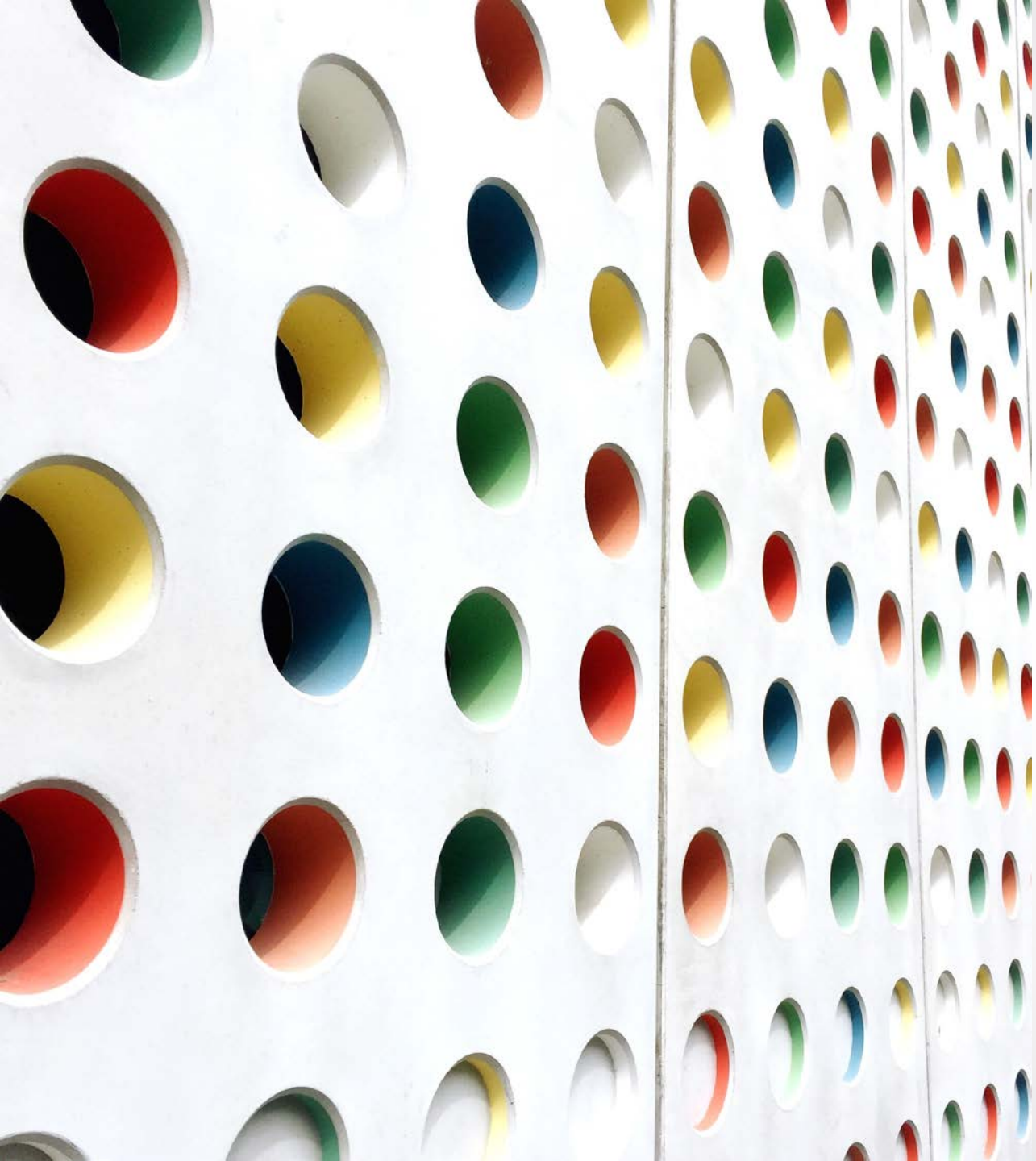

Bases de Datos 1

Alejandra Beatriz Lliteras

Contenidos de la materia

- Modelo de datos
  - Conceptos generales
  - Algunos modelos en particular
    - Modelo de Entidades y Relaciones
    - Modelo relacional
- Transformación entre modelos de datos
- Álgebra Relacional
  - Operaciones y Consultas
  - Optimización de consultas
- **Teoría de diseño de bases de datos relacionales**
  - **Conceptos generales**
  - **Proceso de Normalización**
- SGBD Relacional
- Conceptos generales de bases de datos

De la clase anterior...

- **Claves candidatas**

- Ejemplo
  - EMPRESA(idGerente, nombreGerente, idEmpleado, nombreEmpleado, idEmpleadoMaestranza)
    - Donde:
      - El id de gerente es único
      - El id del empleado es único
      - Cada empledo responde a un único gerente
      - El idEmpleadoMaestranza representa el identificador de cada uno de los empleados de maestranza de la empresa, de los cuales se saben son muchos.

Dependencias funcionales válidas en EMPRESA:

df1) idGerente -> nombreGerente
df2) idEmpleado -> nombreEmpleado, idGerente

Claves Candidatas en EMPRESA

cc1: (idEmpleado, idEmpleadoMaestranza)

Teoría de diseño de bases de datos relaciones

Clausura de un conjunto de ATRIBUTOS

Teoría de diseño de BBDDR

- 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 BBDDR

¿Cómo se puede comprobar que un conjunto de atributos {A1, A2,...,An} permite recuperar al resto de los atributos de la relación?

Teoría de diseño de BBDDR

- 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 BBDDR

- **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 BBDDR

- Ejemplo visto en otra clase
  - 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 BBDDR

- 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 BBDDR

Result:= X

While (hay cambios en result) do

For (cada dependencia funcional Y->Z en F) do

if (Y ⊆ result) then

result := result ∪ 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 BBDDR

- De la misma manera podríamos haber probado con la otra clave candidata

- Recordar que este algoritmo **NO asegura que el conjunto de atributos de partida sea mínimo**

Teoría de diseño de bases de datos relaciones (BBDDR)

Axiomas de Armstrong

Teoría de diseño de BBDDR

¿Cómo deducir nuevas dependencias funcionales a partir de un conjunto dado?

Teoría de diseño de BBDDR

- **Axiomas de Armstrong**
  - Permiten inferir nuevas dependencias funcionales dado un conjunto base que resultó evidente
  - Aplicándolos hallo un conjunto completo y seguro donde todas las dependencias funcionales halladas son correctas
  - Al generar todas las dependencias funcionales algunas son triviales

Teoría de diseño de BBDDR

- **Axiomas de Armstrong**
  - Axiomas Básicos
    - Reflexión
    - Aumento
    - Transitividad
  - Axiomas que se deducen a partir de los básicos
    - Unión
    - Descomposición
    - Pseudotransitividad

Teoría de diseño de BBDDR

Axiomas de Armstrong

- **Reflexión**:

X es un conjunto de atributos

y

Y⊆X entonces X -> Y

Sabemos que a -> a, luego se puede decir que a,b -> a

Demostración:

Si Y⊆X y existen dos tuplas diferentes de R tales que t1[x]=t2[x] por definición de dependencia funcional t1[y]=t2[y]

Teoría de diseño de BBDDR

- **Axiomas de Armstrong**
  - **Aumento**

Si X ->Y ;
Z es un conjunto de atributos,
entonces
Z,X -> Z,Y

Demostración:
Asumamos que X->Y vale pero X,Z->Y,Z no vale
Si X->Y entonces cada vez que
1) t1[x]=t2[x] implica
2) t1[y]=t2[y]
Por otro lado, cada vez que
3) t1[x,z]=t2[x,z] implica
4) t1[y,z] <> t2[y,z]
De 1) y 3)se deduce t1[z]=t2[z]
De 2) y 4) se deduce que t1[y,z]=t2[y,z]

Teoría de diseño de BBDDR

- **Axiomas de Armstrong**
  - **Transitividad**

Si X ->Y ;
Y ->Z,
entonces X -> Z

Demostración:
1) X ->Y
2) Y ->Z
t1[x]=t2[x] implica por 1)
t1[y]=t2[y] implica por 2)
t1[z]=t2[z] entonces
X -> Z

Teoría de diseño de BBDDR

- **Axiomas de Armstrong**
  - **Unión**

Si X ->Y ;
X ->Z,
entonces X -> Y,Z

Demostración:
1) X ->Y
2) X ->Z
Si X ->Y , por aumentación vale que X ->XY
Si X ->Z , por aumentación vale que X,Y ->Y,Z
Luego por transitividad, X ->Y,Z

Teoría de diseño de BBDDR

- **Axiomas de Armstrong**
  - **Descomposición**

Si X ->Y,Z
entonces X -> Y , X -> Z

Demostración:

X ->Y,Z
por reflexividad vale que Y,Z ->Y
Luego, por transitividad X->Y
Por reflexividad también vale que Y,Z ->Z
Luego por transitividad, también vale que X->Z

Teoría de diseño de BBDDR

- **Axiomas de Armstrong**
  - **Pseudotransitividad**

Si X ->Y; Y,Z->W entonces X, Z-> W

Demostración:

X ->Y

por aumento vale que X,Z ->Y,Z

Por otro lado se sabe que Y,Z->W

Luego por transitividad, vale que X,Z->W

Teoría de diseño de BBDDR

- **Hasta ahora vimos, para una relación R**
  - Cómo hallar la o 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)
  - Cómo hallar dependencias funcionales
    - Y su conjunto completo mediante los Axiomas de Armstrong
- **A continuación:**
  - Considerando las dependencias funcionales y las claves candidatas, veremos un **proceso para generar relaciones** que cumplan ciertas condiciones de un buen diseño (quitando anomalías) con el fin de **normalizar** esquemas

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

Teoría de diseño de BBDDR

- **Descomposición o particionamiento de un esquema**
  - Es una forma aceptada de eliminar las anomalías de una relación
  - Consiste en separar los atributos de una relación en dos nuevas relaciones (bajo ciertos criterios)
  - Al particionar, no se debe perder
    - Información
    - Dependencias funcionales

Teoría de diseño de BBDDR

- **Descomposición o particionamiento de un esquema**

  - Dado un esquema **R**

    donde vale una dependencia funcional **X->Y**

    R se descompone/particiona en

    R1(X, Y)

    R2(R-Y)

Teoría de diseño de BBDDR

- **Descomposición**
  - Al descomponer, no se debe perder
    - Información
    - Dependencias funcionales
      - Validación simple
      - Validación formal mediante un algoritmo

Teoría de diseño de BBDDR

- Descomposición
  - **Pérdida de Información**

Si a un esquema **R**,

se lo particiona en dos subesquemas **R1** y **R2**

entonces, se debe cumplir alguna de las siguientes condiciones:

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

o

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

Teoría de diseño de BBDDR

- Descomposición
  - **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
        - Validación simple
      - los atributos de la dependencia funcional original quedaron distribuidos en mas de una partición
        - Validación formal mediante un algoritmo

Teoría de diseño de BBDDR

- Descomposición
  - **Pérdida de dependencias funcionales**
    - los atributos de la dependencia funcional original quedaron todos incluidos en alguna de las particiones generadas
      - **Validación simple**
    - los atributos de la dependencia funcional original quedaron distribuidos en mas de una partición
      - **Validación formal mediante un algoritmo**

Estos conceptos serán retomados y ampliados mas adelante en esta clase y en las venideras.

Teoría de diseño de BBDDR

¿ Cómo usar:

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

para un buen diseño de una relación?

Veremos como normalizar un esquema a partir de estos conceptos.

Teoría de diseño de bases de datos relaciones

Formas normales

Teoría de diseño de BBDDR

- **Forma Normal (FN)**
  - Criterio para determinar grado de vulnerabilidad a inconsistencias y anomalías
  - Existen diferentes formas normales
    - Al lograr aplicar una mayor forma normal se logrará menor vulnerabilidad

Teoría de diseño de BBDDR

- **Formas Normales**
  - Primera Forma Normal (1FN)
  - Segunda Forma Normal (2FN)
  - Tercera Forma Normal (3FN)
  - Forma Normal de Boyce y Codd

Se determinan a partir de las dependencias funcionales

El esquema no debe tener atributos polivalentes o compuestos

Teoría de diseño de BBDDR

- **Forma Normal de Boyce y Codd**
  - Conocida por su acrónimo en inglés de BCNF
  - Particionar llevando un esquema a esta FN, asegura que:
    - las anomalías dejan de estar (sólo puede quedar redundancia),
    - que no se pierda información y,
    - en algunos casos, asegura que no se pierdan dependencias funcionales

Teoría de diseño de BBDDR

- **BCNF** (Forma Normal de Boyce y 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 BBDDR

- Hasta ahora sabemos que:
  - Las formas normales se usan para sacar anomalías
  - A mayor forma normal, menos vulnerabilidad
- Intentaremos llevar, en principio, un esquema a la Forma Normal de Boyce y Codd para normalizarlo

Teoría de diseño de BBDDR

Para normalizar un esquema vamos a valernos de:

- las dependencias funcionales del esquema
- las claves candidatas
- la definición de BCNF
- la descomposición o particionamiento del esquema

Veamos como se analiza en función de estos cuatro conceptos, la Forma Normal de Boyce y Codd en un esquema para normalizarlo.

Iniciemos con el proceso de normalización de un esquema a partir de un ejemplo

Considerando que el esquema ya se encuentra en Primera Forma Normal

Teoría de diseño de BBDDR

FIESTAS (#salon, dirección, capacidad, fecha_fiesta, nombre_invitado, mesa_invitado, servicio_contratado, dni_invitado)

Clave candidata

(#salon, fecha_fiesta, dni_invitado, servicio_contratado)

Dependencias Funcionales

1. #salon → dirección, capacidad
2. #salon, fecha_fiesta, dni_invitado → mesa_invitado
3. dni_invitado-> nombre_invitado

FIESTAS (#salon, dirección, capacidad, fecha_fiesta, nombre_invitado, mesa_invitado, servicio_contratado, dni_invitado)

Clave candidata

cc: (#salon, fecha_fiesta, dni_invitado, servicio_contratado)

Dependencias Funcionales

1. #salon → dirección, capacidad
2. #salon, fecha_fiesta, dni_invitado → mesa_invitado
3. dni_invitado**->** nombre_invitado

FIESTAS cumple con la definición de BCNF?

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 BBDDR

- FIESTAS (#salon, dirección, capacidad, fecha_fiesta, nombre_invitado, mesa_invitado, servicio_contratado, dni_invitado)

cc: (#salon, fecha_fiesta, dni_invitado, servicio_contratado)

Por ejemplo, analizo la dependencia funcional 1:

1. #salon → dirección, capacidad

BCNF
Para toda dependencia funcional se cumple que:

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

{#salon} no es superclave del esquema FIESTAS

FIESTAS no cumple la definición de BCNF

Teoría de diseño de BBDDR

- FIESTAS (#salon, dirección, capacidad, fecha_fiesta, nombre_invitado, mesa_invitado, servicio_contratado, dni_invitado)

cc: (#salon, fecha_fiesta, dni_invitado, servicio_contratado)

Un esquema R donde vale una dependencia funcional X->Y se descompone como
R1(X, Y)
R2(R-Y)

1. #salon → dirección, capacidad

Dado que **FIESTAS NO CUMPLE CON LA DEFINICION DE BCNF**, **descompongo/particiono** FIESTAS considerando la dependencia funcional 1.

F1(#salon, dirección, capacidad)

F2 = Fiestas – {dirección, capacidad}
Es decir, F2 tiene los siguientes atributos:

F2 (#salon, fecha_fiesta, nombre_invitado, mesa_invitado, servicio_contratado, dni_invitado)

Teoría de diseño de BBDDR

- FIESTAS (#salon, dirección, capacidad, fecha_fiesta, nombre_invitado, mesa_invitado, servicio_contratado, dni_invitado)

  cc: (#salon, fecha_fiesta, dni_invitado, servicio_contratado)

  1. #salon → dirección, capacidad

  Dado que FIESTAS NO CUMPLE CON LA DEFINICION DE BCNF,
  descompongo/particiono FIESTAS considerando la df1.

  F1(#salon, dirección, capacidad)
  F2 (#salon, fecha_fiesta, nombre_invitado, mesa_invitado, servicio_contratado, dni_invitado)

  Con el particionamiento propuesto:

  ¿Se perdió información?
  ¿Se perdieron dependencias funcionales?

Teoría de diseño de BBDDR

- FIESTAS (#salon, dirección, capacidad, fecha_fiesta, nombre_invitado, mesa_invitado, servicio_contratado, dni_invitado)

cc: (#salon, fecha_fiesta, dni_invitado, servicio_contratado)

1. #salon → dirección, capacidad

Dado que FIESTAS NO CUMPLE CON LA DEFINICION DE BCNF,
descompongo/particiono FIESTAS considerando la df1.

F1(#salon, dirección, capacidad)
F2 (#salon, fecha_fiesta, nombre_invitado, mesa_invitado, servicio_contratado, dni_invitado)

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

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 {#salon}

Entonces, no se perdió información.

Teoría de diseño de BBDDR

- FIESTAS (#salon, dirección, capacidad, fecha_fiesta, nombre_invitado, mesa_invitado, servicio_contratado, dni_invitado)

  cc: (#salon, fecha_fiesta, dni_invitado, servicio_contratado)

  1. #salon → dirección, capacidad

  Dado que FIESTAS NO CUMPLE CON LA DEFINICION DE BCNF, **descompongo/particiono** FIESTAS considerando la df1.

  F1(#salon, dirección, capacidad)
  F2(#salon, fecha_fiesta, nombre_invitado, mesa_invitado, servicio_contratado, dni_invitado)

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

Con el particionamiento propuesto:

¿Se perdió información? NO

¿Se perdieron dependencias funcionales?

cc: (#salon, fecha_fiesta, dni_invitado, servicio_contratado)

F1(#salon, dirección, capacidad)
F2(#salon, fecha_fiesta, nombre_invitado, mesa_invitado,
servicio_contratado, dni_invitado)

¿Se perdieron dependencias funcionales?

1. #salon → dirección, capacidad
2. #salon, fecha_fiesta, dni_invitado → mesa_invitado
3. dni_invitado**->** nombre_invitado

En F1, vale df1
En F2 valen las dfs 2 y 3

Fue posible validarlo de la manera simple, sin necesitar de un algoritmo particular

Entonces, no se perdieron dependencias funcionales.

Ya particioné y validé que se cumplan las dos condiciones necesarias para que sea un particionamiento válido.

Resta validar, si se dejaron ambas particiones en la Forma Normal de Boyce y Codd ya que eso me aseguraría que saqué ciertas anomalías

cc: (#salon, fecha_fiesta, dni_invitado, servicio_contratado)

F1(#salon, dirección, capacidad)
F2(#salon, fecha_fiesta, nombre_invitado, mesa_invitado,
servicio_contratado, dni_invitado)

BCNF
Para toda dependencia funcional se cumple que:

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

¿F1 y F2 cumplen con la definición de BCNF?

En **F1** vale sólo la df1, en F1 se cumple que {#salon} es superclave, entonces **F1 cumple la definición de BCNF**

En **F2** vale valen las df2 y df3. En particular, {dni_invitado} no es superclave en F2. Entonces, **F2 no cumple con la definición de BCNF**, descompongo/particiono F2, considerando la dependencia funcional 3

1. #salon → dirección, capacidad
2. #salon, fecha_fiesta, dni_invitado → mesa_invitado
3. dni_invitado**->** nombre_invitado

Debo seguir particionando, ahora con foco en F2

cc: (#salon, fecha_fiesta, dni_invitado, servicio_contratado)

F1(#salon, dirección, capacidad)
F2(#salon, fecha_fiesta, nombre_invitado, mesa_invitado,
servicio_contratado, dni_invitado)

En **F2** vale valen las df2 y df3. En particular, {dni_invitado} no es superclave en F2. Entonces, **F2 no cumple con la definición de BCNF**, descompongo/particiono F2, considerando la dependencia funcional 3

F3 (dni_invitado, nombre_invitado)
F4(#salon, fecha_fiesta, mesa_invitado, servicio_contratado,
dni_invitado)

Con el particionamiento propuesto:

¿Se perdió información?
¿Se perdieron dependencias funcionales?

1. #salon → dirección, capacidad
2. #salon, fecha_fiesta, dni_invitado → mesa_invitado
3. dni_invitado**->** nombre_invitado

cc: (#salon, fecha_fiesta, dni_invitado, servicio_contratado)

F1(#salon, dirección, capacidad)
F2(#salon, fecha_fiesta, nombre_invitado, mesa_invitado,
servicio_contratado, dni_invitado)

En **F2** vale valen las df2 y df3. En particular, {dni_invitado} no es superclave en F2. Entonces, **F2 no cumple con la definición de BCNF**, descompongo/particiono F2, considerando la dependencia funcional 3

F3 (dni_invitado, nombre_invitado)
F4(#salon, fecha_fiesta, mesa_invitado, servicio_contratado,
dni_invitado)

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

Con el particionamiento propuesto:

¿Se perdió información?
¿Se perdieron dependencias funcionales?

1. #salon → dirección, capacidad
2. #salon, fecha_fiesta, dni_invitado → mesa_invitado
3. dni_invitado**->** nombre_invitado

cc: (#salon, fecha_fiesta, dni_invitado, servicio_contratado)

F1(#salon, dirección, capacidad)
F2(#salon, fecha_fiesta, nombre_invitado, mesa_invitado,
servicio_contratado, dni_invitado)

En **F2** vale valen las df2 y df3. En particular, {dni_invitado} no es superclave en F2. Entonces, **F2 no cumple con la definición de BCNF**, descompongo/particiono F2, considerando la dependencia funcional 3

F3 (dni_invitado, nombre_invitado)
F4(#salon, fecha_fiesta, mesa_invitado, servicio_contratado,
dni_invitado)

¿Con el particionamiento propuesto se perdió información?

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

Entonces, no se perdió información.

cc: (#salon, fecha_fiesta, dni_invitado, servicio_contratado)

F1(#salon, dirección, capacidad)
F2(#salon, fecha_fiesta, nombre_invitado, mesa_invitado,
servicio_contratado, dni_invitado)

En **F2** vale valen las df2 y df3. En particular, {dni_invitado} no es superclave en F2. Entonces, **F2 no cumple con la definición de BCNF**, descompongo/particiono F2, considerando la dependencia funcional 3

F3 (dni_invitado, nombre_invitado)
F4(#salon, fecha_fiesta, mesa_invitado, servicio_contratado,
dni_invitado)

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

Con el particionamiento propuesto:

¿Se perdió información?
¿Se perdieron dependencias funcionales?

1. #salon → dirección, capacidad
2. #salon, fecha_fiesta, dni_invitado → mesa_invitado
3. dni_invitado**->** nombre_invitado

cc: (#salon, fecha_fiesta, dni_invitado, servicio_contratado)

F1(#salon, dirección, capacidad)
F2(#salon, fecha_fiesta, nombre_invitado, mesa_invitado,
servicio_contratado, dni_invitado)

En **F2** vale valen las df2 y df3. En particular, {dni_invitado} no es superclave en F2. Entonces, **F2 no cumple con la definición de BCNF**, descompongo/particiono F2, considerando la dependencia funcional 3

F3 (dni_invitado, nombre_invitado)
F4(#salon, fecha_fiesta, mesa_invitado, servicio_contratado,
dni_invitado)

¿Con el particionamiento propuesto se perdieron dependencias funcionales?

En **F3** vale la dependencia funcional 3. {dni_invitado} es superclave en F3. Entonces**, F3 cumple con la definición de BCNF.**

En **F4** vale la dependencia funcional 2. {#salon, fecha_fiesta, dni_invitado } NO es superclave en F4. Entonces**, F4 NO cumple con la definición de BCNF.**

Debo seguir particionando, ahora con foco en F4

cc: (#salon, fecha_fiesta, dni_invitado, servicio_contratado)

F3 (dni_invitado, nombre_invitado)
F4(#salon, fecha_fiesta, mesa_invitado, servicio_contratado, dni_invitado)

Dado que **F4 NO cumple con la definición de BCNF,** los particiono considerando la dependencia funcional 2

F5 (#salon, fecha_fiesta, dni_invitado, mesa_invitado)
F6(#salon, fecha_fiesta, servicio_contratado, dni_invitado)

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

Con el particionamiento propuesto:

¿Se perdió información?
¿Se perdieron dependencias funcionales?

1. #salon → dirección, capacidad
2. #salon, fecha_fiesta, dni_invitado → mesa_invitado
3. dni_invitado**->** nombre_invitado

cc: (#salon, fecha_fiesta, dni_invitado, servicio_contratado)

F3 (dni_invitado, nombre_invitado)
F4(#salon, fecha_fiesta, mesa_invitado, servicio_contratado, dni_invitado)

Dado que **F4 NO cumple con la definición de BCNF,** los particiono considerando la dependencia funcional 2

F5 (#salon, fecha_fiesta, dni_invitado, mesa_invitado)
F6(#salon, fecha_fiesta, servicio_contratado, dni_invitado)

¿Con el particionamiento propuesto se perdió información?

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

Entonces, no se perdió información.

1. #salon → dirección, capacidad
2. #salon, fecha_fiesta, dni_invitado → mesa_invitado
3. dni_invitado**->** nombre_invitado

cc: (#salon, fecha_fiesta, dni_invitado, servicio_contratado)

F3 (dni_invitado, nombre_invitado)
F4(#salon, fecha_fiesta, mesa_invitado, servicio_contratado, dni_invitado)

Dado que **F4 NO cumple con la definición de BCNF,** los particiono considerando la dependencia funcional 2

F5 (#salon, fecha_fiesta, dni_invitado, mesa_invitado)
F6(#salon, fecha_fiesta, servicio_contratado, dni_invitado)

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

Con el particionamiento propuesto:

¿Se perdió información?
¿Se perdieron dependencias funcionales?

1. #salon → dirección, capacidad
2. #salon, fecha_fiesta, dni_invitado → mesa_invitado
3. dni_invitado**->** nombre_invitado

cc: (#salon, fecha_fiesta, dni_invitado, servicio_contratado)

F3 (dni_invitado, nombre_invitado)
F4(#salon, fecha_fiesta, mesa_invitado, servicio_contratado, dni_invitado)

Dado que **F4 NO cumple con la definición de BCNF,** los particiono considerando la dependencia funcional 2

F5 (#salon, fecha_fiesta, dni_invitado, mesa_invitado)
F6(#salon, fecha_fiesta, servicio_contratado, dni_invitado)

¿Con el particionamiento propuesto se perdieron dependencias funcionales?

En **F5** vale la dependencia funcional 2. {#salon, fecha_fiesta, dni_invitado} es superclave en F5. Entonces**, F5 cumple con la definición de BCNF.**

En **F6** todos los atributos forman parte de la clave, Cualquier dependencia funcional que detecte va a ser trivial. **F6 cumple con la definición de BCNF.**

BCNF
Para toda dependencia funcional se cumple que:

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

1. #salon → dirección, capacidad
2. #salon, fecha_fiesta, dni_invitado → mesa_invitado
3. dni_invitado**->** nombre_invitado

Teoría de diseño de BBDDR

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

F1(#salon, direccion, capacidad)
F3(dni_invitado, nombre_invitado)
F5(#salon, fecha_fiesta,dni_invitado, mesa_invitado)
F6(#salon, fecha_fiesta,dni_invitado, servicio_contratado)

El esquema FIESTAS queda así normalizado hasta la Forma Normal de Boyce y Codd

Retomaremos este ejemplo cuando analicemos la anomalía de redundancia.

Teoría de diseño de BBDDR

- **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 lleva al esquema a no cumplir 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 BBDDR

- ***Hemos visto***
  - Como llevar un esquema a BCNF cuando no se pierde información ni dependencias funcionales
    - En particular, vimos el caso en el que los atributos de todas las dependencias funcionales quedaron todos incluidos en alguna de las particiones generadas
      - **Validación simple**

- ***Resta ver:***
  - Qué sucede cuando los atributos de alguna dependencia funcional quedaron distribuidos en mas de una partición
    - **Validación formal mediante un algoritmo**
  - Qué hacer en el caso de que alguna dependencia funcional se pierda y **no se pueda** avanzar con el proceso visto para **llevar el esquema a BCNF**

Actividades para el encuentro participativo

Teoría de diseño de BBDDR

- Ejercicio
  - Hallar dependencias funcionales
  - Hallar la o las claves candidatas
  - Opcional: iniciar con el proceso de normalización

ATENCIONES(codHospital, nombreHospital, dniPaciente, legajoPaciente, dniMedico)

Donde:

- Un paciente tiene asignado para cada hospital un número de legajo
- Un legajo en un hospital se asigna a una única persona
- En un hospital trabajan muchos médicos y un médico puede trabajar en diversos hospitales
- Un médico atiende a muchos pacientes
- Cada hospital posee un nombre y el mismo nombre se puede repetir para diferentes hospitales
- Un paciente se atiende en muchos hospitales y de cada hospital que se atiende se registran los médicos que lo atienden

Bibliografía de los temas abordados en esta clase

- Date, C. J. (2019). *Database design and relational theory: normal forms and all that jazz*. Apress.
- Garcia-Molina, H. (2008). *Database systems: the complete book*. Pearson Education India.
- Ullman, J. D. (1988). Principles of database and knowledge-base systems.
- Albarak, M., Bahsoon, R., Ozkaya, I., & Nord, R. L. (2020). Managing Technical Debt in Database Normalization. *IEEE Transactions on Software Engineering*.
- Jadhav, R., Dhabe, P., Gandewar, S., Mirani, P., & Chugwani, R. (2020). A New Data Structure for Representation of Relational Databases for Application in the Normalization Process. In *Machine Learning and Information Processing* (pp. 305-316). Springer, Singapore.
- Ghawi, R. (2019, May). Interactive Decomposition of Relational Database Schemes Using Recommendations. In *International Conference: Beyond Databases, Architectures and Structures* (pp. 97-108). Springer, Cham.
- Stefanidis, C., & Koloniari, G. (2016, November). An interactive tool for teaching and learning database normalization. In *Proceedings of the 20th Pan-Hellenic Conference on Informatics* (pp. 1-4).
- Armstrong, W. W. (1974, August). Dependency Structures of Data Base Relationships. In *IFIP congress* (Vol. 74, pp. 580-583).
- Sug, H. (2020, June). Efficient checking of functional dependencies for relations. In *Journal of Physics: Conference Series* (Vol. 1564, No. 1, p. 012011). IOP Publishing.
- Fernandez, M., & Varga, J. (2020, September). Finding Candidate Keys and 3NF via Strategic Port Graph Rewriting. In *22nd International Symposium on Principles and Practice of Declarative Programming* (pp. 1-14).
- Knowledge Base of Relational and NoSQL Database Management Systems https://db-engines.com/en/ranking_trend

IMPORTANTE: los slides usados en las clases teóricas de esta materia, no son material de estudio por sí solos.