TEMA 05 – NORMALIZACIÓN

Problemas de diseño de base de datos relacional

- El diseño necesita encontrar una buena colección de esquemas de relación (tablas)
- Mal diseño puede provocar:
  - Redundancias
  - Imposibilidad de representar cierta info
  - Operaciones no eficientes
  - Facilitar la aparición de datos inconsistentes
- Objetivos del diseño:
  - Evitar redundancias
  - Asegurar que se representan las relaciones y sus atributos
  - Facilitar la comprobación de actualizaciones para que no violen las restricciones de integridad
- Descomposición
  - Todos los atributos de la tabla original (R) deben aparecer en la descomposición
    - $R = R_1 \cup R_2$
  - Descomposición de unión sin pérdidas
    - Para todos los posibles datos de cada tabla (r en la tabla R), los datos pueden obtenerse a partir de los de R1 y R2
    - $R = \prod_{R1} (r) \bowtie \prod_{R2} (r)$
      - $\bowtie$ (símbolo matemático para JOIN)
- Objetivos de la normalización
  - Decidir cuando una relación R tiene una “buena” forma
  - En el caso de que una relación R no esté en buena forma, indicar cómo descomponerla de forma que:
    - 1º: cada nueva relación esté en una forma buena
    - 2º: la descomposición es de unión sin pérdidas
  - Nuestra teoría se basa en:
    - Dependencias funcionales (las que vemos)
    - Dependencias multievaluadas

Normalización intuitiva

- Estrategia de abajo a arriba (down-to-up)
- Se parte de los atributos y la normalización los va agrupando en tablas
- Conseguimos:
  - Evitar anomalías en inserciones, modificaciones y borrados de datos
  - Mejorar la independencia de los datos
  - No establecer restricciones artificiales en la estructura de los datos
- Se basa en el concepto de dependencias
  - Propiedad inherente a la semántica de los datos (su significado)
  - Forman parte de las restricciones del usuario del modelo relacional
  - Tipos:
    - Dependencias funcionales: completa, parcial y transitiva
    - Dependencias multivaluadas

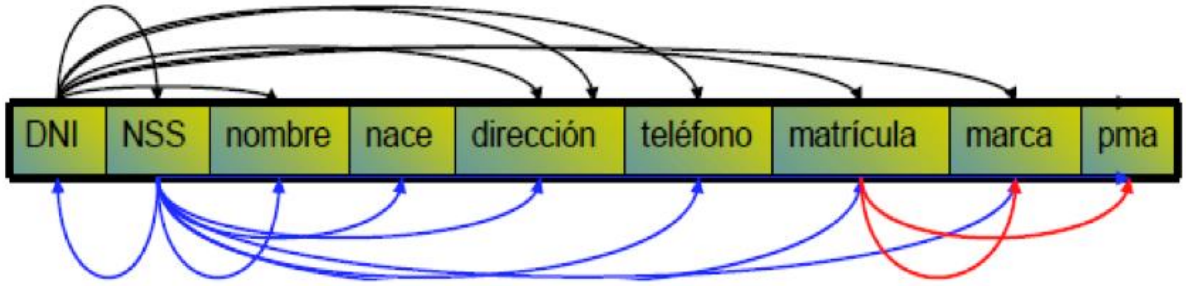
        - Dependencias de reunión
- Dependencias funcionales (DF)
    - Dada una tabla "T," decimos que el atributo(s) "T.y" depende fundamentalmente del atributo(s) "T.x"
    - Se escribe así: T.x → T.y
    - Se lee de estas 2 formas:
        - T.x determina funcionalmente a T.y
        - T.y depende funcionalmente de T.x
    - Si y solo si un único valor de Y está asociado a cada valor de X en la tabla T

Ejemplo. La tabla PERSONA almacena información de personas y sus vehículos:

```
PERSONA = dni + NSS + nombre + nace + dirección + teléfono + matrícula +
          marca + PMA
```

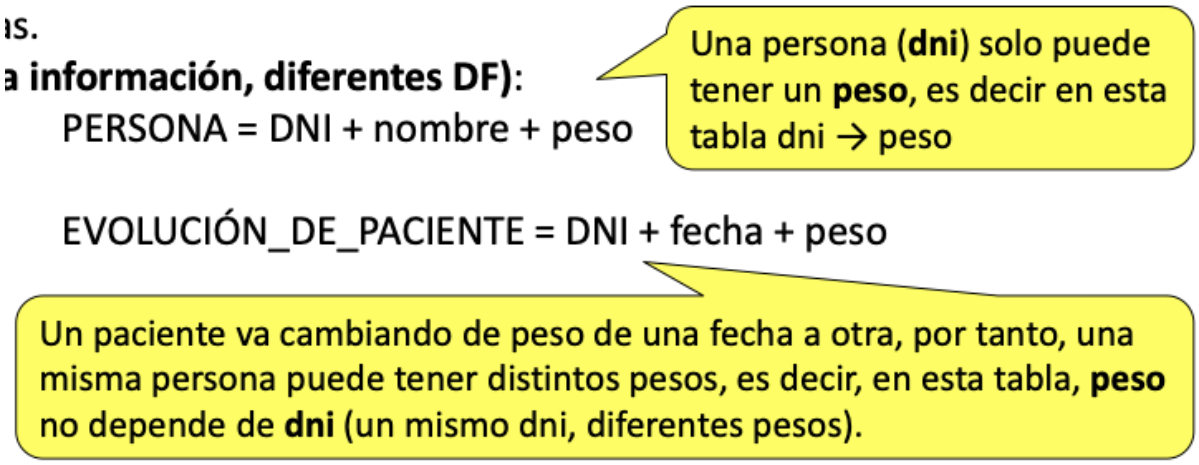
- ***DNI → nombre*, en el caso de que la tabla no pueda tener estas filas:**
    - **55555555A + Jose Luis + ...**
    - **55555555A + Mari Carmen + ...**
- **Si cada vez que aparezca el DNI 55555555A, el nombre debe ser Jose Luis, se dice que nombre depende funcionalmente de dni o que dni determina funcionalmente a nombre (en la tabla PERSONA, un valor de dni no puede tener asociados varios valores de nombre).**

    - Todos los atributos de una tabla dependen funcionalmente de la PK
    - Dependencia funcional completa: se dice que el conjunto de atributos Y de la tabla T es dependiente funcional por completo del conjunto de atributos X
        - Siempre que X → Y
        - Y no exista un subconjunto propio de X del que Y dependa funcional-
    - Las DF se representan mediante un diagrama de dependencias:



    - Observaciones:
        - La DF es una propiedad semántica
            - Reconocer las dependencias funcionales es parte del proceso de entender qué significan los datos en nuestra base de datos
            - El hecho de que "DNI → Nombre" significa que cada persona tiene un único nombre
            - La forma de garantizarlo es especificarlo en el esquema, para que el SGBD pueda hacer que se cumpla
        - Como una DF es una restricción de integridad, hay que expresar "DNI → Nombre" en algún lenguaje

        - Regla de integridad: comprueba que para toda fila F.x y para toda fila F.y (si Fx.dni = Fy.dni, entonces Fx.nombre = Fy.nombre)
    - Las DF representan relaciones de cardinalidad 1:N
        - Cada nombre puede asociarse a muchos DNI (hay muchos Pepe)
        - Cada DNI se asocia a un único nombre (este DNI es de este Pepe)
    - La integridad referencial es parecida a la DF, pero una DF se aplica en el interior de una tabla y la referencia puede superar los confines de la tabla
    - Asumir una DF implica dar un significado a la información
    - Es posible que 2 organizaciones den un significado diferente o que le interese interpretarla de diferente manera en 2 lugares o escenarios
        - No hay reglas generales a la hora de asumir una DF
        - Ejemplo:



Primera forma normal (1FN)

- El dominio de un atributo es atómico si sus elementos se consideran unidades indivisibles
- Elementos no atómicos:
    - Partes del nombre de una persona
    - Código compuesto (N301: N ala norte, piso 3, habitación 01)
    - Lista de teléfonos
- Una tabla T está en primera forma normal (1FN) si los dominios de todos sus atributos son atómicos
- Problema si no lo está: complican el almacena- y permiten que el SGBD relacional no pueda controlar las redundancias
- Un mismo atributo a veces será considerado como atómico o como compuesto
    - Atómico: si se usa de esa forma
    - Compuesto: si en la BD hay trabajos que necesiten usar o manipular de forma aislada alguna de las partes que lo forman
- Convertir a 1FN:

Ej: Jose Miguel de los Prados y todos los Ángeles.
Nombre: 'Jose Miguel'. Par1:'de los'. Ape1: 'Prados'

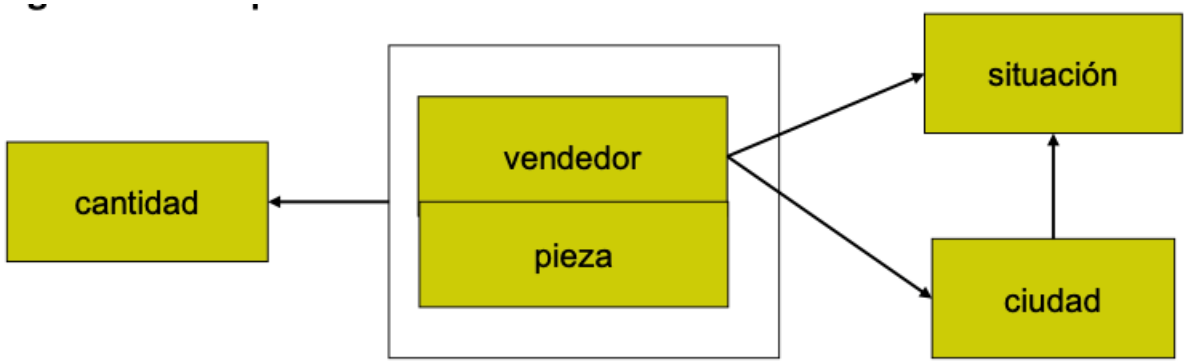
PERSONA = dni + nombre(nombre, par1, ape1, par2, ape2) + {teléfonos}

- **Eliminar las partes:** cada parte será un atributo individual.
  PERSONA1 = dni + nombre + par1 + ape1 + par2 + ape2 + {teléfonos}
- **Eliminar listas:** Rompemos la tabla original en 2 (como el mapeo de atributos multivaluados).
  PERSONA1 = dni + nombre + par1 + ape1 + par2 + ape2
  PERSONA2 = dni + teléfono
  Clave ajena: dni → PERSONA1(dni)

Segunda forma normal (2FN)

- Estamos mezclando la información de un vendedor con la información que describe un pedido que atiende ese vendedor:

VENDEDORES = vendedor + situación + ciudad + pieza + cantidad
Clave Primaria: vendedor + pieza



- Redundancias:
  - Inserción
    - No podemos insertar un nuevo vendedor en la BD hasta que suministre una pieza (la pieza es parte de la PK)
  - Borrado
    - Si borramos la fila de V3, también borramos la ciudad y situación (perdemos más info de la que queremos borrar)
  - Modificación
    - Si el vendedor de V1 cambia de ciudad, debemos buscar todas las filas de ese vendedor para modificarlas todas (si no, inconsistencias)

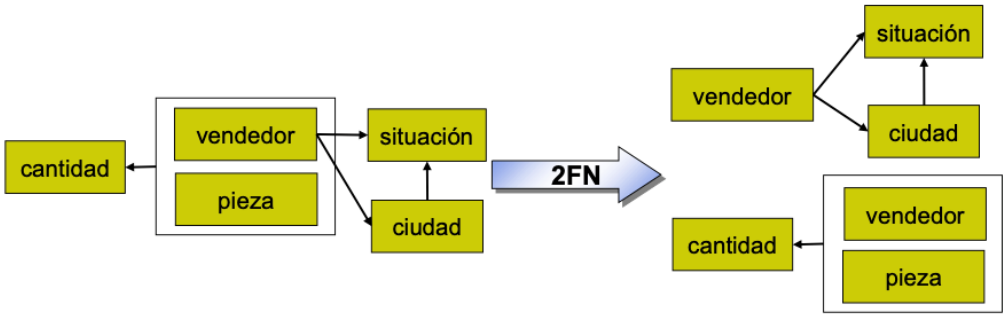
VENDEDORES				
VENDEDOR	SITUACIÓN	CIUDAD	PIEZA	CANTIDAD
V1	20	Alicante	P1	300
V1	20	Alicante	P2	200
V1	20	Alicante	P3	400
V1	20	Alicante	P4	200
V1	20	Alicante	P5	100
V1	20	Alicante	P6	100
V2	10	Castellón	P1	300
V2	10	Castellón	P2	400
V3	10	Torrente	P2	200
V4	20	Valencia	P2	200
V4	20	Valencia	P4	300
V4	20	Valencia	P5	400

- Problema intuitivo: mezcla información de vendedores y pedidos
- Problema técnico: hay atributos que dependen funcional- de una parte de la PK, no de toda
- Solución: separar la info en 2 tablas que no tengan ese problema
- Definición 2FN: si está en 1FN y además todos los atributos no clave dependen por completo de la PK
- Convertir a 2FN:
    - Se descompone en un conjunto de tablas equivalentes que sí lo están
    - Para que una tabla en 1FN no esté en 2FN, su clave debe ser compuesta
    - Proceso:
        - Dada la tabla T = A + B + C + D con PK (A + B) y una DF A → D, se sustituye T por:
        - T1 = A + D
        - T2 = A + B + C y FK A → T1
        - La tabla T puede recuperarse si se hace una reunión de FK a PK de T2 a T1
    - Ejemplo:

VENDEDOR = vendedor + situación + ciudad + pieza + cantidad

- **¿Está en 1FN? Sí**
- **¿Puede no estar en 2FN?** Es decir, **¿Su clave primaria es compuesta?** Sí. **¿Está en 2FN?** No. Su clave primaria es (vendedor + pieza) pero ciudad y situación dependen funcionalmente de vendedor, no de toda la clave primaria. La descomposición es:

  VENDEDOR = vendedor + ciudad + situación
  PEDIDO = vendedor + pieza + cantidad
  Clave ajena: vendedor → VENDEDOR(vendedor)



- Comprobar las ventajas

PEDIDO		
VENDEDOR	PIEZA	CANTIDAD
V1	P1	300
V1	P2	200
V1	P3	400
V1	P4	200
V1	P5	100
V1	P6	100
V2	P1	300
V2	P2	400
V3	P2	200
V4	P2	200
V4	P4	300
V4	P5	400

VENDEDOR		
VENDEDOR	SITUACIÓN	CIUDAD
V1	20	Alicante
V2	10	Castellón
V3	10	Torrente
V4	20	Valencia

ANOMALÍAS:

INSERCIÓN: podemos insertar un nuevo vendedor en la BD, sin que suministre una pieza.

BORRADO: Si borramos la fila de PEDIDOS de V3, solo borramos el envío de 200 piezas P2, pero no perdemos la ciudad y situación de V3.

MODIFICACIÓN: Si el vendedor V1 cambia de ciudad, debemos modificar una sola fila.

Tercera forma normal (3FN)

CLAVES:

- 1FN: se cumple si no hay compuestos ni multivaluados
- 2FN: se cumple si
  - No hay clave compuesta
  - Habiendo clave compuesta, ningún atributo tiene DF de una de las dos claves
- 3FN: se cumple si no hay DF de un atributo respecto a otro