# Normalización en Bases de Datos

La normalización es un proceso fundamental en el diseño de bases de datos relacionales. Este proceso busca organizar los datos de manera eficiente y consistente, eliminando redundancias y dependencias innecesarias que pueden llevar a problemas como la inconsistencia de datos o la dificultad para realizar actualizaciones. La normalización se basa en un conjunto de reglas conocidas como *formas normales*, las cuales proporcionan un marco estructurado para optimizar el diseño de una base de datos.

## ¿Qué es la Normalización?

La normalización es un proceso sistemático utilizado en el diseño de bases de datos relacionales para organizar los datos de manera que se minimicen las redundancias y las anomalías en la inserción, actualización y eliminación de registros. Este proceso se realiza mediante la descomposición de tablas en estructuras más simples y bien definidas, siguiendo un conjunto de reglas llamadas *formas normales*.

### Objetivos de la Normalización

- **Eliminar redundancias**: Evitar que los mismos datos se repitan innecesariamente en múltiples registros.
- **Mejorar la integridad de los datos**: Garantizar que los datos sean consistentes y no contradictorios.
- **Facilitar las operaciones CRUD (Create, Read, Update, Delete)**: Simplificar las operaciones de inserción, consulta, actualización y eliminación de datos.
- **Optimizar el almacenamiento**: Reducir el espacio ocupado por los datos al eliminar duplicados.

## ¿Qué es la dependencia funcional y transitiva?

Son conceptos clave en el diseño de bases de datos relacionales, especialmente en el proceso de normalización.

### Dependencia Funcional

Una dependencia funcional ocurre cuando el valor de un atributo (o conjunto de atributos) determina de forma única el valor de otro atributo.

Se escribe así:

$$A \rightarrow B$$

Se lee: "A determina funcionalmente a B" o "B depende funcionalmente de A". Esto significa que para cada valor de A, existe exactamente un valor de B.

Ejemplo:

Supongamos una tabla *Empleado*:

| **dni** | **nombre** | **departamento** | **jefe_dept** |
|---|---|---|---|
| 12345678 | Ana Pérez | Ventas | Laura |
| 87654321 | Luis Gómez | Contabilidad | Carlos |

Aquí podemos observar:

- *dni* → *nombre* → El DNI determina el nombre (cada DNI tiene un solo nombre).
- *departamento* → *jefe_dept* → Cada departamento tiene un solo jefe.

Esto es una dependencia funcional válida.

**Importancia:**

Las dependencias funcionales ayudan a:

- Entender las relaciones entre atributos.
- Detectar redundancias.
- Aplicar las formas normales (1FN, 2FN, 3FN, etc.).

## Dependencia Transitiva

Una dependencia transitiva ocurre cuando un atributo depende de otro a través de un atributo intermedio.

Es decir:

$$\text{Si } A \rightarrow B \text{ y } B \rightarrow C,$$

$$\text{entonces } A \rightarrow C \text{ (por transitividad),}$$

y si C no depende directamente de A, entonces $A \rightarrow C$ es una dependencia transitiva.

Esta dependencia es problemática en 3FN (Tercera Forma Normal), porque puede generar redundancia e inconsistencia.

**Ejemplo dependencias 1:**

Misma tabla Empleado:

| **dni** | **nombre** | **departamento** | **jefe_dept** |
|---|---|---|---|
| 87654321 | Ana Pérez | Ventas | Laura |
| 11223344 | Luis Gómez | Contabilidad | Carlos |

**Dependencias:**

- *dni → departamento* - el empleado pertenece a un depto.
- *departamento → jefe_dept* - cada depto tiene un jefe.
- Por tanto: *dni → jefe_dept* pero de forma indirecta (transitiva).

Esto es una dependencia transitiva:

*dni → departamento → jefe_dept*

Entonces *dni → jefe_dept* (por transitividad)

**Problema:**

- El nombre del jefe se repite para cada empleado del mismo departamento → redundancia.
- Si cambia el jefe de Ventas, hay que actualizar múltiples filas → anomalía de actualización.

**Solución: Normalización (3FN)**

Se elimina la dependencia transitiva dividiendo la tabla:

Tabla 1: Empleado

| **dni** | **nombre** | **departamento** |
|---|---|---|
| 87654321 | Ana Pérez | Ventas |
| 11223344 | Luis Gómez | Contabilidad |

Tabla 2: Departamento

| **departamento** | **jefe_dept** |
|---|---|
| Ventas | Laura |
| Contabilidad | Carlos |

Ahora:

- No hay redundancia del jefe.
- Si cambia el jefe, solo se actualiza una fila.
- Se ha eliminado la dependencia transitiva.

Resumen

| **Concepto** | **Definición** | **Ejemplo** | **Deseable?** |
|---|---|---|---|
| Dependencia funcional | Un atributo determina a otro | *dni → nombre* | Sí, es base del modelo (BD Relacionales) |
| Dependencia transitiva | A → B → C, entonces A → C (indirectamente) | dni → departamento → jefe_dept | No, debe eliminarse en 3FN |

## Como Identificar las dependencias transitivas

Para detectar dependencias transitivas en tu base de datos, desglosa este proceso:

1. **Comprobar las Dependencias Funcionales de la Clave Primaria**: Empieza por identificar todas las dependencias funcionales en las que los atributos dependen directamente de la clave primaria. Este paso te ayuda a centrarte en las relaciones principales de tu tabla.
2. **Busca Dependencias Indirectas o transitivas**: A continuación, comprueba si algún atributo no clave (los que no forman parte de la clave primaria) depende de otros atributos no clave. Si descubres que un atributo no clave depende de otro, que a su vez depende de la clave primaria, habrás identificado una dependencia transitiva.
3. **Prueba de redundancia o anomalías**: Por último, piensa si esta dependencia indirecta introduce redundancia de datos o complica las actualizaciones.

**Ejemplo dependencias 2:**

- Tabla de Estudiantes
    - Imagina una tabla que almacena información de estudiantes y sus programas:
    - Tabla Original: estudiantes

| **EstudianteID** | **Nombre** | **Programa** | **Director** |
|---|---|---|---|
| 101 | Ana | Ingeniería Sistemas | Carlos |
| 102 | Luis | Marketing | María |
| 103 | Pedro | Ingeniería Sistemas | Carlos |

- Como hallamos la dependencia transitiva?
    - Identificamos la Clave primaria: EstudianteID.
    - Identificamos los atributos no clave: Nombre, Programa, Director.
- Aquí hay una dependencia transitiva:
    - Director depende de Programa (no directamente de EstudianteID).
    - Programa depende de EstudianteID.

Esto significa:

- Si cambia el Director de un Programa (ej: "Ingeniería Sistemas"), hay que actualizar todas las filas donde aparezca "Carlos".
- Hay redundancia: "Carlos" se repite para todos los estudiantes de "Ingeniería Sistemas".
- Solución: Eliminar la Dependencia Transitiva

Dividimos la tabla en dos:

- Tabla 1: estudiantes

| **EstudianteID** | **Nombre** | **Programa** |
|---|---|---|
| 101 | Ana | Ingeniería Sistemas |
| 102 | Luis | Marketing |
| 103 | Pedro | Ingeniería Sistemas |

- Tabla 2: programas

| **Programa** | **Director** |
|---|---|
| Ingeniería Sistemas | Carlos |
| Marketing | María |

- Relación:
  - estudiantes.Programa es una llave foránea que referencia a programas.Programas.
- ¿Por qué es importante?
  - Elimina redundancias: El director se almacena una sola vez por Programa.
  - Evita anomalías:
  - Si Carlos deja de ser director, solo se actualiza 1 fila en Director.
  - No se pueden insertar Directores sin Programas válido (integridad referencial).

https://www.datacamp.com/es/tutorial/transitive-dependency

# Formas Normales

Las formas normales son niveles progresivos de normalización que una base de datos puede alcanzar. Cada forma normal resuelve un conjunto específico de problemas relacionados con la redundancia y las dependencias entre los datos. A continuación, se describen las formas normales más importantes:

## Primera Forma Normal (1FN)

Una tabla está en la Primera Forma Normal si todos sus valores son atómicos, es decir, cada celda contiene un solo valor y no hay grupos repetidos.

**Requisitos:**

- Eliminar grupos repetidos, es decir, cada celda contiene un solo valor, no un conjunto de ellos .
    - Cada columna debe contener un solo valor. (Fecha + Hora, Celular 1+ Celular 2...)
- No debe de existir variación en el numero de atributos o columnas.
- Todos los datos de una columna deben de ser del mismo tipo.
- Cada fila debe ser única.
- Identificar cada conjunto de datos relacionados con una clave principal.
- Todos los atributos son atómicos. Un atributo es atómico si los elementos del dominio son simples e indivisibles.

**Ejemplo 1**:

Antes de aplicar 1FN:

| ID | Nombre | Teléfonos 1 | Teléfonos 2 |
|---|---|---|---|
| 1 | Juan | 123456 | 789012 |

Después de aplicar 1FN:

Tabla Cliente:

| ID | Nombre | Teléfono_1 |
|---|---|---|
| 1 | Juan | 123456 |

| 1 | Juan | 789012 |

**Ejemplo 2**:

Tabla sin normalizar

| ID | Nombre | Emails |
|---|---|---|
| 1 | Juan | juan@email.com, juan@web.com |
| 2 | Maria | maria@email.com, maria@empresa.net |

Después de aplicar 1FN:

Tabla Cliente

| ID | Nombre | Emails |
|---|---|---|
| 1 | Juan | juan@email.com |
| 1 | Juan | juan@web.com |
| 2 | Maria | maria@email.com |
| 2 | Maria | maria@empresa.net |

## Pregunta:

Cual tabla esta en la primera forma normal?

Tabla Estudiante 1:

| ID | Nombre | Asignaturas | Nacionalidad |
|---|---|---|---|
| 1 | Juan | Bases de datos, Logica 1, Fisica II | Colombia |
| 2 | Sofia | Digital 1, Algebra | Panama |
| 3 | Maria | Logica 1, Algebra, Electiva I | Argentia |

Tabla Estudiante 2:

| ID | Nombre | Asignatura 1 | Asignatura 2 | Asignatura 3 | Nacionalidad |
|---|---|---|---|---|---|
| 1 | Juan | Bases de datos | Logica 1 | Fisica II | Colombia |
| 2 | Sofia | Digital 1 | Algebra | | Panama |
| 3 | Maria | Logica 1 | Algebra | Electiva I | Argentina |

La respuesta es … ninguna!! Como deberia de quedar?

| ID | Nombre | Asignatura | Nacionalidad | |
|---|---|---|---|---|
| 1 | Juan | Bases de datos | Colombia | |
| 1 | Juan | Logica 1 | Colombia | |
| 1 | Juan | Fisica II | Colombia | |
| 2 | Sofia | Digital 1 | Panama | |
| 2 | Sofia | Algebra | Panama | |
| 3 | Maria | Logica 1 | Argentina | |
| 3 | Maria | Algebra | Argentina | |
| 3 | Maria | Electiva I | Argentina | |

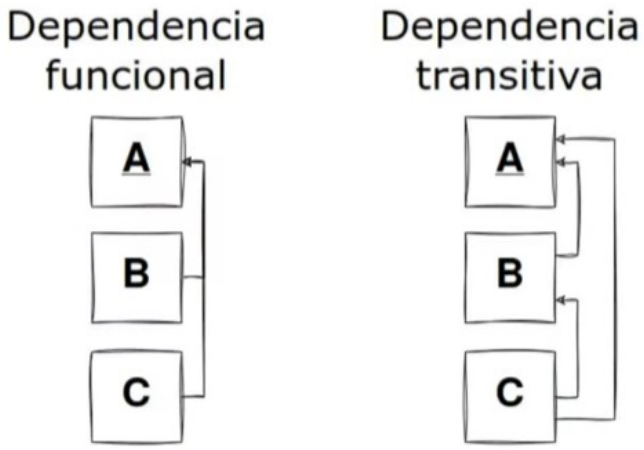
## Segunda Forma Normal (2FN)

Una tabla está en la Segunda Forma Normal si está en 1FN y todos los atributos no clave dependen completamente de la clave primaria.

**Requisitos**:

- La tabla debe estar en 1FN.
- Eliminar dependencias parciales (Una dependencia parcial ocurre cuando un atributo no clave depende solo de una parte de la clave primaria compuesta, y no de toda ella.).

  https://www.youtube.com/watch?v=XqBZ5t_1mko
- Identificar las dependencias funcionales. Si hay un elemento que no depende de la llave primaria, lo separamos en otra tabla.
- Todas las dependencias parciales se deben eliminar y separar dentro de sus propias tablas,
- Relacionar estas tablas con una clave externa.




Ejemplo:
Antes de aplicar 2FN:

| ID_Pedido | Producto | Cantidad | Cliente |
|---|---|---|---|
| 1 | Lápiz | 10 | Juan |
| 2 | Cuaderno | 5 | Diana |

Después de aplicar 2FN:

Tabla Pedidos:

| ID_Pedido (FK) | ID_Cliente (FK) | Cantidad |
|---|---|---|
| 1 | 1 | 10 |
| 2 | 2 | 5 |

Tabla Productos:

| ID_Producto (PK) | Producto |
|---|---|
| 1 | Lápiz |
| 2 | Cuaderno |

Tabla Clientes:

| ID_Cliente (PK) | Cliente |
|---|---|
| 1 | Juan |
| 2 | Diana |

Otra forma:

Después de aplicar 2FN:

Tabla Pedidos:

| ID_Pedido (FK) | Cliente |
|---|---|
| 1 | 1 |
| 2 | 2 |

Tabla Detalles_Pedido:

| ID_Pedido (PK) | Producto | Cantidad |
|---|---|---|
| 1 | Lápiz | 10 |
| 2 | Cuaderno | 5 |

Tabla Clientes:

| ID_Cliente (PK) | Cliente |
|---|---|
| 1 | Juan |
| 2 | Diana |

## Pregunta:

Como quedaría la tabla en la segunda forma normal?

Sigamos con la tabla de estudiantes:

| ID | Nombre | Asignatura | Nacionalidad |
|---|---|---|---|
| 1 | Juan | Bases de datos | Colombia |
| 1 | Juan | Logica 1 | Colombia |
| 1 | Juan | Fisica II | Colombia |
| 2 | Sofia | Digital 1 | Panama |
| 2 | Sofia | Algebra | Panama |
| 3 | Maria | Logica 1 | Argentina |
| 3 | Maria | Algebra | Argentina |
| 3 | Maria | Electiva I | Argentina |

Quedaría con tres tablas.

| ID | Nombre | Nacionalidad |
|---|---|---|
| 1 | Juan | Colombia |
| 2 | Sofia | Panama |
| 3 | Maria | Argentina |

| ID_Curso | Nombre |
|---|---|
| 001 | Algebra |
| 002 | Bases de datos |

| 003 | Logica 1 |
| --- | --- |
| 004 | Fisica II |
| 005 | Digital 1 |
| 006 | Electiva 1 |

| ID | ID_Curso |
| --- | --- |
| 1 | 002 |
| 1 | 003 |
| 1 | 004 |
| 2 | 001 |
| 2 | 005 |
| 3 | 003 |
| 3 | 001 |
| 3 | 006 |

## Tercera Forma Normal (3FN)

Una tabla está en la Tercera Forma Normal si está en 2FN y todos los atributos no clave dependen directamente de la clave primaria, sin dependencias transitivas.

**Requisitos:**

- La tabla debe estar en 2FN.
- Eliminar dependencias transitivas (atributos que dependen de otros atributos no clave) / Eliminar los campos que no dependen de la clave primaria. En otras palabras, las tablas solo deben contener datos sobre la entidad definida por la clave primaria.

**Ejemplo:**

Antes de aplicar 3FN:

| ID_Empleado | Nombre | Departamento | Jefe_Departamento |
|---|---|---|---|
| 1 | Ana | Ventas | Pedro |

Después de aplicar 3FN:

Tabla Empleados:

| ID_Empleado | Nombre | Departamento (FK) |
|---|---|---|
| 1 | Ana | Ventas |

Tabla Departamentos:

| Departamento (PK) | Jefe_Departamento |
|---|---|
| Ventas | Pedro |

## Pregunta

Como quedaria la tercera forma normal del problema de estudiantes?

| ID | Nombre | Nacionalidad |
|---|---|---|
| 1 | Juan | Colombia |
| 2 | Sofia | Panama |
| 3 | Maria | Argentina |

| ID_Curso | Nombre |
|---|---|
| 001 | Algebra |
| 002 | Bases de datos |
| 003 | Logica 1 |
| 004 | Fisica II |
| 005 | Digital 1 |
| 006 | Electiva 1 |

| ID | ID_Curso |
|---|---|
| 1 | 002 |
| 1 | 003 |
| 1 | 004 |
| 2 | 001 |
| 2 | 005 |
| 3 | 003 |
| 3 | 001 |
| 3 | 006 |

Quedaría de la siguiente manera :

Tabla Estudiantes:

| ID | Nombre | Nacionalidad |
|---|---|---|
| 1 | Juan | Colombia |
| 2 | Sofia | Panama |
| 3 | Maria | Argentina |

Tabla Nacionalidad

| ID_Nacionalidad | Nacionalidad |
|---|---|
| 1 | Colombia |
| 2 | Panama |
| 3 | Argentina |

Tabla Cursos

| ID_Curso | Nombre |
|---|---|
| 001 | Algebra |
| 002 | Bases de datos |
| 003 | Logica 1 |
| 004 | Fisica II |
| 005 | Digital 1 |
| 006 | Electiva 1 |

Tabla Asignacion

| ID | ID_Curso |
| --- | --- |
| 1 | 002 |
| 1 | 003 |
| 1 | 004 |
| 2 | 001 |
| 2 | 005 |
| 3 | 003 |
| 3 | 001 |
| 3 | 006 |

## Lectura

https://learn.microsoft.com/es-es/office/troubleshoot/access/database-normalization-description

## Forma Normal de Boyce-Codd (FNBC)

Es una versión más estricta de la 3FN que aborda casos específicos donde existen dependencias funcionales entre atributos que no son claves candidatas.

**Requisitos**:

- La tabla debe estar en 3FN.
- Todas las dependencias funcionales deben tener como determinante una clave candidata.

# Importancia de la Normalización

La normalización es crucial en el diseño de bases de datos porque:

1. Reduce la redundancia: Minimiza la cantidad de datos duplicados, lo que ahorra espacio y mejora la eficiencia.
2. Mejora la integridad de los datos: Evita inconsistencias al garantizar que los datos se almacenen de manera coherente.
3. Facilita el mantenimiento: Hace que las operaciones de inserción, actualización y eliminación sean más sencillas y menos propensas a errores.
4. Optimiza el rendimiento: Aunque puede aumentar el número de tablas, la normalización permite consultas más eficientes y precisas.

Desventajas de la Normalización

A pesar de sus beneficios, la normalización también tiene algunas desventajas:

- Mayor complejidad: El aumento en el número de tablas puede hacer que las consultas sean más complejas.
- Impacto en el rendimiento: En algunos casos, la normalización excesiva puede requerir uniones (joins) frecuentes, lo que puede afectar el rendimiento.
- No siempre es aplicable: En ciertos contextos, como bases de datos orientadas a análisis (OLAP - Procesamiento Analítico en Línea ), la denormalización puede ser preferible para mejorar la velocidad de las consultas.

# Denormalización en Bases de Datos

La *denormalización* es un proceso que implica la introducción intencionada de redundancia en una base de datos para mejorar el rendimiento y simplificar las consultas. A diferencia de la normalización, que busca eliminar redundancias y optimizar la estructura lógica de los datos, la denormalización se enfoca en mejorar la eficiencia operativa, especialmente en sistemas donde el rendimiento de lectura es crítico.

## ¿Qué es la Denormalización?

La denormalización consiste en combinar tablas o agregar datos redundantes en una base de datos previamente normalizada para reducir la necesidad de realizar uniones (joins) complejas y mejorar el rendimiento de las consultas. Este proceso puede implicar la duplicación de datos o la inclusión de atributos derivados que no son estrictamente necesarios desde un punto de vista lógico.

### Objetivos de la Denormalización

- Mejorar el rendimiento de las consultas, especialmente en sistemas de alto tráfico.
- Simplificar las consultas al reducir la necesidad de uniones entre múltiples tablas.
- Optimizar bases de datos orientadas a análisis (OLAP) o sistemas de informes.

### Cuándo Usar la Denormalización

La denormalización no es adecuada para todas las situaciones, pero puede ser útil en los siguientes escenarios:

- Sistemas de Lectura Intensiva: En aplicaciones donde las consultas de lectura son mucho más frecuentes que las operaciones de escritura (como en sistemas de informes o análisis de datos), la denormalización puede mejorar significativamente el rendimiento.
- Bases de Datos OLAP (Procesamiento Analítico en Línea): Las bases de datos OLAP están diseñadas para análisis complejos y consultas que involucran grandes volúmenes de datos. La denormalización ayuda a reducir la complejidad de las consultas y mejora los tiempos de respuesta.
- Reducción de Uniones Complejas: Si una consulta requiere unir múltiples tablas para obtener los datos necesarios, la denormalización puede simplificar esta tarea al almacenar los datos en una sola tabla.
- Optimización del Rendimiento: En sistemas con requisitos de tiempo real o bajo latencia, la denormalización puede reducir el tiempo necesario para recuperar datos.

## Ventajas de la Denormalización

- Mejora el Rendimiento de Lectura: Al reducir el número de uniones necesarias, las consultas pueden ejecutarse más rápidamente.
- Simplifica las Consultas:  Las consultas son más fáciles de escribir y mantener cuando los datos están en una sola tabla.
- Optimiza Sistemas de Informes: En sistemas de informes y análisis, la denormalización permite generar resultados más rápidos sin sacrificar la precisión.
- Reduce la Carga del Servidor: Al minimizar las operaciones de unión, se reduce la carga computacional sobre el servidor de bases de datos.

## Desventajas de la Denormalización

A pesar de sus beneficios, la denormalización también tiene inconvenientes que deben considerarse:

- Aumenta la Redundancia: La duplicación de datos puede llevar a inconsistencias si no se maneja adecuadamente.
- Mayor Costo de Mantenimiento: Las actualizaciones, inserciones y eliminaciones pueden volverse más complicadas debido a la redundancia.
- Incremento del Uso de Almacenamiento: La duplicación de datos consume más espacio en disco.
- Riesgo de Anomalías: Si no se implementan correctamente mecanismos de control, pueden surgir anomalías durante las operaciones de actualización.

**Ejemplo 1: Base de Datos Normalizada**

Supongamos que tenemos dos tablas normalizadas:

Tabla Clientes (c):

| ID_Cliente | Nombre |
|---|---|
| 1 | Juan |
| 2 | María |

Tabla Pedidos (p):

| ID_Pedido | ID_Cliente | Producto | Cantidad |
|---|---|---|---|
| 101 | 1 | Lápiz | 10 |
| 102 | 2 | Cuaderno | 5 |

Para obtener el nombre del cliente junto con sus pedidos, se necesita una consulta con una unión (JOIN):

*SELECT c.Nombre, p.Producto, p.Cantidad*

*FROM Clientes c*

*JOIN Pedidos p ON c.ID_Cliente = p.ID_Cliente;*

**Ejemplo 2: Base de Datos Denormalizada**

Podemos denormalizar las tablas combinando los datos en una sola tabla:

Tabla Denormalizada:

| ID_Pedido | Nombre | Producto | Cantidad |
|---|---|---|---|
| 101 | Juan | Lápiz | 10 |
| 102 | María | Cuaderno | 5 |

Ahora, la consulta es más simple:

SELECT Nombre, Producto, Cantidad

FROM Tabla_Denormalizada;

Aunque esto mejora el rendimiento de la consulta, introduce redundancia en el campo *Nombre*.

## Estrategias para Manejar la Denormalización

Para mitigar los riesgos asociados con la denormalización, se pueden adoptar las siguientes estrategias:

- Uso de Vistas Materializadas: Las vistas materializadas permiten almacenar resultados precalculados de consultas complejas sin modificar la estructura subyacente de la base de datos.
- Actualización Automática de Datos Redundantes: Utilizar disparadores (triggers) o procedimientos almacenados para garantizar que los datos redundantes se actualicen automáticamente cuando cambian los datos originales.
- Almacenamiento en Caché: Implementar mecanismos de caché para almacenar temporalmente los resultados de consultas frecuentes.
- Documentación Clara: Documentar cuidadosamente las decisiones de denormalización para facilitar el mantenimiento futuro.

La denormalización es una técnica poderosa que puede mejorar significativamente el rendimiento de las consultas en ciertos escenarios, especialmente en sistemas de lectura intensiva o análisis de datos. Sin embargo, debe usarse con precaución, ya que introduce redundancia y aumenta la complejidad del mantenimiento.

Como ingenieros de sistemas, es fundamental comprender tanto la normalización como la denormalización para tomar decisiones informadas sobre el diseño de bases de datos. El equilibrio entre estas técnicas depende de los requisitos específicos del sistema y de los compromisos aceptables en términos de rendimiento, almacenamiento y mantenibilidad.

**Referencias**:


- Date, C. J. (2004). *An Introduction to Database Systems*. Addison-Wesley.
- Kimball, R., & Ross, M. (2013). *The Data Warehouse Toolkit*. Wiley.
- Oracle Documentation. "Database Denormalization." [https://docs.oracle.com](https://docs.oracle.com)
- https://learn.microsoft.com/es-es/office/troubleshoot/access/database-normalization-description
- https://bookdown.org/paranedagarcia/database/modelamiento-de-datos.html

# Llaves en Bases de Datos

En bases de datos relacionales, las llaves (keys) son fundamentales para garantizar la integridad, evitar redundancias y establecer relaciones entre tablas. Este documento explica los diferentes tipos de llaves utilizadas en el diseño de bases de datos.

## Tipos de Llaves

### Llave Super (Super Key)

Conjunto de campos que incluye una llave candidata, pero puede tener atributos adicionales. En otras palabras, es cualquier conjunto de columnas que, combinadas, son únicas. Normalmente hay varias superclaves por tabla, y una misma columna puede ser compartida por varias. No son muy útiles por sí mismas, sino que sirven más como una herramienta mental para identificar claves candidatas

- Ejemplo: En una tabla *Persona*, la combinación (*dni, nombre*) es una superllave, aunque *dni* por sí solo ya es suficiente para identificar registros.

Ejemplo:

| **ID** | **DNI** | **Nombre** |
|---|---|---|
| 1234 | 8791581 | Jhon |
| 5879 | 1658758 | Carolina |

    - Posibles super llaves:
        - ID
        - DNI
        - ID – DNI
        - ID – DNI - Nombre

### Llave Candidata (Candidate Key)

Campo (o conjunto de campos) que podría ser llave primaria porque cumple con la unicidad. En otras palabras, es una superclave mínima: si se elimina alguna columna, deja de ser única. Normalmente, hay muchas menos claves candidatas que superclaves.

Es un campo o le menor combinación de campo que identifica de forma unica cada registro de la tabla, puede ser una o varias claves candidatas.

- Características:
    - Puede haber varias llaves candidatas en una tabla.
    - No admite valores duplicados ni *NULL*.

Ejemplo:

| ID | DNI | Nombre |
|---|---|---|
| 1234 | 8791581 | Jhon |
| 5879 | 1658758 | Carolina |

    - Posibles llaves Candidatas:
        - ID
        - DNI

## Llave Primaria (Primary Key - PK)

Campo (o conjunto de campos) que identifica de manera única cada registro en una tabla.

- Características:
    - No admite valores duplicados.
    - No puede ser *NULL*.
    - Una tabla solo puede tener una llave primaria.
- Ejemplo:

```
sql
CREATE TABLE Estudiante (
    id_estudiante INT PRIMARY KEY,
    nombre VARCHAR(50)
);
```

## Llave Foranea (Foreign Key - FK)

Campo que establece una *relación* con la llave primaria de otra tabla.

- Propósito: Mantener la integridad referencial.
- Ejemplo:

```
sql
CREATE TABLE Matricula (
    id_matricula INT PRIMARY KEY,
    id_estudiante INT,
    curso VARCHAR(50),
    FOREIGN KEY (id_estudiante) REFERENCES Estudiante(id_estudiante)
);
```

## Llave Compuesta (Composite Key)

Llave primaria formada por dos o más campos para garantizar unicidad.

- Ejemplo:

```
sql
CREATE TABLE Pedido_Producto (
    id_pedido INT,
    id_producto INT,
```

```
    cantidad INT,
    PRIMARY KEY (id_pedido, id_producto)
);
```

### Llave Alterna (Alternate Key)

Cualquier llave candidata que no fue elegida como llave primaria.

- Ejemplo: En `Empleado`, si `id_empleado` es PK, entonces `dni` es una llave alterna.

## Comparación entre Llaves

| Tipo de Llave | Unicidad | Puede ser NULL | Relaciones |
|---|---|---|---|
| Primaria (PK) | Sí | No | Si (referenciada por FK) |
| Foránea (FK) | Depende* | Sí** | Sí (referencia a PK) |
| Candidata | Sí | No | No (potencial PK) |
| Super Llave | Sí (redundante) | Depende | No |
| Compuesta | Sí (combinación) | No | Si |

* Puede repetirse a menos que sea parte de una PK compuesta.
** Depende del diseño (puede ser `NULL` si la relación es opcional).

- Las llaves son esenciales para la *integridad* y *organización* de los datos.
- La elección del tipo de llave depende del modelo de datos y los requerimientos del sistema.
- Las relaciones entre tablas se gestionan principalmente con PK y FK.

# Cardinalidad N:M con Llaves Foraneas

Ejemplo:

Analicemos estas 3 entidades:

**Entidad alumno:**

| dni | nom |
|---|---|
| 0101 | Marcela Hernandez |
| 0987 | Carlos Molina |

SQL

```
CREATE TABLE alumno (
    dni INT,
    nom VARCHAR(50),
    PRIMARY KEY (dni)
);
```

**Entidad asignatura:**

| cod | asignatura |
|---|---|
| 101 | Math |
| 201 | English |

SQL

```
CREATE TABLE asignatura (
        cod INT,
        asignatura VARCHAR(50),
        PRIMARY KEY (cod)
);
```

**Entidad matricula:**

| alu | asi |
|---|---|
| 0101 | 101 |
| 0101 | 201 |
| 0987 201 | |

SQL

```
CREATE TABLE matricula (
        alu INT,        -- Debe ser INT para coincidir con alumno.dni
        asi INT,        -- Coincide con asignatura.cod
        PRIMARY KEY (alu, asi),
        FOREIGN KEY (alu) REFERENCES alumno(dni),
        FOREIGN KEY (asi) REFERENCES asignatura(cod) -- Llave foránea añadida
);
```