



# *Bases de Datos*

## Tema 6: Diseño lógico

*Dpto. de Ingeniería Informática*

# Contenidos

- Introducción
- Guías generales de diseño
- Dependencias funcionales
- Normalización de Codd
- Mapeo de (E)E-R a relaciones
- Dependencias multivaluadas
- Normalización avanzada
- Referencias

# Introducción

- Por ahora hemos visto modelos E-E/R
  - Son diseños lógicos/conceptuales
- Y hemos visto esquemas relacionales
  - Que son los que “se implementan” en los SGBD
- Hay un algoritmo que nos ayuda a pasar de conceptual a relacional
- No obstante, veremos antes algunas medidas rigurosas de la “calidad/bondad” de un esquema relacional

# Introducción

- La bondad de un esquema relacional se puede evaluar desde dos perspectivas:
  - Conceptual: cómo el E/R permite que el usuario entienda los esquemas. Esto facilita que las consultas se realicen de manera sencilla (tanto a relaciones base como a vistas)
  - Físico: cómo se almacenan las tuplas (de las relaciones base) en disco. Afecta al rendimiento, siendo transparente por lo demás (queda fuera del alcance de esta asignatura)

# Introducción

- Como en otros problemas de diseño hay dos alternativas:
  - *Bottom-up*: se basa en las relaciones entre atributos individuales, y va añadiendo otros al resultado. No es muy usada
  - *Top-down*: parte de agrupaciones de atributos fruto del mapeo de un diseño conceptual. Se aplica “Design by analysis”. Es la más usada
  - No obstante, lo que veremos es aplicable a las dos alternativas, aunque se maneja mejor con la segunda

# Guías generales de diseño

- Las indicaciones que veremos a continuación son informales, pero suelen prevenir problemas en fases posteriores:
  - Semántica de los atributos
  - Reducción de valores redundantes en tuplas
  - Reducción de valores nulos en tuplas
  - Evitar la generación de tuplas espurias
- Estas guías a veces están interconectadas
- Las veremos con un ejemplo sobre proyectos de empresa más complejo que la biblioteca ...

# Tablas de ejemplo: BD empresa

- Relaciones de ejemplo:
  - Empleado(nombre\_e, nss, fecha\_nac, direcc, *cod\_dept*)
  - Departamento(nombre\_d, cod\_dept, *nss*)
  - Localizaciones\_departam(*cod\_dept*, localiz\_dept)
  - Proyecto(nombre\_p, cod\_proy, localiz\_p, *cod\_dept*)
  - Trabaja(*nss*, cod\_proy, horas)

Las claves primarias van subrayadas (todos sus atributos) y las claves foráneas van en cursiva compartiendo nombre en las relaciones

# Semántica de los atributos

- La base del modelo relacional es la agrupación de atributos en torno a relaciones
- La semántica es lo que permite interpretar la información del mundo a modelar que almacena una tupla concreta de una relación
  - En general, cuanto más fácil de leer sea una relación, mejor estará diseñada
    - A veces el único diseño posible no es fácil de leer ... :(
- Un diseño adecuado debe de reflejar (casi) toda la semántica del mundo a modelar
  - Sólo dejar fuera aquello que el modelo no soporta



# Semántica de los atributos

- En el ejemplo:
  - Empleado(nombre\_e, nss, fecha\_nac, direcc, *cod\_dept*)
    - Hay un empleado que se llama *nombre\_e*, con número de seguridad social *nss*, que nació en *fecha\_nac*, vive en *direcc*, y trabaja para el departamento *cod\_dept*
    - El *cod\_dept* es una clave foránea que representa una relación implícita entre Empleado y Departamento
  - Departamento(nombre\_d, *cod\_dept*, nss)
    - ¿?
  - Proyecto (nombre\_p, cod\_proy, localiz\_p, *cod\_dept*)
    - ¿?

# Semántica de los atributos

- En el ejemplo:
  - Empleado(nombre\_e, nss, fecha\_nac, direcc, *cod\_dept*)
    - Hay un empleado que se llama *nombre\_e*, con número de seguridad social *nss*, que nació en *fecha\_nac*, vive en *direcc*, y trabaja para el departamento *cod\_dept*
    - El *cod\_dept* es una clave foránea que representa una relación implícita entre Empleado y Departamento
  - Departamento(nombre\_d, *cod\_dept*, *nss*)
    - Hay un departamento que se llama *nombre\_d*, con código *cod\_dept*, que tiene de gestor al trabajador con número de seguridad social *nss*
  - Proyecto(nombre\_p, *cod\_proy*, localiz\_p, *cod\_dept*)
    - También es inmediata

# Semántica de los atributos

- En el ejemplo:
  - Localizaciones\_departam(cod\_dept, localiz\_dept)
    - ¿Es igual de fácil de leer?
  - Trabaja(nss, cod\_proy, horas)
    - ¿Es igual de fácil de leer?

# Semántica de los atributos

- En el ejemplo:
  - Localizaciones\_departam(*cod\_dept*, *localiz\_dept*)
    - Cada tupla indica un departamento *cod\_dept* y una de las (posibles varias) localizaciones que tiene *localiz\_dept*
  - Trabaja(*nss*, *cod\_proy*, horas)
    - Cada tupla indica el código de la seguridad social de un empleado *nss*, y uno de los (posibles varios) proyectos en que trabaja *cod\_proy* y el número de horas (semanales) que el empleado dedica a ese proyecto *horas*
  - Evidentemente son más complicados de leer, sin embargo no son ambiguos
    - ¿De qué parte de un E/R provienen?

# Semántica de los atributos

- En el ejemplo:
  - Localizaciones\_departam(cod\_dept, localiz\_dept)
    - Es un atributo multivaluado de proyecto: un proyecto tiene varias localizaciones
  - Trabaja(nss, cod\_proy, horas)
    -
  - Más adelante veremos cómo se generan estas relaciones a partir de un E/R
    - Pero lo que está claro es que no siendo tan cómodas como las otras relaciones, son un buen diseño

# Semántica de los atributos

- En el ejemplo:
  - Localizaciones\_departam(cod\_dept, localiz\_dept)
    - Es un atributo multivaluado de proyecto: un proyecto tiene varias localizaciones
  - Trabaja(nss, cod\_proy, horas)
    - Es una relación M:N entre los proyecto y empleado
  - Más adelante veremos cómo se generan estas relaciones a partir de un E/R
    - Pero lo que está claro es que no siendo tan cómodas como las otras relaciones, son un buen diseño

# Semántica de los atributos

- Guía 1 para el diseño:
  - Diseñar esquemas sencillos y autoexplicativos
  - Que no incluyan atributos de varias entidades (o varias relaciones entre entidades) en una misma relación
  - De manera intuitiva, si el esquema proviene de una única entidad o relación entre entidades del mundo a modelar suele ser adecuado

# Semántica de los atributos

- Si cambiamos el ejemplo anterior:
  - Pasamos de tener:
    - Empleado(nombre\_e, nss, fecha\_nac, direcc, *cod\_dept*)
    - Departamento(nombre\_d, *cod\_dept*, nss)
  - A tener (renombrando el nss del gestor):
    - Empleado\_dept(nombre\_e, nss, fecha\_nac, direcc, *cod\_dept*, nombre\_d, nss\_gestor)
  - ¿Qué ventajas e inconvenientes presentaría?



# Semántica de los atributos

- Si cambiamos el ejemplo anterior:
  - Pasamos de tener:
    - Empleado(nombre\_e, nss, fecha\_nac, direcc, *cod\_dept*)
    - Departamento(nombre\_d, *cod\_dept*, nss)
  - A tener (renombrando el nss del gestor):
    - Empleado\_dept(nombre\_e, nss, fecha\_nac, direcc, *cod\_dept*, nombre\_d, nss\_gestor)
  - ¿Qué ventajas e inconvenientes presentaría?
    - Nos ahorramos una relación y una clave foránea
    - Pero se incluyen atributos de dos entidades

# Semántica de los atributos

- Si cambiamos el ejemplo anterior otra:
  - Pasamos de tener:
    - Trabaja(nss, cod\_proy, horas)
    - Proyecto(nombre\_p, cod\_proy, localiz\_p, cod\_dept)
  - A tener:
    - Trabaja\_proy(nss, cod\_proy, horas, nombre\_e, nombre\_p, localiz\_p)
  - Idéntico problema:
    - Estamos incluyendo atributos de dos relaciones
- No son malos diseños para vistas, pero sí como relaciones base

# Reducción de valores redundantes en tuplas

- Uno de los diversos objetivos del diseño de esquemas es reducir el espacio que ocupan en disco las relaciones base
- Volviendo al ejemplo anterior:
  - Pasamos de:
    - Empleado(nombre\_e, nss, fecha\_nac, direcc, *cod\_dept*)
    - Departamento(nombre\_d, *cod\_dept*, nss\_gestor)
  - A tener:
    - Empleado\_dept(nombre\_e, nss, fecha\_nac, direcc, *cod\_dept*, nombre\_d, nss\_gestor)
  - ¿Existe un ahorro de espacio?

# Reducción de valores redundantes en tuplas

- Uno de los diversos objetivos del diseño de esquemas es reducir el espacio que ocupan en disco las relaciones base
- Volviendo al ejemplo anterior:
  - Pasamos de:
    - Empleado(nombre\_e, nss, fecha\_nac, direcc, *cod\_dept*)
    - Departamento(nombre\_d, cod\_dept, nss\_gestor)
  - A tener:
    - Empleado\_dept(nombre\_e, nss, fecha\_nac, direcc, *cod\_dept*, nombre\_d, nss\_gestor)
  - Se ahorra un atributo en el esquema, pero ¿se ahorra espacio en disco?

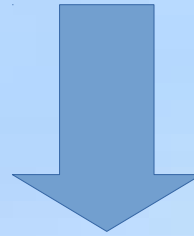
# Reducción de valores redundantes en tuplas

- Uno de los diversos objetivos del diseño de esquemas es reducir el espacio que ocupan en disco las relaciones base
- Volviendo al ejemplo anterior:
  - Pasamos de:
    - Empleado(nombre\_e, nss, fecha\_nac, direcc, *cod\_dept*)
    - Departamento(nombre\_d, cod\_dept, nss\_gestor)
  - A tener:
    - Empleado\_dept(nombre\_e, nss, fecha\_nac, direcc, *cod\_dept*, nombre\_d, nss\_gestor)
  - Se ahorra un atributo en el esquema, pero ¿se ahorra espacio en disco? Empleado\_dept es un ✕

# Reducción de valores redundantes en tuplas

- Ejemplo: Empleado  $\bowtie$  Departamento

Nombre_e	...	Cod_Dept
Charo	...	1
María José	...	1
Alejandra	...	2
Pedro	...	2
Martina	...	2



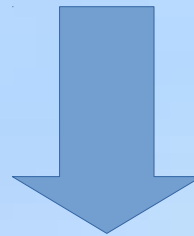
Nombre_d	Cod_Dept	nss_gestor
Ventas	1	1
Compras	2	4

# Reducción de valores redundantes en tuplas

- Ejemplo: Empleado  $\bowtie$  Departamento

Nombre_e	...	Cod_Dept
Charo	...	1
María José	...	1
Alejandra	...	2
Pedro	...	2
Martina	...	2

Nombre_d	Cod_Dept	nss_gestor
Ventas	1	1
Compras	2	4



Nombre_e	...	Cod_Dept	Nombre_d	nss_gestor
Charo		1	Ventas	1
María José	...	1	Ventas	1
Alejandra	...	2	Compras	4
Pedro	...	2	Compras	4
Martina	...	2	Compras	4

# Reducción de valores redundantes en tuplas

- ¿Es el producto natural una forma de ahorrar espacio?
  - Evidentemente no, porque se repiten ...



# Reducción de valores redundantes en tuplas

- ¿Es el producto natural una forma de ahorrar espacio?
  - Evidentemente no, porque se repiten todos los atributos de cada departamento  $N$  veces (siendo  $N$  el número medio de empleados por departamento)

# Reducción de valores redundantes en tuplas

- ¿Es el producto natural una forma de ahorrar espacio?
  - Evidentemente no, porque se repiten todos los atributos de cada departamento N veces (siendo N el número medio de empleados por departamento)
  - Al tener tablas separadas también hay valores redundantes, pero sólo ...

# Reducción de valores redundantes en tuplas

- ¿Es el producto natural una forma de ahorrar espacio?
  - Evidentemente no, porque se repiten todos los atributos de cada departamento N veces (siendo N el número medio de empleados por departamento)
  - Al tener tablas separadas también hay valores redundantes, pero sólo el conjunto de atributos que son clave foránea

# Reducción de valores redundantes en tuplas

- Pero además existen otros problemas:
  - Anomalías de actualización:
    - Anomalías de inserción
    - Anomalía de borrado
    - Anomalía de modificación

Nombre_e	...	Cod_Dept	Nombre_d	nss_gestor
Charo		1	Ventas	1
María José	...	1	Ventas	1
Alejandra	...	2	Compras	4
Pedro	...	2	Compras	4
Martina	...	2	Compras	4

# Reducción de valores redundantes en tuplas

- Anomalías de inserción
  - Primer caso: todo nuevo empleado debe tener todos los datos de su departamento (o nulos si no tiene departamento asignado)
    - Eso implica que para insertar un empleado hay que ...

Nombre_e	...	Cod_Dept	Nombre_d	nss_gestor
Charo		1	Ventas	1
María José	...	1	Ventas	1
Alejandra	...	2	Compras	4
Pedro	...	2	Compras	4
Martina	...	2	Compras	4

# Reducción de valores redundantes en tuplas

- Anomalías de inserción
  - Primer caso: todo nuevo empleado debe tener todos los datos de su departamento (o nulos si no tiene departamento asignado)
    - Eso implica que para insertar un empleado hay que hacer una búsqueda previa de todos los atributos del departamento (problema adicional de incremento del coste temporal)

Nombre_e	...	Cod_Dept	Nombre_d	nss_gestor
Charo		1	Ventas	1
María José	...	1	Ventas	1
Alejandra	...	2	Compras	4
Pedro	...	2	Compras	4
Martina	...	2	Compras	4

# Reducción de valores redundantes en tuplas

- Anomalías de inserción
  - Segundo caso: para dar de alta un departamento sin empleados asignados es complicado
    - Violaría la clave primaria (al dejarla a nulo)
    - Y aunque modificara la clave para permitirlo ...

Nombre_e	...	Cod_Dept	Nombre_d	nss_gestor
Charo		1	Ventas	1
María José	...	1	Ventas	1
Alejandra	...	2	Compras	4
Pedro	...	2	Compras	4
Martina	...	2	Compras	4

# Reducción de valores redundantes en tuplas

- Anomalías de inserción
  - Segundo caso: para dar de alta un departamento sin empleados asignados es complicado
    - Violaría la clave primaria (al dejarla a nulo)
    - Y aunque modificara la clave para permitirlo, al hacerlo violaría la semántica, porque habría tuplas que no representarían empleados

Nombre_e	...	Cod_Dept	Nombre_d	nss_gestor
Charo		1	Ventas	1
María José	...	1	Ventas	1
Alejandra	...	2	Compras	4
Pedro	...	2	Compras	4
Martina	...	2	Compras	4



# Reducción de valores redundantes en tuplas

- Anomalía de borrado
  - Relacionada con la segunda anomalía de inserción: si se borran todos los empleados de un departamento ...

Nombre_e	...	Cod_Dept	Nombre_d	nss_gestor
Charo		1	Ventas	1
María José	...	1	Ventas	1
Alejandra	...	2	Compras	4
Pedro	...	2	Compras	4
Martina	...	2	Compras	4

# Reducción de valores redundantes en tuplas

- Anomalía de borrado
  - Relacionada con la segunda anomalía de inserción: si se borran todos los empleados de un departamento se pierde el departamento

Nombre_e	...	Cod_Dept	Nombre_d	nss_gestor
Charo		1	Ventas	1
María José	...	1	Ventas	1
Alejandra	...	2	Compras	4
Pedro	...	2	Compras	4
Martina	...	2	Compras	4

# Reducción de valores redundantes en tuplas

- Anomalía de borrado
  - Relacionada con la segunda anomalía de inserción: si se borran todos los empleados de un departamento se pierde el departamento
    - Tendría que comprobar si el borrado incluye a todos los empleados y dar de alta uno con nulos ...

Nombre_e	...	Cod_Dept	Nombre_d	nss_gestor
Charo		1	Ventas	1
María José	...	1	Ventas	1
Alejandra	...	2	Compras	4
Pedro	...	2	Compras	4
Martina	...	2	Compras	4

# Reducción de valores redundantes en tuplas

- Anomalía de modificación
  - Modificar los datos de un departamento implica modificar ...

Nombre_e	...	Cod_Dept	Nombre_d	nss_gestor
Charo		1	Ventas	1
María José	...	1	Ventas	1
Alejandra	...	2	Compras	4
Pedro	...	2	Compras	4
Martina	...	2	Compras	4

# Reducción de valores redundantes en tuplas

- Anomalía de modificación
  - Modificar los datos de un departamento implica modificar las tuplas de todos sus empleados
    - Esto puede suponer un sobrecoste considerable de rendimiento

Nombre_e	...	Cod_Dept	Nombre_d	nss_gestor
Charo		1	Ventas	1
María José	...	1	Ventas	1
Alejandra	...	2	Compras	4
Pedro	...	2	Compras	4
Martina	...	2	Compras	4

# Reducción de valores redundantes en tuplas

- Guía 2 para el diseño:
  - Un buen diseño debe evitar anomalías de actualización
  - Lo que no quiere decir que no vayan a producir (a veces es necesario por rendimiento)
    - Pero en ese caso hay que documentarlo adecuadamente, justificando la necesidad y manteniendo coherencia en todos los programas que realizan actualizaciones sobre los datos
    - ¿Solución alternativa?

# Reducción de valores redundantes en tuplas

- Guía 2 para el diseño:
  - Un buen diseño debe evitar anomalías de actualización
  - Lo que no quiere decir que no vayan a producir (a veces es necesario por rendimiento)
    - Pero en ese caso hay que documentarlo adecuadamente, justificando la necesidad y manteniendo coherencia en todos los programas que realizan actualizaciones sobre los datos
    - Una alternativa, si el SGBD lo permite, es definir relaciones bases sin anomalías y crear vistas que almacenen los productos naturales más comunes precalculados

# Reducción de valores redundantes en tuplas

- Las dos primeras guías están interrelacionadas
- Serán formalizadas más adelante mediante las dependencias funcionales
  - Que serán parte posteriormente del proceso de normalización



# Reducción de valores nulos en tuplas

- Algunas relaciones puede tener un número muy alto de atributos. Si hay muchos atributos sin valor en bastantes tuplas
  - ¿Qué problema puede haber?

# Reducción de valores nulos en tuplas

- Algunas relaciones puede tener un número muy alto de atributos. Si hay muchos atributos sin valor en bastantes tuplas
  - Se pierde mucho espacio
  - Se puede “complicar” la semántica de cada atributo
  - Hay que tener cuidado con los comportamiento de los joins (*inner* versus *outer*)
  - Se complica la agregación con *SUM*, ...
  - Además, un nulo puede ser “no aplica”, “existe pero con valor desconocido”, “no se sabe si existe”, ...

# Reducción de valores nulos en tuplas

- Guía 3 para el diseño:
  - En la medida de lo posible, evitar atributos que frecuentemente tengan valores nulos
  - En los casos en los que los nulos sean necesarios: confirmar que no se dan en la mayoría de tuplas de la relación

# Reducción de valores nulos en tuplas

- Guía 3 para el diseño:
  - En la medida de lo posible, evitar atributos que frecuentemente tengan valores nulos
  - En los casos en los que los nulos sean necesarios: confirmar que no se dan en la mayoría de tuplas de la relación
    - Ej: si hay pocos empleados con móvil no es buena idea añadirlo como atributo a Empleado. Mejor ...

# Reducción de valores nulos en tuplas

- Guía 3 para el diseño:
  - En la medida de lo posible, evitar atributos que frecuentemente tengan valores nulos
  - En los casos en los que los nulos sean necesarios: confirmar que no se dan en la mayoría de tuplas de la relación
    - Ej: si hay pocos empleados con móvil no es buena idea añadirlo como atributo a Empleado. Mejor crear una relación propia Empleado\_movil(nss, movil)

# Evitar la generación de tuplas espurias

- Cambiando el ejemplo mejorable de “Reducción de valores redundantes en tuplas”:
  - Pasamos de tener (que ya vimos que era mejorable):
    - Trabaja\_proy(nss, cod\_proy, horas, nombre\_e, nombre\_p, localiz\_p)
  - A tener
    - Empleado\_locs(nombre\_e, localiz\_p)
      - Las tuplas indican que el empleado que se llama *nombre\_e* trabaja en un proyecto que está en *localiz\_p*
    - Trabaja\_proy2(nss, cod\_proy, horas, nombre\_p, localiz\_p)
      - Las tuplas indican que el empleado con número de seguridad social *nss* trabaja *horas* horas semanales en un proyecto con código *cod\_proy*, denominado *nombre\_p* y localizado en *localiz\_p*

# Evitar la generación de tuplas espurias

- Este cambio tiene un problema grave, y es que puede no recuperarse la información que almacenaba inicialmente
- El producto natural de las relaciones nuevas Empleado\_locs y Trabaja\_proy2 da más resultados que la relación original Trabaja\_proy:
  - Trabaja\_proy(nss, cod\_proy, horas, nombre\_e, nombre\_p, localiz\_p)

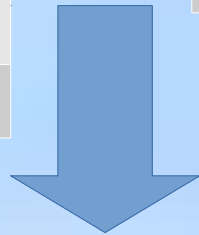
*versus*

  - Empleado\_locs(nombre\_e, localiz\_p)
  - Trabaja\_proy2(nss, cod\_proy, horas, nombre\_p, localiz\_p)

# Evitar la generación de tuplas espurias

- Ejemplo: Empleado\_locs  $\bowtie$  Trabaja\_proy2

Nombre_e	Localiz_p
Charo	Cádiz
María José	Cádiz
Alejandra	Jerez
Pedro	Jerez
Martina	Jerez



nss	Cod_proy	horas	nombre_p	localiz_p
888	P12	10	Archivalia	Cádiz
777	P30	33	Montaña	Cádiz
456	P22	8	Risoto	Jerez

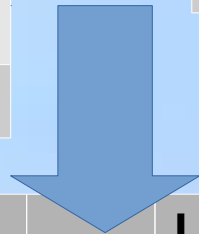


# Evitar la generación de tuplas espurias

- Ejemplo: Empleado\_locs  $\bowtie$  Trabaja\_proy2

Nombre_e	Localiz_p
Charo	Cádiz
María José	Cádiz
Alejandra	Jerez
Pedro	Jerez
Martina	Jerez

nss	Cod_proy	horas	nombre_p	localiz_p
888	P12	10	Archivalia	Cádiz
777	P30	33	Montaña	Cádiz
456	P22	8	Risoto	Jerez



Nombre_e	...	Localiz_p	Nombre_d
Charo		Cádiz	Archivalia
Charo	...	Cádiz	Montaña
María José	...	Cádiz	Archivalia
María José		Cádiz	Montaña
...	...	...	...
Martina	...	Jerez	Risoto

# Evitar la generación de tuplas espurias

- ¿Cuál es el origen del problema?
  - Trabaja\_proy(nss, cod\_proy, horas, nombre\_e, nombre\_p, localiz\_p)
  - versus*
  - Empleado\_locs(nombre\_e, localiz\_p)
  - Trabaja\_proy2(nss, cod\_proy, horas, nombre\_p, localiz\_p)

# Evitar la generación de tuplas espurias

- ¿Cuál es el origen del problema?
  - Trabaja\_proy(nss, cod\_proy, horas, nombre\_e, nombre\_p, localiz\_p)
  - versus*
  - Empleado\_locs(nombre\_e, localiz\_p)
  - Trabaja\_proy2(nss, cod\_proy, horas, nombre\_p, localiz\_p)
- Que se ha usado para hacer el producto un atributo que no es clave primaria (ni foránea – ni UNIQUE) en las relaciones nuevas

# Evitar la generación de tuplas espurias

- Guía 4 para el diseño:
  - Los esquemas de una relación debe permitir unirse con JOIN usando igualdad de atributos que sean claves primarias o foráneas, de modo que no se generen tuplas espurias
  - Lo ideal es evitar tablas que tengan atributos comunes que no sean combinación del claves primarias-foráneas
    - Si no hay más remedio que tenerlos, no usarlos para operaciones JOIN
  - Más adelante se formaliza esta guía

# Guías generales de diseño

- Resumen:
  - Se han mostrado cuatro diseños que pueden llevar a situaciones problemáticas en el manejo de datos
    - No tienen que dar problemas siempre necesariamente, por son “sólo” un mal diseño (situaciones evitables)
    - Sus problemas se pueden mitigar con una implementación cuidadosa
    - Pero son problemas, pueden estar latentes durante mucho tiempo y “salir” el día menos esperado
  - A continuación se formalizan estas guías, partiendo del concepto de dependencia funcional

# Dependencias funcionales

- La dependencia funcional (DF) es un concepto clave en el diseño relacional
  - Las DF muestran interrelaciones entre atributos
  - Son invariantes en el tiempo, mientras no cambie el problema del mundo a modelar
    - Igual que los esquemas de las relaciones, ...
  - Forman unas restricciones al conjunto de relaciones
- La teoría de la normalización de BBDD se basa en las DF

# Conceptos básicos

- Definición formal de clave:
  - Sea  $R$  un esquema de relaciones. Un subconjunto  $K$  de  $R$  es una clave de  $R$  si, en cualquier relación  $r(R)$  que se pueda definir, para todos los pares  $t_1$  y  $t_2$  de tuplas en  $r$  se cumple que

$$t_1 \neq t_2$$

$$t_1[K] \neq t_2[K]$$

- Nótese que se habla de “en cualquier relación  $r(R)$ ”, porque las DF relacionan conjuntos de atributos, independientemente de que vayan en la misma relación o no
  - Para analizar las claves (y las DF) puede considerarse que todos los atributos están en una misma (única) relación  $r(R)$


# Conceptos básicos

- Definición formal de DF:
  - La noción de DF generaliza a la de clave. Sea  $\alpha \subseteq R$  y  $\beta \subseteq R$ . La DF  $\alpha \rightarrow \beta$  se cumple en  $R$  si en cualquier relación  $r(R)$  que se pueda definir para todos los pares de tuplas  $t_1$  y  $t_2$  en  $r$  se cumpla

$$t_1[\alpha] = t_2[\alpha]$$

también se cumple que

$$t_1[\beta] = t_2[\beta]$$

- Se habla de que:
  - Los valores de  $\alpha$  determinan funcionalmente los de  $\beta$
  - Los valores de  $\beta$  están determinados por los de  $\alpha$
  - Hay una dependencia funcional de  $\alpha$  a  $\beta$
  - $\beta$  determina funcionalmente a  $\alpha$  



# Conceptos básicos

- Definición de DF entre
  - Dada una relación  $R$ , se dice que el atributo  $R.y \in R$  es funcionalmente dependiente de otro atributo  $R.x \in R$ , y se expresa de la forma  $R.x \rightarrow R.y$ , si, y sólo si cada valor de  $R.x$  tiene asociado a él exactamente un valor de  $R.y$  para cualquier extensión de la relación  $R$
- Si además se tiene que sólo puede haber una tupla con ese valor de  $R.x$  en toda  $r(R)$ , entonces  $R.x$  es una clave candidata de  $R$

# Conceptos básicos

- La DF  $X \rightarrow Y$  especifica una restricción en R e indica que  $\forall t, u \in R$ , se cumple que:

$$t[X] = u[X] \Rightarrow t[Y] = u[Y]$$

- Si X es una clave candidata, se cumple que  $X \rightarrow Y$ , para cualquier Y
- $X \rightarrow Y$  no implica que  $Y \rightarrow X$
- Las relaciones  $r(R)$  que cumplen las DF indicadas como restricciones en el mundo a modelar se denominan *extensiones legales*

# Conceptos básicos

- La DF se pueden representar de diferentes maneras.

- La forma más usada es:

$$R.x \rightarrow R.y$$

- Otra forma es:

$$R.y \text{ DF } R.x$$

- Y otra forma es:

$$R.y \text{ d.f. } R.x$$

# Conceptos básicos

- Una DF es una propiedad de la semántica de los atributos determinada por el mundo a modelar
- El concepto de DF tiene en cuenta atributos de una misma relación y no de relaciones diferentes
- La DF es independiente del estado de la relación
- Los atributos funcionalmente dependientes pueden ser simples o compuestos y pueden formar parte de la clave primaria o de alguna clave candidata

# Conceptos básicos

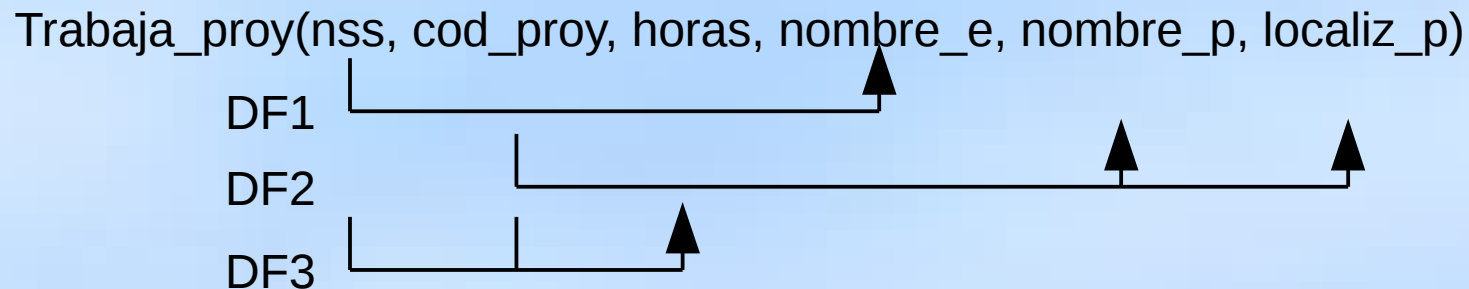
- Del ejemplo anterior:
  - Trabaja\_proy(nss, cod\_proy, horas, nombre\_e, nombre\_p, localiz\_p)
- ¿Qué tres DF podemos afirmar?

# Conceptos básicos

- Del ejemplo anterior:
  - Trabaja\_proy(nss, cod\_proy, horas, nombre\_e, nombre\_p, localiz\_p)
- ¿Qué tres DF podemos afirmar?
  - nss → nombre\_e
  - cod\_proy → {nombre\_p, localiz\_p}
  - {nss, cod\_proy} → horas
- Recordatorio: estas afirmaciones dependen del mundo a modelar, del contexto

# Conceptos básicos

- Del ejemplo anterior:
  - Trabaja\_proy(nss, cod\_proy, horas, nombre\_e, nombre\_p, localiz\_p)
- ¿Qué tres DF podemos afirmar?



- Notación (gráfica) alternativa

# Propiedades

- Dada una relación  $r(R)$  con el conjunto de atributos  $(a, b, c, d)$  se cumplen las siguientes propiedades:
  - *Reflexiva*: si  $R.b \subseteq R.a$  entonces en la relación existe la DF  $R.a \rightarrow R.b$ 
    - Esta propiedad recibe el nombre de DF trivial
    - Cualquier DF de la forma  $\alpha \rightarrow \beta$  es trivial si  $\beta \subseteq \alpha$
  - *Aumentativa*: si  $a \rightarrow b$  entonces
$$(a + c) \rightarrow (b + c)$$
  - *Transitiva*: si  $a \rightarrow b$  y  $b \rightarrow c$ , entonces  $a \rightarrow c$



# Propiedades

- Las propiedades reflexiva, aumentativa y transitiva se denominan *Axiomas de Armstrong*
- Existen otras más:
  - *Unión*: si  $a \rightarrow b$  y  $a \rightarrow c$  entonces
    - $a \rightarrow (b + c)$
  - *Pseudo-transitiva*: si  $a \rightarrow b$  y  $(b + c) \rightarrow d$ , entonces
    - $(a + c) \rightarrow d$
  - *Descomposición*: si  $a \rightarrow bc$  entonces
    - $a \rightarrow b$  y  $a \rightarrow c$
- Estas propiedades se demuestran formalmente

# Clausura

- La clausura de un conjunto de dependencias  $F$  se denota  $F^+$ , y es el conjunto de dependencias de todos atributos de cada dependencia funcional (aplicando las propiedades anteriores)
- Ejemplo: Sea  $F$  el conjunto de DF anterior:  
 $\{ nss \rightarrow nombre\_e,$   
 $cod\_proy \rightarrow \{nombre\_p, localiz\_p\},$   
 $\{nss, cod\_proy\} \rightarrow horas \}$

# Clausura

- Ejemplo: Sea  $F$  el conjunto de DF anterior:

$\{ nss \rightarrow nombre\_e,$   
 $cod\_proy \rightarrow \{nombre\_p, localiz\_p\},$   
 $\{nss, cod\_proy\} \rightarrow horas \}$

- Su clausura  $F^+$  es:

$\{ nss \}^+ \rightarrow \{nss, nombre\_e\}$   
 $\{ cod\_proy \}^+ \rightarrow \{cod\_proy, nombre\_p, localiz\_p\},$   
 $\{nss, cod\_proy \}^+ \rightarrow \{nss,$   
 $nombre\_e, horas, cod\_proy, nombre\_p, localiz\_p\}$

# Conjunto mínimo de DF

- Es una forma canónica (o estándar) de expresar un conjunto de DF sin redundancias
- Un conjunto de DF es mínimo si:
  - Tiene atributos simples a la derecha de todas DF
  - Para toda DF  $X \rightarrow A$  no existe otra DF  $Y \rightarrow A$  tal que  $Y$  sea un subconjunto estricto de  $X$  y se mantenga las mismas DF
  - No se pueden eliminar DF de  $F$  manteniendo las DF equivalentes
- Una cobertura mínima de  $F$  es el conjunto mínimo de DF de  $F$  (puede no ser único)

# Proceso de normalización

- En la década de los 60, los datos se manejaban en ficheros diseñados y gestionados por programas. Se realizan estudios de rendimiento en dos aspectos:
  - Las aplicaciones que gestionan los datos
  - La evolución de los requisitos
- Se llega a las mismas conclusiones:
  - Nuevas aplicaciones requerían sus propias estructuras de datos
  - La información tendía a duplicarse
  - Actualizaciones difíciles de realizar
  - Rendimiento insuficiente

# Proceso de normalización

- Se cambia el concepto de almacenamiento de datos pasando de ficheros tradicionales a almacenes de datos
- Se agrupan los datos en relaciones generando una estructura óptima para el tratamiento de los datos
- Concepto introducido por Codd pensado inicialmente para aplicarse a sistemas relacionales
  - Hoy en día se aplica a cualquier sistema de información
- Se basa en que los datos son independientes de las aplicaciones que los gestionan. Por lo que se generan relaciones tales que tengan los atributos necesarios para representar a la entidad o a la relación entre ellas

# Proceso de normalización

- Ventajas:
  - *Facilidad de uso*: datos agrupados en tablas que identifican a un objeto o relación
  - *Flexibilidad*: posibilidad de relacionar información procedente de diferentes tablas utilizando álgebra o cálculo relacional
  - *Precisión*: la interrelación entre tablas consigue información exacta
  - *Seguridad*: es fácil mantener controles de acceso a los datos
  - *Facilidad de implementación*: las tablas se almacenan como ficheros planos

# Proceso de normalización

- *Independencia de datos*: las estructuras y almacenamiento de la información es independiente de las aplicaciones
- *Claridad*: la información se representa de manera sencilla
- *Facilidad de gestión*: los lenguajes manejan la información de manera sencilla al estar basados en el álgebra y cálculo relacional
- *Mínima redundancia*: permite que no exista más redundancia que la necesaria para la interrelación de tablas
- *Máximo rendimiento*: las aplicaciones sólo utilizan la información necesaria



# Proceso de normalización

- Codd propuso el proceso de normalización como un proceso para comprobar que un esquema de BD cumplía una serie de requisitos, estando en una “forma normal”
- Es un proceso *top-down* que descompone sucesivamente una relación en otras
- Codd presentó inicialmente tres formas normales, basadas en el concepto de DF
  - Posteriormente propuso junto a Boyce otra más
- Posteriormente se definieron dos más complejas (cuarta y quinta)

# Proceso de normalización

- Básicamente la normalización asegura que:
  - Existe una redundancia mínima en el esquema
  - Las anomalías de actualización se minimizan
- Cuando se habla de la forma normal en que está un esquema, se nombra la más alta
- Normalmente, se desea una forma normal lo más alta posible
  - Pero por requisitos no funcionales (eficiencia, fiabilidad, etc) puede ser deseable mantener una forma normal inferior

# Formas normales de Codd

- *Primera Forma Normal*: una relación está en 1FN, si cumple la propiedad de que sus dominios no tengan elementos que su vez sean conjuntos
  - Básicamente consiste en no admitir atributos multivaluados, compuestos o combinaciones de ambos
  - Realmente estos violan el concepto de “relación” que definimos por el dominio de sus atributos
  - “*Una relación que se puede almacenar en un fichero plano está en 1FN*”

# Formas normales de Codd

- Existen tres formas de normalizar un esquema con un multivaluado para que cumpla FN1:
  - Eliminar los atributos de la relación y ponerlos en una relación aparte que tenga la misma clave que la original y una tupla por cada valor del atributo repetido
  - Poner en la relación una tupla por cada valor del atributo repetido
    - Puede ser necesario ampliar la clave primaria
    - En sucesivas FN habrá que sacarla de la tabla de todas formas
  - Si se conoce el número máximo de ocurrencias, poner N atributos ... pero podemos tener muchos nulos :(

# Formas normales de Codd

- Existen tres formas de normalizar un esquema con un multivaluado para que cumpla FN1
- Ejemplo: Persona(DNle, nombre, ..., móviles)
  - Descomponer en  
Persona(DNle, nombre, ...)  
Móviles(DNle, móvil)
  - Cambiar el nombre (semántica) del atributo:  
Persona(DNle, nombre, ..., móvil)
  - Poner máximo tres móviles: Persona(DNle, nombre, ..., móvil1, móvil2, móvil3)

# Formas normales de Codd

- Para normalizar un esquema con un atributo compuesto para que cumpla FN1:
  - Se “aplanan” sus sub-atributos

# Formas normales de Codd

- Para normalizar un esquema con un atributo compuesto para que cumpla FN1:
  - Se “aplanan” sus sub-atributos
  - Ejemplo:  
Persona(DNle, nombre, ..., dirección(calle, ciudad, provincia))
  - Pasa a  
Persona(DNle, nombre, ..., calle, ciudad, provincia):

# Formas normales de Codd

- Para normalizar un esquema con un atributo compuesto multivaluado para que cumpla FN1:
  - Realmente el atributo es una relación en sí misma, por lo que tiene su “clave interna”
  - El proceso es eliminar el atributo de la relación y ponerlo en una relación aparte cuya clave sea la clave de la original más la “clave interna”



# Formas normales de Codd

- Para normalizar un esquema con un atributo compuesto multivaluado para que cumpla FN1:
- Ejemplo: si una persona puede tener varias viviendas
  - Persona(DNIe, nombre, ..., direcciones(calle, ciudad, provincia))
- Descomponer en:
  - Persona(DNIe, nombre, ...)
  - Dirección(DNIe, calle, ciudad, provincia)
    - CP: porque la provincia no es necesaria dada la ciudad

# Formas normales de Codd

- *Segunda Forma Normal*: una relación está en 2FN, si además de estar en 1FN, todos los atributos no primarios tienen DF plena con la clave primaria de la relación
  - Un atributo no primario es aquel que no es miembro de ninguna clave candidata
  - Una DF  $X \rightarrow Y$  es plena si al eliminar cualquier atributo de  $X$  dejar de mantenerse  $X \rightarrow Y$ 
    - En otro caso es una DF parcial
  - Caso particular: si la Clave Primaria de  $R$  es un atributo atómico,  $R$  está en FN2

# Formas normales de Codd

- Para normalizar un esquema a FN2:
  - Descomponer la relación en tantas relaciones como sea necesario, de forma que en cada una de ellas los atributos no primarios tengan DF plena
  - “Evitar una CP innecesariamente grande para algunos atributos”

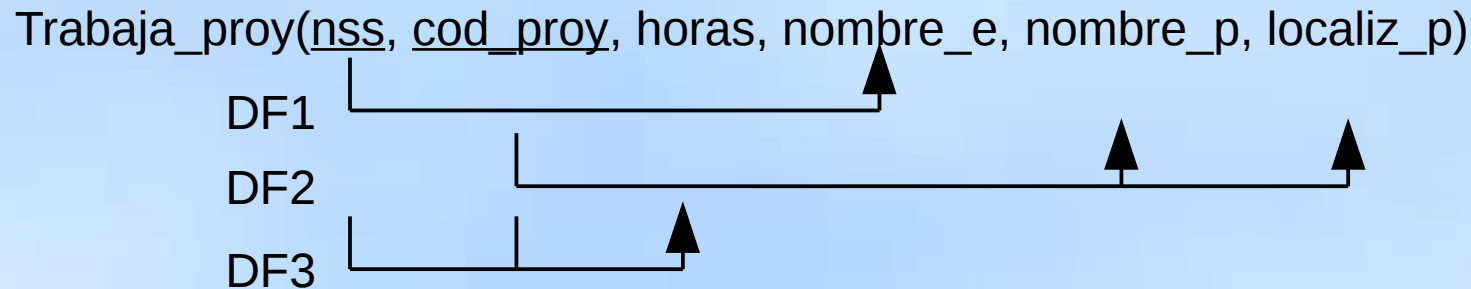
# Formas normales de Codd

- Ejemplo de pasar a FN2:
  - Trabaja\_proy(nss, cod\_proy, horas, nombre\_e, nombre\_p, localiz\_p) está en FN1 pero no en FN2 porque tiene las siguientes DF:
    - $nss \rightarrow nombre\_e$
    - $cod\_proy \rightarrow \{nombre\_p, localiz\_p\}$
    - $\{nss, cod\_proy\} \rightarrow horas$
  - Normalización:
    - Rel1(nss, nombre\_e)
    - Rel2(cod\_proy, nombre\_p, localiz\_p)
    - Rel3(nss, cod\_proy, horas)

# Formas normales de Codd

- Ejemplo de pasar a FN2:

- A veces se ve más fácil gráficamente:



- FN2 si “no salen flechas de parte de la clave primaria”
- Normalización:
  - `Rel1(nss, nombre_e)`
  - `Rel2(cod_proy, nombre_p, localiz_p)`
  - `Rel3(nss, cod_proy, horas)`

# Formas normales de Codd

- *Tercera Forma Normal*: una relación está en 3FN, si además de estar en 2FN, ninguno de sus atributos no primarios tiene dependencias transitivas respecto de la clave primaria
  - $X \rightarrow Y$  es una dependencia transitiva en R si existe un Z tal que  $X \rightarrow Z$  y  $Z \rightarrow Y$ 
    - Z no puede ser una clave candidata ni un subconjunto de una clave de R

# Formas normales de Codd

- Para normalizar un esquema a FN3:
  - Descomponer la relación en tantas relaciones como sea necesario, de forma que en cada una de ellas estén los atributos no primarios que soportan la DF transitiva
    - En esas nuevas relaciones las claves primarias serán los atributos “origen” de las dependencias

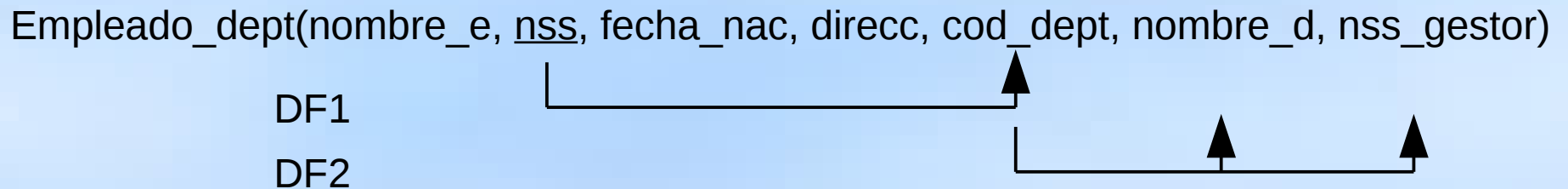
# Formas normales de Codd

- Ejemplo de pasar a FN3:
  - Empleado\_dept(nombre\_e, nss, fecha\_nac, direcc, cod\_dept, nombre\_d, nss\_gestor)
    - Está en FN2 pero no en FN3 porque tiene la siguiente DF transitiva:
      - $nss \rightarrow \{ nombre\_d, nss\_gestor \}$  mediante cod\_dept
      - Osea:  $nss \rightarrow cod\_dept$  y  $cod\_dept \rightarrow \{ nombre\_d, nss\_gestor \}$
  - Normalización
    - Empleado\_dept(nombre\_e, nss, fecha\_nac, direcc, cod\_dept)
    - Rel2(cod\_dept, nombre\_d, nss\_gestor)



# Formas normales de Codd

- Ejemplo de pasar a FN3:
  - Empleado\_dept(nombre\_e, nss, fecha\_nac, direcc, cod\_dept, nombre\_d, nss\_gestor)
  - Está en FN2 pero no en FN3 porque tiene la siguiente DF transitiva:



- Normalización
  - Empleado\_dept(nombre\_e, nss, fecha\_nac, direcc, cod\_dept)
  - Rel2(cod\_dept, nombre\_d, nss\_gestor)

# Formas normales de Codd

- Formalmente, una relación está en
  - *Primera Forma Normal*: si sólo tiene atributos atómicos indivisibles
  - *Segunda Forma Normal*: si todo atributo no primo tiene DF plena de toda clave de R
    - O si todo atributo no primo de A de R no tiene DF parcial de ninguna clave de R
  - *Tercera Forma Normal*: si toda DF no trivial  $X \rightarrow A$  cumple que X es superclave de R o A es un atributo primario
    - Es decir, si todo atributo no primario tiene DF plena de toda clave de R y no tiene DF transitiva de toda clave de R

# Formas normal de Boyce-Codd

- *Forma Normal de Boyce-Codd*: una relación está en FNBC, si además de estar en 3FN, para toda DF no trivial  $X \rightarrow A$ ,  $X$  es superclave de  $R$ 
  - Idéntica a FN3, pero se elimina la exigencia de que  $A$  sea un atributo primario
- Se definió inicialmente como una forma normal menos estricta que FN3, pero posteriormente demostró ser más estricta que esta
- *Toda relación con uno o dos atributos, está en FNBC*

# Formas normales de Codd

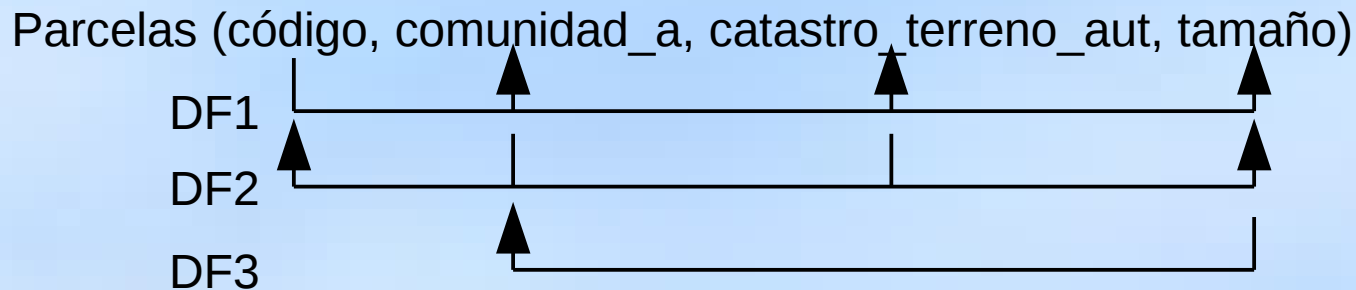
- Para normalizar un esquema a FNBC:
  - Descomponer la relación en tantas relaciones como sea necesario, de forma que en cada una de ellas estén los atributos que soportan la DF
    - En esas nuevas relaciones las claves primarias serán los atributos que causan la dependencia

# Formas normales de Codd

- Ejemplo de pasar a FNBC:
  - Sea una base de datos de parcelas rústicas de Andalucía y Extremadura
    - La Junta de Andalucía sólo permite tener parcelas de 1, 2 o 3 hectáreas
    - La Junta de Extremadura sólo permite tener parcelas de 10 hectáreas
  - Parcelas (código, comunidad\_a, catastro\_terreno\_aut, tamaño) está en FN3 porque se cumple:
    - {código} → comunidad\_a, catastro\_terreno\_aut, tamaño
    - {comunidad\_a, catastro\_terreno\_aut} → {código, tamaño}
  - Pero no en FNBC porque tiene la siguiente DF:
    - tamaño → {comunidad\_a}
- Normalización
  - Parcelas (código, catastro\_terreno\_aut, tamaño)
  - Comunidades\_a (tamaño, comunidad\_a)

# Formas normales de Codd

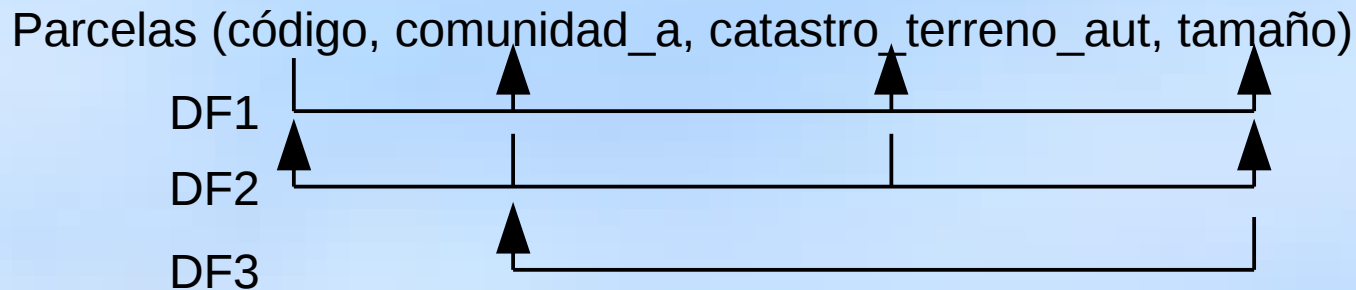
- Ejemplo de pasar a FNBC:
  - Sea una base de datos de parcelas rústicas de Andalucía y Extremadura
    - La Junta de Andalucía sólo permite tener parcelas de 1, 2 o 3 hectáreas
    - La Junta de Extremadura sólo permite tener parcelas de 10 hectáreas



- Parcelas (código, comunidad\_a, catastro\_terreno\_aut, tamaño) está en FN3 porque se cumple DF1 y DF2, pero no FNBC por DF3

# Formas normales de Codd

- Ejemplo de pasar a FNBC:
  - Sea una base de datos de parcelas rústicas de Andalucía y Extremadura
    - La Junta de Andalucía sólo permite tener parcelas de 1, 2 o 3 hectáreas
    - La Junta de Extremadura sólo permite tener parcelas de 10 hectáreas
  - Parcelas (código, comunidad\_a, catastro\_terreno\_aut, tamaño) está en FN3 porque se cumple DF1 y DF2, pero no FNBC por DF3



- Normalización
  - Parcelas (código, catastro\_terreno\_aut, tamaño)
  - Comunidades\_a (tamaño, comunidad\_a)

# Formas normales de Codd

- ¿Es un problema real que una relación no está en FNBC?

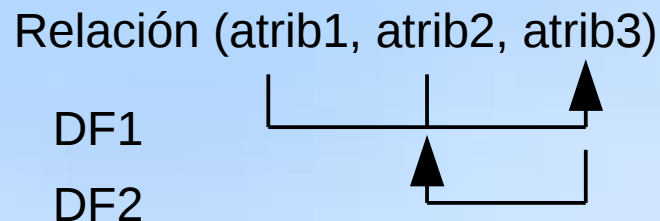


# Formas normales de Codd

- ¿Es un problema real que una relación no está en FNBC?
- Sí, porque puede presentar los problemas vistos por los valores redundantes en tuplas
  - En el ejemplo anterior, si tengo en la BD información de 1000 terrenos:
    - Terrenos (código, comunidad\_a, catastro\_terreno\_aut, tamaño)
    - Tendría como mínimo 1000-4 valores repetidos de comunidad\_a

# Formas normales de Codd

- La mayoría de relaciones que están en FN3 suelen estar en FNBC
- En general, las que no cumplen FNBC suelen tener la forma:



- Si hay otras DF y no se usa {atrib1, atrib2} como CP puede costar más darse cuenta

# Formas normales de Codd

- A veces la normalización a FNBC no es única
  - Ejemplo: sea una universidad donde los profesores son expertos que sólo imparten una única asignatura
    - Estudia(alumno, asignatura, profesor)
  - DF:
    - {alumno, asignatura} → profesor
    - profesor → asignatura
  - Tres descomposiciones posibles:
    - a)
    - b)
    - c)

# Formas normales de Codd

- A veces la normalización a FNBC no es única
  - Ejemplo: sea una universidad donde los profesores son expertos que sólo imparten una única asignatura
    - Estudia(alumno, asignatura, profesor)
  - DF:
    - {alumno, asignatura} → profesor
    - profesor → asignatura
  - Tres descomposiciones posibles: ¿cuál es mejor?
    - a) Estudia(alumno, profesor), rel2(alumno, asignatura)
    - b) Estudia(profesor, asignatura), rel2(alumno, asignatura)
    - c) Estudia(profesor, asignatura), rel2(profesor, alumno)

# Formas normales de Codd

- A veces la normalización a FNBC no es única
  - Tres descomposiciones posibles: ¿cuál es mejor?
    - a) Estudia(alumno, profesor), rel2(alumno, asignatura)
    - b) Estudia(profesor, asignatura), rel2(alumno, asignatura)
    - c) Estudia(profesor, asignatura), rel2(profesor, alumno)
  - La primera pierde la DF profesor → asignatura
    - No deseable
  - Todas pierden la DF {alumno, asignatura} → profesor
    - Pero la tercera es la única que no genera tuplas espurias tras un JOIN, porque:
      - Es una *descomposición no-aditiva* (se estudia en FN4 y FN5)

# Mapeo de (E)E-R a relaciones

- El paso de (E)E-R a relaciones puede ser guiado por un algoritmo:
  - Más directo para E-R (7 pasos)
  - Con varias opciones para EE-R
    - Esto es porque el EE-R monta estructuras más abstractas (cercanas al mundo real) que después tienen que implementarse con las mismas “herramientas”: relaciones y claves
- Como veremos al final, el algoritmo no nos va a evitar tener que pensar en la normalización

# Mapeo de E-R a relaciones

- Paso 1: por cada tipo de entidad fuerte E
  - Se crea una relación R con todos los atributos simples de E
  - Para los atributos compuestos incluir sólo sus componentes
  - La clave primaria es el conjunto de atributos indicado como clave
    - Si la clave primaria incluía al atributo compuesto, sus miembros formarán parte de ella

# Mapeo de E-R a relaciones

- Paso 2: por cada tipo de entidad débil W
  - Se crea una relación R con todos los atributos simples (y componentes de los compuestos) de W
  - Se incluyen en R los atributos de la clave primaria de la entidad fuerte de que depende
    - Estos atributos serán clave foránea de la relación que se defina en base a esa entidad fuerte
  - La clave primaria es la unión de la clave primaria de la entidad fuerte y la clave parcial de la débil (si la tiene)
  - *Suele usarse propagación (on CASCADE) para modificaciones y borrados*



# Mapeo de E-R a relaciones

- Paso 3: por cada tipo de relación 1:1 R, siendo S y T las relaciones que participan en ella
  - Incluir en una cualquiera de las relaciones (S) los atributos de la CP de la otra (T) como clave foránea
    - Si una de las dos entidades tiene participación total, coger esa como S (ej: todo Departamento tiene director)
  - Incorporar los atributos simples de la relación R en S (y los componentes de los compuestos)
  - *Una alternativa interesante para el caso de participación total de ambas entidades, es montar una única relación con R, S y T*

# Mapeo de E-R a relaciones

- Paso 4: por cada tipo de relación 1:N R, en la que participan S (con N) y T (con 1)
  - Incluir en S los atributos de la clave primaria de T como clave foránea
  - Incorporar los atributos simples de la relación R en S (y los componentes de los compuestos)
- Nota: para una relación de cualquier tipo, si es una relación reflexiva, la incorporación de la clave implica renombrar los atributos

# Mapeo de E-R a relaciones

- Paso 5: por cada tipo de relación M:N R
  - Crear una nueva relación S
  - Incluir en S los atributos de la claves primarias de las dos relaciones
    - Esos atributos conformarán la clave primaria de S
  - Incorporar los atributos simples de la relación R en S (y los componentes de los compuestos)
  - *Suele usarse propagación (on CASCADE) para actualizaciones y borrados*

# Mapeo de E-R a relaciones

- Paso 5: por cada tipo de relación M:N R
  - Una alternativa interesante para evitar exceso de nulos en relaciones 1:1 y 1:N es mapearlas como relaciones M:N
  - También puede tomarse como estrategia para relaciones que “creamos” que pueden evolucionar
    - Se parece un poco a la política de poner un auto\_incremento como clave a toda tabla: facilita el crecimiento
      - Las relaciones 1:1 y 1:N “caben” en el mapeo de una M:N
      - Pero la M:N no cabe en el mapeo de una 1:1 ni 1:N
      - Ni la 1:N cabe en el mapeo de una 1:1
    - Si evoluciona una entidad 1:1 a 1:N “cabe”, igual que si una 1:N evoluciona a M:N

# Mapeo de E-R a relaciones

- Paso 6: por cada atributo multivaluado A
  - Crear una nueva relación R
  - Incluir en R como atributos A y los atributos K de la clave primaria de entidad E a la que pertenecen
    - Esos atributos conformarán la clave primaria de R
  - Si el atributo es compuesto incorporar sus componentes

# Mapeo de E-R a relaciones

- Paso 7: por cada tipo de relación N-aria R (con  $N > 2$ )
  - Crear una nueva relación S
  - Incluir en S los atributos de las claves primarias de las N relaciones (como claves foráneas)
    - Esos atributos suelen conformar la clave primaria de S
  - Incorporar los atributos simples de la relación R en S (y los componentes de los compuestos)

# Mapeo de E-R a relaciones

- Resumen

Modelo E/R	Modelo Relacional
Tipo de entidad (fuerte o débil)	Relación “de entidad”
Tipo de relación 1:1 o 1:N	Clave foránea (o Relación “de relación”)
Tipo de relación M:N	Relación “de relación” con dos claves foráneas
Tipo de relación n-aria ( $n > 2$ )	Relación “de relación” con $n$ claves foráneas
Atributo simple	Atributo
Atributo compuesto	Serie de atributos
Atributo multivaluado	Relación con clave foránea
Conjunto de valores	Dominio
Atributo clave	Clave (primaria o candidata)

# Mapeo de EE-R a relaciones

- Mapeo de generalizaciones/especializaciones
  - Sería el paso 8 del algoritmos de mapeado
  - Existen varias alternativas (veremos 4), cada una con sus ventajas e inconvenientes
    - En una generalización multinivel se pueden combinar varias de las opciones de este paso
    - Una subclase compartida se mapea mejor como *categoría*
  - Sea:
    - C la superclase:
      - Con atributos  $\{k, a_1, \dots, a_n\}$
      - Siendo k su clave primaria
    - Las subclases:  $\{S_1, \dots, S_m\}$



# Mapeo de EE-R a relaciones

- Paso 8 (opción a):
  - Crear una relación  $L$  con los atributos de  $C$  y su clave primaria
  - Crear una relación  $L_i$  por cada subclase  $S_i$ 
    - Con los atributos de  $S_i \cup \{k\}$  y su clave primaria  $k$

# Mapeo de EE-R a relaciones

- Paso 8 (opción a):
  - El producto theta de las claves primarias de  $L$  y  $L_i$  da como resultado todos los atributos específicos y heredados de las entidades de  $S_i$
  - Exige que para todo  $L_i$  se cumpla la *dependencia de inclusión*  $\Pi_{\langle K \rangle}(L_i) \subseteq \Pi_{\langle K \rangle}(L)$ 
    - Al hablar de FN4 y FN5 se entrará en detalle
- Válido para toda generalización
  - Pero no quiere decir que sea lo más adecuado para todo caso

# Mapeo de EE-R a relaciones

- Paso 8 (opción b):
  - No se crea la relación L con los atributos de C
  - Crear una relación  $L_i$  por cada subclase  $S_i$ 
    - Con los atributos de  $S_i \cup \{k, a_1, \dots, a_n\}$  y su clave primaria k

# Mapeo de EE-R a relaciones

- Paso 8 (opción b):
  - El producto theta “se hace” al definir los esquemas de  $L_i$ , por lo que la relación  $L$  sobra
  - Funciona bien para disjunto y total
    - Si no es total no se pueden representar entidades que no pertenezcan a ninguna de las subclases
    - Si no es disjunta, una entidad que perteneciera a varias subclases tendría atributos redundantes de la superclase en cada una de ellas
  - “Se pierde” la superclase  $C$ , por lo que habría que recuperarla con una OUTER UNION (o FULL OUTER UNION) con los  $L_i$  (que daría como resultado lo que se propone directamente en 8.c y 8.d)
    - Para buscar un elemento en  $C$  hay que buscarlo en todo  $L_i$

# Mapeo de EE-R a relaciones

- Paso 8 (opción c):
  - Crear una única relación L con:
    - Los atributos de C  $\cup$  atributos de  $S_1 \cup \dots \cup$  atributos de  $S_m \cup t$ 
      - El atributo  $t$  es el atributo de tipo o discriminante
        - Si es una especialización definida por atributo, ese atributo hace las veces de  $t$  (y  $t$  sobra)
    - La clave primaria es k

# Mapeo de EE-R a relaciones

- Paso 8 (opción c):
  - Crear una única relación para la superclase y todas sus subclases con un atributo  $t$  discriminante
    - Si las subclases tiene muchos atributos puede aparecer una elevada cantidad de nulos
      - Pero si tienen pocos evita la necesidad de usar operadores de producto que había en 8.a y 8.b
  - Sólo válido para *disjoint*
  - Soporta una especialización parcial si el atributo  $t$  admite nulos

# Mapeo de EE-R a relaciones

- Paso 8 (opción d):
  - Crear una única relación L con:
    - Los atributos de C  $\cup$  atributos de  $S_1 \cup \dots$  atributos de  $S_m \cup \{k, t_1, \dots, t_m\}$ 
      - El atributo  $t_i$  es el atributo booleano que indica si la tupla pertenece al tipo  $S_i$  (*flag*)
    - La clave primaria es  $k$

# Mapeo de EE-R a relaciones

- Paso 8 (opción d):
  - Similar a 8.c pero con  $m$  atributos  $t_i$
  - Recomendado para overlapping
    - Pero también válido para disjoint cuidando que sólo un  $t_i$  sea TRUE
  - Soporta una especialización parcial si los atributos  $t_i$  admiten nulos



# Mapeo de EE-R a relaciones

- Mapeo de categorías (paso 9)
  - Una categoría es una subclase de varias superclases que no tienen la misma clave
    - Obliga a definir un nuevo atributo clave (subrogado)
    - Se crea una relación nueva con ese atributo subrogado y los atributos y todos los atributos de la categoría
    - Se añade el atributo subrogado a cada superclase de la categoría (haciendo de clave foránea para indicar la correspondencia entre los valores de la clave de cada superclase y la subrogada)
  - Si las superclase comparten clave se usa esa y no hay que crear la subrogada

# Mapeo de EE-R a relaciones

- ¿Qué forma normal nos asegura este algoritmo?
  - ¿FN1?
  - ¿FN2?
  - ¿FN3?
  - ¿FNBC?

# Mapeo de EE-R a relaciones

- ¿Qué forma normal nos asegura este algoritmo?
  - FN1: si sólo tiene atributos atómicos indivisibles
    -
  - FN2: si todo atributo no primo tiene DF plena de toda clave de R
    -
  - FN3: si toda DF no trivial  $X \rightarrow A$  cumple que X es superclave de R o A es un atributo primario
    -
  - FNBC: si toda DF no trivial  $X \rightarrow A$  cumple que X es superclave de R
    -

# Mapeo de EE-R a relaciones

- ¿Qué forma normal nos asegura este algoritmo?
  - FN1: si sólo tiene atributos atómicos indivisibles
    - Sí, porque elimina atributos compuestos y multivaluados
  - FN2: si todo atributo no primo tiene DF plena de toda clave de R
    - Si la clave que definida en el E/R no tiene atributos “de sobra”
  - FN3: si toda DF no trivial  $X \rightarrow A$  cumple que X es superclave de R o A es un atributo primario
    - No, por ejemplo si tengo Persona con DNle, provincia, CCAA
  - FNBC: si toda DF no trivial  $X \rightarrow A$  cumple que X es superclave de R
    - Tampoco

# Normalización de Codd

- Frente al esquema de diseño *top-down* que hemos usado, existe una alternativa *bottom-up*
  - Es más “purista”
  - Tiene la ventaja de que define un algoritmo de normalización para llegar a FN3
    - Este algoritmo parte de una única relación inicial “universal” con todos los atributos y las DF entre ellos
    - Descompone esa relación en N relaciones normalizadas
    - Asegura que no se pierde ninguna DF inicial
      - Y que se mantiene en relaciones simples, no implícitas en productos (que ya se ha visto que no permiten controlarlas)

# Normalización de Codd

- El algoritmo sin embargo tiene varios problemas:
  - No es cómodo especificar las DF de todos los atributos de una BD grande
  - Son algoritmos no deterministas (porque dependen de las coberturas mínimas)
    - Algunos resultados no cumplen FNBC
- Más allá de eso, el problema es que existen formas normales más estrictas que solucionan problemas que FNBC puede presentar

# Dependencias multivaluadas

- El concepto de Dependencia Funcional (DF) ha sido clave para mejorar relaciones
  - Ha permitido definir cuatro FN (FN1, FN2, FN3 y FNBC)
- Sin embargo, no todas las restricciones de determinadas relaciones se pueden expresar mediante DF
- Se define el concepto de Dependencia Multivaluada (DMV), que es base de la FN4

# Dependencias multivaluadas

- Las DMV las origina la FN1, que no permite múltiples valores en un atributo de una tupla
  - Esto, con varios atributos independientes en un mismo esquema de relación obliga a repetir los valores de cada atributo con los de los demás para tener una BD consistente
- La notación de una DMV de X sobre Y es

$$X \twoheadrightarrow Y$$



# Dependencias multivaluadas

- Formalmente  $X \twoheadrightarrow Y$  ( $X$  multidetermina a  $Y$ ), siendo  $X$  e  $Y$  subconjuntos de el esquema de relación  $R$ , si para todo estado  $r$  de  $R$ :
  - Cuando existen dos tuplas  $t_1$  y  $t_2$  tal que  $t_1[X]=t_2[X]$
  - Entonces existen otras dos tuplas  $t_3$  y  $t_4$  en  $r$  que cumplan que:

$$t_1[X]=t_2[X]=t_3[X]=t_4[X]$$

$$t_1[Y]=t_3[Y] \text{ y } t_2[Y]=t_4[Y]$$

$$t_1[Z]=t_4[Z] \text{ y } t_2[Z]=t_3[Z]$$

$$\text{siendo } Z = (R - (X \cup Y))$$

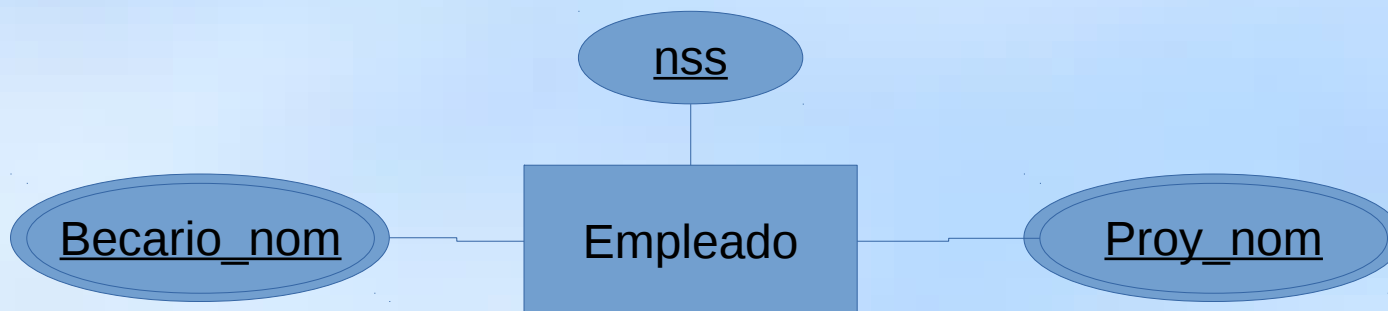
- Nota:  $t_1, t_2, t_3$  y  $t_4$  no tienen que ser distintas

# Dependencias multivaluadas

- Con palabras informales,  $X \twoheadrightarrow Y$  indica que
  - Si tengo un valor concreto de  $X$  los valores de  $Y$  los determinan ese valor de  $X$ , y no los valores del resto de atributos  $Z$
  - Por lo tanto, si hay dos tuplas con distintos valores de  $Y$  pero el mismo valor de  $X$ , los valores de  $Y$  se deben repetir en tuplas distintas con los valores de  $Z$  que se dan para ese mismo valor de  $X$
- Una  $X \twoheadrightarrow Y$  es trivial si se da una de las condiciones
  - $Y$  es un subconjunto de  $X$
  - $X \cup Y = R$

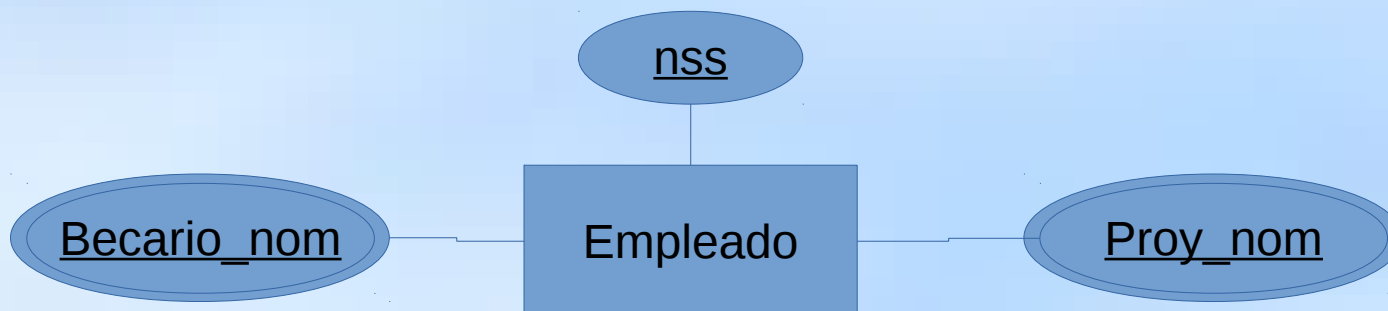
# Dependencias multivaluadas

- Ejemplo:
  - Relación Empleado(NSS, Proy\_nom, Becario\_nom)
    - Indica que el empleado con número de seguridad social *NSS* trabaja en el proyecto *Proy\_nom*, y que ese empleado supervisa al becario *Becario\_nom*
    - Pero ojo, los empleados no están asignados a proyectos (ni los proyectos a becarios)
      - Por lo tanto tenemos que  $NSS \twoheadrightarrow Proy\_nom \mid Becario\_nom$



# Dependencias multivaluadas

- Ejemplo:
  - Relación Empleado(NSS, Proy\_nom, Becario\_nom)
    - Tenemos que  $NSS \twoheadrightarrow Proy\_nom \mid Becario\_nom$
  - Si el empleado con NSS 1111 trabaja en dos proyectos (Diraya y Extenda) y tiene tres becarios (Juanmi, Luis y Manolo) ¿cómo se almacena?
    - Hay varias alternativas



# Dependencias multivaluadas

- Ejemplo: almacenamiento de
  - Empleado(NSS, Proy\_nom, Becario\_nom)
    - Primera opción: pierde la semántica

NSS	Proy_nom	Becario_nom
1111	Diraya	Juanmi
1111	Extenda	Luis
1111	<i>NULL</i>	Manolo

# Dependencias multivaluadas

- Ejemplo: almacenamiento de
  - Empleado(NSS, Proy\_nom, Becario\_nom)
    - Primera opción: pierde la semántica
    - Segundas: igual esquema pero redundancia “aleatoria”

NSS	Proy_nom	Becario_nom
1111	Diraya	Juanmi
1111	Extenda	Luis
1111	<i>NULL</i>	Manolo

...

NSS	Proy_nom	Becario_nom
1111	Extenda	Juanmi
1111	Extenda	Luis
1111	Diraya	Manolo

versus

NSS	Proy_nom	Becario_nom
1111	Extenda	Juanmi
1111	Diraya	Luis
1111	Diraya	Manolo

# Dependencias multivaluadas

- Ejemplo: almacenamiento de
  - Empleado(NSS, Proy\_nom, Becario\_nom)
    - Tercera: El almacenamiento implica redundancia de valores ... a pesar de estar en FNBC

NSS	Proy_nom	Becario_nom
1111	Diraya	Juanmi
1111	Extenda	Luis
1111	<i>NULL</i>	Manolo

...

NSS	Proy_nom	Becario_nom
1111	Extenda	Juanmi
1111	Extenda	Luis
1111	Diraya	Manolo

NSS	Proy_nom	Becario_nom
1111	Diraya	Juanmi
1111	Diraya	Luis
1111	Diraya	Manolo
1111	Extenda	Juanmi
1111	Extenda	Luis
1111	Extenda	Manolo

# Dependencias multivaluadas

- Las DMV tiene propiedades similares a las DF:
  - Los axiomas de Armstrong para DF:
    - Reflexiva: si  $Y$  es subconj de  $X$ , entonces  $X \rightarrow Y$
    - Aumentativa: si  $X \rightarrow Y$ , entonces  $XZ \rightarrow YZ$
    - Transitiva: si  $X \rightarrow Y$  y  $Y \rightarrow Z$ , entonces  $X \rightarrow Z$
  - Reglas de inferencia de DMV:
    - Complemento: si  $X \twoheadrightarrow Y$ , entonces  $X \twoheadrightarrow (R - (X \cup Y))$
    - Aumentativa: si  $X \twoheadrightarrow Y$  y  $Z$  subconj de  $W$ , entonces  $WX \twoheadrightarrow YZ$
    - Transitiva: si  $X \twoheadrightarrow Y$  y  $Y \twoheadrightarrow Z$ , entonces  $X \twoheadrightarrow (Z-Y)$



# Dependencias multivaluadas

- Las DMV están relacionadas con las DF:
  - ¿Qué es más general, una DF o una DMV?

# Dependencias multivaluadas

- Las DMV están relacionadas con las DF:
  - La DF como caso concreto de DMW:
    - Replicación: si  $X \rightarrow Y$  entonces  $X \twoheadrightarrow Y$ 
      - En concreto, una DF es una DMV en el que a cada X le corresponde como mucho un valor de Y
  - Relación entre DF y DMV:
    - *Coalescencia* (DF y DMV): si  $X \twoheadrightarrow Y$  y existe un W que ( $W \cap Y$  es vacío ,  $W \rightarrow Z$  y Z subconj de Y), entonces  $X \rightarrow Z$
- Si F es un conj de DF y DMV,  $F^+$  es su clausura al aplicarle todas estas propiedades

# Normalización avanzada

- *Cuarta Forma Normal*: una relación está en FN4, si cumple para toda  $X \twoheadrightarrow Y$  no trivial en  $F^+$ ,  $X$  es una superclave de  $R$
- Definiciones alternativas (de Esther):
  - Una relación  $R$  está en 4FN si, y sólo si siempre que exista una DMV en  $R$  de la forma  $R.x \twoheadrightarrow R.y$ , todos los demás atributos de  $R$  son funcionalmente dependientes de  $R.x$ , donde se cumple que:  
$$(R.x \rightarrow R.z, \forall R.z \in R)$$
  - O bien, las únicas DMV existentes en  $R$  son las DFs de la clave primaria con los atributos no primarios

# Normalización avanzada

- Para normalizar un esquema a FN4:
  - Descomponer la relación en relaciones de acuerdo a las DMW
  - Se divide/“rompe” R, que cumple que  $X \rightarrow Y$ , en:
    - $R_1 = X \cup Y$
    - $R_2 = (R - Y)$
    - R se elimina
    - Esta descomposición asegura la “lossless join property”, que se puede recuperar toda la información con un JOIN

# Normalización avanzada

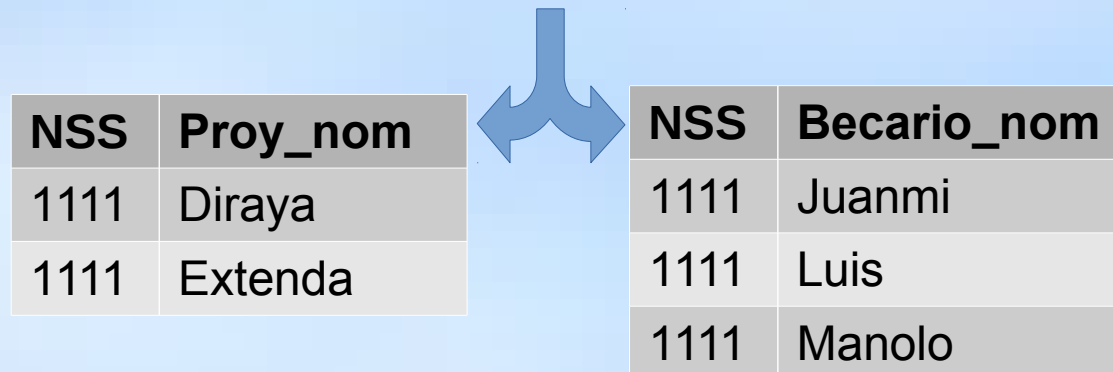
- Ej: Empleado(NSS, Proy\_nom, Becario\_nom)
  - Se cumple que  $NSS \rightarrow \text{Proy\_nom} \mid \text{Becario\_nom}$

NSS	Proy_nom	Becario_nom
1111	Diraya	Juanmi
1111	Diraya	Luis
1111	Diraya	Manolo
1111	Extenda	Juanmi
1111	Extenda	Luis
1111	Extenda	Manolo

# Normalización avanzada

- Ej: Empleado(NSS, Proy\_nom, Becario\_nom)
  - Se cumple que  $NSS \rightarrow \text{Proy\_nom} \mid \text{Becario\_nom}$

NSS	Proy_nom	Becario_nom
1111	Diraya	Juanmi
1111	Diraya	Luis
1111	Diraya	Manolo
1111	Extenda	Juanmi
1111	Extenda	Luis
1111	Extenda	Manolo

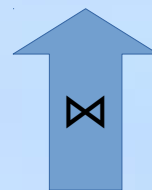


# Normalización avanzada

- Ej: Empleado(NSS, Proy\_nom, Becario\_nom)
  - Se cumple que  $R1 \times R2 = \text{Empleado}$

NSS	Proy_nom	Becario_nom
1111	Diraya	Juanmi
1111	Diraya	Luis
1111	Diraya	Manolo
1111	Extenda	Juanmi
1111	Extenda	Luis
1111	Extenda	Manolo

NSS	Proy_nom
1111	Diraya
1111	Extenda



NSS	Becario_nom
1111	Juanmi
1111	Luis
1111	Manolo

# Normalización avanzada

- Estar en FN4 ahorra espacio:
  - El espacio desperdiciado se multiplica:
    - Cuantos más atributos haya en cada elemento de la relación (X, Y ó Z)
    - Cuantos más valores tengan esos atributos
- Y también evitan anomalías de actualización



# Normalización avanzada

- Una BD puede tener problemas con FN4 por:
  - Atributos multivaluados
  - Varias relaciones 1:N independiente en una misma entidad
  - Además, al modificar un esquema “alegremente” se puede estar añadiendo un campo que viole la FN4

# Normalización avanzada

- Existen situaciones en las que no hay una descomposición en dos relaciones que permita recuperar toda la información con un JOIN
  - Y puede ser que no haya DMV no triviales que violen la FN4
- Para esos casos se definió la Dependencia de Producto (Join Dependency) JD

# Normalización avanzada

- Dependencia de Producto (Join Dependency)  
 $JD(R_1, R_2, \dots, R_n)$  sobre el esquema  $R$  indica que:
  - Todo estado  $r$  de  $R$  debe tener una descomposición “lossless join property” en  $R_1, R_2, \dots, R_n$
- Esto es, para todo  $r$  se tiene que
$$\bowtie(\Pi_{R_1}(r), \Pi_{R_2}(r), \dots, \Pi_{R_n}(r)) = r$$
- En concreto, una DMV es una JD con  $n=2$

# Normalización avanzada

- *Quinta Forma Normal*: una relación está en FN5 (o project-join normal form PJNF), si cumple para toda  $JD(R_1, R_2, \dots, R_n)$  no trivial en  $F^+$ , todo  $R_i$  es una superclave de  $R$
- Definición alternativa (de Esther):
  - Una relación  $R$  está en FN5 si, y sólo si siempre toda JD en  $R$  está implicada por las claves candidatas entre sí y no por cualquier otro atributo de  $R$ , forme o no parte de las claves candidatas

# Normalización avanzada

- Para normalizar un esquema a FN5:
  - Descomponer la relación en relaciones de acuerdo a las JD
  - Se divide  $R$ , donde  $JD(R_1, R_2, \dots, R_n)$ , en  $R_1, R_2, \dots, R_n$  y se elimina  $R$

# Normalización avanzada

- Ejemplo:
  - Suministra(proveedor,material,obra)
    - Cuando un proveedor atiende a una obra, provee todos los materiales que esa obra usa que él ofrezca
      - Es una especie de “proveedor potencial”
      - ¿Tiene sentido o es un ejemplo rebuscado?

# Normalización avanzada

- Ejemplo:
  - Suministra(proveedor,material,obra)
    - Cuando un proveedor atiende a una obra, provee todos los materiales que esa obra usa que él ofrezca
      - Es una especie de “proveedor potencial”
      - ¿Tiene sentido o es un ejemplo rebuscado?
        - IMHO es un ejemplo real y con sentido: si un proveedor puede suministrar alguna a una de mis obras, me interesa que aparezca como proveedor potencial (por si ofrece mejor precio/plazo de entrega, por si “se cae” otro proveedor, ...)
        - Otra cosa (que sí es cierta) es que no suele ser una restricción muy común.
          - Pero cosa es que algo se use pocas veces (como pasa con el Pretérito pluscuamperfecto de subjuntivo) y otra cosa que no tenga sentido

# Normalización avanzada

- Ejemplo:
  - Suministra(proveedor,material,obra)
    - Cuando un proveedor atiende a una obra, provee todos los materiales que esa obra usa que él ofrezca

Proveedor	Material	Obra
José Luis	Cobre	Cádiz
José Luis	Hierro	Jerez
Ángel	Cobre	Jerez
Carmen	Hierro	Puerto Real
Ángel	Acero	Cádiz

- ¿Es válido este estado de la BD?



# Normalización avanzada

- Ejemplo:
  - Suministra(proveedor,material,obra)
    - Cuando un proveedor atiende a una, provee todos los materiales que esa obra usa que él ofrezca

Proveedor	Material	Obra
José Luis	Cobre	Cádiz
José Luis	Hierro	Jerez
Ángel	Cobre	Jerez
Carmen	Hierro	Puerto Real
Ángel	Acero	Cádiz

- ¿Es válido este estado de la BD?
  - No, porque Ángel ofrece Cobre, pero no lo lleva a Cádiz (que sí lo usa). Ni José Luis no lleva Cobre a Jerez (que también lo usa)

# Normalización avanzada

- Ejemplo:
  - Suministra(proveedor,material,obra)
    - Cuando un proveedor atiende a una, provee todos los materiales que esa obra usa que él ofrezca

Proveedor	Material	Obra
José Luis	Cobre	Cádiz
José Luis	Hierro	Jerez
Ángel	Cobre	Jerez
Carmen	Hierro	Puerto Real
Ángel	Acero	Cádiz
<i>Ángel</i>	<i>Cobre</i>	<i>Cádiz</i>
<i>José Luis</i>	<i>Cobre</i>	<i>Jerez</i>

- Ahora sí está coherente

# Normalización avanzada

- Ejemplo:
  - Suministra(proveedor,material,obra)
    - Cuando un proveedor atiende a una, provee todos los materiales que esa obra usa que él ofrezca

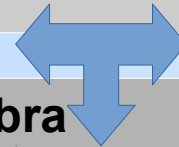
Proveedor	Material	Obra
José Luis	Cobre	Cádiz
José Luis	Hierro	Jerez
Ángel	Cobre	Jerez
Carmen	Hierro	Puerto Real
Ángel	Acero	Cádiz
<i>Ángel</i>	<i>Cobre</i>	<i>Cádiz</i>
<i>José Luis</i>	<i>Cobre</i>	<i>Jerez</i>

- Ahora sí está coherente, con problemas de actualización

# Normalización avanzada

- Ej: Suministra(proveedor,material,obra) en FN5

Proveedor	Material	Obra
José Luis	Cobre	Cádiz
José Luis	Hierro	Jerez
Ángel	Cobre	Jerez
Carmen	Hierro	Puerto Real
Ángel	Acero	Cádiz
<i>Ángel</i>	<i>Cobre</i>	<i>Cádiz</i>
<i>José Luis</i>	<i>Cobre</i>	<i>Jerez</i>



Proveedor	Material
José Luis	Cobre
José Luis	Hierro
Ángel	Cobre
Ángel	Acero
Carmen	Hierro

Proveedor	Obra
José Luis	Cádiz
José Luis	Jerez
Ángel	Jerez
Ángel	Cádiz
Carmen	Puerto Real

Material	Obra
Cobre	Cádiz
Hierro	Jerez
Cobre	Jerez
Hierro	Puerto Real
Acero	Cádiz

# Normalización avanzada

- Los problemas de la FN5 no son fáciles de detectar en general
- Y menos en una BD con decenas de tablas y cientos de campos
- Lo que no quiere decir que no existan ni vayamos a sufrir sus consecuencias

# Normalización avanzada

- Hay más formas normales
  - Normalmente para aspectos muy específicos
  - Por ejemplo:
    - Essential tuple normal form (ETNF)
    - FN6, que no la veremos:
      - Más info en Wikipedia:  
[https://en.wikipedia.org/wiki/Sixth\\_normal\\_form](https://en.wikipedia.org/wiki/Sixth_normal_form)

# Normalización avanzada

- Sí que veremos la Forma Normal Dominio-Clave (DKNF)
  - Tiene por objetivo definir la forma normal más alta que se podría lograr, teniendo en cuenta toda dependencia y restricción
  - Una relación  $R$  está en DKNF si todas sus dependencias y restricciones se pueden “garantizar” con restricciones de dominios y de claves (sin depender de productos, etc)
  - Define un ideal que no se puede garantizar para todo caso

# Fases para el diseño lógico

- Las fases para el diseño de una BBDD son:
  - Diseño conceptual
  - Diseño lógico
  - Diseño físico
- Diseño lógico: objetivo principal obtención del esquema lógico basado en el esquema conceptual obtenido en la fase anterior
  - ¿Qué SGBD se va a utilizar?
  - Utilización de modelo de datos ajustado al SGBD
- Documentación con todas las etapas de esta fase



# Fases para el diseño lógico

- Documentación:
  - Restricciones semánticas
  - Estudio de atributos:
    - Nombre
    - Dominio
    - Rango
  - Estudio de dependencias funcionales:
    - Transitivas
    - Multivaluadas
    - Reunión (JD)
  - Estudio de claves candidatas
  - Proceso de normalización: hasta dónde llegar
  - Obtención del Conjunto de esquemas de la BD

# Referencias

- Apuntes Esther Gadeschi
- Libro Elmasri, 3<sup>a</sup> ed.
- [https://en.wikipedia.org/wiki/Fifth\\_normal\\_form](https://en.wikipedia.org/wiki/Fifth_normal_form)

Gracias por la atención  
*¿Preguntas?*