Universidad
Europea de Madrid

LAUREATE INTERNATIONAL UNIVERSITIES

# NORMALIZACIÓN

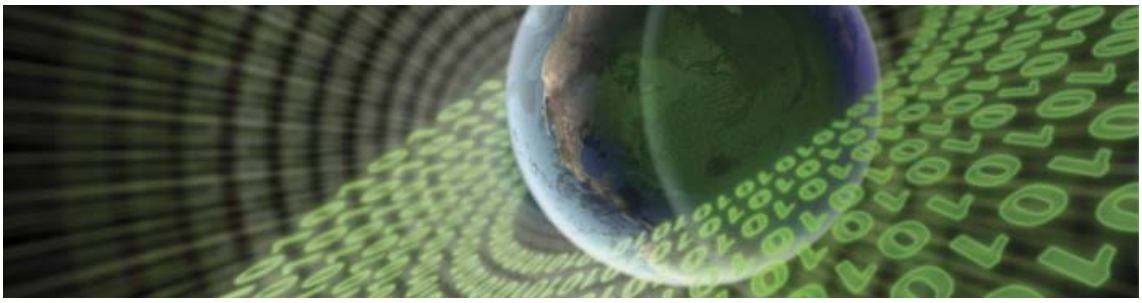
## PROBLEMAS DE UN MAL DISEÑO DE BD

## Índice

## Presentación

Para construir un sistema de base de datos es necesario realizar, tras la fase de análisis, un diseño adecuado tanto de la base de datos como de las aplicaciones que utilizarán los usuarios para el acceso a la información. Ya aprendimos anteriormente que el diseño de la BD se iniciaba con los diagramas ER que modelizaban la información que maneja la empresa u organización y de los cuales se puede extraer un esquema relacional inicial.

Un esquema relacional óptimo tiene que evitar las redundancias para reducir el espacio de almacenamiento en disco, la posibilidad de cometer incoherencias y mantener un tiempo de respuesta óptimo ante una consulta realizada por un usuario.



En este tema aprenderás a:

- Comprender los problemas de un mal esquema relacional.
- Refinar el esquema relacional inicial obtenido a partir de los diagramas ER utilizando un procedimiento denominado **normalización**.

Al conjunto de tablas resultado de este proceso lo denominaremos **esquema relacional normalizado**. Este proceso hay que realizarlo antes de empezar a construir la base de datos, utilizando el LDD estudiado anteriormente.

## Propiedades deseables de una BD

El esquema relacional de una base de datos es mejor si tiene las siguientes propiedades deseables:

- **Que no haya redundancia**. Hay que reducir al mínimo los datos que aparecen repetidos en la misma ó en distintas tablas, ya que supone:
    - Mayor necesidad de almacenamiento. Cuantos más datos repitamos más espacio de almacenamiento necesitaremos.
    - Mayor dificultad de mantenimiento. Puesto que hay que comprobar que la información sea coherente. Por ejemplo, que la dirección del cliente 'González' sea siempre la misma aunque aparezca repetida en distintos sitios dentro de la base de datos. Esto obligará a que si realizamos alguna modificación de este dato se modifique convenientemente en todos los lugares donde se almacene.
- **Que no se pierda información**. Tenemos que conseguir un esquema relacional que refleje todas las restricciones semánticas del problema en el mundo real.

Un mal diseño de una base de datos relacional puede generar muchos problemas (la intuición no siempre da buenos resultados).

Hay que ser metódicos a la hora de realizar el diseño de la BD. Un **procedimiento de diseño** que puede ahorrarnos muchos problemas consiste en:

- Realizar el diagrama ER y obtener un esquema preliminar a partir de él.
- Refinar dicho esquema hasta obtener un diseño adecuado a las restricciones del sistema. Este último paso se denomina **normalización** y se realiza analizando las restricciones semánticas del mundo real y cómo se ven reflejadas en las relaciones que existen entre los atributos de las tablas. Estas relaciones se denominan **dependencias**.

## Normalización

La **normalización** es una técnica utilizada en el diseño de bases de datos. Define un procedimiento a seguir basado en el estudio de las restricciones semánticas del mundo real y su correspondencia con las relaciones entre los atributos del problema. El objetivo es obtener un esquema relacional resultado que satisfaga las propiedades deseables de toda base de datos estudiadas en el apartado anterior: que evite redundancias innecesarias y que no se pierda información.

Las relaciones, que por la propia semántica del problema del mundo real, se establecen entre los atributos se denominan **dependencias**. Existen tres tipos de dependencias: **funcionales** (DF), **multivaluadas** (DMV) y **de reunión** (DR). Las estudiaremos a lo largo de esta unidad de aprendizaje.

Un esquema relacional de calidad estará compuesto por un conjunto de relaciones que cumplan las siguientes características:

- Tener el mínimo número de atributos necesario para cumplir con las restricciones semánticas del problema del mundo real que estamos modelizando.
- Mantener las dependencias funcionales (DF) en la misma relación. Los atributos con una relación lógica fuerte entre ellos (lo que se denomina dependencia funcional), se encuentran en la misma relación. De esta forma, la restricción semántica del mundo real que modela esta DF se seguirá manteniendo en el esquema relacional final y, por tanto, no se perderá información.
- Tener la mínima redundancia posible, sin repetir atributos innecesariamente. Hay una excepción a esta regla: las claves externas, que mantienen las relaciones que se establecen entre las entidades del problema y, por tanto, son inevitables.

## Redundancia de datos y problemas de actualización

Para estudiar los problemas que se generan derivados de la redundancia o repetición de los datos, vamos a tratar un ejemplo introductorio. Supongamos una base de datos que almacena información sobre jugadores de fútbol:

```
Futbol (#jugador, nombre, posición, nación, #equipo, estadio)
```

Una posible instancia de esta base es:

| #jugador | nombre | posicion | nacion | #equipo | estadio |
|---|---|---|---|---|---|
| 001 | ortiz | medio | esp | cor | nuevo arcangel |
| 002 | henry | delantero | fra | ars | highbury |
| 003 | romaric | medio | cmf | cor | nuevo arcangel |
| 074 | puyol | defensa | esp | bar | nou camp |
| 005 | joaquín | medio | esp | bet | ruiz de lopera |
| 032 | casillas | portero | esp | mad | bernabeu |
| 055 | besagno | defensa | usa | cor | nuevo arcangel |

Como se puede comprobar, esta relación tiene redundancias: el estadio del equipo aparece repetido para cada uno de los jugadores del equipo. Veamos los problemas que aparecen derivados de esta redundancia:

- **Problemas al insertar**. Podemos encontrarnos con dos tipos de **problemas**.
- **Problemas al eliminar**. Si eliminamos el último jugador que nos queda de un equipo, también estaremos eliminando la información que se almacena de dicho equipo.
- **Problemas al modificar**. Si se modificase el nombre de algún estadio que apareciese varias veces (ej. “Nuevo Arcángel”), habría que **garantizar** que ese cambio se traslada a todas las repeticiones. Si no fuese así, la BD quedaría en un estado de **incoherencia**.

**Problemas al insertar**

- A la hora de insertar un jugador, debe indicarse la información del propio jugador y además volver a introducir la del equipo. Esto puede ocasionar inconsistencias si se introducen mal los datos, que no corresponderían con los que ya hay almacenados.
- No podría introducirse un equipo que inicialmente no tuviera jugadores, ya que no permitiría tener a NULL el atributo #jugador que es la clave de la relación.

## Solución a los problemas de actualización: evitar redundancias

Para evitar las redundancias que aparecían en el esquema anterior vamos a crear dos tablas: una que contenga la **información de los equipos de fútbol** y otra que contenga la **información de los jugadores**. Hay que utilizar una **clave externa**, en este caso **#equipo**, que permita mantener la relación del equipo al que pertenece cada jugador.

Nos quedaría como esquema:
Futbol ( #jugador, nombre, posición, nación, #equipo)
Equipo (#equipo, estadio)

En estas **tablas** podemos ver una posible instancia de esta base de datos.

Analicemos ahora si se solventan en esta nueva situación los problemas descritos en el apartado anterior:

- **Problemas al insertar**:
  - Cuando se inserta un nuevo jugador no es necesario volver a introducir la información del equipo, por lo que se evitan las inconsistencias.
  - Se puede introducir la información del equipo aunque inicialmente no tuviera jugadores.
- **Problemas al eliminar**. Si eliminamos el último jugador que nos queda de un equipo, no estamos eliminando la información que se almacena de dicho equipo puesto que se almacena en una tabla aparte.
- **Problemas al modificar**. Cada estadio aparece sólo una vez por lo que no hay peligro de que la BD quede en un estado de incoherencia cuando se modifique el nombre de un estadio.

**Tablas**

| #jugador | nombre | posicion | nacion | #equipo |
|---|---|---|---|---|
| 001 | ortiz | medio | esp | cor |
| 002 | henry | delantero | fra | ars |
| 003 | romaric | medio | cmf | cor |
| 074 | puyol | defensa | esp | bar |
| 005 | joaquin | medio | esp | bet |
| 032 | casillas | portero | esp | mad |
| 055 | besagno | defensa | usa | cor |

| #equipo | estadio |
|---|---|
| cor | nuevo arcangel |
| ars | highbury |
| bet | ruiz de lopera |
| mad | bernabeu |
| bar | nou camp |

## Concepto de dependencia funcional

Derivadas de las restricciones que se cumplen en el mundo real, se establecen relaciones entre los atributos de una tabla que las modelizan. En el ejemplo de la empresa comercial una restricción podría ser que cada cliente solo puede realizar un pedido en un mismo día. Esto implica que existe una relación entre los atributos dni_cliente, n_pedido y fecha_pedido. La relación es que, si conozco el cliente y la fecha de pedido, puedo localizar el pedido del que estoy hablando.

Estas relaciones se denominan **dependencias funcionales** (**DF**) y se notan con el símbolo ' $\longrightarrow$ '. En nuestro caso la DF que se establece es: *dni_cliente,fecha_pedido* $\longrightarrow$ *n_pedido.*

Formalmente:

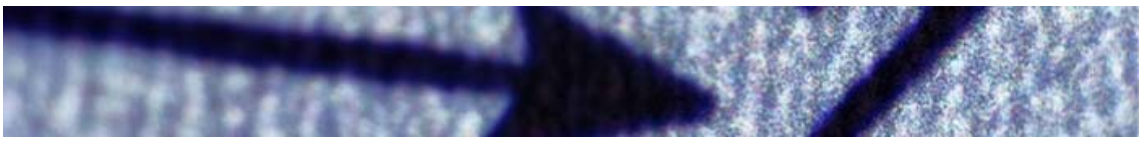
Supóngase la relación R compuesta por los atributos A1, A2 … An: R (A1, A2, … , An). Supónganse X e Y, dos subconjuntos de los atributos de R. Una relación R satisface la **dependencia funcional X** $\longrightarrow$ **Y**, si para todo par de tuplas $\mu$ y $\nu$ de R, se cumple que:

$$\mu[X] = \nu[X] \Rightarrow \mu[Y] = \nu[Y]$$

Se dice que **Y depende funcionalmente de X** o **X determina funcionalmente a Y**.

Derivada de la definición de DF, se define el concepto de clave candidata como:

Dado X un subconjunto de atributos de R, si X es **clave candidata** de R, entonces X determina funcionalmente todos los atributos de R.



Ejemplo **Ejemplo 1** Ejemplo **Ejemplo 2**

**Ejemplo**

**Ejemplo 1**

En la relación que se muestra abajo se cumplen las siguientes DF:

- Apellido → Teléfono: Siempre que coinciden los valores para apellido, coinciden también para teléfono.
- Nombre → Edad: Siempre que coinciden los valores para nombre, coinciden también para edad.

| apellido | nombre | telefono | edad |
|---|---|---|---|
| orozco | daniel | 372 321 444 | 28 |
| beltran | alejandro | 300 983 212 | 28 |
| orozco | raquel | 372 321 444 | 29 |
| muñoz | raquel | 721 303 004 | 29 |
| beltran | ernesto | 300 983 212 | 56 |

**Ejemplo**

**Ejemplo 2**

Las DF que se establecen en la siguiente relación: A → C, C → A, AB → D.

| A | B | C | D |
|---|---|---|---|
| a | b | z | q |
| e | b | r | p |
| a | d | z | t |
| e | d | r | q |
| a | f | z | t |
| e | f | r | t |

## Características de las DF

### Axiomas de Amstrong

Los axiomas de Amstrong ó reglas de inferencia, son tres reglas que permiten obtener todas las DF lógicamente implicadas a partir de un conjunto dado.


**Los tres axiomas de Amstrong**

### Reglas de inferencia adicionales

Existe otro conjunto de reglas de inferencia que se obtienen a partir de los axiomas de Amstrong. Podemos considerar las siguientes como las más significativas:


**Reglas de inferencia adicionales**

### Cierre de F

Se denomina cierre de un conjunto de DF F, $F^+$, al conjunto de todas las DF implicadas lógicamente por F.

$$F^+ = \{ X \rightarrow Y \mid F \models X \rightarrow Y \}$$


**Características de las DF. Ejemplos**

**En detalle**

**Los tres axiomas de Amstrong**

Sea R una relación con un conjunto de atributos; sean X, Y, Z conjuntos de atributos pertenecientes a R; y sea F un conjunto de dependencias funcionales:

- Axioma de **Reflexividad**: $Y \subseteq X \Rightarrow X \rightarrow Y$
- Axioma de **Aumento**: $\{X \rightarrow Y\} \models \{XZ \rightarrow YZ\}$
  - Ej.: DNI → NOM ⊨ DNI DIR → NOM DIR
- Axioma de **Transitividad**: $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$
  - Ej.:{CCC → SALDO, DNI → CCC} ⊨ {DNI → SALDO}

Los axiomas de Amstrong son **seguros** y **completos**. Son seguros porque sólo conducen a conclusiones ciertas y son completos porque toda DF lógicamente implicada por un conjunto de DF puede obtenerse a partir de estos axiomas.

**En detalle**

**Reglas de inferencia adicionales**

- Regla de la **unión**: $\{X \rightarrow Y, X \rightarrow Z\} \models \{X \rightarrow YZ\}$
  - Ej.: {DNI → NOM, DNI → DIR} ⊨ {DNI → NOM DIR}
- Regla de **seudotransitividad**: $\{X \rightarrow Y, WY \rightarrow Z\} \models WX \rightarrow Z$
- Regla de **descomposición**: $Z \subseteq Y, X \rightarrow Y \Rightarrow X \rightarrow Z$
  - Ej.: {DNI → NOM DIR} ⊨ {DNI → NOM}
  - {DNI → NOM DIR} ⊨ {DNI → DIR}

## Clave

### Cierre de un conjunto de atributos

El cierre de un conjunto de atributos está compuesto por todos los atributos que dependen funcionalmente de X aplicando los axiomas de Amstrong. Formalmente:

Sea R una relación, X un conjunto de atributos de R y F un conjunto de DF definidas sobre R, se define el cierre de X con respecto a F como:

$$X^{+} = \{A \in R / F \vDash X \longrightarrow A\}$$

El siguiente **algoritmo** permite calcular el cierre de X con respecto a F.

### Definición funcional de clave

Sea R (A1, A2, …, An) con un conjunto de DF F. Aquí, X es **clave** si:

- El cierre de X incluye todos los atributos de la relación. $X^{+}$ = R = A1, A2, …, An
- No existe ningún Y $\subset$ X tal que $Y^{+}$ = R.

Derivado del concepto de clave se obtienen los siguientes **conceptos** que utilizaremos a lo largo de la materia.

### Recubrimiento mínimo ó canónico de F, Fm

Decimos que un **conjunto de DF** es mínimo o canónico, y se nota como Fm, si:

- La parte derecha de todas las DF f $\in$ F tienen un solo atributo.
- No existe ninguna DF X $\longrightarrow$ A en Fm tal que Fm-{X $\longrightarrow$ A} es equivalente a Fm.
- Para ninguna DF X $\longrightarrow$ A en Fm y un subconjunto propio Z $\subset$ X, se cumple que Fm – {X $\longrightarrow$ A} $\cup$ {Z $\longrightarrow$ A} es equivalente a Fm.

Ejemplo **Cierre de un conjunto de atributos**

Ejemplo **Definición funcional de clave**

**Algoritmo**

El siguiente algoritmo permite calcular el cierre de X con respecto a F.

- Hacer $X^{0}$ = X
- $X^{i+1} = X^{i} \cup \{Y / \exists Z \rightarrow Y \, y \, Z \in X^{i}\}$
- Repetir 2 hasta $X^{i+1} = X^{i}$

**Conceptos**

- **Superclave**: cualquier superconjunto de una clave.
- **Clave candidata**: cualquier conjunto que cumpla las propiedades.
- **Clave primaria**: clave candidata elegida.

**Conjunto de DF**

Todo conjunto de DF tiene al menos un recubrimiento mínimo. El SBD debe comprobar que no se viola ninguna DF. Si trabajamos con Fm en lugar de con F podemos reducir el esfuerzo necesario para las comprobaciones.

**Ejemplo**

**Cierre de un conjunto de atributos**

Dada la relación R={A,B,C,H,G,I} y el conjunto de dependencias F={A $\rightarrow$ B, A $\rightarrow$ C, CG $\rightarrow$ H, CG $\rightarrow$ I, B $\rightarrow$ H} que se satisfacen en R, calcular el cierre de X={AG}

- $X^0$={AG}
- $X^1$={AG} U {B} (porque A $\rightarrow$ B) U {C} (porque A $\rightarrow$ C) = {AGBC}
- $X^2$={AGBC} U {H} (porque CG $\rightarrow$ H) U {I} (porque CG $\rightarrow$ I) = {AGBCHI}
- $X^3$={AGBCHI}

X es clave candidata (CK), ya que todos los atributos dependen de él.

**Ejemplo**

**Definición funcional de clave**

Dada la relación BANCO (DNI, NOM, DIR, CCC, SALDO) y el conjunto de dependencias funcionales F= {DNI $\rightarrow$ NOM DIR, DNI $\rightarrow$ CCC, CCC $\rightarrow$ SALDO}:

$\{NOM\}^+$ = {NOM}

$\{DIR\}^+$ = {DIR}

$\{SALDO\}^+$ = {SALDO}

$\{CCC\}^+$ = {CCC, SALDO}

$\{DNI\}^+$ = {DNI, NOM, DIR, CCC, SALDO}; Clave candidata

$\{DNI, NOM\}^+$ = {DNI, NOM, DIR, CCC, SALDO}; Superclave

## Problemas asociados a la descomposición de una relación

En apartados anteriores vimos que la redundancia y, por tanto, los problemas derivados de ésta podían ser solucionados en muchos casos descomponiendo la relación en otras de tamaño más pequeño. Sin embargo, también aparecen problemas derivados de la descomposición de una relación:

### Pérdida de información

Debe garantizarse que cualquier instancia de la relación de origen pueda obtenerse a partir de las relaciones más pequeñas en las que se descompuso. Aquí surge la **propiedad de descomposición sin pérdida**, que desarrollamos en la siguiente pantalla.

Siguiendo con el ejemplo de los equipos de fútbol, debe garantizarse que al hacer la reunión de las tablas **jugador** y **equipo** se obtenga la relación original. En este caso es posible gracias a la clave externa **#equipo** que mantiene la información original.

**Contraejemplo**: si en la descomposición anterior utilizamos como clave externa el **nombre del estadio** en lugar del código del equipo y consideramos que dos equipos de fútbol distintos pueden compartir estadio, tendríamos el problema que se muestra en el documento *Pérdida de información*. Aparecen dos tuplas que no formaban parte de la relación original y que no expresan información real. Forlán no es jugador del Real Madrid y Casillas no lo es del Atlético de Madrid.


**Pérdida de información**

### No conservación de las dependencias

Debe garantizarse que las dependencias que se establecen entre los atributos de la relación original se sigan manteniendo en la descomposición realizada. Se deriva la **propiedad de la conservación de las dependencias**, que se desarrolla en las siguientes pantallas.

## Descomposición de producto sin pérdidas

Dado el esquema de una relación R, su descomposición $\rho$=(R1, R2, ..., Rk), y un conjunto de DF F para R:

$\rho$ es una **descomposición de producto sin pérdida** de R respecto a F si, para toda instancia de R que satisface F se cumple:

$$R=\pi_{R1}(R)|X|\pi_{R2}(R)|X|\ldots\pi_{Rk}(R)$$

**Criterio**: Sean R y F, y la descomposición $\rho$={R1, R2}

$\rho$ es una descomposición de producto sin pérdida si alguna de las DF siguientes está en F+.

$$R1 \cap R2 \longrightarrow R1$$

$$R1 \cap R2 \longrightarrow R2$$

El siguiente **algoritmo** permite comprobar si una descomposición es sin pérdidas, siguiendo los pasos que se indican más abajo.

Dadas R=($A_1$, $A_2$, …, $A_n$), F y $\rho$={$R_1$, $R_2$, …, $R_m$}.

- Crear una tabla de mxn que contenga como columnas los atributos de la tabla, como filas las relaciones que forman la descomposición y en las celdas el valor $a_j$ si el atributo $A_j$ pertenece a la relación $R_i$, y $b_{ij}$ en caso contrario.
- Para cada una de las DF X $\longrightarrow$ Y en F de R, si los valores de las celdas en X coinciden, hacer que coincidan en Y.
- Repetir el paso 2 hasta que no haya modificaciones.
- Si finalmente en la tabla hay una fila en la que todos los elementos son $a_j$, la descomposición es sin pérdidas; en caso contrario no lo es.


**Ejemplo**

## Descomposición que conserva las dependencias

El siguiente **algoritmo** permite detectar si una descomposición preserva las dependencias.

Dadas $R=A_1, A_2, .., A_n$, un conjunto F de DF y una descomposición $\rho = \{R_1, R_2, ..., R_m\}$:

Para cada DF $X \longrightarrow Y$ de F y mientras no se demuestre que no preserva las dependencias.

- Inicializar Z=X.
- Para cada relación $R_i$
  - Calculo $\alpha = Z \cap R_i$.
  - $Z = Z \cup \{\alpha_+ \cap Ri\}$ , ($\alpha_+$ representa el cierre de los atributos de $\alpha$)
- Si Y $\nsubseteq$ Z entonces **no** preserva las dependencias.

### Ejercicio

Dadas R(L,M,N,O,P,Q,R), F={ QL $\longrightarrow$ NRO, N $\longrightarrow$ R, N $\longrightarrow$ O, L $\longrightarrow$ M} y $\rho$={$R_1$, $R_2$} donde $R_1$=(L,M,N), $R_2$=(N,O,P,Q,R), comprobar que $\rho$ es una descomposición sin pérdidas.

Ejercicio **Solución**

**Ejercicio**

**Solución**

Aplicamos el algoritmo para comprobar si se preservan las dependencias. Empezamos comprobando con la dependencia QL $\longrightarrow$ NR puesto que los atributos de esta dependencia no se encuentran en la misma relación y, por tanto, es probable que no se preserve.

Z={QL}

- **R1(LMN)**
  - $\alpha$ = z $\cap$ R1 = {L}
  - Z=Z U{ $\cap$ $^+\alpha$ R1} = {QL} u {LM} = {QLM}
- **R2(NOPQR)**
  - $\cap$ = Z $\cap$ R2 = {QLM} $\alpha$ R2={Q}
  - Z=Z U{ $\alpha$ $^+\alpha$ R2} ={QLM} U {{Q} $\cap$ {NOPQR}}= {QLM}

Como NR no está incluido en Z ({QLN}) la descomposición no preserva las dependencias.

## Resumen

Para que un esquema relacional sea de calidad tiene que tener las siguientes propiedades deseables: **reducir** al mínimo las **redundancias** y **no perder información**. Esto se consigue mediante el proceso de **Normalización**.

La normalización estudia las dependencias funcionales (**DF**), multivaluadas (**DMV**) y de reunión (**DR**) que se establecen entre los atributos del problema. Como resultado de este estudio puede requerir descomponer la relación original en varias. Cuanto esto sea necesario, tenemos que asegurarnos de que esta **descomposición** sea **de producto sin pérdidas** y **conserve las dependencias**.

Las DF modelizan las restricciones semánticas que aparecen en el mundo real.

Las DF cumplen los **Axiomas de Amstrong** (Reflexividad, Aumento y Transitividad) y, derivados de éstos, otras **reglas de inferencia** (unión, seudotransitividad, y descomposición). La clave de una relación puede ser matemáticamente encontrada aplicando estos axiomas y reglas en un proceso denominado **cierre de un conjunto de atributos**.