# Material Teórico Base de datos 1 (BD 1)

Recopilación de todas las unidades y su respectivo tema de la cátedra base de datos 1 (FaCena Unne - LSI)

Producido por Garay, Kevin Emiliano

# Índice de unidades

# 1. UNIDAD 1

## 1.1 Sistema gestor de bases de datos.

Un **sistema gestor de bases de datos**, o **SGBD**, es el software diseñado para colaborar en el mantenimiento y empleo de grandes conjuntos de datos.

Pueden gestionar los datos de un **modo robusto y eficiente**

### 1.1.1 Desventajas

- La aplicación debe organizar grandes conjuntos de datos entre la memoria principal y el almacenamiento secundario
- Hay que escribir programas especiales para responder a diferentes consultas
- Hay que proteger los datos de inconsistencias realizadas por usuarios diferentes que acceden a los datos de manera concurrente
- Recuperación de fallos. Hay que garantizar que los datos vuelvan a un estado consistente si el sistema falla
- Seguridad y control de acceso. Los sistemas operativos sólo ofrecen para la seguridad un mecanismo de contraseñas

1.1.2 Ventajas

- Independencia con respecto a los datos
- Acceso eficiente a los datos
- Integridad y seguridad de los datos
- Administración de los datos
- Acceso concurrente y recuperación en caso de fallo
- Reducción del tiempo de desarrollo de las aplicaciones

## 1.2 Componentes SGBD.

### 1.2.1 Datos

Los datos de las bases de datos serán tanto integrados como compartidos.

Integrados: Podemos imaginar a la BD como la unificación de varios archivos.

- Por ejemplo, los datos de un “alumno” están organizados internamente en varias estructuras relacionadas.

Compartida: Los elementos de la BD pueden ser compartidas entre diferentes usuarios, y cada uno de ellos puede tener acceso al mismo conjunto de datos.

### 1.2.2 Hardware

Los componentes de hardware del sistema constan de:

- Los volúmenes de almacenamiento secundario que se emplean para contener los datos almacenados, junto con los dispositivos asociados de E/S (unidades de discos, etc.), los controladores de dispositivos, los canales de E/S, etc…
- Los procesadores de hardware y la memoria principal asociada usados para apoyar la ejecución del software del sistema de base de datos

### 1.2.3 Software

Entre la base de datos física y los usuarios del sistema, hay una capa de software conocida de manera indistinta como el administrador de base de datos o el servidor de base de datos; o más comúnmente como el sistema de administración de base de datos (DBMS).

### 1.2.4 Usuarios

Se consideran tres grandes clases de usuarios:

- Las aplicaciones (.NET, JAVA, PHP) que acceden a la base de datos emitiendo la solicitud apropiada al DBMS.
- Los usuarios finales, quienes interactúan con el sistema, accediendo desde la aplicación o desde una interfaz proporcionada por el SGBD.
- El administrador de base de datos o DBA

## 1.3 Visión de los datos.

**Niveles de abstracción**:

- **Nivel físico**: Cómo se almacenan realmente los datos.
- **Nivel lógico**: Cómo están estructurados los datos y las relaciones entre ellos.
- **Nivel de vista**: Lo que ven los usuarios (interfaces simplificadas).

## 1.4 Bases de Datos.

Una Base de Datos es un **conjunto de datos persistentes (o no)** utilizados por los sistemas de aplicación de alguna empresa o institución.

Una **colección de datos** que cumplen las siguientes propiedades:

- Están estructurados independientemente de las aplicaciones y del soporte de almacenamiento que los contiene.-
- Presentan la menor redundancia posible.-
- Son compartidos por varios usuarios y/o aplicaciones.-

Desde el punto de vista informático, una base de datos es un **sistema formado por un conjunto de datos almacenado**s en 'discos' que permiten el acceso directo a ellos y un conjunto de programas que manipulan ese conjunto de datos.-

Una base de datos es una **descripción de una colección particular de datos**, utilizando un modelo de datos determinado.

## 1.5 Independencia de los datos.

- **Independencia lógica**: Los cambios en el diseño lógico no afectan al nivel físico.
- **Independencia física**: Los cambios en el almacenamiento físico no afectan al diseño lógico.

## 1.6 Los sistemas relacionales y otros sistemas.

- **Relacionales**: Basados en tablas (relaciones) con claves primarias y foráneas.
- **Otros sistemas**:
    - Jerárquicos: Datos organizados como un árbol.
    - Redes: Usan gráficos para representar relaciones complejas.
    - NoSQL: Diseñados para datos no estructurados (JSON, documentos).

## 1.7 Tendencias de las bases de datos relacionales.

Una razón por la que los sistemas de bases de datos relacionales se han vuelto tan dominantes, tanto en el mundo industrial como en el académico, es que **manejan en forma muy directa la interpretación precedente de los datos y las bases de datos.**

Los sistemas relacionales están basados en una teoría formal denominada el **modelo de datos relacional.**

- **Escalabilidad**: Bases de datos distribuidas y en la nube.
- **Compatibilidad**: Integración con Big Data y tecnologías modernas como inteligencia artificial.
- **Mejoras en rendimiento**: Uso de índices avanzados, almacenamiento en memoria, y optimización de consultas.

# 2. UNIDAD 2

## 2.1 Diseño de bases de datos y el modelo Entidad-Relación

### 2.1.1 Modelo de datos

Un **modelo de datos** es una definición **lógica, independiente y abstracta** de los objetos, operadores y demás elementos que en conjunto constituyen la máquina abstracta con la que interactúan los usuarios.

- Los objetos nos permiten modelar la estructura de los datos.
- Los operadores nos permiten modelar su comportamiento.

Un **modelo de datos** es un conjunto de estructuras descriptivas de datos de **alto nivel** que oculta muchos detalles de almacenamiento de bajo nivel.

### 2.1.2 Modelo E-R

El **modelo de datos entidad-relación (ER)** permite describir los datos implicados en empresas/organizaciones reales en términos de **objetos y de sus relaciones**, y se emplea para desarrollar el diseño preliminar de las bases de datos.

El modelo de datos de entidad-relación (ER) se basa en una p**ercepción de un mundo real** que consiste en un conjunto de objetos básicos llamados **entidades** y de **relaciones** entre estos objetos.

## 2.2 Visión general del proceso de diseño (Etapas del Diseño).

Se compone de las siguientes etapas:

1. **Análisis de requisitos**
   El primer paso del diseño de aplicaciones de bases de datos es **comprender los datos** que se deben guardar en la base de datos, las **aplicaciones** que se deben construir sobre ellos y las **operaciones** que son más frecuentes e imponen requisitos de rendimiento.
2. **Diseño conceptual de la BD**
   La información reunida en el análisis de requisitos se emplea para desarrollar una descripción de alto nivel de los datos que se van a guardar en la base de datos, junto con las restricciones que se sabe que se impondrán sobre esos datos. El **modelo ER** es uno de los modelos de datos de alto nivel, o semánticos, empleados en el diseño de bases de datos.
3. **Diseño lógico de la base de datos**
   Hay que escoger un SGBD que implemente nuestro diseño de la base de datos y transformar el diseño conceptual de la base de datos en un esquema de base de datos del modelo de datos del SGBD elegido. Sólo se considerarán SGBD relacionales.
4. **Refinamiento de los esquemas**
   Es el análisis del conjunto de relaciones del esquema relacional de la base de datos para identificar posibles problemas y refinarlo. El refinamiento del esquema se puede guiar por la teoría de **normalización** de relaciones.
5. **Diseño físico de la BD**
   Se toman en consideración las cargas de trabajo típicas esperadas que deberá soportar la base de datos y se refinó aún más el diseño de la base de datos para garantizar que cumple los criterios de rendimiento deseados.

6. **Diseño de aplicaciones y de la seguridad**
   Se toman en consideración los aspectos de la aplicación. Determinar claramente las entidades y procesos relacionados. Los roles de los usuarios, y el acceso o restricciones a los datos.

## 2.3 El modelo entidad-relación. Restricciones.

### 2.3.1 Entidad

Una **entidad** es un objeto del mundo real que puede distinguirse de otros objetos. Se identifica con los demás de su especie a través de sus atributos.

### 2.3.2 Atributos

Cada entidad se describe empleando un conjunto de **atributos.** La selección de los atributos refleja el nivel de detalle con el que se desea representar la información relativa a las entidades. Para cada atributo asociado con un conjunto de entidades hay que identificar el **dominio** de valores posibles.

### 2.3.3 Claves

Una **clave** es un conjunto mínimo de atributos cuyos valores identifican de manera unívoca a cada entidad del conjunto. Puede haber más de una **clave candidata**; en ese caso, se escogerá una de ellas como **clave principal.**

### 2.3.4 Relación

Una **relación** es una asociación entre dos o más entidades. Es el grado de asociatividad entre entidades.

Las relaciones también pueden tener **atributos descriptivos**. Los atributos descriptivos se emplean para registrar la información sobre la relación, más que sobre las entidades participantes.

**Dominio**. Conjunto de datos que cumplen con restricciones determinadas, mismos que serán almacenados en los objetos en caso de cumplirlas.

## 2.3.5 Restricciones

En el modelo entidad-relación están presentes las siguientes restricciones:

**Tipos de restricciones**:

- **Clave primaria**: Identifica de forma única cada entidad o fila.
- **Cardinalidad**: Define cuántas entidades de un tipo pueden asociarse con entidades de otro tipo (1:1, 1, N).
- **Restricciones de integridad**: Reglas como valores no nulos o únicos.

### 2.3.6 Cardinalidad

- **Relaciones de uno a uno**

  Se denota por (1:1). Es cuando un dato de la entidad A se relaciona únicamente con un dato de la unidad B. Una ocurrencia de la entidad A está asociada con a lo sumo una ocurrencia de la entidad B y una ocurrencia de la entidad B está asociada con a lo sumo una ocurrencia de la entidad A. Ej.: el DNI con el Legajo de un empleado Bancario.-

- **Relaciones de uno a muchos**

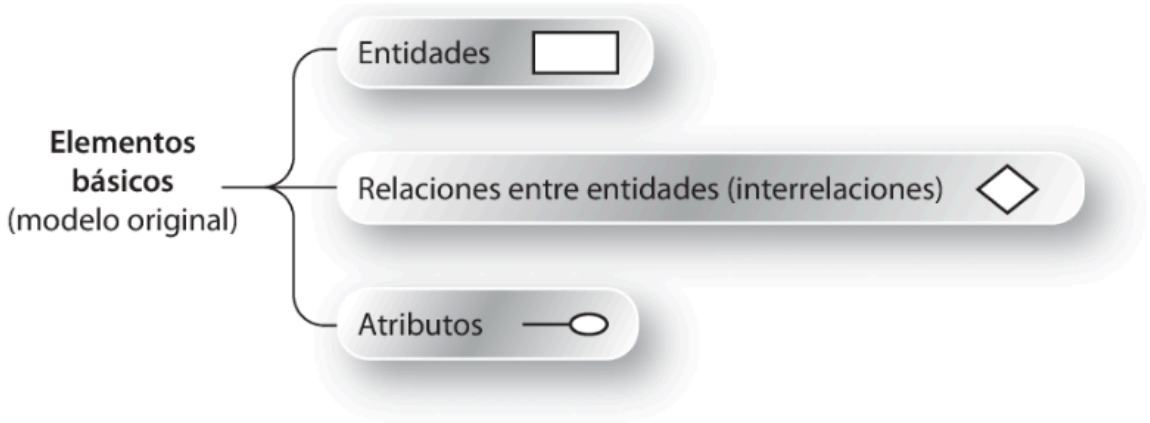
  Se denota por (1:M). Es cuando un dato de la entidad A se relaciona con varios datos de una unidad B, pero los registros de B solamente se relacionan con un registro de A. Una ocurrencia de la entidad A está asociada con cualquier número de ocurrencias de la entidad B, pero toda ocurrencia de una entidad B solo puede estar asociada con una ocurrencia de la entidad A. Ej: Un País tiene muchas provincias, pero la provincia pertenece a un solo País.-

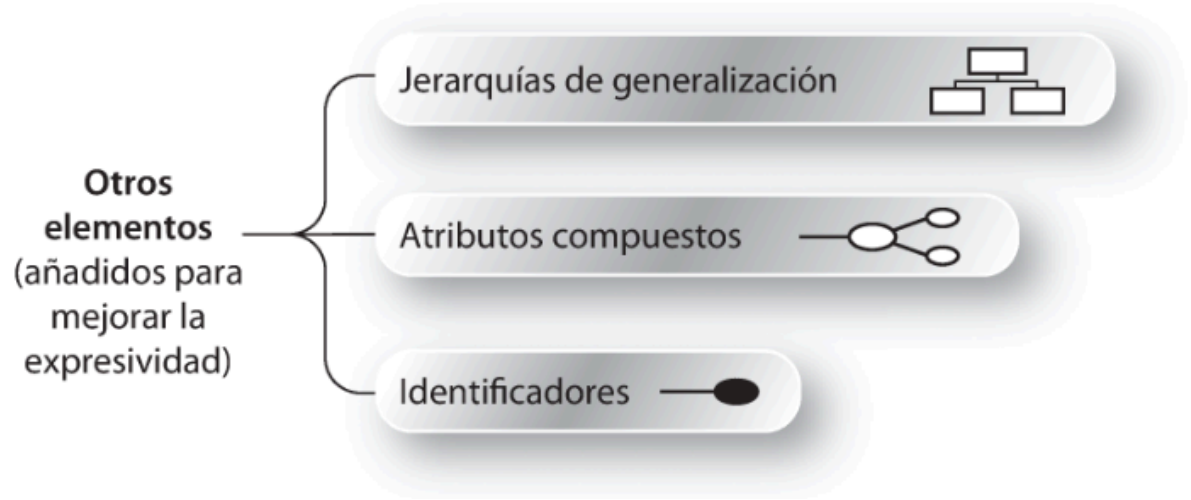
- **Relaciones de muchos a muchos**

  Se denota por (N:M) es cuando un dato de la entidad en A se puede relacionar con 0 o varios datos de la entidad B y viceversa. Cualquier ocurrencia de la entidad A, está asociada con varias ocurrencias de la entidad B, y cualquier ocurrencia de la entidad B está asociada con varias ocurrencias de la entidad A. Ej.: Un alumno cursa varias materias y una materia tiene varios alumnos cursantes.

## 2.4 Diagramas entidad-relación.

En los diagramas de entidad-relación se tienen presentes los siguientes elementos:




### 2.4.1 Aspectos del diseño entidad-relación.

**Atributos compuestos**: Atributos que pueden dividirse en subpartes (e.g., Nombre completo → Nombre, Apellido).

**Atributos multivaluados**: Atributos que pueden tener múltiples valores (e.g., Teléfonos).

### 2.4.2 Conjuntos de entidades débiles.

**Entidades débiles**:

- Entidades que dependen de una entidad "fuerte" para existir.
- Tienen una clave parcial y una relación identificadora.

### 2.4.3 Características del modelo E-R extendido.

**Atributos derivados**: Atributos calculados a partir de otros (e.g., Edad a partir de Fecha de Nacimiento).

**Herencia y especialización**:

- Especialización: Subtipos de una entidad general (e.g., Empleado → Gerente, Técnico).
- Generalización: Agrupación de entidades específicas en una general.

## 2.5 Reducción a esquemas relacionales.

Para la reducción a un esquema relacional hay que seguir esta secuencia de pasos

A partir de un diagrama de diseño relacional debemos determinar los siguientes datos:

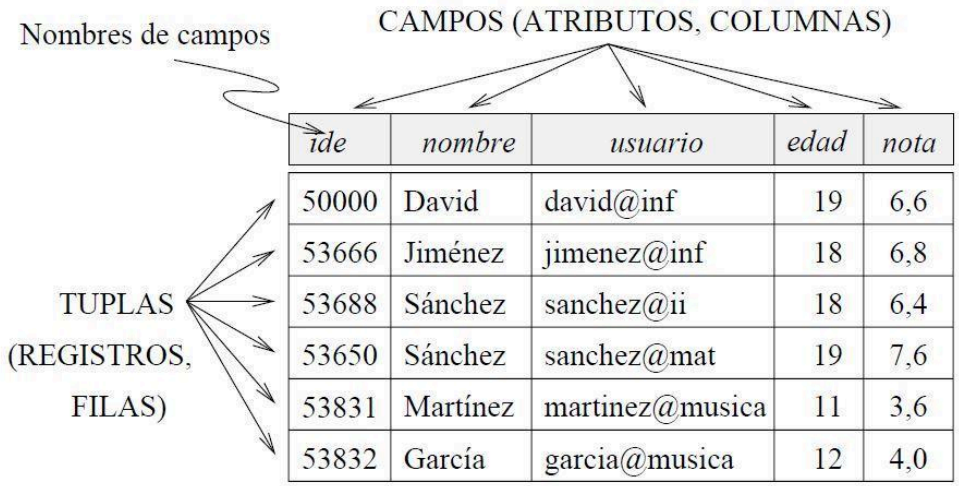
1. Identificamos las entidades del problema
2. Identificamos las relaciones entre las entidades
3. Identificamos atributos posibles
4. Determinamos la cardinalidad de la relación

Luego de obtener estos datos podemos

1. Identificar las PK (Claves Primarias)
2. Establecer las FK (Claves Foráneas)
3. Resolver las relaciones N:M

# 3. UNIDAD 3

## 3.1 Estructura del Modelo Relacional




| *ide* | *nombre* | *usuario* | *edad* | *nota* |
|---|---|---|---|---|
| 50000 | David | david@inf | 19 | 6,6 |
| 53666 | Jiménez | jimenez@inf | 18 | 6,8 |
| 53688 | Sánchez | sanchez@ii | 18 | 6,4 |
| 53650 | Sánchez | sanchez@mat | 19 | 7,6 |
| 53831 | Martínez | martinez@musica | 11 | 3,6 |
| 53832 | García | garcia@musica | 12 | 4,0 |

### 3.1.1 Entidad

La entidad es una **tabla. El** esquema de la entidad describe las cabeceras de las columnas de esa tabla.

### 3.1.2 Esquema

El esquema especifica el nombre de la relación, el de cada **campo** , y el dominio de cada campo.

En la entidad  se hace referencia al **dominio** por su nombre de dominio y tiene un **conjunto de valores** asociados.

### 3.1.3 Grado

El **grado**, también denominado aridad (número de argumentos que una función requiere), de una relación es su número de campos.

### 3.1.4 Cardinalidad

La **cardinalidad** de un ejemplar de la relación es el número de tuplas(registros o filas) que contiene.

## 3.2 Restricciones de integridad

**Una base de datos sólo es tan buena como la información almacenada en ella** y, por tanto, el SGBD debe ayudar a evitar la introducción de información incorrecta.

Una **restricción de integridad** (IC) es una condición especificada en el esquema de la base de datos que restringe los datos que pueden almacenarse en los ejemplares de la base de datos.

Las restricciones de **dominio**, de **clave principal** y de **clave externa** se consideran una parte fundamental del modelo relacional de datos y se les presta especial atención en la mayor parte de los sistemas comerciales.

### 3.2.1 Restricciones de clave

Una restricción de clave es una declaración de que un cierto subconjunto mínimo de los campos de una relación constituye un identificador  único de cada tupla.

### 3.2.2 Restricciones de clave externa

La RI que implica a dos relaciones más frecuentes es la restricción de clave externa. Si se modifica una de las relaciones hay que comprobar la otra y, quizás, modificarla para hacer que los datos sigan siendo consistentes.

### 3.2.3 Restricciones generales.

Se pueden aplicar otras **restricciones generales** como ser de  unicidad, y control.

## 3.3 Operaciones fundamentales del álgebra relacional.

Estas operaciones permiten consultar y manipular datos:

1. **Selección (SELECT / σ)**:
    - Filtra filas basándose en una condición.
    - Ejemplo: Obtener productos con precio mayor a 50.
2. **Proyección (SELECT * FROM tabla π)**:
    - Selecciona columnas específicas de una tabla.
    - Ejemplo: Obtener nombres y precios de productos.
3. **Unión (∪)**:
    - Combina filas de dos tablas con el mismo esquema.
4. **Intersección (∩)**:
    - Devuelve filas comunes entre dos tablas.
5. **Diferencia (−)**:
    - Devuelve filas que están en una tabla pero no en otra.
6. **Producto cartesiano (×)**:
    - Combina todas las filas de dos tablas.
    - Base para otras operaciones como uniones y combinaciones.
7. **Join**:
    - Combina filas relacionadas de dos tablas.

## 3.4 Valores nulos.

**Definición**: Representan datos desconocidos o no aplicables.
**Implicaciones**:

- Dificultan algunas operaciones como comparaciones y agregaciones.
- Necesitan manejo cuidadoso en consultas y diseño de base de datos.

## 3.5 Diseño lógico de bases de datos: del modelo E-R a relacional.

Para transformar de un diseño lógico del modelo E-R a relacional

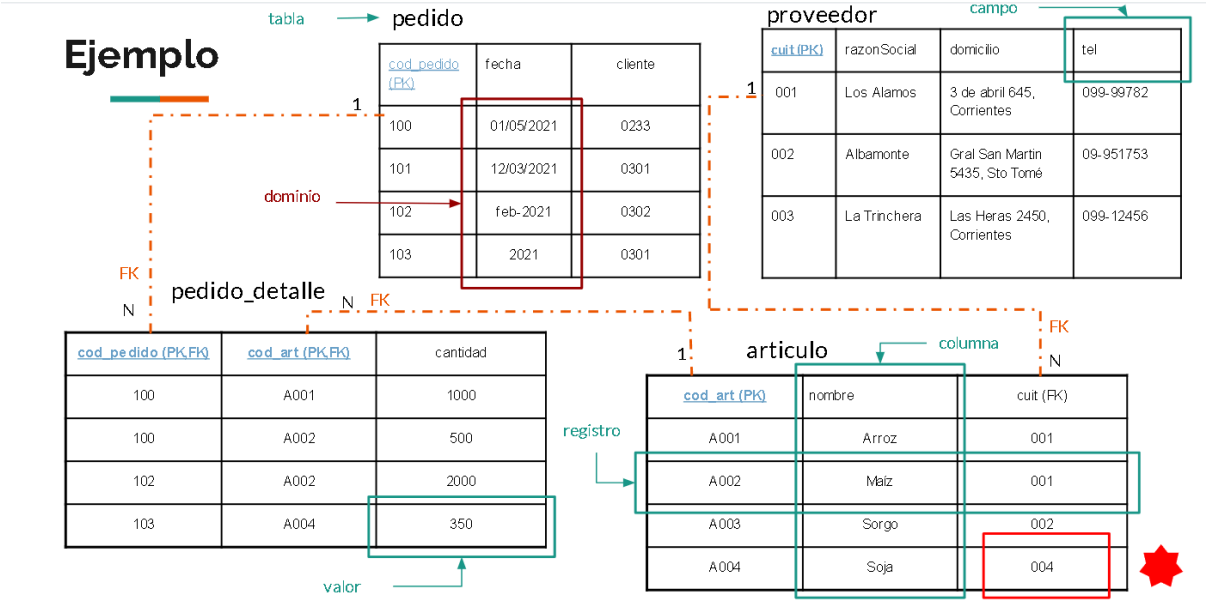
podemos realizar los siguientes pasos, siempre teniendo en cuenta los objetivos

Objetivo del diseño relacional:

- atomicidad
- coherencia/integridad
- redundancia

Pasos a seguir para la transformación:

1.- Identificar las PK
2.- Establecer las FK
3.- Resolver las relaciones N:M




Breve ejemplo de cómo sería la transformación

# 4. UNIDAD 4

## 4.1 Refinamiento y formas normales.

Refinamiento es el proceso de garantizar que los datos que se introducen en la plataforma sean relevantes, estén categorizados y homogeneizados. Esto permite que los usuarios puedan obtener resultados significativos y detectar discrepancias.

En esta etapa hay que tener presente las restricciones de negocio

- Son todos aquellos requerimientos funcionales y no funcionales que se pueden trasladar al modelo de datos, como ser restricción de dominio, de clave foránea, claves únicas, etc. y que no fueron contemplados en la

etapa del diseño lógico.

## 4.2 Características de los buenos diseños relacionales.

Un diseño relacional eficiente debe cumplir con:

1. **Eliminación de redundancias**:
    - Evitar duplicación innecesaria de datos.
    - Reduce inconsistencias y ahorra espacio de almacenamiento.
2. **Integridad de los datos**:
    - Uso de restricciones (claves primarias, claves externas, restricciones de dominio).
3. **Flexibilidad y escalabilidad**:
    - Facilidad para adaptarse a cambios futuros.
4. **Acceso eficiente**:
    - Diseño optimizado para consultas comunes.

## 4.3 Dependencias funcionales.

**Definición**: Relación en la que el valor de un atributo (o conjunto de atributos) determina el valor de otro atributo.

- Ejemplo: En una tabla de empleados, el **ID del empleado** determina el conjunto de todos sus datos..

**Dependencia completa**:

- Cuando un atributo depende de todas las claves de una tabla.
- Ejemplo:

**Dependencia parcial**:

- Cuando un atributo depende solo de parte de una clave compuesta.
- Ejemplo:

## 4.4 Formas normales basadas en claves principales (1FN, 2FN y 3FN).

A la hora de establecer la clave primaria de una tabla debemos escoger un atributo, o conjunto de ellos, de los que dependan funcionalmente el resto de los atributos.

### 4.4.1 Primera Forma Normal

Una relación está en 1era FN si y sólo si cada atributo es atómico.

Supongamos que tenemos este diseño relacional, observemos que la

columna teléfonos no es atómica (Múltiple cantidad de valores para el mismo registro).

| Alumnos | | | | | | | |
|---|---|---|---|---|---|---|---|
| ID | Nombre | Curso | Fecha Matrícula | Tutor | Localidad | Provincia Alumno | Teléfonos |
| 11111111A | Eva | 1ESO-A | 01-Julio-2016 | Isabel | Ecatepec | México | 660111222 |
| 22222222B | Ana | 1ESO-A | 09-Julio-2016 | Isabel | Ecatepec | México | 660222333<br>660333444<br>660444555 |
| 33333333C | Susana | 1ESO-B | 11-Julio-2016 | Roberto | Ecatepec | México | |
| 44444444D | Juan | 2ESO-A | 05-Julio-2016 | Federico | Aragón | CDMX | |
| 55555555E | José | 2ESO-A | 02-Julio-2016 | Federico | Aragón | CDMX | 661000111<br>661000222 |

Una posible solución sería, separar la columna en una nueva tabla asociada con la clave primaria de la tabla

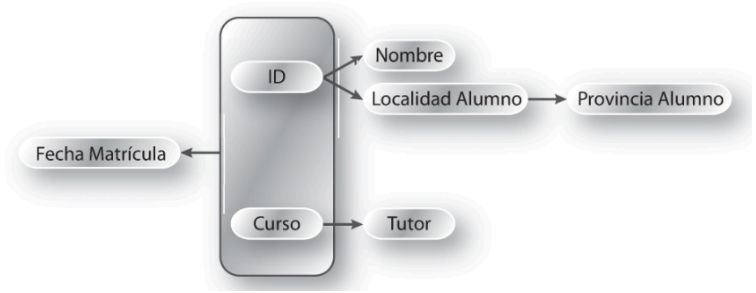
| Teléfonos | |
|---|---|
| DNI | Teléfono |
| 11111111A | 660111222 |
| 22222222B | 660222333 |
| 22222222B | 660333444 |
| 22222222B | 660444555 |
| 55555555E | 661000111 |
| 55555555E | 661000222 |

### 4.4.2 Segunda Forma Normal

Una relación está en 2FN si y sólo si está en 1era FN y todos los atributos que no forman parte de la clave principal tienen dependencia funcional completa de ella. Es decir, no existen referencias parciales.

Observando el ejemplo previamente planteado

| Alumnos | | | | | | |
|---|---|---|---|---|---|---|
| ID | Nombre | Curso | Fecha Matrícula | Tutor | Localidad | Provincia Alumno |
| 11111111A | Eva | 1ESO-A | 01-Julio-2016 | Isabel | Ecatepec | México |
| 22222222B | Ana | 1ESO-A | 09-Julio-2016 | Isabel | Ecatepec | México |
| 33333333C | Susana | 1ESO-B | 11-Julio-2016 | Roberto | Ecatepec | México |
| 44444444D | Juan | 2ESO-A | 05-Julio-2016 | Federico | Aragón | CDMX |
| 55555555E | José | 2ESO-A | 02-Julio-2016 | Federico | Aragón | CDMX |




La posible solucion podria ser

| Alumnos | | | |
|---|---|---|---|
| **ID** | **Nombre** | **Localidad** | **Provincia** |
| 11111111A | Eva | Ecatepec | México |
| 22222222B | Ana | Ecatepec | México |
| 33333333C | Susana | Ecatepec | México |
| 44444444D | Juan | Aragón | CDMX |
| 55555555E | José | Aragón | CDMX |

| Cursos | |
|---|---|
| **Curso** | **Tutor** |
| 1ESO-A | Isabel |
| 1ESO-B | Roberto |
| 2ESO-A | Federico |

| Matrículas | | |
|---|---|---|
| **ID** | **Curso** | **FechaMatrícula** |
| 11111111A | 1ESO-A | 01-Julio-2016 |
| 22222222B | 1ESO-A | 09-Julio-2016 |
| 33333333C | 1ESO-B | 11-Julio-2016 |
| 44444444D | 2ESO-A | 05-Julio-2016 |
| 55555555E | 2ESO-A | 02-Julio-2016 |

### 4.4.3 Tercera Forma Normal

Una relación está en 3era FN si y sólo si está en 2da FN y no existen dependencias transitivas. Todas las dependencias funcionales deben darse respecto a la clave principal.

Planteando nuevamente el ejemplo anterior

| Alumnos | | | |
|---|---|---|---|
| **ID** | **Nombre** | **Localidad** | **Provincia** |
| 11111111A | Eva | Ecatepec | México |
| 22222222B | Ana | Ecatepec | México |
| 33333333C | Susana | Ecatepec | México |
| 44444444D | Juan | Aragón | CDMX |
| 55555555E | José | Aragón | CDMX |


Podemos observar, existe una dependencia funcional transitiva: ID-> Localidad -> Provincia.

Una posible solución sería

| Alumnos | | |
|---|---|---|
| **ID** | **Nombre** | **Localidad** |
| 11111111A | Eva | Ecatepec |
| 22222222B | Ana | Ecatepec |
| 33333333C | Susana | Ecatepec |
| 44444444D | Juan | Aragón |
| 55555555E | José | Aragón |

| Localidades | |
|---|---|
| **Localidad** | **Provincia** |
| Ecatepec | México |
| Aragón | CDMX |

## 4.5 Forma normal de Boyce-Codd.

Una relación está en la cuarta forma normal(4NF), si y solo si esta en la tercera(3NF) y en todo momento cada tupla de la relación está formada por una clave primaria que identifica a una Instancia de una Entidad, más un conjunto de atributos que tienen dependencia funcional total, que son los que modelizan a una Entidad.-

Esto significa que:

- Si un atributo (o conjunto de atributos) determina otros en la tabla (es decir, es un determinante), ese atributo debe ser una **clave candidata**.
- Una **clave candidata** es un conjunto mínimo de atributos que puede identificar de manera única cada fila en una tabla.

En términos prácticos, BCNF corrige ciertas situaciones en las que la **Tercera Forma Normal (3FN)** puede fallar en relaciones más complejas. Estas situaciones ocurren cuando:

1. Una relación tiene más de una clave candidata.
2. Una de las claves candidatas no incluye todos los atributos determinantes.

## 4.6 Normalización.

Proceso de estructuración de una base de datos siguiendo las formas normales:

1. **Objetivo**:
    - Minimizar redundancias.
    - Evitar anomalías en la inserción, eliminación y actualización.

2. **Pasos**:
    - Identificar dependencias funcionales.
    - Aplicar las formas normales en secuencia (1FN → 2FN → 3FN).
    - Descomponer tablas problemáticas manteniendo integridad referencial.

### 4.6.1 Técnica de descomposición sin pérdida

- **Definición**: Descomponer una tabla en dos o más tablas sin perder información.
- **Condición**:
    - La unión de las tablas descompuestas debe equivaler a la tabla original.

### 4.6.2 Desnormalización

- **Definición**: Proceso inverso a la normalización que introduce redundancias controladas para mejorar el rendimiento de consultas.
- **Motivo**:
    - Reducir la cantidad de uniones necesarias en consultas frecuentes.
- **Cuidado**:
    - Puede aumentar el riesgo de inconsistencias.

# 5. UNIDAD 5

## 5.1 SQL y SQL avanzado. Introducción.

El lenguaje de consulta estructurado (SQL) es un lenguaje de base de datos normalizado, utilizado por un motor de base de datos, la mayoría de los DBMS más modernos hoy ya lo soportan. Este SQL se utiliza para concretar una conexión entre una base de datos y el medio externo a la misma. También se puede utilizar con algún método para crear y manipular directamente las bases de datos remotas del tipo Cliente-Servidor.

Componentes del SQL: El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.Existen tres tipos de comandos SQL.

- DDL (Data Definition Language)
- DML (Data Manipulation Language)
- DCL (Data Control Language)

## 5.2 Definición de datos.

Los “DDL” (Data Definition Language), lenguaje de definición de datos, que permiten crear y definir nuevas bases de datos, campos e índices.

| Comando | **Descripción del Comando** |
|---|---|
| **CREATE** | Utilizado para crear nuevas tablas, campos e índices |
| **DROP** | Empleado para eliminar tablas e índices |
| **ALTER** | Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos. |

Los tipos de datos SQL se clasifican en 13 tipos de datos primarios y de varios sinónimos válidos reconocidos por dichos tipos de datos.

- **BINARY**, 1 byte, Para consultas sobre tabla adjunta de productos de bases de datos que definen un tipo de datos Binario.
- **BIT**, 1 byte Valores Si/No ó True/False
- **BYTE**, 1 byte Un valor entero entre 0 y 255.
- **COUNTER**, 4 bytes Un número incrementado automáticamente (de tipo Long)
- **CURRENCY**, 8 bytes Un entero escalable entre 922.337.203.685.477,5808 y 922.337.203.685.477,5807.
- **DATETIME**, 8 bytes Un valor de fecha u hora entre los años 100 y 9999.
- **SINGLE**, 4 bytes Un valor en punto flotante de precisión simple con un rango de -3.402823*1038 a -1.401298*10-45 para valores negativos, 1.401298*10-45 a 3.402823*1038 para valores positivos, y 0.
- **DOUBLE**, 8 bytes Un valor en punto flotante de doble precisión con un rango de -1.79769313486232*10308 a -4.94065645841247*10-324 para valores negativos, 4.94065645841247*10-324 a 1.79769313486232*10308 para valores positivos, y 0.
- **SHORT**, 2 bytes Un entero corto entre -32,768 y 32,767
- **LONG**, 4 bytes Un entero largo entre -2,147,483,648 y 2,147,483,647.
- **LONGTEXT**, 1 byte por carácter de cero a un máximo de 1.2 gigabytes.
- **LONG BINARY**, Según se necesite de cero 1 gigabyte. Utilizado para objetos OLE.
- **TEXT**, 1 byte por carácter de cero a 255 caracteres.

Tipos definidos de usuario suelen ayudar para unificar el diseño de las tablas; por ejemplo, se puede crear un tipo llamado NIF que corresponde al tipo de datos CHAR(20), y admite valores nulos.

## 5.3 Estructura básica de las consultas SQL.

- Las sentencias SQL no son sensibles a mayúsculas/minúsculas.
- Las sentencias SQL pueden ocupar una o más líneas.
- Las palabras claves no se pueden abreviar ni dividir entre líneas.
- Las cláusulas suelen estar colocadas en líneas separadas.
- Los sangrados se utilizan para mejorar la legibilidad.

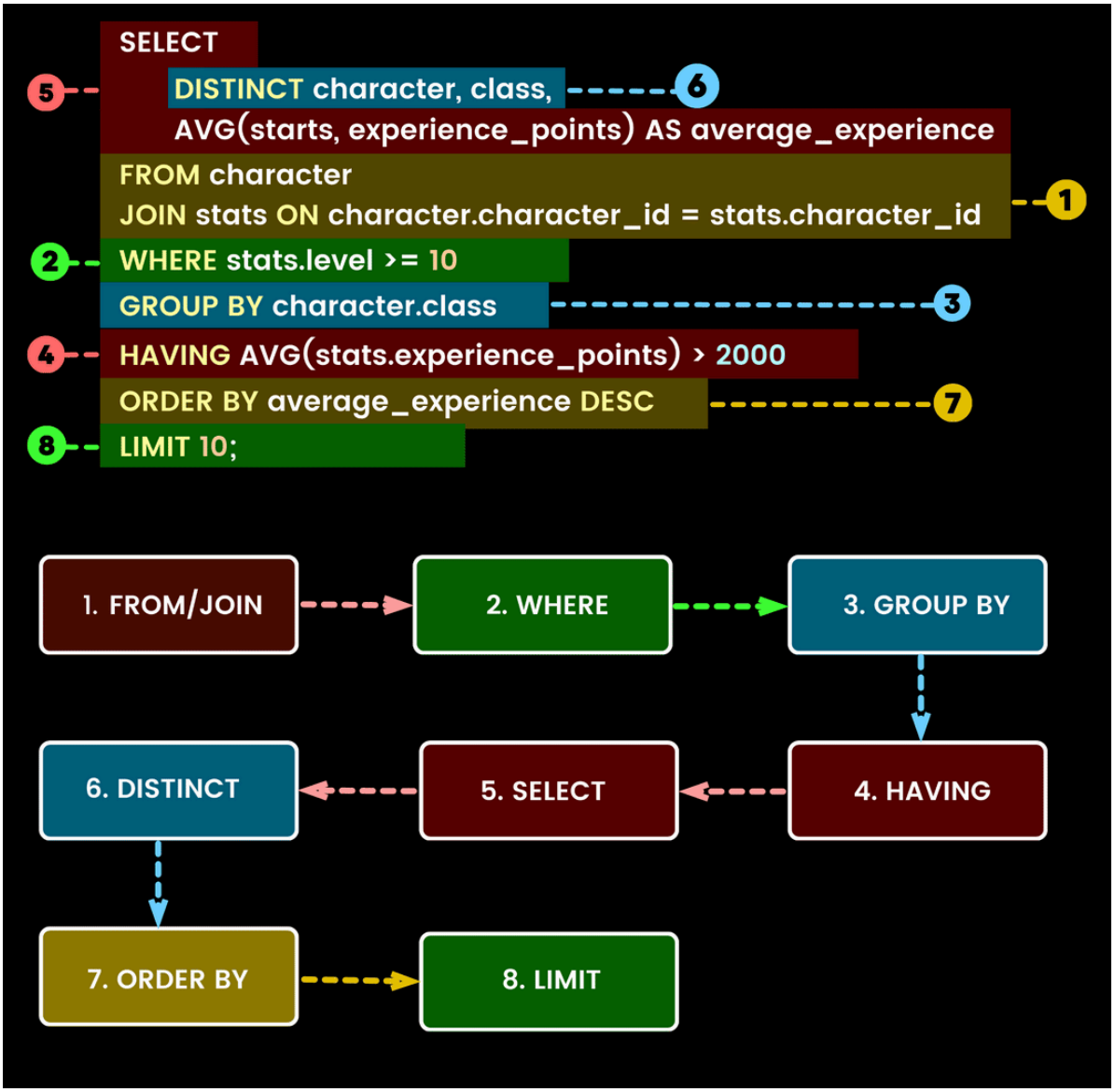
### 5.3.1 Cláusulas

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

| Cláusula | **Descripción de la Cláusula** |
|---|---|
| **FROM** | Utilizada para especificar la tabla de la cual se van a seleccionar los Registros.- |
| **WHERE** | Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar.- |
| **GROUP BY** | Utilizada para separar los registros seleccionados en grupos |

### 5.3.2 Orden de ejecución

Sql Server sigue este orden de ejecución en las consultas:



# 5.4 Operaciones sobre conjuntos.

## 5.4.1 Consultas de selección

Las consultas de selección se utilizan para indicar al motor de datos que devuelva información de las bases de datos, esta información es devuelta en forma de conjunto de registros que se pueden almacenar en una nueva tabla o base de datos, pudiendo ser esta modificables.

```
SELECT Campos/Columnas FROM Tabla;
```

Las condiciones de selección son las condiciones que pueden aparecer en la cláusula WHERE. En SQL tenemos cinco condiciones básicas:

1. Test de comparación.
2. Rango..
3. Pertenencia a un conjunto.
4. Valor nulo.
5. Correspondencia con patrón.

## 5.4.2 Consultas de Predicados

| Predicado | Descripción |
|---|---|
| **ALL** | Devuelve todos los campos de la tabla.- |
| **TOP** | Devuelve un determinado número de registros de la Tabla.- |
| **DISTINCT** | Omite los registros cuyos campos seleccionados coincidan totalmente.- |
| **DISTINCTROW** | Omite los registros duplicados basándose en la totalidad del registro y no sólo en los campos seleccionados.- |

## 5.4.3 Operadores de Conjuntos

- **UNION**: Combina resultados, eliminando duplicados.
- **INTERSECT**: Devuelve resultados comunes.
- **EXCEPT**: Devuelve resultados presentes en una consulta pero no en la otra.

# 5.5 Funciones de agregación.

Las funciones de agregado se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

| Función | Descripción |
|---|---|
| **AVG** | Utilizada para calcular el promedio de los valores de un campo Determinado.- |
| **COUNT** | Utilizada para devolver el número de registros de la selección.- |
| **SUM** | Utilizada para devolver la suma de todos los valores de un campo Determinado.- |
| **MAX** | Utilizada para devolver el valor más alto de un campo especificado.- |
| **MIN** | Utilizada para devolver el valor más bajo de un campo especificado.- |

## 5.6 Valores nulos.

- Indica que el valor es desconocido.
- No hay dos valores NULL que sean iguales.
- La comparación entre dos valores NULL, o entre un valor NULL y cualquier otro valor, tiene un resultado desconocido porque el valor de cada NULL es desconocido.
- Cuando se ven los resultados de la consulta en el Editor de código de SQL Server Management Studio, los valores NULL se muestran como (null) en el conjunto de resultados.

## 5.7 Subconsultas anidadas.

Una subconsulta es una instrucción SELECT anidada dentro de una instrucción SELECT, SELECT...INTO, INSERT...INTO, DELETE, o UPDATE o dentro de otra subconsulta. Puede utilizar tres formas de sintaxis para crear una subconsulta:

- Comparación [ANY | ALL | SOME] (instrucción SQL)
  Es una expresión y un operador de comparación que compara la expresión con el resultado de la subconsulta.
- Expresión [NOT] IN (instrucción SQL)
  Es una expresión por la que se busca el conjunto resultante de la subconsulta.
- [NOT] EXISTS (instrucción SQL)
  Es una instrucción SELECT, que sigue el mismo formato y reglas que cualquier otra instrucción SELECT. Debe ir entre paréntesis.

## 5.8 Consultas complejas.

Operaciones o consultas que combinan distintos tipos de operaciones sobre conjuntos o funciones avanzadas

Supongamos como ejemplo el siguiente código

```
SELECT nombre FROM empleados
WHERE salario > 5000 AND departamento IN ('Ventas', 'IT');
```

## 5.9 Vistas.

Una vista es una tabla virtual creada mediante una consulta que define sus filas y columnas. A diferencia de las tablas físicas, una vista no almacena datos, sino que genera dinámicamente los resultados de la consulta cada vez que se accede a ella. Puede utilizar datos de una o varias tablas e incluso de otras vistas en la misma o distintas bases de datos, permitiendo también consultas distribuidas para obtener datos de fuentes heterogéneas. Esto facilita la combinación de información, por ejemplo, consolidando datos similares de diferentes regiones.

## 5.10 Modificación de la base de datos.

Los “DML” (Data Manipulation Language) lenguaje de manipulación de datos, que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

| Comando | **Descripción del Comando** |
|---|---|
| **SELECT** | Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado.- |
| **INSERT** | Utilizado para cargar lotes de datos en la base de datos en una única operación.- |
| **UPDATE** | Utilizado para modificar los valores de los campos y registros Especificados.- |
| **DELETE** | Utilizado para eliminar registros de una tabla de una base de datos.- |

## 5.11 Reunión de relaciones.

Estas cláusulas son utilizadas para la vinculación entre tablas siempre y cuando haya una relación establecida entre dos tablas

### 5.11.1 Inner Join

Las vinculaciones entre tablas se realizan mediante la cláusula INNER que combina registros de dos tablas siempre que haya concordancia de valores en un campo común.
Su sintaxis es:

**SELECT campos FROM tb1 INNER JOIN tb2 ON tb1.campo1 comp tb2.campo2**

tb1 y tb2: Son los nombres de las tablas desde las que se combinan los registros.

Se puede utilizar una operación INNER JOIN en cualquier cláusula FROM. Esto crea una combinación por equivalencia, conocida también como unión interna.

### 5.11.2 Left Join

La Cláusula LEFT toma todos los registros de la tabla de la izquierda aunque no tengan ningún registro en la tabla de la derecha..

### 5.11.3 Right Join

La Cláusula RIGHT realiza la misma operación pero al contrario, toma todos los registros de la tabla de la derecha aunque no tenga ningún registro en la tabla de la izquierda.

### 5.11.4 Full Join

La Cláusula Full toma todos los registros de ambas tablas, sin importar si hay o no registros

# 5.12 Tipos de datos y esquemas.

Los tipos de datos permitidos en sql se mencionaron previamente aquí

Los tipos de datos SQL se clasifican en 13 tipos de datos primarios y de varios sinónimos válidos reconocidos por dichos tipos de datos.

- **BINARY**, 1 byte, Para consultas sobre tabla adjunta de productos de bases de datos que definen un tipo de datos Binario.
- **BIT**, 1 byte Valores Si/No ó True/False
- **BYTE**, 1 byte Un valor entero entre 0 y 255.
- **COUNTER**, 4 bytes Un número incrementado automáticamente (de tipo Long)
- **CURRENCY**, 8 bytes Un entero escalable entre 922.337.203.685.477,5808 y 922.337.203.685.477,5807.
- **DATETIME**, 8 bytes Un valor de fecha u hora entre los años 100 y 9999.
- **SINGLE**, 4 bytes Un valor en punto flotante de precisión simple con un rango de -3.402823*1038 a -1.401298*10-45 para valores negativos, 1.401298*10-45 a 3.402823*1038 para valores positivos, y 0.
- **DOUBLE**, 8 bytes Un valor en punto flotante de doble precisión con un rango de -1.79769313486232*10308 a -4.94065645841247*10-324 para valores negativos, 4.94065645841247*10-324 a 1.79769313486232*10308 para valores positivos, y 0.
- **SHORT**, 2 bytes Un entero corto entre -32,768 y 32,767
- **LONG**, 4 bytes Un entero largo entre -2,147,483,648 y 2,147,483,647.
- **LONGTEXT**, 1 byte por carácter de cero a un máximo de 1.2 gigabytes.
- **LONG BINARY**, Según se necesite de cero 1 gigabyte. Utilizado para objetos OLE.
- **TEXT**, 1 byte por carácter de cero a 255 caracteres.

## 5.12.1 Esquemas

Un **esquema** es una descripción formal de la estructura de una base de datos, que incluye las tablas, relaciones, vistas, índices, procedimientos almacenados y restricciones. Define la organización de los datos y establece las reglas para su manipulación.

**Tipos de esquemas**

Según la arquitectura de bases de datos (como el modelo de tres niveles), los esquemas se clasifican en:

1. **Esquema conceptual**:
    - Representa la estructura lógica global de la base de datos.
    - Incluye entidades, relaciones, atributos y restricciones.
    - Independiente del modelo físico o de implementación.
    - Ejemplo:
        - Entidades como Empleado, Departamento.
        - Relación: "Un empleado trabaja en un departamento".

2. **Esquema lógico**:
    - Traducción del esquema conceptual al modelo relacional.
    - Define tablas, columnas, claves primarias y claves externas.

   Ejemplo:
```
CREATE TABLE Empleado (
   id INT PRIMARY KEY,
   nombre VARCHAR(50),
   salario DECIMAL(10, 2),
   id departamento INT,
   FOREIGN KEY (id departamento) REFERENCES Departamento(id)
);
```

3. **Esquema físico**:
    - Describe cómo se almacenan los datos físicamente en el sistema de archivos.
    - Incluye índices, particiones, organización de datos y ajustes específicos del sistema gestor.
    - Ejemplo:
        - Índices para optimizar consultas:
          CREATE INDEX idx_nombre_empleado ON Empleado(nombre);

## 5.13 Restricciones de integridad.

La cláusula CONSTRAINT se usa en las instrucciones ALTER TABLE y CREATE TABLE para crear o eliminar restricciones. Hay dos tipos de cláusulas CONSTRAINT: uno para crear una restricción en un único campo y otro para crear una restricción en varios campos.

| Tipo de índice | Descripción |
|---|---|
| UNIQUE | Genera un índice de clave única. Lo que implica que los registros de la tabla no pueden contener el mismo valor en los campos indexados. |
| PRIMARY KEY | Genera un índice primario el campo o los campos especificados. Todos los campos de la clave principal deben ser únicos y no nulos, cada tabla sólo puede contener una única clave principal. |
| FOREIGN KEY | Genera un índice externo (toma como valor del índice campos contenidos en otras tablas). Si la clave principal de la tabla externa consta de más de un campo, se debe utilizar una definición de índice de múltiples campos, listando todos los campos de referencia, el nombre de la tabla externa, y los nombres de los campos referenciados en la tabla externa en el mismo orden que los campos de referencia listados. Si los campos referenciados son la clave principal de la tabla externa, no tiene que especificar los campos referenciados, predeterminado por valor, el motor Jet se comporta como si la clave principal de la tabla externa fueran los campos referenciados . |

## 5.14 Autorización.

Los “DCL” (Data Control Language), lenguaje de control de datos, contiene elementos útiles para trabajar en un entorno multiusuario, en el que es importante la protección de los datos, la seguridad de las tablas y el establecimiento de restricciones en el acceso, así como elementos para coordinar la compartición de datos por parte de usuarios concurrentes, asegurando que no interfieren unos con otros.

Estas cláusulas son:

- GRANT: Concede privilegios específicos a un usuario o grupo existente.
- REVOKE: Revoca privilegios específicos a un usuario o grupo existente.
- DENY: Deniega un permiso a una entidad de seguridad. Evita que la entidad de seguridad herede permisos por su pertenencia a grupos o roles. DENY tiene prioridad sobre todos los permisos, salvo que DENY no se aplique a los propietarios de objetos o miembros del rol fijo de servidor sysadmin.

En SQL Server, todos los elementos protegibles tienen permisos asociados que se pueden asignar a entidades de seguridad. La gestión de permisos en el Motor de base de datos se lleva a cabo tanto a nivel de servidor, mediante la asignación a inicios de sesión y roles de servidor, como a nivel de base de datos, mediante la asignación a usuarios y roles específicos de la base de datos. Esta estructura permite un control granular sobre las acciones que pueden realizarse en los diferentes objetos y recursos del sistema.

## 5.15 SQL incorporado.

El **SQL incorporado** (o *Embedded SQL*) es una forma de escribir consultas SQL directamente en el código fuente de un lenguaje de programación como C, Java o COBOL. Este enfoque permite que las consultas se preparen y se ejecuten durante la compilación de la aplicación.

**Características principales:**

1. **Compilación estática**:
    - Las consultas SQL se verifican y procesan en tiempo de compilación.
2. **Interacción directa con variables del programa**:
    - Los datos recuperados de la base de datos pueden asignarse directamente a variables del lenguaje anfitrión.
3. **Declaraciones predefinidas**:
    - Las sentencias SQL deben estar claramente delimitadas en el código (por ejemplo, usando EXEC SQL y END-EXEC).

## 5.16 SQL Dinámico.

El **SQL dinámico** permite construir y ejecutar consultas SQL en tiempo de ejecución. Es ideal para escenarios donde las consultas no se conocen de antemano o dependen de la entrada del usuario.

**Características principales:**

1. **Flexibilidad**:
    - Las consultas se generan dinámicamente según las necesidades de la aplicación.
2. **Ejecución en tiempo de ejecución**:
    - Las sentencias se procesan y validan cuando la aplicación está en ejecución.
3. **Seguridad**:
    - Requiere precaución para evitar problemas como *SQL Injection*.

# 5.17 Funciones y procedimientos.

Teniendo presente estas observaciones para poder diferenciarlas:

- Los procedimientos almacenados están orientados a ejecutar operaciones complejas de manipulación de datos, mientras que las funciones se enfocan en realizar cálculos o transformaciones que devuelven un valor o conjunto de valores.
- Capacidades de modificación: Los procedimientos almacenados permiten modificar datos en las tablas, a diferencia de las funciones.
- Invocación: Los procedimientos almacenados se ejecutan con EXEC, mientras que las funciones se llaman dentro de una consulta.

## 5.17.1 Procedimientos

Un procedimiento almacenado es un conjunto de instrucciones SQL que se almacenan en el servidor y se pueden ejecutar de forma repetida. Permiten la reutilización de código en operaciones complejas que se realizan frecuentemente. Pueden, además, mejorar el rendimiento y la seguridad del sistema.

Los procedimientos se crean con la sentencia 'CREATE PROCEDURE' de la siguiente forma:

```
CREATE PROCEDURE altaCategoria
```

Aceptan parámetros de entrada y pueden retornar valores o incluso tablas.

```
--Parámetros y valores por defecto
@Descripcion    nvarchar(100)   = NULL,
@Estado         INT             = NULL,
@Fecha_registro DATE            = NULL
```

Permiten la manipulación de datos mediante sentencias INSERT, UPDATE o DELETE; además del manejo de excepciones usando ‘BEGIN TRY’ y ‘BEGIN CATCH’

```
BEGIN TRY
    BEGIN TRANSACTION; --Aplicamos una transaccion entonces el id no autoincrementara
        INSERT INTO Categorias (Descripcion_Categoria, Estado, Fecha_Registro) VALUES
        (@Descripcion, @Estado, @Fecha_registro);
    COMMIT TRANSACTION; --Si todo salio bien entonces subimos el cambio
END TRY
BEGIN CATCH
    PRINT 'Error al insertar la categoria';
        ROLLBACK TRANSACTION; -- Si algo falla regresamos
END CATCH;
```

Los procedimientos se ejecutan usando el comando ‘EXEC’ seguido del nombre del procedimiento y sus parámetros.

### 5.17.2 Funciones

Las funciones son fragmentos de código que realizan un tarea y devuelve un valor o tabla. Al igual que los Procedimientos, pueden recibir parámetros de entrada y, a diferencia de estos, no permiten realizar operaciones de datos con INSERT, UPDATE o DELETE, limitándose a operaciones de lectura.

Las funciones se crean con el comando 'CREATE FUNCTION'

```
CREATE FUNCTION porcentajeDeVenta ( @Meses INT = 1)
```

Devuelven valores únicos o tablas y son más utilizadas para cálculos, conversiones o formato de datos

### Tipos de funciones:

- Funciones escalares: Devuelven un valor único, como un número o texto.
- Funciones con valor de tabla: Devuelve una tabla y se utilizan en operaciones de consulta más complejas.

# 6. UNIDAD 6

## 6.1 Almacenamiento e índices.

El almacenamiento y los índices son componentes fundamentales en los sistemas de bases de datos, ya que determinan cómo se almacenan los datos en el disco y cómo se accede eficientemente a ellos. La organización adecuada del almacenamiento y el uso de índices permiten mejorar significativamente el rendimiento de las consultas y operaciones.

# 6.2 Datos en almacenamiento externo.

Los datos de las bases de datos no se almacenan en memoria principal, sino en almacenamiento secundario (discos duros o SSDs).

Esto asegura persistencia y disponibilidad, incluso después de un fallo en el sistema.

**Características:**

1. **Acceso secuencial vs. aleatorio**:
    - El acceso secuencial es más rápido que el aleatorio debido a la naturaleza de los dispositivos de almacenamiento.
2. **Estructuras de bloques**:
    - Los datos se dividen en bloques que representan unidades de lectura/escritura desde y hacia el disco.

**Implicaciones:**

- Es necesario organizar los datos de forma que minimicen el tiempo de búsqueda y lectura.

# 6.3 Organizaciones de archivo e indexación.

## 6.3.1 Organizaciones de archivo:

Son métodos que determinan cómo se almacenan los registros en los bloques de disco.

1. **Archivo secuencial**:
    - Los registros se almacenan en un orden específico (por ejemplo, por una clave).
    - Beneficio: Acceso eficiente si se leen registros en orden.
    - Problema: Difícil de mantener si hay muchas inserciones o eliminaciones.
2. **Archivo de acceso directo (hashing)**:
    - Utiliza una función hash para determinar la ubicación del registro.
    - Beneficio: Acceso muy rápido para búsquedas por clave exacta.
    - Problema: Ineficiente para rangos de búsqueda.
3. **Archivo de múltiples tablas**:
    - Almacena múltiples tablas en un mismo archivo físico.
    - Beneficio: Minimiza la fragmentación de datos en bases de datos pequeñas.

## 6.3.2 Indexación:

Los índices son estructuras de datos que permiten localizar rápidamente registros sin tener que escanear toda la tabla.

- **Índices primarios**:
    - Basados en la clave primaria de una tabla.
    - La tabla debe estar ordenada por la clave primaria.
- **Índices secundarios**:
    - Apuntan a claves que no son la clave primaria.
    - Permiten búsquedas rápidas en columnas no ordenadas.
- **Índices compuestos**:
    - Utilizan más de una columna para generar el índice.

# 6.4 Estructuras de datos de índices.

## 6.4.1 Indexación basada en Hash:

Utiliza una función hash para mapear claves a ubicaciones específicas en el almacenamiento.

- **Ventajas**:
    - Acceso rápido para búsquedas exactas.
    - Eficiencia constante en promedio.
- **Desventajas**:
    - No es eficiente para rangos de búsqueda.
    - Colisiones: Cuando dos claves diferentes producen la misma ubicación.

## 6.4.2. Indexación basada en árboles:

Se basa en árboles balanceados como los árboles B y B +.

- **Árbol B**:
    - Estructura jerárquica donde cada nodo contiene un rango de claves.
    - Balanceado: Todos los caminos desde la raíz a las hojas tienen la misma longitud.
- **Árbol B +**:
    - Variante del árbol B con nodos hoja conectados en una lista enlazada.
    - Ideal para rangos de búsqueda porque permite acceso secuencial eficiente.

| Aspecto | Hashing | Árbol B/B+ |
|---|---|---|
| **Eficiencia de búsqueda** | Alta (claves exactas) | Alta (claves y rangos) |
| **Soporte para rangos** | No | Sí |
| **Balance de datos** | No necesario | Siempre balanceado |
| **Coste de mantenimiento** | Bajo | Moderado (por reequilibrio) |

## 6.5 Comparación entre las organizaciones de archivo.

| Organización | Ventajas | Desventajas |
|---|---|---|
| **Secuencial** | Eficiente para lecturas secuenciales | Ineficiente para búsquedas aleatorias |
| **Hash** | Rápido para búsquedas exactas | No admite búsquedas por rango |
| **Basada en árboles** | Admite búsquedas exactas y por rango | Más costoso en mantenimiento |

## 6.6 Técnicas Avanzadas

**Índices bitmaps**:

- Útiles para columnas con valores repetidos (por ejemplo, columnas booleanas).
- Almacenan bits en lugar de valores reales.

**Índices parciales**:

- Crean índices solo para un subconjunto de la tabla.
- Reducen espacio y tiempo de mantenimiento.

**Índices únicos y no únicos**:

- Los índices únicos garantizan que no haya duplicados en la columna indexada.

## 6.7 Indexación asociativa (hash) y basada en árboles (B y B +).

**Hashing:**

- Utilizado cuando las búsquedas exactas son frecuentes y se requiere rapidez.
- Ejemplo: Una tabla de usuarios con búsquedas por ID.

**Árbol B/B+:**

- Utilizado cuando las búsquedas incluyen rangos o acceso secuencial.
- Ejemplo: Búsquedas por fechas en registros de transacciones.

# 7. UNIDAD 7

## 7.1 Procesamiento y optimización de consultas.

El procesamiento y la optimización de consultas son aspectos críticos en el funcionamiento de los sistemas de bases de datos. Su objetivo es ejecutar consultas de manera eficiente, reduciendo los tiempos de respuesta y minimizando el consumo de recursos.

El **procesamiento de consultas** incluye todas las etapas necesarias para convertir una consulta SQL en un conjunto de operaciones físicas ejecutables por el sistema. Por otro lado, la **optimización de consultas** busca encontrar el plan de ejecución más eficiente entre varias alternativas.

**Objetivos clave:**

- Reducir el tiempo de ejecución de consultas.
- Minimizar el uso de recursos (CPU, memoria, I/O).
- Garantizar resultados correctos y consistentes.

## 7.2 Medidas del coste.

El coste de una consulta depende principalmente de las siguientes medidas:

1. **Número de accesos a disco**:
    - Lectura/escritura de bloques de datos.
    - Es el factor más significativo en bases de datos grandes.
2. **Uso de CPU**:
    - Procesamiento de operadores lógicos y aritméticos.
3. **Uso de memoria**:
    - Cantidad de memoria necesaria para las operaciones intermedias.
4. **Tiempo de comunicación**:
    - Aplicable en sistemas distribuidos.

**Ejemplo de factores que afectan el coste:**

- Tamaño de las tablas implicadas.
- Uso de índices.
- Tamaño de los bloques de disco.
- Cardinalidad (número de filas) y selectividad (porcentaje de filas que cumplen una condición).

# 7.3 Operación selección, ordenación y reunión.

## 7.3.1 Operación Selección:

- Extrae filas que cumplen una condición específica.
- **Optimización**:
    - Uso de índices (cuando están disponibles).
    - Evaluación temprana de predicados para reducir el número de filas.

## 7.3.2 Operación Ordenación:

- Reorganiza los datos en un orden específico (por ejemplo, por una clave primaria o columna).
- **Técnicas comunes**:
    - Algoritmos de ordenación externos (por ejemplo, *merge-sort*).
    - Ordenación interna para conjuntos pequeños.

## 7.3.3 Operación Reunión (Join):

- Combina datos de dos o más tablas basándose en una condición.
- **Métodos comunes**:
    - **Nested Loop Join**: Compara cada fila de una tabla con cada fila de la otra.
    - **Hash Join**: Utiliza una función hash para emparejar filas.
    - **Sort-Merge Join**: Ordena ambas tablas y luego las combina.

# 7.4 SQL al álgebra relacional.

El procesamiento de consultas traduce el lenguaje SQL en álgebra relacional, una representación intermedia que define las operaciones a realizar sobre las tablas.

**Ejemplo de traducción:**

```
SELECT nombre
FROM empleados
WHERE salario > 3000;
```

Álgebra relacional:

1. Selección: Filtrar filas con salario > 3000.
2. Proyección: Obtener solo la columna nombre.

**Optimización basada en álgebra relacional:**

- Reorganización de operaciones para minimizar costes.
- Aplicación temprana de operaciones restrictivas (como selección).

## 7.5 Optimización heurística.

La optimización heurística utiliza reglas predefinidas para transformar y reordenar las operaciones de una consulta con el fin de mejorar su eficiencia.

**Reglas comunes:**

1. Realizar **selecciones** lo antes posible.
2. **Proyecciones** tempranas para reducir el número de columnas procesadas.
3. Combinar operaciones equivalentes para evitar redundancia.
4. Reordenar **joins** basados en cardinalidad (procesar tablas más pequeñas primero).

## 7.6 Estimación de costes.

La estimación de costes evalúa el impacto de diferentes planes de ejecución antes de ejecutar una consulta.

**Factores evaluados:**

1. **Tamaño de la tabla**: Número de filas.
2. **Selectividad de condiciones**: Porcentaje de filas que cumplen cada predicado.
3. **Disponibilidad de índices**: Mejora las búsquedas y reducciones.
4. **Uso de memoria temporal**: Cantidad necesaria para almacenar datos intermedios.

**Uso de estadísticas:**

- Los sistemas de bases de datos mantienen estadísticas de tablas e índices para realizar estimaciones más precisas.

## 7.7 Índices y Funciones de Coste.

Los índices son fundamentales para mejorar el rendimiento de consultas en bases de datos grandes, los índices aceleran operaciones de búsqueda y reducción, pero su mantenimiento tiene un coste.

1. **Índices de clave primaria**:
    - Eficientes para búsquedas exactas.
2. **Índices secundarios**:
    - Soportan búsquedas en columnas no clave.
3. **Índices compuestos**:
    - Útiles para consultas con múltiples condiciones.

**Función de coste:**

Evalúa el tiempo estimado para ejecutar cada operación.

- **Acceso secuencial**: Alto coste en tablas grandes.
- **Acceso mediante índices**: Bajo coste, pero depende de la selectividad.

# 8. UNIDAD 8

## 8.1 Evolución y Comparación con Archivo.

Antes de los sistemas de bases de datos, las aplicaciones dependían de archivos planos, los cuales tienen limitaciones significativas:

- **Rigidez**: Cada aplicación define su propia estructura, dificultando cambios.
- **Redundancia**: Múltiples copias de la misma información en diferentes archivos.
- **Inconsistencia**: Modificaciones no sincronizadas en distintos lugares.

Los **DBMS (Sistemas de Gestión de Bases de Datos)** surgieron como una solución estructurada y eficiente, permitiendo:

- **Integración de datos**: Centralización y unificación.
- **Control de redundancia**: Eliminación de datos duplicados.
- **Acceso concurrente**: Múltiples usuarios pueden interactuar con los datos simultáneamente.

## 8.2 Importancia y Justificación de su uso de una Base de Datos.

Las bases de datos son esenciales para la gestión eficiente de información en diversos ámbitos (empresas, investigación, administración pública).

- **Ventajas**:
    - Mayor **control sobre los datos**.
    - **Seguridad** y control de acceso.
    - **Optimización** del almacenamiento.
    - **Facilidad de consultas** y generación de informes.

El volumen y complejidad de los datos han crecido exponencialmente a través del tiempo, exigiendo herramientas especializadas para un mejor manejo de los datos.

## 8.3 Definiciones y Uso en la Actualidad.

La definición es la presente aqui,
Una base de datos es un conjunto estructurado de datos que modela un aspecto del mundo real, accesible para múltiples aplicaciones.

Los usos comunes que tiene en la actualidad las bases de datos son:

1. **Sistemas transaccionales**: Bancos, e-commerce, gestión de inventarios.
2. **Análisis de datos**: Big Data y Business Intelligence.
3. **Sistemas en la nube**: Google Drive, OneDrive.
4. **Sistemas distribuidos**: Blockchain.

## 8.4 Comparación generalizada entre Modelos de Bases de Datos.

Existen varios modelos de bases de datos, cada uno adaptado a necesidades específicas:

| Modelo | Características | Uso Común |
|---|---|---|
| Relacional | Estructurado en tablas. Basado en álgebra relacional. | Gestión de datos transaccionales. |
| Jerárquico | Datos organizados en forma de árbol (padre-hijo). | Aplicaciones heredadas y sistemas legados. |
| En red | Más flexible que el jerárquico, permite relaciones múltiples. | Redes de transporte y telecomunicaciones. |
| Orientado a objetos | Integra características de programación orientada a objetos. | Aplicaciones multimedia. |
| NoSQL | Diseñado para grandes volúmenes de datos no estructurados. | Big Data, IoT y análisis de datos. |

## 8.5 Definición de un DBMS.

Un **Sistema de Gestión de Bases de Datos (DBMS)** es un software que permite crear, gestionar y manipular bases de datos de manera eficiente, proporcionando:

- **Interfaz** para usuarios y aplicaciones.
- **Seguridad** y control de acceso.
- **Consistencia** y gestión de transacciones.

**Ejemplos de DBMS:**

- **Relacionales**: MySQL, PostgreSQL, Oracle.
- **NoSQL**: MongoDB, Cassandra.

## 8.5.1 Componentes.

Un DBMS está formado por los siguientes componentes:

1. **Procesador de consultas**:
    - Interpreta y ejecuta las consultas de los usuarios.
2. **Gestor de almacenamiento**:
    - Controla cómo se almacenan los datos en disco.
3. **Gestor de transacciones**:
    - Garantiza la consistencia y atomicidad de las transacciones.
4. **Catálogo de metadatos**:
    - Contiene información sobre la estructura de los datos.

## 8.5.2 Objetivos.

Los objetivos de un DBMS son:

- **Facilitar el acceso** a los datos para usuarios finales.
- **Minimizar redundancia**.
- **Proteger los datos** de accesos no autorizados.
- **Permitir escalabilidad** para grandes volúmenes de datos.

## 8.5.3 Modelos de Datos.

Un modelo de datos define cómo se organizan, almacenan y acceden los datos. Los más comunes son:

1. **Relacional**.
2. **Jerárquico**.
3. **En red**.
4. **Orientado a objetos**.
5. **Documental (NoSQL)**.

### 8.5.4 Esquemas e Instancias.

#### 8.5.4.1 Esquema

Un esquema es la descripción estática de la estructura de la base de datos, como el diseño de tablas y relaciones.

#### 8.5.4.2 Instancia

Una instancia son los datos concretos almacenados en un momento específico.

### 8.5.5 Uniformidad e Independencia de Datos.

La **independencia de datos** es la capacidad del DBMS para modificar esquemas sin afectar las aplicaciones:

1. **Independencia lógica**: Cambiar el esquema lógico sin alterar las aplicaciones.
2. **Independencia física**: Modificar cómo se almacenan los datos sin cambiar el esquema lógico.

### 8.5.6 Conceptos del Entorno DBMS.

El entorno DBMS se compone de:

1. **Usuarios**:
    - Administradores.
    - Desarrolladores.
    - Usuarios finales.
2. **Aplicaciones**: Interfaces y programas que interactúan con el DBMS.
3. **Hardware**: Servidores y dispositivos de almacenamiento.
4. **Software**: El DBMS y sus herramientas asociadas.

# 9. UNIDAD 9

## 9.1 Introducción a la gestión de transacciones.

Una **transacción** en bases de datos es una unidad de trabajo lógica que se ejecuta como una serie de operaciones indivisibles. El objetivo principal es garantizar que las operaciones de una base de datos se ejecuten de manera consistente, incluso frente a fallos.

## 9.2 Las propiedades ACID.

Las transacciones deben cumplir las propiedades **ACID**, que son fundamentales para la integridad de los datos en sistemas de bases de datos:

- **Atomicidad**: Una transacción debe ejecutarse completamente o no ejecutarse en absoluto. Esto asegura que no haya estados parciales.
- **Consistencia**: Las transacciones deben mantener la consistencia del sistema antes y después de su ejecución.
- **Aislamiento**: Las operaciones de una transacción no deben ser visibles para otras transacciones hasta que estén completas.
- **Durabilidad**: Una vez confirmada, la transacción será persistente, incluso ante fallos en el sistema.

## 9.3 Consistencia y aislamiento.

#### 9.3.1 Consistencia

La consistencia garantiza que el estado de la base de datos pase de un estado válido a otro válido tras una transacción, respetando todas las restricciones establecidas (integridad referencial, restricciones de clave).

#### 9.3.2 Aislamiento

El aislamiento protege las transacciones concurrentes para evitar problemas como:

- **Lecturas sucias**: Lectura de datos modificados por una transacción no confirmada.
- **Lecturas no repetibles**: Relectura de datos obteniendo valores diferentes debido a cambios realizados por otras transacciones.
- **Fantasmas**: Nuevas filas introducidas por otra transacción durante la ejecución.

## 9.4 Atomicidad y durabilidad.

#### 9.4.1 Atomicidad

La Atomicidad asegura que, en caso de fallo, la base de datos vuelva a su estado previo a la transacción.

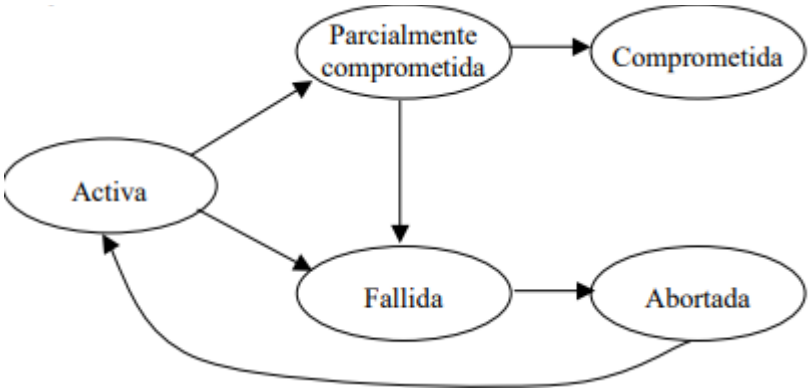
#### 9.4.2 Durabilidad

La Durabilidad es que los cambios realizados por una transacción confirmada son permanentes, incluso si el sistema falla posteriormente.

# 9.5 Transacciones y planificaciones.

## 9.5.1 Estados de una transacción

- Activa: Durante su ejecución Ficheros y bases de datos 7-3
- Parcialmente comprometida: Después de ejecutar su última instrucción.
- Fallida: Imposible de continuar su ejecución normal.
- Abortada: Transacción retrocedida y base de datos restaurada al estado anterior a su ejecución. Se puede reiniciar o cancelar.




## 9.5.2 Planificaciones

Una **planificación** (schedule) es el orden en que se ejecutan las operaciones de varias transacciones concurrentes.

- **Planificaciones serializables**: Son equivalentes a una ejecución secuencial, manteniendo la consistencia.
- **Planificaciones no serializables**: Pueden generar inconsistencias debido a interacciones no controladas entre transacciones.

# 9.6 Ejecución concurrente de transacciones.

La ejecución concurrente de transacciones SQL es la capacidad de un sistema de base de datos para manejar varias transacciones o consultas al mismo tiempo sin afectar la precisión o integridad de los datos

# 9.7 Motivación para la ejecución concurrente.

Una motivación para la ejecución concurrente aumenta el rendimiento general del sistema, pero también introduce complejidades como conflictos de acceso a datos, algunos de estos problemas comunes son:

**Problemas comunes:**

1. **Condiciones de carrera**.
2. **Bloqueos mutuos (deadlocks)**.

## 9.8 Técnicas de recuperación de bases de datos.

Las técnicas de recuperación aseguran que las transacciones fallidas no afecten la integridad de la base de datos. Algunas técnicas comunes son:

- **Registros de transacciones (logs)**: Registro de operaciones para reversiones o confirmaciones.
- **Puntos de control (checkpoints)**: Momentos seguros desde los cuales reiniciar después de fallos.
- **Protocolo Undo/Redo**:
    - **Undo**: Revierte cambios no confirmados.
    - **Redo**: Reaplica cambios confirmados.

## 9.9 Rendimiento del bloqueo.

El bloqueo es una técnica de control de concurrencia que garantiza el aislamiento de las transacciones:

- **Bloqueos exclusivos**: Impiden el acceso a los datos bloqueados por otras transacciones.
- **Bloqueos compartidos**: Permiten lecturas concurrentes pero no modificaciones.

**Impacto en el rendimiento:**

- **Ventaja**: Mejora la consistencia.
- **Desventaja**: Puede reducir el rendimiento debido a la contención y bloqueos mutuos.

## 9.10 Soporte de transacciones en SQL.

El estándar SQL proporciona soporte para transacciones mediante comandos como:

- **BEGIN TRANSACTION**: Inicia una nueva transacción.
- **COMMIT**: Confirma los cambios realizados por la transacción.
- **ROLLBACK**: Revierte los cambios realizados por la transacción.

## 9.11 Creación y terminación de transacciones

**Creación**

Las transacciones pueden iniciarse explícitamente mediante comandos SQL o implícitamente por el DBMS.

**Terminación**

Se realiza mediante:

- **COMMIT**: Confirmación de cambios.
- **ROLLBACK**: Reversión de cambios ante errores.

# 10. UNIDAD 10

## 10.1 Control y seguridad.

El **control y seguridad** en bases de datos se centra en proteger los datos contra accesos no autorizados, modificaciones indebidas y garantizar la integridad y confidencialidad de la información.
En SQL Server, todos los elementos protegibles tienen permisos asociados que se pueden asignar a entidades de seguridad

## 10.2 Introducción a la seguridad de las bases de datos.

La seguridad de las bases de datos implica:

1. **Protección del acceso**: Evitar accesos no autorizados.
2. **Confidencialidad**: Restringir el acceso a usuarios o procesos autorizados.
3. **Integridad**: Garantizar que los datos sean consistentes y confiables.
4. **Disponibilidad**: Asegurar que los datos estén disponibles para los usuarios autorizados.

**Amenazas comunes:**

- **Intrusiones externas**: Hackers o actores maliciosos.
- **Errores internos**: Fallos humanos o lógicos.
- **Accesos no controlados**: Usuarios con permisos excesivos.

## 10.3 Control de acceso.

El control de acceso en SQL es un conjunto de políticas y mecanismos que regulan la interacción de los usuarios con una base de datos

El control de acceso regula quién puede ver o manipular datos específicos. Los enfoques principales incluyen:

1. **Control discrecional de acceso (DAC)**.
2. **Control obligatorio de acceso (MAC)**.

## 10.4 Control discrecional de acceso.

En el DAC, el propietario del recurso tiene autoridad para conceder o restringir el acceso. Este modelo es flexible pero más vulnerable.

**Características:**

- Basado en permisos asignados explícitamente (lectura, escritura, ejecución).
- Implementado en bases de datos mediante comandos como **GRANT** y **REVOKE**.

Ejemplo:

```
GRANT SELECT, INSERT ON empleados TO usuario1;
REVOKE INSERT ON empleados FROM usuario1;
```

**Ventaja:**

- Fácil de implementar en sistemas multiusuario.

**Desventaja:**

- Puede ser vulnerable a errores de configuración y abuso de permisos.

## 10.5 Concesión y revocación de vistas y restricciones de integridad.

Las vistas son una herramienta importante para el control discrecional, ya que limitan el acceso a un subconjunto de datos.

**Información sobre las vistas aquí**

**Ejemplo:**

```
CREATE VIEW vista_empleados AS
SELECT nombre, departamento FROM empleados WHERE departamento =
'Ventas';
GRANT SELECT ON vista_empleados TO usuario2;
```

**Restricciones de integridad**: Garantiza que las operaciones de los usuarios no violen reglas definidas, como claves primarias, claves externas y otros límites.

## 10.6 Control obligatorio de acceso.

El MAC regula el acceso según políticas estrictas definidas por el administrador, basado en niveles de seguridad predefinidos.

- Las clasificaciones de seguridad (por ejemplo, confidencial, secreto) se asignan a usuarios y datos.
- Impide que usuarios con menor nivel accedan a datos más sensibles.

Ejemplo:

En un sistema MAC, un usuario con nivel "Confidencial" no podrá acceder a datos clasificados como "Secreto".

**Ventaja:**

- Mayor control y seguridad en entornos críticos.

**Desventaja:**

- Menos flexible y requiere configuraciones más complejas.

## 10.7 Relaciones multinivel y poli-instanciación.

### 10.7.1 Relaciones multinivel:

Permiten gestionar datos con diferentes niveles de seguridad en una misma base de datos.

### 10.7.2 Poli-instanciación:

Ocurre cuando múltiples usuarios con diferentes niveles de acceso observan diferentes versiones de los datos para garantizar confidencialidad.

**Ejemplo:**

En una base de datos militar, un usuario con nivel bajo ve que "Inventario de tanques = 20", mientras que otro usuario con acceso alto ve "Inventario de tanques = 200".

# 11. UNIDAD 11

## 11.1 Fundamentos de bases de datos NOSQL.

Las bases de datos **NoSQL** surgen como una alternativa al modelo relacional, diseñadas para manejar grandes volúmenes de datos, estructuras flexibles y aplicaciones distribuidas. A continuación, se describen los conceptos, tipos, ventajas y tendencias de estas bases de datos

## 11.2 Conceptos.

NoSQL (Not Only SQL) es un paradigma de bases de datos que no utiliza modelos tabulares tradicionales. Estas bases de datos son diseñadas para ofrecer:

- **Alta escalabilidad**.
- **Flexibilidad en el esquema**.
- **Rendimiento mejorado** en operaciones específicas.

Las características claves del NOSQL son:

1. **Modelo de datos flexible**: No requiere esquemas predefinidos.
2. **Alta disponibilidad**: Optimizado para entornos distribuidos.
3. **Tolerancia a fallos**: Diseñado para entornos con grandes volúmenes de datos y múltiples usuarios.

## 11.3 Límites del SQL.

Los límites del SQL son los siguientes:

1. Rigidez en el esquema relacional.
2. Dificultad para manejar grandes volúmenes de datos distribuidos.
3. Problemas de rendimiento en aplicaciones con operaciones frecuentes de lectura/escritura.

## 11.4 Tipos de bases de datos NoSQL.

### 11.4.1 Bases de datos de documentos:

Utilizan documentos en formato JSON, BSON o XML para almacenar datos.

- Ejemplo: MongoDB, CouchDB.

- **Ventajas**:
  - Fácil manejo de datos semiestructurados.
  - Relación directa con objetos en programación.
- **Ejemplo de uso**: Una tienda en línea para almacenar catálogos de productos con campos dinámicos.

### 11.4.2 Bases de datos de grafos:

Modelan datos como nodos (entidades) y aristas (relaciones).

- Ejemplo: Neo4j, ArangoDB.
- **Ventajas**:
  - Ideal para analizar redes sociales, sistemas de recomendación y conexiones complejas.
- **Ejemplo de uso**: Una red social para mapear amigos, intereses y relaciones entre usuarios.

### 11.4.3 Bases de datos clave-valor:

Almacenan pares clave-valor para un acceso rápido.

- Ejemplo: Redis, DynamoDB.
- **Ventajas**:
  - Simplicidad y rapidez.
- **Ejemplo de uso**: Caché para almacenar sesiones de usuario en una aplicación web.

### 11.4.4 Bases de datos orientadas a columnas:

Organizan datos por columnas en lugar de filas, optimizando operaciones analíticas.

- Ejemplo: Apache Cassandra, HBase.
- **Ventajas**:
  - Excelente para aplicaciones de big data y análisis masivo.
- **Ejemplo de uso**: Un sistema de análisis financiero que procesa millones de transacciones.

### 11.4.5 Bases de datos de objetos:

Almacenan datos como objetos definidos por clases.

- Ejemplo: Db4o, ZODB.
- **Ventajas**:
  - Integra directamente con lenguajes de programación orientados a objetos.
- **Ejemplo de uso**: Aplicaciones en tiempo real con datos complejos, como simulaciones científicas.

## 11.5 Historia y tendencias.

### 11.5.1 Historia

- **Década de 2000**: Las empresas tecnológicas, como Google, Amazon y Facebook, enfrentaron desafíos con bases de datos relacionales debido al crecimiento exponencial de datos.
- **2009**: Se populariza el término "NoSQL", impulsado por la necesidad de escalabilidad y flexibilidad.

### 11.5.2 Tendencias

1. **Adopción híbrida**:
    - Muchas empresas combinan SQL y NoSQL para aprovechar las fortalezas de ambos modelos.
2. **Bases de datos distribuidas**:
    - Soluciones como Apache Cassandra y DynamoDB ofrecen replicación global y alta disponibilidad.
3. **Integración con big data**:
    - Herramientas como Hadoop y Spark interactúan directamente con bases de datos NoSQL.
4. **Seguridad mejorada**:
    - Las bases de datos NoSQL evolucionan para cumplir con estándares de seguridad empresarial.
5. **Tecnologías emergentes**:
    - Bases de datos multilingües que soportan múltiples modelos de datos en una sola solución, como ArangoDB.

# ANEXO

## Comparación: SQL vs NoSQL

| Característica | SQL | NoSQL |
|---|---|---|
| **Modelo** | Relacional | No relacional |
| **Esquema** | Estricto | Flexible |
| **Escalabilidad** | Vertical | Horizontal |
| **Tipos de datos** | Estructurados | Semi/No estructurados |
| **Casos de uso** | ERP, CRM | Big Data, IoT, redes |

# Temas Técnicos

## Manejo de permisos a nivel de usuarios de base de datos

En SQL Server, todos los elementos protegibles tienen permisos asociados que se pueden asignar a entidades de seguridad. La gestión de permisos en el Motor de base de datos se lleva a cabo tanto a nivel de servidor, mediante la asignación a inicios de sesión y roles de servidor, como a nivel de base de datos, mediante la asignación a usuarios y roles específicos de la base de datos. Esta estructura permite un control granular sobre las acciones que pueden realizarse en los diferentes objetos y recursos del sistema.

**Para mas info aqui**

## Procedimientos y Funciones

Un procedimiento almacenado es un conjunto de instrucciones SQL que se almacenan en el servidor y se pueden ejecutar de forma repetida. Permiten la reutilización de código en operaciones complejas que se realizan frecuentemente. Pueden, además, mejorar el rendimiento y la seguridad del sistema.

Las funciones son fragmentos de código que realizan un tarea y devuelve un valor o tabla. Al igual que los Procedimientos, pueden recibir parámetros de entrada y, a diferencia de estos, no permiten realizar operaciones de datos con INSERT, UPDATE o DELETE, limitándose a operaciones de lectura.

**Tipos de funciones:**

- Funciones escalares: Devuelven un valor único, como un número o texto.
- Funciones con valor de tabla: Devuelve una tabla y se utilizan en operaciones de consulta más complejas.

**Para mas info aqui**

## Optimización de consultas a través de índices

Los índices son fundamentales para mejorar el rendimiento de consultas en bases de datos grandes. En un sistema de ventas, los índices pueden optimizar consultas frecuentes, como la búsqueda de productos, el registro de ventas, o la consulta de inventarios, haciendo que el sistema responda de manera más ágil.

- Existen dos tipos de índices: “Índices Agrupados” y los “Índices NO Agrupados”

### Índice Agrupado

Un índice agrupado organiza la información de una tabla de manera física, es decir, los datos se almacenan en el disco en el mismo orden en que están organizados por el índice. Solo puede haber un índice agrupado por tabla, ya que no se pueden ordenar físicamente los datos de más de una forma.

### Índice NO Agrupado

Un índice no agrupado no altera el orden físico de los datos en la tabla. En lugar de esto, crea una estructura separada que contiene las claves del índice y punteros a las ubicaciones físicas de los datos. Actúa como un índice de un libro en el cual puedes acceder a la información de forma directa

**Para mas info aqui**

## Índices columnares en SQL server

Los índices columnares permiten almacenar los datos por columnas, es decir, que primero se almacenarán uno tras otro los datos de la primera columna de todos los registros, luego se almacenará todos los datos de la segunda columna, y así sucesivamente hasta almacenar toda la información.

Una vez llegado el momento de rescatar la información, primero se accede al dato de la primer columna del primer registro, luego al dato de la primer columna del segundo registro, y así hasta leer todos los datos de la primer columna, tras lo cual se encontrará el dato de la segunda columna del primer registro, haciendo un proceso análogo hasta leer toda la información.

Esta forma de almacenar y acceder a los datos puede disminuir considerablemente los tiempos de respuesta de un sistema si se implementa de manera correcta.
En aquellos sistemas que se centran en consultas analíticas, que requieren de operaciones de agregación sobre un gran volumen de datos o un análisis de grandes cantidades de información, el uso sensato de índices reduce enormemente el tiempo de respuesta de las consultas.

Ejemplo de la creación de un índice columnar

- CREATE COLUMNSTORE INDEX idx_columnstore_indice ON tabla_refencia (Campos a los que referencia);

## Triggers

Un trigger es un procedimiento almacenado en la base de datos que se ejecuta automáticamente cada vez que ocurre un evento especial en la base de datos. Por ejemplo, un desencadenante puede activarse cuando se inserta una fila en una tabla específica o cuando ciertas columnas de la tabla se actualizan.

La creación de un disparador o trigger se realiza en dos pasos:
- En primer lugar, se crea la función disparadora.
- En segundo lugar, se crea el propio disparador SQL con el comando CREATE TRIGGER al que se introducen los parámetros para ejecutar la función disparadora.

Por lo general, estos eventos que desencadenan los triggers son cambios en las tablas mediante operaciones de inserción, eliminación y actualización de datos (insert, delete y update).

## Triggers DDL (Data Definition Language)
Esta clase de Triggers se activa en eventos que modifican la estructura de la base de datos (como crear, modificar o eliminar una tabla) o en ciertos eventos relacionados con el servidor, como cambios de seguridad o actualización de eventos estadísticos.

## Triggers DML (Data Modification Language)
Esta es la clase más común de Triggers. En este caso, el evento de disparo es una declaración de modificación de datos; podría ser una declaración de inserción, actualización o eliminación en una tabla o vista.

## Tipos de TRIGGER en SQL Server

AFTER
- Se ejecutan después de que una operación INSERT o UPDATE o DELETE haya ocurrido.
- Útiles para realizar acciones una vez que los datos ya están modificados en la base de datos, como registros en una tabla de auditoría.

INSTEAD OF
- Se ejecutan en lugar de la operación INSERT o UPDATE o DELETE
- Útiles cuando quieres anular o modificar el comportamiento predeterminado de la operación, o cuando quieres controlar complejamente qué datos se insertan o actualizan.

Ventajas:

- Generar automáticamente algunos valores de columna derivados.
- Aplicar la integridad referencial.
- Registro de eventos y almacenamiento de información sobre el acceso a la tabla. -
- Auditoría.
- Replicación sincrónica de tablas.
- Imponer autorizaciones de seguridad.
- Prevenir transacciones inválidas.

La instrucción CREATE TRIGGER permite crear un nuevo trigger que se activa automáticamente cada vez que ocurre un evento, como INSERT, DELETE o UPDATE en una tabla.

```
CREATE TRIGGER nombre_trigger
ON nombre_tabla (AFTER o INSTEAD OF) (INSERT o UPDATE o DELETE) AS
BEGIN
  -- SQL
END;
```

**Un TRIGGER no puede existir sin una tabla o vista asociada.**

## Manejo de tipos de datos JSON.

El manejo de JSON (JavaScript Object Notation) es relevante cuando se trabaja con integraciones de sistemas externos, ya que permite el intercambio de datos de manera eficiente y estructurada. JSON es útil para almacenar configuraciones de productos o integrar datos de ventas desde otras plataformas. Otro beneficio es que los datos de tipo JSON es su longitud que puede ser variable. Dentro de un Json se puede agregar tags nuevos e insertar nuevos datos sin afectar a los demás campos sin la necesidad de crear nuevas columnas.

En SQL Server, en lugar de un tipo de datos JSON explícito, se usa el tipo de datos NVARCHAR (normalmente NVARCHAR(MAX)) para almacenar los datos JSON. Luego, se utilizan funciones JSON nativas para manipular y consultar datos.

SQL Server proporciona funciones como JSON_VALUE, JSON_QUERY y OPENJSON para leer y manipular datos JSON.

- JSON_VALUE: Extrae un valor específico de un documento JSON.
- JSON_QUERY: Extrae un objeto o arreglo completo de un documento JSON.
- OPENJSON: Convierte un texto JSON en un conjunto de filas.

## Vistas y vistas indexadas

Una vista es una tabla virtual creada mediante una consulta que define sus filas y columnas. A diferencia de las tablas físicas, una vista no almacena datos, sino que genera dinámicamente los resultados de la consulta cada vez que se accede a ella. Puede utilizar datos de una o varias tablas e incluso de otras vistas en la misma o distintas bases de datos, permitiendo también consultas distribuidas para obtener datos de fuentes heterogéneas.

***Las vistas son útiles para simplificar y personalizar la percepción de la base de datos según el usuario y también actúan como un mecanismo de seguridad, permitiendo acceso a datos específicos sin exponer las tablas subyacentes. Además, las vistas pueden emular tablas cuyo esquema ha cambiado, crear interfaces compatibles con versiones anteriores, y mejorar el rendimiento mediante particiones de datos.***

Mientras que una vista indexada, también llamada vista materializada, es una vista que sí almacena datos en disco mediante la creación de un índice clusterizado basado en la consulta que la define. A diferencia de las vistas tradicionales, que solo actúan como consultas almacenadas, las vistas indexadas retienen los datos en la base de datos, funcionando de manera similar a una tabla. Lo que puede mejorar significativamente el rendimiento en consultas complejas o con grandes volúmenes de datos. Además, se pueden crear índices no clusterizados sobre la vista indexada para mejorar aún más el rendimiento en consultas específicas.

***Beneficios de las vistas indexadas:***

1. ***Mejoran el rendimiento de las consultas al almacenar agregaciones, evitando cálculos costosos durante la ejecución.***
2. ***Permiten almacenar combinaciones de tablas ya unidas (pre-join), facilitando el acceso a datos complejos de manera eficiente.***
3. ***Las combinaciones de uniones y agregaciones pueden materializarse, optimizando la recuperación de datos. Contras de las vistas indexadas:***
4. ***Introducen un importante sobrecargo en la base de datos, ya que cualquier cambio en las tablas base debe reflejarse en la vista indexada, lo cual consume recursos.***
5. ***Requieren mantenimiento adicional de los índices y estadísticas asociados, incrementando el costo en términos de espacio y recursos.***
6. ***Restricciones de uso: deben tener un índice único, solo permiten índices no clusterizados después de crear el índice clusterizado, y deben ser determinísticas (una sola salida posible para cada consulta).***

## Backup y restore

Un backup es una copia de seguridad de una base de datos o archivos que se realiza para proteger los datos frente a posibles pérdidas, fallos del sistema o corrupción. El objetivo de un backup es permitir la recuperación de la información en caso de un evento adverso.

La restauración (o restore) es el proceso de utilizar un backup previamente realizado para recuperar una base de datos o sistema a un estado anterior. Se emplea cuando ocurre una pérdida de datos o un fallo que requiere volver a una versión segura y funcional de la base de datos.

El backup en línea es un tipo de backup que se realiza mientras la base de datos está en funcionamiento y accesible para los usuarios. En el modo de recuperación FULL o BULK_LOGGED, permite realizar copias de seguridad sin interrumpir las operaciones de la base de datos, proporcionando continuidad operativa.

El log de transacciones es un registro detallado de todas las transacciones y cambios realizados en la base de datos. Permite la recuperación a un punto específico en el tiempo (restauración de un punto de recuperación) y es esencial para el proceso de restauración de datos incrementales. Este archivo de log captura cada operación y es clave para aplicar restauraciones de forma progresiva, asegurando que todos los cambios se implementen hasta un momento específico.

Efectividad de los backups (completos y de logs):

- Backup completo: Es una copia completa de la base de datos, esencial para restaurar el estado inicial antes de aplicar otros backups incrementales o de logs.
- Backup de logs de transacciones: Permite capturar los cambios que ocurren después de un backup completo. La efectividad de este tipo de backup radica en su capacidad para registrar los cambios de la base de datos y permitir la recuperación granular en un punto específico.

Observación clave: Modelo de recuperación: Configurar la base de datos en modo de recuperación FULL es esencial para permitir la captura de logs de transacciones y asegurar la capacidad de restaurar a puntos específicos.

## Manejo de transacciones y transacciones anidadas

Se habla un poco más de las transacciones aquí

Introducción a las Transacciones en SQL Server En SQL Server, una transacción es un conjunto de operaciones que se ejecutan de manera conjunta y deben considerarse como una unidad de trabajo. En términos generales, las transacciones son esenciales para preservar la coherencia y fiabilidad de la base de datos, ya que aseguran que las operaciones realizadas dentro de la transacción se completen en su totalidad o, de lo contrario, se reviertan si ocurre algún error.
Esto significa que una transacción es un mecanismo que permite mantener la base de datos en un estado consistente, incluso frente a errores o fallas durante la ejecución de operaciones.

**Tipos de Transacciones en SQL Server**

- Transacciones Implícitas: Inician automáticamente una nueva transacción cada vez que se ejecuta una instrucción que modifica datos, sin necesidad de comandos

explícitos para comenzar la transacción. El sistema trata cada instrucción DML (Data Manipulation Language) como una transacción separada.
- Transacciones Explícitas: Requieren que el usuario comience y termine explícitamente la transacción mediante comandos específicos (como BEGIN TRANSACTION, COMMIT TRANSACTION y ROLLBACK TRANSACTION).
- Transacciones Anidadas: Son transacciones que se inician dentro del contexto de una transacción en curso. SQL Server permite manejar múltiples niveles de transacciones, donde una transacción principal contiene una o más sub-transacciones.

Transacciones Anidadas en SQL Server Las transacciones anidadas permiten estructurar y dividir una operación compleja en varias sub-operaciones, cada una de las cuales puede considerarse como una transacción en sí misma. Esto facilita la modularización y el control de errores. Sin embargo, las transacciones anidadas en SQL Server funcionan como una única transacción, en el sentido de que el resultado final depende del estado de todas las sub-transacciones.

# Usuarios

SQL Server ofrece la opción de crear diferentes tipos de usuarios. Un usuario es una entidad de seguridad de la base de datos. Dependiendo del usuario, puede haber o no un inicio de sesión ligado a él. Un usuario y un inicio de sesión NO SON LO MISMO.
Un usuario puede tener permisos dentro de la base de datos, pero sin un inicio de sesión no podrá conectarse al servidor.
A continuación, se detallan los tipos de usuarios más comunes:
- SQL User sin Login: Usuario que existe solo en la base de datos y no tiene un inicio de sesión asociado a nivel de servidor.
- SQL User con Login: Usuario de la base de datos que está asociado a un inicio de sesión de SQL Server (Login). Este login puede ser de autenticación SQL o autenticación de Windows.
- Windows User: Usuario de Windows que tiene permisos de acceso a la base de datos a través de su cuenta de Active Directory.
- Application Role (Rol de Aplicación): Rol que se utiliza dentro de una aplicación para ejecutar con permisos específicos, sin estar ligado a un usuario individual.

## Inicio de sesión

Un inicio de sesión es una entidad de seguridad o una entidad que puede ser autenticada por un sistema seguro.
Los inicios de sesión deben estar asignados a un usuario para poder conectarse a una base de datos. Un inicio de sesión se puede asignar a bases de datos diferentes como usuarios diferentes, pero solo se puede asignar como un usuario en cada base de datos.
Alguno de los inicios de sesión de SQL Server son los siguientes:
- SQL Login: Usuario autenticado mediante SQL Server Authentication, donde se crea un nombre de usuario y una contraseña dentro de SQL Server.

- Windows Login: Usuario autenticado mediante Windows Authentication, utilizando las credenciales de Active Directory.
- Login de Certificado: Basado en certificados para la autenticación.
- Login de Asimetría (Asymmetric Key Login): Basado en claves asimétricas para la autenticación

### Permisos

Permisos a nivel de Usuario:

| | |
|---|---|
| Select | Permite ver o consultar datos (leer información de una tabla). |
| Insert | Permite agregar nuevos registros a una tabla. |
| Update | Permite modificar registros existentes. |
| Delete | Permite eliminar registros de una tabla. |
| Execute | Permite ejecutar procedimientos almacenados |

Algunos de los permisos a nivel servidor aquí

## Roles

Los roles son conjuntos predefinidos de permisos que pueden ser asignados a usuarios para facilitar la gestión de permisos. Al igual que los inicios de sesión, existen roles a nivel de Base de datos y a nivel del Servidor.

Roles predeterminados en Base de Datos aquí

Roles predeterminados a nivel de Servidor aquí

# Bibliografía Utilizada


1. Begg, C. E., Connolly, T. M. (2005). Sistemas de bases de datos: un enfoque práctico para diseño, implementación y gestión. España: Pearson Educación.
2. Date, C. J. (2001). Introducción a los sistemas de bases de datos. México: Pearson Educación. ISBN: 9684444192. [Disponible físicamente]

3. de Miguel Castaño, A. (2000). Diseño de bases de datos relacionales RA-MA Editorial. MADRID , España. ISBN 9701505263. [Disponible físicamente]
4. Elmasri, R. (2007). Fundamentos de bases de datos, 5a ed. Pearson. México. ISBN: 9788478290857. [Disponible físicamente]
5. Nieto Bernal, W. y Capacho Portilla, J. R. (2017). Diseño de base de datos. Barranquilla, Colombia: Universidad del Norte. https://elibro.net/es/lc/unne/titulos/70030. [Disponible E-Libro UNNE Virtual]
6. Piattini Velthuis, M. G., Martines, E. M., Muñoz, C. C., & Sánchez, B. V. (2007). Tecnología y diseño de bases de datos. España: RA-MA EDITORIAL. ISBN: 9789701512685. [Disponible físicamente]
7. Pulido Romero, E. Escobar Domínguez, Ó. y Núñez Pérez, J. Á. (2019). Base de datos. Ciudad de México, Grupo Editorial Patria. https://elibro.net/es/lc/unne/titulos/121283. [Disponible E-Libro UNNE Virtual]
8. Ramakrishnan, R. 2007. Sistemas de gestión de bases de datos, Tercera Edición. McGraw-Hill/Interamericana de España, S.A.U. ISBN: 9788448156381
9. Silberschatz, A., Korth, H. F., Sudarshan, S. (2014). Fundamentos de bases de datos, 6th Edición. España: McGraw-Hill Interamericana de España S.L. ISBN 9788448190330.
10. Aguirre Sánchez, M. J. (2021). Tecnologías de Seguridad en Bases de Datos: Revisión Sistemática (Bachelor's thesis). http://dspace.ups.edu.ec/handle/123456789/20566 [En linea, 2023]
11. Codd, E. F. (1972). Further normalization of the data base relational model. Data base systems, 6, 33-64.
12. Conesa Caralt, J. y Casas Roma, J. (2014). Diseño conceptual de bases de datos en UML. Barcelona, Spain: Editorial UOC. Recuperado de https://elibro.net/es/lc/unne/titulos/57635. [Disponible E-Libro UNNE Virtual]

13. Conesa Caralt, J., & Rodríguez-González, M. E. (2013). Bases de datos: conceptos básicos, diseño físico y rendimiento, septiembre 2013. http://hdl.handle.net/10609/77645 [En linea, 2023]
14. E. F. Codd. (1990). The relational model for database management: version 2. Addison-Wesley Longman Publishing Co., Inc., USA.
15. Meier, A., & Kaufmann, M. (2019). SQL & NoSQL databases. Berlin/Heidelberg, Germany: Springer Fachmedien Wiesbaden. ISBN: 9783658245481
16. Meier, A., & Kaufmann, M. (2023). SQL and NoSQL Databases: Modeling, Languages, Security and Architectures for Big Data Management. Estados Unidos: Springer Nature Switzerland. ISBN:9783031279072
17. Muñoz-Reja, I. C. Gómez Carretero, A. I. y Gualo Cejudo, F. (2018). Calidad de datos. Paracuellos de Jarama, Madrid, RA-MA Editorial. https://elibro.net/es/lc/unne/titulos/106516. [Disponible E-Libro UNNE Virtual]
18. Pérez Rodríguez, M. D. (Coord.) (2018). Nuevo reglamento europeo de protección de datos (RGPD). 1. Málaga, España, Editorial ICB. https://elibro.net/es/lc/unne/titulos/225258. [Disponible E-Libro UNNE Virtual]
19. Sánchez Serrano, F. R. (2010). Propuesta de un modelo de datos SQL multiplataforma basado en el estándar SQL: 2003 (Doctoral dissertation). http://tesis.ipn.mx/handle/123456789/10399 [En linea, 2023]
20. Tiebas, Javier. (2017). Desarrollo de un esquema de seguridad en SQL Server. (Trabajo Final de Posgrado. Universidad de Buenos Aires.) http://bibliotecadigital.econ.uba.ar/download/tpos/1502-0496_TiebasJ.pdf [En linea, 2023]