Normalización

Bases de Datos 1 (CC3088) - 2026

Normalización

Semestre 01, 2026

Definición

La normalización es un proceso para:

- Organizar los datos
- Reducir redundancia
- Evitar inconsistencias
- Mejorar la integridad de la base de datos

Normalizar es diseñar bien, no solo separar tablas.

Necesidad

Una base de datos no normalizada puede provocar:

- Datos duplicados
- Errores al actualizar información
- Pérdida de datos

- Dificultad para mantener el sistema

Problemas típicos

- Redundancia de datos
- Anomalía de inserción
- Anomalía de eliminación
- Anomalía de actualización

Anomalías en bases de datos

Una anomalía es un problema que ocurre en una base de datos mal diseñada cuando se realizan operaciones básicas como:

- Insertar datos
- Actualizar datos
- Eliminar datos

Estas anomalías son consecuencia directa de la redundancia y de dependencias mal definidas.

Tipos de anomalías

Existen tres tipos principales de anomalías:

- Inserción
- Eliminación
- Actualización

Anomalía de inserción

Ocurre cuando no podemos insertar información sin depender de otros datos que no deberían ser obligatorios.

Ejemplo:

No se puede registrar un cliente nuevo si todavía no ha realizado ningún pedido.

Anomalía de eliminación

Ocurre cuando al eliminar un dato se pierde información importante que no queríamos borrar.

Ejemplo:

Si eliminamos el último pedido de un cliente, también perdemos sus datos personales.

Anomalía de actualización

Ocurre cuando una misma información debe actualizarse en varias filas.

Esto puede provocar inconsistencias.

Ejemplo:

Si un cliente cambia de dirección, debemos actualizarla en todas sus filas. Si olvidamos una, la base queda inconsistente.

Justificación

La normalización busca:

- Eliminar redundancia
- Separar correctamente la información
- Evitar estas anomalías

Las formas normales existen precisamente para prevenir este tipo de problemas.

Dependencias funcionales

Una dependencia funcional ocurre cuando:

A → B

A determina de forma única a B.

En bases de datos

- A = determinante
- B = dependiente

Ejemplo:

ClienteID → Nombre

ClienteID → Dirección

Tipos de dependencias funcionales

Dependencia funcional completa

Un atributo depende de toda la clave primaria.

(PedidoID, ProductoID) → Cantidad

Dependencia funcional parcial

Un atributo depende solo de parte de la clave compuesta.

(PedidoID, ProductoID) → NombreProducto

Dependencia funcional transitiva

Un atributo depende de otro atributo que no es clave.

ClienteID → CiudadID → NombreCiudad

Formas normales

Las formas normales son reglas que nos ayudan a:

- Diseñar tablas correctamente
- Eliminar dependencias problemáticas
- Mejorar la integridad de los datos

Primera Forma Normal (1NF)

Reglas de 1NF

Una relación está en Primera Forma Normal si:

- Cada atributo contiene valores atómicos
- No existen listas ni grupos repetitivos
- Cada celda contiene un solo valor

La 1NF se enfoca en la estructura básica de la tabla.

Ejemplo

Tenemos una tabla que almacena información de clientes y sus teléfonos de contacto.

NO en 1NF

| Cliente | Teléfonos | | ------ | ---------- | | Juan | 555-1234, 555-5678 | | María | 555-8765 |

Problema:

- Un atributo contiene múltiples valores
- No es posible identificar cada teléfono individualmente
- Se dificulta hacer búsquedas o actualizaciones

Problema estructural

Un solo campo almacena más de un dato.

Esto rompe la idea de que cada fila represente una sola ocurrencia.

Aplicar 1NF

Separamos los valores múltiples en filas individuales.

Tabla en 1NF

| Cliente | Teléfono | | ------ | -------- | | Juan | 555-1234 | | Juan | 555-5678 | | María | 555-8765 |

Resultado tras aplicar 1NF

- Cada celda contiene un solo valor
- Los datos son atómicos
- La tabla es más fácil de consultar y mantener

Segunda Forma Normal (2NF)

Reglas de 2NF

Una relación está en Segunda Forma Normal si:

- Cumple con 1NF
- No existen dependencias parciales
- Todos los atributos no clave dependen de toda la clave primaria

Esta forma normal solo aplica cuando la clave primaria es compuesta.

Ejemplo

Tenemos una tabla que registra productos incluidos en pedidos.

La clave primaria es compuesta:

(PedidoID, ProductoID)

NO en 2NF

| PedidoID | ProductoID | NombreProducto | Precio | | -------- | ---------- | ------------- | ------ | | 1 | 101 | Laptop | 1200 | | 1 | 102 | Mouse | 25 | | 2 | 101 | Laptop | 1200 |

Problema:

- NombreProducto y Precio dependen solo de ProductoID
- No dependen del PedidoID completo

Dependencia parcial detectada

ProductoID → NombreProducto

ProductoID → Precio

Problema

- La información del producto se repite
- Cambios de precio requieren múltiples actualizaciones
- Existe riesgo de inconsistencias

Aplicar 2NF

Separamos los atributos que no dependen de toda la clave primaria.

Tabla Productos

| ProductoID | Nombre | Precio | | ---------- | ------ | ------ | | 101 | Laptop | 1200 | | 102 | Mouse | 25 |

Tabla DetallePedido

| PedidoID | ProductoID | Cantidad | | -------- | ---------- | -------- | | 1 | 101 | 1 | | 1 | 102 | 2 | | 2 | 101 | 1 |

Resultado tras aplicar 2NF

- Eliminamos dependencias parciales
- La información del producto ya no se repite
- Cada tabla tiene una responsabilidad clara

Tercera Forma Normal (3NF)

Reglas de 3NF

Una relación está en Tercera Forma Normal si:

- Cumple con 2NF
- No existen dependencias transitivas
- Los atributos no clave dependen solo de la clave primaria

La 3NF elimina dependencias entre atributos que no forman parte de la clave.

Ejemplo

Tenemos una tabla que almacena información de clientes y su ubicación.

NO en 3NF

| ClienteID | Nombre | Ciudad | CódigoPostal | | --------- | ------ | ------ | ---------- | | 1 | Juan | Madrid | 28001 | | 2 | María | Barcelona | 08002 |

Problema:

- CódigoPostal no depende directamente de ClienteID
- Depende de Ciudad, que no es clave primaria

Dependencia transitiva detectada

ClienteID → Ciudad

Ciudad → CódigoPostal

Problema

- El código postal se repite innecesariamente
- Cambios en códigos postales requieren múltiples actualizaciones
- Existe riesgo de inconsistencias

Aplicar 3NF

Separamos los atributos que no dependen directamente de la clave primaria.

Tabla Clientes

| ClienteID | Nombre | CiudadID | | --------- | ------ | -------- | | 1 | Juan | 1 | | 2 | María | 2 |

Tabla Ciudades

| CiudadID | Nombre | CódigoPostal | | -------- | ---------- | ------------ | | 1 | Madrid | 28001 | | 2 | Barcelona | 08002 |

Resultado tras aplicar 3NF

- Eliminamos dependencias transitivas
- Cada atributo depende solo de la clave primaria
- El diseño es más claro y consistente

Forma Normal de Boyce-Codd (BCNF)

Reglas de BCNF

Una relación está en Forma Normal de Boyce-Codd si:

- Cumple con 3NF

- Para toda dependencia funcional X → Y
- X es una clave candidata

BCNF es una versión más estricta de la Tercera Forma Normal.

Ejemplo

Tenemos una tabla que relaciona profesores, materias y aulas.

NO en BCNF

| Profesor | Materia | Aula | | -------- | ---------- | ---- | | Pérez | Matemáticas | 101 | | López | Historia | 202 | | Pérez | Matemáticas | 101 |

Dependencias funcionales:

Profesor → Materia
Aula → Materia

Problema

- Aula determina Materia
- Aula no es clave candidata
- La tabla viola BCNF aunque cumple 3NF

¿Por qué 3NF no es suficiente aquí?

Porque existe una dependencia donde el determinante no identifica filas de forma única.

Esto puede generar inconsistencias si cambia la relación entre aula y materia.

Aplicar BCNF

Separamos la relación para que cada dependencia tenga una clave candidata como determinante.

Tabla Profesores_Materias

| Profesor | Materia | | -------- | ----------- | | Pérez | Matemáticas | | López | Historia |

Tabla Materias_Aulas

| Materia | Aula | | ----------- | ---- | | Matemáticas | 101 | | Historia | 202 |

Resultado tras aplicar BCNF

- Todas las dependencias tienen claves como determinantes
- Se elimina la ambigüedad
- El diseño es más robusto

Actividad práctica

Contexto

Tenemos un sistema sencillo para registrar pedidos de una tienda.

La base de datos fue creada rápidamente y no fue normalizada.

Nuestro objetivo es mejorar el diseño, no escribir SQL.

Tabla original (no normalizada)

| PedidoID | Cliente | Dirección | Producto | Cantidad | Precio | | -------- | ---------- | --------- | -------- | -------- | ------ | | 1 | Juan Pérez | Zona 1 | Laptop | 1 | 1200 | | 1 | Juan Pérez | Zona 1 | Teclado | 1 | 50 | | 2 | María López| Zona 5 | Celular | 2 | 800

|

Observación inicial

Toda la información está mezclada en una sola tabla:

- Datos del cliente
- Datos del pedido
- Datos del producto

Análisis inicial

Antes de normalizar, pensemos:

- ¿Qué información se repite?
- ¿Qué datos describen clientes?
- ¿Qué datos describen productos?
- ¿Qué datos describen pedidos?

Dependencias funcionales detectadas

A partir de la tabla observamos:

PedidoID → Cliente

PedidoID → Dirección

Producto → Precio

Cliente → Dirección

Anomalías presentes

Esta estructura genera:

- Actualización: cambiar una dirección implica varias filas
- Eliminación: eliminar un pedido puede borrar datos del cliente
- Inserción: no se puede registrar un cliente sin pedido

Paso 1: Primera Forma Normal (1NF)

Verificamos si la tabla cumple 1NF.

La tabla sí cumple 1NF porque:

- No hay listas en una celda
- Cada atributo es atómico
- Cada fila representa una ocurrencia

Cumplir 1NF no significa que la tabla esté bien diseñada.

Paso 2: Identificar la clave primaria

Un pedido puede incluir varios productos.

Por lo tanto:

- PedidoID por sí solo no identifica una fila
- Producto tampoco

La clave primaria correcta es:

(PedidoID, Producto)

Paso 3: Segunda Forma Normal (2NF)

Analizamos dependencias parciales.

Detectamos:

Producto → Precio

Precio depende solo del producto, no de todo el identificador compuesto.

Problema de no cumplir 2NF

Esto provoca que:

- El precio se repita innecesariamente
- Cambios de precio requieran múltiples actualizaciones
- Aumenten las inconsistencias

Aplicar 2NF

Separamos la información que no depende de toda la clave primaria.

Tabla Productos

| ProductoID | Nombre | Precio | | ---------- | ------- | ------ | | 101 | Laptop | 1200 | | 102 | Teclado | 50 | | 103 | Celular | 800 |

Tabla DetallePedido

| PedidoID | ProductoID | Cantidad | | -------- | ---------- | -------- | | 1 | 101 | 1 | | 1 | 102 | 1 | | 2 | 103 | 2 |

Resultado tras aplicar 2NF

- Eliminamos dependencias parciales
- La información del producto ya no se repite
- Cada tabla tiene un propósito claro

Paso 4: Tercera Forma Normal (3NF)

Ahora revisamos dependencias transitivas.

Observamos:

PedidoID → Cliente
Cliente → Dirección

Dirección no depende directamente del pedido.

Problema de no cumplir 3NF

Esto provoca:

- Repetición innecesaria de direcciones
- Riesgo de inconsistencias
- Dificultad para mantener datos del cliente

Aplicar 3NF

Separamos la información del cliente en una tabla independiente.

Tabla Clientes

| ClienteID | Nombre | Dirección | | --------- | ---------- | --------- | | 1 | Juan Pérez | Zona 1 | | 2 | María López | Zona 5 |

Tabla Pedidos

| PedidoID | ClienteID | | -------- | --------- | | 1 | 1 | | 2 | 2 |

Estructura final del sistema

Después de normalizar tenemos:

- Clientes
- Pedidos
- Productos
- DetallePedido

Beneficios del diseño final

- No hay redundancia
- No hay anomalías
- Cambios simples y seguros
- Diseño claro y mantenible
- Base sólida para DER y SQL