NORMALIZACIÓN

Índice

- Características de un buen diseño relacional
- Descomposición utilizando Dependencias Funcionales
- Teoría de Dependencias Funcionales
- Más Formas Normales
- Proceso de diseño de bases de datos
- Algoritmos para dependencias funcionales

¿Combinar esquemas?

- Supongamos que combinamos *docente* y *departamento* en *docente_depto*
- El resultado es una posible repetición de información

ID	*nombre*	*salario*	*nombre_dpto*	*edificio*	*presupuesto*
22222	Ana	45000	Física	Fac. Ciencias	90000
12121	Felipe	40000	Economía	Fac. Económicas	120000
32343	Juan	30000	Historia	Fac. Humanidades	40000
45565	María	35000	Telemática	Esc. Teleco	230000
98345	José	40000	Señal y Comun.	Esc. Teleco	180000
76766	Asunción	32000	Biología	Fac. Ciencias	95000
10101	Mariano	35000	Telemática	Esc. Teleco	230000
58583	Loreto	50000	Historia	Fac. Humanidades	40000
83821	Pedro	47000	Telemática	Esc. Teleco	230000
15151	Luis	33000	Música	Fac. Humanidades	12000
33456	Paula	40000	Física	Fac. Ciencias	90000
76543	Lucía	42000	Economía	Fac. Económicas	120000

Relación *docente_dpto*

Un esquema combinado sin redundancia

- Si ahora combinamos las relaciones
  - *grupo_aula(id_grupo, edificio, codigo_aula)*
  - *grupo(id_materia, id_grupo, cuatrimestre, año)*

  En una relación
  - *grupo(id_materia, id_grupo, cuatrimestre, año, edificio, codigo_aula)*
- No hay información repetida.

¿Definir esquemas más pequeños?

- Supongamos que empezamos nuestro diseño con *docente_depto*. ¿Cómo podríamos saber que hay que dividirla (**descomponerla**) en *docente* y *departamento*?
- Fijémonos en esto: "si hubiera un esquema (nombre_*depto, edificio, presupuesto*), entonces nombre_*depto* sería una clave candidata"
- Lo denominamos una **dependencia funcional**:

  nombre_dpto ➛ *edificio, presupuesto*

- En *docente_depto*, dado que *nombre_depto* no es una clave candidata, el edificio y el presupuesto de un departamento pueden tener que repetirse.
  - Esto indica la necesidad de descomponer *docente_depto*
- No todas las descomposiciones son adecuadas. Supongamos que descomponemos

  empleado (ID, nombre, calle, ciudad, salario) en

  empleado1 (*ID*, *nombre*)

  empleado2 (*nombre*, *calle, ciudad, salario*)

- ¡Perdemos información! – no podemos reconstruir la relación original *empleado* – y, por tanto, es una **descomposición con pérdidas**.

Una descomposición con pérdidas

ID	*nombre*	*calle*	*ciudad*	*salario*
…				
57766	Ana	Florida	Madrid	75000
98776	Ana	Norte	Barcelona	67000
…				

empleado

ID	*nombre*
…	
57766	Ana
98776	Ana
…	

nombre	*calle*	*ciudad*	*salario*
Ana	Florida	Madrid	75000
Ana	Norte	Barcelona	67000

Join natural

ID	*nombre*	*calle*	*ciudad*	*salario*
…				
57766	Ana	Florida	Madrid	75000
57766	Ana	Norte	Barcelona	67000
98776	Ana	Florida	Madrid	75000
98776	Ana	Norte	Barcelona	67000
…				

Ejemplo de una descomposición sin pérdidas (Reversible por Join)

- Descomposición de $R = (A, B, C)$

$R_1 = (A, B)$ $R_2 = (B, C)$

A	B	C
α	1	A
β	2	B

r

A	B
α	1
β	2

$\Pi_{A,B}(r)$

B	C
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_A(r) \bowtie \Pi_B(r)$

A	B	C
α	1	A
β	2	B

Objetivo: Definir una teoría para:

- Decidir cuándo una determinada relación *R* presenta una forma “adecuada”.
- En el caso de que la relación *R* no esté en una forma “adecuada”, descomponerla en un conjunto de relaciones $\{R_1, R_2, ..., R_n\}$ tales que:
  - cada relación presente una forma adecuada.
  - La descomposición es una descomposición reversible por join
- Nuestra teoría se base en:
  - dependencias funcionales.
  - dependencias multivaluadas.

Dependencias funcionales

- Restricciones sobre el conjunto de valores permitidas en una relación.
- Indica que el valor de un cierto conjunto de atributos determina de manera unívoca el valor de otro conjunto de atributos.
- El concepto de dependencia funcional es una generalización del concepto de *clave*.

Dependencias funcionales

- Dado un esquema de relación *R*

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

- La **dependencia funcional**

$$\alpha \rightarrow \beta$$

se da en *R* si y sólo si para cualquier instancia legal *r*(R), si dos tuplas cualquieras t_1 y t_2 de *r* coinciden en los valores de los atributos α, entonces también coinciden en los valores de los atributos β. Es decir:

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

- Ejemplo: Dado *R*(A,B) con la siguiente instancia de *R.*

1	**4**
1	**5**
3	**7**

- En esta instancia, $A \rightarrow B$ **NO** se cumple, pero $B \rightarrow A$ se cumple.

Dependencias funcionales

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

  docente_depto (*ID*, *nombre*, *salario*, *nombre_dpto*, *edificio*, *presupuesto*).

  se deberían cumplir las siguientes dependencias funcionales:

  $$nombre\_depto \rightarrow edificio$$

  y $ID \rightarrow edificio$

  pero no se debería cumplir:

  $$nombre\_dpto \rightarrow salario$$

Utilidades de las dependencias funcionales

- Las dependencias funcionales se usan para:
  - Comprobar si las instancias de relaciones son legales dado un determinado conjunto de dependencias funcionales.
    - Si una instancia de relación *r* es legal bajo un conjunto de dependencias funcionales *F*, decimos que *r* **satisface** *F.*
  - Especificar restricciones sobre una relación.
    - Decimos que *F* **se cumple en** *R* si todas las instancias legales de *R* satisfacen el conjunto de dependencias funcionales *F.*
- Nota: Una instancia determinada de un esquema de relación puede satisfacer una dependencia funcional aún cuando esa dependencia funcional no se cumpla para todas las instancias legales.
  - Por ejemplo, una instancia específica de *docente* puede, por casualidad, satisfacer

    nombre → *ID* o *nombre* → *salario.*

Dependencias funcionales

- Una dependencia funcional es **trivial** si se satisface para todas las instancias posibles de una relación.
  - Ejemplo*:*
    - *ID, nombre* → *ID*
    - *nombre* → *nombre*
  - En general, $\alpha \rightarrow \beta$ es trivial si $\beta \subseteq \alpha$

Cierre de un conjunto de dependencias funcionales

- Dado un conjunto de de dependencias funcionales *F*, hay otras dependencias funcionales que se pueden inferir lógicamente a partir de *F*.
  - Por ejemplo: Si $A \rightarrow B$ y $B \rightarrow C$, entonces podemos inferir que $A \rightarrow C$
- El conjunto de **todas** las dependencias funcionales inferidas a partir de *F* forma el **cierre** de *F*.
- Denotaremos el cierre de *F* como **F^+**.
- F^+ es un superconjunto de *F*.

Forma normal de Boyce-Codd

Un esquema de relación *R* está en FNBC con respecto a un conjunto de dependencias funcionales *F* si para todas las dependencias funcionales de F^+ expresadas como

$$\alpha \rightarrow \beta$$

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:

docente_depto (*ID*, *nombre*, *salario*, *nombre_depto*, *edificio*, *presupuesto*)

ya que *nombre_dept* → *edificio*, *presupuesto*
se cumple en *docente_depto*, pero *nombre_depto* no es una superclave.

Descomponer un esquema a FNBC

- Dado un esquema *R* y una dependencia no trivial α→β que viola la FNBC.

  Descomponemos *R* en:
  - (α U β)
  - (*R* - (β - α))
- En nuestro ejemplo,
  - α = *nombre_depto*
  - β = *edificio, presupuesto*

  y *docente_depto* se sustituye por
  - (α U β) = (*nombre_depto*, *edificio, presupuesto*)
  - (*R* - (β - α)) = (*ID*, *nombre, salario, nombre_depto*)

FNBC y preservación de dependencias

- Las restricciones, incluyendo las dependencias funcionales, son costosas de comprobar en la práctica, a menos que afecten a una sola relación.
- Si es suficiente con comprobar sólo las dependencias funcionales de cada relación individual de una descomposición para asegurar que **todas** las dependencias funcionales se cumplen, entonces la descomposición **preserva las dependencias**.
- No siempre es posible conseguir cumplir la FNBC y preservar las dependencias.
  - Para estos casos consideramos una forma normal más débil, denominada **tercera forma normal**.

Tercera forma normal

- Un esquema de relación *R* está en **tercera forma normal (3FN)** si, para todas las:

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

  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*
  - Cada atributo *A* de $\beta - \alpha$ forma parte de una clave candidata de *R.*

    (**NOTA**: cada atributo puede formar parte de una clave candidata distinta)
- Si una relación está en FNBC entonces está en 3FN (dado que en la FNBC se deben cumplir alguna de las dos primeras condiciones anteriores).
- La tercera condición representa la relajación mínima que es necesario realizar para asegurar que se preservan las dependencias.

Objetivos de la normalización

- Dado el esquema de relación *R* con un conjunto de dependencias funcionales *F*:
  - Decidir si el esquema *R* está en una forma "adecuada".
  - En el caso de que el esquema *R* no esté en una forma "adecuada", descomponerlo en un conjunto de esquemas de relación $\{R_1, R_2, ..., R_n\}$ tales que:
    - cada esquema de relación esté en una forma adecuada.
    - la descomposición es una descomposición reversible por join.
    - a ser posible, la descomposición debería preservar las dependencias.

Comparación de FNBC y 3FN

- Siempre es posible descomponer un esquema en un conjunto de esquemas que estén en 3FN, de tal forma que:
  - la descomposición sea reversible por join.
  - se preservan las dependencias.
- Siempre es posible descomponer un esquema en un conjunto de esquemas que estén en FNBC, de tal forma que:
  - la descomposición sea reversible por join.
  - puede que no sea posible preservar las dependencias.

Objetivos de diseño

- Los objetivos del diseño de una base de datos relacional son:
  - FNBC.
  - Descomposición reversible por join.
  - Preservar dependencias.
- Si no se puede conseguir, debemos aceptar:
  - perder la preservación de dependencias, o
  - admitir redundancia y utilizar la 3FN.

- SQL no proporciona una forma directa de especificar dependencias funcionales. Sólo soporta el concepto de superclave.

  Podemos especificar DFs utilizando *aseveraciones*, pero son operaciones costosas (y no soportadas por los SGBD más utilizados)
- Aunque tengamos una descomposición que preserve las dependencias, utilizando SQL no seremos capaces de comprobar las DFs de una forma eficiente cuando la parte izquierda no es clave.

FNBC: Propiedades

- Determinados esquema en FNBC no parecen estar lo suficientemente normalizados.
- Dada la relación

  docente_info (ID, nombre_hijo, telefono)
  - en la que un docente puede tener varios teléfonos y varios hijos

ID	*nombre_hijo*	*telefono*
99999	David	555-555-1234
99999	David	555-555-4321
99999	Guillermo	555-555-1234
99999	Guillermo	555-555-4321

docente_info

FNBC: Propiedades

- No hay dependencias funcionales no triviales y, por tanto, está en FNBC
- Anomalías de inserción: por ejemplo, si añadimos un teléfono 981-992-3443 a 99999, necesitamos añadir dos tuplas

(99999, David, 981-992-3443)
(99999, Guillermo, 981-992-3443)

FNBC: Propiedades

- Es mejor descomponer *docente_info* en:

docente_hijo

ID	*nombre_hijo*
99999	David
99999	David
99999	Guillermo
99999	Guillermo

docente_tlfno

ID	*telefono*
99999	512-555-1234
99999	512-555-4321
99999	512-555-1234
99999	512-555-4321

Esto sugiere la necesidad de formas normales superiores, tal como la **Cuarta Forma Normal**

Otras formas normales de orden superior

- las **dependencias multivaluadas** generalizan las dependencias funcionales.
  - dan lugar a la **cuarta forma normal** (**4FN**)
- Las **dependencias de join** generalizan las dependencias multivaluadas.
  - dan lugar a la **forma normal por proyección de join** (**FNPJ**) (también llamada **quinta forma normal**)
- Un tipo de restricciones aún más generales, dan lugar a una forma normal denominada **forma normal clave-dominio**.
- Problema de esta restricciones generalizadas: es difícil razonar sobre ellas y no existe un conjunto de reglas de inferencia consistente y completo.
- Muy raramente utilizadas.

Primera Forma Normal

- Un dominio es **atómico** si sus elementos se consideran unidades indivisibles
  - Ejemplos de dominios no atómicos:
    - Conjuntos de nombres, atributos compuestos
    - Identificadores como IT101 que se pueden dividir en partes
- Un esquema relacional R está en **primera forma normal** si los dominios de todos sus atributos son atómicos.
- Los valores no atómicos complican el almacenamiento y provocan el almacenamiento de información redundante.
  - Ejemplo: Conjunto de asignaturas almacenadas con cada alumno, y conjunto de alumnos almacenado con cada asignatura.
  - Asumiremos que todas nuestras relaciones están en primera forma normal.

Primera Forma Normal

- La atomicidad es realmente una propiedad derivada de cómo se van a utilizar los datos.
  - Ejemplo: Los strings normalmente se deberían considerar indivisibles.
  - Supongamos que asignamos códigos de matrícula a los alumnos, que son strings de la forma *IT0012* o *TE1127*.
  - Si los dos primeros caracteres se extraen para identificar el departamento, el dominio de los códigos de matrícula no es atómico.
  - Es una mala idea: de la codificación de la información debería realizarse en los programas de aplicación, no en la base de datos.

Modelo E-A y normalización

- Cuando un diagrama E-A se diseña adecuadamente, identificando correctamente todas las entidades, las relaciones que se generan a partir de dicho diagrama no deberían necesitar ninguna normalización adicional.
- En un diseño real (imperfecto), puede haber dependencias funcionales desde atributos que no son clave hacia otros atributos de una entidad.
  - Ejemplo: una entidad *empleado* con atributos *nombre_departamento* y *edificio*, y una dependencia funcional *nombre_departamento* → *edificio*
  - Un buen diseño debería haber hecho departamento una entidad
- Es posible que también existan dependencias funcionales de atributos no clave en una asociación, pero no es habitual – la mayoría de las asociaciones son binarias.

Desnormalización por prestaciones

- A veces podemos querer utilizar esquemas no normalizados por motivos de prestaciones.
- Por ejemplo, para mostrar *prerreqs* junto con *id_materia* y *nombre_materia* necesitamos hacer el join entre *materia y prerreq*
- Alternativa 1: Utilizar una relación desnormalizada que contenga atributos de *materia* y también de *prerreq,* incluyendo todos los indicados
  - búsquedas más rápidas
  - necesario más espacio y tiempo de ejecución para actualizaciones
  - necesario más código y, por tanto, más posibilidades de errores de codificación
- Alternativa 2: utilizar una vista definida como:

  $materia \bowtie prerreq$
  - Los mismo beneficios y problemas que antes, excepto que no necesitamos código extra y, por tanto, evita errores de programación.

Otras cuestiones de diseño

- La normalización no previene respecto a todos los posibles errores de diseño.
- Ejemplo de malos diseños de bases de datos:

  En vez de utilizar *ganancias* (*id_compania, ano, cantidad*), utilizar
  - *ganancias_2010, ganancias_2011, ganancias_2012*, etc., todas con el esquema (*id_compania, cantidad*).
    - Estos esquemas están en FNBC, pero realizar consultas que abarquen varios años es complicado, y además hay que crear una nueva relación cada año.
  - *compania_ano* (*id_compania, ganancias_2010, ganacias_2011, ganancias_2012*)
    - También están en FNBC, pero también es complicado realizar consultas que abarquen varios años, y además requiere un nuevo atributo cada año.
    - Es un ejemplo de una **tabla cruzada**, en la que los valores de un atributo se convierten en nombres de columnas.
    - Se utilizan en hojas de cálculo, y en herramientas de análisis de datos.

Teoría de dependencias funcionales

- Teoría formal que permite calcular qué dependencias funcionales de pueden inferir lógicamente a partir de un conjunto dado de dependencias funcionales.
- Permite definir algoritmos para:
  - comprobar si una descomposición es reversible por join.
  - comprobar si una descomposición preserva las dependencias.
  - generar descomposiciones a FNBC que preserven las dependencias, si es posible.
  - generar descomposiciones a 3FN
- Es una teoría compleja que no vamos a ver, pero a continuación veremos algunos de sus resultados (algoritmos), por su utilidad práctica.

Descomposición reversible por join

- Dado $R = (R_1, R_2)$, necesitamos que para todas las posibles instancias r sobre el esquema R

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

- Una descomposición de R en R_1 y R_2 es una descomposición reversible por join si, al menos, una de las siguientes dependencias están 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
- $R_1 = (A, B), \quad R_2 = (B, C)$
  - Descomposición reversible por join:
    $R_1 \cap R_2 = \{B\}$ y $B \rightarrow BC$
  - Preserva las dependencias
- $R_1 = (A, B), \quad R_2 = (A, C)$
  - Reversible por join:
    $R_1 \cap R_2 = \{A\}$ y $A \rightarrow AB$
  - No preserva las dependencias
    (no podemos comprobar $B \rightarrow C$ sin calcular $R_1 \bowtie R_2$)

Comprobación de preservación de dependencias

- Para comprobar si una dependencia $\alpha \rightarrow \beta$ se preserva en una descomposición de R en $R_1, R_2, \ldots, R_n$ se aplica la siguiente prueba (con cierre de atributos respecto a F)
  - *resultado* = α
    while (*resultado* cambie) do
      for each R_i
        $t = (resultado \cap R_i)^+ \cap R_i$
        $resultado = resultado \cup t$
  - Si *resultado* contiene todos los atributos de β, entonces la dependencia funcional $\alpha \rightarrow \beta$ se preserva.
- Se debe aplicar la prueba a todas las dependencias del conjunto F para comprobar si la descomposición preserva las dependencias.
- Este procedimiento es de complejidad polinomial, frente a la complejidad exponencial de comprobar directamente $F^+ = (F_1 \cup F_2 \cup \ldots \cup F_n)^+$

Cálculo del cierre de un conjunto de atributos

- Dado un conjunto de atributos α, definimos el cierre de α respecto a *F* (α^+) como el conjunto de atributos que son determinados funcionalmente por α utilizando *F*

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

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

Ejemplo de comprobación de preservación de dependencias

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C\}$
  Clave = $\{A\}$
- R no está en FNBC
- Descomposición $R_1 = (A, B)$, $R_2 = (B, C)$
  - R_1 y R_2 están en FNBC
  - Descomposición reversible por join
  - Preserva las dependencias

Comprobación de FNBC

- Para comprobar si una dependencia no trivial $\alpha \rightarrow \beta$ viola la FNBC:
  1. calcular α^+
  2. comprobar que α^+ incluye todos los atributos de *R*, es decir, es una superclave de *R*.
- **Comprobación simplificada**: Para comprobar si un esquema de relación *R* está en FNBC, es suficiente comprobar si alguna de las dependencias del conjunto dado viola la FNBC, en vez de comprobar todas las dependencias en F^+.
  - Si ninguna de las dependencias de *F* viola la FNBC, entonces ninguna de las dependencias de F^+ viola la FNBC.
- No obstante, **la comprobación simplificada utilizando sólo *F* es incorrecta cuando comprobamos una relación de una descomposición de R**
  - Dada $R = (A, B, C, D, E)$, con $F = \{ A \rightarrow B, BC \rightarrow D\}$
    - Descomponemos *R* en $R_1 = (A,B)$ y $R_2 = (A,C,D, E)$
    - Ninguna de las dependencias de *F* contiene sólo atributos de (A,C,D,E), por eso nos podemos equivocar pensando que R_2 satisface la FNBC.
    - De hecho, la dependencia $AC \rightarrow D$ de F^+ hace que R_2 no esté en FNBC.

Comprobación de FNBC en una descomposición

- Para comprobar si un esquema R_i de una descomposición de R está en FNBC,
  - O bien comprobamos si R_i está en FNBC con respecto a la **proyección** de F en R_i (es decir, todas las DFs en F^+ que contienen solamente atributos de R_i)
  - O bien utilizamos el conjunto de dependencias original F que se cumplen en R, pero con la siguiente comprobación:
    - para cada conjunto de atributos $\alpha \subseteq R_i$, comprobar que α^+ o bien no incluye ningún atributo de R_i-α, o bien incluye todos los atributos de R_i.
  - Si alguna $\alpha \rightarrow \beta$ de F viola la condición, entonces la dependencia
    $\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$
    se cumplirá en R_i, y R_i violará la FNBC.
  - Utilizaremos ésta dependencia para descomponer R_i

Algoritmo de descomposición de FNBC

resultado := {*R*};
fin := false;
calcular F^+;
while (not *fin)* **do**
    if (hay un esquema R_i en *resultado* que no está en FNBC)
        then begin
            dado $\alpha \rightarrow \beta$, dependencia funcional no trivial que se cumple
            en R_i tal que $\alpha \rightarrow R_i$ no pertenece a F^+,y $\alpha \cap \beta = \varnothing$;
                resultado := (*resultado* – R_i *)* $\cup$ (R_i – β) $\cup$ (α, β *);*
            end
        else *fin* := true;

Cada R_i estará en FNBC, y la descomposición será reversible por join.

Ejemplo de descomposición 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 superclave)
- Descomposición
  - $R_1 = (B, C)$
  - $R_2 = (A, B)$

Ejemplo de descomposición FNBC

- *clase* (*id_materia*, *nombre_materia*, nombre_*depto*, *creditos*, id_*grupo*, *cuatrimestre*, *año*, *edificio*, *codigo_aula*, *capacidad*, *id_horario*)
- Dependencias funcionales:
  - *id_materia*→ *nombre_materia*, nombre_*depto*, *creditos*
  - *edificio*, *codigo_aula*→*capacidad*
  - *id_materia*, *id_grupo*, *cuatrimestre*, *año*→*edificio*, *codigo_aula*, *id_horario*
- Una clave candidata {*id_materia*, *id_grupo*, *cuatrimestre*, *año*}.
- Descomposición FNBC:
  - *id_materia*→ *nombre_materia*, *nombre_depto*, *creditos se cumple,*
    - pero *id_materia* no es una superclave.
  - Reemplazamos *clase* por:
    - *materia* (*id_materia*, *nombre_materia*, nombre_*depto*, *creditos*)
    - *clase-1* (*id_materia*, *id_grupo*, *cuatrimestre*, año, *edificio*, *codigo_aula, capacidad*, *id_horario*)

Ejemplo de descomposición FNBC

- *materia* está en FNBC
- *edificio*, *codigo_aula*→*capacidad* se cumple en *clase-1*
  - pero {*edificio*, *codigo_aula*} no es una superclave para *clase-1*.
  - sustituimos *clase-1* por:
    - *aula* (*edificio*, *codigo_aula*, *capacidad*)
    - *grupo* (*id_materia*, *id_grupo*, *cuatrimestre*, *año*, *edificio*, *codigo_aula*, *id_horario*)
- *aula* y *grupo* están en FNBC.

FNBC y preservación de dependencias

No siempre se posible descomponer en FNBC preservando las dependencias

- $R = (J, K, L)$
  $F = \{JK \rightarrow L$
  $L \rightarrow K\}$
  Dos claves candidatas = JK y JL
- R no está en FNBC
- Cualquier descomposición de R “perderá”:

$$JK \rightarrow L$$

Esto implicaría que comprobar $JK \rightarrow L$ requerirá un join

Redundancia en 3FN

- En este esquema hay alguna redundancia.
- Ejemplo de problemas debidos ala redundancia de la 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

- Información repetida (por ejemplo, la asociación l_1, k_1)
  - (*d_ID, nombre_dept)*
- necesidad de usar valores nulos (por ejemplo, para representar la asociación l_2, k_2, que no tiene un valor para *J*).
  - (*d_ID, nombre_depto)* si no hay una relación aparte que relaciones profesores con departamentos.

Comprobación de 3FN

- Optimización: Necesidad de comprobar sólo las dependencias funcionales de *F*, no todas las de F^+.
- Utilizamos el cierre de atributos para comprobar para cada dependencia $\alpha \rightarrow \beta$, si α es una superclave.
- Si α no es una superclave, tenemos que comprobar si cada atributo de β forma parte de una clave candidata de *R*
  - Esta comprobación es bastante más costosa, dado que supone encontrar las claves candidatas
  - La comprobación de 3FN se ha demostrado que es NP-hard
  - Pero la descomposición a 3FNse puede hacer en tiempo polinomial.

Algoritmo de descomposición en 3FN

Dado F_c (cobertura canónica de F);
$i := 0$;
for each $\alpha \rightarrow \beta$ en F_c **do**
  if ningún R_j, $1 \leq j \leq i$ contiene $\alpha\beta$ **then begin**
    $i := i + 1$;
    $R_i := \alpha\beta$;
  end
if ningún R_j, $1 \leq j \leq i$ contiene una clave candidata de R **then begin**
  $i := i + 1$;
  R_i := cualquier calve candidata de R;
  end
/* Opcionalmente, eliminar relaciones redundantes */

repeat
if algún R_j está incluido en otro R_k **then** /* borrar R_j */
  $R_j = R$;;
  $i=i-1$;
return $(R_1, R_2, ..., R_i)$

Cada R_i estará en 3FN, y la descomposición será reversible por join y preservará las dependencias.

Ejemplo de descomposición en 3FN

- Esquema de relación:

  cliente_banquero_sucursal = (*id_cliente, id_empleado, nombre_sucursal, tipo*)

- Dependencias funcionales:

  1. *id_cliente, id_empleado* → *nombre_sucursal, tipo*
  2. *id_empleado* → *nombre_sucursal*
  3. *id_cliente, nombre_sucursal* → *id_empleado*

- Calculamos la cobertura canónica
  - F_C = *id_cliente, id_empleado* → *tipo*
    id_empleado → *nombre_sucursal*
    id_cliente, nombre_sucursal → *id_empleado*

Ejemplo de descomposición en 3FN

- El bucle **for** genera los siguientes esquemas en 3FN:

  (*id_cliente, id_empleado, tipo*)

  (*id_empleado, nombre_sucursal*)

  (*id_cliente, nombre_sucursal, id_empleado)*

  - Observar que (*id_cliente, id_empleado, tipo*) contiene una clave candidata del esquema original, por lo que no es necesario añadir más esquemas de relación

- Al finalizar el bucle, eliminamos esquemas redundantes, como (*id_empleado, nombre_sucursal*), que es un subconjunto de otros esquemas
  - el resultado no depende del orden en que se consideren las DFs

- El esquema 3FN simplificado resultante es:

  (*id_cliente, id_empleado, tipo*)

  (*id_cliente, nombre_sucursal, id_empleado)*

FIN DEL TEMA 3