# **BASES DE DATOS RELACIONALES**

## **1. Introducción**

En la era digital, el manejo eficiente de la información es clave para el éxito de cualquier organización. Las bases de datos relacionales surgieron como una solución estructurada y organizada para almacenar grandes volúmenes

de datos, permitiendo su acceso, modificación y gestión de manera eficiente. Su fundamento teórico, desarrollado por Edgar F. Codd, ha marcado un antes y un después en la administración de datos, estableciendo principios que siguen vigentes hoy en día.

Este informe explorará en detalle las 12 reglas de Codd, el proceso de normalización y el modelo relacional, proporcionando ejemplos prácticos que ilustran su aplicación en el mundo real. Además, una línea de tiempo enriquecida con los hitos más importantes en la evolución de las bases de datos relacionales, destacando su impacto y relevancia en la actualidad.

## **2. Las 12 Reglas de Codd**

Edgar F. Codd definió 12 reglas que establecen los principios fundamentales de las bases de datos relacionales:

* 1. **Regla de la información:** Toda la información debe representarse de forma explícita en tablas.
* Ejemplo: Una empresa de ventas almacena información de sus clientes. En lugar de mantener archivos separados o documentos de texto, la empresa utiliza una tabla llamada "Clientes" con las siguientes columnas: ID\_Cliente, Nombre, Apellido, Correo, Teléfono. Cada cliente tiene un registro único con su información estructurada en filas y columnas.
  1. **Regla del acceso garantizado:** Cada dato debe ser accesible de forma única mediante una clave primaria.
* **Ejemplo:** En un sistema de gestión de empleados, cada trabajador tiene un ID\_Empleado único en la tabla "Empleados". Si se quiere obtener el salario de un empleado específico y asegurarse que acceda al registro correcto, se empleara el ID\_Empleado; se puede realizar la consulta SQL:
* SELECT Salario FROM Empleados WHERE ID\_Empleado = 12345
  1. **Tratamiento sistemático de valores nulos:** Los valores nulos deben gestionarse de manera uniforme.
* **Ejemplo:** En una tabla "Pedidos", el campo Fecha\_Entrega puede estar vacío (nulo) si el pedido aún no ha sido enviado. Para manejar estos casos en SQL, se pueden usar funciones como COALESCE(Fecha\_Entrega, 'Pendiente') para mostrar "Pendiente" cuando el campo es nulo.
  1. **Catálogo en línea basado en el modelo relacional:** La estructura de la base de datos debe ser accesible mediante consultas SQL.
* Ejemplo: Para conocer las tablas existentes en una base de datos MySQL, se puede ejecutar la consulta SHOW TABLES;. Esto permite obtener información sobre la estructura de la base de datos directamente desde el sistema.
  1. **Regla del sub-lenguaje de datos:** Debe existir un lenguaje estándar para manipular datos.
* Ejemplo: Para permite obtener los nombres de los clientes que residen en Lima; en consulta SQL es:
* SELECT Nombre FROM Clientes WHERE Ciudad = 'Lima'
  1. **Actualización de vistas:** Las vistas deben ser modificables si la estructura lo permite.
* Ejemplo: Se crea una vista llamada ClientesActivos que solo muestra clientes que no han sido dados de baja:
* CREATE VIEW ClientesActivos AS
* SELECT ID\_Cliente, Nombre, Apellido FROM Clientes WHERE Estado = 'Activo';

Si un cliente cambia de estado, se puede actualizar directamente en la vista con:

* UPDATE ClientesActivos SET Estado = 'Inactivo' WHERE ID\_Cliente = 101;

### **Alto Nivel de Inserción, actualización y cancelación:** permite gestionar datos de forma rápida y sencilla mediante SQL. Esto optimiza el proceso, reduce errores y garantiza la integridad de la información sin necesidad de programar operaciones complejas manualmente.

* Ejemplo: para agregar un nuevo cliente, utilizamos INSERT INTO Clientes VALUES (3, 'Ana', 'Arequipa'), lo que añade a Ana a la tabla. Si queremos actualizar la ciudad de María, ejecutamos UPDATE Clientes SET Ciudad = 'Lima' WHERE ID = 2, cambiando su ubicación a Lima. Finalmente, si deseamos eliminar a Juan, usamos DELETE FROM Clientes WHERE ID = 1, eliminándolo de la base de datos.

### **Independencia física de los datos:** Los cambios en la estructura física de almacenamiento (como cambiar el tipo de disco, reorganizar archivos o mejorar el hardware) no deben afectar la lógica de las aplicaciones que interactúan con la base de datos.

* Ejemplo: Una empresa almacena datos en un disco HDD y luego los migra a un SSD para mayor velocidad. A pesar del cambio, siguen funcionando igual, sin necesidad de modificar el código.
* SELECT Nombre FROM Clientes WHERE ID = 5;

### **Independencia Lógica de los Datos:** La independencia lógica de los datos significa que podemos modificar la estructura de la base de datos (como agregar o dividir tablas) sin afectar las aplicaciones que acceden a la información. Esto permite hacer cambios sin alterar el funcionamiento del sistema.

* Ejemplo: Si una empresa separa las direcciones de los clientes en otra tabla, las consultas pueden seguir funcionando con un **JOIN**, sin afectar las aplicaciones. Esto muestra que los datos pueden reorganizarse sin cambiar su acceso.

### **Independencia de la Integridad:** La independencia de la integridad significa que las reglas que garantizan la validez, exactitud y coherencia de los datos deben gestionarse dentro de la base de datos y no depender del código de las aplicaciones. Esto asegura que, sin importar qué aplicación acceda a los datos, estos siempre cumplan con las restricciones establecidas.

* Ejemplo: Supongamos que tenemos una tabla **Usuarios** y queremos asegurarnos de que los correos electrónicos sean únicos. En lugar de validar esto desde la aplicación, la base de datos define una restricción

CREATE TABLE Usuarios (

ID INT PRIMARY KEY,

Nombre VARCHAR(50),

Correo VARCHAR(100) UNIQUE

);

Si una aplicación intenta registrar un correo duplicado, la base de datos lo rechazará automáticamente, sin necesidad de que el programa realice validaciones adicionales.

### **Independencia de Distribución:** Permite que los datos estén en varios servidores sin afectar su acceso. Las aplicaciones pueden consultar la información sin preocuparse por su ubicación física.

* Ejemplo: Una empresa almacena datos de clientes europeos en un servidor en Europa y de América en otro, pero las consultas siguen funcionando igual.

### **Regla de la No Subversión:** Ningún método de acceso debe evitar las reglas de integridad de la base de datos.

* Ejemplo: Si una restricción impide eliminar pedidos pendientes, no debería ser posible hacerlo ni con comandos directos.

1. **Normalización de Base de Datos**

La normalización organiza los datos para evitar redundancias y asegurar la integridad. Sus formas son:

### **1NF: Primera Forma Normal:** La Primera Forma Normal (1NF) establece que los datos deben estar organizados en una tabla donde cada columna contenga valores atómicos (no divisibles) y cada fila sea única. Esto elimina la repetición de grupos de datos y garantiza que cada celda contenga un solo valor.

* **Ejemplo:** Una tabla “Pedidos” mal diseñada puede almacenar varios productos en una misma celda:

|  |  |  |
| --- | --- | --- |
| **ID\_Pedido** | **Cliente** | **Productos** |
| 1 | Juan | Pan, Leche |
| 2 | María | Arroz |

Para cumplir con 1NF, se deben separar los productos en filas individuales:

|  |  |  |
| --- | --- | --- |
| **ID\_Pedido** | **Cliente** | **Producto** |
| 1 | Juan | Pan |
| 1 | Juan | Leche |
| 2 | María | Arroz |

### **2NF: Segunda Forma Normal:** La Segunda Forma Normal (2NF) exige que una base de datos esté en 1NF y que todos los atributos dependan completamente de la clave primaria. Esto evita la duplicación de datos y mejora la integridad de la información.

* Ejemplo: Si en una tabla “pedidos” se repiten datos de clientes:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID\_Pedido** | **Cliente** | **Dirección** | **Producto** |
| 1 | Juan | Calle 123 | Pan |
| 1 | Juan | Calle 123 | Leche |

Para cumplir con 2NF, se separan los datos en dos tablas:

**Tabla Clientes**

|  |  |  |
| --- | --- | --- |
| **ID\_Cliente** | **Nombre** | **Dirección** |
| 1 | Juan | Calle 123 |

**Tabla Pedidos**

|  |  |  |
| --- | --- | --- |
| **ID\_Pedido** | **ID\_Cliente** | **Producto** |
| 1 | 1 | Pan |
| 1 | 1 | Leche |

### **3NF: Tercera Forma Normal**: La Tercera Forma Normal (3NF) requiere que una base de datos esté en 2NF y que no haya dependencias transitivas, es decir, que los atributos dependan solo de la clave primaria y no de otros atributos no clave.

* Ejemplo: Si en una tabla de “Clientes” la ciudad depende del código postal, se genera redundancia:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID\_Cliente** | **Nombre** | **Código Postal** | **Ciudad** |
| 1 | Juan | 15001 | Lima |
| 2 | María | 20002 | Arequipa |

Para cumplir con 3NF, se crea una tabla separada para los códigos postales:

**Tabla Clientes**

|  |  |  |
| --- | --- | --- |
| **ID\_Cliente** | **Nombre** | **Código Postal** |
| 1 | Juan | 15001 |
| 2 | María | 20002 |

**Tabla Ciudades**

|  |  |
| --- | --- |
| **Código Postal** | **Ciudad** |
| 15001 | Lima |
| 20002 | Arequipa |

## **4. Línea de Tiempo: Evolución de las Bases de Datos Relacionales**

* **1970 - El nacimiento del modelo relacional**: Edgar F. Codd, matemático e investigador de IBM, publica su artículo pionero "A Relational Model of Data for Large Shared Data Banks", estableciendo los fundamentos teóricos de las bases de datos relacionales. Su propuesta introduce el concepto de almacenamiento basado en tablas y relaciones.
* **1974 - IBM desarrolla System R**: IBM desarrolla el primer prototipo de base de datos relacional, System R, que introduce SQL (Structured Query Language) como lenguaje de consulta estándar.
* **1986 - SQL se convierte en un estándar**: El American National Standards Institute (ANSI) e International Organization for Standardization (ISO) oficializan SQL como el lenguaje estándar para bases de datos relacionales.
* **1990s - La era de los grandes sistemas de bases de datos**: Surgen bases de datos comerciales como Oracle, MySQL y Microsoft SQL Server, que revolucionan el almacenamiento y procesamiento de datos en empresas de todo el mundo.
* **2000s - La evolución hacia la escalabilidad y la nube**: Se desarrollan bases de datos relacionales en la nube, como Amazon RDS y Google Cloud SQL, permitiendo escalabilidad global y acceso distribuido.
* **Actualidad - Bases de datos híbridas y la Inteligencia Artificial**: Se combinan bases de datos relacionales y NoSQL, optimizando el rendimiento en aplicaciones modernas como Big Data, Machine Learning e Inteligencia Artificial.