores. Sin embargo, es posible que se quiera localizar todos los documentos de los cuales Jones haya sido uno de los autores. En este caso lo que interesa es una parte componente del elemento del dominio *conjunto de autores.*

- **Palabras claves.** Si se almacena un conjunto de palabras clave para cada documento, lo normal es que se quiera localizar todos los documentos entre cuyas palabras clave se encuentren una o más palabras determinadas. Así el dominio de *lista de palabras clave* se considera no atómico.

- **Fecha.** A diferencia de *palabras clave* y *autores, fecha* no tiene un dominio de conjunto. Sin embargo, puede considerarse que *fecha* está constituido por los

subcampos dia, mes y año. Esto hace que el dominio de *fecha* sea no atómico.

■

A continuación se muestra una instancia del esquema DOCUMENTO. Esta instancia puede representarse en 1NF, pero la instancia que resulta es difícil de manejar.

TITULO	AUTORES	FECHA	PCLAVE
Plan de ventas	*{Smith,Jones}*	*1 abr 79*	*{utilidades, estrategia}*
Reporte de situación	*{Jones, Frick}*	*17 jun 85*	*{utilidades, personal}*

Como en 1NF es preciso tener dominios atómicos, y se requiere tener acceso a autores y palabras clave separados, se necesita una tupla para cada pareja *<palabra clave, autor>.* El atributo *FECHA* se sustituye por tres atributos, uno para cada subcampo:

TITULO	AUTOR	DIA	MES	AÑO	PCLAVE
Plan de ventas	*Smith*	*1*	*abr*	*79*	*utilidades*
Plan de ventas	*Jones*	*1*	*abr*	*79*	*utilidades*
Plan de ventas	*Smith*	*1*	*abr*	*79*	*estrategia*
Plan de ventas	*Jones*	*1*	*abr*	*79*	*estrategia*
Reporte de situación	*Jones*	*17*	*jun*	*85*	*utilidades*
Reporte de situación	*Frick*	*17*	*jun*	*85*	*utilidades*
Reporte de situación	*Jones*	*17*	*jun*	*85*	*personal*
Reporte de situación	*Frick*	*17*	*jun*	*85*	*personal*

Esta relación se vuelve mucho más manejable si se supone que se cumplen *TITULO->->AUTOR, TITULO->->PCLAVE, TITULO-->DIA MES AÑO.* Con lo cual puede descomponerse el esquema a 4NF empleando los subesquemas:

- *RS1 = ({TITULO,AUTOR}, DMN1, dom1, M1, SC1)*
- *RS2 = ({TITULO,PCLAVE}, DMN2, dom2, M2, SC2)*
- *RS3 = ({TITULO,DIA,MES,AÑO}, DMN3, dom3, M3, SC3)*

Observaciones: Aunque la 1NF puede representar en forma adecuada la base de datos de documentos, es probable que la representación que no está en 1NF sea un modelo más fácil de comprender para el usuario promedio de un sistema de recuperación de documentos. Los usuarios de este tipo de sistemas conciben la base de datos en términos del diseño que no está en 1NF. El diseño 4NF requeriría de los usuarios la inclusión de joins en sus consultas, lo cual hará más compleja la interacción con el sistema.

Bibliografía.

[Korth 1992]. Korth, H.F., Silberschatz, A. 1992. *Fundamentos de Bases de Datos,* McGraw-Hill.

[Paredaens 1989]. Paredaens, J., De Bra, P., Gyssens, M., Van Gucht, D. 1989. *The Structure of the Relational Data Model,* Springer-Verlag.

[Ullman 1982]. Ullman, J.D. 1982. *Principles of Database Systems,* Mass. Press.

■

---oOo---