Capítulo 7: Diseño de bases de datos relacionales
Normalización

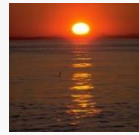
Fundamentos de Bases de datos, 5ª Edición.

Capítulo 7: Diseño de bases de datos relacionales

- Características de los buenos diseños relacionales
- Dominios atómicos y primera forma normal
- Descomposición mediante dependencias funcionales
- Teoría de las dependencias funcionales
- Algoritmos de descomposición
- Descomposición mediante dependencias multivaloradas
- Más formas normales
- Proceso de diseño de bases de datos
- Modelado de datos temporales

The Banking Schema

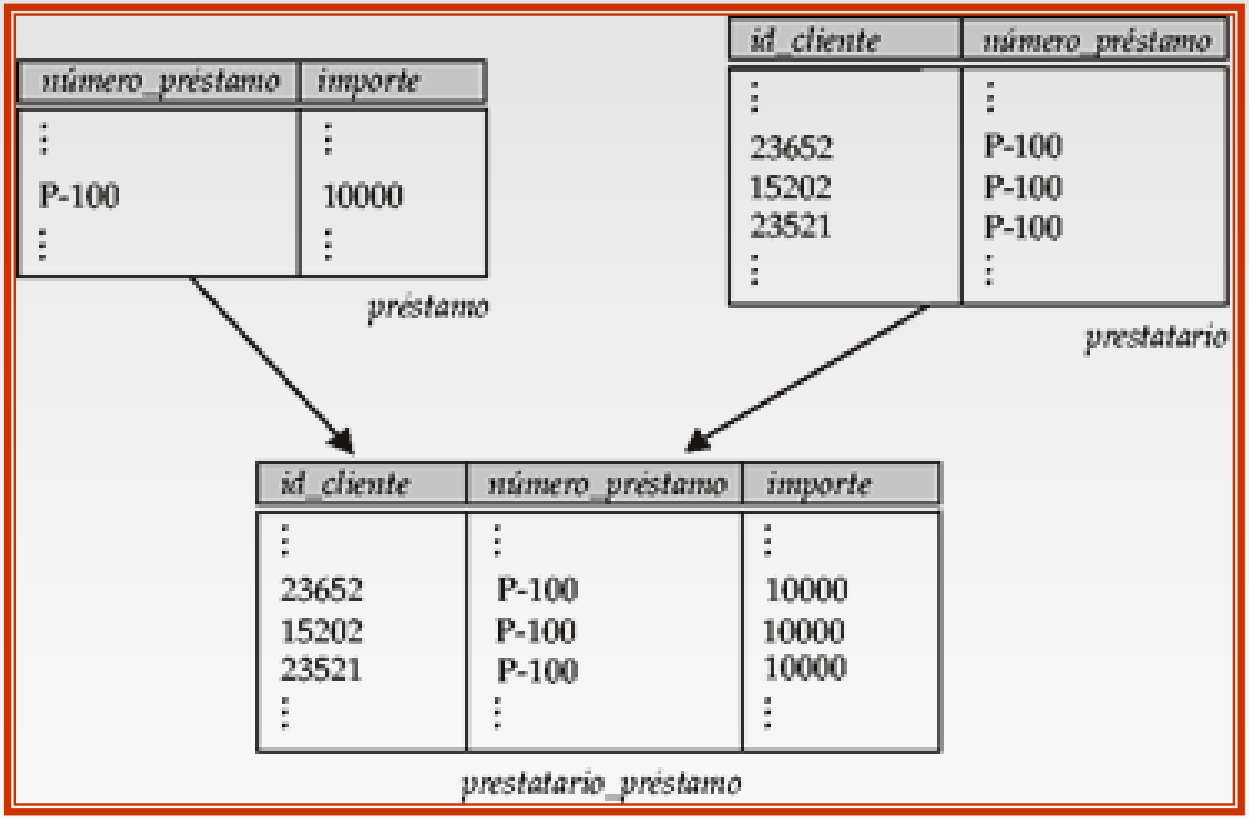
- *sucursal (nombre_sucursal, ciudad_sucursal, activos)*
- *cliente (id_cliente, nombre_cliente, calle_cliente, ciudad_cliente)*
- *préstamo (número_préstamo, importe)*
- *cuenta (número_cuenta, saldo)*
- *empleado (id_empleado, nombre_empleado, número_teléfono, fecha_contratación)*
- *nombre_subordinado (id_empleado, nombre_subordinado)*
- *sucursal_cuenta (número_cuenta, nombre_sucursal)*
- *sucursal_préstamo (número_préstamo, nombre_sucursal)*
- *prestatario (id_cliente, número_préstamo)*
- *impositor (id_cliente, número_cuenta)*
- *asesor (id_cliente, id_empleado, tipo)*
- *trabaja_para (id_empleado_trabajador, id_empleado_jefe)*
- *pago (número_préstamo, número_pago, fecha_pago, importe_pago)*
- *cuenta_ahorro (número_cuenta, tasa_interés)*
- *cuenta_corriente (número_cuenta, importe_descubierto)*



¿Combinar esquemas?

- Suponga que se combina *prestatario* y *préstamo* para obtener

  prestatario_préstamo = (*id_cliente*, *número_préstamo*, *importe*)
- El resultado genera una repetición de información (P-100 en el ejemplo)

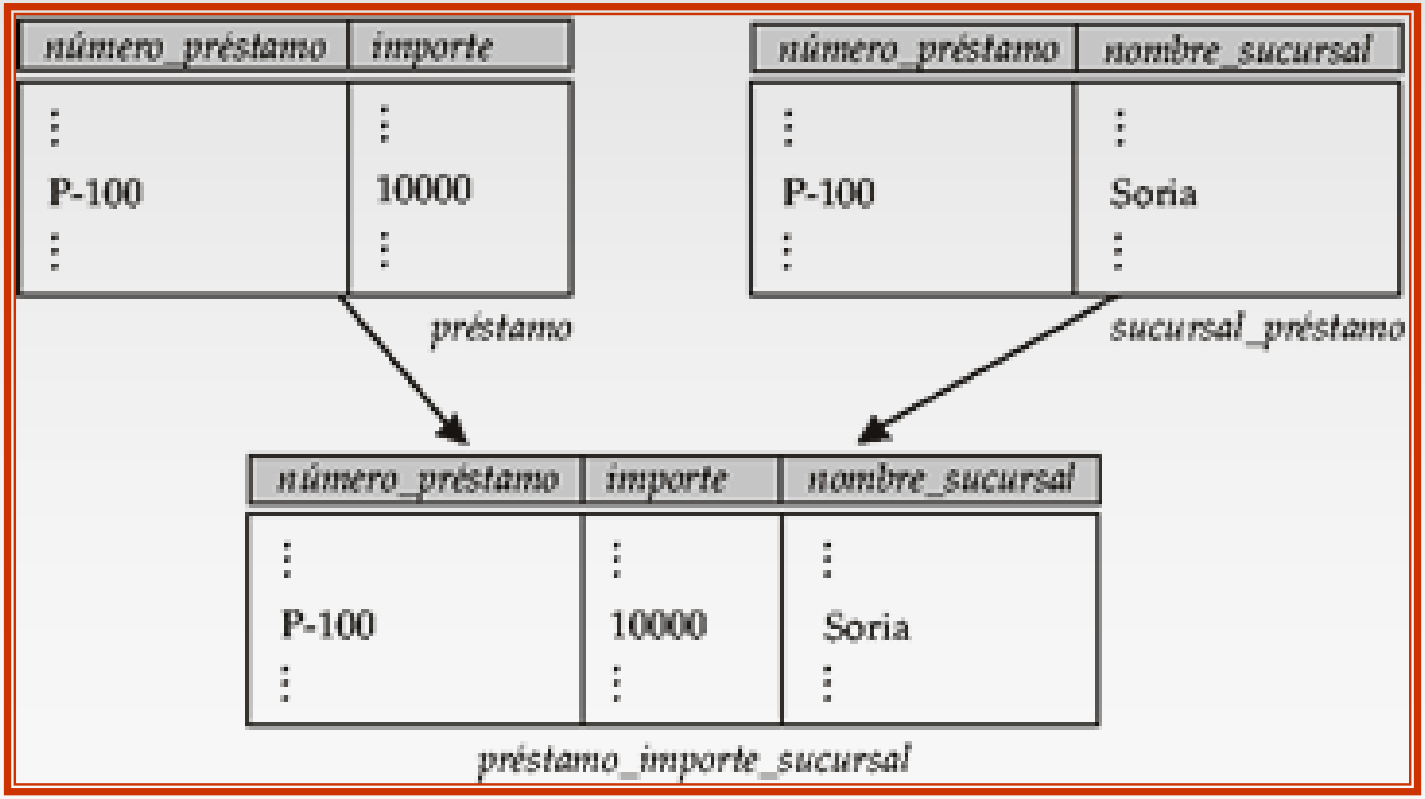


Combinación de esquemas sin repetición

- Considere combinar *sucursal_préstamo* y *préstamo*

  préstamo_importe_sucursal = (*número_préstamo, importe, nombre_sucursal)*

- No hay repeticiones (como se ve en el ejemplo)



¿Y con esquemas pequeños?

- Suponga que se ha empezado con *prestatario_préstamo*. Como se podría dividir (**descomponer**) en *prestatario* y *préstamo*?
- Escriba una regla “si hubiese un esquema (*número_préstamo, importe*), entonces *número_préstamo* podría ser una clave candidata?
- Escribalo como una **dependencia funcional**

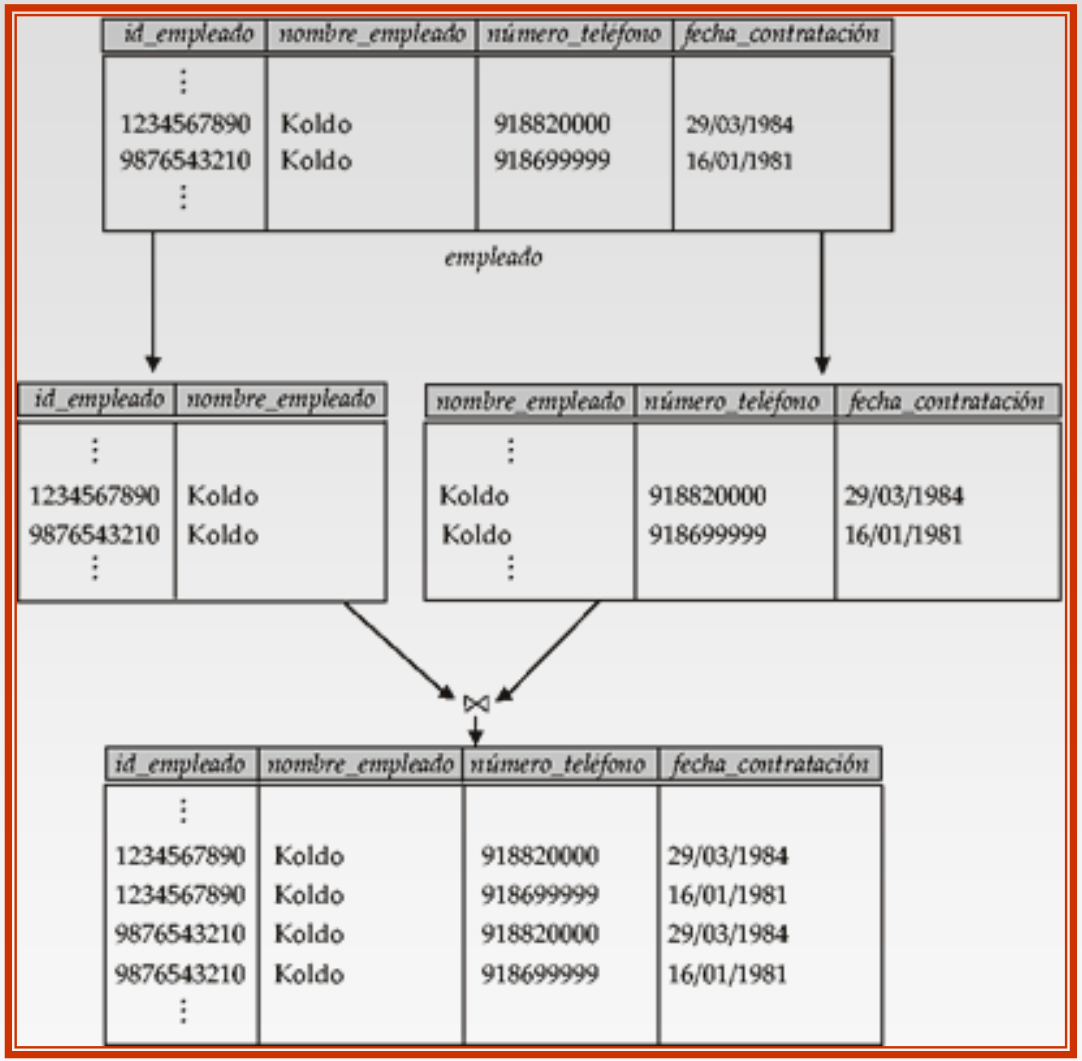
  $$número\_préstamo \rightarrow importe$$

- En *prestatario_préstamo*, como *número_préstamo* no es una clave candidata, el importe del préstamo se puede llegar a repetir. Esto indica la necesidad de descomponer *prestatario_préstamo*
- No todas las descomposiciones son buenas. Suponga que se desea descomponer *empleado* en

  empleado1 = (*id_empleado*, *nombre_empleado*)

  empleado2 = (*nombre_empleado*, *número_teléfono*, *fecha_contratación*)
- En las siguientes transparencias se muestra cómo se pierde información – no se puede reconstruir la relación *empleado* original, y por tanto, es una descomposición con pérdidas

Una mala descomposición



Primera forma normal

- Un dominio es atómico si se considera que los elementos del dominio son unidades indivisibles
  - Ejemplos de dominios no atómicos:
    - Conjunto de nombres, atributos compuestos
    - Identificación de números como CS101 que se pueden descomponer en partes
- Un esquema relacional R está en primera forma normal si los dominios de todos los atributos de R son atómicos
- Los valores no atómicos complican el almacenamiento de datos redundantes (repetidos) de almacenamiento y estimulación
  - Por ejemplo el conjunto de cuentas almacenadas con cada cliente, y el conjunto de titulares almacenados con cada cuenta
  - Se asume que todas las relaciones están en la primera forma normal (repasar este concepto en el Capítulo 9)

Primera forma normal (cont)

- La atomicidad es realmente una propiedad de como se utilizan los elementos del dominio.
  - Por ejemplo las cadenas se considerarían normalmente indivisibles
  - Supóngase que a los estudiantes se les da números de identificación que son cadenas de la forma *IN*0012 o *EE1127*
  - Si los dos primeros caracteres se extraen para hallar el departamento, el dominio de los números de identificación no es atómico.
  - Hacerlo así es una mala idea: implica codificar la información en el programa de la aplicación en vez de en la base de datos.

Objetivo — Idear una teoría para lo siguiente

- Decidir si una relación particular *R* es una forma “buena”.
- En el caso de que una relación *R* no esté en la forma “buena”, descomponerla en un conjunto de relaciones $\{R_1, R_2, ..., R_n\}$ de forma que
  - cada relación esté en la forma buena
  - la descomposición es una descomposición de reunión sin pérdida
- Nuestra teoría se basa en:
  - dependencias funcionales
  - dependencias multivaloradas

Dependencias funcionales

- Restricciones del conjunto de relaciones legales.
- Requieren que el valor de un cierto conjunto de atributos determine únicamente el valor para otro conjunto de atributos.
- Una dependencia funcional es una generalización de la noción de una *clave.*

Dependencias funcionales (cont.)

- Sea R el esquema de una relación

$$\alpha \subseteq R \;\; y \;\; \beta \subseteq R$$

- La dependencia funcional

$$\alpha \rightarrow \beta$$

se cumple en *R* si y sólo si en cualquier relación legal *r*(R), para todos los pares de tuplas t_1 y t_2 de *r* que concuerdan con los atributos α, también concuerdan con los atributos β. Es decir,

$$t_1[\alpha] = t_2[\alpha] \;\; \Rightarrow \;\; t_1[\beta] = t_2[\beta]$$

- Ejemplo: Considérese *r (A,B)* con el siguiente ejemplo de *r.*

1	4
1	5
3	7

- En este ejemplo, $A \rightarrow B$ **NO** se cumple, pero $B \rightarrow A$ se cumple.

Dependencias funcionales (cont.)

- *K* es una superclave para el esquema de relación *R* si y sólo si $K \rightarrow R$
- *K* es una clave candidata para *R* si y sólo si
  - $K \rightarrow R$, y
  - para ningún $\alpha \subset K$, $\alpha \rightarrow R$
- Las dependencias funcionales nos permiten expresar restricciones que no se pueden expresar utilizando superclaves. Considérese el esquema:

  prestatario_préstamo = (id_cliente, número-préstamo, importe).

  Se espera que este conjunto de dependencias funcionales cumpla:

  número-préstamo $\rightarrow$ *importe*

  pero no se espera que se cumpla lo siguiente:

  número-préstamo $\rightarrow$ *nombre-cliente*

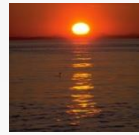
Uso de las dependencias funcionales

- Se utilizan las dependencias funcionales para:
  - comprobar las relaciones y ver si son legales según un conjunto dado de dependencias funcionales.
    - Si una relación r es legal según el conjunto F de dependencias funcionales, se dice que r satisface F.
  - especificar las restricciones del conjunto de relaciones legales
    - se dice que F se cumple en *R* si todas las relaciones legales de *R* satisfacen el conjunto de dependencias funcionales *F.*
- Nota: Un ejemplo específico de un esquema de relación puede satisfacer una dependencia funcional incluso si la dependencia funcional no cumple todos los ejemplares legales.
  - Por ejemplo, un ejemplar específico de *préstamo* puede, por casualidad, satisfacer

    importe $\rightarrow$ *nombre-cliente*

Dependencias funcionales (cont.)

- Una dependencia funcional es trivial si la satisficen todos los ejemplares de una relación
  - *Por ejemplo.*
    - *nombre_cliente, número_préstamo* $\rightarrow$ *nombre_cliente*
    - *nombre_cliente* $\rightarrow$ *nombre_cliente*
  - En general, $\alpha \rightarrow \beta$ es trivial si $\beta \subseteq \alpha$



Cierre de un conjunto de dependencias funcionales

- Dado un conjunto *F* de dependencias funcionales, hay otras ciertas dependencias funcionales que están implicadas lógicamente por *F*.
  - Por ejemplo si $A \rightarrow B$ y $B \rightarrow C$, entonces se puede inferir que $A \rightarrow C$
- El conjunto de **todas** las dependencias funcionales lógicamente implicadas en *F* es el ***cierre*** de *F*.
- Se indica el *cierre* de *F* por F^+.
- F^+ es un superconjunto de *F*.

Forma normal de Boyce–Codd

Un esquema de relación R está en FNBC respecto a un conjunto de dependencias funcionales F si, para todas las dependencias funcionales de F+ de la forma

$$\alpha \rightarrow \beta$$

donde $\alpha \subseteq$ R y $\beta \subseteq$ R, se cumple al menos una de las siguientes condiciones:

- $\alpha \rightarrow \beta$ es trivial (es decir $\beta \subseteq \alpha$)
- α es una superclave de *R*

Ejemplo de esquema que no está en FNBC:

prestatario_préstamo = (*id_cliente, número_préstamo, importe*)

porque *número_préstamo* $\rightarrow$ *importe* pero *número_préstamo* no es una superclave

Descomposición de un esquema en FNBC

- Suponga que se tiene un esquema *R* y una dependencia no trivial $\alpha \rightarrow \beta$ genera un incumplimiento de la FNBC.

  Se descompone *R* en:

  - $(\alpha \cup \beta)$
  - $(R - (\beta - \alpha))$
- En el ejemplo,
  - α = *prestatario_préstamo*
  - β = *importe*

  Y se sustituye *prestatatio_préstamo* por
  - $(\alpha \cup \beta)$ = (*número_préstamo, importe*)
  - $(R - (\beta - \alpha))$ = (*id_cliente, número_préstamo*)

FNBC y conservación de dependencias

- Las restricciones, incluyendo las dependencias funcionales, son muy costosas de comprobar excepto si pertenecen a una única relación
- Si es suficiente comprobar sólo aquellas dependencias de la relaciones individuales de una descomposición para asegurar que se cumplen *todas* las dependencias funcionales, entonces dicha descomposición *preserva las dependencias.*
- Como no es siempre posible conseguir tanto FNBC y conservación de las dependencias, se considera una forma normal más débil, denominada, *tercera forma normal*.

Tercera forma normal

- Un esquema de relación *R* está en la tercera forma normal (3FN) si para todo:

  $$\alpha \rightarrow \beta \text{ en } F^+$$

  al menos se cumple lo siguiente:

  - $\alpha \rightarrow \beta$ es trivial (es decir, $\beta \in \alpha$)
  - α es una superclave de *R*
  - Cada atributo *A* de $\beta - \alpha$ está contenido en una clave candidata de *R.*

    (NOTA: cada atributo puede estar en una clave candidata diferente)

- Si una relación está en FNBC está en 3FN (ya que debe cumplirse en FNBC una de las dos primeras condiciones anteriores).
- La tercera condición es una relajación mínima de FNBC para asegurar la conservación de dependencias (se verá la causa más adelante).

Objetivos de la normalización

- Sea *R* un esquema de relación con un conjunto *F* de dependencias funcionales.
- Se decide si un esquema de relación *R* está en una forma “buena”.
- En el caso de que un esquema de relación *R* no esté en una forma “buena”, se descompone en un conjunto de esquemas de relación $\{R_1, R_2, ..., R_n\}$ tales que
  - todos los esquemas de relación este en una forma buena
  - La descomposición sea una descomposición sin pérdidas frente a la unión
  - preferiblemente, la descomposición debería ser que conservase las dependencias.

¿Cómo de buena es la FNBC?

- Hay esquemas de bases de datos en FNBC que no parecen estar suficientemente normalizados
- Suponga la siguiente base de datos

  clases (*curso, profesor, libro*)

  tal que (*c, t, b*) $\in$ *clases* significa que *t* puede impartir *c*, y *b* es el libro de texto obligatorio para *c*
- En la base de datos se supone que hay una lista de todos los cursos con el conjunto de profesores que pueden impartirlos como profesores y el conjunto de libros que son necesarios para el curso (independientemente de quién sea el profesor)

¿Cómo de buena es la FNBC? (cont.)

curso	*profesor*	*libro*
bases de datos	Angel	Conceptos BD
bases de datos	Angel	Ullman
bases de datos	Luis	Conceptos BD
bases de datos	Luis	Ullman
bases de datos	Manuel	Conceptos BD
bases de datos	Manuel	Ullman
sistemas operativos	Angel	Conceptos SO
sistemas operativos	Angel	Stallings
sistemas operativos	Pedro	Conceptos SO
sistemas operativos	Pedro	Stallings

clases

- No existen dependencias funcionales no triviales y, por tanto, la relación está en FNBC
- Inserción de anomalías, por ejemplo, Si Mónica es una nueva profesora que puede dar bases de datos se necesitan dos tuplas

  (bases de datos, Mónica, Conceptos BD)
  (bases de datos, Mónica, Ullman)

¿Cómo de buena es la FNBC? (Cont.)

- Por tanto, es mejor descomponer *clases* en:

curso	*profesor*
base de datos	Angel
base de datos	Luis
base de datos	Manuel
sistemas operativos	Angel
sistemas operativos	Pedro

enseña

curso	*libro*
base de datos	Conceptos BD
base de datos	Ullman
sistemas operativos	Conceptos SO
sistemas operativos	Stallings

texto

De aquí se sugiere la necesidad de mayores formas normales, como la Cuarta Forma Normal (4FN), que se verá más adelante

Teoría de dependencia funcional

- A continuación se considera la teoría formal que nos indica que dependencias funcionales se implican lógicamente de un conjunto dado de dependencias funcionales
- Después se desarrollan los algoritmos que generan descomposiciones sin pérdidas en FNBC y 3FN.
- Por último se desarrollan algoritmos que comprueban si una descomposición conserva las dependencias.

Cierre de un conjunto de dependencias funcionales

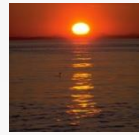
- Dado un conjunto *F* de dependencias funcionales, hay otras ciertas dependencias funcionales que están implicadas lógicamente por *F*.
  - Por ejemplo si $A \rightarrow B$ y $B \rightarrow C$, entonces se puede inferir que $A \rightarrow C$
- El conjunto de todas las dependencias funcionales lógicamente implicadas en *F* es el *cierre* de *F*.
- Se indica el *cierre* de *F* por F^+.
- Se puede hallar todo lo concerniente a F^+ aplicando los Axiomas de Armstrong:
  - si $\beta \subseteq \alpha$, entonces $\alpha \rightarrow \beta$ **(reflexividad)**
  - si $\alpha \rightarrow \beta$, entonces $\gamma\ \alpha \rightarrow \gamma\ \beta$ **(aumentatividad)**
  - si $\alpha \rightarrow \beta$, y $\beta \rightarrow \gamma$, entonces $\alpha \rightarrow \gamma$ **(transitividad)**
- Estas reglas son
  - correctas (generan sólo dependencias funcionales que se cumplen en la realidad) y
  - completas (generan todas las dependencias funcionales que se cumplen).

Ejemplo

- $R = (A, B, C, G, H, I)$
  $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- algunos miembros de F^+
  - $A \rightarrow H$
    - por transitividad de $A \rightarrow B$ *and* $B \rightarrow H$
  - $AG \rightarrow I$
    - por aumentatividad $A \rightarrow C$ con G, para lograr que $AG \rightarrow CG$ y después transitividad con $CG \rightarrow I$
  - $CG \rightarrow HI$
    - por aumentatividad $CG \rightarrow I$ se infiere $CG \rightarrow CGI$,

      y por aumentatividad de $CG \rightarrow H$ se infiere $CGI \rightarrow HI$,

      y después por transitividad



Procedimiento para calcular F^+

- Para calcular el cierre de un conjunto de dependencias funcionales F:

$F^+ = F$
repeat
  for each dependencia funcional f de F^+
    aplicar las reglas de reflexividad y de aumentatividad sobre f
    añadir las dependencias funcionales resultantes a F^+
  for each pareja de dependencias funcionales f_1 y f_2 de F^+
    if f_1 y f_2 se pueden combinar mediante la transitividad
      then añadir la dependencia funcional resultante a F^+
until F^+ no cambie más

NOTA: Más adelante se verá un procedimiento alternativo para este cometido

Cierre de dependencias funcionales (cont.)

- Se puede simplificar más el cálculo manual de F^+ utilizando las siguientes reglas adicionales.
  - Si se cumple que $\alpha \rightarrow \beta$ y que $\alpha \rightarrow \gamma$, entonces se cumple $\alpha \rightarrow \beta\gamma$ **(unión)**
  - Si se cumple que $\alpha \rightarrow \beta\gamma$, entonces se cumple $\alpha \rightarrow \beta$ y que $\alpha \rightarrow \gamma$ **(descomposición)**
  - Si se cumple que $\alpha \rightarrow \beta$ y que $\gamma\beta \rightarrow \delta$, entonces se cumple que $\alpha\gamma \rightarrow \delta$ **(pseudotransitividad)**

  Las reglas anteriores se pueden inferir a partir de los axiomas de Armstrong.

Cierre de conjuntos de atributos

- Para comprobar si un conjunto de atributos α es superclave, hay que diseñar un algoritmo para el cálculo del conjunto de atributos determinados funcionalmente por α.
- Una manera de hacerlo es calcular F^+, tomar todas las dependencias funcionales con α como término de la izquierda y tomar la unión de los términos de la derecha de todas las dependencias funciones.
- **Sin embargo esto puede ser bastante costoso, ya que F^+ puede ser de gran tamaño.**

- Intentamos construir un algoritmo para determinar si α es superclave de la relación R para el conjunto de F dependencias funcionales dado.

Cierre de conjuntos de atributos

- Dado un conjunto de atributos α, se define el *cierre* de α bajo *F* (denotado por α^+) como el conjunto de atributos que se determina funcionalmente por α bajo *F:*

$$\alpha \rightarrow \beta \text{ está en } F^+ \Leftrightarrow \beta \subseteq \alpha^+$$

- Algoritmo para calcular α^+, el cierre de α bajo *F*

```
resultado := α;
while (cambios en resultado) do
        for each β → γ in F do
          begin
            if β ⊆ resultado then  resultado := resultado ∪ γ
          end
```

Ejemplo de cierre del conjunto de atributos

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- $(AG)^+$
  1. *resultado* = AG
  2. *resultado* = $ABCG$ $(A \rightarrow C$ y $A \rightarrow B)$
  3. *resultado* = $ABCGH$ $(CG \rightarrow H$ y $CG \subseteq AGBC)$
  4. *resultado* = $ABCGHI$ $(CG \rightarrow I$ y $CG \subseteq AGBCH)$
- ¿Es AG una clave candidata?
  1. ¿Es AG una superclave?
     1. ¿Se cumple $AG \rightarrow R$? == Es $(AG)^+ \supseteq$ R
  2. ¿Es algún subconjunto de AG una superclave?
     1. ¿Se cumple $A \rightarrow R$? == Is $(A)^+ \supseteq$ R
     2. ¿Se cumple $G \rightarrow R$? == Is $(G)^+ \supseteq$ R

Usos del cierre de atributos

Hay varios usos del algoritmo de cierre de atributos:

- Prueba de superclave:
  - Para probar si α es una superclave, se calcula α^+, y se comprueba si α^+ contiene todos los atributos de R.
- Prueba de dependencias funcionales
  - Para comprobar si se cumple una dependencia funcional $\alpha \rightarrow \beta$ (o, en otras palabras, está en F^+), sólo se comprueba si $\beta \subseteq \alpha^+$.
  - Es decir, se calcula α^+ utilizando el cierre de atributos, y después se comprueba si contiene β.
  - Es una prueba simple y económica, y muy útil
- Cálculo del cierre de F
  - Para cada $\gamma \subseteq R$, se halla el cierre γ^+, y para cada $S \subseteq \gamma^+$, se introduce una dependencia funcional $\gamma \rightarrow S$.

Recubrimiento canónico

- Los conjuntos de dependencias funcionales pueden tener dependencias redundantes que pueden inferirse desde otras
  - Ejemplo: $A \rightarrow C$ es redundante en: $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow C\}$
  - Las partes de una dependencia funcional pueden ser redundantes
    - Por ejemplo en RHS: $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow CD\}$ se puede simplificar a

      $$\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow D\}$$

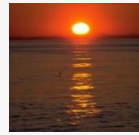
    - ejemplo en LHS: $\{A \rightarrow B, \quad B \rightarrow C, \quad AC \rightarrow D\}$ se puede simplificar a

      $$\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow D\}$$

- Intuitivamente, un recubrimiento canónico de F es un conjunto “mínimo” de dependencias funcionales equivalentes a F, sin dependencias redundantes o que tienen partes redundantes de las dependencias

Atributos raros

- Considérese un conjunto F de dependencias funcionales y la dependencia funcional $\alpha \rightarrow \beta$ de F.
  - El atributo A es raro en α si $A \in \alpha$
    y F implica lógicamente a $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.
  - El atributo A es raro en β si $A \in \beta$
    y el conjunto de dependencias funcionales
    $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ implica lógicamente a F.
- *Nota:* la implicación en la dirección opuesta es trivial en cada uno de los casos anteriores, ya que una dependencia funcional "más fuerte" siempre implica una más débil
- Ejemplo: Dado $F = \{A \rightarrow C, AB \rightarrow C\}$
  - B es raro en $AB \rightarrow C$ porque $A \rightarrow C$ implica lógicamente $AB \rightarrow C$.
- Ejemplo: Dado $F = \{A \rightarrow C, AB \rightarrow CD\}$
  - C es raro en $AB \rightarrow CD$ ya que $A \rightarrow C$ se puede inferir incluso después de borrar C



Comprobación si un atributo es raro

- Considérese un conjunto *F* de dependencias funcionales y la dependencia funcional $\alpha \rightarrow \beta$ de *F*.
- Para comprobar si el atributo $A \in \alpha$ es raro en α
  - calcular $(A - \{\alpha\})^+$ utilizando las dependencias de *F*
  - comprobar que $(A - \{\alpha\})^+$ incluye α; si es así, *A* es raro
- Para comprobar si el atributo $A \in \beta$ es raro en β
  - calcular α^+ utilizando sólo las dependencias de
    $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$,
  - comprobar que α^+ incluye *A;* si es así, *A* es raro

Recubrimiento canónico

- Un *recubrimiento canónico* de F es un conjunto de dependencias F_c tales que
  - F implica lógicamente todas las dependencias de F_c, y
  - F_c implica lógicamente todas las dependencias de F, y
  - Ninguna dependencia funcional de F_c contiene un atributo raro, y
  - El lado izquierdo de cada dependencia funcional de F_c es único.
- Para calcular un recubrimiento canónico de F:

  repeat

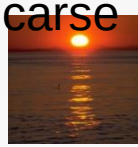
  Utilizar la regla de unión para sustituir las dependencias de F

  $\alpha_1 \rightarrow \beta_1$ y $\alpha_1 \rightarrow \beta_1$ con $\alpha_1 \rightarrow \beta_1 \beta_2$

  Hallar una dependencia funcional $\alpha \rightarrow \beta$ con un atributo raro en α o en β

  Si se halla un atributo raro, borrarlo de $\alpha \rightarrow \beta$

  until F no cambie
- Nota: La regla de la unión puede aplicarse después de que se hayan borrado algunos atributos raros, de forma que tengan que volver a aplicarse



Cálculo de un recubrimiento canónico

- $R = (A, B, C)$
  $F = \{A \rightarrow BC$
  $B \rightarrow C$
  $A \rightarrow B$
  $AB \rightarrow C\}$
- Combinar $A \rightarrow BC$ y $A \rightarrow B$ dentro de $A \rightarrow BC$
  - El conjunto es ahora $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$
- A es raro en $AB \rightarrow C$
  - Comprobar si el resultado de eliminar A de $AB \rightarrow C$ se puede deducir de otras dependencias
    - Si, de hecho $B \rightarrow C$ ya está presente
  - El conjunto es ahora $\{A \rightarrow BC, B \rightarrow C\}$
- C es raro en $A \rightarrow BC$
  - Comprobar si $A \rightarrow$ C se deduce lógicamente de $A \rightarrow B$ y el resto de dependencia
    - Sí: usando la transitividad de $A \rightarrow B$ y $B \rightarrow$ C.
- El recubrimiento canónico es:

$$A \rightarrow B$$
$$B \rightarrow C$$

Descomposición de reunión sin pérdida

- En el caso de $R = (R_1, R_2)$, es necesario que para todas las posibles relaciones *r en* el esquema *R*

$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

- Una descomposición de R en R_1 y R_2 es una reunión sin pérdida si y sólo si al menos una de las siguientes dependencias está en F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$

Ejemplo

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$
  - Se puede descomponer de dos formas diferentes
- $R_1 = (A, B)$, $R_2 = (B, C)$
  - Descomposición de reunión sin pérdida:

    $$R_1 \cap R_2 = \{B\} \text{ y } B \rightarrow BC$$

  - Dependencia que conserva
- $R_1 = (A, B)$, $R_2 = (A, C)$
  - Descomposición de reunión sin pérdida:

    $$R_1 \cap R_2 = \{A\} \text{ y } A \rightarrow AB$$

  - Dependencia que no conserva
    (no puede comprobar $B \rightarrow C$ sin calcular $R_1 \bowtie R_2$)

Conservación de dependencias

- Sea F_i el conjunto de dependencia F^+ que sólo incluye atributos en R_i.
  - Una descomposición conserva las dependencias, si

    $$(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$$

  - Si no se cumple, la comprobación en las actualizaciones del incumplimiento de las dependendencias puede implicar el cálculo de uniones, que es muy costoso.

Comprobación de la conservación de dependencias

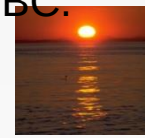
- Para comprobar si se conserva una dependencia $\alpha \rightarrow \beta$ en una descomposición de R dentro de R_1, R_2, …, R_n se aplica la siguiente prueba simplificada (con el cierre de atributo hecho w.r.t. F)
  - *resultado* = α
    while (cambios en *resultado*) **do**
      for each R_i de la descomposición
        $t = (resultado \cap R_i)^+ \cap R_i$
        $resultado = resultado \cup t$
  - Si *resultado* contiene todos los atributos de β, entonces se conserva la dependencia funcional $\alpha \rightarrow \beta$.
- Se aplica la prueba en todas las dependencias de F para comprobar si una descomposición es la conservación de la dependencia
- Este procedimiento tarda un tiempo polinómico, en vez del tiempo exponencial necesario para calcular F^+ y $(F_1 \cup F_2 \cup \ldots \cup F_n)^+$

Ejemplo

- $R = (A, B, C)$
  $F = \{A \rightarrow B$
  $B \rightarrow C\}$
  Clave = $\{A\}$
- R no está en FNBC
- Descomposición $R_1 = (A, B)$, $R_2 = (B, C)$
  - R_1 y R_2 en FNBC
  - Descomposición de reunión sin pérdida
  - Conserva las dependencias

Prueba de la FNBC

- Para comprobar si una dependencia no trivial $\alpha \rightarrow \beta$ incumple la forma normal de Boyce–Codd (FNBC)
  1. calcular α^+ (el cierre de los atributos de α), y
  2. comprobar que incluye todos los atributos de *R*, es decir, que es una superclave de *R*.
- Prueba simplificada: Para comprobar si un esquema de relación *R* con un conjunto dado de dependencias funcionales F está en FNBC, basta con comprobar sólo las dependencias del conjunto dado *F* en búsqueda de la violación de FNBC, sin tener que comprobar todas las dependencias de F^+.
  - Se puede observar que si ninguna de las dependencias de *F* produce una violación de FNBC, entonces tampoco ninguna de las dependencias de F^+ producirá una violación de FNBC.
- Sin embargo, el uso único de F es incorrecto cuando se comprueba una relación en una descomposición de R
  - Por ejemplo considérese *R* (*A, B, C, D*), con $F = \{ A \rightarrow B, B \rightarrow C\}$
    - Descompóngase *R* en $R_1(A,B)$ y $R_2(A,C,D)$
    - Ninguna de las dependencias de *F* contienen sólo los atributos de (*A,C,D*) así que se debería dejar de pensar que R_2 satisface a FNBC.
    - De hecho, la dependencia $A \rightarrow C$ de F^+ muestra que R_2 no está en FNBC.



Comprobación de la descomposición de FNBC

- Para comprobar si una relación R_i de una descomposición de R está en FNBC,
  - Bien probar R_i para FNBC con respecto a la restricción de F a R_i (es decir, todos las DF de F^+ que contienen sólo los atributos de R_i)
  - o utilizar el conjunto original de dependencias F que se incluyen en R, pero con la siguiente prueba:
    - para cada conjunto de atributos $\alpha \subseteq R_i$, comprobar que α^+ (el cierre del atributo de α) bien no incluye atributos de $R_i - \alpha$, o incluye todos los atributos de R_i.
    - Si la condición se viola por algún $\alpha \rightarrow \beta$ de F, la dependencia
      $\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$
      se puede mostrar para incluirse en R_i, y R_i viola FNBC.
    - Utilizamos la anterior dependencia para descomponer R_i

Algoritmo de descomposición de FNBC

resultado := {*R*};
hecho := falso;
cálculo F^+;
while (not *hecho)* **do**
  if (hay un esquema R_i de *resultado* que no esté en FNBC)
    then begin
      sea $\alpha \rightarrow \beta$ una dependencia funcional no
        trivial que se cumple en R_i
        de forma que $\alpha \rightarrow R_i$ no esté en F^+,
        y $\alpha \cap \beta = \varnothing$;
      resultado := (*resultado* $- R_i$) $\cup$ ($R_i - \beta$) $\cup$ (α, β);
    end
  else *hecho* := **true;**

Nota: cada R_i está en FNBC, y la descomposición es la reunión sin pérdida.

Ejemplo de descomposición de FNBC

- $R = (A, B, C)$
  $F = \{A \rightarrow B$
  $B \rightarrow C\}$
  Clave = $\{A\}$
- R no está en FNBC ($B \rightarrow C$ pero B no es una superclave)
- Descomposición
  - $R_1 = (B, C)$
  - $R_2 = (A,B)$

Ejemplo de descomposición de FNBC

- Relación *R* original y dependencia funciona *F*

  R = (nombre_sucursal, ciudad_sucursal, activo, nombre_cliente, número_préstamo, importe)
  F = {nombre_sucursal → activo ciudad_sucursal número_préstamo → importe nombre_sucursal}
  Clave = *{número_préstamo, nombre_cliente}*

- Descomposición
  - R_1 = (*nombre_sucursal, ciudad_sucursal, activo)*
  - R_2 = *(nombre_sucursal, nombre_cliente, número_préstamo, importe)*
  - R_3 = *(nombre_sucursal, número_préstamo, importe)*
  - R_4 = *(nombre_cliente, número_préstamo)*

- Descomposición final

  R_1, R_3, R_4

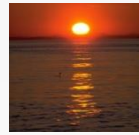
FNBC y su conservación de dependencias

No siempre es posible obtener una descomposición FNBC que conserve las dependencias

- $R = (J, K, L)$
  $F = \{JK \rightarrow L$
  $L \rightarrow K\}$
  Dos claves de candidato = JK y JL
- R no está en FNBC
- Fallará alguna descomposición de R al conservar

$$JK \rightarrow L$$

Ello implica que para comprobar $JK \rightarrow L$ se requiere una unión



Tercera forma normal: Motivación

- Hay algunas situaciones en las que
  - FNBC no está en la conservación de dependencias, y
  - es importante la comprobación eficiente de la violación de DF en las actualizaciones
- Solución: definir una forma normal más débil, llamada tercera forma normal.
  - Permite alguna redundancia (con problemas en los resultados; se verán ejemplos más adelante)
  - Pero se pueden comprobar las DF en las relaciones individuales sin calcular una reunión.
  - Hay siempre una reunión sin pérdida, la descomposición de conservación de dependencias dentro de 3FN.

Ejemplo de 3FN

- Relación R:
  - $R = (J, K, L)$
    $F = \{JK \rightarrow L, L \rightarrow K\}$
  - Dos claves candidatas: JK y JL
  - R está en 3FN

$JK \rightarrow L$	JK es una superclave
$L \rightarrow K$	K esta contenida en una clave candidata

Redundancia en 3FN

- Existen algunas redundancias en este esquema
- Ejemplo de problemas debido a la redundancia de 3FN
  - $R = (J, K, L)$
    $F = \{JK \rightarrow L, L \rightarrow K\}$

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
null	l_2	k_2

- repetición de la información (p.e., la relación l_1, k_1)
- necesita utilizar valores nulos (p.e., para representar la relación l_2, k_2 donde no existe valor que se corresponda con J).

Prueba de 3FN

- Optimización: Se necesita comprobar sólo las DF de *F*, no se necesita comprobar todos las DF de F^+.
- Se utiliza el cierre de atributos para comprobar, de cada dependencia $\alpha \rightarrow \beta$, si α es una superclave.
- Si α no es una superclave, se tiene que verificar si cada atributo de β está incluido en una clave candidata de *R*
  - esta comprobación resulta bastante más costosa, ya que implica buscar las claves candidatas
  - La comprobación de 3FN resulta ser NP-duro
  - Curiosamente, la descomposición en la tercera forma normal (descrita brevemente) se puede hacer en tiempo polinomial

Algoritmo de descomposición de 3FN

Sea F_c un recubrimiento canónico de F;
$i := 0$;
for each dependencia funcional $\alpha \rightarrow \beta$ de F_c **do**
  if ninguno de los esquemas R_j, $1 \leq j \leq i$ contiene $\alpha\ \beta$
    then begin
      $i := i + 1$;
      $R_i := \alpha\ \beta$
    end
if ninguno de los esquemas R_j, $1 \leq j \leq i$ contiene una clave candidata de R
  then begin
    $i := i + 1$;
    R_i := cualquier clave candidata de R;
  end
return $(R_1, R_2, ..., R_i)$

Algoritmo de descomposición de 3FN (cont.)

- El algoritmo anterior asegura que:
  - cada esquema de relación R_i está en 3FN
  - la descomposición es la conservación de dependencias y la reunión sin pérdida

Comparación entre FNBC y 3FN

- Siempre es posible descomponer una relación en un conjunto de relaciones que están en 3FN tal que
  - la descomposición se hace sin pérdida
  - se conservan las dependencias
- Siempre es posible descomponer una relación en un conjunto de relaciones que están en FNBC tal que
  - la descomposición se hace sin pérdida
  - puede que no sea posible conservar las dependencias.

Objetivos del diseño

- El objetivo de un diseño de bases de datos relacionales es:
  - FNBC.
  - Reunión sin pérdida.
  - Conservación de las dependencias.
- Si no resulta posible conseguir esto, se elige entre
  - La pérdida de conservación de las dependencias
  - La redundancia debido al uso de 3FN
- Curiosamente, SQL no proporciona una manera directa de especificar las dependencias funcionales distinta de las superclaves.
- Aunque se tenga una descomposición que conserve las dependencias, si se utiliza SQL no se podrá comprobar de forma eficiente una dependencia funcional cuyo lado izquierdo no sea una clave.
- Por lo tanto, en caso de que no se pueda obtener una descomposición en la FNBC, que conserve las dependencias, suele resultar preferible optar por la 3FN y emplear técnicas como vistas para reducir el costo de la comprobación de las DF

Dependencias multivaloradas (MVD)

- Sea R un esquema de relación y sea $\alpha \subseteq R$ y $\beta \subseteq R$. La *dependencia multivalorada*

$$\alpha \twoheadrightarrow \beta$$

se cumple en R si en toda relación legal $r(R)$, para todo par de tuplas t_1 y t_2 de r tales que $t_1[\alpha] = t_2[\alpha]$, existen tuplas t_3 y t_4 de r tales que:

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$
$$t_3[\beta] = t_1[\beta]$$
$$t_3[R - \beta] = t_2[R - \beta]$$
$$t_4[\beta] = t_2[\beta]$$
$$t_4[R - \beta] = t_1[R - \beta]$$

MVD (cont.)

- Representación tabular de $\alpha \rightarrow\rightarrow \beta$

	α	β	$R - \alpha - \beta$
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$
t_4	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

Ejemplo

- Sea *R* un esquema de relación con un conjunto de atributos que se dividen en 3 subconjuntos no vacíos.

  $$Y, Z, W$$

- Se dice que $Y \rightarrow\rightarrow Z$ (*Y* multidetermina *Z*)
  si y sólo si para todas las relaciones posibles *r(R)*

  $$< y_1, z_1, w_1 > \in r \text{ y } < y_2, z_2, w_2 > \in r$$

  entonces

  $$< y_1, z_1, w_2 > \in r \text{ y } < y_1, z_2, w_1 > \in r$$

- Tener en cuenta que ya que el comportamiento de *Z* y *W* es idéntico se cumple que $Y \rightarrow\rightarrow Z$ si $Y \rightarrow\rightarrow W$

Ejemplo (cont.)

- En nuestro ejemplo:

$$curso \twoheadrightarrow profesor$$
$$curso \twoheadrightarrow libro$$

- La definición formal anterior se supone que es para formalizar la noción de que dado un valor particular de Y (*curso*) se le ha asociado con un conjunto de valores Z *(profesor)* y un conjunto de valores W *(libro)*, y estos dos conjuntos son en algún sentido independientes el uno del otro.
- Nota:
  - Si $Y \rightarrow Z$ entonces $Y \twoheadrightarrow Z$
  - Efectivamente se tienen (en la notación anterior) $Z_1 = Z_2$ La reclamación continúa.

Uso de las dependencias multivaloradas

- Las dependencias multivaloradas se utilizan de dos maneras:
  1. Para verificar las relaciones y determinar si son legales bajo un conjunto dado de dependencias funcionales y multivaloradas
  2. Para especificar restricciones del conjunto de relaciones legales. De este modo, sólo habrá que preocuparse de las relaciones que satisfagan un conjunto dado de dependencias funcionales y multivaloradas.
- Si una relación *r* falla al satisfacer una dependencia multivalorada dada, se puede crear unas relaciones *r´* que satisfagan la dependencia multivalorada añadiendo tuplas

Teoría de las MVD

- De la definición de dependencia multivalorada, se puede derivar la siguiente regla:
  - Si $\alpha \rightarrow \beta$, entonces $\alpha \twoheadrightarrow \beta$

  Es decir, cada dependencia funcional es también una dependencia multivalorada
- El **cierre** D^+ de D es el conjunto de todas las dependencias funcionales y multivaloradas implicadas lógicamente por *D*.
  - Se puede calcular D^+ a partir de *D*, utilizando las definiciones formales de las dependencias funcionales y de las dependencias multivaloradas.
  - Con este razonamiento se puede trabajar con las dependencias multivaloradas muy sencillas, que parece ser en la práctica lo más común
  - Para las dependencias complejas, es mejor razonar con conjuntos de dependencias empleando un sistema de reglas de inferencia (véase el Apéndice C).

Cuarta forma normal

- Un esquema de relación *R* está en 4FN con respecto a un conjunto *D* de dependencias funcionales y multivaloradas si para todas las dependencias multivaloradas de D+ de la forma $\alpha \rightarrow\rightarrow \beta$, donde $\alpha \subseteq$ R y $\beta \subseteq$ R, se cumple, como mínimo, una de las siguientes condiciones:
  - $\alpha \rightarrow\rightarrow \beta$ es trivial (es decir, $\beta \subseteq \alpha$ or $\alpha \cup \beta$ = *R)*
  - α es una superclave del esquema *R*
- Si una relación está en 4FN está en FNBC

Restricción de dependencias multivaloradas

- La restricción de D a R_i es el conjunto D_i que consta de
  - Todas las dependencias funcionales de D^+ que incluyen sólo atributos de R_i
  - Todas las dependencias multivaloradas de la forma

    $$\alpha \rightarrow\!\!\!\rightarrow (\beta \cap R_i)$$

    donde $\alpha \subseteq R_i$ y $\alpha \rightarrow\!\!\!\rightarrow \beta$ está en D^+

Algoritmo de descomposición de 4FN

resultado: = {*R*};
hecho:= falso;
calcular D^+;
Supongamos que D_i denota la restricción de D^+ a R_i

while (**not** hecho)
  if (hay un esquema Ri en resultado que no se halla en 4FN) **then**
    begin
      supongamos que $\alpha \rightarrow\rightarrow \beta$ es una dependencia multivalorada no trivial que se cumple en Ri de forma que $\alpha \rightarrow R_i$ no se halla en D_i, y $\alpha \cap \beta = \phi$;
      resultado := (*resultado* - R_i) $\cup$ (R_i - β) $\cup$ (α, β);
    end
  else *hecho*:= verdadero;

Nota: cada R_i se halla en 4FN, y la descomposición es la reunión sin pérdida

Ejemplo

- $R = (A, B, C, G, H, I)$

  $F = \{ A \rightarrow\rightarrow B$

  $B \rightarrow\rightarrow HI$

  $CG \rightarrow\rightarrow H \}$

- R no se halla en 4FN ya que $A \rightarrow\rightarrow B$ y A no es una superclave de R
- Descomposición

  a) $R_1 = (A, B)$ (R_1 se halla en 4NF)

  b) $R_2 = (A, C, G, H, I)$ (R_2 no se halla en 4NF)

  c) $R_3 = (C, G, H)$ (R_3 se halla en 4NF)

  d) $R_4 = (A, C, G, I)$ (R_4 no se halla en 4NF)

- Ya que $A \rightarrow\rightarrow B$ y $B \rightarrow\rightarrow HI$, $A \rightarrow\rightarrow HI$, $A \rightarrow\rightarrow I$

  e) $R_5 = (A, I)$ (R_5 se halla en 4NF)

  f) $R_6 = (A, C, G)$ (R_6 se halla en 4NF)

Otras formas normales

- Las **dependencias de reunión** generalizan las dependencias multivaloradas
  - Llevan a la **forma normal de reunión por proyección (FNRP**) (también llamada **quinta forma normal**)
- Hay una clase de restricciones todavía más generales, que lleva a una forma normal denominada **forma normal de dominios y claves (FNDC)**.
- Problema con estas restricciones generalizadas: es difícil razonar con ellas, y no hay un conjunto de reglas de inferencia seguras y completas.
- Por consiguiente, rara vez se utilizan

Proceso general del diseño de bases de datos

- Se da por supuesto que se dispone de un esquema R
  - *R* puede haberse generado al convertir un diagrama E-R en un conjunto de tablas.
  - *R* puede haberse generado de una única relación que contenga *todos* los atributos de interés (llamada **relación universal**)
  - La normalización divide *R* en relaciones más pequeñas.
  - *R* puede haber sido el resultado de algún diseño al uso de relaciones, que hay que comprobar-convertir después a la forma normal.

El modelo ER y la normalización

- Cuando se define con cuidado un diagrama E-R, identificando correctamente todas las entidades, las tablas generadas a partir del diagrama E-R no necesitan más normalización.
- No obstante, en un diseño real (imperfecto) puede haber DF entre los atributos sin clave y los otros atributos de la entidad
- Ejemplo la entidad *empleado* con los atributos *número_departamento* y *dirección_departamento*, y una DF *número_departamento* $\rightarrow$ *dirección_departamento*
  - Un buen diseño habría creado una entidad *departamento*
- Son posibles las DF de atributos sin clave de un conjunto de relaciones, pero son raras --- la mayoría de las relaciones son binarias

Desnormalización por rendimiento

- Puede que se quiera utilizar esquemas no normalizados de rendimiento
- Por ejemplo visualizar *nombre_cliente* junto con *número_cuenta* y *saldo* requiere la reunión de *cuenta* con *impositor*
- Alternativa 1: Utilizar la relación no normalizada que contenga los atributos de *cuenta* además de *impositor* con todos los atributos anteriores
  - Búsqueda más rápida
  - Espacio y tiempo de ejecución adicional para las actualizaciones
  - Trabajo de codificación adicional para el programador y posibilidad de error en códigos adicionales
- Alternativa 2: utiliza una vista materializada definida como
  cuenta ⋈ impositor
  - Los mismos beneficios e inconvenientes que la anterior, excepto que no hay trabajo adicional para el programador y evita posibles errores

Otros problemas de diseño

- Algunos aspectos del diseño de bases de datos no los aborda la normalización
- Ejemplos de mal diseño de bases de datos, que se deben evitar:

  En vez de beneficios(id_empresa, año, importe) utilizar
  - *beneficios_2000, beneficios_2001, beneficios_2002*, etc., todo en el esquema (id_empresa, beneficios).
    - En el caso anterior están en FNBC, pero las consultas se hacen difíciles a través de los años y se necesitan nuevas tablas cada año
  - *empresa_año*(*id_empresa, beneficios_2000, beneficios_2001, beneficios_2002*)
    - También en FNBC, pero también se hacen difíciles las consultas a través de los años y se necesita un atributo nuevo cada año.
    - Es un ejemplo de una **tabla cruzada**, en la que los valores de un atributo se convierten en nombres de columnas
    - Utilizados en hojas de cálculo, y en las herramientas de análisis de datos

Fin del capítulo