Tabla de contenido

[1. Introducción 3](#_Toc192183038)

[Datos 3](#_Toc192183039)

[Información 3](#_Toc192183040)

[Evolución de las Bases de Datos desde los Ficheros 3](#_Toc192183041)

[2. Perfiles Profesionales en el Trabajo con Datos 6](#_Toc192183042)

[3. Sistemas de Gestión de Bases de Datos (SGBD) 9](#_Toc192183043)

[4. El Principio ACID en Bases de Datos 11](#_Toc192183044)

[5. Modelización de Datos y Etapas de Construcción del Modelo de Datos 14](#_Toc192183045)

[**6.** El Modelo Relacional en Bases de Datos 16](#_Toc192183046)

[7. Conceptos básicos de bases de datos relacionales 18](#_Toc192183047)

[Tablas y registros 18](#_Toc192183048)

[Claves primarias y claves foráneas 18](#_Toc192183049)

[Relaciones entre tablas 18](#_Toc192183050)

[Normalización 18](#_Toc192183051)

[Consultas SQL 19](#_Toc192183052)

[SQL 19](#_Toc192183053)

[DML (Data Manipulation Language) 20](#_Toc192183054)

[DDL (Data Definition Language) 20](#_Toc192183055)

[DCL (Data Control Language) 21](#_Toc192183056)

[8. TIPOS DE DATOS 21](#_Toc192183057)

[Tipos de Datos Numéricos 21](#_Toc192183058)

[Tipos de Datos de Caracteres 22](#_Toc192183059)

[Tipos de Datos de Fecha y Hora 22](#_Toc192183060)

[Tipos de Datos LOB (Large Objects) 22](#_Toc192183061)

[Tipos de Datos Especiales 23](#_Toc192183062)

[9. COMANDOS DDL 24](#_Toc192183063)

[Creación de una tabla 24](#_Toc192183064)

[Modificación de la tabla 24](#_Toc192183065)

[Añadir una nueva columna 24](#_Toc192183066)

[Modificar el tipo de un campo 24](#_Toc192183067)

[Eliminar una columna 24](#_Toc192183068)

[Truncar la tabla 24](#_Toc192183069)

[Borrar la tabla 25](#_Toc192183070)

[10. Constraints 25](#_Toc192183071)

[1. PRIMARY KEY (Clave primaria) 25](#_Toc192183072)

[Clave primaria compuesta: 25](#_Toc192183073)

[FOREIGN KEY (Clave foránea) 26](#_Toc192183074)

[NOT NULL 26](#_Toc192183075)

[UNIQUE 26](#_Toc192183076)

[CHECK 27](#_Toc192183077)

[DEFAULT 27](#_Toc192183078)

[IDENTITY (Autoincremento) 28](#_Toc192183079)

[Ejemplo Completo con todas las restricciones 28](#_Toc192183080)

[Resumen de restricciones 29](#_Toc192183081)

[11. Comandos DML (Data Manipulation Language) 30](#_Toc192183082)

[INSERT - Insertar Datos 30](#_Toc192183083)

[UPDATE - Modificar Datos 31](#_Toc192183084)

[DELETE - Eliminar Datos 31](#_Toc192183085)

[SELECT - Consultar Datos 32](#_Toc192183086)

[MERGE - Insertar o Actualizar Datos 33](#_Toc192183087)

[12. Funciones en Oracle 34](#_Toc192183088)

[1. Funciones de cadena (String) 34](#_Toc192183089)

[2. Funciones numéricas 35](#_Toc192183090)

[3. Funciones de fecha y hora 35](#_Toc192183091)

[4. Funciones de conversión 36](#_Toc192183092)

[5. Funciones de agregación 36](#_Toc192183093)

[6. Funciones condicionales 37](#_Toc192183094)

[13. Relaciones entre Tablas en Oracle SQL 37](#_Toc192183095)

[Relación Uno a Uno (1:1) 37](#_Toc192183096)

[Relación Uno a Muchos (1:M) 39](#_Toc192183097)

[Relación Muchos a Muchos (M:M) 40](#_Toc192183098)

# Introducción

En el ámbito de las bases de datos, es fundamental entender la diferencia entre **datos** e **información**, ya que son conceptos relacionados, pero no son lo mismo.

## Datos

Los **datos** son hechos, valores o registros en bruto, sin contexto ni significado específico por sí solos. Pueden ser números, caracteres, símbolos o cualquier otro tipo de representación sin procesar. Un dato por sí mismo no comunica nada útil hasta que es interpretado correctamente.

Ejemplo de datos:

* "Juan Pérez"
* "35"
* "TI"
* "5000.00"

Por sí solos, estos elementos no tienen mucho sentido, pero cuando se organizan adecuadamente, pueden convertirse en información útil.

## Información

La **información** es el resultado del procesamiento y organización de los datos de manera que adquieren significado y utilidad. Cuando los datos se estructuran dentro de un sistema, como una base de datos, pueden responder preguntas y ayudar en la toma de decisiones.

Ejemplo de información:

* "Juan Pérez tiene 35 años, trabaja en el departamento de TI y su salario es de 5000.00."

Aquí, los datos individuales han sido organizados en una estructura que tiene sentido y puede ser utilizada para análisis o gestión.

## Evolución de las Bases de Datos desde los Ficheros

Antes del desarrollo de los sistemas de bases de datos, la información se almacenaba y gestionaba en **ficheros** o archivos individuales dentro de los sistemas informáticos. Estos ficheros contenían datos en formatos como **texto plano, archivos CSV, o estructuras propias de aplicaciones específicas**. Sin embargo, con el tiempo, se hicieron evidentes las **limitaciones y desventajas** de este método, lo que llevó al desarrollo de las bases de datos.

**1. Almacenamiento en Ficheros: Primeros Métodos de Gestión de Datos**

En los primeros sistemas informáticos, los datos se almacenaban en **ficheros secuenciales**, lo que significaba que, para acceder a un dato específico, el programa tenía que leer el archivo completo hasta encontrar la información requerida. Con el tiempo, se introdujeron ficheros indexados y de acceso directo, que permitían búsquedas más rápidas, pero aun así presentaban problemas en términos de **integridad, redundancia y mantenimiento**.

Ejemplo de almacenamiento en un fichero de texto:

1, Juan Pérez, 35, TI, 5000.00

2, Ana García, 29, Finanzas, 4500.00

3, Carlos Gómez, 42, Recursos Humanos, 5200.00

Este método podía funcionar en pequeñas aplicaciones, pero a medida que crecía la cantidad de datos, surgían problemas importantes.

**2. Desventajas del almacenamiento basado en ficheros**

Aunque los ficheros eran útiles en los primeros sistemas, tenían varias limitaciones que afectaban la eficiencia y la integridad de los datos.

**2.1. Redundancia y Duplicación de Datos**

Cada fichero almacenaba datos de manera independiente, lo que provocaba duplicación de información. Por ejemplo, si una empresa tenía un archivo de empleados y otro de nómina, ambos podrían contener los mismos datos de los empleados, como su nombre y cargo.

Problema:

* Si un empleado cambia de departamento, habría que modificar su información en múltiples ficheros.
* Si un fichero no se actualiza correctamente, se generan **inconsistencias**.

**2.2. Inconsistencia de Datos**

Dado que la misma información podía almacenarse en varios ficheros, si los datos no se actualizaban en todos ellos, se producían **inconsistencias**.

Ejemplo:

* Un empleado cambia su salario, pero la actualización solo se realiza en el fichero de recursos humanos y no en el de nómina. Esto provocaría discrepancias en los cálculos de pago.

**2.3. Dificultad en el Acceso a los Datos**

El acceso a los datos en ficheros era **poco flexible**. Para recuperar información, se necesitaban programas específicos que leyeran los archivos de una forma determinada.

Problemas:

* Se requería **conocer la estructura del fichero** para extraer la información.
* Si los datos estaban en ficheros grandes, **las búsquedas eran lentas** porque el programa debía recorrer todo el archivo.
* No era posible realizar consultas complejas sin escribir código adicional.

**2.4. Falta de Integridad y Seguridad**

Los ficheros no proporcionaban mecanismos sólidos para **controlar la integridad** de los datos.

Ejemplo de problema de integridad:

* Un empleado es eliminado del fichero de empleados, pero sigue apareciendo en el fichero de nómina.

Además, no había controles estrictos sobre quién podía acceder a los datos, lo que aumentaba el riesgo de errores o accesos no autorizados.

**2.5. Dificultad para Manejar Grandes Volúmenes de Datos**

A medida que las organizaciones crecían, la cantidad de información almacenada en ficheros aumentaba significativamente. Esto llevaba a problemas como:

* **Dificultad para buscar datos rápidamente.**
* **Archivos demasiado grandes y propensos a corrupción.**
* **Problemas de compatibilidad entre diferentes programas que usaban los mismos ficheros.**

**2.6. No Existía un Lenguaje Estándar para Consultar Datos**

Cada sistema de ficheros requería un programa específico para leer, modificar o eliminar datos, lo que dificultaba la portabilidad y la estandarización del acceso a la información.

**3. Surgimiento de las Bases de Datos**

Para resolver estos problemas, en la década de **1960 y 1970** surgieron los **primeros sistemas de bases de datos**, que ofrecían soluciones organizadas y eficientes para el almacenamiento de información.

Las bases de datos introdujeron conceptos como:

* **Estructuración de los datos en tablas relacionales**.
* **Eliminación de redundancia mediante normalización**.
* **Control de concurrencia y acceso multiusuario**.
* **Lenguajes de consulta estandarizados como SQL**.

Ejemplo de una consulta SQL que permite acceder a la información de manera flexible sin necesidad de recorrer todo un fichero manualmente:

SELECT nombre, salario FROM empleados WHERE departamento = 'TI';

Esto resolvió los problemas de acceso rápido, integridad y consistencia de los datos.

Las bases de datos existen para almacenar **datos** de manera organizada y permitir su transformación en **información** útil. A través de la gestión eficiente de los datos, las bases de datos permiten la consulta, el análisis y la toma de decisiones en diversos entornos, como empresas, instituciones educativas y sistemas gubernamentales.

Este proceso de transformación de datos en información se realiza mediante consultas, procesamiento de registros y generación de reportes, lo que hace que las bases de datos sean una herramienta esencial en la administración de información.

# Perfiles Profesionales en el Trabajo con Datos

En el mundo del **Big Data**, la **inteligencia de negocio** y la **gestión de bases de datos**, existen diferentes perfiles profesionales especializados en **almacenamiento, análisis, procesamiento y visualización de datos**. Cada uno tiene funciones y habilidades específicas que permiten gestionar datos de manera eficiente para la toma de decisiones en empresas y organizaciones.

**1. Administrador de Bases de Datos (DBA - Database Administrator)**

**Descripción**

El **Administrador de Bases de Datos (DBA)** es responsable de la instalación, configuración, mantenimiento y optimización de bases de datos en un entorno empresarial.

**Responsabilidades**

* Administrar bases de datos relacionales y NoSQL.
* Garantizar la integridad, seguridad y disponibilidad de los datos.
* Configurar copias de seguridad y recuperación ante fallos.
* Optimizar el rendimiento de consultas SQL.
* Gestionar accesos y permisos de usuarios.

**Habilidades necesarias**

* SQL avanzado (Oracle, MySQL, PostgreSQL, SQL Server).
* Manejo de herramientas de respaldo y recuperación.
* Monitorización y ajuste de rendimiento.
* Conocimientos en seguridad de bases de datos.

**2. Ingeniero de Datos (Data Engineer)**

**Descripción**

El **Ingeniero de Datos** se encarga de diseñar, construir y mantener infraestructuras de datos escalables para la recopilación, almacenamiento y procesamiento de grandes volúmenes de información.

**Responsabilidades**

* Desarrollar **pipelines de datos** para la integración de múltiples fuentes.
* Optimizar el almacenamiento en **data lakes** o **data warehouses**.
* Implementar procesos ETL (Extract, Transform, Load).
* Trabajar con bases de datos distribuidas y en la nube.

**Habilidades necesarias**

* Lenguajes de programación como **Python, Java o Scala**.
* Bases de datos **SQL y NoSQL** (PostgreSQL, MongoDB, Cassandra).
* Herramientas Big Data (Apache Spark, Hadoop, Kafka).
* Servicios en la nube (AWS, Google Cloud, Azure).

**3. Analista de Datos (Data Analyst)**

**Descripción**

El **Analista de Datos** interpreta la información almacenada en bases de datos para responder preguntas de negocio mediante estadísticas, visualizaciones y reportes.

**Responsabilidades**

* Realizar consultas SQL para analizar datos.
* Crear dashboards y reportes con herramientas BI.
* Identificar tendencias y patrones en los datos.
* Apoyar la toma de decisiones basada en datos.

**Habilidades necesarias**

* Lenguajes como **SQL y Python/R** para análisis de datos.
* Herramientas BI (Tableau, Power BI, Looker).
* Métodos estadísticos y de machine learning básico.
* Manipulación de datos con **Pandas, NumPy**.

**4. Científico de Datos (Data Scientist)**

**Descripción**

El **Científico de Datos** aplica modelos de inteligencia artificial y estadística avanzada para descubrir patrones y hacer predicciones a partir de los datos.

**Responsabilidades**

* Desarrollar modelos de machine learning y deep learning.
* Limpiar y transformar datos para su análisis.
* Utilizar técnicas de NLP, visión por computadora, etc.
* Experimentar con diferentes algoritmos para mejorar predicciones.

**Habilidades necesarias**

* Programación en **Python o R**.
* Frameworks de Machine Learning (Scikit-Learn, TensorFlow, PyTorch).
* Análisis estadístico y modelado predictivo.
* Procesamiento de grandes volúmenes de datos.

**5. Ingeniero de Machine Learning (ML Engineer)**

**Descripción**

El **Ingeniero de Machine Learning** implementa y despliega modelos de IA en entornos productivos, optimizando su rendimiento y escalabilidad.

**Responsabilidades**

* Entrenar y optimizar modelos de machine learning.
* Desplegar modelos en producción con APIs o contenedores.
* Trabajar con arquitecturas en la nube para procesamiento de IA.
* Monitorizar y mejorar el rendimiento de los modelos.

**Habilidades necesarias**

* Programación avanzada en **Python o Java**.
* Integración de modelos con APIs REST y servidores.
* Uso de Kubernetes, Docker y frameworks en la nube.

# Sistemas de Gestión de Bases de Datos (SGBD)

Un **Sistema de Gestión de Bases de Datos (SGBD)** es un software diseñado para **crear, gestionar y administrar** bases de datos de manera eficiente y segura. Permite a los usuarios almacenar, consultar, modificar y eliminar datos de forma estructurada, garantizando su **integridad, consistencia y disponibilidad**.

**1. Funciones principales de un SGBD**

Un SGBD proporciona un conjunto de herramientas y funcionalidades clave para la gestión de bases de datos:

1. **Definición de Datos**
   * Permite definir la estructura de la base de datos (tablas, relaciones, restricciones).
   * Se realiza a través del **Lenguaje de Definición de Datos (DDL)** con comandos como CREATE TABLE.
2. **Manipulación de Datos**
   * Permite insertar, actualizar, eliminar y consultar datos dentro de la base de datos.
   * Se realiza mediante el **Lenguaje de Manipulación de Datos (DML)** con comandos como INSERT, UPDATE, DELETE, SELECT.
3. **Control de Acceso y Seguridad**
   * Gestiona permisos y restricciones para proteger la información.
   * Define qué usuarios pueden leer, modificar o administrar los datos.
4. **Control de Concurrencia y Multiusuario**
   * Permite que múltiples usuarios accedan a la base de datos al mismo tiempo sin conflictos.
   * Evita problemas como la **lectura de datos inconsistentes** o **actualizaciones erróneas** mediante bloqueos y transacciones.
5. **Mantenimiento de la Integridad de los Datos**
   * Garantiza que los datos sean correctos y cumplan con las reglas establecidas.
   * Se implementan restricciones como **PRIMARY KEY**, **FOREIGN KEY**, **CHECK**, **NOT NULL**.
6. **Respaldo y Recuperación de Datos**
   * Permite realizar copias de seguridad y recuperar datos en caso de fallos o pérdida de información.
7. **Optimización y Rendimiento**
   * Usa índices, caché y otras técnicas para mejorar la velocidad de acceso a los datos.

**2. Tipos de SGBD**

Existen varios tipos de sistemas de gestión de bases de datos según la forma en que organizan y manejan la información:

**2.1. SGBD Relacionales (RDBMS)**

* Organizan los datos en **tablas** con filas y columnas.
* Utilizan **SQL (Structured Query Language)** para consultar y manipular datos.
* Permiten relaciones entre tablas mediante **claves primarias y foráneas**.
* Ejemplos: **Oracle, MySQL, PostgreSQL, Microsoft SQL Server**.

**2.2. SGBD No Relacionales (NoSQL)**

* Diseñados para manejar grandes volúmenes de datos **no estructurados o semiestructurados**.
* Existen varios tipos:
  + **Bases de datos de documentos** (MongoDB).
  + **Bases de datos clave-valor** (Redis).
  + **Bases de datos en grafos** (Neo4j).
  + **Bases de datos en columnas** (Apache Cassandra).

**2.3. SGBD Jerárquicos**

* Organizan los datos en una estructura de **árbol** con relaciones padre-hijo.
* Son rápidos para acceder a datos relacionados de forma jerárquica.
* Ejemplo: **IBM IMS (Information Management System)**.

**2.4. SGBD de Red**

* Similar a los jerárquicos, pero permite que un registro tenga **múltiples relaciones** con otros.
* Se usaban en sistemas antiguos antes de la aparición de los SGBD relacionales.

**3. Ventajas de un SGBD sobre los archivos tradicionales**

Antes de los SGBD, los datos se almacenaban en **ficheros**, lo que tenía muchas limitaciones. Los SGBD resolvieron estos problemas proporcionando:

* **Menor redundancia y duplicación de datos**
* **Mayor seguridad y control de acceso**
* **Mejor integridad y consistencia de la información**
* **Acceso más rápido y eficiente a los datos**
* **Soporte para múltiples usuarios al mismo tiempo**
* **Facilidad para hacer copias de seguridad y recuperar información**

**4. Ejemplos de SGBD Populares**

| **SGBD** | **Tipo** | **Uso principal** |
| --- | --- | --- |
| **Oracle** | Relacional (SQL) | Empresas, sistemas financieros |
| **MySQL** | Relacional (SQL) | Aplicaciones web, e-commerce |
| **PostgreSQL** | Relacional (SQL) | Sistemas empresariales, ciencia de datos |
| **SQL Server** | Relacional (SQL) | Empresas, grandes volúmenes de datos |
| **MongoDB** | NoSQL (documentos) | Aplicaciones con datos no estructurados |
| **Redis** | NoSQL (clave-valor) | Caché y almacenamiento en memoria |
| **Neo4j** | NoSQL (grafos) | Redes sociales, análisis de relaciones |

# El Principio ACID en Bases de Datos

El **principio ACID** es un conjunto de propiedades fundamentales que garantizan la confiabilidad y la integridad de las transacciones en un **Sistema de Gestión de Bases de Datos (SGBD)**. Estas propiedades aseguran que una base de datos funcione correctamente, incluso en situaciones de fallos del sistema o concurrencia de múltiples usuarios.

ACID es un acrónimo de cuatro propiedades clave:

* **Atomicidad (Atomicity)**
* **Consistencia (Consistency)**
* **Aislamiento (Isolation)**
* **Durabilidad (Durability)**

Cada una de estas propiedades es crucial para garantizar que las transacciones en la base de datos sean seguras y confiables.

**1. Atomicidad (Atomicity)**

La **atomicidad** garantiza que una **transacción** se realice de forma **completa** o **no se ejecute en absoluto**. Es decir, si una transacción incluye múltiples operaciones, todas deben completarse correctamente o ninguna debe aplicarse a la base de datos.

**Ejemplo:**

Supongamos que un cliente quiere transferir **100€** de su cuenta bancaria a otra.  
Esta transacción consta de dos operaciones:

1. **Restar 100€** de la cuenta del cliente A.
2. **Sumar 100€** a la cuenta del cliente B.

Si la primera operación se realiza, pero la segunda falla (por un corte de energía, por ejemplo), el sistema debe **revertir la primera operación** para evitar inconsistencias.

BEGIN TRANSACTION;

UPDATE cuentas SET saldo = saldo - 100 WHERE id = 1;

UPDATE cuentas SET saldo = saldo + 100 WHERE id = 2;

COMMIT; -- Confirma la transacción si todo funciona

Si algo falla antes de COMMIT, la transacción se **revierte automáticamente (ROLLBACK)**, dejando la base de datos en su estado original.

**2. Consistencia (Consistency)**

La **consistencia** garantiza que la base de datos pase de un **estado válido a otro estado válido**, cumpliendo todas las reglas y restricciones definidas.

**Ejemplo:**

Si tenemos una **restricción de clave foránea**, no deberíamos poder agregar una orden de compra para un cliente que no existe.

INSERT INTO ordenes (id, cliente\_id, total) VALUES (1, 999, 250.00);

Si cliente\_id = 999 no existe en la tabla clientes, la base de datos **rechazará la operación** para mantener la consistencia.

**3. Aislamiento (Isolation)**

El **aislamiento** asegura que múltiples transacciones pueden ejecutarse simultáneamente **sin interferirse entre sí**. Si varias transacciones intentan acceder a los mismos datos al mismo tiempo, el sistema debe garantizar que cada una de ellas se ejecute como si fuera la única operación en la base de datos.

**Ejemplo:**

Dos clientes intentan comprar el **último producto disponible** en una tienda online al mismo tiempo.

1. Cliente A lee que hay **1 unidad disponible**.
2. Cliente B también lee que hay **1 unidad disponible**.
3. Cliente A compra el producto y se reduce el stock a **0**.
4. Cliente B intenta comprar el producto, pero **ya no hay stock**.

Sin aislamiento, ambos clientes podrían haber comprado el producto simultáneamente, generando un **stock negativo**.

Para evitarlo, se pueden usar **niveles de aislamiento de transacciones**, como SERIALIZABLE, que bloquea el acceso a los datos hasta que una transacción se complete.

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN TRANSACTION;

UPDATE productos SET stock = stock - 1 WHERE id = 10;

COMMIT;

Esto impide que otra transacción modifique el stock hasta que la primera haya terminado.

**4. Durabilidad (Durability)**

La **durabilidad** garantiza que, una vez que una transacción ha sido confirmada (COMMIT), sus cambios **persisten en la base de datos**, incluso en caso de fallos del sistema.

**Ejemplo:**

Si un banco confirma una transferencia y el sistema se apaga inmediatamente después, el dinero no puede desaparecer ni volver a la cuenta original. Los cambios deben permanecer guardados.

Los SGBD logran la durabilidad mediante:

* **Registros de transacción (redo log)** que guardan los cambios antes de confirmarlos.
* **Copia de seguridad y recuperación** en caso de fallos del sistema.

# Modelización de Datos y Etapas de Construcción del Modelo de Datos

La **modelización de datos** es el proceso de diseñar la estructura de los datos en un sistema de información. Su objetivo es representar de manera lógica y estructurada cómo los datos serán almacenados, organizados y relacionados dentro de una base de datos.

Este proceso es esencial para garantizar que la base de datos cumpla con los requerimientos de la organización, optimice el acceso a la información y garantice su integridad.

**1. ¿Qué es un Modelo de Datos?**

Un **modelo de datos** es una abstracción que describe la estructura de los datos, sus relaciones y reglas de integridad. Se representa mediante diagramas y descripciones formales que sirven de guía para la construcción de la base de datos.

Existen **tres niveles** de modelos de datos:

1. **Modelo Conceptual:** Representa los datos a alto nivel sin detalles técnicos.
2. **Modelo Lógico:** Define la estructura de los datos en un SGBD específico sin implementar detalles físicos.
3. **Modelo Físico:** Implementa el modelo lógico en la base de datos real con detalles de almacenamiento.

**2. Etapas de Construcción del Modelo de Datos**

El desarrollo de un modelo de datos sigue varias etapas, cada una con un nivel creciente de detalle y especificidad.

**2.1. Modelo Conceptual de Datos**

* Es la primera etapa de la modelización.
* Se centra en identificar **entidades**, **atributos** y **relaciones** sin definir cómo serán almacenados.
* Se representa mediante el **Modelo Entidad-Relación (E-R)**.

**Ejemplo:**

Un sistema de gestión de empleados puede tener estas entidades y relaciones:

* **Entidad:** Empleado
  + Atributos: ID\_Empleado, Nombre, Edad, Salario
* **Entidad:** Departamento
  + Atributos: ID\_Departamento, Nombre\_Departamento
* **Relación:** Trabaja\_en
  + Un empleado pertenece a un departamento.

**Representación con un diagrama E-R:**

[Empleado] -----(Trabaja\_en)------ [Departamento]

**2.2. Modelo Lógico de Datos**

* Es una representación más detallada basada en el modelo conceptual.
* Se define la estructura de los datos de acuerdo con un tipo de base de datos (relacional, NoSQL, orientada a grafos, etc.).
* En una base de datos relacional, se expresa en **tablas, claves primarias y claves foráneas**.

**Ejemplo en SQL (Modelo Relacional)**

CREATE TABLE empleados (

id\_empleado NUMBER(5) PRIMARY KEY,

nombre VARCHAR2(50) NOT NULL,

edad NUMBER(3),

salario NUMBER(10,2),

id\_departamento NUMBER(3),

CONSTRAINT fk\_departamento FOREIGN KEY (id\_departamento) REFERENCES departamentos(id\_departamento)

);

CREATE TABLE departamentos (

id\_departamento NUMBER(3) PRIMARY KEY,

nombre\_departamento VARCHAR2(50) NOT NULL

);

* Se crean **tablas** (empleados, departamentos).
* Se definen **claves primarias** y **claves foráneas** para establecer relaciones.

**2.3. Modelo Físico de Datos**

* Se implementa en un SGBD específico (Oracle, MySQL, SQL Server, PostgreSQL, etc.).
* Define detalles técnicos como:
  + Tipos de datos específicos (VARCHAR2, NUMBER, DATE).
  + Índices para optimizar consultas.
  + Estrategias de particionamiento y replicación.
* Se optimiza la estructura de almacenamiento para mejorar el rendimiento.

**Ejemplo con optimización en Oracle**

CREATE TABLE empleados (

id\_empleado NUMBER(5) PRIMARY KEY,

nombre VARCHAR2(50) NOT NULL,

edad NUMBER(3) CHECK (edad > 18), -- Restricción de integridad

salario NUMBER(10,2),

id\_departamento NUMBER(3),

CONSTRAINT fk\_departamento FOREIGN KEY (id\_departamento) REFERENCES departamentos(id\_departamento)

)

TABLESPACE empleados\_ts; -- Especifica un tablespace para mejorar rendimiento

* Se agregan restricciones (CHECK, FOREIGN KEY).
* Se define el TABLESPACE para optimizar almacenamiento.

# El Modelo Relacional en Bases de Datos

El **modelo relacional** es un modelo de organización de datos basado en el uso de **tablas** para representar la información y sus relaciones. Fue propuesto por **Edgar F. Codd** en 1970 y se convirtió en el modelo dominante en la gestión de bases de datos, gracias a su **simplicidad, flexibilidad y facilidad de consulta mediante SQL**.

**1. Características del Modelo Relacional**

1. **Datos organizados en tablas (relaciones)**
   * La información se almacena en estructuras llamadas **tablas** o **relaciones**.
   * Cada tabla representa una **entidad** (como empleados, clientes, productos).
2. **Uso de tuplas y atributos**
   * **Tuplas**: Filas que representan **instancias de una entidad** (ejemplo: un cliente específico).
   * **Atributos**: Columnas que representan **propiedades** de la entidad (ejemplo: nombre, edad).
3. **Cada tabla tiene una clave primaria**
   * Una columna (o combinación de columnas) que identifica de manera única cada fila.
   * Ejemplo: en una tabla de empleados, la clave primaria puede ser id\_empleado.
4. **Uso de claves foráneas para representar relaciones**
   * Una clave foránea en una tabla establece una relación con otra tabla.
   * Ejemplo: en la tabla de empleados, id\_departamento puede ser clave foránea que hace referencia a departamentos(id\_departamento).
5. **Independencia lógica y física**
   * La estructura de la base de datos es independiente del almacenamiento físico.
   * Los usuarios pueden acceder a los datos sin preocuparse por cómo se almacenan.
6. **Uso del álgebra relacional y SQL**
   * El modelo relacional permite consultas mediante **lenguajes de consulta estructurados**, como **SQL**.
   * Ejemplo de consulta SQL:
   * SELECT nombre, salario FROM empleados WHERE departamento\_id = 1;

**Ventajas del Modelo Relacional**

1. **Estructura simple y flexible:** Usa tablas fácilmente comprensibles.
2. **Reducción de redundancia:** Mediante normalización, evita duplicación de datos.
3. **Integridad de los datos:** Garantiza que las relaciones sean consistentes.
4. **Acceso eficiente a la información:** Mediante **SQL**, se pueden realizar consultas rápidas y precisas.
5. **Facilidad de escalabilidad:** Permite agregar nuevas tablas y relaciones sin afectar la estructura existente.
6. **Soporte para transacciones:** Cumple con el **principio ACID** (Atomicidad, Consistencia, Aislamiento, Durabilidad).

El **modelo relacional** es el estándar más utilizado en bases de datos gracias a su estructura organizada basada en **tablas, relaciones y claves**. Permite **gestionar datos de manera eficiente**, reducir redundancias y garantizar la integridad de la información. Gracias al uso de **SQL**, es posible consultar y manipular la información de forma sencilla y potente.

# Conceptos básicos de bases de datos relacionales

Una **base de datos relacional** es un sistema organizado de almacenamiento de datos que utiliza tablas para representar la información y las relaciones entre diferentes conjuntos de datos. Se basa en el **modelo relacional**, propuesto por Edgar F. Codd en 1970.

## Tablas y registros

La información en una base de datos relacional se organiza en **tablas** (también llamadas relaciones). Cada tabla está compuesta por **filas** y **columnas**.

* Una **fila** representa un **registro** o **tupla**, que es una instancia concreta de la entidad representada en la tabla.
* Una **columna** representa un **atributo** o **campo**, que es una propiedad específica de la entidad.

Ejemplo: una tabla llamada **Clientes** podría tener las siguientes columnas: ID\_Cliente, Nombre, Apellido, Email.

## Claves primarias y claves foráneas

Una **clave primaria** es un campo (o combinación de campos) que identifica de manera única cada fila dentro de una tabla. No puede contener valores nulos ni repetidos.  
Una **clave foránea** es un campo en una tabla que hace referencia a la clave primaria de otra tabla. Se utiliza para establecer relaciones entre tablas.

Ejemplo: en una base de datos de pedidos, la tabla **Pedidos** podría tener una clave foránea llamada ID\_Cliente que hace referencia a la clave primaria de la tabla **Clientes**.

## Relaciones entre tablas

Las bases de datos relacionales permiten establecer conexiones entre diferentes tablas. Existen tres tipos principales de relaciones:

* **Uno a uno (1:1)**: Un registro de la tabla A se asocia con un solo registro de la tabla B.
* **Uno a muchos (1:M)**: Un registro de la tabla A se asocia con varios registros de la tabla B, pero cada registro de B solo está vinculado a un registro de A.
* **Muchos a muchos (M:N)**: Varios registros de la tabla A pueden estar relacionados con varios registros de la tabla B. Para gestionar esta relación, se usa una **tabla intermedia**.

Ejemplo: Un cliente puede hacer muchos pedidos, pero cada pedido pertenece a un único cliente (relación 1:M).

## Normalización

Es el proceso de estructurar una base de datos para minimizar la redundancia y mejorar la integridad de los datos. Se logra a través de las **formas normales**, que son reglas que guían el diseño de las tablas.

Las tres primeras formas normales más comunes son:

1. **Primera Forma Normal (1NF)**: Se eliminan los grupos repetitivos, asegurando que cada campo contenga solo un valor.
2. **Segunda Forma Normal (2NF)**: Se eliminan dependencias parciales, asegurando que cada campo dependa completamente de la clave primaria.
3. **Tercera Forma Normal (3NF)**: Se eliminan dependencias transitivas, asegurando que los campos no dependan de otros campos que no sean clave primaria.

## Consultas SQL

Para interactuar con una base de datos relacional, se utiliza **SQL (Structured Query Language)**. Algunas operaciones básicas incluyen:

* **SELECT**: Recupera datos de una tabla.

SELECT Nombre, Apellido FROM Clientes WHERE ID\_Cliente = 1;

* **INSERT**: Agrega un nuevo registro.

INSERT INTO Clientes (ID\_Cliente, Nombre, Apellido, Email) VALUES (1, 'Ana', 'García', 'ana@email.com');

* **UPDATE**: Modifica registros existentes.

UPDATE Clientes SET Email = 'nuevo@email.com' WHERE ID\_Cliente = 1;

* **DELETE**: Elimina registros.

DELETE FROM Clientes WHERE ID\_Cliente = 1;

Integridad referencial y restricciones

Las bases de datos relacionales aseguran la integridad de los datos mediante restricciones como:

* **NOT NULL**: Evita que un campo tenga valores nulos.
* **UNIQUE**: Garantiza que no haya valores duplicados en una columna.
* **CHECK**: Establece condiciones que deben cumplir los valores de un campo.
* **FOREIGN KEY**: Define claves foráneas para asegurar la relación correcta entre tablas.

Índices y optimización

Un **índice** mejora la velocidad de búsqueda en una base de datos, permitiendo acceso más rápido a los registros. Se pueden crear índices sobre columnas frecuentemente consultadas.  
Ejemplo:  
CREATE INDEX idx\_nombre ON Clientes (Nombre);

El diseño de una base de datos relacional eficiente implica conocer estos conceptos y aplicarlos adecuadamente según las necesidades del sistema.

## SQL

En bases de datos relacionales, SQL se divide en varios subconjuntos según el tipo de operación que realizan sobre los datos. Los tres más importantes son **DML (Data Manipulation Language)**, **DDL (Data Definition Language)** y **DCL (Data Control Language)**.

### DML (Data Manipulation Language)

Incluye comandos que permiten manipular los datos almacenados en la base de datos. Son operaciones que afectan a los registros de las tablas sin modificar su estructura.

Comandos DML en Oracle XE

1. **INSERT**: Inserta nuevos registros en una tabla.

INSERT INTO Clientes (ID\_Cliente, Nombre, Apellido, Email)

VALUES (1, 'Ana', 'García', 'ana@email.com');

1. **UPDATE**: Modifica registros existentes en una tabla.

UPDATE Clientes

SET Email = 'nuevo@email.com'

WHERE ID\_Cliente = 1;

1. **DELETE**: Elimina registros de una tabla.

DELETE FROM Clientes WHERE ID\_Cliente = 1;

1. **SELECT**: Consulta datos almacenados en la base de datos.

SELECT \* FROM Clientes;

### DDL (Data Definition Language)

Incluye comandos que definen la estructura de la base de datos, como la creación y modificación de tablas, índices y restricciones.

Comandos DDL en Oracle XE

1. **CREATE**: Crea objetos como bases de datos, tablas e índices.

CREATE TABLE Clientes (

ID\_Cliente NUMBER PRIMARY KEY,

Nombre VARCHAR2(50),

Apellido VARCHAR2(50),

Email VARCHAR2(100) UNIQUE

);

1. **ALTER**: Modifica la estructura de una tabla (agregar, modificar o eliminar columnas).

ALTER TABLE Clientes ADD Telefono VARCHAR2(20);

1. **DROP**: Elimina completamente una tabla o cualquier otro objeto de la base de datos.

DROP TABLE Clientes;

1. **TRUNCATE**: Elimina todos los registros de una tabla sin afectar su estructura y sin posibilidad de reversión.

TRUNCATE TABLE Clientes;

### DCL (Data Control Language)

Incluye comandos que gestionan los permisos y el control de acceso a los datos.

Comandos DCL en Oracle XE

1. **GRANT**: Otorga permisos a un usuario para realizar acciones específicas.

GRANT SELECT, INSERT ON Clientes TO usuario1;

1. **REVOKE**: Revoca los permisos previamente concedidos a un usuario.

REVOKE INSERT ON Clientes FROM usuario1;

Cada uno de estos conjuntos de comandos cumple una función específica en la gestión de una base de datos Oracle XE. Mientras que **DML** se usa para manipular los datos, **DDL** define la estructura de la base de datos y **DCL** gestiona los permisos y la seguridad.

# TIPOS DE DATOS

Oracle XE soporta una amplia variedad de tipos de datos para manejar diferentes tipos de información. Se pueden clasificar en las siguientes categorías principales:

## Tipos de Datos Numéricos

Se utilizan para almacenar valores numéricos, tanto enteros como decimales.

* **NUMBER(p, s)**: Número de precisión variable.
  + p: Número total de dígitos (máximo 38).
  + s: Número de dígitos a la derecha del punto decimal.
  + Ejemplo: NUMBER(10,2) almacena hasta 10 dígitos, con 2 después del decimal.
* **INTEGER**: Equivalente a NUMBER(38,0). Almacena enteros sin decimales.
  + Ejemplo: INTEGER almacena valores como 100, -50, 2500.
* **FLOAT (p)**: Número de punto flotante.
  + p define la precisión en bits.
  + Ejemplo: FLOAT(10) almacena números con precisión de 10 dígitos.
* **BINARY\_FLOAT** y **BINARY\_DOUBLE**: Números de punto flotante en precisión simple y doble.
  + BINARY\_FLOAT: Más rápido pero menos preciso.
  + BINARY\_DOUBLE: Mayor precisión para cálculos científicos.

## Tipos de Datos de Caracteres

Se utilizan para almacenar texto.

* **CHAR(n)**: Cadena de longitud fija de n caracteres (máximo 2000 bytes).
  + Ejemplo: CHAR(10) almacena exactamente 10 caracteres.
* **VARCHAR2(n)**: Cadena de longitud variable, con un máximo de n caracteres (hasta 4000 bytes).
  + Ejemplo: VARCHAR2(100) almacena hasta 100 caracteres sin desperdiciar espacio.
* **NCHAR(n)** y **NVARCHAR2(n)**: Versiones Unicode de CHAR y VARCHAR2 para almacenar caracteres multibyte.

## Tipos de Datos de Fecha y Hora

Se utilizan para almacenar fechas y horas.

* **DATE**: Almacena fecha y hora con precisión de segundos.
  + Ejemplo: DATE almacena valores como 2025-02-17 14:30:00.
* **TIMESTAMP [(n)]**: Similar a DATE, pero permite mayor precisión en los segundos (n define la cantidad de dígitos en la fracción de segundo, hasta 9).
  + Ejemplo: TIMESTAMP(3) almacena 2025-02-17 14:30:15.123.
* **TIMESTAMP WITH TIME ZONE**: Almacena fecha, hora y zona horaria.
  + Ejemplo: TIMESTAMP '2025-02-17 14:30:00 -05:00'.
* **TIMESTAMP WITH LOCAL TIME ZONE**: Similar al anterior, pero ajusta automáticamente la zona horaria según el usuario.
* **INTERVAL YEAR TO MONTH**: Almacena diferencias de tiempo en años y meses.
  + Ejemplo: INTERVAL '3-6' YEAR TO MONTH (3 años y 6 meses).
* **INTERVAL DAY TO SECOND**: Almacena diferencias de tiempo en días, horas, minutos y segundos.
  + Ejemplo: INTERVAL '5 12:30:45' DAY TO SECOND (5 días, 12 horas, 30 minutos y 45 segundos).

## Tipos de Datos LOB (Large Objects)

Se utilizan para almacenar grandes volúmenes de datos como documentos, imágenes o vídeos.

* **CLOB (Character Large Object)**: Almacena hasta 128 TB de texto.
  + Ejemplo: CLOB se usa para almacenar documentos XML o JSON largos.
* **BLOB (Binary Large Object)**: Almacena datos binarios (imágenes, vídeos, audio).
  + Ejemplo: BLOB se usa para guardar archivos multimedia.
* **NCLOB**: Similar a CLOB, pero para caracteres Unicode.
* **BFILE**: Referencia un archivo binario almacenado fuera de la base de datos.

## Tipos de Datos Especiales

Se usan en aplicaciones específicas.

* **RAW(n)**: Almacena datos binarios de longitud fija (máximo 2000 bytes).
  + Ejemplo: RAW(16) se usa para almacenar valores hash o claves encriptadas.
* **ROWID**: Representa la ubicación física de una fila en la base de datos.
  + Ejemplo: ROWID se usa en consultas internas para identificar registros de manera única.
* **UROWID**: Similar a ROWID, pero para tablas organizadas con índices.

# COMANDOS DDL

## Creación de una tabla

CREATE TABLE empleados (

id NUMBER(5) PRIMARY KEY, -- Clave primaria, número de hasta 5 dígitos

nombre VARCHAR2(50) NOT NULL, -- Campo obligatorio

fecha\_nacimiento DATE, -- Tipo fecha

salario NUMBER(10,2) CHECK (salario > 0), -- Número con dos decimales y restricción

departamento\_id NUMBER(3) REFERENCES departamentos(id) -- Clave foránea

);

Explicación:

* id es la clave primaria.
* nombre es un **campo obligatorio** (NOT NULL).
* fecha\_nacimiento es de tipo **DATE**.
* salario tiene una **restricción** (CHECK) que impide valores negativos.
* departamento\_id es una **clave foránea** (REFERENCES), que enlaza con la tabla departamentos.

## Modificación de la tabla

### Añadir una nueva columna

ALTER TABLE empleados ADD email VARCHAR2(100);

Esto agrega un nuevo campo email a la tabla empleados.

### Modificar el tipo de un campo

ALTER TABLE empleados MODIFY salario NUMBER(12,2);

Aumenta el tamaño del campo salario de NUMBER(10,2) a NUMBER(12,2).

### Eliminar una columna

ALTER TABLE empleados DROP COLUMN email;

Elimina la columna email.

## Truncar la tabla

TRUNCATE TABLE empleados;

Explicación:

* Elimina todos los registros de la tabla, pero mantiene la estructura.
* No se puede revertir (no genera registros en el log de transacciones).

## Borrar la tabla

DROP TABLE empleados;

Explicación:

* Elimina completamente la tabla y su estructura.
* Se pierden los datos y la tabla deja de existir en la base de datos.

# Constraints

En **Oracle XE**, las limitaciones (**constraints**) permiten definir reglas sobre los datos almacenados en las tablas para garantizar su integridad. A continuación, te detallo los principales tipos de restricciones que puedes aplicar a los campos de una tabla, con ejemplos explicados.

## PRIMARY KEY (Clave primaria)

Garantiza que cada fila en la tabla tiene un valor único y no nulo en la columna o columnas definidas.

Ejemplo:

CREATE TABLE empleados (

id NUMBER(5) PRIMARY KEY, -- No permite valores duplicados ni nulos

nombre VARCHAR2(50) NOT NULL

);

* La columna id no puede contener valores repetidos ni NULL.
* Se puede definir sobre una única columna o múltiples columnas (clave primaria compuesta).

### Clave primaria compuesta:

CREATE TABLE asistencia (

empleado\_id NUMBER(5),

fecha DATE,

PRIMARY KEY (empleado\_id, fecha) -- Ambos campos juntos deben ser únicos

);

## FOREIGN KEY (Clave foránea)

Asegura la integridad referencial entre dos tablas, impidiendo que se inserten valores en una tabla si no existen en la tabla referenciada.

Ejemplo:

CREATE TABLE departamentos (

id NUMBER(3) PRIMARY KEY,

nombre VARCHAR2(50) NOT NULL

);

CREATE TABLE empleados (

id NUMBER(5) PRIMARY KEY,

nombre VARCHAR2(50) NOT NULL,

departamento\_id NUMBER(3),

CONSTRAINT fk\_departamento FOREIGN KEY (departamento\_id)

REFERENCES departamentos(id) ON DELETE CASCADE

);

* departamento\_id en empleados debe existir en departamentos.
* **ON DELETE CASCADE**: Si un departamento es eliminado, sus empleados también se eliminan.

## NOT NULL

Impide que una columna almacene valores NULL.

Ejemplo:

CREATE TABLE clientes (

id NUMBER(5) PRIMARY KEY,

nombre VARCHAR2(100) NOT NULL -- Obligatorio, no puede ser NULL

);

* La columna nombre **siempre debe tener un valor** al insertar un nuevo registro.

## UNIQUE

Asegura que todos los valores en una columna sean únicos, pero permite valores NULL.

Ejemplo:

CREATE TABLE productos (

id NUMBER(5) PRIMARY KEY,

codigo VARCHAR2(20) UNIQUE, -- No se pueden repetir valores

nombre VARCHAR2(50) NOT NULL

);

* codigo **no puede repetirse**, pero puede tener NULL.

Diferencia entre PRIMARY KEY y UNIQUE:

* Una tabla solo puede tener una **PRIMARY KEY**.
* Puede tener múltiples **UNIQUE** en diferentes columnas.

## CHECK

Permite definir una condición lógica que los valores de una columna deben cumplir.

Ejemplo:

CREATE TABLE empleados (

id NUMBER(5) PRIMARY KEY,

salario NUMBER(10,2) CHECK (salario > 0), -- Solo valores positivos

edad NUMBER(2) CHECK (edad BETWEEN 18 AND 65) -- Solo edades válidas

);

* **salario** debe ser mayor que 0.
* **edad** debe estar entre 18 y 65 años.

## DEFAULT

Define un valor por defecto si no se especifica otro al insertar datos.

Ejemplo:

CREATE TABLE pedidos (

id NUMBER(5) PRIMARY KEY,

estado VARCHAR2(20) DEFAULT 'Pendiente', -- Si no se inserta, usa 'Pendiente'

fecha\_pedido DATE DEFAULT SYSDATE -- Toma la fecha actual

);

* Si no se proporciona estado, se asigna 'Pendiente'.
* fecha\_pedido usa la fecha actual del sistema.

## IDENTITY (Autoincremento)

Genera valores secuenciales automáticamente (en Oracle 12c+).

Ejemplo:

CREATE TABLE clientes (

id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY, -- Autoincrementado

nombre VARCHAR2(100) NOT NULL

);

* id se genera automáticamente sin necesidad de usar SEQUENCES.

Alternativa en versiones antiguas de Oracle (como XE 11g):

CREATE SEQUENCE clientes\_seq START WITH 1 INCREMENT BY 1;

CREATE TABLE clientes (

id NUMBER(5) PRIMARY KEY,

nombre VARCHAR2(100) NOT NULL

);

INSERT INTO clientes (id, nombre) VALUES (clientes\_seq.NEXTVAL, 'Juan Pérez');

## Ejemplo Completo con todas las restricciones

CREATE TABLE empleados (

id NUMBER(5) GENERATED ALWAYS AS IDENTITY PRIMARY KEY, -- Autoincremental y clave primaria

nombre VARCHAR2(50) NOT NULL, -- No puede ser NULL

email VARCHAR2(100) UNIQUE, -- No se puede repetir

salario NUMBER(10,2) CHECK (salario > 0), -- No permite valores negativos

fecha\_contratacion DATE DEFAULT SYSDATE, -- Por defecto, la fecha actual

departamento\_id NUMBER(3),

CONSTRAINT fk\_departamento FOREIGN KEY (departamento\_id) REFERENCES departamentos(id) ON DELETE CASCADE

);

## Resumen de restricciones

| Restricción | Función |
| --- | --- |
| PRIMARY KEY | Garantiza valores únicos y no nulos en una columna o conjunto de columnas. |
| FOREIGN KEY | Vincula una columna con otra tabla, asegurando integridad referencial. |
| NOT NULL | Impide que una columna tenga valores nulos. |
| UNIQUE | Garantiza que los valores de la columna sean únicos (permite NULL). |
| CHECK | Define condiciones lógicas para los valores de una columna. |
| DEFAULT | Especifica un valor predeterminado si no se proporciona otro. |
| IDENTITY | Genera valores secuenciales automáticamente (versión 12c+) |

# Comandos DML (Data Manipulation Language)

Los comandos **DML (Data Manipulation Language)** permiten manipular los datos almacenados en las tablas de una base de datos Oracle. Son los siguientes:

1. **INSERT** – Insertar datos en una tabla.
2. **UPDATE** – Modificar datos existentes en una tabla.
3. **DELETE** – Eliminar datos específicos de una tabla.
4. **MERGE** – Combinar datos de dos tablas en una sola operación.
5. **SELECT** – Consultar datos (aunque se considera parte de **DQL - Data Query Language**, lo incluimos porque permite visualizar los cambios hechos con DML).

## INSERT - Insertar Datos

El comando INSERT permite agregar registros a una tabla.

**Ejemplo 1: Insertar un registro en todas las columnas**

INSERT INTO empleados (id, nombre, edad, salario, departamento\_id)

VALUES (1, 'Juan Pérez', 30, 3500.50, 2);

Explicación:

* Se insertan valores en todas las columnas en el orden definido en la tabla.

**Ejemplo 2: Insertar un registro en algunas columnas**

INSERT INTO empleados (id, nombre, salario)

VALUES (2, 'Ana García', 4000.00);

Explicación:

* Solo se insertan valores en las columnas id, nombre y salario.
* Las demás columnas recibirán NULL o el valor por defecto.

**Ejemplo 3: Insertar múltiples registros**

INSERT ALL

INTO empleados (id, nombre, edad, salario, departamento\_id) VALUES (3, 'Carlos Gómez', 45, 5000.00, 1)

INTO empleados (id, nombre, edad, salario, departamento\_id) VALUES (4, 'Laura Torres', 28, 3200.00, 2)

SELECT \* FROM DUAL;

Explicación:

* Se insertan dos registros en una sola ejecución.

## UPDATE - Modificar Datos

El comando UPDATE permite actualizar valores en una tabla.

**Ejemplo 1: Modificar un solo campo**

UPDATE empleados

SET salario = 4200.00

WHERE id = 2;

Explicación:

* Modifica el salario del empleado con id = 2.

**Ejemplo 2: Modificar múltiples campos**

UPDATE empleados

SET salario = 5500.00, edad = 46

WHERE id = 3;

Explicación:

* Modifica el salario y la edad del empleado con id = 3.

**Ejemplo 3: Modificar registros con una condición más amplia**

UPDATE empleados

SET salario = salario \* 1.10 -- Aumenta el salario en un 10%

WHERE departamento\_id = 2;

Explicación:

* Aumenta el salario en un 10% para todos los empleados del departamento 2.

## DELETE - Eliminar Datos

El comando DELETE permite eliminar registros de una tabla.

**Ejemplo 1: Eliminar un registro específico**

DELETE FROM empleados

WHERE id = 4;

Explicación:

* Elimina el empleado con id = 4.

**Ejemplo 2: Eliminar varios registros con una condición**

DELETE FROM empleados

WHERE salario < 3500.00;

Explicación:

* Elimina todos los empleados cuyo salario sea menor a 3500.00.

**Ejemplo 3: Eliminar todos los registros de la tabla**

DELETE FROM empleados;

Explicación:

* Elimina todos los registros, pero la estructura de la tabla permanece intacta.
* Si se necesita reiniciar el contador de identidad en versiones 12c+, se puede usar:

ALTER TABLE empleados MODIFY id GENERATED BY DEFAULT AS IDENTITY (START WITH 1);

## SELECT - Consultar Datos

El comando SELECT se usa para recuperar datos de una tabla.

**Ejemplo 1: Consultar todos los registros**

SELECT \* FROM empleados;

Explicación:

* Muestra todos los registros de la tabla empleados.

**Ejemplo 2: Seleccionar columnas específicas**

SELECT nombre, salario FROM empleados;

Explicación:

* Muestra solo las columnas nombre y salario.

**Ejemplo 3: Filtrar datos con WHERE**

SELECT \* FROM empleados

WHERE salario > 4000.00;

Explicación:

* Muestra los empleados con salario superior a 4000.00.

**Ejemplo 4: Ordenar resultados**

SELECT nombre, salario

FROM empleados

ORDER BY salario DESC;

Explicación:

* Ordena los empleados de mayor a menor salario.

**Ejemplo 5: Contar registros**

SELECT COUNT(\*) AS total\_empleados

FROM empleados

WHERE departamento\_id = 1;

Explicación:

* Cuenta cuántos empleados pertenecen al departamento 1.

## MERGE - Insertar o Actualizar Datos

El comando MERGE permite realizar una **actualización o inserción** en una misma operación.

**Ejemplo: Insertar o actualizar empleados**

MERGE INTO empleados e

USING (SELECT 5 AS id, 'Pedro López' AS nombre, 33 AS edad, 4100.00 AS salario, 1 AS departamento\_id FROM DUAL) src

ON (e.id = src.id)

WHEN MATCHED THEN

UPDATE SET e.nombre = src.nombre, e.edad = src.edad, e.salario = src.salario, e.departamento\_id = src.departamento\_id

WHEN NOT MATCHED THEN

INSERT (id, nombre, edad, salario, departamento\_id)

VALUES (src.id, src.nombre, src.edad, src.salario, src.departamento\_id);

Explicación:

* Si el id = 5 ya existe en empleados, se actualizan sus datos.
* Si id = 5 no existe, se inserta un nuevo registro.

**Resumen de los Comandos DML**

1. **INSERT** – Agrega registros nuevos a la tabla.
2. **UPDATE** – Modifica los datos de registros existentes.
3. **DELETE** – Elimina registros de una tabla.
4. **SELECT** – Recupera datos de la base de datos.
5. **MERGE** – Realiza inserción o actualización en una sola operación.

# Funciones en Oracle

En **Oracle**, las funciones se dividen en varias categorías:

1. **Funciones de cadena (String)**
2. **Funciones numéricas**
3. **Funciones de fecha y hora**
4. **Funciones de conversión**
5. **Funciones de agregación**
6. **Funciones condicionales**

## 1. Funciones de cadena (String)

Estas funciones permiten manipular cadenas de texto.

**1.1 UPPER() - Convertir a mayúsculas**

SELECT UPPER('Hola Mundo') AS resultado FROM DUAL;

-- Resultado: 'HOLA MUNDO'

**1.2 LOWER() - Convertir a minúsculas**

SELECT LOWER('Hola Mundo') AS resultado FROM DUAL;

-- Resultado: 'hola mundo'

**1.3 INITCAP() - Primera letra en mayúscula**

SELECT INITCAP('hola mundo') AS resultado FROM DUAL;

-- Resultado: 'Hola Mundo'

**1.4 LENGTH() - Longitud de una cadena**

SELECT LENGTH('Oracle') AS resultado FROM DUAL;

-- Resultado: 6

**1.5 SUBSTR() - Extraer una parte de la cadena**

SELECT SUBSTR('Oracle Database', 1, 6) AS resultado FROM DUAL;

-- Resultado: 'Oracle'

* Extrae desde la posición 1, los siguientes 6 caracteres.

**1.6 INSTR() - Posición de un carácter en la cadena**

SELECT INSTR(nombre, 'D') AS resultado FROM empleados;

-- Resultado: 8

* Retorna la posición donde aparece la letra 'D'.

**1.7 REPLACE() - Reemplazar un valor dentro de una cadena**

SELECT REPLACE('Hola Mundo', 'Mundo', 'Oracle') AS resultado FROM DUAL;

-- Resultado: 'Hola Oracle'

## 2. Funciones numéricas

Se usan para realizar cálculos matemáticos.

**2.1 ROUND() - Redondeo**

SELECT ROUND(3.567, 2) AS resultado FROM DUAL;

-- Resultado: 3.57

**2.2 TRUNC() - Truncar decimales**

SELECT TRUNC(3.567, 2) AS resultado FROM DUAL;

-- Resultado: 3.56

**2.3 MOD() - Obtener el módulo de una división**

SELECT MOD(10, 3) AS resultado FROM DUAL;

-- Resultado: 1

* 10 dividido por 3 da 3 y sobra 1.

**2.4 ABS() - Valor absoluto**

SELECT ABS(-15) AS resultado FROM DUAL;

-- Resultado: 15

**2.5 CEIL() y FLOOR() - Redondeo hacia arriba o abajo**

SELECT CEIL(4.3) AS resultado, FLOOR(4.3) AS resultado2 FROM DUAL;

-- Resultado: CEIL: 5, FLOOR: 4

## 3. Funciones de fecha y hora

Para manipular fechas en Oracle.

**3.1 SYSDATE - Obtener la fecha actual**

SELECT SYSDATE FROM DUAL;

-- Resultado: 18-FEB-2025 (dependiendo del día actual)

**3.2 CURRENT\_DATE - Fecha actual con zona horaria**

SELECT CURRENT\_DATE FROM DUAL;

-- Resultado: 18-FEB-2025 15:30:00 (dependiendo de la zona horaria)

**3.3 ADD\_MONTHS() - Sumar meses a una fecha**

SELECT ADD\_MONTHS(SYSDATE, 3) AS nueva\_fecha FROM DUAL;

-- Resultado: Fecha actual + 3 meses

**3.4 MONTHS\_BETWEEN() - Diferencia en meses entre dos fechas**

SELECT MONTHS\_BETWEEN(TO\_DATE('2025-12-01', 'YYYY-MM-DD'), SYSDATE) FROM DUAL;

-- Resultado: Cantidad de meses entre ambas fechas

**3.5 NEXT\_DAY() - Próximo día específico**

SELECT NEXT\_DAY(SYSDATE, 'LUNES') FROM DUAL;

-- Resultado: Fecha del siguiente lunes

**3.6 LAST\_DAY() - Último día del mes**

SELECT LAST\_DAY(SYSDATE) FROM DUAL;

-- Resultado: Último día del mes en curso

## 4. Funciones de conversión

Permiten convertir tipos de datos.

**4.1 TO\_CHAR() - Convertir número o fecha a texto**

SELECT TO\_CHAR(SYSDATE, 'DD-MON-YYYY') FROM DUAL;

-- Resultado: '18-FEB-2025'

**4.2 TO\_DATE() - Convertir texto a fecha**

SELECT TO\_DATE('18-02-2025', 'DD-MM-YYYY') FROM DUAL;

-- Resultado: 18-FEB-2025

**4.3 TO\_NUMBER() - Convertir texto a número**

SELECT TO\_NUMBER('12345') FROM DUAL;

-- Resultado: 12345 (como número)

## 5. Funciones de agregación

Se usan en consultas con GROUP BY.

**5.1 SUM() - Suma total**

SELECT SUM(salario) AS total\_salarios FROM empleados;

**5.2 AVG() - Promedio**

SELECT departamento\_id , AVG(salario) AS salario\_promedio FROM empleados GROUP BY departamento\_id;

**5.3 COUNT() - Contar registros**

SELECT COUNT(\*) AS total\_empleados FROM empleados;

**5.4 MAX() y MIN() - Valores máximo y mínimo**

SELECT MAX(salario) AS salario\_maximo, MIN(salario) AS salario\_minimo FROM empleados;

## 6. Funciones condicionales

**6.1 CASE - Evaluar condiciones**

SELECT nombre, salario,

CASE

WHEN salario > 5000 THEN 'Alto'

WHEN salario BETWEEN 3000 AND 5000 THEN 'Medio'

ELSE 'Bajo'

END AS nivel\_salarial

FROM empleados;

**6.2 NVL() - Sustituir valores NULL**

SELECT nombre, NVL(salario, 0) AS salario FROM empleados;

# Relaciones entre Tablas en Oracle SQL

En una base de datos **relacional**, las tablas pueden estar **vinculadas** mediante relaciones. Estas relaciones permiten conectar información de diferentes tablas para mantener la integridad de los datos y evitar redundancias.

**Tipos de Relaciones entre Tablas**

1. **Uno a Uno (1:1)**
2. **Uno a Muchos (1:M)**
3. **Muchos a Muchos (M:M)**

## Relación Uno a Uno (1:1)

En esta relación, **cada registro en la tabla A tiene exactamente un registro relacionado en la tabla B**.

**Ejemplo:**

Supongamos que tenemos una tabla empleados y otra tabla empleados\_detalles, donde cada empleado tiene un solo detalle adicional (como su dirección o número de seguro).

CREATE TABLE clientes (

id NUMBER(5) PRIMARY KEY,

nombre VARCHAR2(50) NOT NULL

);

CREATE TABLE clientes\_detalles (

id NUMBER(5) PRIMARY KEY,

direccion VARCHAR2(100),

telefono VARCHAR2(20),

cliente\_id NUMBER(5) UNIQUE,

CONSTRAINT fk\_cliente FOREIGN KEY (cliente\_id) REFERENCES clientes (id)

);

**Explicación:**

* empleados.id es **clave primaria** en la tabla empleados.
* empleados\_detalles.empleado\_id es una **clave foránea** que apunta a empleados.id, con una restricción UNIQUE, asegurando una relación **1:1** (un empleado solo puede tener un único detalle asociado).

**Insertar datos**

INSERT INTO clientes VALUES (1, 'Juan Pérez');

INSERT INTO clientes VALUES (2, 'Ana García');

INSERT INTO clientes \_detalles VALUES (1, 'Calle Mayor 123', '654321987', 1);

INSERT INTO clientes \_detalles VALUES (2, 'Avenida Central 45', '678905432', 2);

**Consulta con JOIN para ver qué profesores enseñan qué cursos**

SELECT c.id, c.nombre, d.direccion, d.telefono

FROM clientes c

JOIN clientes\_detalles d ON c.id = d.empleado\_id;

## Relación Uno a Muchos (1:M)

En esta relación, **un registro en la tabla A puede estar relacionado con múltiples registros en la tabla B**, pero **cada registro en la tabla B pertenece solo a un registro en la tabla A**.

**Ejemplo:**

Un **departamento** puede tener **muchos empleados**, pero cada **empleado pertenece a un solo departamento**.

CREATE TABLE departamentos (

id NUMBER(3) PRIMARY KEY,

nombre VARCHAR2(50) NOT NULL

);

CREATE TABLE empleados (

id NUMBER(5) PRIMARY KEY,

nombre VARCHAR2(50) NOT NULL,

departamento\_id NUMBER(3),

CONSTRAINT fk\_departamento FOREIGN KEY (departamento\_id) REFERENCES departamentos(id)

);

**Explicación:**

* departamentos.id es la **clave primaria**.
* empleados.departamento\_id es la **clave foránea** que referencia departamentos.id, estableciendo una relación **uno a muchos** (1:M).

**Insertar datos**

INSERT INTO departamentos VALUES (1, 'Recursos Humanos');

INSERT INTO departamentos VALUES (2, 'TI');

INSERT INTO empleados VALUES (1, 'Juan Pérez', 1);

INSERT INTO empleados VALUES (2, 'Ana García', 1);

INSERT INTO empleados VALUES (3, 'Carlos Gómez', 2);

**Consulta con JOIN para ver qué empleados pertenecen a qué departamento**

SELECT e.id, e.nombre, d.nombre AS departamento

FROM empleados e

JOIN departamentos d ON e.departamento\_id = d.id;

## Relación Muchos a Muchos (M:M)

En esta relación, **varios registros de la tabla A pueden estar relacionados con varios registros de la tabla B**.

Para implementarla correctamente, necesitamos **una tabla intermedia** que actúe como **puente**.

**Ejemplo:**

Un **empleado** puede trabajar en varios **proyectos**, y un **proyecto** puede tener varios **empleados** trabajando en él.

CREATE TABLE empleados (

id NUMBER(5) PRIMARY KEY,

nombre VARCHAR2(50) NOT NULL

);

CREATE TABLE proyectos (

id NUMBER(5) PRIMARY KEY,

nombre VARCHAR2(50) NOT NULL

);

CREATE TABLE empleados\_proyectos (

empleado\_id NUMBER(5),

proyecto\_id NUMBER(5),

horas\_asignadas NUMBER(3),

CONSTRAINT pk\_empleados\_proyectos PRIMARY KEY (empleado\_id, proyecto\_id),

CONSTRAINT fk\_empleado FOREIGN KEY (empleado\_id) REFERENCES empleados(id),

CONSTRAINT fk\_proyecto FOREIGN KEY (proyecto\_id) REFERENCES proyectos(id)

);

**Explicación:**

* empleados.id y proyectos.id son claves primarias en sus respectivas tablas.
* empleados\_proyectos es la **tabla intermedia** que contiene:
  + empleado\_id y proyecto\_id como **claves foráneas**.
  + Una **clave primaria compuesta** (empleado\_id, proyecto\_id) para evitar registros duplicados.

**Insertar datos**

INSERT INTO profesores VALUES (1, 'Juan Pérez');

INSERT INTO profesores VALUES (2, 'Ana García');

INSERT INTO cursos VALUES (101, 'Bases de Datos');

INSERT INTO cursos VALUES (102, 'Programación en Java');

INSERT INTO profesores\_cursos VALUES (1, 101); -- Juan Pérez enseña Bases de Datos

INSERT INTO profesores\_cursos VALUES (1, 102); -- Juan Pérez enseña Programación en Java

INSERT INTO profesores\_cursos VALUES (2, 101); -- Ana García enseña Bases de Datos

**Consulta con JOIN para ver qué profesores enseñan qué cursos**

SELECT p.nombre AS profesor, c.nombre AS curso

FROM profesores\_cursos pc

JOIN profesores p ON pc.profesor\_id = p.id

JOIN cursos c ON pc.curso\_id = c.id;