Bases de Datos 1

Alejandra Lliteras
alejandra.lliteras@lifia.info.unlp.edu.ar

En la clase anterior…

Conceptos vistos

Teoría de diseño para bases de datos relaciones

- **Relación**
- **Anomalía**
- **Dependencia Funcional**
- **Dependencia Funcional Trivial**
- **Clave**
  - **Clave Candidata**
  - **Super Clave**
- **Axiomas de Armstrong**

Teoría de diseño para bases de datos relaciones

Clausura de un conjunto de ATRIBUTOS

Teoría de diseño de BBDD relacionales

- Clave de una relación
  - Los atributos {A1, A2,…,An} son la clave de una relación R si cumplen:
    - {A1, A2,…,An} determinan funcionalmente a todos los restantes atributos de la relación R
    - No existe un subconjunto de {A1, A2,…,An} que determine funcionalmente a todos los atributos de R *-Esto implica que una clave es un conjunto minimal-*

Teoría de diseño de BBDD relacionales

¿Cómo se puede comprobar que un conjunto de atributos {A1, A2,…,An} es la clave de una relación?

Teoría de diseño de BBDD relacionales

- Clausura de un conjunto de atributos (X^+)

Sea **F** un conjunto de dependencias funcionales sobre un esquema R y sea **X** un subconjunto de R.

La clausura de X respecto de F, se denota $\mathbf{X}^+$ y es el conjunto de atributos **A** tal que la dependencia **X** -> **A** puede deducirse a partir de F, por los axiomas de Armstrong

Es decir, $\mathbf{X}^+$ son todos los atributos determinados por X en R

Teoría de diseño de BBDD relacionales

- **Clausura de un conjunto de atributos (X^+)**

Algoritmo para encontrar X^+

Result:= X
While (hay cambios en result) do
  For (cada dependencia funcional Y->Z en F) do
    if ($Y \subseteq$ result) then
      result := result $\cup$ Z

Teoría de diseño para bases de datos relaciones

Algoritmo para hallar la clausura de un conjunto de ATRIBUTOS

¿Cómo funciona?

Teoría de diseño de BBDD relacionales

- Ejemplo
  - Dada la relación: PERSONA(dni, nombre, edad, fechaNacimiento, nroLegajo, carrera)
    - *Donde*
      - *Una persona puede cursar diversas carreras*
      - *Nombre indica como se llama la persona*
      - *Una persona posee un único número de legajo asignado para cada carrera que cursa*
      - *Un número de legajo pertenece a una sola persona de una carrera*

        df1) dni -> nombre, edad, fechaNac

        df2) nroLegajo, carrera -> dni

        df3) dni, carrera -> nroLegajo

Clave candidata 1 (cc1): {nroLegajo, carrera }

Clave candidata 2 (cc2): {dni, carrera }

Teoría de diseño de BBDD relacionales

- Ejemplo
  - Dada la relación: PERSONA(dni, nombre, edad, fechaNacimiento, nroLegajo, carrera)

    df1) dni -> nombre, edad, fechaNac

    df2) nroLegajo, carrera -> dni

    df3) dni, carrera -> nroLegajo

Clave candidata 1 (cc1): {nroLegajo, carrera }

Clave candidata 2 (cc2): {dni, carrera }

- Por ejemplo, podríamos preguntarnos: ¿Es cierto que a partir de los atributos de cc1, puedo recuperar los atributos restantes de PERSONA?
  - Para ello podemos ejecutar el algoritmo de X^+ instanciándolo con la información de PERSONA

Teoría de diseño de BBDD relacionales

Result:= X
While (hay cambios en result) do
    For (cada dependencia funcional Y->Z en F) do
        if (Y $\subseteq$ result) then
            result := result $\cup$ Z

F={dni -> nombre, edad, fechaNac;
nroLegajo, carrera -> dni ;
dni, carrera -> nroLegajo}

PERSONA(dni, nombre, edad, fechaNacimiento, nroLegajo, carrera)

Hallar (nroLegajo, carrera **)$^+$**

Result= (nroLegajo, carrera)

Paso 1) Tomamos la dep. fun: dni -> nombre, edad, fechaNac**,** {dni} **no está incluido en result, no agrego nada a result => (**nroLegajo, carrera)

Paso 2) Tomamos la dep. fun. nroLegajo, carrera -> dni **,** {nroLegajo, carrera} **está incluido en result, agrego** {dni} **a result => (**nroLegajo, carrera, dni)

Paso 3) Tomamos la dep. fun. dni, carrera -> nroLegajo**,** {dni, carrera} **está incluido en result, agrego** {nroLegajo} **a result => (**nroLegajo, carrera, dni **)**

Como ya recorrí todas las dependencias funcionales y result cambió vuelvo a iterar

Paso 1) Tomamos la dep. fun. dni -> nombre, edad, fechaNac**,** {dni} **está incluido en result, agrego** {nombre, edad, fechaNac} **a result => (**nroLegajo, carrera, dni , nombre, edad, fechaNac**)**

RECUPERE A TODOS LOS ATRIBUTOS DE PERSONA!

Teoría de diseño de BBDD relacionales

- De la misma manera podríamos haber probado con la otra clave candidata
- Recordar que instanciar este algoritmo NO asegura que el conjunto de atributos de partida sea mínimo

Teoría de diseño de BBDD relacionales

- Hasta ahora vimos, para una relación R
  - Cómo hallar dependencias funcionales
    - Y su conjunto completo mediante los Axiomas de Armstrong
  - Cómo hallar las claves candidatas
    - Y como usar el algoritmo de la clausura de atributos para corroborar que a partir de un subconjunto de atributos de R se puede recuperar al resto de los atributos de la relación (aunque éste no asegura que dicho subconjunto sea mínimo)
  - Considerando las dependencias funcionales y las claves candidatas, veremos un proceso para generar relaciones que cumplan ciertas condiciones de un buen diseño.

¿ Cómo generar relaciones que cumplan ciertas condiciones de un buen diseño?

Teoría de diseño de BBDD relacionales

- **Descomposición**
  - Es una forma aceptada de eliminar las anomalías de una relación
  - Consiste en separar los atributos de una relación en dos nuevas relaciones
  - No se debe perder
    - Información
    - Dependencias funcionales

Teoría de diseño de BBDD relacionales

- **Descomposición**
  - No se debe perder
    - Información
    - Dependencias funcionales

Teoría de diseño de BBDD relacionales

- Descomposición
  - No se debe perder
    - Información
    - Dependencias funcionales

- **Pérdida de Información**

  Si a un esquema **R**, se lo particiona en dos subesquemas **R1** y **R2** se debe cumplir alguna de las siguientes condiciones:

  R1 ∩ R2 es clave en el esquema **R1**

  o

  R1 ∩ R2 es clave en el esquema **R2**

Teoría de diseño de BBDD relacionales

- Descomposición
  - No se debe perder
    - Información
    - Dependencias funcionales

- **Pérdida de dependencias funcionales**

  - verificar que cada una de las dependencias funcionales que valían en el esquema **R**, sigan valiendo en alguna de las particiones **Ri.**

Cuando se chequean las dependencias funcionales pueden ocurrir dos cosas:

  - los atributos de la dependencia funcional original quedaron todos incluidos en alguna de las particiones generadas

  - los atributos de la dependencia funcional original quedaron distribuidos en mas de una partición

Teoría de diseño para bases de datos relaciones

Algoritmo para analizar la pérdida de dependencias funcionales

Teoría de diseño de BBDD relacionales

Algoritmo para analizar la pérdida de dependencias funcionales

Res = x

Mientras Res cambia

Para i= 1 to cant_de_ particiones_realizadas

Res = Res ∪ ((Res ∩ Ri)$^+$ ∩ Ri)

Donde:

X es el determinante de la dependencia funcional que quiero analizar si se perdió.

Res es un temporal donde se van guardando los atributos que se pueden recuperar en cada iteración.

Ri es el conjunto de atributos de la partición representada por Ri

((Res ∩ Ri)$^+$ ∩ Ri), asegura que quedan sólo los atributos que pertenecen a la partición que se está tratando.

Teoría de diseño de BBDD relacionales

Supongamos:

R(a,b,c,d)

Donde vale:

F = {a→b, b→c, c→d, d→a}

Y alguien propone el siguiente particionamiento del esquema R

R1(a,b)
R2(b,c)
R3(c,d)

No se pierde información

¿Se pierden dependencias funcionales?

a→b vale en **R1**
b→c vale en **R2**
c→d vale en **R3**

d →a??????

Res = ?
Mientras Res cambia
  Para i= 1 to cant_de_ particiones_realizadas
    Res = Res ∪ ((Res ∩ Ri)$^+$ ∩ Ri)

R(a,b,c,d) F= {a->b, b ->c, c->d, d->a}

R1(a,b) **R2**(b,c) **R3**(c,d) d=>a ?

Res= d

i=1

Res= d ∪ ((d ∩ {a,b})$^+$ ∩ {a,b}) = d

Paso i=2

Res= d ∪ ((d ∩ {b,c})$^+$ ∩ {b,c}) = d

i=3

Res= d ∪ ((d ∩ {c,d})$^+$ ∩ {c,d})
Res= d ∪ ((d)$^+$ ∩ {c,d})
Res= d ∪ ({a,b,c,d} ∩ {c,d})
Res= d ∪ {c,d} = {c,d}

Se itera nuevamente.

i=1

Res= {c,d} ∪ (({c,d} ∩ {a,b})$^+$ ∩ {a,b})
= {c,d}

i=2

Res= {c,d} ∪ (({c,d} ∩ {b,c})$^+$ ∩ {b,c})
Res= {c,d} ∪ ((c)$^+$ ∩ {b,c})
Res= {c,d} ∪ ({a,b,c,d} ∩ {b,c})
Res= {c,d} ∪ ({b,c}) = {c,d,b}

i=3

Res= {c,d,b} ∪ (({c,d,b} ∩ {c,d})$^+$ ∩ {c,d})
= {c,d,b}

Se itera nuevamente.

i=1

Res= {c,d,b} ∪ (({c,d,b} ∩ {a,b})$^+$ ∩ {a,b})
= {c,d,b,a}

Se logra incorporar al atributo “a”, a partir del atributo “d”. Entonces no se pierde la dependencia funcional.

Teoría de diseño de BBDD relacionales

¿ Cómo usar:

-la descomposición,
-las dependencias funcionales y
-las claves candidatas

para un buen diseño de una relación?

Teoría de diseño para bases de datos relaciones

Formas normales
BCNF

Teoría de diseño de BBDD relacionales

- Forma normal
  - Propiedad sobre la relación.
    - BCNF (Forma normal de Boyce Codd)
      - Provee un mecanismo para asegurar que:
        - las anomalías dejan de estar en un particionamiento (sólo puede quedar redundancia),
        - que no se pierda información y,
        - en algunos casos, asegura que no se pierdan dependencias funcionales

Teoría de diseño de BBDD relacionales

- **BCNF** (Forma normal de Boyce Codd)

  Un esquema de relación está en BCNF si, siempre que una dependencia funcional de la forma X->A es válida en R, entonces se cumple que:

  - X es superclave de R

  o bien

  - X->A es una dependencia funcional trivial

Teoría de diseño de BBDD relacionales

FIESTAS (#salon, dirección, capacidad, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, mesa_invitado, servicio_contratado, dir_contratante, dni_invitado)

Clave candidata

(#salon, fecha_fiesta, dni_invitado, nom_contratante, servicio_contratado)

Dependencias Funcionales

1. #salon → dirección, capacidad
2. nom_contratante → dir_contratante
3. #salon, fecha_fiesta, dni_invitado → mesa_invitado
4. #salon, fecha_fiesta → cant_invitados, cant_mesas
5. dni_invitado-> nombre_invitado

FIESTAS (#salon, dirección, capacidad, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, mesa_invitado, servicio_contratado, dir_contratante, dni_invitado)

Clave candidata

(#salon, fecha_fiesta, dni_invitado, nom_contratante, servicio_contratado)

Dependencias Funcionales

1. #salon → dirección, capacidad
2. nom_contratante → dir_contratante
3. #salon, fecha_fiesta, dni_invitado → mesa_invitado
4. #salon, fecha_fiesta → cant_invitados, cant_mesas
5. dni_invitado-> nombre_invitado

Fiestas cumple con la definición de BCNF?

Para toda dependencia funcional se cumple que:

X es superclave de R

o bien

X->A es una dependencia funcional trivial

Teoría de diseño de BBDD relacionales

FIESTAS (#salon, dirección, capacidad, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, mesa_invitado, servicio_contratado, dir_contratante, dni_invitado)

Clave candidata: (#salon, fecha_fiesta, dni_invitado, nom_contratante, servicio_contratado)

Cómo el esquema FIESTAS no cumple con la definición de BCNF, ya que al menos encontramos a la df1 donde {#salon} no es superclave del esquema Fiestas y sabemos que se puede particionar para eliminar anomalías, procedemos a particionar contemplando la df1

1. #salon → dirección, capacidad

Teoría de diseño de BBDD relacionales

FIESTAS (#salon, dirección, capacidad, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, mesa_invitado, servicio_contratado, dir_contratante, dni_invitado)

Clave candidata: (#salon, fecha_fiesta, dni_invitado, nom_contratante, servicio_contratado)

1. #salon → dirección, capacidad

En base al análisis anterior, dividimos FIESTAS:

F1(#salon, direccion, capacidad)

F2 = Fiestas – {direccion, capacidad}

F2(#salon, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, mesa_invitado, servicio_contratado, dir_contratante, dni_invitado)

FIESTAS (#salon, dirección, capacidad, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, mesa_invitado, servicio_contratado, dir_contratante, dni_invitado)

Clave candidata: (#salon, fecha_fiesta, dni_invitado, nom_contratante, servicio_contratado)

1. #salon → dirección, capacidad

En base al análisis anterior, dividimos FIESTAS:

F1(#salon, direccion, capacidad)

F2(#salon, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, mesa_invitado, servicio_contratado, dir_contratante, dni_invitado)

Con el particionamiento propuesto:

¿Se perdió información?
¿Se perdieron dependencias funcionales?

Clave candidata: (#salon, fecha_fiesta, dni_invitado, nom_contratante, servicio_contratado)

F1(#salon, direccion, capacidad)

F2(#salon, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, mesa_invitado, servicio_contratado, dir_contratante, dni_invitado)

Con el particionamiento propuesto:

¿Se perdió información?

F1 ∩ F2 es clave en el esquema **F1** **o**
F1 ∩ F2 es clave en el esquema **F2**

Entonces, no se perdió información.

Clave candidata: (#salon, fecha_fiesta, dni_invitado, nom_contratante, servicio_contratado)

F1(#salon, direccion, capacidad)

F2(#salon, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, mesa_invitado, servicio_contratado, dir_contratante, dni_invitado)

¿Se perdieron dependencias funcionales?

1. #salon → dirección, capacidad
2. nom_contratante → dir_contratante
3. #salon, fecha_fiesta, dni_invitado → mesa_invitado
4. #salon, fecha_fiesta → cant_invitados, cant_mesas
5. dni_invitado-> nombre_invitado

En F1, vale df1
En F2 valen las dfs 2 a 5

Entonces, no se perdieron dependencias funcionales.

Clave candidata: (#salon, fecha_fiesta, dni_invitado, nom_contratante, servicio_contratado)

F1(#salon, direccion, capacidad)

F2(#salon, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, mesa_invitado, servicio_contratado, dir_contratante, dni_invitado)

- En F1, vale df1. Donde {#salon} es superclave del esquema F1. F1 cumple BCNF.

- En F2 valen las dfs 2 a 5. Donde, al menos {nom_contratante} no es superclave en F2. F2 no está en BCNF.

1. #salon → dirección, capacidad
2. nom_contratante → dir_contratante
3. #salon, fecha_fiesta, dni_invitado → mesa_invitado
4. #salon, fecha_fiesta → cant_invitados, cant_mesas
5. dni_invitado-> nombre_invitado

Teoría de diseño de BBDD relacionales

F2(#salon, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, mesa_invitado, servicio_contratado, dir_contratante, dni_invitado)

2. nom_contratante → dir_contratante
3. #salon, fecha_fiesta, dni_invitado → mesa_invitado
4. #salon, fecha_fiesta → cant_invitados, cant_mesas
5. dni_invitado-> nombre_invitado

Dvidimos contemplando la df2, por el análisis que realizamos previamente

F3 (nom_contratante, dir_contratante)

F4 (#salon, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, mesa_invitado, servicio_contratado, dni_invitado)

F2(#salon, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, mesa_invitado, servicio_contratado, dir_contratante, dni_invitado)

2. nom_contratante → dir_contratante
3. #salon, fecha_fiesta, dni_invitado → mesa_invitado
4. #salon, fecha_fiesta → cant_invitados, cant_mesas
5. dni_invitado-> nombre_invitado

Dvidimos contemplando la df2, por el análisis que realizamos previamente

F3 (nom_contratante, dir_contratante)

F4 (#salon, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, mesa_invitado, servicio_contratado, dni_invitado)

Con el particionamiento propuesto:

¿Se perdió información?

F3 ∩ F4 es clave en el esquema **F3 {nom_contratante}**

¿Se perdieron dependencias funcionales?

En F3, vale df2
En F4 valen las dfs 3 a 5

F2(#salon, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, mesa_invitado, servicio_contratado, dir_contratante, dni_invitado)

2. nom_contratante → dir_contratante
3. #salon, fecha_fiesta, dni_invitado → mesa_invitado
4. #salon, fecha_fiesta → cant_invitados, cant_mesas
5. dni_invitado-> nombre_invitado

Dvidimos contemplando la df2, por el análisis que realizamos previamente

F3 (nom_contratante, dir_contratante)

F4 (#salon, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, mesa_invitado, servicio_contratado, dni_invitado)

- En F3, vale df2. Donde {nom_contratante} es superclave del esquema F3. F3 cumple BCNF.

- En F4 valen las dfs 3 a 5. Donde, al menos {#salon, fecha_fiesta, dni_invitado } no es superclave en F4. F4 no está en BCNF.

3. #salon, fecha_fiesta, dni_invitado → mesa_invitado
4. #salon, fecha_fiesta → cant_invitados, cant_mesas
5. dni_invitado-> nombre_invitado

Teoría de diseño de BBDD relacionales

F4(#salon, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, mesa_invitado, servicio_contratado, dni_invitado)

3. #salon, fecha_fiesta, dni_invitado → mesa_invitado
4. #salon, fecha_fiesta → cant_invitados, cant_mesas
5. dni_invitado-> nombre_invitado

En base al análisis anterior, dividimos F4 contemplando la df3:

F5(#salon, fecha_fiesta,dni_invitado, mesa_invitado)

F6(#salon, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, servicio_contratado, dni_invitado)

F4(#salon, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, mesa_invitado, servicio_contratado, dni_invitado)

3. #salon, fecha_fiesta, dni_invitado → mesa_invitado
4. #salon, fecha_fiesta → cant_invitados, cant_mesas
5. dni_invitado-> nombre_invitado

En base al análisis anterior, dividimos F4 contemplando la df3:

F5(#salon, fecha_fiesta,dni_invitado, mesa_invitado)

F6(#salon, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, servicio_contratado, dni_invitado)

Con el particionamiento propuesto:

¿Se perdió información?

F5 ∩ F6 es clave en el esquema **F5**
{#salon, fecha_fiesta,dni_invitado}

¿Se perdieron dependencias funcionales?

En F5, vale df3
En F6 valen las dfs 4 y 5

F4(#salon, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, mesa_invitado, servicio_contratado, dni_invitado)

3. #salon, fecha_fiesta, dni_invitado → mesa_invitado
4. #salon, fecha_fiesta → cant_invitados, cant_mesas
5. dni_invitado-> nombre_invitado

En base al análisis anterior, dividimos F4 contemplando la df3:

F5(#salon, fecha_fiesta,dni_invitado, mesa_invitado)

F6(#salon, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, servicio_contratado, dni_invitado)

- En F5, vale df3. Donde {#salon, fecha_fiesta,dni_invitado} es superclave del esquema F5. F5 cumple BCNF.

- En F6 valen las dfs 4 y 5. Donde, al menos {#salon, fecha_fiesta} no es superclave en F6. F6 no está en BCNF.

4. #salon, fecha_fiesta → cant_invitados, cant_mesas
5. dni_invitado-> nombre_invitado

Teoría de diseño de BBDD relacionales

F6(#salon, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, servicio_contratado, dni_invitado)

4. #salon, fecha_fiesta → cant_invitados, cant_mesas
5. dni_invitado-> nombre_invitado

Dividimos F6 empleando la df4, a raiz del análisis previo:

F7(#salon, fecha_fiesta, cant_invitados, cant_mesas)

F8(#salon, fecha_fiesta, nom_contratante, nombre_invitado, servicio_contratado, dni_invitado)

Teoría de diseño de BBDD relacionales

F6(#salon, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, servicio_contratado, dni_invitado)

F7(#salon, fecha_fiesta, cant_invitados, cant_mesas)

F8(#salon, fecha_fiesta, nom_contratante, nombre_invitado, servicio_contratado, dni_invitado)

Con el particionamiento propuesto:

¿Se perdió información?

F7 ∩ F8 es clave en el esquema **F7**
{#salon, fecha_fiesta}

¿Se perdieron dependencias funcionales?

En F7, vale df4
En F8 vale la df 5

Teoría de diseño de BBDD relacionales

F6(#salon, fecha_fiesta, nom_contratante, cant_invitados, nombre_invitado, cant_mesas, servicio_contratado, dni_invitado)

F7(#salon, fecha_fiesta, cant_invitados, cant_mesas)

F8(#salon, fecha_fiesta, nom_contratante, nombre_invitado, servicio_contratado, dni_invitado)

- En F7, vale df4. Donde {#salon, fecha_fiesta} es superclave del esquema F7. F7 cumple BCNF.

- En F8 vale la df 5. Donde, {dni_invitado} no es superclave en F8. F8 no está en BCNF.

5. dni_invitado-> nombre_invitado

F8(#salon, fecha_fiesta, nom_contratante, nombre_invitado, servicio_contratado, dni_invitado)

5. dni_invitado-> nombre_invitado

En base al análisis anterior, dividimos el esquema F8 contemplando la df5:

F9(dni_invitado, nombre_invitado)

F10(#salon, fecha_fiesta, nom_contratante, servicio_contratado, dni_invitado)

F8(#salon, fecha_fiesta, nom_contratante, nombre_invitado, servicio_contratado, dni_invitado)

5. dni_invitado-> nombre_invitado

En base al análisis anterior, dividimos el esquema F8 contemplando la df5:

F9(dni_invitado, nombre_invitado)

F10(#salon, fecha_fiesta, nom_contratante, servicio_contratado, dni_invitado)

Con el particionamiento propuesto:

¿Se perdió información?

F9 ∩ F10 es clave en el esquema **F9** **{dni_invitado}**

¿Se perdieron dependencias funcionales?

En F9, vale df4
En F10 vale la df 5

F8(#salon, fecha_fiesta, nom_contratante, nombre_invitado, servicio_contratado, dni_invitado)

5. dni_invitado-> nombre_invitado

En base al análisis anterior, dividimos el esquema F8 contemplando la df5:

F9(dni_invitado, nombre_invitado)

F10(#salon, fecha_fiesta, nom_contratante, servicio_contratado, dni_invitado)

- En F9, vale df5. Donde {dni_invitado} es superclave del esquema F9. F9 cumple BCNF.

- En F10, las únicas dependencias funcionales son las triviales F10 cumple BCNF.

Teoría de diseño de BBDD relacionales

Las particiones del FIESTAS, que quedaron en BCNF, son:

F1(#salon, direccion, capacidad)
F3 (nom_contratante, dir_contratante)
F5(#salon, fecha_fiesta,dni_invitado, mesa_invitado)
F7(#salon, fecha_fiesta, cant_invitados, cant_mesas)
F9(dni_invitado, nombre_invitado)
F10(#salon, fecha_fiesta, nom_contratante, servicio_contratado, dni_invitado)

Teoría de diseño de BBDD relacionales

- **Cómo llevar un esquema R a BCNF**

De manera esquemática y simplificada, una vez halladas las dependencias funcionales y las claves candidatas

1-analizar si en el esquema R existe alguna dependencia funcional que no cumple con la definición de BCNF

1.1) si existe tal dependencia funcional, particionar el esquema en dos nuevos esquemas R_i, R_{i+1}, contemplando la dependencia funcional en cuestión. Analizar las 2 particiones generadas

1.1.1) Se pierde información?

1.1.1.1: NO, entonces sigo a 1.1.2

1.1.1.2: SI. La partición es errónea. Reanalizar

1.1.2) Se pierden Dependencias funcionales?

1.1.2.1 NO, entonces sigo a 1.1.3

1.1.2.2 Si. Entonces no es posible llevar a BCNF. Cambia la forma normal analizada.

1.1.3) Determinar en que forma normal esta R_i, $R_{i+1,}$ si no están en BCNF, reiniciar desde 1, sino pasar a 1.2

1.2) Si no existe, el esquema está en BCNF

Teoría de diseño de BBDD relacionales

- Vimos que:
  - BCNF (Forma normal de Boyce Codd)
    - Provee un mecanismo para asegurar que las anomalías dejan de estar en un particionamiento (puede quedar redundancia) y, **en algunos casos,** asegura que no se pierdan dependencias funcionales

Teoría de diseño de BBDD relacionales

¿ Qué pasa en los casos en los que, al particionar para llevar a BCNF, se pierden dependencias funcionales?

Teoría de diseño para bases de datos relaciones

Formas normales
3FN

Teoría de diseño de BBDD relacionales

- Ejemplo

  LIBROS (titulo, teatro, ciudad)
  * Un teatro se encuentra en una ciudad,
  * para cada ciudad en la que se presenta un título de libro, se conoce el teatro.

  Valen las siguientes dependencias funcionales

  df1) teatro-> ciudad
  df2) titulo, ciudad->teatro

  Claves candidatas:

  cc1: {titulo, ciudad}
  cc2: {teatro, titulo}

- Ejemplo

LIBROS (titulo, teatro, ciudad)

Valen las siguientes dependencias funcionales

df1) teatro-> ciudad
df2) titulo, ciudad->teatro

Claves candidatas:

cc1: {titulo, ciudad}
cc2: {teatro, titulo}

LIBROS cumple con la definición de BCNF?

Para toda dependencia funcional se cumple que:

X es superclave de R
o bien
X->A es una dependencia funcional trivial

- Ejemplo

LIBROS (titulo, teatro, ciudad)

Valen las siguientes dependencias funcionales

df1) teatro-> ciudad
df2) titulo, ciudad->teatro

Claves candidatas:

cc1: {titulo, ciudad}
cc2: {teatro, titulo}

LIBROS no cumple con la definición de BCNF

Existe la df1, tal que teatro no es superclave del esquema.

LIBROS (titulo, teatro, ciudad)

Valen las siguientes dependencias funcionales

df1) teatro-> ciudad
df2) titulo, ciudad->teatro

Claves candidatas:

cc1: {titulo, ciudad}
cc2: {teatro, titulo}

Divido el esquema LIBROS contemplando la df1 ya que {teatro} no es superclave de dicho esquema.

L1(<u>teatro</u>, ciudad)

L2(<u>teatro, titulo</u>)

LIBROS (titulo, teatro, ciudad)

Valen las siguientes dependencias funcionales

df1) teatro-> ciudad
df2) titulo, ciudad->teatro

Claves candidatas:

cc1: {titulo, ciudad}
cc2: {teatro, titulo}

Divido el esquema LIBROS contemplando la df1 ya que {teatro} no es superclave de dicho esquema.

L1(teatro, ciudad)
L2(teatro, titulo)

Con el particionamiento propuesto:

¿Se perdió información?

L1 ∩ **L2** es clave en el esquema **L1** {teatro}

¿Se perdieron dependencias funcionales?

LIBROS (titulo, teatro, ciudad)

Valen las siguientes dependencias funcionales

df1) teatro-> ciudad
df2) titulo, ciudad->teatro

Claves candidatas:

cc1: {titulo, ciudad}
cc2: {teatro, titulo}

Divido el esquema LIBROS contemplando la df1 ya que {teatro} no es superclave de dicho esquema.

L1(teatro, ciudad)
L2(teatro, titulo)

Con el particionamiento propuesto:

¿Se perdieron dependencias funcionales?

En L1, vale la df1

¿Que pasó con la df2 (titulo, ciudad->teatro)?

Con el particionamiento propuesto:

¿Se perdieron dependencias funcionales?

¿Que pasó con la df2 (titulo, ciudad->teatro)?

Apliquemos el algoritmo para determinar pérdida de dependencias funcionales en LIBROS

Res = ?
Mientras Res cambia
  Para i= 1 to cant_de_ particiones_realizadas
    Res = Res ∪ ((Res ∩ Ri)$^+$ ∩ Ri)

LIBROS (titulo, teatro, ciudad)
F= {teatro-> ciudad
titulo, ciudad->teatro}

L1(teatro, ciudad)
L2(teatro, titulo)

titulo, ciudad->teatro ?

Res= (titulo, ciudad)
i=1
Res= (titulo, ciudad) ∪ (((titulo, ciudad) ∩ {teatro,ciudad})$^+$ ∩ {teatro,ciudad}) =

(titulo, ciudad) ∪ ({ciudad})$^+$ ∩ {teatro,ciudad}) =

(titulo, ciudad) ∪ ({∅} ∩ {teatro,ciudad})= (titulo, ciudad) ∪ {∅} = (titulo, ciudad)

i=2
Res= (titulo, ciudad) ∪ (((titulo, ciudad) ∩ {teatro,titulo})$^+$ ∩ {teatro,titulo}) =

(titulo, ciudad) ∪ ({titulo})$^+$ ∩ {teatro,titulo}) =

(titulo, ciudad) ∪ ({∅} ∩ {teatro,titulo})= (titulo, ciudad) ∪ {∅} = (titulo, ciudad)

Terminamos de recorrer todas las particiones de LIBROS y res no cambió

La df: titulo, ciudad->teatro se perdió en el particionamiento!!

Con el particionamiento propuesto:

¿Se perdieron dependencias funcionales?

¿Que pasó con la df2 (titulo, ciudad->teatro)?

La df: titulo, ciudad->teatro se perdió en el particionamiento!!

Cuando no se puede llevar a BCNF porque se pierden dependencias funcionales, entonces, se lleva el esquema a 3FN

Teoría de diseño de BBDD relacionales

- Cuando no se puede llevar a BCNF porque se pierden dependencias funcionales, entonces, se lleva el esquema a **3FN**, lo que asegura que:
  - no se pierde información,
  - no se pierdan dependencias funcionales,
  - pero no siempre se quitan las anomalías.

Teoría de diseño de BBDD relacionales

- **3FN** (Tercera forma normal)
  - Un esquema de relación R está en 3FN si para toda dependencia de la forma X->A, se cumple que:
    - X->A es trivial

    O bien,
    - X es superclave

    O bien
    - A es primo

Atributo primo: atributo que forma parte de alguna clave candidata

Teoría de diseño de BBDD relacionales

- ## Cómo llevar un esquema R a 3FN

De manera esquemática y simplificada, una vez halladas las dependencias funcionales y las claves **candidatas y habiendo detectado que no se puede llevar a BCNF**

  - Se construye una tabla por cada dependencia funcional

  - Si la clave de la tabla original, no está incluida en ninguna de las tablas del punto anterior, se construye una tabla con la clave

LIBROS (titulo, teatro, ciudad)

Valen las siguientes dependencias funcionales

df1) teatro-> ciudad
df2) titulo, ciudad->teatro

Claves candidatas:

cc1: {titulo, ciudad}
cc2: {teatro, titulo}

Como no es posible llevar el esquema a BCNF sin perder dependencias funcionales, entonces, aplico el proceso para dejar el esquema en 3FN.

L1.1 (teatro, ciudad)
L2.1 (teatro, titulo,ciudad)

(en este caso, como la clave quedo en una de las particiones, no se agrega una nueva partición con ella)

Las particiones L1.1 y L1.2 están en 3FN

Teoría de diseño de BBDD relacionales

En síntesis:

Llamamos **normalizar** hasta BCNF o 3FN, al **proceso** que involucra, hasta ahora, los siguientes pasos:

- Encontrar las dependencias funcionales y las claves candidatas
- Llevar a BCNF aplicando el proceso de división sin pérdida de información
  - Comprobar que no se pierden dependencias funcionales en la división
  - Si se pierden dependencias funcionales al dividir, se lleva el esquema correspondiente a 3NF

Teoría de diseño de BBDD relacionales

- Otras formas normales
  - **1FN**

    Los atributos de la relación son simples y atómicos

  - **2FN**:

    Un esquema de relación R está en 2FN si para toda dependencia de la forma X->A, se cumple que:

    A depende de manera total de la clave

Teoría de diseño de BBDD relacionales

- **2FN**:

  Un esquema de relación R está en 2FN si para toda dependencia de la forma X->A, se cumple que: A depende de manera total de la clave

Empleados (#emp, #area, nombre, nombre_area, años)

Donde

- "nombre" es el nombre de un empleado.
- "años" es la cantidad de años que un empleado trabajó en un área determinada.
- "nombre_area" es el nombre de un área de trabajo, puede repetirse para distintos #area.

- Clave candidata: (#empleado, #área)
- Dependencias funcionales: 1) #empleado->nombre
  2) #area->nombre_area
  3) #empleado,#area->años

Teoría de diseño de BBDD relacionales

- **2FN**:

  Un esquema de relación R está en 2FN si para toda dependencia de la forma X->A, se cumple que: A depende de manera total de la clave

Empleados (#emp, #area, nombre, nombre_area, años)

- Clave candidata: (#empleado, #área)
- Dependencias funcionales:

  df1) #empleado->nombre

  df2) #area->nombre_area

  df3) #empleado,#area->años

La relación R no se encuentra en 2FN por existen atributos que no dependen funcionalmente de toda la clave de R.

Por ejemplo, existe la DF1 en donde *Nombre* no depende funcionalmente de toda la clave y *Nombre* no es un atributo primo.

Teoría de diseño de BBDD relacionales

En general:

- Si un esquema está en BCNF, se puede asegurar que además cumple, 3FN, 2FN y 1FN
- Si un esquema está en 3FN, se puede asegurar que además cumple, 2FN y 1FN

Y por ello, no analizaremos 1Fn y 2FN, por el proceso de normalización que se lleva a cabo en la materia (siguiendo al autor: J.D.Ullman).

Teoría de diseño de BBDD relacionales

Alcanza con dejar los esquemas en BCNF o 3FN para quitar la anomalía de redundancia?

Teoría de diseño de BBDD relacionales

Las particiones del FIESTAS, que quedaron en BCNF, son:

F1(#salon, direccion, capacidad)
F3 (nom_contratante, dir_contratante)
F5(#salon, fecha_fiesta,dni_invitado, mesa_invitado)
F7(#salon, fecha_fiesta, cant_invitados, cant_mesas)
F9(dni_invitado, nombre_invitado)
F10(#salon, fecha_fiesta, nom_contratante, servicio_contratado, dni_invitado)

Teoría de diseño de BBDD relacionales

Como es una instancia de F10?

(#salon, fecha_fiesta, nom_contratante, servicio_contratado, dni_invitado

#salón	fecha_fiesta	nom_contratante	serv_contratado	dni_invitado
#1	10/03/12	Pedro Guti	empanadas	11111111
#1	10/03/12	Pedro Guti	empanadas	11222222
#1	10/03/12	Pedro Guti	pizza	11111111
#1	10/03/12	Pedro Guti	pizza	11222222
#1	10/03/12	María Zeta	empanadas	11111111
#1	10/03/12	María Zeta	empanadas	11222222
#1	10/03/12	María Zeta	piza	11111111
#1	10/03/12	María Zeta	piza	11222222
#1	11/03/12	Juan Zeballos	Mesa de quesos	11333333
#1	11/03/12	Juan Zeballos	calentitos	11333333
…	…	…	…	…

Teoría de diseño para bases de datos relaciones

Dependencia Multivaluada

Bibliografía de la clase

- Garcia-Molina, H. (2008). *Database systems: the complete book*. Pearson Education India.
- Ullman, J. D. (1988). Principles of database and knowledge-base systems.