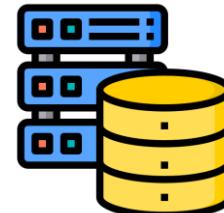


## Introducción al Master en SQL Server

# Requisitos

➤ *Conocimientos mínimos sobre base de datos.*



➤ *Sistema Operativo Windows*



➤ *¡Muchas ganas de aprender!*



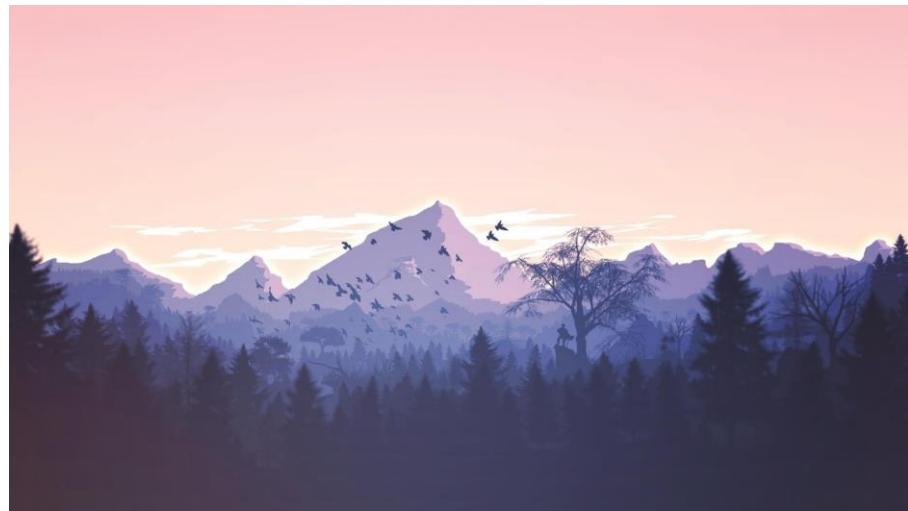
## Descripción de la Sección

Que son las Bases de Datos

Tipos de Base de datos SQL

Definición de Tabla, Campo y Registro

SQL(Structured Query Language)



# ¿Qué son las Bases de Datos?

# ¿Qué son las Bases de Datos?

Banco de Datos

Producto de la necesidad humana  
de almacenar la información



1  
Guías telefónicas



2  
Archivos personales



3  
Bibliotecas



4  
Historial médico



# ¿Qué son las Bases de Datos?

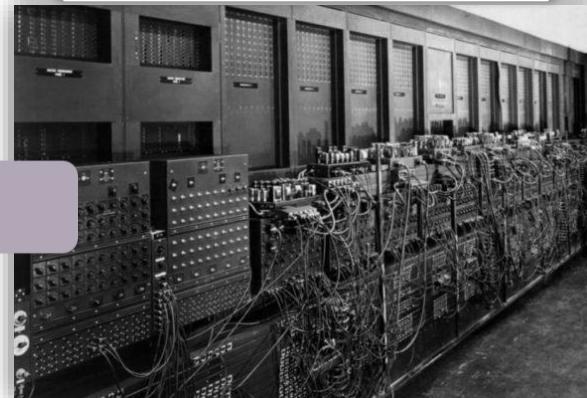
*Banco de Datos*

60

*Electrónica*



*Computación*



# ¿Qué son las Bases de Datos?

## *Banco de Datos*

Son sistemas organizados para almacenar y gestionar grandes cantidades de información

Fácil recuperación

manipulación

actualización

Aplicaciones  
personales

Sistemas  
empresariales



# Tipos de Bases de Datos

# Tipos de Bases de Datos

Bases de Datos Relacionales

*Bases de Datos SQL*

Bases de Datos No Relacionales

*Bases de Datos No SQL*

# Tipos de Bases de Datos

Bases de Datos Relacionales

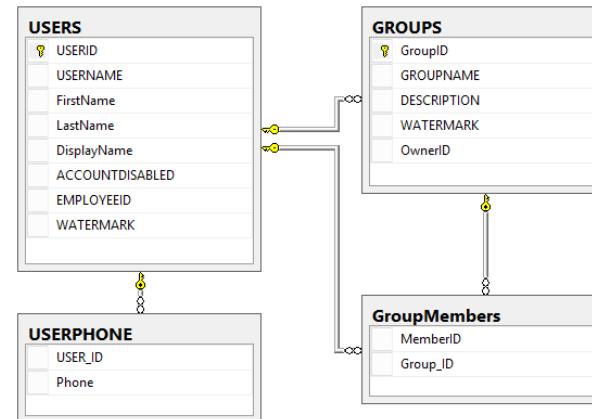
*Bases de Datos SQL*

Bases de Datos No Relacionales

*Bases de Datos No SQL*

Las Bases de Datos Relacionales son las más utilizadas como tecnología para la industria.

**SQL**



Inventario

Ventas

Finanzas

# Tipos de Bases de Datos

Bases de Datos Relacionales

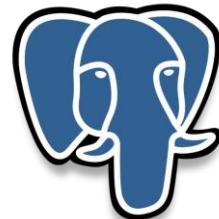
*Bases de Datos SQL*



ORACLE®

Bases de Datos No Relacionales

*Bases de Datos No SQL*



PostgreSQL



# Tipos de Bases de Datos

Bases de Datos Relacionales

*Bases de Datos SQL*

Ventajas

Estructura clara y definida

Consistencia e integridad de datos

Soporte maduro y establecido

Capacidad de consulta avanzada

Transacciones

Bases de Datos No Relacionales

*Bases de Datos No SQL*

Desventajas

Escalabilidad limitada en algunos casos

Cambios de esquema y flexibilidad

Costos y licencias

Rendimiento bajo cargas masivas

Complejidad en la gestión y mantenimiento

# Tipos de Bases de Datos

Bases de Datos Relacionales

*Bases de Datos SQL*

Bases de Datos No Relacionales

*Bases de Datos No SQL*



**SQL**

**API**

Las Bases de Datos No Relacionales NO son estructuradas,  
es decir no tienen un estructura bien definida.

# Tipos de Bases de Datos

Bases de Datos Relacionales

*Bases de Datos SQL*

Clave - Valor



Documentales



elastic



Couchbase

Bases de Datos No Relacionales

*Bases de Datos No SQL*

Grafos



Amazon  
Neptune



neo4j

Columnas



Amazon Redshift



cassandra

APACHE  
HBASE



# Tipos de Bases de Datos

Bases de Datos Relacionales

*Bases de Datos SQL*

Ventajas

Escalabilidad Horizontal

Flexibilidad de esquema

Rendimiento optimizado para cargas específicas

Gestión de Big Data

Adaptabilidad a las tendencias actuales

Bases de Datos No Relacionales

*Bases de Datos No SQL*

Desventajas

Falta de estandarización

Madurez relativa

Consistencia vs Disponibilidad

Menor énfasis en la integridad de datos

Curva de aprendizaje

## Definición de Tabla, Campo y Registro

# CLIENTES

nombre	paterno	materno	fecha nacimiento	direccion	ciudad
Alberto	Woods	Olsson	06/08/2000	Cra 50#32-16	Medellín
Ana	Williams	Coria	04/01/2001	Cra 100#6819	Bogotá
Sara	Borg	Byron	25/11/2003	Cra 72#95-20	Cali
...	...	...	...	...	...

Texto

Texto

Fecha

Texto

Texto

Campo o  
Columna

Fila o  
Tupla

Tabla

# SQL (Structured Query Language)

## SQL (Structured Query Language)

SQL es un lenguaje de programación estándar para interactuar con las bases de datos relacionales.



**(IBM) San José - California**



Donald D. Chamberlin y Raymond F. Boyce

## SQL (Structured Query Language)

SQL es un lenguaje de programación estándar para interactuar con las bases de datos relacionales.

### **Structured Query Language**

Lenguaje de Consulta Estructurado

### **SEQUEL**



### **SQL**

## SQL (Structured Query Language)

SQL es un lenguaje de programación estándar para interactuar con las bases de datos relacionales.

(70's)

1974 - Primer sistema de gestión de bases de datos relacional(IBM)

**System R**

# SQL (Structured Query Language)

SQL es un lenguaje de programación estándar para interactuar con las bases de datos relacionales.



**IBM → SQL**

**1977 ORACLE**  
(Relational Software, Inc)

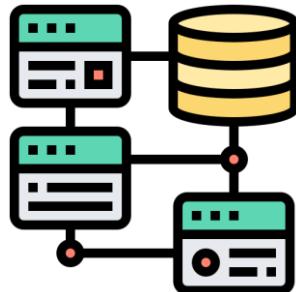
# SQL (Structured Query Language)

80

American National Standards Institute (ANSI)

Organization for Standardization (ISO)

1986



SQL-86, SQL-89, SQL-92

SQL:1999, SQL:2003, SQL:2006

SQL:2008, SQL:2011, SQL:2016

El estándar ANSI SQL es un lenguaje de programación maduro y estable

## SQL (Structured Query Language)

1987 – Sybase System (Microsoft)

1989 – Microsoft SQL Server 1.0

90

1995 – MySQL (PHP)

1996 – PostgreSQL



# SQL (Structured Query Language)



2000 – SQLite

2009 – Oracle compra Sun Microsystems

2009 – MariaDB

# SQL (Structured Query Language)

**SQL tiene los siguientes tipos de instrucciones**

**DDL**

**Data Definition  
Language**

Lenguaje de  
Definición de Datos

**DML**

**Data Manipulation  
Language**

Lenguaje de  
Manipulación de Datos

**DCL**

**Data Control  
Language**

Lenguaje de Control  
de Datos

**TCL**

**Transaction  
Control Language**

Lenguaje de control  
de transacciones

Estos 4 grupos de instrucciones son los que forman SQL

# SQL (Structured Query Language)

## DDL

Lenguaje de Definición de Datos

Para definir las estructura disponemos de tres sentencias:

**CREATE:** Crear Objetos

**ALTER:** Modificar Objetos

**DROP:** Eliminar Objetos

Definir las estructuras que almacenarán los datos.

# DML

Lenguaje de Manipulación de Datos

Los elementos que se utilizan para manipular los datos,  
son los siguientes:

**INSERT:** Insertar Datos

**UPDATE:** Modificar Datos

**DELETE:** Eliminar Datos



Realizar tareas de consultas o  
modificación de los datos

## SQL (Structured Query Language)

### DML

Lenguaje de Manipulación de Datos

Los elementos que se utilizan para manipular los datos,  
son los siguientes:

**SELECT:** Consultar Datos

DQL: Data Query Language  
Lenguaje de Consulta de Datos

## SQL (Structured Query Language)

### DCL

Lenguaje de Control de Datos

Los comandos para controlar los permisos son los siguientes:

**GRANT:** Otorga Permiso

**REVOKE:** Elimina Permiso

Administrar el Sistema Gestor de Base de Datos y controlar el acceso a los objetos

## SQL (Structured Query Language)

# TCL

Lenguaje de control de transacciones

Los comandos para controlar los permisos son los siguientes:

**COMMIT:** Confirma Transacción

**ROLLBACK:** Deshace Transacción

**SAVEPOINT:** Punto de Guardado

**SET TRANSACTION:** Inicializa una transacción

controlar las transacciones en una base de datos

## Control de Flujo

IF

ELSE

WHILE



MariaDB



ORACLE

## Álgebra Relacional

**SELECT**       $\text{col}_1, \text{col}_2, \dots, \text{col}_n$   
**FROM**         $\text{tabla}_a$   
**WHERE**       $\text{condición}_a$

## Álgebra Relacional

**SELECT**     $\text{col}_1, \text{col}_2, \dots, \text{col}_n$   
**FROM**       $\text{tabla}_a$   
**WHERE**      $\text{condición}_a$

**UNION** ————— **Unión de Conjuntos**

**SELECT**     $\text{col}_1, \text{col}_2, \dots, \text{col}_n$   
**FROM**       $\text{tabla}_a$   
**WHERE**      $\text{condición}_a$

## Álgebra Relacional

**SELECT**       $\text{col}_1, \text{col}_2, \dots, \text{col}_n$   
**FROM**         $\text{tabla}_a$   
**WHERE**       $\text{condición}_a$

**INTERSECT** → **Intersección de Conjuntos**

**SELECT**       $\text{col}_1, \text{col}_2, \dots, \text{col}_n$   
**FROM**         $\text{tabla}_a$   
**WHERE**       $\text{condición}_a$

## Álgebra Relacional

**SELECT**     $\text{col}_1, \text{col}_2, \dots, \text{col}_n$   
**FROM**       $\text{tabla}_a$   
**WHERE**     $\text{condición}_a$

**EXCEPT** —————→ **Diferencia de Conjuntos**

**SELECT**     $\text{col}_1, \text{col}_2, \dots, \text{col}_n$   
**FROM**       $\text{tabla}_a$   
**WHERE**     $\text{condición}_a$

## Álgebra Relacional

```
SELECT    col1, col2, ..., coln  
FROM      tablaa, tablab
```

## Álgebra Relacional

**SELECT**       $\text{col}_{1a}, \text{col}_{2a}, \dots, \text{col}_{1b}, \text{col}_{2b}$   
**FROM**         $\text{tabla}_a$   
**JOIN**  
 $\text{tabla}_b$   
**ON**     $\text{col}_{1a} = \text{col}_{1b}$

  
**Dominio Común**

# SQL (Structured Query Language)

## Comando SQL + Comandos Propios



**ORACLE**  
***PL/SQL***

***Transact-SQL***

# SQL (Structured Query Language)



Versión	Año	Nombre de la versión
1	1989	SQL Server 1.0
4.21	1993	SQL Server 4.21
6	1995	SQL Server 6.0
6.5	1996	SQL Server 6.5
7	1998	SQL Server 7.06
-	19	SQL Server 7.0
	99	OLAP Tools
8	2000	SQL Server 20007
8	20	SQL Server 2000
	03	64-bit Edition
9	2005	SQL Server 20058
10	2008	SQL Server 20089
10.25	2010	SQL Azure DB
10.5	2010	<u>SQL Server 2008 R210</u>
11	2012	<u>SQL Server 201211</u>
12	2015	<u>SQL Server 201412</u>
13	2016	SQL Server 2016
14	2017	SQL Server 2017
15	2019	SQL Server 2019
16	2022	SQL Server 2022

## Descripción de la Sección

¿Qué es Microsoft SQL Server?

Instalación de SQL Server 2022

Validar el estado del Servidor SQL Server

Instalar SQL Server Management Studio(SSMS)

Distintas formas de conectarse al Servidor

Solucionar Error al conectarse al Servidor

Bases de Datos del Sistema

Conociendo el entorno SSMS

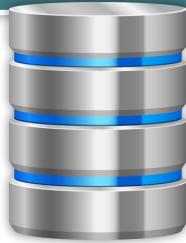


## ¿Qué es Microsoft SQL Server?

# ¿Qué es Microsoft SQL Server?



Sistema de Administración de Bases de Datos  
Relacionales (RDBMS)



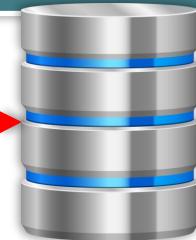
# ¿Qué es Microsoft SQL Server?



Sistema de Administración de Bases de Datos Relacionales (RDBMS)



T-SQL



Microsoft SQL Server es uno de los principales Sistemas de Administración de Bases de Datos Relacional del mercado



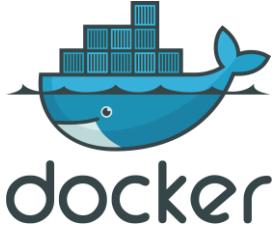
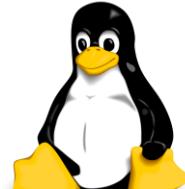
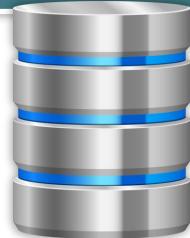
Es ideal para almacenar toda la información deseada.



# ¿Qué es Microsoft SQL Server?



Sistema de Administración de Bases de Datos  
Relacionales (RDBMS)



# ¿Qué es Microsoft SQL Server?



Sistema de Administración de Bases de Datos Relacionales (RDBMS)



## Enterprise

Siendo la edición más completa, está destinada para organizaciones con altos niveles de trabajo críticas

## Estándar

Destinada para ofrecer a las pequeñas empresas una herramienta que les brinde una administración básica de datos.

## Express

Diseñada para aquellas personas principiantes que deseen disponer de una base de datos gratuita como aprendizaje.

## Developer

Integra toda la funcionalidad del Enterprise, para entornos de prueba y no debe utilizarse en entorno de producción.

# ¿Qué es Microsoft SQL Server?



Sistema de Administración de Bases de Datos Relacionales (RDBMS)



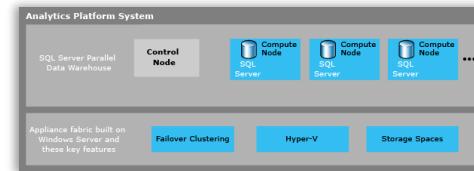
Azure SQL Database



Azure SQL Edge



Instancia administrada  
de Azure SQL



Microsoft Analytics Platform System



Azure Synapse Analytics

El motor de Base de Datos SQL Server también  
lo usan los siguientes productos y servicios

# ¿Qué es Microsoft SQL Server?



Sistema de Administración de Bases de Datos Relacionales (RDBMS)



SQL Server Integration Services(SSIS)

Algunos componentes y tecnologías en SQL Server son:

SQL Server Analysis Services(SSAS)

Master Data Services(MDS)

Data Quality Services(DQS)

SQL Server Reporting Services(SSRS)

# ¿Qué es Microsoft SQL Server?



Sistema de Administración de Bases de Datos Relacionales (RDBMS)



¿Qué es *SQL Server Management Studio*?

Es una interfaz gráfica de usuario que proporciona un entorno integrado para administrar cualquier infraestructura relacionada con SQL Server.

## Descripción de la Sección

Creando nuestra Primer Base de Datos

Introducción a los Tipos de Datos

Creando nuestra Primera Tabla

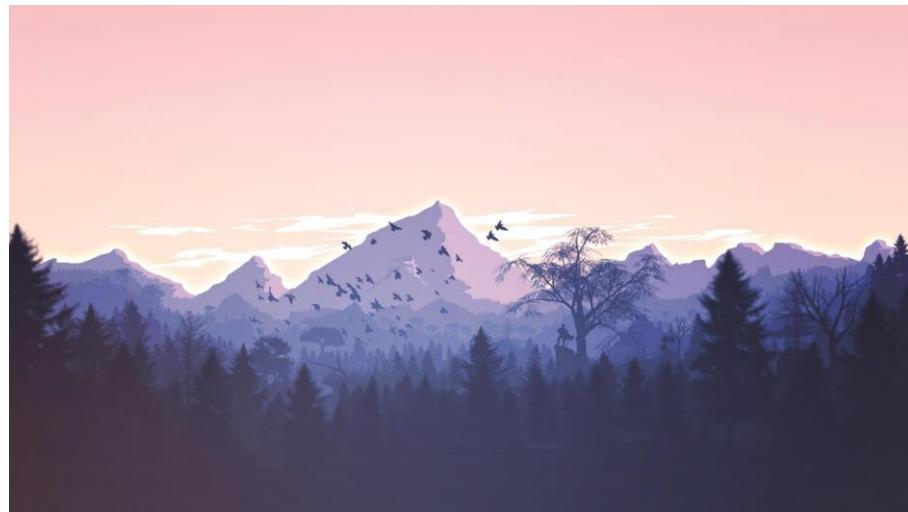
¿Qué es Primary Key?

¿Qué es la Propiedad IDENTITY ?

Introducción a Foreign Key

Diagrama de Base de Datos

Tabla con Primary Key Compuesta



# Introducción a los Tipos de Datos

# Introducción a los Tipos de Datos

Los Tipos de Datos son utilizados para definir el tipo de información que puede almacenarse en una columna de una Tabla.

Cada columna en una Tabla debe tener un Tipo de Datos específico que determine el formato y el rango de valores que puede contener.

# Introducción a los Tipos de Datos

*Numéricos Exactos - Entero*

**INT**

**4 Bytes**

DESDE -2,147,483,648 HASTA 2,147,483,647

*Numéricos Exactos - Decimal*

**DECIMAL(p, s)**

1 al 9 → 5 Bytes

10 al 19 → 9 Bytes

20 al 28 → 13 Bytes

29 al 38 → 17 Bytes

*Numéricos Exactos - Lógico*

**BIT**

**1 Byte**

“1” o “0”

# Introducción a los Tipos de Datos

## Fecha y hora

### TIME

5 Bytes

Formato: hh:mm:ss[.nnnnnn]

### DATE

3 Bytes

Formato: yyyy-MM-dd

### DATETIME

8 Bytes

Formato: yyyy-MM-dd hh:mm:ss[.nnn]

## Cadenas de caracteres

### CHAR(n)

Longitud Fija

Valor de n: 1 hasta 8000

Storage: n Bytes

*Donde “n” especifica el tamaño de la columna en “Bytes”.*

### VARCHAR(n)

Longitud Variable

Valor de n: 1 hasta 8000

Storage: n + 2 Bytes

# Primeros pasos en SQL Server

Profesores		
	nombre	VARCHAR(50)
	paterno	VARCHAR(50)
	materno	VARCHAR(50)
	fechaNacimiento	DATE
	domicilio	VARCHAR(50)
	telefono	INT
	email	VARCHAR(50)



Nombre del Campo



Tipo de Dato

# ¿Qué es Primary Key?

## ¿Qué es Primary Key?



Una primary key (clave primaria) en bases de datos es un campo o conjunto de campos que identifica de manera única cada fila en una tabla.

# ¿Qué es Primary Key?



Una primary key (clave primaria) en bases de datos es un campo o conjunto de campos que identifica de manera única cada fila en una tabla.

## Características:

- ❖ Sólo se permite una Primary Key por tabla.
- ❖ Los valores dentro de la Primary Key deben ser únicos; No se permiten duplicados.
- ❖ Todas las columnas de una clave primaria deben establecerse en NOT NULL.
- ❖ La creación de una Primary Key también crea un índice agrupado único en las columnas incluidas.

## ¿Qué es Primary Key?



Una Primary Key que consta de una sola columna se denomina **Primary Key ÚNICA**.

Cuando la Primary Key consta de varias columnas, se la conoce como **Primary Key COMPUESTA**.

## Establecer Primary Key

# Establecer Primary Key

Profesores		
	nombre	VARCHAR(50)
	paterno	VARCHAR(50)
	materno	VARCHAR(50)
	fechaNacimiento	DATE
	domicilio	VARCHAR(50)
	telefono	INT
	email	VARCHAR(50)

# Establecer Primary Key

Profesores		
nombre	VARCHAR(50)	
paterno	VARCHAR(50)	
materno	VARCHAR(50)	
fechaNacimiento	DATE	
domicilio	VARCHAR(50)	
telefono	INT	
email	VARCHAR(50)	

nombre	paterno	materno	fechaNacimiento	domicilio	telefono	email
Isabel	Campos	Torres	01/01/2000	Jr. Las flores 520	9999999999	isabel@example.com
Oscar	Gallegos	Flores	02/02/2000	Calle Balta 650	9888888888	oscar@example.com
Isabel	Guillen	Morales	03/03/2000	Calle Prado 150	9777777777	isabel@example.com

# Establecer Primary Key

Profesores		
nombre	VARCHAR(50)	
paterno	VARCHAR(50)	
materno	VARCHAR(50)	
fechaNacimiento	DATE	
domicilio	VARCHAR(50)	
telefono	INT	
email	VARCHAR(50)	

nombre	paterno	materno	fechaNacimiento	domicilio	telefono	email
Isabel	Campos	Torres	01/01/2000	Jr. Las flores 520	9999999999	isabel@example.com
Oscar	Gallegos	Flores	02/02/2000	Calle Balta 650	9888888888	oscar@example.com
Isabel	Guillen	Morales	03/03/2000	Calle Prado 150	9777777777	isabel@example.com

# Establecer Primary Key

Profesores		
PK	idProfesor	INT
	nombre	VARCHAR(50)
	paterno	VARCHAR(50)
	materno	VARCHAR(50)
	fechaNacimiento	DATE
	domicilio	VARCHAR(50)
	telefono	INT
	email	VARCHAR(50)

características especiales  
del campo

Nombre del Campo

Tipo de Dato

# Establecer Primary Key



Profesores		
PK	idProfesor	INT
	nombre	VARCHAR(50)
	paterno	VARCHAR(50)
	materno	VARCHAR(50)
	fechaNacimiento	DATE
	domicilio	VARCHAR(50)
	telefono	INT
	email	VARCHAR(50)

<u>idProfesor</u>	nombre	paterno	materno	fechaNacimiento	domicilio	telefono	email
101	Isabel	Campos	Torres	01/01/2000	Jr. Las flores 520	9999999999	isabel@example.com
520	Oscar	Gallegos	Flores	02/02/2000	Calle Balta 650	9888888888	oscar@example.com
602	Isabel	Guillen	Morales	03/03/2000	Calle Prado 150	9777777777	isabel@example.com

¿Qué es la Propiedad IDENTITY ?

## ¿Qué es la Propiedad IDENTITY ?

IDENTITY es una propiedad que se puede aplicar a una columna de una Tabla para que el motor de base de datos genere automáticamente valores numéricos únicos para esa columna.

valor de inicialización: 101

Incremento: 3

### Valor Predeterminado:

valor de inicialización: 1

Incremento: 1

## ¿Qué es la Propiedad IDENTITY ?

IDENTITY es una propiedad que se puede aplicar a una columna de una Tabla para que el motor de base de datos genere automáticamente valores numéricos únicos para esa columna.

valor de inicialización: **101**

Incremento: **3**



<u>idProfesor</u>	nombre	paterno	materno
<b>101</b>	Isabel	Campos	Torres
<b>104</b>	Oscar	Gallegos	Flores
<b>107</b>	Isabel	Guillen	Morales

Valor Predeterminado:

valor de inicialización: **1**

Incremento: **1**

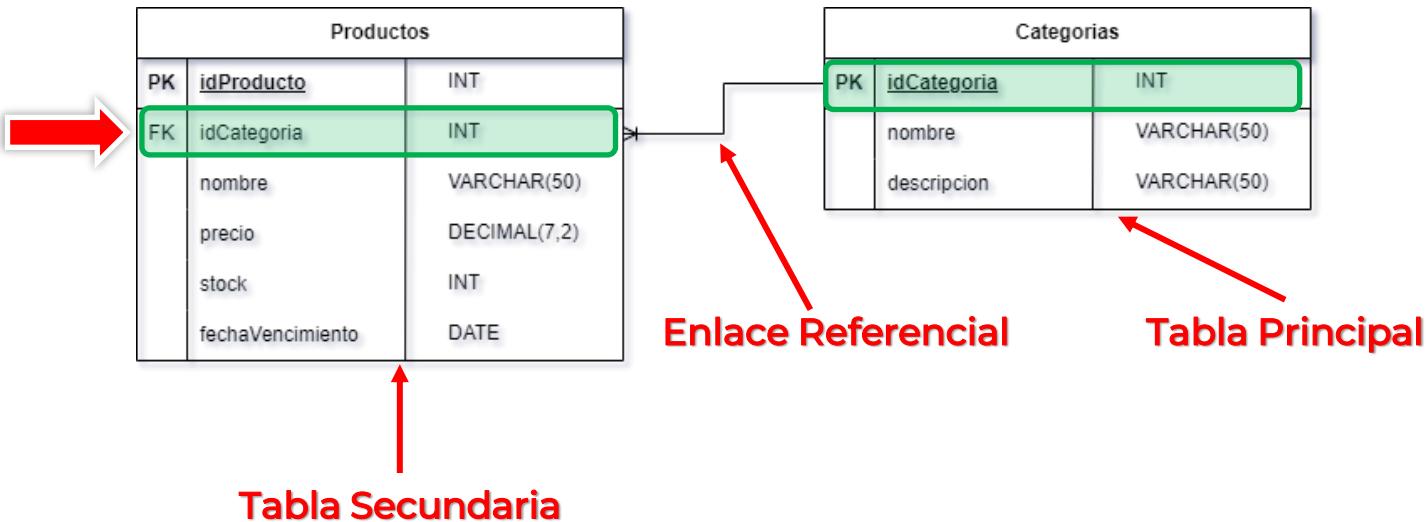


<u>idProfesor</u>	nombre	paterno	materno
<b>101</b>	Isabel	Campos	Torres
<b>102</b>	Oscar	Gallegos	Flores
<b>103</b>	Isabel	Guillen	Morales

## Introducción a Foreign Key

# Introducción a Foreign Key

Una Foreign Key o Llaves Foráneas es una columna o conjunto de columnas que nos permiten establecer un vínculo referencial entre los datos de dos tablas.



# Introducción a Foreign Key

Una Foreign Key o Llaves Foráneas es una columna o conjunto de columnas que nos permiten establecer un vínculo referencial entre los datos de dos tablas.

Enlace autorreferencia →

← Tabla autorreferencia

Empleados		
PK	<u>idEmpleado</u>	INT
	nombre	VARCHAR(50)
	paterno	VARCHAR(50)
	materno	VARCHAR(50)
	cargo	VARCHAR(50)
	reportaA	INT

# Establecer Foreign Key

Profesores		
PK	<u><a href="#">idProfesor</a></u>	INT
	nombre	VARCHAR(50)
	paterno	VARCHAR(50)
	materno	VARCHAR(50)
	fechaNacimiento	DATE
	domicilio	VARCHAR(50)
	telefono	INT
	email	VARCHAR(50)

Asignaturas		
PK	<u><a href="#">idAsignatura</a></u>	INT
	FK	<u><a href="#">idProfesor</a></u>
	nombre	VARCHAR(50)
	horainicio	TIME
	horaFin	TIME

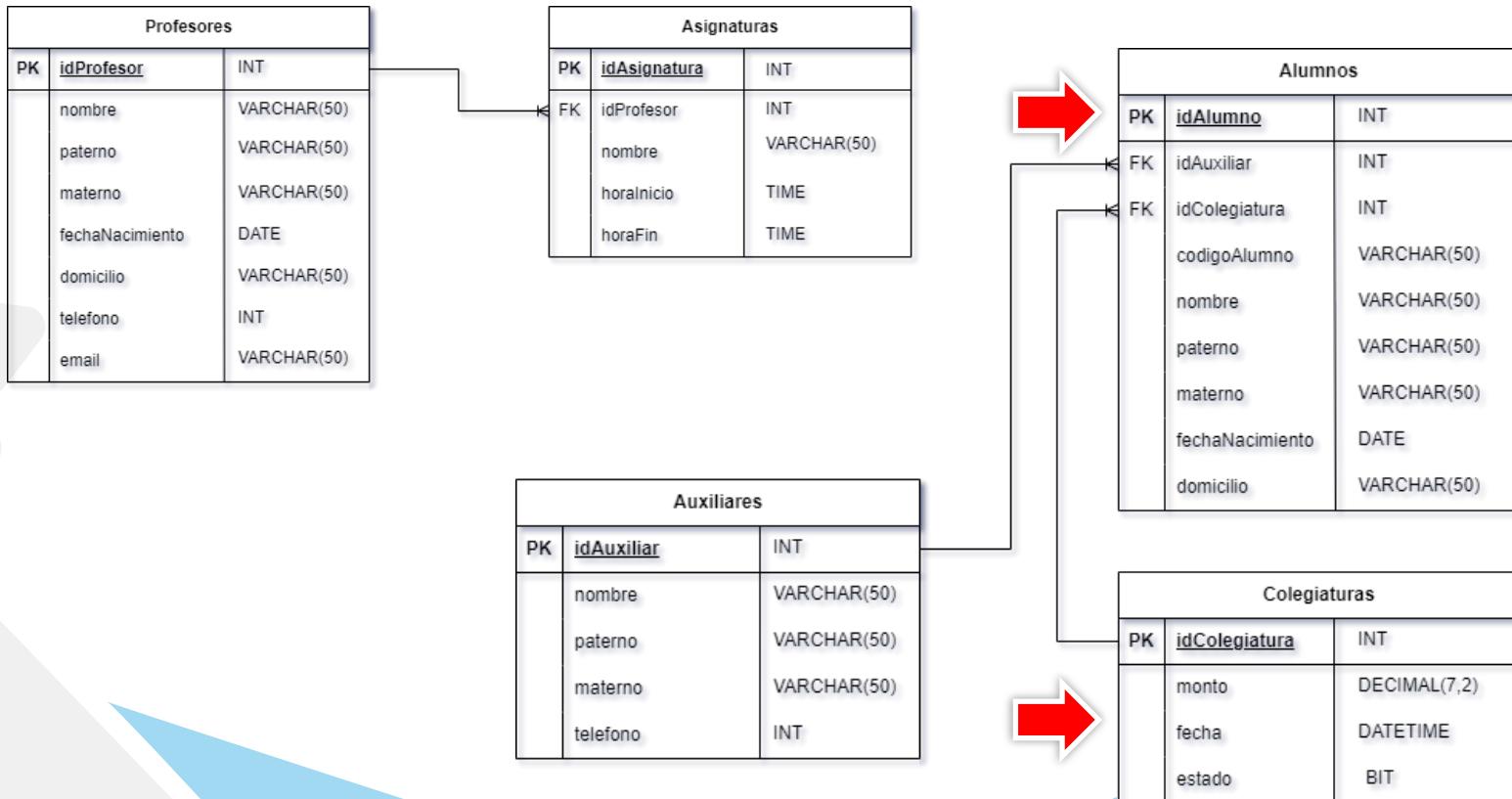
Auxiliares		
PK	<u><a href="#">idAuxiliar</a></u>	INT
	nombre	VARCHAR(50)
	paterno	VARCHAR(50)
	materno	VARCHAR(50)
	telefono	INT

## Diagrama de Base de Datos

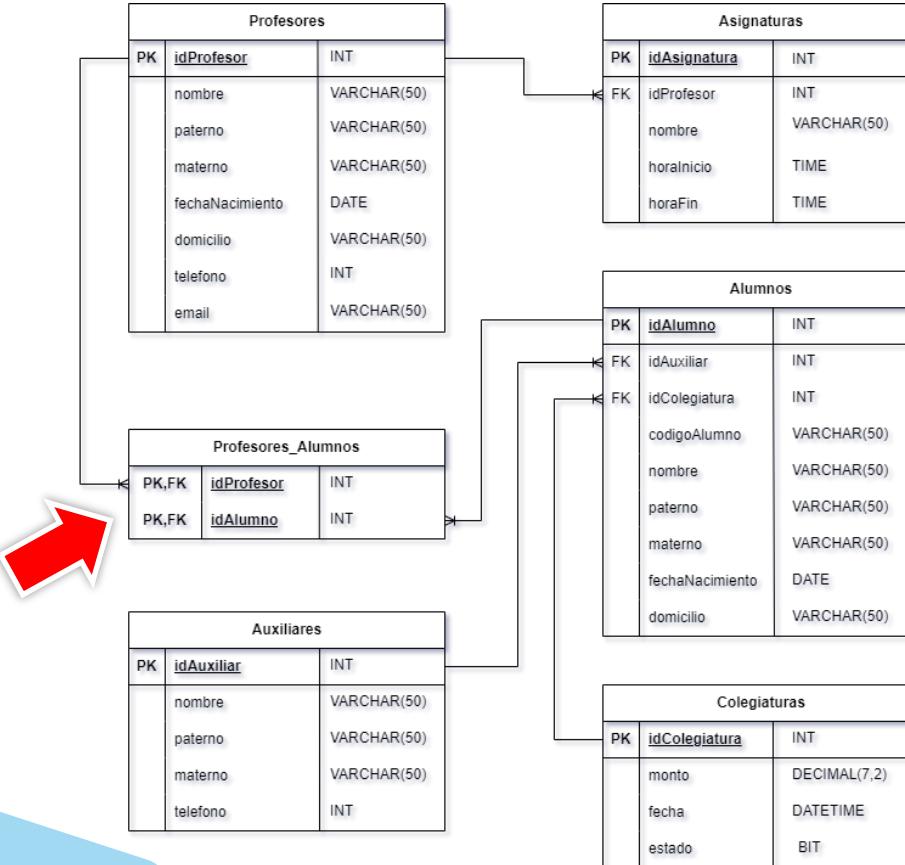
El Diseñador de Diagramas de Base de Datos es una herramienta visual que le permite Diseñar y Visualizar una Base de Datos a la que está conectado.

Cuando diseña una base de datos, puede utilizar el Diseñador de Bases de Datos para crear, editar o eliminar tablas, columnas, claves, índices, relaciones y restricciones.

# Diagrama de Base de Datos



# Tabla con Primary Key Compuesta



## Descripción de la Sección

Instrucción INSERT

Insertar Registro - Entorno Gráfico

Insertar Registro - Transact SQL

Deshabilitar la propiedad IDENTITY

Insertar Múltiples Registros - Transact SQL

Insertar Registro a una Tabla con Foreign Key

Insertar Registro a una Tabla con Primary Key Compuesta



# Instrucción INSERT

La instrucción INSERT INTO se usa principalmente para agregar una o más filas a la Tabla

### Sintaxis

**INSERT INTO table\_name (col<sub>1</sub>, col<sub>2</sub>, col<sub>3</sub>, ..., col<sub>n</sub>)**



*Auto-Generadas  
Calculadas*

*NOT NULL*

*NULL*



*PRIMARY KEY*

La instrucción INSERT INTO se usa principalmente para agregar una o más filas a la Tabla

### Sintaxis

**INSERT INTO table\_name (col<sub>1</sub>, col<sub>2</sub>, col<sub>3</sub>, ..., col<sub>n</sub>)**

**VALUES (val<sub>1</sub>, val<sub>2</sub>, val<sub>3</sub>, ..., val<sub>n</sub>);**

La instrucción INSERT INTO se usa principalmente para agregar una o más filas a la Tabla

### Sintaxis

**INSERT INTO table\_name (col<sub>1</sub>, col<sub>2</sub>, col<sub>3</sub>, ..., col<sub>n</sub>)**

**VALUES (val<sub>1</sub>, val<sub>2</sub>, val<sub>3</sub>, ..., val<sub>n</sub>);**

La instrucción INSERT INTO se usa principalmente para agregar una o más filas a la Tabla

### Sintaxis

```
INSERT INTO table_name (col1, col2, col3, ..., coln)  
VALUES (val1, val2, val3, ..., valn);
```

La instrucción INSERT INTO se usa principalmente para agregar una o más filas a la Tabla

### Sintaxis

**INSERT INTO table\_name (col<sub>1</sub>, col<sub>2</sub>, col<sub>3</sub>, ..., col<sub>n</sub>)**

**VALUES (val<sub>1</sub>, val<sub>2</sub>, val<sub>3</sub>, ..., val<sub>n</sub>),**

**(val<sub>1</sub>, val<sub>2</sub>, val<sub>3</sub>, ..., val<sub>n</sub>),**

**(val<sub>1</sub>, val<sub>2</sub>, val<sub>3</sub>, ..., val<sub>n</sub>);**

Una sola vez

La instrucción INSERT INTO se usa principalmente para agregar una o más filas a la Tabla

### Sintaxis

**INSERT INTO table\_name (col<sub>1</sub>, col<sub>2</sub>, col<sub>3</sub>, ..., col<sub>n</sub>)**

**SELECT val<sub>1</sub>, val<sub>2</sub>, val<sub>3</sub>, ..., val<sub>n</sub>;**

La instrucción INSERT INTO se usa principalmente para agregar una o más filas a la Tabla

### Sintaxis

**INSERT INTO table\_name (col<sub>1</sub>, col<sub>2</sub>, col<sub>3</sub>, ..., col<sub>n</sub>)**

**SELECT val<sub>1</sub>, val<sub>2</sub>, val<sub>3</sub>, ..., val<sub>n</sub>**

**FROM tabla;**

*Las columnas consultadas deben coincidir con las columnas a insertar*

La instrucción INSERT INTO se usa principalmente para agregar una o más filas a la Tabla

### Sintaxis

**INSERT INTO table\_name**

Todas las columnas de la Tabla

**SELECT val<sub>1</sub>, val<sub>2</sub>, val<sub>3</sub>, ..., val<sub>n</sub>**  
**FROM tabla;**

*Las columnas consultadas deben coincidir con las columnas a insertar*

La instrucción INSERT INTO se usa principalmente para agregar una o más filas a la Tabla

*Primary Key*

*Foreign Key*

*Columnas Auto-Incrementales*

## Insertar Registro - Transact SQL

# Insertar Registro - Transact SQL

## *Numéricos Exactos*

INT → 13

DECIMAL → 13 o 13,5

BIT → 1 o 0

## *Cadenas de caracteres*

CHAR(n) → 'texto'

VARCHAR(n) → 'texto'

## *Fecha y hora*

TIME → '13:10:20.123'

DATE → '01-01-2024' o '2024-01-01'

'01/01/2024' o '2024/01/01'

DATETIME → '01-01-2024 13:10:20.123'

## *Valor NULL*

**NULL**

' ' o 0

## Insertar Registro a una Tabla con Foreign Key

## Insertar Registro a una Tabla con Foreign Key

Podemos insertar registros donde su valor del campo Foreign Key se encuentren relacionado con el valor del campo Primary Key.

Productos		
PK	<u>idProducto</u>	INT
FK	<u>idCategoria</u>	INT
	nombre	VARCHAR(50)
	precio	DECIMAL(7,2)
	stock	INT
	fechaVencimiento	DATE

Categorias		
PK	<u>idCategoria</u>	INT
	nombre	VARCHAR(50)
	descripcion	VARCHAR(50)

### Productos

idProducto	idCategoria	nombre	precio	stock	fechaVencimiento
1	1	Gaseosa	5.90	100	01/01/2030
2	7	Laptop	1500	50	

### Categorias

idCategoria	nombre	descripcion
1	Bebidas	
2	Ropa	
3	Tecnología	

## Insertar Registro a una Tabla con Foreign Key

Podemos insertar registros donde su valor del campo Foreign Key se encuentren relacionado con el valor del campo Primary Key.

Productos		
PK	<u>idProducto</u>	INT
FK	<u>idCategoria</u>	INT
	nombre	VARCHAR(50)
	precio	DECIMAL(7,2)
	stock	INT
	fechaVencimiento	DATE

Categorias		
PK	<u>idCategoria</u>	INT
	nombre	VARCHAR(50)
	descripcion	VARCHAR(50)

### Productos

idProducto	idCategoria	nombre	precio	stock	fechaVencimiento
1	1	Gaseosa	5.90	100	01/01/2030
2	3	Laptop	1500	50	

### Categorias

idCategoria	nombre	descripcion
1	Bebidas	
2	Ropa	
3	Tecnología	

## Tarea – Sentencias DML(Insert)

Clientes		
PK	<u>idCliente</u>	INT
	nombre	VARCHAR(50)
	paterno	VARCHAR(50)
	materno	VARCHAR(50)
	edad	INT
	fechaNacimiento	DATE

1.

Insertar un registro en la Tabla Clientes.

2.

Insertar Múltiples registros en la Tabla Clientes(Más de dos)

**Recomendación: La columna "idCliente" es un campo "Identity".**

## Descripción de la Sección

Instrucción SELECT

Instrucción SELECT - Entorno Gráfico

Instrucción SELECT - Transact SQL

Cláusula DISTINCT

Cláusula WHERE

Cláusula ORDER BY

Cláusula TOP

Cláusula PERCENT

Cláusula WITH TIES



# Instrucción SELECT

La instrucción **SELECT** es la mas utilizada para consultar registros de las Tablas

### Sintaxis

Consulta todas las columnas

**SELECT \* FROM nombre\_tabla**

La instrucción **SELECT** es la mas utilizada para consultar registros de las Tablas

### Sintaxis

Consulta columnas específicas

**SELECT col<sub>1</sub>, col<sub>2</sub>, ..., col<sub>n</sub> FROM nombre\_tabla**

La Cláusula DISTINCT se utiliza para devolver sólo valores distintos (diferentes).

### Sintaxis

**SELECT DISTINCT columna**

**FROM nombre\_Tabla**

La cláusula **WHERE** se utiliza para Filtrar las filas devueltas por una consulta

### Sintaxis

**SELECT** lista\_columna

**FROM** nombre\_Tabla

**WHERE** condición;

## Cláusula ORDER BY

La cláusula **ORDER BY** se utiliza para ordenar el conjunto de resultados en orden ascendente o descendente.

### Sintaxis

**SELECT** lista\_columna

**FROM** nombre\_Tabla

**ORDER BY** columna [**ASC** | **DESC**];

SQL Server lo utiliza **ASC** como orden de clasificación predeterminado

## Cláusula ORDER BY

La cláusula **ORDER BY** se utiliza para ordenar el conjunto de resultados en orden ascendente o descendente.

### Sintaxis

**SELECT** lista\_columna

**FROM** nombre\_Tabla

**WHERE** condición;

**ORDER BY** columna [**ASC | DESC**];

La **TOP** cláusula le permite limitar el número de filas devueltas en un conjunto de resultados de consulta.

### Sintaxis

**SELECT TOP n lista\_columna**

**FROM nombre\_Tabla**

La Cláusula PERCENT es opcional y devuelve las filas superiores basadas en un porcentaje del conjunto de resultados total.

### Sintaxis

```
SELECT TOP n [PERCENT | WITH TIES] lista_columna  
FROM   nombre_Tabla
```

La Cláusula **WITH TIES** permite incluir las filas en el conjunto de resultados que coinciden con la última fila.

### Sintaxis

```
SELECT TOP n [PERCENT | WITH TIES] lista_columna  
FROM nombre_Tabla
```

# Tarea – Sentencias DML(Select)

**Tabla: Peliculas**

<b>idPelicula</b>	<b>titulo</b>	<b>presupuesto</b>	<b>fechaLanzamiento</b>	<b>ganancia</b>	<b>tiempoDuracion</b>	<b>calificacion</b>
101	Avatar	237000000	10/12/2009	2787965087	162	7.2
102	Titanic	200000000	18/11/1997	1845034188	194	7.5
103	The Avengers	220000000	25/04/2012	1519557910	143	7.4
104	Jurassic World	150000000	9/06/2015	1513528810	124	6.5
105	Furious 7	190000000	1/04/2015	1506249360	137	7.3
106	Avengers: Age of Ultron	280000000	22/04/2015	1405403694	141	7.3
107	Frozen	150000000	27/11/2013	1274219009	102	7.3
108	Iron Man 3	200000000	18/04/2013	1215439994	130	6.8
109	Minions	74000000	17/06/2015	1156730962	91	6.4
110	Captain America: Civil War	250000000	27/04/2016	1153304495	147	7.1
111	Transformers: Dark of the Moon	195000000	28/06/2011	1123746996	154	6.1
112	The Lord of the Rings: The Return of the King	94000000	1/12/2003	1118888979	201	8.1
113	Skyfall	200000000	25/10/2012	1108561013	143	6.9
114	Transformers: Age of Extinction	210000000	25/06/2014	1091405097	165	5.8
115	The Dark Knight Rises	250000000	16/07/2012	1084939099	165	7.6
116	Toy Story 3	200000000	16/06/2010	1066969703	103	7.6
117	Pirates of the Caribbean: Dead Man's Chest	200000000	20/06/2006	1065659812	151	7
118	Pirates of the Caribbean: On Stranger Tides	380000000	14/05/2011	1045713802	136	6.4
119	Alice in Wonderland	200000000	3/03/2010	1025491110	108	6.4
120	The Hobbit: An Unexpected Journey	250000000	26/11/2012	1021103568	169	7

## Tarea – Sentencias DML(Select)

1.

Mostrar todos los registros de la Tabla Películas.

2.

Mostrar los distintos tiempos de duración de las Películas.

3.

Mostrar las Películas que tengan un tiempos de duración de 165 minutos.

4.

Mostrar las Películas ordenados de manera Ascendente por la Fecha de Lanzamiento.

5.

Mostrar las 5 mejores Películas ordenados de manera Descendente por la Calificación.

6.

Mostrar el 50% de las Películas ordenados de manera Descendente por la Ganancia.

## Descripción de la Sección

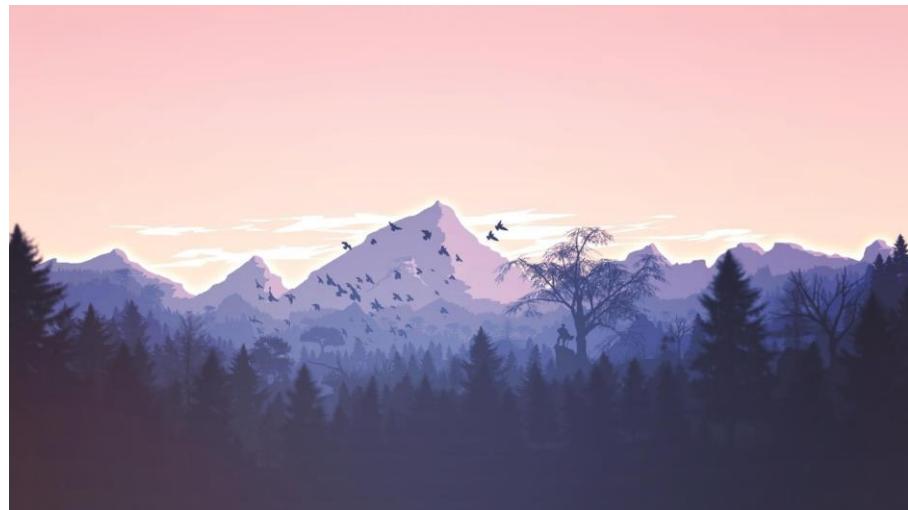
Instrucción UPDATE

Instrucción UPDATE - Entorno Gráfico

Instrucción UPDATE sin Condicional  
Transact SQL

Instrucción UPDATE con Condicional  
Transact SQL

Instrucción UPDATE con Foreign Key



# Instrucción UPDATE

La instrucción UPDATE se utiliza para actualizar registros existentes en una tabla en una base de datos de SQL Server.

### Sintaxis

**UPDATE nombre\_tabla**

**SET      column1 = expresion1,**

**column2 = expresion2**

...



**Podemos especificar mas columnas**

**[WHERE condición]**

## Tarea – Sentencias DML(Update)

Tabla: Peliculas

<b>idPelicula</b>	<b>titulo</b>	<b>presupuesto</b>	<b>fechaLanzamiento</b>	<b>ganancia</b>	<b>tiempoDuracion</b>	<b>calificacion</b>
101	Avatar	237000000	10/12/2009	2787965087	162	7.2
102	Titanic	200000000	18/11/1997	1845034188	194	7.5
103	The Avengers	220000000	25/04/2012	1519557910	143	7.4
104	Jurassic World	150000000	9/06/2015	1513528810	124	6.5
105	Furious 7	190000000	1/04/2015	1506249360	137	7.3
106	Avengers: Age of Ultron	280000000	22/04/2015	1405403694	141	7.3
107	Frozen	150000000	27/11/2013	1274219009	102	7.3
108	Iron Man 3	200000000	18/04/2013	1215439994	130	6.8
109	Minions	74000000	17/06/2015	1156730962	91	6.4
110	Captain America: Civil War	250000000	27/04/2016	1153304495	147	7.1
111	Transformers: Dark of the Moon	195000000	28/06/2011	1123746996	154	6.1
112	The Lord of the Rings: The Return of the King	94000000	1/12/2003	1118888979	201	8.1
113	Skyfall	200000000	25/10/2012	1108561013	143	6.9
114	Transformers: Age of Extinction	210000000	25/06/2014	1091405097	165	5.8
115	The Dark Knight Rises	250000000	16/07/2012	1084939099	165	7.6
116	Toy Story 3	200000000	16/06/2010	1066969703	103	7.6
117	Pirates of the Caribbean: Dead Man's Chest	200000000	20/06/2006	1065659812	151	7
118	Pirates of the Caribbean: On Stranger Tides	380000000	14/05/2011	1045713802	136	6.4
119	Alice in Wonderland	200000000	3/03/2010	1025491110	108	6.4
120	The Hobbit: An Unexpected Journey	250000000	26/11/2012	1021103568	169	7

## Tarea – Sentencias DML(Update)

1.

Actualizar la calificación a 8.5 donde el valor del campo “idPelicula” es 110.

2.

Actualizar el tiempo de duración a 140 de la película “Avatar”.

3.

Actualizar la fecha de lanzamiento al “17/06/2010” de la película “Toy Story 3”.

4.

Actualizar el nombre de todas las películas a “Desconocido”.

# Descripción de la Sección

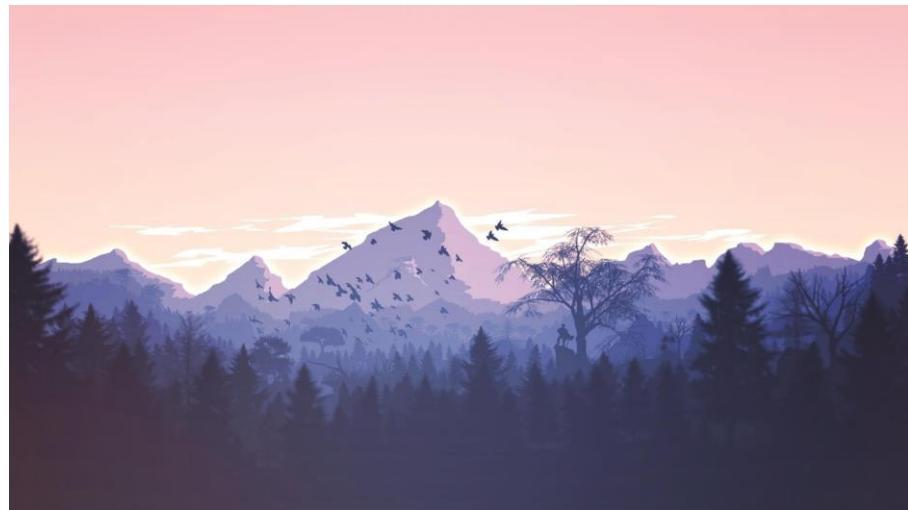
Instrucción DELETE

Instrucción DELETE - Entorno Gráfico

Instrucción DELETE sin Condicional  
Transact SQL

Instrucción DELETE con Condicional  
Transact SQL

Instrucción DELETE con Foreign Key



# Instrucción DELETE

La instrucción **DELETE** se utiliza para eliminar registros existentes en una tabla en una base de datos de SQL Server

### Sintaxis

**DELETE FROM nombre\_tabla**

**[WHERE condición]**

# Tarea – Sentencias DML(Delete)

**Tabla: Peliculas**

<b>idPelicula</b>	<b>titulo</b>	<b>presupuesto</b>	<b>fechaLanzamiento</b>	<b>ganancia</b>	<b>tiempoDuracion</b>	<b>calificacion</b>
101	Avatar	237000000	10/12/2009	2787965087	162	7.2
102	Titanic	200000000	18/11/1997	1845034188	194	7.5
103	The Avengers	220000000	25/04/2012	1519557910	143	7.4
104	Jurassic World	150000000	9/06/2015	1513528810	124	6.5
105	Furious 7	190000000	1/04/2015	1506249360	137	7.3
106	Avengers: Age of Ultron	280000000	22/04/2015	1405403694	141	7.3
107	Frozen	150000000	27/11/2013	1274219009	102	7.3
108	Iron Man 3	200000000	18/04/2013	1215439994	130	6.8
109	Minions	74000000	17/06/2015	1156730962	91	6.4
110	Captain America: Civil War	250000000	27/04/2016	1153304495	147	7.1
111	Transformers: Dark of the Moon	195000000	28/06/2011	1123746996	154	6.1
112	The Lord of the Rings: The Return of the King	94000000	1/12/2003	1118888979	201	8.1
113	Skyfall	200000000	25/10/2012	1108561013	143	6.9
114	Transformers: Age of Extinction	210000000	25/06/2014	1091405097	165	5.8
115	The Dark Knight Rises	250000000	16/07/2012	1084939099	165	7.6
116	Toy Story 3	200000000	16/06/2010	1066969703	103	7.6
117	Pirates of the Caribbean: Dead Man's Chest	200000000	20/06/2006	1065659812	151	7
118	Pirates of the Caribbean: On Stranger Tides	380000000	14/05/2011	1045713802	136	6.4
119	Alice in Wonderland	200000000	3/03/2010	1025491110	108	6.4
120	The Hobbit: An Unexpected Journey	250000000	26/11/2012	1021103568	169	7

## Tarea – Sentencias DML(Delete)

1. Eliminar la película donde el valor del campo “idPelicula” es igual a 114.
2. Eliminar las películas que tengan una calificación de 6.5.
3. Eliminar la película de “Frozen”.
4. Eliminar la película donde su fecha de lanzamiento es “26/11/2012”.
5. Eliminar todas las películas.

## Descripción de la Sección

Descripción - Operadores de Comparación

Operador de Comparación - “Es igual a”

Operador de Comparación - “Mayor que”

Operador de Comparación - “Mayor o igual que”

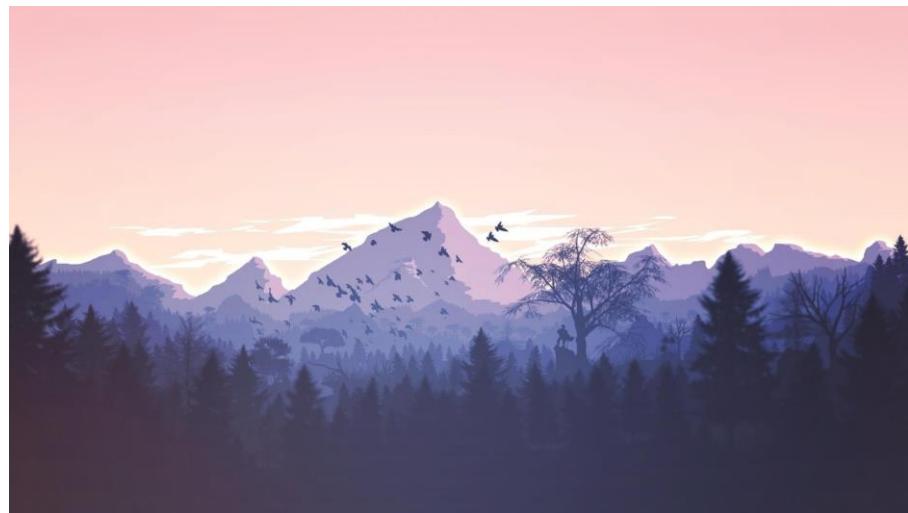
Operador de Comparación - “Menor que”

Operador de Comparación - “Menor o igual que”

Operador de Comparación - “No es igual a”

Operador de Comparación - “No es menor que”

Operador de Comparación - “No es mayor que”



## Operadores de Comparación

# Operadores de Comparación

Los operadores de comparación son utilizados para comparar valores en una consulta y devolver un resultado booleano (verdadero o falso) que indica si la comparación es cierta.

```
SELECT lista_columna  
FROM nombre_Tabla  
WHERE condición;
```

```
DELETE FROM nombre_tabla  
[WHERE condición]
```

```
UPDATE nombre_tabla  
SET column1 = expresion1,  
    column2 = expresion2  
    ...  
[WHERE condición]
```

Los operadores de comparación son utilizados para comparar valores en una consulta y devolver un resultado booleano (verdadero o falso) que indica si la comparación es cierta.

## Operadores:

=	Es igual a
---	------------

>	Mayor que
---	-----------

>=	Mayor o igual que
----	-------------------

<	Menor que
---	-----------

<=	Menor o igual que
----	-------------------

<>	No es igual a
----	---------------

!=	No es igual a(No es del estándar ISO)
----	---------------------------------------

!<	No es menor que(No es del estándar ISO)
----	---

!>	No es mayor que(No es del estándar ISO)
----	---

### Utilizar la Base de Datos “pubs”

1.

Mostrar los Autores que sean del estado de “UT”.

2.

Mostrar los títulos donde el precio sea mayor o igual a 20.

3.

Mostrar los títulos donde el precio sea menor o igual a 7.

## Descripción de la Sección

Descripción - Operadores Lógicos

Operador Lógico - "AND"

Operador Lógico - "OR"

Operador Lógico - "IN"

Operador Lógico - "LIKE"

Operador Lógico - "BETWEEN"

Operador Lógico - "NOT"



# Operadores Lógicos

# Operadores Lógicos

Los operadores lógicos se utilizan para combinar múltiples condiciones en una consulta y evaluar si se cumplen todas o algunas de ellas. Estos operadores lógicos devuelven el tipo de datos booleano (**verdadero o falso**) que indica si la comparación es cierta.

```
SELECT lista_columna  
FROM nombre_Tabla  
WHERE condición;
```

```
DELETE FROM nombre_tabla  
[WHERE condición]
```

```
UPDATE nombre_tabla  
SET column1 = expresion1,  
    column2 = expresion2  
    ...  
[WHERE condición]
```

Los operadores lógicos se utilizan para combinar múltiples condiciones en una consulta y evaluar si se cumplen todas o algunas de ellas. Estos operadores lógicos devuelven el tipo de datos booleano (**verdadero o falso**) que indica si la comparación es cierta.

## Operadores:

AND

Operador Lógico AND

BETWEEN

Operador Lógico BETWEEN

OR

Operador Lógico OR

IN

Operador Lógico IN

LIKE

Operador Lógico LIKE

EXISTS

Operador Lógico EXISTS

NOT

Operador Lógico NOT

*Módulo Subconsultas*

# Operadores Lógicos

Los operadores lógicos se utilizan para combinar múltiples condiciones en una consulta y evaluar si se cumplen todas o algunas de ellas. Estos operadores lógicos devuelven el tipo de datos booleano (**verdadero o falso**) que indica si la comparación es cierta.

AND

E1	E2	E1 Y E2
V	V	<b>V</b>
V	F	<b>F</b>
F	V	<b>F</b>
F	F	<b>F</b>

OR

E1	E2	E1 O E2
V	V	<b>V</b>
V	F	<b>V</b>
F	V	<b>V</b>
F	F	<b>F</b>

NOT

E	NOT E
V	<b>F</b>
F	<b>V</b>

Los operadores lógicos se utilizan para combinar múltiples condiciones en una consulta y evaluar si se cumplen todas o algunas de ellas. Estos operadores lógicos devuelven el tipo de datos booleano (**verdadero o falso**) que indica si la comparación es cierta.

## AND

E1	E2	E1 Y E2
V	V	<b>V</b>
V	F	<b>F</b>
F	V	<b>F</b>
F	F	<b>F</b>

## OR

E1	E2	E1 O E2
V	V	<b>V</b>
V	F	<b>V</b>
F	V	<b>V</b>
F	F	<b>F</b>

## NOT

E	NOT E
V	<b>F</b>
F	<b>V</b>

Los operadores lógicos se utilizan para combinar múltiples condiciones en una consulta y evaluar si se cumplen todas o algunas de ellas. Estos operadores lógicos devuelven el tipo de datos booleano (**verdadero o falso**) que indica si la comparación es cierta.

## AND

E1	E2	E1 Y E2
V	V	<b>V</b>
V	F	<b>F</b>
F	V	<b>F</b>
F	F	<b>F</b>

## OR

E1	E2	E1 O E2
V	V	<b>V</b>
V	F	<b>V</b>
F	V	<b>V</b>
F	F	<b>F</b>

## NOT

E	NOT E
V	<b>F</b>
F	<b>V</b>

### Utilizar la Base de Datos “pubs”

1.

Mostrar los Autores que sean del estado de “CA” y la ciudad de “Palo Alto”.

2.

Mostrar los títulos donde el precio sea mayor o igual a 20 o menor o igual a 7.

3.

Mostrar todos los empleados donde su nombre empiece por la letra M.

4.

Mostrar los Autores que sean del estado de “MD, IN, OR”.

# Descripción de la Sección

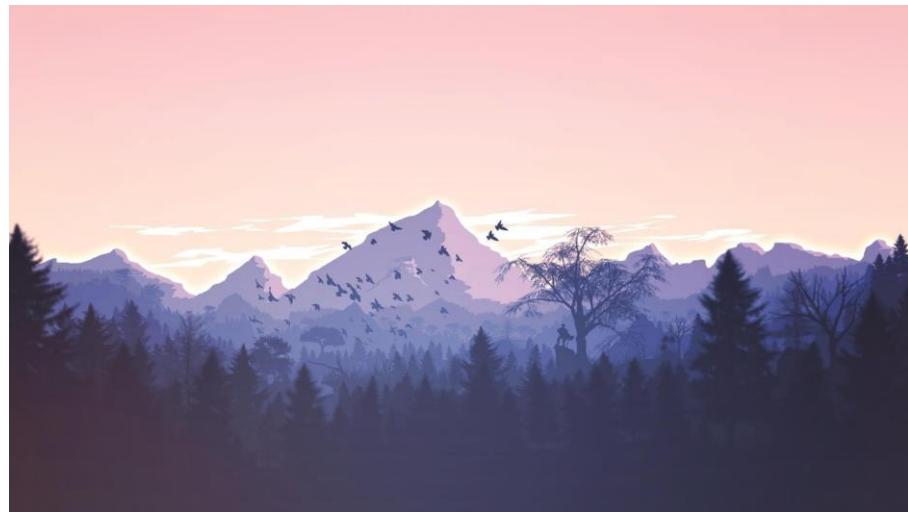
Descripción - Operadores de Conjunto

Operador - “UNION”

Operador - “UNION ALL”

Operador - “INTERSECT”

Operador - “EXCEPT”



# Operadores de Conjunto

## Operadores de Conjunto

Los operadores de Conjuntos se utilizan para combinar dos o más datos similares presentes en uno, dos o más bases de datos SQL; fusionan el resultado obtenido.

```
SELECT lista_columna  
FROM nombre_Tabla  
WHERE condición;
```

# Operadores de Conjunto

Los operadores de Conjuntos se utilizan para combinar dos o más datos similares presentes en uno, dos o más bases de datos SQL; fusionan el resultado obtenido.

```
SELECT lista_columna  
FROM   nombre_Tabla  
WHERE  condición;
```

## Requisitos:

- Cada instrucción **SELECT** dentro del operador **UNION** debe tener el mismo número de columnas.
- Las columnas también deben tener **tipos de datos** similares.
- Las columnas de la instrucción **SELECT** deben estar en el **mismo orden**.

# Operadores de Conjunto

Los operadores de Conjuntos se utilizan para combinar dos o más datos similares presentes en uno, dos o más bases de datos SQL; fusionan el resultado obtenido.

## Operadores:

UNION

Operador de Conjunto UNION

UNION ALL

Operador de Conjunto UNION ALL

INTERSECT

Operador de Conjunto INTERSECT

EXCEPT

Operador de Conjunto EXCEPT

### Utilizar la Base de Datos “pubs”

1.

Unir los nombres de la tabla “authors” y “employee” eliminando los valores duplicados si existen.

2.

Unir los estados de la tabla “authors” y “publishers” conservando los valores duplicados si existen.

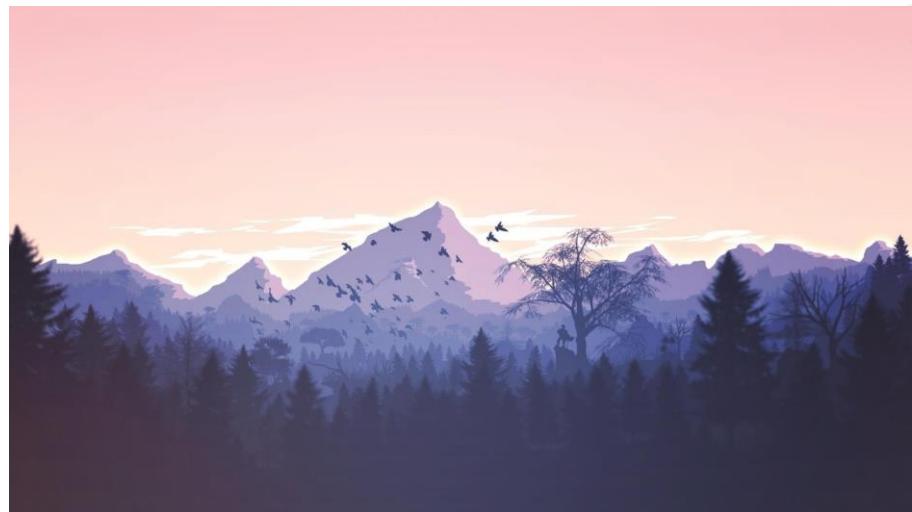
## Descripción de la Sección

Descripción - Operadores Aritméticos

Operador - "Suma" y "Resta"

Operador - "Multiplicar" y "Dividir"

Operador - "Módulo"



# Operadores Aritméticos

# Operadores Aritméticos

Los operadores aritméticos se utilizan para realizar cálculos y operaciones aritméticas en los operandos o los datos presentes en las tablas de la base de datos.

## Operadores:

+

Operador Aritmético SUMA

-

Operador Aritmético RESTA

\*

Operador Aritmético MULTIPLICAR

/

Operador Aritmético DIVIDIR

%

Operador Aritmético MÓDULO

# Descripción de la Sección

Descripción - Sentencias DDL

Crear de una Base de Datos

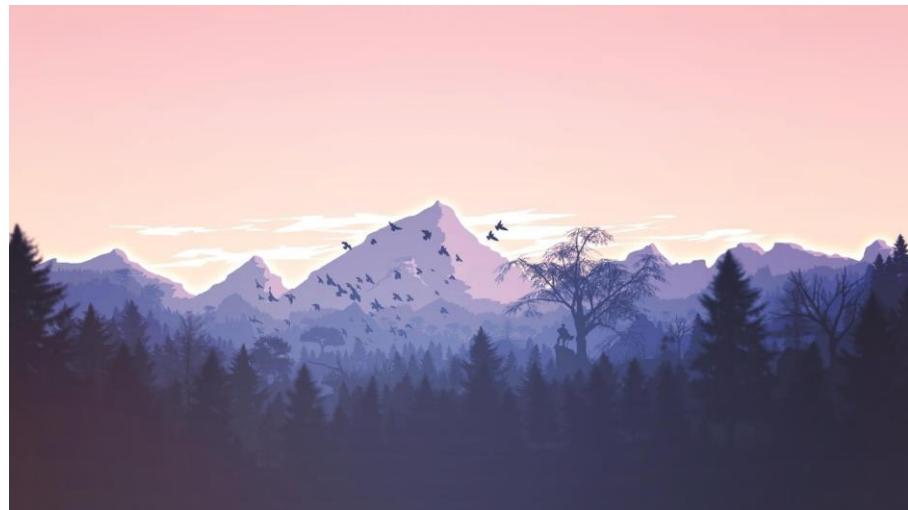
Esquemas(schema)

Utilizar las Sentencias DDL

Descripción - Foreign Key

Validar la Acción Foreign Key

Sentencia TRUNCATE



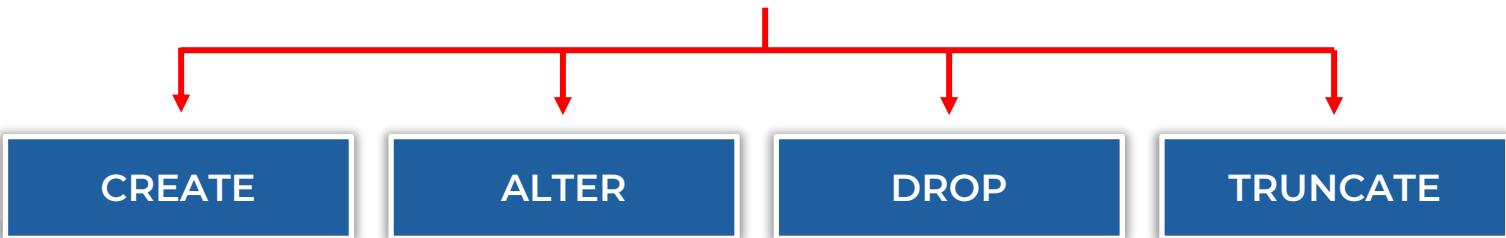
# Sentencias DDL

# Sentencias DDL

Las sentencias DDL (del inglés: Data Definition Language o en español: Lenguaje de Definición de Datos) son un conjunto de comandos utilizados para definir, modificar y gestionar la estructura de una base de datos.

Estas sentencias se utilizan para crear, modificar y eliminar objetos en la base de datos, como tablas, índices, vistas, restricciones y otros elementos que definen la organización y la estructura de los datos en un sistema de administración de Base de Datos Relacionales(RDBMS).

## Las sentencias DDL son:



## CREATE

La sentencia **CREATE** se utiliza para **crear nuevos objetos** en la base de datos. Estos objetos pueden ser tablas, vistas, índices, procedimientos almacenados, funciones y otros elementos que forman parte de la estructura de la base de datos.

### Sintaxis:

**CREATE [tipo\_de\_objeto] [nombre\_de\_objeto];**

Table, Database, View, Index, etc

Nombre que deseas asignar

## ALTER

La sentencia **ALTER** se utiliza para **modificar** la estructura de objetos ya existentes en la base de datos, como agregar, eliminar o modificar columnas de una tabla, cambiar el nombre de un objeto, etc.

### Sintaxis:

**ALTER [tipo\_objeto] [nombre\_objeto] [accion];**

Table, Column, Index, etc

Add, Drop, Alter, etc

Nombre del objeto a modificar

## DROP

La sentencia **DROP** se utiliza para eliminar objetos de la base de datos, como tablas, vistas o índices.

Ten en cuenta que esta acción es **irreversible** y **elimina** permanentemente los datos asociados.

### Sintaxis:

**DROP [tipo\_objeto] IF EXISTS [nombre\_objeto];**

Table, Index, Database, etc

Nombre del objeto a eliminar

Es opcional, se utiliza para evitar que se genere un error si el objeto no existe

## TRUNCATE

La sentencia **TRUNCATE** elimina todas las filas de una tabla o particiones específicas de una tabla, sin registrar las eliminaciones de filas individuales.

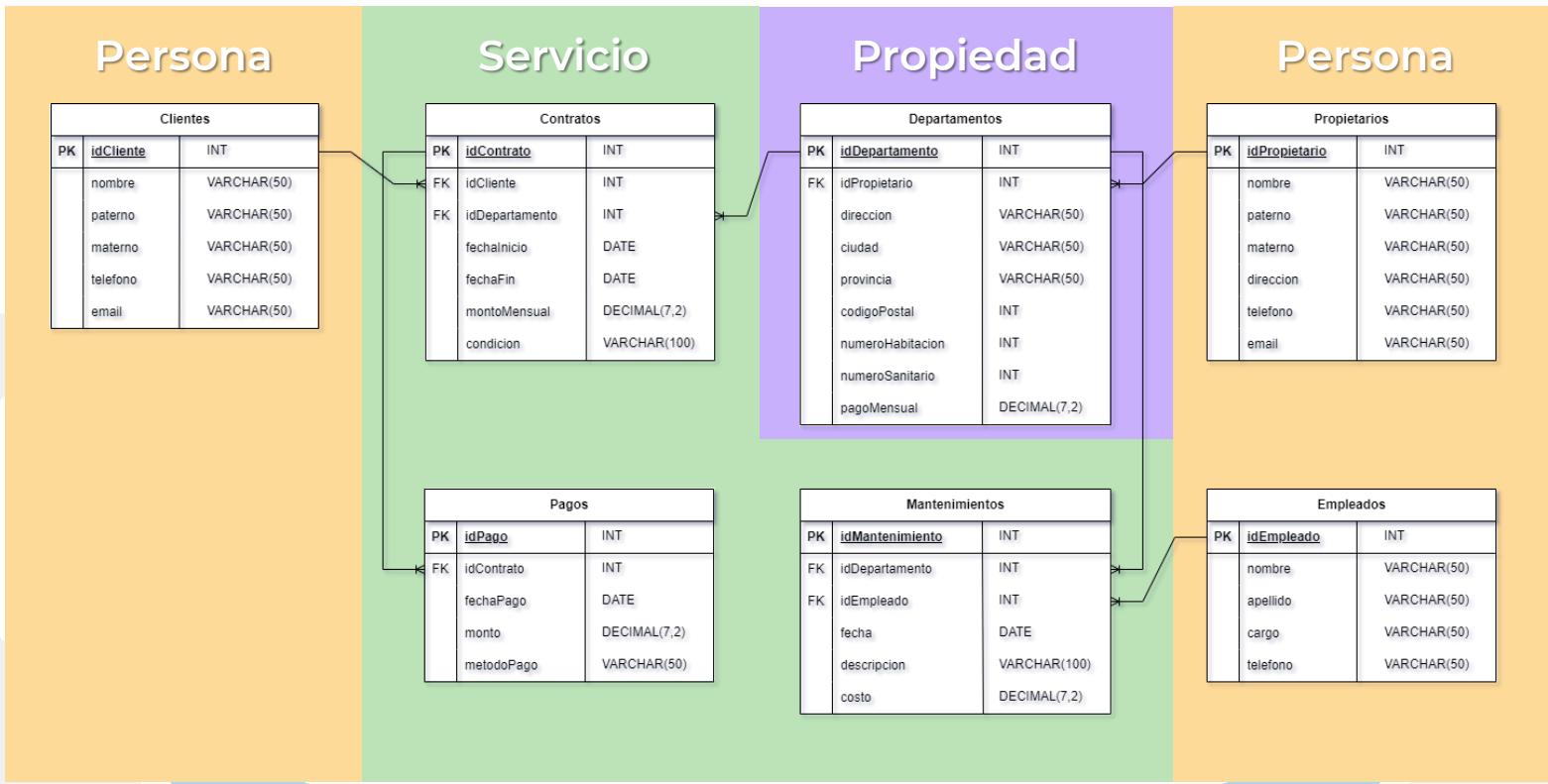
**TRUNCATE** es similar a la declaración **DELETE** sin la cláusula **WHERE**; sin embargo, **TRUNCATE** es más rápido y utiliza menos recursos del sistema y del registro de transacciones.

### Sintaxis:

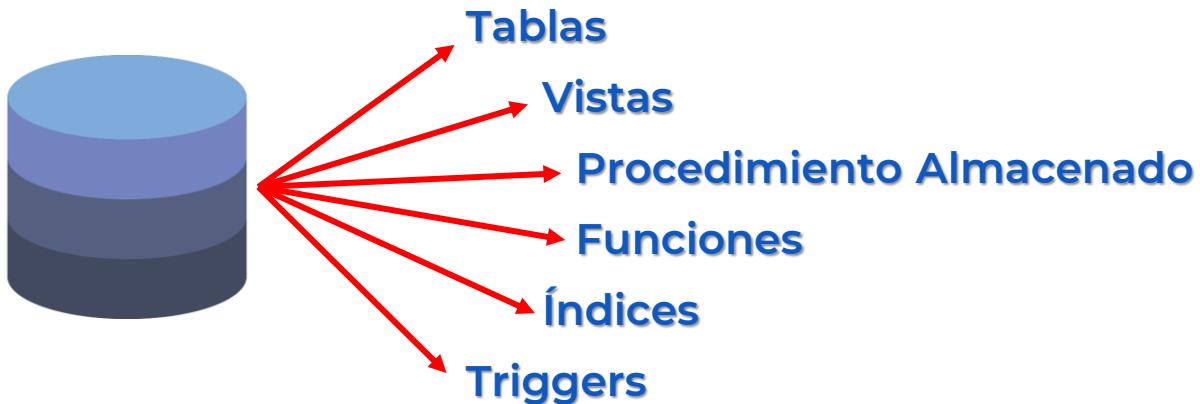
**TRUNCATE TABLE [nombre\_objeto];**



Nombre del objeto a truncar



# Esquemas(schema)

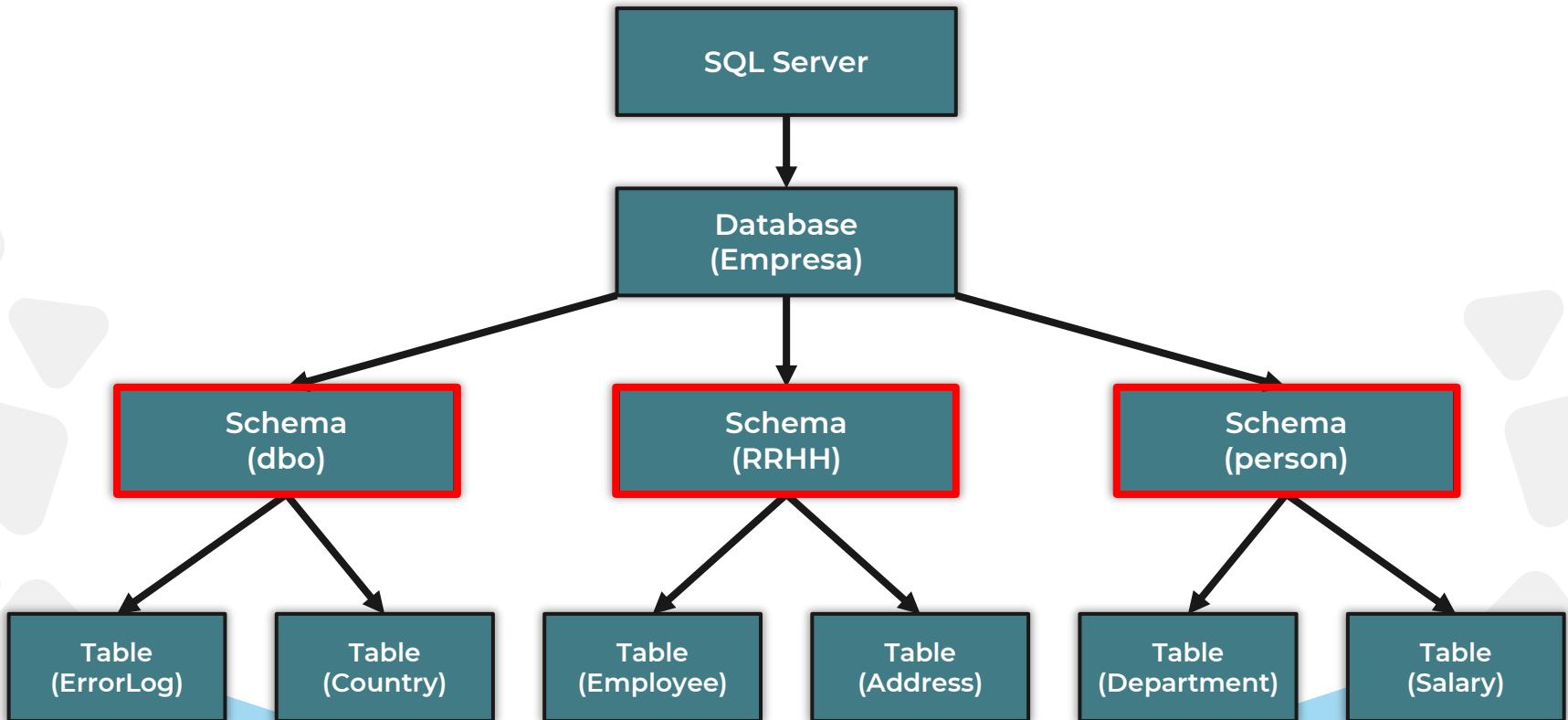


Un esquema es un objeto de base de datos que actúa como contenedor.



Los esquemas agrupan lógicamente objetos relacionados en un único contenedor que los mantiene separados de otros objetos de la base de datos en otros contenedores (esquemas).

# Esquemas(schema)



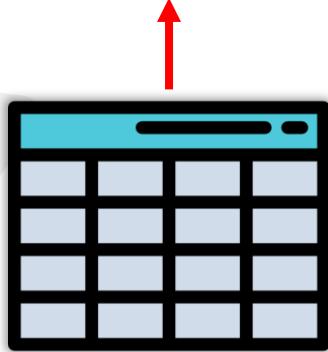
## Esquemas predeterminados de SQL Server

Schema  
(dbo)

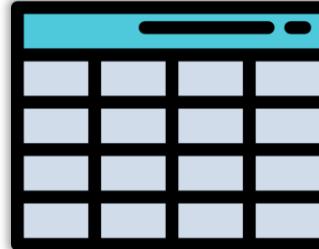
Schema  
(sys)

Schema  
(guest)

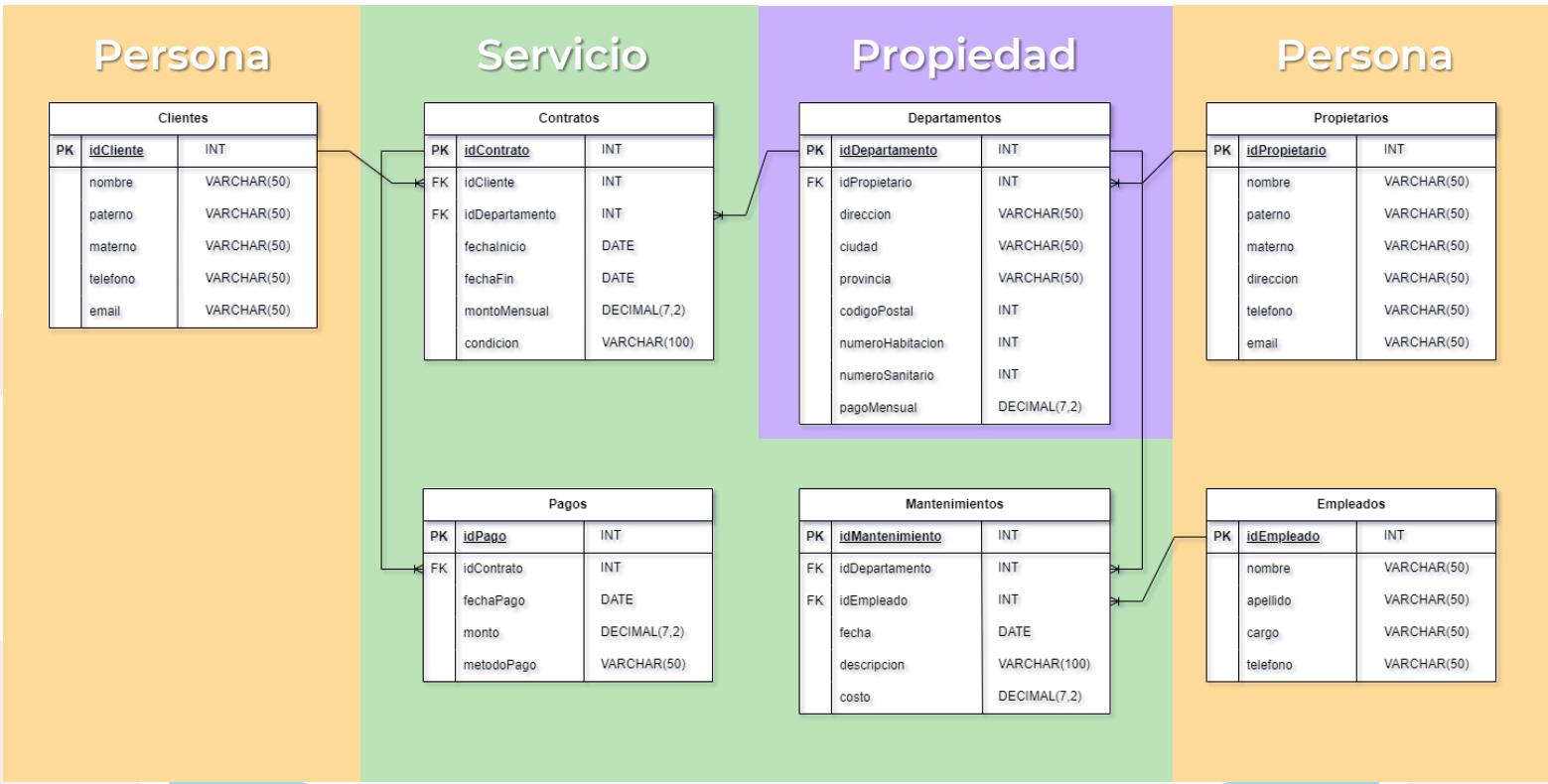
Schema  
(INFORMATION\_SCHEMA)



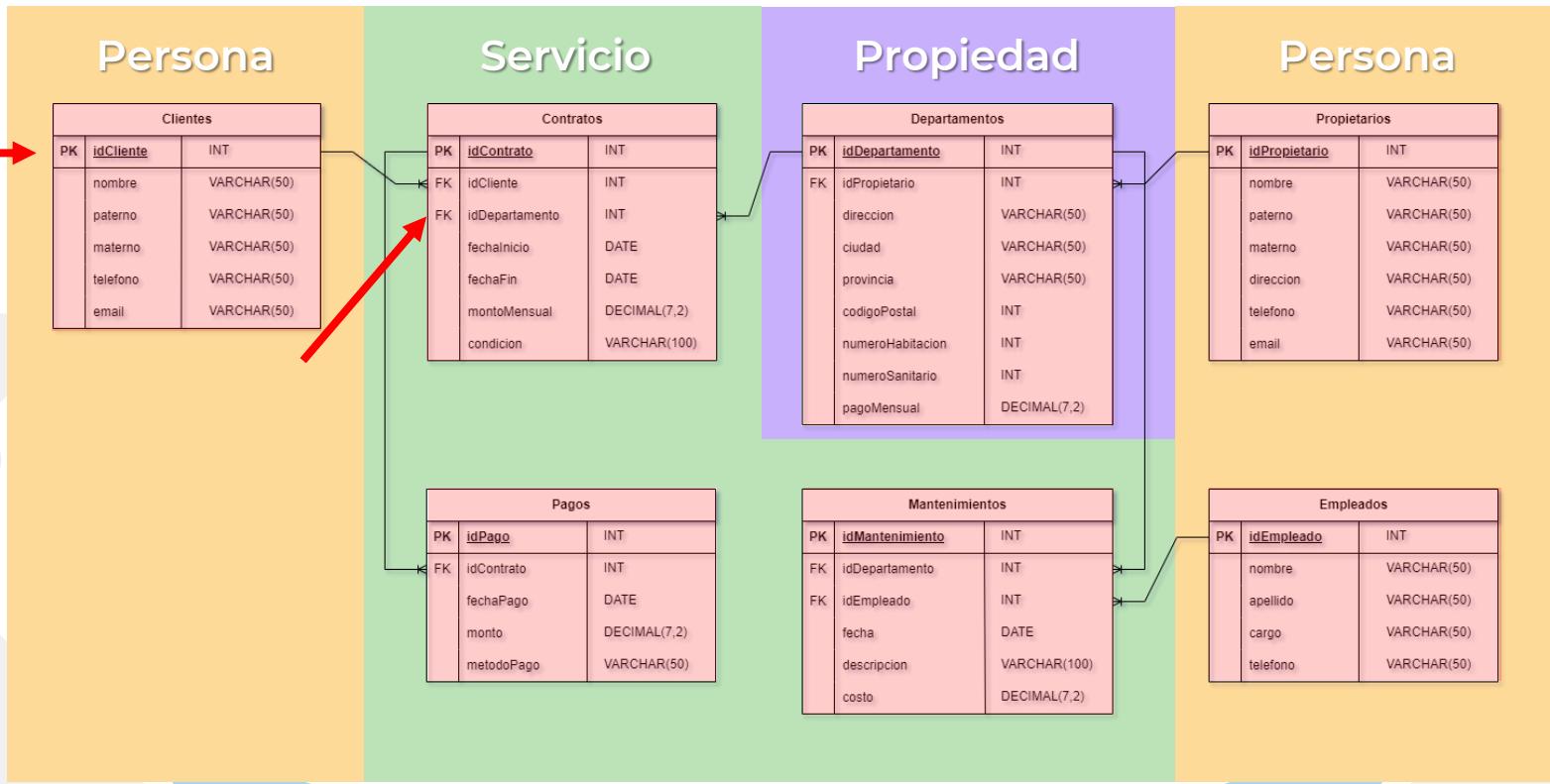
`CREATE SCHEMA NOMBRE_ESQUEMA;`



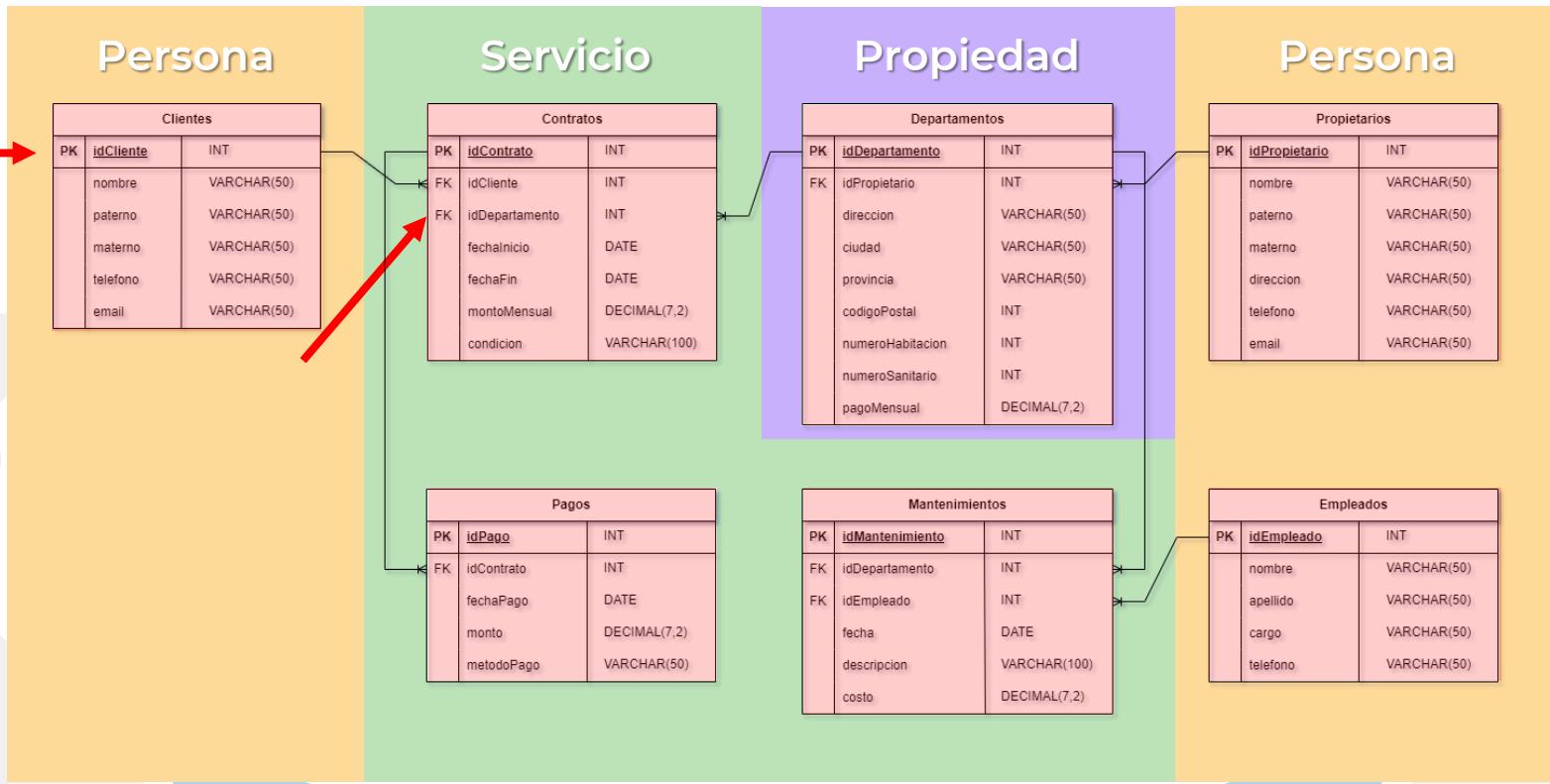
# Crear Esquema(Schema)

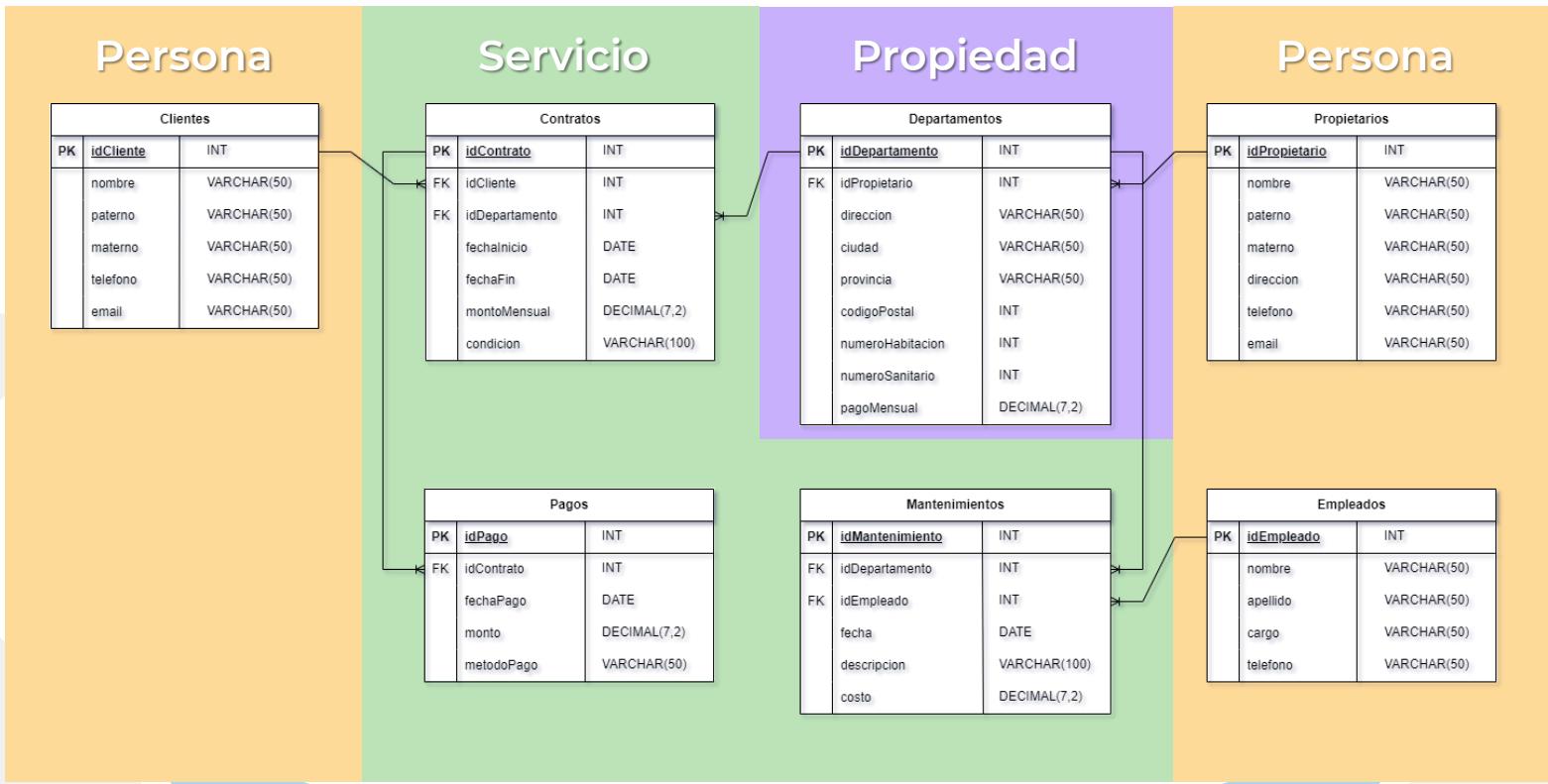


# Antes de la implementación

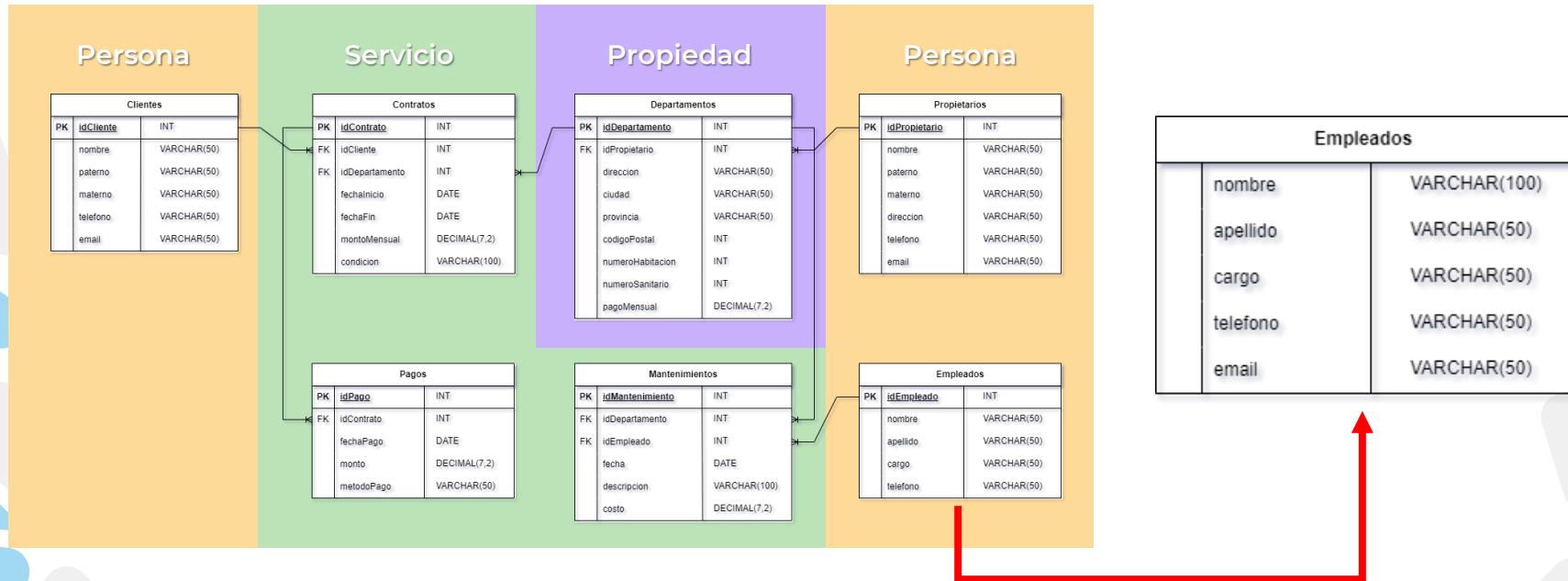


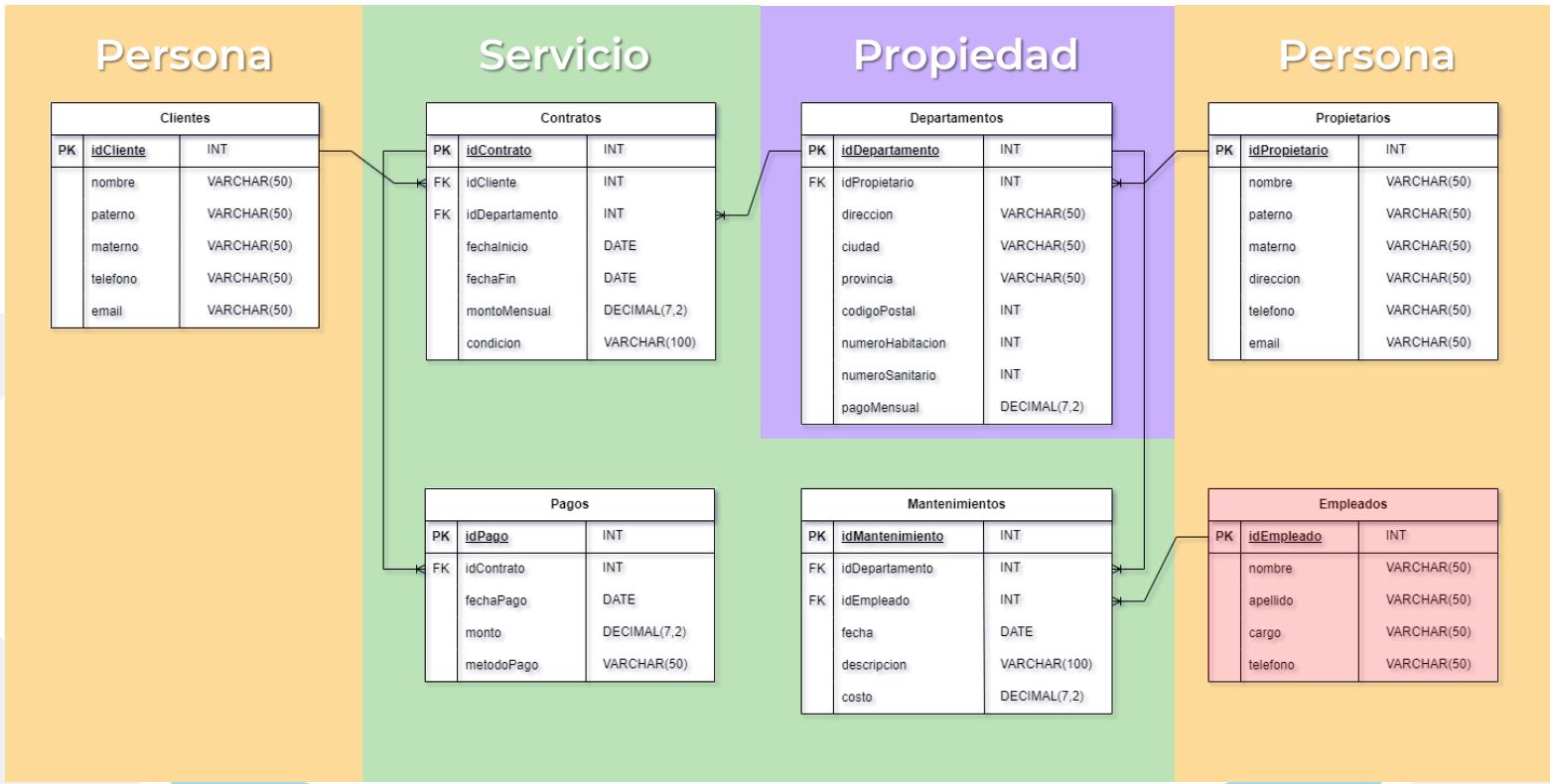
# Antes de la implementación



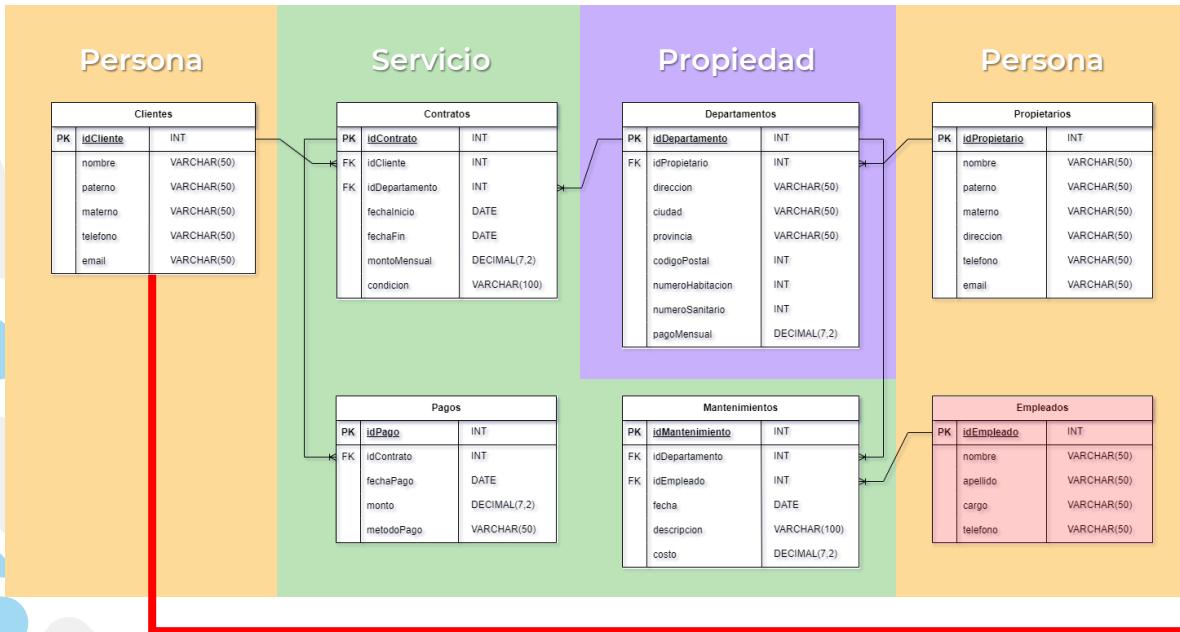


# Crear Tabla - Parte 1





# Crear Tabla - Parte 2



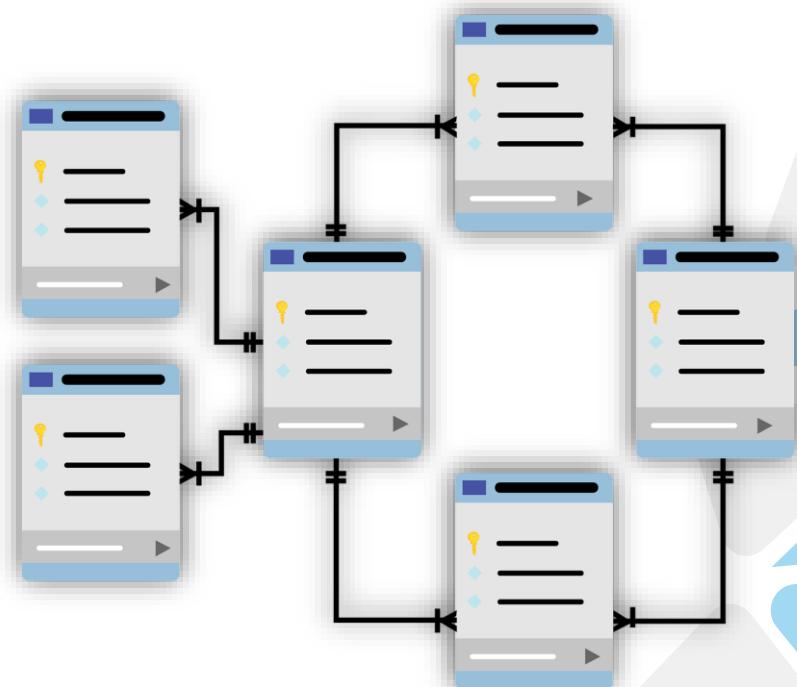
**Clientes**

PK	idCliente	INT
	nombre	VARCHAR(50)
	paterno	VARCHAR(50)
	materno	VARCHAR(50)
	telefono	VARCHAR(50)
	email	VARCHAR(50)

# Foreign Key

Una **Foreign Key** (clave foránea) es una **restricción** que se utiliza para asegurar la integridad referencial entre **dos tablas**. Es un campo (o conjunto de campos) en una tabla que se referencia a la **primary key** (clave primaria) en otra tabla.

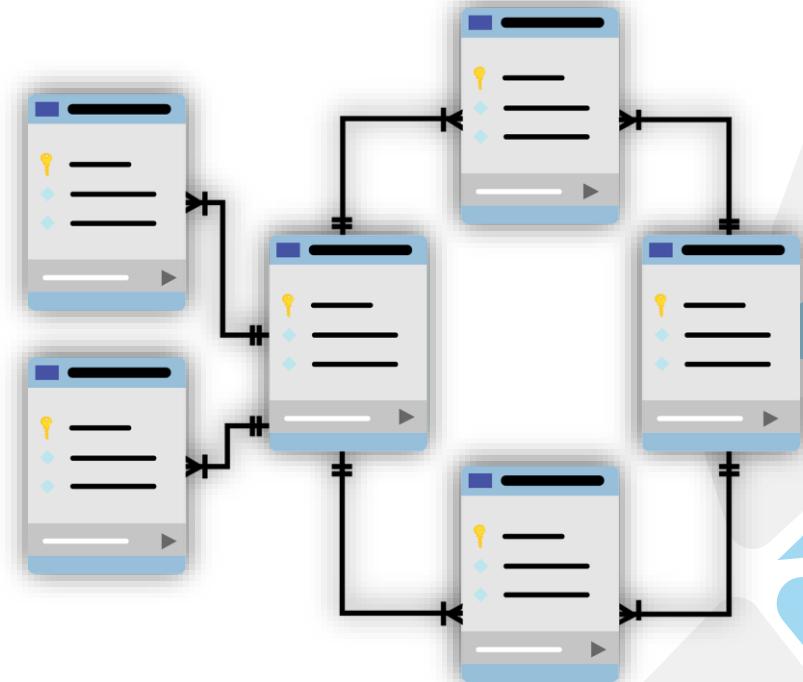
La **clave foránea** garantiza que los valores en una columna o conjunto de columnas **coincidan** con los **valores** en la clave primaria de la tabla **relacionada**, de manera que se mantenga la **coherencia** de los **datos** en la base de datos.



```
CREATE TABLE NOMBRE_TABLA  
(  
    colum1 ... ,  
    colum2 ... ,  
    colum3 ... ,  
    colum4 ...  
) ;
```

```
ALTER TABLE NOMBRE_TABLA ...
```

ON UPDATE ...  
ON DELETE ...



# Foreign Key

## ON UPDATE ... ON DELETE ...

Las cláusulas admitidas que se pueden realizar al eliminar o actualizar los valores de la “TABLA PRINCIPAL” incluyen:

### NO ACTION

Cuando las cláusulas **ON UPDATE** o **ON DELETE** se establecen en **NO ACTION**, la operación de actualización o eliminación realizada en la “**TABLA PRINCIPAL**” fallará con un error.  
Están son las cláusulas por “**defecto**” si no se establece las cláusulas **ON UPDATE** o **ON DELETE** dentro de las instrucciones.

### CASCADE

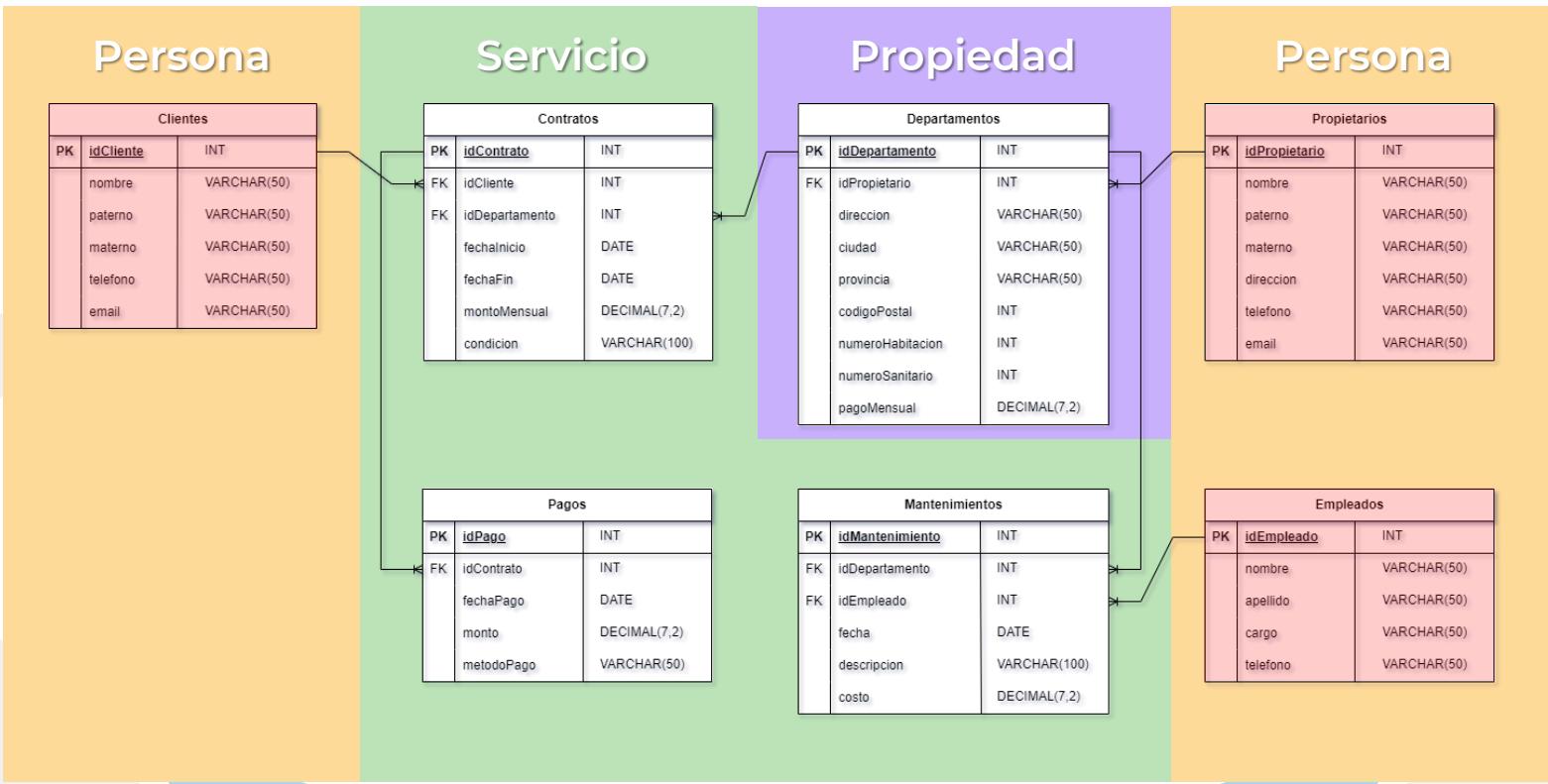
Al establecer las cláusulas **ON UPDATE** o **ON DELETE** en **CASCADE**, la misma acción realizada en los valores referenciados de la “**TABLA PRINCIPAL**” se reflejará en los valores relacionados en la “**TABLA SECUNDARIA**”. Por ejemplo, si el valor al que se hace referencia se elimina en la “**TABLA PRINCIPAL**”, también se eliminan todas las filas relacionadas en la “**TABLA SECUNDARIA**”.

### SET NULL

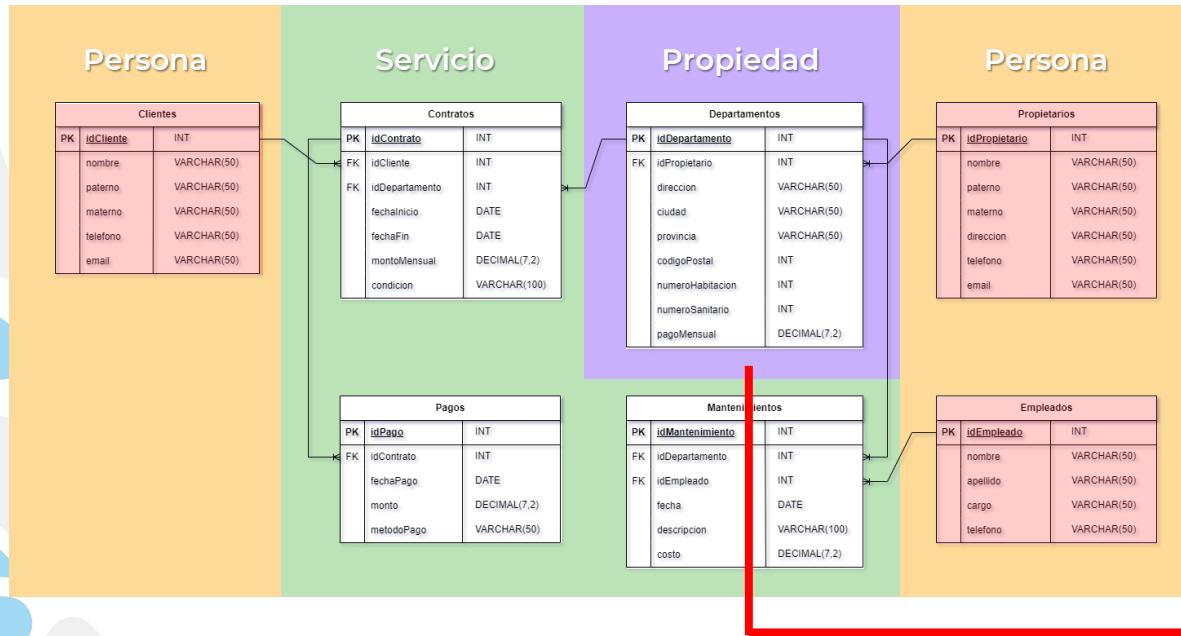
Con esta opción de cláusulas **ON UPDATE** y **ON DELETE**, si los valores a los que se hace referencia en la “**TABLA PRINCIPAL**” se eliminan o modifican, todos los valores relacionados en la “**TABLA SECUNDARIA**” se establecen en valor **NULL**.

### SET DEFAULT

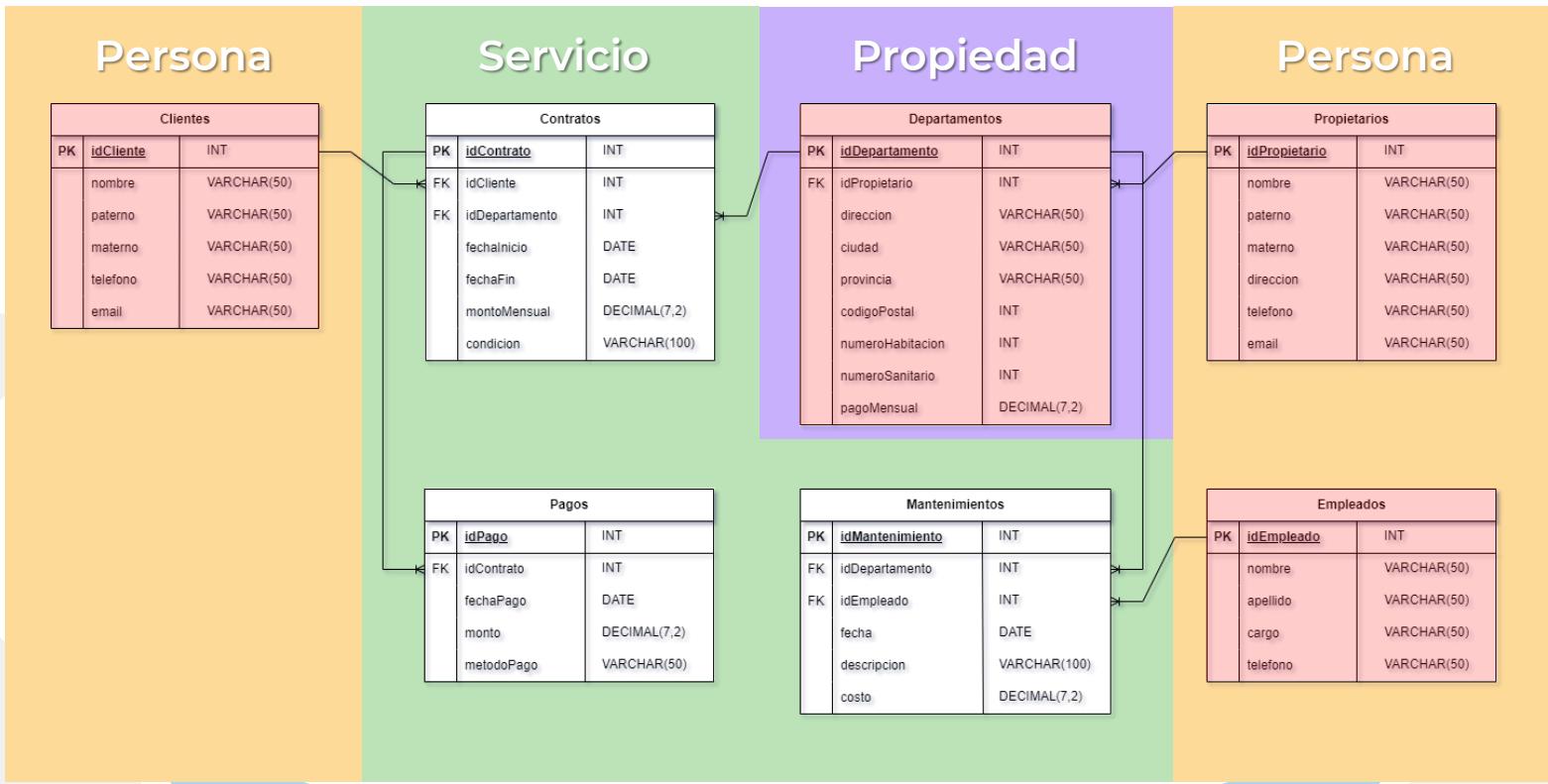
El uso de la opción **SET DEFAULT** de las cláusulas **ON UPDATE** y **ON DELETE** especifica que, si los valores a los que se hace referencia en la “**TABLA PRINCIPAL**” se actualizan o eliminan, los valores relacionados en la “**TABLA SECUNDARIA**” con columnas **FOREIGN KEY** se establecerán en su valor predeterminado.



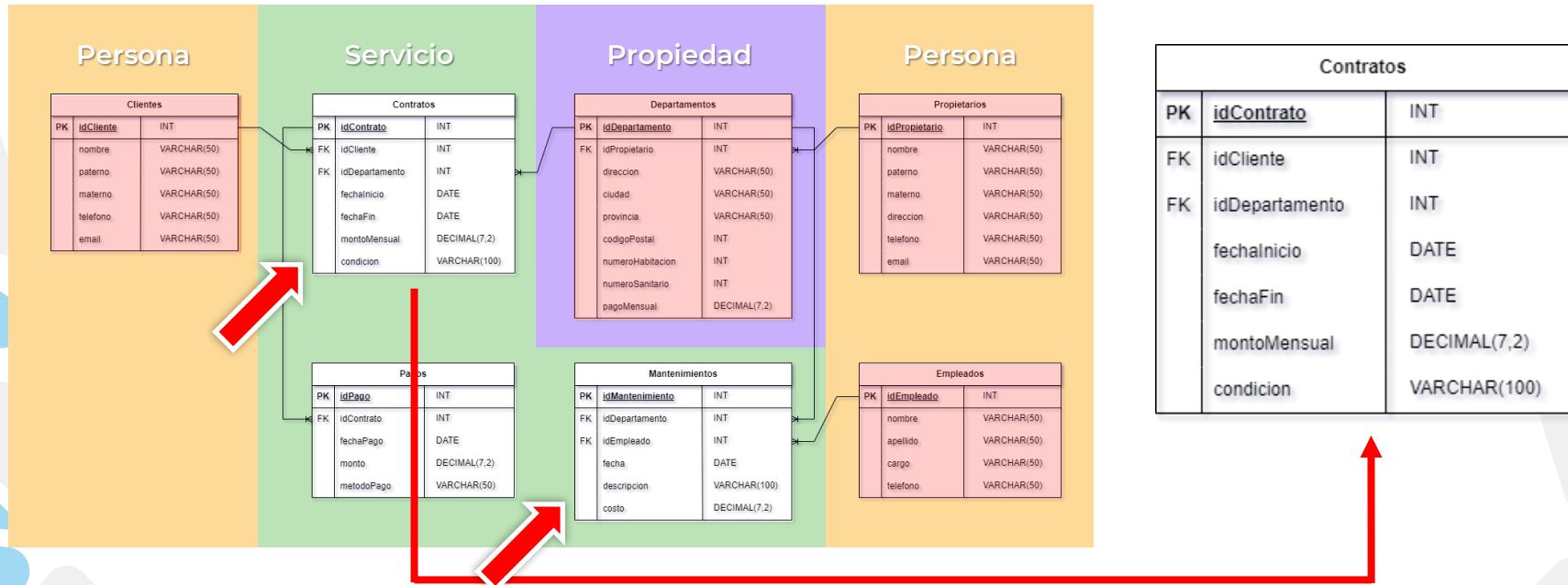
# Crear Tabla - Parte 4



Departamentos		
PK	idDepartamento	INT
FK	idPropietario	INT
	direccion	VARCHAR(50)
	ciudad	VARCHAR(50)
	provincia	VARCHAR(50)
	codigoPostal	INT
	numeroHabitacion	INT
	numeroSanitario	INT
	pagoMensual	DECIMAL(7,2)



# Crear Tabla - Parte 5



# TRUNCATE

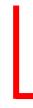
# TRUNCATE

La sentencia **TRUNCATE** es un comando del lenguaje de definición de datos (**DDL**) la cual elimina todas las filas de una tabla.

SQL Server almacena datos de una tabla en las **páginas** y el comando **TRUNCATE** elimina filas desasignando las páginas.

## Sintaxis:

**TRUNCATE TABLE [nombre\_bbdd] . [nombre\_esquema] .[nombre\_tabla];**



**DELETE FROM [nombre\_bbdd] . [nombre\_esquema] .[nombre\_tabla];**

# TRUNCATE

**TRUNCATE TABLE [nombre\_bbdd] . [nombre\_esquema] .[nombre\_tabla];**

Al utilizar la Sentencia **TRUNCATE** no se puede revertir los datos eliminados ya que no registra el registro durante la ejecución de esta operación. La ejecución de este comando bloquea las páginas en lugar de las filas; por lo tanto, requiere menos bloqueos y recursos. .

No se puede **TRUNCAR** una tabla a la que hace referencia una clave externa.

Si **TRUNCA** una tabla, se restablecerán los contadores de cualquier columna de IDENTITY.

Antes de poder **TRUNCAR** una tabla, debe tener los privilegios necesarios, como **ALTER TABLE**.

TRUNCAR una tabla es una forma rápida de Borrar registros de una tabla  
si no necesita preocuparse por revertirla.

Una recomendación, si no estas seguro de eliminar todos los registros de la Tabla te  
aconsejo que hagas un respaldo de la Base de Datos antes eliminar los Datos.

# TRUNCATE

## TRUNCATE

Es un comando **DDL**.

El comando **TRUNCATE** elimina todas las filas de una tabla. No podemos usar una cláusula **Where** en esto.

El comando **TRUNCATE** coloca un bloqueo de tabla y página para eliminar todos los registros.

El comando **TRUNCATE** no registra entradas para cada fila eliminada en el registro de transacciones.

Es más rápido que el comando **DELETE**.

## DELETE

Es un comando **DML**.

El comando **DELETE** es útil para eliminar todas o filas específicas de una tabla especificada mediante una cláusula **Where**.

El comando **DELETE** coloca un bloqueo en cada fila que requiere eliminarse de una tabla.

El comando **DELETE** registra una entrada por cada fila eliminada en el registro de transacciones.

El comando **DELETE** es más lento que el comando **TRUNCATE**.

# Descripción de la Sección

Tipos de Datos

Casuísticas de los Tipos de Datos

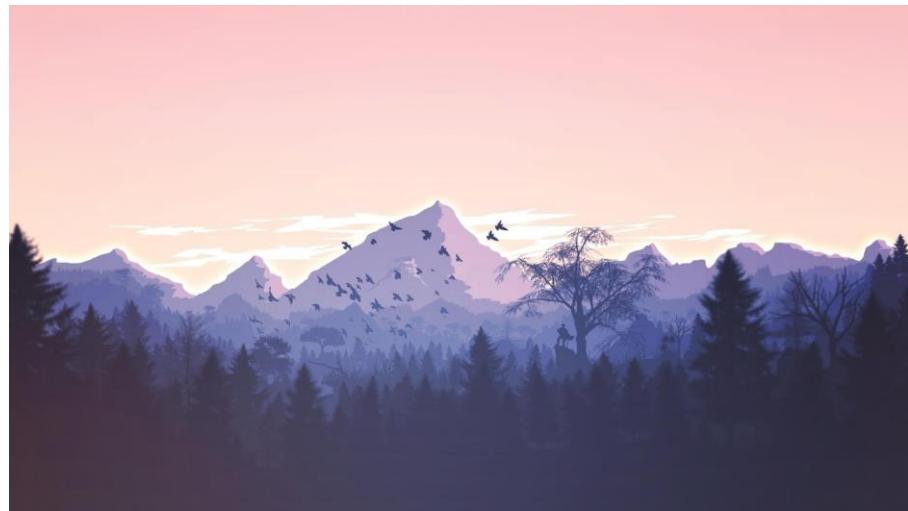
Tipos de Datos - Numéricos Exactos

Tipos de Datos - Numéricos Aproximados

Tipos de Datos - Cadenas de Caracteres

Tipos de Datos - Fecha y hora

Tipos de Datos - Cadenas Binarias



# Tipos de Datos

## Tipos de Datos

Los tipos de datos son una parte fundamental para definir la naturaleza de la información que puede almacenarse en una base de datos. Estos tipos de datos permiten a SQL Server gestionar y almacenar datos de manera eficiente, garantizando la integridad y exactitud de la información.

Un tipo de datos define qué tipo de valor puede contener una columna: datos enteros, datos de caracteres, datos monetarios, datos de fecha y hora, cadenas binarias, etc.

Se requiere que cada columna en una tabla de base de datos tenga un nombre y un tipo de datos.

Un desarrollador de SQL debe decidir qué tipo de datos se almacenarán dentro de cada columna al crear una tabla. El tipo de datos es una guía para que SQL entienda qué tipo de datos se espera dentro de cada columna, y también identifica cómo SQL interactuará con los datos almacenados.

Cada tipo de datos tiene su propio uso y características específicas, y la elección del tipo de datos adecuado es crucial para optimizar el rendimiento y la integridad de las Bases de Datos.

# Tipos de Datos

## Valores Numéricos Exactos

*Numéricos Exactos  
Entero*

*Numéricos Exactos  
Decimal*

*Numéricos Exactos  
Lógico*

*Numéricos Exactos  
Moneda*

## Valores Numéricos Aproximados

Cadenas de Caracteres

Cadenas de Caracteres Unicode

Fecha y Hora

Cadenas Binarias

Otros Tipos de Datos

# Tipos de Datos

## Valores Numéricos Exactos

### *Numéricos Exactos Entero*

#### **tinyint**

Intervalo: De 0 a 255

Expresión de intervalo:  $2^0 - 1$  a  $2^8 - 1$

Storage: 1 Byte

#### **smallint**

Intervalo: De -32 768 a 32 767

Expresión de intervalo:  $-2^{15} - 1$  a  $2^{15} - 1$

Storage: 2 Bytes

#### **int**

Intervalo: De -2.147.483.648 a  
2.147.483.647

Expresión de intervalo:  $-2^{31} - 1$  a  $2^{31} - 1$

Storage: 4 Bytes

#### **bigint**

Intervalo: De -9.223.372.036.854.775.808 a  
9.223.372.036.854.775.807

Expresión de intervalo:  $-2^{63} - 1$  a  $2^{63} - 1$

Storage: 8 Bytes

# Tipos de Datos

## Valores Numéricos Exactos

### Numéricos Exactos Decimal

`decimal(p, s)`

`numeric(p, s)`

#### p (precisión)

El número total máximo de dígitos decimales que se almacenarán.

#### s (escala)

El número de dígitos decimales que se almacenarán a la derecha del separador decimal.

Expresión de intervalo:  $-10^{38} + 1$  a  $10^{38} - 1$

Precisión	Almacenamiento
1 – 9	(5 Bytes)
10 – 19	(9 Bytes)
20 – 28	(13 Bytes)
29 – 38	(17 Bytes)

Estos Tipos de Datos son idénticos, sólo que tienen nombres diferentes.

# Tipos de Datos

## Valores Numéricos Exactos

### *Numéricos Exactos Lógico*

#### **bit**

Intervalo: 1 ó 0

Storage: 1 bit

## Tipos de Datos

### Valores Numéricos Exactos

#### *Numéricos Exactos Moneda*

##### **smallmoney**

Intervalo: De -214,748.3648 a 214,748.3647

Storage: 4 Bytes

##### **money**

Intervalo: De -922,337,203,685,477.5808 a 922,337,203,685,477.5807

Storage: 8 Bytes

### Decimales vs Moneda

Entonces, a opinión personal no recomiendo utilizar el Tipo de Dato **Money** debido a que no es preciso; utilice mejor el Tipo de Dato **Decimal**.

# Tipos de Datos

## Valores Numéricos Aproximados

### Float(n)

Intervalo: De - 1,79E+308 a -2,23E-308, 0 y de 2,23E-308 a 1,79E+308

Storage: 4 Bytes (n = 1 – 24, 7 dígitos)  
8 Bytes (n = 25 – 53, 15 dígitos)

### real

Intervalo: De - 3,40E + 38 a -1,18E - 38, 0 y de 1,18E - 38 a 3,40E + 38

Storage: 4 Bytes

n → Es el número de bits que se usan para almacenar la mantisa del número de float.

Si se especifica n, debe ser un valor entre 1 y 53. El valor predeterminado de n es 53.

### Decimales vs Números Aproximados

Entonces, a opinión personal no recomiendo utilizar el Tipo de Dato de **Valores Numéricos Aproximados** debido a que no es preciso; utilice mejor el tipo de dato **Decimal**.

# Tipos de Datos

## Cadenas de Caracteres

### char(n)

Longitud Fija

Valor n: 1 hasta 8000  
Storage: n Bytes

### varchar(n)

Longitud Variable

Valor n: 1 hasta 8000  
Storage: n + 2 Bytes

### text

Longitud Variable

**Obsoleto**



Donde “n” especifica el tamaño de la columna en “Bytes”.



### varchar(max)

Longitud Variable

Storage:  $2^{31}-1$  Bytes (2GB)

# Tipos de Datos

## Cadenas de Caracteres Unicode

### nchar(n)

Longitud Fija

Valor n: 1 hasta 4000  
Storage:  $n^2$  Bytes

### nvarchar(n)

Longitud Variable

Valor n: 1 hasta 4000  
Storage:  $n^2 + 2$  Bytes

### ntext

Longitud Variable

Obsolete

Donde el parámetro “n” especifica el tamaño de la columna en “pares de Bytes”.



### nvarchar(max)

Longitud Variable

Storage:  $2^{31}-1$  Bytes (2GB)

# Tipos de Datos

## Fecha y Hora

### time

Formato: hh:mm:ss[.nnnnnnn]

Storage: 5 Bytes

### date

Formato: yyyy-MM-dd

Storage: 3 Bytes

### smalldatetime

Formato:  
yyyy-MM-dd hh:mm:ss

Storage: 4 Bytes

### datetime

Formato:  
yyyy-MM-dd hh:mm:ss[.nnn]

Storage: 8 Bytes

### datetime2

Formato:  
yyyy-MM-dd hh:mm:ss[. nnnnnnnn]

Storage: 6/7/8 Bytes

### datetimeoffset

Formato:  
yyyy-MM-dd hh:mm:ss[.nnnnnnnn]  
[{+|-}hh:mm]

Storage: 10 Bytes

# Tipos de Datos

## Cadenas Binarias

### **binary(n)**

Longitud Fija

Valor n : 1 hasta 8000  
Storage: n Bytes

### **varbinary(n | max)**

Longitud Variable

Valor n : 1 hasta 8000  
Valor max:  $2^{31} - 1$  Bytes  
Storage : n + 2 Bytes

### **image**

Longitud Variable

Storage: desde 0 hasta  
 $2^{31} - 1$  Bytes

Donde “n” especifica el tamaño de la columna en “Bytes”.

# Tipos de Datos

## Otros Tipos de Datos

**cursor**

**rowversion**

**geography**

**Sql\_variant**

**geometry**

**table**

**hierarchyid**

**uniqueidentifier**

**json**

**xml**

# Tipos de Datos

## COLLATE(Collation)

COLLATE representa la nomenclatura de símbolos; es decir, es la **propiedad de la base de datos** que proporciona la distinción entre **caracteres** como acentos, mayúsculas, minúsculas, otros caracteres soportados y **reglas de ordenación** para los datos que la base de datos es capaz de manejar.

COLLATE especificará a la base de datos tanto **los patrones de bits que representan cada carácter** contenido en el juego de caracteres vinculados a una región, como las reglas de comparación u ordenación de datos.

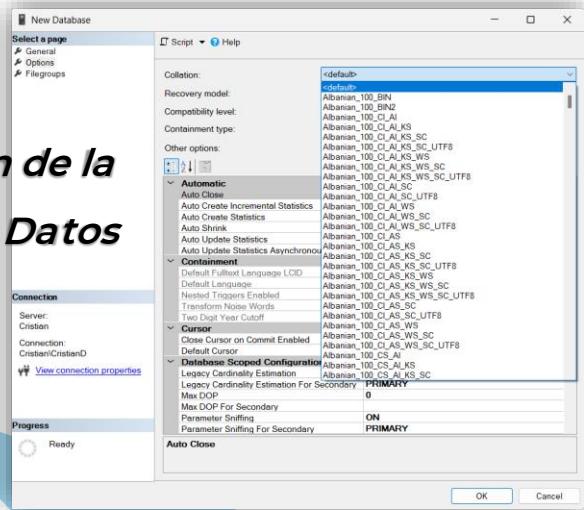
Cada juego de caracteres es diferente

Ñ

COLLATE viene condicionado por la región donde se vaya a instalar y explotar la base de datos

## COLLATE(Collation)

```
SELECT FirstName, LastName  
FROM Employees  
WHERE LastName COLLATE Modern_Spanish_CI_AS = 'Davolio';
```



## *Creación de la Base de Datos*



## *Instalar servidor SQL Server*

## COLLATE(Collation)

```
SELECT FirstName, LastName  
FROM Employees  
WHERE LastName COLLATE Modern_Spanish_CI_AS = 'Davolio';
```

**DD-MM-YYYY**

**YYYY-MM-DD**

Otro aspecto importante del COLLATE es que en Latinoamérica la sintaxis de consulta NO ES key sensitive pero si se cambia el COLLATE si se puede convertir en KEY SENSITIVE SQL Server.

## Casuísticas de los Tipos de Datos

## Valores Numéricos Exactos

**tinyint**

Representa la edad de un cliente(18 - 90) que solicita un crédito.  
1 Byte

**int → 4 Bytes**

CREDITOS	
	nombre
	edad
	monto
	seguro
	disponible

**decimal**

Representa el valor numérico decimal(5,2).  
5 Bytes

**bit**

Establece si su crédito se encuentra disponible o no.  
1 bit

**int → 4 byte**

**char**

**Verdadero / Falso**

1 -> Disponible

0 -> No disponible

## Fecha y Hora

**date**

Almacena la fecha de registro de un paquete.  
3 Bytes

**time**

Almacena la hora de registro de un paquete.  
5 Bytes

Paquete	
PK	<u>codigo</u>
	fecha_registro
	hora
	fecha_envio
	fecha_entrega

**datetime/datetime2**

Almacena la fecha y hora de la fecha de envío y la fecha de entrega del paquete.  
6/7/8 Bytes

## Cadenas de Caracteres

Longitud Fija

LIBROS	
PK	<u>ISBN</u>
	autor
	editorial
	categoria
	año
	fecha_publicacion

Longitud  
Variable

### Cadenas de Caracteres

**Definir el tipo de dato que se ajuste los  
Datos que se van a almacenar**

## Cadenas de Caracteres

Definir el tipo de dato que se ajuste los  
Datos que se van a almacenar

LIBROS	
PK	<u>ISBN</u>
	autor
	editorial
	categoria
	año
	fecha_publicacion



**ISBN**

**978-3-16-148410-0**

**17 caracteres**

**char(17) → 17 bytes**

**varchar(17) → 19 bytes**

**nchar(17) → 34 bytes**

**nvarchar(17) → 36 bytes**

## Cadenas de Caracteres

Definir el tipo de dato que se ajuste los  
Datos que se van a almacenar

LIBROS	
PK	<u>ISBN</u>
	autor
	editorial
	categoria
	año
	fecha_publicacion



**ISBN**

**9783161484100**

**13 caracteres**

**char(13) → 13 bytes**

**bigint → 8 bytes**

## Cadenas de Caracteres

Definir el tipo de dato que se ajuste los  
Datos que se van a almacenar

PRODUCTOS	
PK	
	<u>codigo</u>
	nombre
	precio
	stock
	descripcion

## Longitud Variable

¿Sabemos la longitud del nombre?

**varchar(9)** → Muy pequeño

**varchar(90)** → Muy grande

**varchar(30)** → Ideal

**varchar(255)** → Muy pequeño

**varchar(max)** → Ideal

## Cadenas de Caracteres

Definir el tipo de dato que se ajuste los  
Datos que se van a almacenar

PRODUCTOS	
PK	<u>codigo</u>
	nombre
	precio
	stock
	descripcion

## Longitud Variable

¿Cuando utilizar Unicode?

nvarchar(30) → Ideal

Cuanto se almacena datos en  
distintos idiomas

Cuando las políticas de la empresa  
así lo especifiquen

## Cadenas de Caracteres(Unicode y No Unicode)

## Tabla ASCII

Los Caracteres que se encuentran dentro de la Tabla **ASCII(0-127)** ocupan solo **1 Byte** para los Tipos de Datos **CHAR** y **VARCHAR**. Para el Tipo de Dato **NCHAR** y **NVARCHAR** ocupa un almacenamiento de **2 Bytes**.

Dec	Hex	Binary	Char	Dec	Hex	Binary	Char	Dec	Hex	Binary	Char	Dec	Hex	Binary	Char
0	0x00	00 00000	NUL	32	0x20	01 00000	SPACE	64	0x40	10 00000	Ø	96	0x60	11 00000	`
1	0x01	00 00001	SOH	33	0x21	01 00001	!	65	0x41	10 00001	A	97	0x61	11 00001	a
2	0x02	00 00010	STX	34	0x22	01 00010	"	66	0x42	10 00010	B	98	0x62	11 00010	b
3	0x03	00 00011	ETX	35	0x23	01 00011	#	67	0x43	10 00011	C	99	0x63	11 00011	c
4	0x04	00 00100	EOT	36	0x24	01 00100	\$	68	0x44	10 00100	D	100	0x64	11 00100	d
5	0x05	00 00101	ENQ	37	0x25	01 00101	%	69	0x45	10 00101	E	101	0x65	11 00101	e
6	0x06	00 00110	ACK	38	0x26	01 00110	&	70	0x46	10 00110	F	102	0x66	11 00110	f
7	0x07	00 00111	BEL	39	0x27	01 00111	'	71	0x47	10 00111	G	103	0x67	11 00111	g
8	0x08	00 01000	BS	40	0x28	01 01000	(	72	0x48	10 01000	H	104	0x68	11 01000	h
9	0x09	00 01001	HT	41	0x29	01 01001	)	73	0x49	10 01001	I	105	0x69	11 01001	i
10	0x0a	00 01010	LF	42	0x2a	01 01010	*	74	0x4a	10 01010	J	106	0x6a	11 01010	j
11	0x0b	00 01011	VT	43	0x2b	01 01011	+	75	0x4b	10 01011	K	107	0x6b	11 01011	k
12	0x0c	00 01100	FF	44	0x2c	01 01100	,	76	0x4c	10 01100	L	108	0x6c	11 01100	l
13	0x0d	00 01101	CR	45	0x2d	01 01101	-	77	0x4d	10 01101	M	109	0x6d	11 01101	m
14	0x0e	00 01110	SO	46	0x2e	01 01110	.	78	0x4e	10 01110	N	110	0x6e	11 01110	n
15	0x0f	00 01111	SI	47	0x2f	01 01111	/	79	0x4f	10 01111	O	111	0x6f	11 01111	o
16	0x10	00 10000	DLE	48	0x30	01 10000	0	80	0x50	10 10000	P	112	0x70	11 10000	p
17	0x11	00 10001	DC1	49	0x31	01 10001	1	81	0x51	10 10001	Q	113	0x71	11 10001	q
18	0x12	00 10010	DC2	50	0x32	01 10010	2	82	0x52	10 10010	R	114	0x72	11 10010	r
19	0x13	00 10011	DC3	51	0x33	01 10011	3	83	0x53	10 10011	S	115	0x73	11 10011	s
20	0x14	00 10100	DC4	52	0x34	01 10100	4	84	0x54	10 10100	T	116	0x74	11 10100	t
21	0x15	00 10101	NAK	53	0x35	01 10101	5	85	0x55	10 10101	U	117	0x75	11 10101	u
22	0x16	00 10110	SYN	54	0x36	01 10110	6	86	0x56	10 10110	V	118	0x76	11 10110	v
23	0x17	00 10111	ETB	55	0x37	01 10111	7	87	0x57	10 10111	W	119	0x77	11 10111	w
24	0x18	00 11000	CAN	56	0x38	01 11000	8	88	0x58	10 11000	X	120	0x78	11 11000	x
25	0x19	00 11001	EM	57	0x39	01 11001	9	89	0x59	10 11001	Y	121	0x79	11 11001	y
26	0x1a	00 11010	SUB	58	0x3a	01 11010	:	90	0x5a	10 11010	Z	122	0x7a	11 11010	z
27	0x1b	00 11011	ESC	59	0x3b	01 11011	;	91	0x5b	10 11011	[	123	0x7b	11 11011	{
28	0x1c	00 11100	FS	60	0x3c	01 11100	<	92	0x5c	10 11100	\	124	0x7c	11 11100	
29	0x1d	00 11101	GS	61	0x3d	01 11101	=	93	0x5d	10 11101	]	125	0x7d	11 11101	}
30	0x1e	00 11110	RS	62	0x3e	01 11110	>	94	0x5e	10 11110	^	126	0x7e	11 11110	-
31	0x1f	00 11111	US	63	0x3f	01 11111	?	95	0x5f	10 11111	_	127	0x7f	11 11111	DEL

Para los Caracteres que son de tipo Asiáticos ocupan un almacenamiento de **2 o 3 Bytes** por carácter si se utilizar el Tipo de Dato **CHAR** o **VARCHAR**.

Para el Tipo de Dato **NCHAR** o **NVARCHAR** ocupa un almacenamiento de **2 Bytes**.

Chinese: 你好世界

Japanese: こんにちは世界

Korean: 안녕하세요 세계

# Descripción de la Sección

Fundamentos de las Variables

BATCHES o LOTES de instrucciones

Variable

Ámbito de la Variable

Variable con la instrucción SELECT

Instrucción EXECUTE



# Variables

# Variables

Las variables son objetos de almacenamiento temporal que pueden contener datos y se utilizan dentro de un “Batch o Lote” de instrucciones. Las variables son esenciales para la programación en T-SQL porque permiten almacenar datos, realizar cálculos y controlar el flujo de ejecución.

Las variables se pueden crear para diferentes tipos de datos y también se les pueden asignar valores. Además, los valores asignados a las variables se pueden cambiar durante el período de ejecución. El ciclo de vida de la variable comienza desde el punto en el que se declara y debe finalizar al final del “Batch o Lote” de instrucciones.

Las variables son siempre locales dentro del “Batch o Lote” de instrucción en el que se declara.

## Sintaxis:

```
DECLARE @NOMBRE_VARIABLE TIPO_DATO [=VALOR] , ...
```

```
SET @NOMBRE_VARIABLE = VALOR
```

```
[ PRINT | SET ] @NOMBRE_VARIABLE
```

```
GO
```



# Batches(Lotes)

## Batches(Lotes)

Un “Batch o Lote” de instrucción es una colección de una o más instrucciones Transact-SQL (T-SQL) enviadas al servidor para su ejecución como una sola unidad. Los batches permiten agrupar y enviar múltiples instrucciones SQL al servidor en una sola operación, lo cual puede mejorar la eficiencia y la gestión del código SQL.

El “Batch o Lote” terminan con la instrucción “GO” por defecto.

Los “Batch o Lote” limitan el alcance de las Variables.

## Batches(Lotes)

La instrucción “GO” fue introducida por las herramientas de Microsoft como una forma de separar declaraciones por lotes, la instrucción “GO” es compatible con las herramientas de Microsoft SQL, pero formalmente no forma parte de otras herramientas.

No puede poner a GO en una cadena de SQL y enviarlo como parte de un objeto de comando ADO.NET ya que el propio SQL no comprende el término.

Algo importante a aclarar es el punto y coma(); la cual se utiliza para indicar el final de una declaración en sí, no necesariamente de un lote completo. Además es un estándar en SQL y ayuda a evitar ambigüedades en la interpretación de las instrucciones.

`SELECT column1, column2, ... columnN  
FROM nombre_tabla`

`GO`

`SELECT column1, column2, ... columnN  
FROM nombre_tabla`

`GO`

`SELECT column1, column2, ... columnN  
FROM nombre_tabla;`

`SELECT column1, column2, ... columnN  
FROM nombre_tabla;`

# Descripción de la Sección

Fundamentos de Estructuras de Control de Fluxos

Bloque de Ejecución - BEGIN ... END

Variables

Instrucción IF ... ELSE

Instrucción CASE

Instrucción WHILE

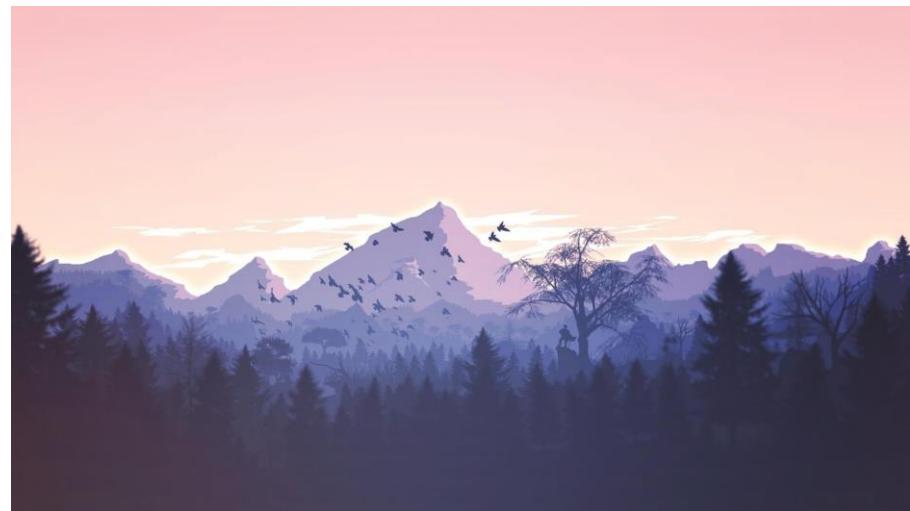
Instrucción BREAK y CONTINUE

Instrucción RETURN

Instrucción GOTO

Iniciando con el Control de Errores - TRY...CATCH

Iniciando con la instrucción THROW



# Estructuras de Control de Flujos

Esta sección tiene como objetivo **desarrollar la lógica** para poder resolver problemas y ejercicios que se van a plantear y van a ir aumentando la dificultad a lo largo del curso.

La estructura de control de flujo se refiere a las instrucciones y construcciones que permiten dirigir la ejecución del código Transact-SQL de acuerdo con condiciones específicas, decisiones lógicas y bucles.

Estas estructuras permiten controlar cómo se ejecutan las sentencias SQL y cómo se manejan las condiciones y los errores en los scripts.

Esta sección tiene como objetivo **desarrollar la lógica** para poder resolver problemas y ejercicios que se van a plantear y van a ir aumentando la dificultad a lo largo del curso.

BEGIN...END

IF...ELSE

CASE

WHILE

BREAK y CONTINUE

RETURN

GOTO

TRY...CATCH

Las estructuras de control de flujo son esenciales para escribir scripts Transact-SQL robustos y flexibles. Permiten realizar tareas condicionales, tareas repetitivas, manejo de errores y mejorar la lógica de procesamiento de datos. Estas herramientas son fundamentales para desarrollar nuestra lógica, poder resolver problemas, ejercicios y mediante el código Transact-SQL solucionar diversas situaciones de manera eficiente y efectiva.

## OPERADORES DE COMPARACIÓN

- Es igual a (=)
- Mayor que (>)
- Mayor o igual que (>=)
- Menor o igual que(<=)
- No es igual a(!= , <>)
- No es menor que (!<)
- No es mayor que (!>)

## OPERADORES LÓGICOS

- Operador Lógico - "AND"
- Operador Lógico - "OR"
- Operador Lógico - "IN"
- Operador Lógico - "LIKE"
- Operador Lógico - "BETWEEN"
- Operador Lógico - "NOT"
- **Operador Lógico - "EXISTS"**

## OPERADORES ARITMÉTICOS

- Suma (+) y Resta (-)
- Multiplicar (\*) y Dividir (/)
- Módulo (%)

# Descripción de la Sección

Fundamentos de las Funciones(Functions)

Funciones Matemáticas

Funciones Lógicas

Funciones de Cadena

Funciones de Conversión

Funciones de Fecha y Hora

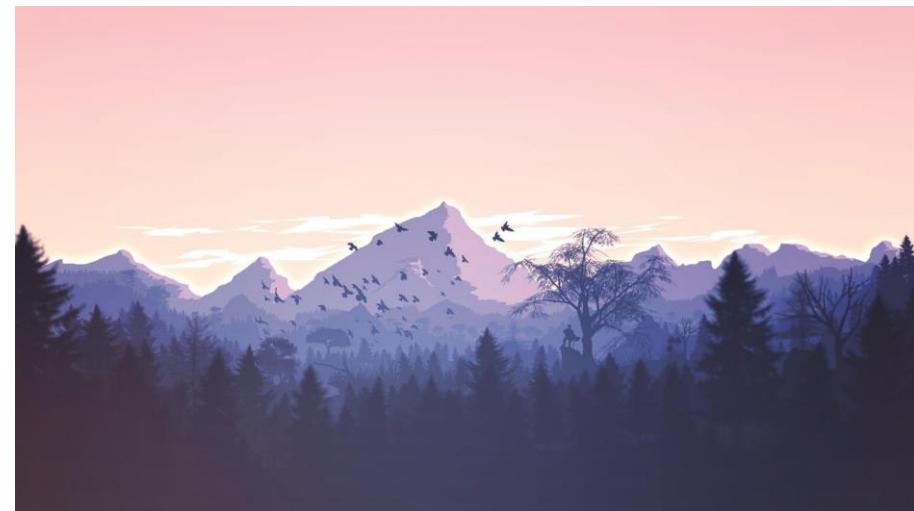
Funciones del Sistema

Funciones de Agregación(Aggregate Functions)

Funciones de Clasificación

Funciones Analíticas

Funciones Definidas por el Usuario



# Funciones(Functions)

- Las funciones son programas predefinidos o definidos por el usuario
  - que realizan una tarea específica y devuelven un valor escalar o una tabla. Las funciones se utilizan para realizar operaciones en los datos almacenados en una base de datos y pueden ser invocadas en consultas SQL o en otras funciones.

Existen dos tipos de Funciones dentro de SQL Server

**Funciones del Sistema**

**Funciones Definidas por el Usuario**

## Funciones del Sistema

### Built-In Functions (Funciones Integradas)

**Aggregate Functions**  
(Funciones de Agregación)

**Analytic Functions**  
(Funciones Analíticas)

**Bit Manipulation Functions**  
(Funciones de Manipulación de Bits)

**Collation Functions**  
(Funciones de Intercalación)

**Configuration Functions**  
(Funciones de Configuración)

**Conversion Functions**  
(Funciones de Conversión)

**Cryptographic Functions**  
(Funciones de Criptográficas)

**Cursor Functions**  
(Funciones de Cursor)

**Data Type Functions**  
(Funciones de Tipos de Datos)

**Date and Time Functions**  
(Funciones de Fecha y Hora)

**Graph Functions**  
(Funciones de Grafico)

**JSON Functions**  
(Funciones JSON)

**Mathematical Functions**  
(Funciones Matemáticas)

**Logical Functions**  
(Funciones Lógicas)

**Metadata Functions**  
(Funciones de Metadatos)

**Ranking Functions**  
(Funciones de Clasificación)

**Replication Functions**  
(Funciones de Replicación)

**Security Functions**  
(Funciones de Seguridad)

**String Functions**  
(Funciones de Cadena)

**System Functions**  
(Funciones del Sistema)

**System Statistical Functions**  
(Funciones Estadísticas del Sistema)

**Text and Image Functions**  
(Funciones de Texto e Imagen)

**Trigger Functions**  
(Funciones de Desencadenador)

## Funciones del Sistema

### Built-In Functions (Funciones Integradas)

**Scalar Functions**  
(Funciones Escalares)

**Aggregate Functions**  
(Funciones de Agregado)

**Window Functions**  
(Funciones de Ventana)

**Rowset Functions**  
(Funciones de Conjuntos de Filas)

## Funciones Escalares(Scalar Functions)

## Funciones del Sistema

### Scalar Functions (Funciones Escalares)

Las funciones escalares son las más parecidas a las funciones que se utilizan dentro de los lenguajes de programación.

**SELECT**     $col_1, col_2$   
**FROM**        $tabla_a$   
**WHERE**       $condición_a$

## Funciones del Sistema

### Scalar Functions

(Funciones Escalares)

Las funciones escalares son las más parecidas a las funciones que se utilizan dentro de los lenguajes de programación.

```
SELECT  col1, col2, function(coln)
FROM    tablaa
WHERE   function(coln)
```

SET @var = function( ... )

IF @var = function( ... )

## Funciones del Sistema

### Scalar Functions

(Funciones Escalares)

### Funciones Deterministas

Devuelven siempre el mismo resultado cada vez que se llaman con un conjunto específico de valores de entrada.

## Funciones del Sistema

### Scalar Functions

(Funciones Escalares)

### Funciones Deterministas

`LOG10(100) = 2`

`SQRT(100) = 10`

`YEAR('01-01-2024') = 2024`

`POWER(5, 2) = 25`

`MONTH('01-01-2024') = 1`

`LEFT('Base de Datos', 4) = Base`

## Funciones del Sistema

### Scalar Functions

(Funciones Escalares)

### Funciones NO Deterministas

Nos devuelven diferentes resultados cada vez que se les llama con un conjunto específico de valores de entrada

## Funciones del Sistema

### Scalar Functions

(Funciones Escalares)

### Funciones NO Deterministas

`GETDATE()`

2024-02-19 13:54:46.333

2024-02-19 14:27:42.397

`CURRENT_TIMESTAMP`

2024-02-19 13:54:46.333

2024-02-19 14:27:42.397

`NEWID()`, `NEWSEQUENTIALID()`

### Funciones del Sistema

#### Scalar Functions (Funciones Escalares)

1 Valor



Transformar los datos

1 Valor

*“No estamos generando cambios reales sobre la tabla original, solo en la salida”*

## Funciones del Sistema

### Scalar Functions

(Funciones Escalares)

### Funciones Matemáticas

ABS

COS

LOG10

SIGN

ACOS

COT

PI

SIN

ASIN

DEGREES

POWER

SQRT

ATAN

EXP

RADIANS

SQUARE

ATN2

FLOOR

RAND

TAN

CEILING

LOG

ROUND

### Funciones del Sistema

#### Scalar Functions

(Funciones Escalares)

### Funciones Lógicas

CHOOSE

IIF

COALESCE

LEAST

GREATEST

NULLIF

## Funciones del Sistema

### Scalar Functions

(Funciones Escalares)

### Funciones de Cadena

ASCII	FORMAT	PATINDEX	RTRIM	STUFF
CHAR	LEFT	QUOTENAME	SOUNDEX	SUBSTRING
CHARINDEX	LEN	REPLACE	SPACE	TRANSLATE
CONCAT	LOWER	REPLICATE	STR	TRIM
CONCAT_WS	LTRIM	REVERSE	STRING_AGG	UNICODE
DIFFERENCE	NCHAR	RIGHT	STRING_ESCAPE	UPPER

## Funciones del Sistema

### Scalar Functions

(Funciones Escalares)

### Funciones de Conversión

CAST

TRY\_CAST

CONVERT

TRY\_CONVERT

PARSE

TRY\_PARSE

## Funciones del Sistema

### Scalar Functions

(Funciones Escalares)

### Funciones de Fecha y Hora

@@DATEFIRST	DATEFROMPARTS	EOMONTH	SYSDATETIMEOFFSET
CURRENT_TIMESTAMP	DATENAME	GETDATE	SYSUTCDATETIME
CURRENT_TIMEZONE	DATEPART	GETUTCDATE	TIMEFROMPARTS
CURRENT_TIMEZONE_ID	DATETIME2FROMPARTS	ISDATE	TODATETIMEOFFSET
DATE_BUCKET	DATETIMEFROMPARTS	MONTH	YEAR
DATEADD	DATETIMEOFFSETFROMPARTS	SMALLDATETIMEFROMPARTS	
DATEDIFF	DATETRUNC	SWITCHOFFSET	
DATEDIFF_BIG	DAY	SYSDATETIME	

**SYSDATETIME**

Devuelve un valor **datetime2(7)** que contiene la fecha y hora del equipo en el que la instancia de SQL Server se está ejecutando.

**Sintaxis:**

SYSDATETIME()

**SYSDATETIMEOFFSET**

Devuelve un valor **datetimeoffset(7)** que contiene la fecha y hora del equipo en el que la instancia de SQL Server se está ejecutando. El ajuste de zona horaria está incluido.

**Sintaxis:**

SYSDATETIMEOFFSET()

**SYSUTCDATETIME**

Devuelve un valor **datetime2** que contiene la fecha y hora del equipo en el que la instancia de SQL Server se está ejecutando. La fecha y hora se devuelven como una hora universal coordinada (UTC).

**Sintaxis:**

SYSUTCDATETIME()

**CURRENT\_TIMESTAMP**

Esta función devuelve la marca de tiempo del sistema de base de datos actual como un valor **datetime** sin el desplazamiento de zona horaria de la base de datos. **CURRENT\_TIMESTAMP** deriva este valor del sistema operativo del equipo en el que se ejecuta la instancia de SQL Server.

**Sintaxis:****CURRENT\_TIMESTAMP****GETDATE**

Devuelve la marca de tiempo del sistema de base de datos actual como un valor **datetime** sin el desplazamiento de zona horaria de la base de datos. Este valor se deriva del sistema operativo del equipo donde la instancia de SQL Server se está ejecutando.

**Sintaxis:****GETDATE()****GETUTCDATE**

Devuelve la marca de tiempo del sistema de la base de datos actual como un valor **datetime**. El ajuste de zona horaria de la base de datos no está incluido. Este valor representa la hora UTC actual (Hora universal coordinada).

**Sintaxis:****GETUTCDATE()**

## DATENAME

**Sintaxis:**

**DATENAME( datepart , date )**

Devuelve una cadena de caracteres que representa el parámetro **datepart** especificado del argumento **date** especificado.

## DATEPART

**Sintaxis:**

**DATEPART ( datepart , date )**

Devuelve un entero que representa el parámetro **datepart** especificado del parámetro **date** especificado.

## DATETRUNC

**Sintaxis:**

**DATETRUNC(datepart , date )**

Devuelve una fecha de entrada truncada a un **datepart** especificado.

- **Weekday**
- **Nanosecond**
- **TZoffset**

<b>datepart</b>	<b>Abreviaturas</b>	<b>date</b>
year	yy, yyyy	smalldatetime
quarter	qq, q	datetime
month	mm, m	date
dayofyear	dy, y	time
day	dd, d	datetime2
week	wk, ww	datetimeoffset
weekday	dw	
hour	hh	
minute	mi, n	
second	ss, s	
millisecond	ms	
microsecond	mcs	
nanosecond	ns	
tzoffset	tz	
iso_week	isowk, isoww	

**DAY**

Devuelve un entero que representa el día (del mes) del argumento date especificado.

**MONTH**

Devuelve un entero que representa la parte del mes de la fecha date especificada.

**YEAR**

Devuelve un entero que representa la parte del año del parámetro date especificado.

**Sintaxis:****DAY( date )****Sintaxis:****MONTH( date )****Sintaxis:****YEAR( date )**

smalldatetime	datetime	date	time	datetime2	datetimeoffset

# Funciones que Devuelven Valores de Fecha y Hora de sus Elementos

## TIMEFROMPARTS

Devuelve un valor time para la hora especificada y con la precisión indicada.

### Sintaxis:

**TIMEFROMPARTS( hour, minute, seconds, fractions, precision )**

## DATETIMEFROMPARTS

Devuelve un valor datetime para los argumentos de fecha y hora especificados.

### Sintaxis:

**DATETIMEFROMPARTS( year , month , day , hour , minute , seconds , milliseconds )**

## DATEFROMPARTS

Devuelve un valor date que se asigna a los valores de año, mes y día especificados.

### Sintaxis:

**DATEFROMPARTS( year, month, day )**

## SMALLDATETIMEFROMPARTS

Devuelve un valor smalldatetime de la fecha y la hora especificadas.

### Sintaxis:

**SMALLDATETIMEFROMPARTS( year, month, day, hour, minute )**

## DATETIME2FROMPARTS

Devuelve un valor datetime2 para los argumentos de fecha y hora especificados. El valor devuelto tiene una precisión especificada por el argumento de precisión.

### Sintaxis:

**DATETIME2FROMPARTS( year, month, day, hour, minute, seconds, fractions, precision )**

## DATETIMEOFFSETFROMPARTS

Devuelve un valor datetimeoffset para los argumentos de fecha y hora especificados. El valor devuelto tiene una precisión especificada por el argumento precisión y un desplazamiento especificado por los argumentos de desplazamiento.

### Sintaxis:

**DATETIMEOFFSETFROMPARTS( year, month, day, hour, minute, seconds, fractions, hour\_offset, minute\_offset, precision )**

## Funciones que Devuelven Valores de Fecha y Hora de sus Elementos

TIMEFROMPARTS

**TIMEFROMPARTS( hour, minute, seconds, fractions, precision )**

DATEFROMPARTS

**DATEFROMPARTS( year, month, day )**

SMALLDATETIMEFROMPARTS

**SMALLDATETIMEFROMPARTS( year, month, day, hour, minute )**

DATETIMEFROMPARTS

**DATETIMEFROMPARTS( year , month , day , hour , minute , seconds , milliseconds )**

DATETIME2FROMPARTS

**DATETIME2FROMPARTS( year, month, day, hour, minute, seconds, fractions, precision )**

DATETIMEOFFSETFROMPARTS

**DATETIMEOFFSETFROMPARTS( year, month, day, hour, minute, seconds, fractions, hour\_offset, minute\_offset, precision )**

➤ **year**

Expresión entera que especifica un año.

➤ **month**

Expresión entera que especifica un mes.

➤ **day**

Expresión entera que especifica un día.

➤ **hour**

Expresión entera que especifica las horas.

➤ **minute**

Expresión entera que especifica los minutos.

➤ **seconds**

Expresión entera que especifica los segundos.

➤ **milliseconds**

Expresión entera que especifica los milisegundos.

➤ **fractions**

Expresión entera que especifica un valor de fracciones de segundo.

➤ **hour\_offset**

Expresión entera que especifica la parte de hora del desplazamiento de zona horaria.

➤ **minute\_offset**

Expresión entera que especifica la parte de los minutos del desplazamiento de zona horaria.

➤ **precision**

Valor literal entero que especifica la precisión del valor que va a devolver.

## Funciones que Devuelven Valores de Diferencia de Fecha y Hora

### DATEDIFF

Devuelve la diferencia entre **dos fechas**, devolviendo el resultado en la unidad especificada (año, trimestre, mes, día, semana, hora, minuto, segundo, milisegundo, etc.). Devuelve el resultado en un valor de tipo de dato **int**.

#### Sintaxis:

**DATEDIFF( datepart , startdate , enddate )**

### DATEDIFF\_BIG

Esta función es Similar a **DATEDIFF** pero devuelve el resultado en un valor de tipo de dato **bigint** para manejar diferencias muy grandes.

#### Sintaxis:

**DATEDIFF\_BIG( datepart , startdate , enddate )**

datepart	Abreviaturas	Startdate y Enddate
year	yy, yyyy	smalldatetime
quarter	qq, q	datetime
month	mm, m	date
dayofyear	dy, y	time
day	dd, d	datetime2
week	wk, ww	datetimeoffset
hour	hh	
minute	mi, n	
second	ss, s	
millisecond	ms	
microsecond	mcs	
nanosecond	ns	

## DATEADD

○ Agrega un intervalo de tiempo a una fecha(año, trimestre, mes, día, semana, hora, minuto, segundo, milisegundo, etc.).

### Sintaxis:

**DATEADD( datepart , number , date )**

○ El argumento “**datepart**” especifica la fecha a la que DATEADD agrega un número int.

El argumento “**number**” es una expresión de un tipo de dato “**int**” y su valor puede ser positivo o negativo.

datepart	Abreviaturas	date
year	yy, yyyy	smalldatetime
quarter	qq, q	datetime
month	mm, m	date
dayofyear	dy, y	time
day	dd, d	datetime2
week	wk, ww	datetimeoffset
weekday	dw	
hour	hh	
minute	mi, n	
second	ss, s	
millisecond	ms	
microsecond	mcs	
nanosecond	ns	

## EOMONTH

Devuelve el último día del mes que contiene la fecha especificada, con un desplazamiento opcional.

### Sintaxis:

**DATEADD( start\_date [ , month\_to\_add ] )**

El argumento “**start\_date**” es una expresión de fecha que especifica la fecha para la que se devuelve el último día del mes.

El argumento opcional “**month\_to\_add**” es una expresión de tipo entero que especifica el número de meses que se van a agregar o restar al argumento “**start\_date**”.

# Funciones que Establecen o Devuelven Funciones de Formato de Sesión

## SET DATEFORMAT

Establece el orden de las partes correspondientes al mes, día y año de una fecha para interpretar las cadenas de caracteres de fecha para la sesión actual.

### Sintaxis:

`SET DATEFORMAT{ format }`

`mdy, dmy, ymd, ydm, myd y dym`

## SET LANGUAGE

Establece el idioma de la sesión actual. Esto establece la opción **SET DATEFORMAT** de forma implícita.

### Sintaxis:

`SET LANGUAGE{ 'language' }`

El argumento “language” es el nombre del idioma tal como está almacenado en sys.syslanguages o el procedimiento almacenado sp\_HELPLANGUAGE.

## sp\_HELPLANGUAGE

Es un procedimiento almacenado de la Tabla **“sys.syslanguages”** que nos muestra información sobre un lenguaje alternativo determinado o sobre todos los lenguajes de SQL Server.

### Sintaxis:

`sp_HELPLANGUAGE[ 'language' ]`

En el argumento opcional “language” se debe poner el idioma alternativo para el que se va a mostrar información.

## @@LANGUAGE

Devuelve el nombre del idioma actual de la sesión.

### Sintaxis:

`@@LANGUAGE`

## Funciones del Sistema

### Scalar Functions

(Funciones Escalares)

## Funciones del Sistema

\$PARTITION	CURRENT_REQUEST_ID	GETANSINULL	NEWID
@@ERROR	CURRENT_TRANSACTION_ID	HOST_ID	NEWSEQUENTIALID
@@IDENTITY	DECOMPRESS	HOST_NAME	ROWCOUNT_BIG
@@PACK_RECEIVED	ERROR_LINE	ISNULL	SESSION_CONTEXT
@@ROWCOUNT	ERROR_MESSAGE	ISNUMERIC	SESSION_ID
@@TRANCOUNT	ERROR_NUMBER	MIN_ACTIVE_ROWVERSION	XACT_STATE
BINARY_CHECKSUM	ERROR_PROCEDURE		
CHECKSUM	ERROR_SEVERITY		
COMPRESS	ERROR_STATE		
CONNECTIONPROPERTY	FORMATMESSAGE		
CONTEXT_INFO	GET_FILESTREAM_TRANSACTION_CONTEXT		

## Funciones de Agregación

## Funciones del Sistema

### Aggregate Functions

(Funciones de Agregación)

- Son funciones especializadas que realizan cálculos sobre grupos de variables y devuelven un único resultado. A diferencia de las funciones tradicionales, las funciones de agregado trabajan sobre grupos de filas de datos. Esto permite calcular estadísticas de forma eficaz o generar información resumida a partir de un conjunto de datos.

## Funciones del Sistema

### Aggregate Functions

(Funciones de Agregación)

## Funciones de Agregación

APPROX\_COUNT\_DISTINCTAVG

GROUPING

STDEVP

AVG

GROUPING\_ID

STRING\_AGG

CHECKSUM\_AGG

MAX

SUM

COUNT

MIN

VAR

COUNT\_BIG

STDEV

VARP

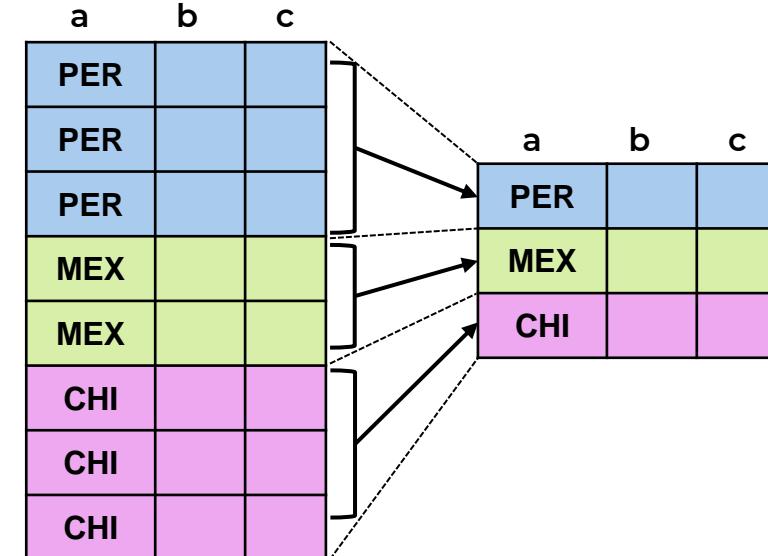
## Funciones del Sistema

Conjunto de Valores



Transformar los datos

1 Valor



*“No estamos generando cambios reales sobre la tabla original, solo en la salida”*

## Funciones del Sistema

**Aggregate Functions**  
(Funciones de Agregación)

## Cláusula GROUP BY

La Cláusula **GROUP BY** agrupa los resultados de la instrucción **SELECT** según los valores en una lista con una o varias expresiones de columna.

## Funciones del Sistema

**Aggregate Functions**

(Funciones de Agregación)

**Cláusula GROUP BY**

```
SELECT    Columna, Columnb, ... ,  
          [ AggregateFunction1(expresión), AggregateFunction2(expresión), ... ]  
FROM      TableName  
[ WHERE    Expression ]  
GROUP BY  ColumnN
```

## Funciones del Sistema

## Aggregate Functions

(Funciones de Agregación)

## Cláusula HAVING

La cláusula **HAVING** especifica una condición de búsqueda para un grupo o agregado. La cláusula **HAVING** solo se puede utilizar con la instrucción **SELECT** y **GROUP BY**. La Cláusula **HAVING** se agregó a SQL porque la Cláusula **WHERE** no se puede usar con las Funciones Agregadas.

## Funciones del Sistema

**Aggregate Functions**

(Funciones de Agregación)

**Cláusula HAVING**SELECT Column<sub>a</sub>, Column<sub>b</sub>, ... ,[ AggregateFunction<sub>1</sub>(expresión), AggregateFunction<sub>2</sub>(expresión), ... ]

FROM TableName

[ WHERE Expression ] → Filtro a la Tabla

GROUP BY Column<sub>N</sub>

HAVING AggregateFunction(expresión) → Filtro a la Función de Agregación

## Funciones de Ventana

## Funciones del Sistema

### Window Functions

(Funciones de Ventana)

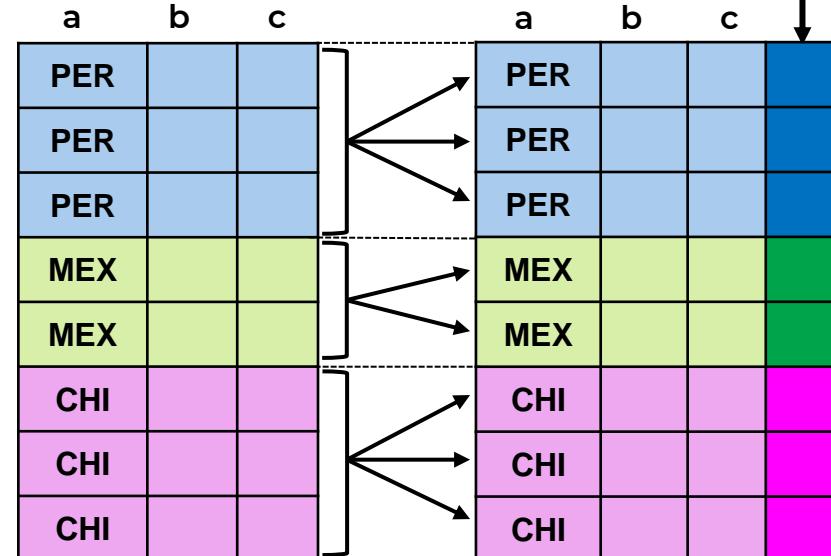
Las funciones de ventana realizan cálculos sobre un conjunto de filas relacionadas, pero a diferencia de las funciones de agregación, no agrupan los resultados en una sola fila, sino que devuelven un conjunto de valores.

Conjunto de Valores

Transformar los datos  
Cláusula OVER

Conjunto de Valores

### Funciones del Sistema



Resultado de la  
Función de Ventana

*"No estamos generando cambios reales sobre la tabla original, solo en la salida"*

### Funciones del Sistema

#### Window Functions

(Funciones de Ventana)

### Funciones de Clasificación

DENSE\_RANK

NTILE

RANK

ROW\_NUMBER

# Cláusula OVER

La cláusula OVER determina las **particiones** y el **orden** de un conjunto de filas antes de que se aplique la función de ventana asociada. Es decir, la cláusula OVER define una ventana o un conjunto de filas definido por el usuario en un conjunto de resultados de la consulta. Una función de ventana calcula entonces un valor para cada fila de la ventana. Se puede utilizar la cláusula OVER con funciones para **calcular valores agregados** tales como medias móviles, agregados acumulados, totales acumulados, etc.

Sintaxis:

```
OVER (
    [ PARTITION BY value_expression ]
    [ ORDER BY order_by_expression [ASC | DESC] ]
)
```



## Sintaxis:

```
OVER ( [ PARTITION BY value_expression ]  
      [ ORDER BY order_by_expression [ASC | DESC] ]  
    )
```

DENSE\_RANK  
NTILE  
RANK  
ROW\_NUMBER  
...

Si no se especifica **PARTITION BY**, la función trata todas las filas del conjunto de resultados de la consulta como una única partición. La función se aplicará en todas las filas de la partición si no especifica la cláusula **ORDER BY**.

value\_expression solo puede hacer referencia a las columnas disponibles mediante la cláusula FROM. value\_expression no puede hacer referencia a expresiones o alias en la lista de la selección. value\_expression puede ser una expresión de columna, una subconsulta escalar, una función escalar o una variable definida por el usuario.

## Sintaxis:

```
OVER ( [ PARTITION BY value_expression ]  
       [ ORDER BY order_by_expression [ASC | DESC] ] )
```



La cláusula **ORDER BY** dentro de la función **OVER()** se utiliza para definir el orden en el que la función de ventana procesa las filas dentro de cada partición. Le permite controlar cómo se ordenan los datos antes de que se realicen los cálculos de la función de ventana.

**ORDER BY** Especifica una columna o expresión según la cual ordenar.

**order\_by\_expression** solo puede hacer referencia a las columnas disponibles mediante la cláusula FROM. No se puede especificar un número entero para representar un nombre o alias de columna.

Si no se especifica, el orden predeterminado es **ASC** y la función de ventana utilizará todas las filas de la partición.

## Funciones del Sistema

## Window Functions

(Funciones de Ventana)

## Funciones Analíticas

CUME\_DIST

LEAD

FIRST\_VALUE

PERCENTILE\_CONT

LAG

PERCENTILE\_DISC

LAST\_VALUE

PERCENT\_RANK

Existen dos tipos de Funciones dentro de SQL Server

**Funciones del Sistema**

**Funciones Definidas por el Usuario**

## Funciones Definidas por el Usuario

Funciones  
escalares

Funciones de  
Tipo Tabla

Funciones de valor de  
tabla en línea

Funciones de valor de tabla de  
múltiples declaraciones

## Funciones Definidas por el Usuario

Funciones escalares

**Crear Objeto****Nombre Objeto**`CREATE FUNCTION Nombre(@par1 DateType, . . . ,@parN DateType)`**Tipo Objeto****Parámetro**

## Funciones Definidas por el Usuario

Funciones escalares

CREATE FUNCTION Nombre(@par1 DataType, . . . ,@parN DataType)

RETURNS [DataType | Table]

Función Escalar

Función de Tipo Tabla

## Funciones Definidas por el Usuario

Funciones escalares

```
CREATE FUNCTION Cantidad_Empieados_EstadoCivil(@MaritalStatus CHAR(1))
```

```
RETURNS INT
```

Inicio

```
AS
```

```
BEGIN
```

```
DECLARE @cantidad_empleado INT;
```

Función del Sistema

```
SET @cantidad_empleado = (  
    SELECT COUNT(1)  
    FROM HumanResources.Employee  
    WHERE MaritalStatus = @MaritalStatus  
)
```

Bloque de Ejecución

```
END;
```

## Funciones Definidas por el Usuario

Funciones escalares

```
CREATE FUNCTION Cantidad_Employees_EstadoCivil(@MaritalStatus CHAR(1))
```

```
RETURNS INT
```

Inicio  
AS  
BEGIN

```
DECLARE @cantidad_empleado INT;
```

```
SET @cantidad_empleado = (  
    SELECT COUNT(1)  
    FROM HumanResources.Employee  
    WHERE MaritalStatus = @MaritalStatus  
)
```

Bloque de  
Ejecución

```
END;
```

## Funciones Definidas por el Usuario

Funciones escalares

```
CREATE FUNCTION Cantidad_Empuestos_EstadoCivil(@MaritalStatus CHAR(1))
```

```
RETURNS INT
```

```
AS  
BEGIN
```

```
DECLARE @cantidad_empleado INT;
```

```
SET @cantidad_empleado = (  
    SELECT COUNT(1)  
    FROM HumanResources.Employee  
    WHERE MaritalStatus = @MaritalStatus  
)
```

```
RETURN @cantidad_empleado;
```

```
END;
```

Inicio

Bloque de Ejecución

## Funciones Definidas por el Usuario

Funciones escalares

```
ALTER FUNCTION Cantidad_Empieados_EstadoCivil(@MaritalStatus CHAR(1))
RETURNS INT
AS
BEGIN
    DECLARE @cantidad_empleado INT;
    SET @cantidad_empleado = (
        SELECT COUNT(1)
        FROM HumanResources.Employee
        WHERE MaritalStatus = @MaritalStatus
    )
    RETURN @cantidad_empleado;
END;
```

## Funciones Definidas por el Usuario

Funciones escalares

```
ALTER FUNCTION Cantidad_Empieados_EstadoCivil(@MaritalStatus CHAR(1))
RETURNS INT
AS
BEGIN
    DECLARE @cantidad_empleado INT;
    SELECT @cantidad_empleado = COUNT(1)
    FROM HumanResources.Employee
    WHERE MaritalStatus = @MaritalStatus
    RETURN @cantidad_empleado;
END;
```

## Funciones Definidas por el Usuario

Funciones escalares

```
CREATE OR ALTER FUNCTION Cantidad_Employees_EstadoCivil(@MaritalStatus CHAR(1))
RETURNS INT
AS
BEGIN
    DECLARE @cantidad_empleado INT;
    SELECT @cantidad_empleado = COUNT(1)
    FROM HumanResources.Employee
    WHERE MaritalStatus = @MaritalStatus
    RETURN @cantidad_empleado;
END;
```

## Funciones Definidas por el Usuario

Funciones escalares

```
CREATE OR ALTER FUNCTION Cantidad_Employees_Status(@MaritalStatus CHAR(1))
RETURNS INT
AS
BEGIN
    DECLARE @cantidad_empleado INT;

    SELECT @cantidad_empleado = COUNT(1)
    FROM HumanResources.Employee
    WHERE MaritalStatus = @MaritalStatus

    RETURN @cantidad_empleado;
END;
```

## Funciones Definidas por el Usuario

Funciones escalares

- ```
SELECT dbo.Cantidad_Employees_Status('M') AS Cantidad  
GO
```
  
- ```
DECLARE @CantidadDoble INT = 0  
SET @CantidadDoble = dbo.Cantidad_Employees_Status('M') * 2  
SELECT @CantidadDoble AS Cantidad  
GO
```

## Funciones Definidas por el Usuario

Funciones escalares

```
CREATE FUNCTION Calcular_Pitagoras (@cateto_a FLOAT, @cateto_b FLOAT)
RETURNS FLOAT
```

AS

BEGIN

```
    DECLARE @hipotenusa FLOAT;
```

```
    → SET @hipotenusa = SQRT( POWER(@cateto_a, 2) + POWER(@cateto_b, 2) )
```

```
    RETURN @hipotenusa;
```

```
END;
```

```
SELECT dbo.calcular_pitagoras(6, 8) = 10
```

## Funciones Definidas por el Usuario

Funciones de Tabla

## Funciones Definidas por el Usuario

Funciones de Tabla(En Línea)

```
CREATE FUNCTION Nombre(@par1 DateType, . . . ,@parN DateType)
```

```
RETURNS [Table]
```

Tabla



## Funciones Definidas por el Usuario

Funciones de Tabla(En Línea)

```
CREATE FUNCTION Nombre(@par1 DateType, . . . ,@parN DateType)
```

```
    RETURNS [Table]
```

Inicio ← AS

```
    RETURN(
```

```
        Instrucción SQL;
```

Bloque de Ejecución

```
);
```

## Funciones Definidas por el Usuario

Funciones de Tabla(En Línea)

```
CREATE OR ALTER FUNCTION Mostrar_Employees_HorasVacaciones(@hora SMALLINT)
```

```
    ○ RETURNS TABLE
```

```
        AS
```

```
        RETURN(
```

```
            SELECT BusinessEntityID, JobTitle, BirthDate, HireDate,
```

```
                Gender, MaritalStatus, VacationHours
```

```
            FROM HumanResources.Employee
```

```
            WHERE VacationHours <= @hora
```

```
)
```

```
GO
```

## Funciones Definidas por el Usuario

Funciones de Tabla(En Línea)

```
SELECT *
FROM Mostrar_Empleados_HorasVacaciones(50)
ORDER BY VacationHours DESC
GO
```

```
SELECT BusinessEntityID, JobTitle, HireDate, VacationHours
FROM Mostrar_Empleados_HorasVacaciones(50)
ORDER BY VacationHours DESC
GO
```

## Funciones Definidas por el Usuario

Funciones de Tabla(Múltiples Declaraciones)

CREATE FUNCTION Nombre(@par1 DataType, . . . , @parN DataType)

RETURNS @NOMBRE [Table] (col1 DataType, . . . , colN DataType)

Inicio  
AS  
BEGIN

Bloque de  
Ejecución

Insertar filas en la Tabla @NOMBRE

RETURN

END;

## Funciones Definidas por el Usuario

Funciones de Tabla(Múltiples Declaraciones)

```
CREATE OR ALTER FUNCTION Lista_Empuestos_HorasVacaciones(@hora SMALLINT)
RETURNS @Lista TABLE(BusinessEntityID INT, JobTitle NVARCHAR(50),
HireDate DATE, VacationHours SMALLINT)
AS
BEGIN
    INSERT INTO @LISTA
    SELECT BusinessEntityID, JobTitle, HireDate, VacationHours
    FROM HumanResources.Employee
    WHERE VacationHours <= @hora
    RETURN
END
GO
```

## Funciones Definidas por el Usuario

Funciones de Tabla(Múltiples Declaraciones)

```
CREATE OR ALTER FUNCTION Lista_Empuestos_HorasVacaciones(@hora SMALLINT)
```

```
RETURNS @Lista TABLE(BusinessEntityID INT, JobTitle NVARCHAR(50),  
HireDate DATE, VacationHours SMALLINT)
```

```
AS  
BEGIN
```

```
INSERT INTO @LISTA  
SELECT BusinessEntityID, JobTitle, HireDate, VacationHours  
FROM HumanResources.Employee  
WHERE VacationHours <= @hora
```

```
RETURN
```

```
END  
GO
```

## Funciones Definidas por el Usuario

Funciones de Tabla(Múltiples Declaraciones)

```
SELECT *
FROM Lista_Empuestos_HorasVacaciones(50)
ORDER BY VacationHours DESC
GO
```

```
SELECT BusinessEntityID, JobTitle, HireDate, VacationHours
FROM Lista_Empuestos_HorasVacaciones(50)
ORDER BY VacationHours DESC
GO
```

# Descripción de la Sección

Tipos de JOINS

Cláusula INNER JOIN

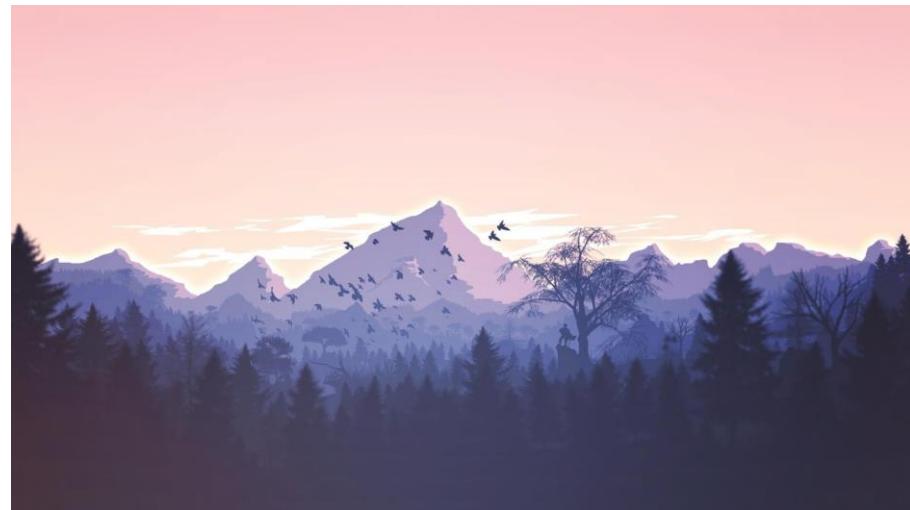
Cláusula LEFT JOIN

Cláusula RIGHT JOIN

Cláusula FULL JOIN

Cláusula CROSS JOIN

Cláusula SELF JOIN



## Tipos de JOINS

- En bases de datos relacionales, como SQL Server, Oracle, MySQL y otras, los datos se almacenan en varias tablas que están relacionadas entre sí con un valor de clave común. En consecuencia, existe una necesidad constante de extraer registros de dos o más tablas en una tabla de resultados en función de alguna condición.
- En SQL Server, esto se puede lograr fácilmente con la cláusula JOIN.

JOIN es una cláusula SQL que se utiliza para consultar y acceder a

- datos de varias tablas, basándose en relaciones lógicas entre esas tablas.

- En otras palabras, JOINS indica cómo SQL Server debe utilizar los datos de una tabla para seleccionar las filas de otra tabla..

### Todo parte del producto cartesiano

Un producto cartesiano es el resultado de una combinación de todas las filas de dos tablas, esto significa que cada fila de la primera tabla se combina con cada fila de la segunda tabla, generando un número de filas en el resultado que es el producto del número de filas de ambas tablas.

Name
Karen
Hugo
Oscar



Product
Yogurt
Agua
Dulce



Name	Product
Karen	Yogurt
Karen	Yogurt
Karen	Yogurt
Hugo	Agua
Hugo	Agua
Hugo	Agua
Oscar	Dulce
Oscar	Dulce
Oscar	Dulce

Tipos	Descripción
<b>Inner Join</b> (Join Interno)	Inicia con el producto cartesiano; aplica el filtro para que coincidan con filas entre cuadros basados en predicado.
<b>Outer Join</b> (Join Externo)	Inicia con el producto cartesiano; todas las filas de la tabla designada se conservan, igualando filas de otra tabla recuperada. Los valores NULL se adicionan como marcadores de posición.
<b>Cross Join</b> (Join Cruzada)	Combina todas las fila de ambas tablas (crea el producto cartesiano).
<b>Self Join</b> (Join consigo mismo)	Inicia con el producto cartesiano; aplica el filtro y es una técnica en la cual una tabla se une consigo misma.

## ANSI SQL-92

Las Tablas son unidas por la  
instrucción JOIN en la cláusula FROM

```
SELECT ...  
FROM Table1 <Type Join> Table2  
ON <on_predicate>
```

## Sintaxis:

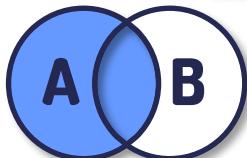
## ANSI SQL-89

Las Tablas son unidas por comas en  
la cláusula FROM

No es recomendable: produce productos  
cartesianos accidentales.

```
SELECT ...  
FROM Table1, Table2  
WHERE <where_predicate>
```

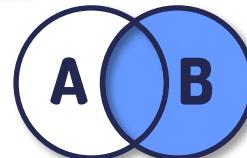
# Tipos de JOINS



```
SELECT <list> FROM  
TableA A LEFT JOIN TableB B  
ON A.KEY = B.KEY
```



```
SELECT <list> FROM  
TableA A INNER JOIN TableB B  
ON A.KEY = B.KEY
```



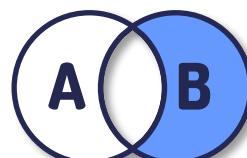
```
SELECT <list> FROM  
TableA A RIGHT JOIN TableB B  
ON A.KEY = B.KEY
```



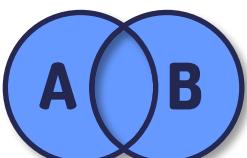
```
SELECT <list> FROM  
TableA A LEFT JOIN TableB B  
ON A.KEY = B.KEY  
WHERE B.KEY IS NULL
```



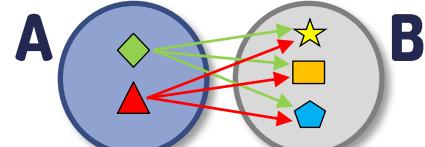
```
SELECT <list> FROM  
TableA A INNER JOIN TableB B  
ON A.KEY = B.COLUMN
```



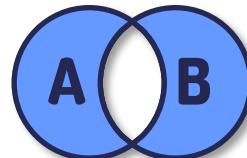
```
SELECT <list> FROM  
TableA A RIGHT JOIN TableB B  
ON A.KEY = B.KEY  
WHERE A.KEY IS NULL
```



```
SELECT <list> FROM  
TableA A FULL JOIN TableB B  
ON A.KEY = B.KEY
```

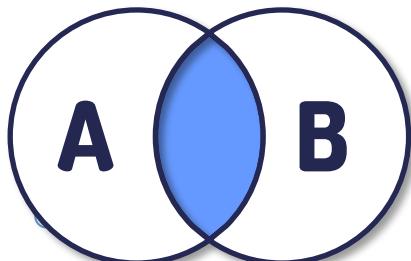


```
SELECT <list> FROM  
TableA A CROSS JOIN TableB B
```



```
SELECT <list> FROM  
TableA A FULL JOIN TableB B  
ON A.KEY = B.KEY  
WHERE A.KEY IS NULL OR B.KEY IS NULL
```

## Cláusula INNER JOIN

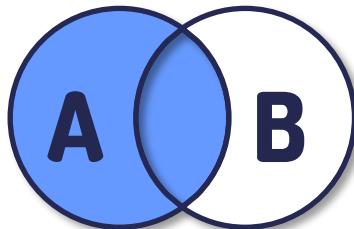


```
SELECT <list> FROM  
TableA A INNER JOIN TableB B  
ON A.KEY = B.KEY
```

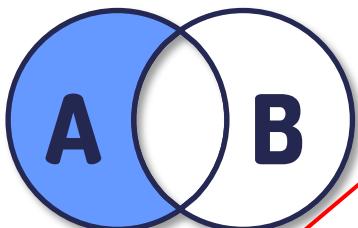
La cláusula **INNER JOIN**(Join Interno) se utiliza para combinar filas de dos o más tablas basadas en una condición que especifica una relación entre las columnas de las tablas. La cláusula **INNER JOIN** devuelve solo aquellas filas que tienen coincidencias en ambas tablas según la condición de unión especificada.



## Cláusula LEFT JOIN



```
SELECT <list> FROM  
TableA A LEFT JOIN TableB B  
ON A.KEY = B.KEY
```



```
SELECT <list> FROM  
TableA A LEFT JOIN TableB B  
ON A.KEY = B.KEY  
WHERE B.KEY IS NULL
```

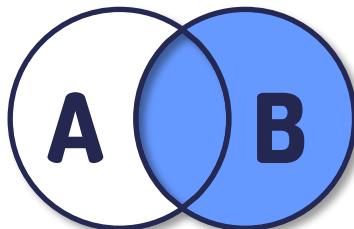
La cláusula **LEFT JOIN**(Join Externo), también conocido como **LEFT OUTER JOIN** se utiliza para combinar filas de dos tablas de manera que devuelve todas las filas de la tabla de la izquierda (la primera tabla mencionada) y las filas coincidentes de la tabla de la derecha (la segunda tabla mencionada). Si no hay ninguna coincidencia en la tabla derecha, se devuelven **NULL** para las columnas de la tabla derecha.

OPCIONAL  
`<join_type> ::=`  
[ { **INNER** | { { **LEFT** | **RIGHT** | **FULL** } [ **OUTER** ] } ]  
**JOIN**

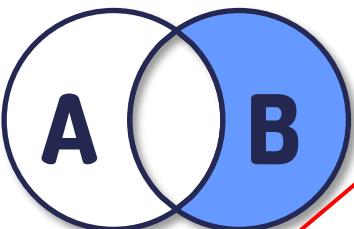
**LEFT OUTER JOIN**

**LEFT JOIN**

## Cláusula RIGHT JOIN



```
SELECT <list> FROM  
TableA A RIGHT JOIN TableB B  
ON A.KEY = B.KEY
```



```
SELECT <list> FROM  
TableA A RIGHT JOIN TableB B  
ON A.KEY = B.KEY  
WHERE A.KEY IS NULL
```

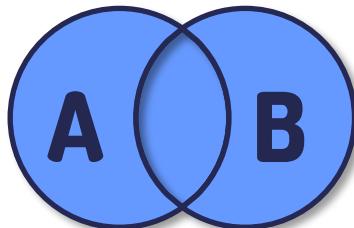
La cláusula **RIGHT JOIN**(Join Externo), también conocido como **RIGHT OUTER JOIN** se utiliza para combinar filas de dos tablas de manera que devuelve todas las filas de la tabla de la derecha (la segunda tabla mencionada) y las filas coincidentes de la tabla de la izquierda (la primera tabla mencionada). Si no hay ninguna coincidencia en la tabla izquierda, se devuelven **NULL** para las columnas de la tabla izquierda.

**OPCIONAL**  
**<join\_type>** ::=  
[ { **INNER** | { { **LEFT** | **RIGHT** | **FULL** } [ **OUTER** ] } ]  
**JOIN**

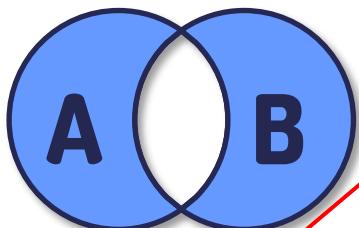
**RIGHT OUTER JOIN**

**RIGHT JOIN**

## Cláusula FULL JOIN



```
SELECT <list> FROM  
TableA A FULL JOIN TableB B  
ON A.KEY = B.KEY
```



```
SELECT <list> FROM  
TableA A FULL JOIN TableB B  
ON A.KEY = B.KEY  
WHERE A.KEY IS NULL OR B.KEY IS NULL
```

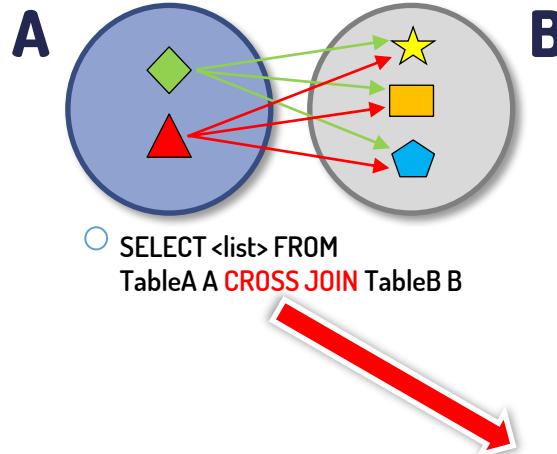
La cláusula **FULL JOIN**(Join Externo), también conocido como **FULL OUTER JOIN** se utiliza para combinar filas de dos tablas de manera que devuelve todas las filas cuando hay una coincidencia en una de las tablas. El resultado incluirá todas las filas de ambas tablas y, cuando no haya coincidencia, las columnas de la tabla faltante tendrán valores **NULL**.

OPCIONAL  
<join\_type> ::=  
[ { INNER | { { LEFT | RIGHT | FULL } [ OUTER ] } } ]  
JOIN

**FULL OUTER JOIN**

**FULL JOIN**

## Cláusula CROSS JOIN



La cláusula **CROSS JOIN**(Join Cruzado), es una operación que combina cada fila de una tabla con cada fila de otra tabla, generando un producto cartesiano.

El resultado de un **CROSS JOIN** incluye todas las combinaciones posibles de filas entre las dos tablas involucradas.

### Sintaxis:

```
SELECT column1, column2, columnN  
FROM TableA CROSS JOIN TableB
```

Algo importante a tener en cuenta es que la cláusula **CROSS JOIN** no se utiliza mucho en un ambiente real o de producción, es raro su utilización y se suele utilizar para testear datos.

## Cláusula SELF JOIN



```
SELECT <list> FROM  
TableA A INNER JOIN TableA B  
ON A.KEY = B.COLUMN
```



## Sintaxis:

```
SELECT column1, column2, columnN  
FROM TableA [INNER | LEFT | RIGHT] TableB  
ON TableA.Key = B.Column
```

La cláusula **SELF JOIN(Auto Unión)** en SQL Server no existe como tal, sino es una técnica en la que una tabla se une consigo misma. Esto se utiliza cuando se desea comparar filas dentro de la misma tabla o cuando los datos en una sola tabla necesitan ser relacionados entre sí de alguna manera. Para realizar un **SELF JOIN**, se usan alias de tabla para diferenciar las diferentes instancias de la misma tabla.

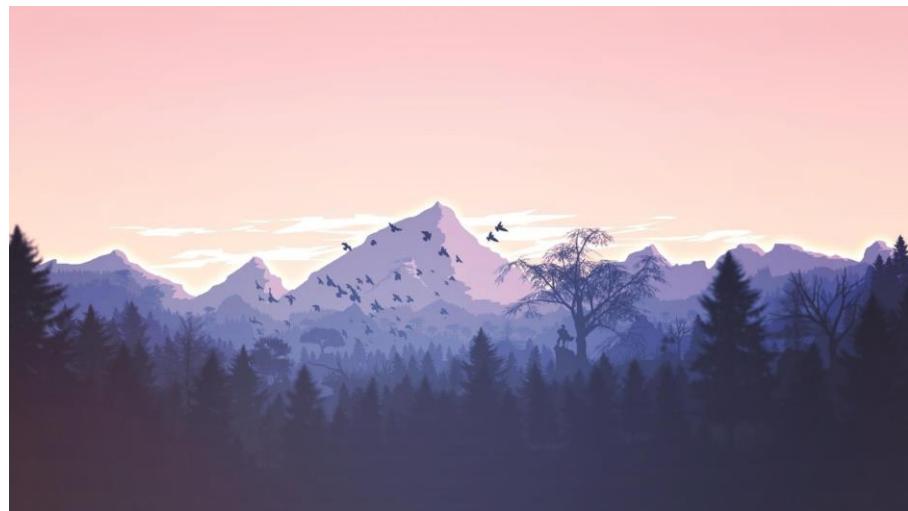
Un caso típico de uso de **SELF JOIN** es en una tabla de “empleados” donde cada “empleado” tiene un campo de identificación de su “gerente”. Al usar un **SELF JOIN**, se pueden relacionar los empleados con sus gerentes para obtener una jerarquía de informes.

# Descripción de la Sección

Descripción de Subconsultas

Subconsultas Escalares

Subconsultas Multivalor



# Subconsultas

# Subconsultas

# Sub consultas

- Que algo depende algo
- Que algo esta debajo de algo
- Que algo esta dentro de algo

La consulta es una instrucción escrita en el lenguaje SQL.

Una subconsulta es una consulta anidada dentro de otra consulta SQL.

Los resultados de la consulta interna pasan a la consulta externa y la consulta interna actúa como una expresión desde la perspectiva de la consulta externa.

Las subconsultas pueden devolver un único valor, una lista de valores o un conjunto completo de resultados y se utilizan en diversas cláusulas de la consulta principal, como **SELECT**, **FROM**, **WHERE**, **HAVING** y **otras**. Son útiles para realizar consultas más complejas y específicas, permitiendo la reutilización de resultados de una consulta en otra.



Las subconsultas pueden ser autónomas o correlacionadas

- Las consultas autónomas se ejecutan por separado
- Las consultas correlacionadas dependen de los valores de la consulta externa.

Existen dos tipos principales de subconsultas:

1. Subconsultas Escalares: Devuelven un solo valor y suelen usarse en cláusulas **SELECT** o **WHERE**.
2. Subconsultas Multivalor: Devuelven múltiples filas o columnas y pueden utilizarse con operadores como **IN**, **ALL**, **EXISTS**, entre otros.



## Subconsultas Escalares

Las subconsultas escalares en SQL Server son subconsultas que devuelven un único valor (un solo dato) como resultado. Este valor puede ser un número, una cadena, una fecha, etc.

Las subconsultas escalares se utilizan comúnmente en las cláusulas **SELECT**, **WHERE**, o en cualquier lugar donde se espera un valor simple.

Una subconsulta escalar está encerrada entre paréntesis y se ejecuta una vez para cada fila de la consulta principal. Debido a que devuelven un único valor, pueden utilizarse en expresiones de asignación, comparaciones, o en combinaciones con operadores aritméticos y lógicos.

Si la consulta interna devuelve un conjunto vacío, el resultado se convierte en **NULL**.

## Subconsulta Escalar en la Cláusula SELECT

```
SELECT NombreProducto, Precio,  
       (SELECT AVG(Precio) FROM Productos) AS PrecioPromedio  
FROM Productos;
```

## Subconsulta Escalar en la Cláusula WHERE

```
SELECT NombreEmpleado, Salario  
FROM Empleados  
WHERE Salario > (SELECT AVG(Salario) FROM Empleados);
```

## Subconsulta Escalar en Expresiones de Asignación

```
DECLARE @SalarioPromedio DECIMAL(10, 2);  
SET @SalarioPromedio = (SELECT AVG(Salario) FROM Empleados);
```

## Subconsultas Multivalor

Las subconsultas multivalor en SQL Server son subconsultas que pueden devolver múltiples filas y/o columnas como resultado. A diferencia de las subconsultas escalares, que devuelven un solo valor, las subconsultas multivalor pueden proporcionar una lista de valores o un conjunto de resultados completo.

Las subconsultas multivalor se utilizan en diferentes contextos dentro de una consulta principal, como en las cláusulas **WHERE**, **HAVING**, y **FROM**. Se usan con operadores que pueden manejar conjuntos de resultados, como **IN**, **EXISTS**, **ALL**, y también pueden formar parte de una lista de selección en consultas anidadas más complejas.

## Subconsulta Multivalor con IN

```
SELECT NombreProducto  
FROM Productos  
WHERE ProductID IN (SELECT ProductID FROM Ventas);
```

## Subconsulta Multivalor con EXISTS

```
SELECT NombreDepartamento  
FROM Departamentos d  
WHERE EXISTS (SELECT 1 FROM Empleados e WHERE e.DepartamentoID = d.DepartamentoID);
```

## Subconsulta Multivalor en la cláusula FROM

```
SELECT AVG(Precio)  
FROM (SELECT Precio FROM Productos WHERE Categoria = 'Electrónica') AS PreciosElectronica;
```

## ¿Qué ventajas tiene utilizar Subconsultas?

- No alterar el Script ya construido.
- Permite desarrollar lógica para solucionar problemas.
- Las Subconsultas es fundamental para las Vistas.
- Forma parte del aprendizaje de SQL Server.

# Descripción de la Sección

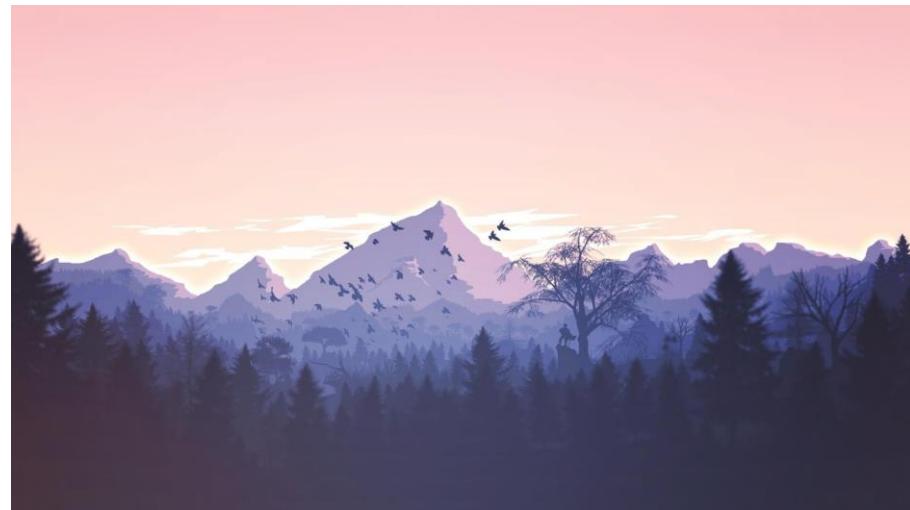
Objetivo de la Sección

Funciones de Metadato

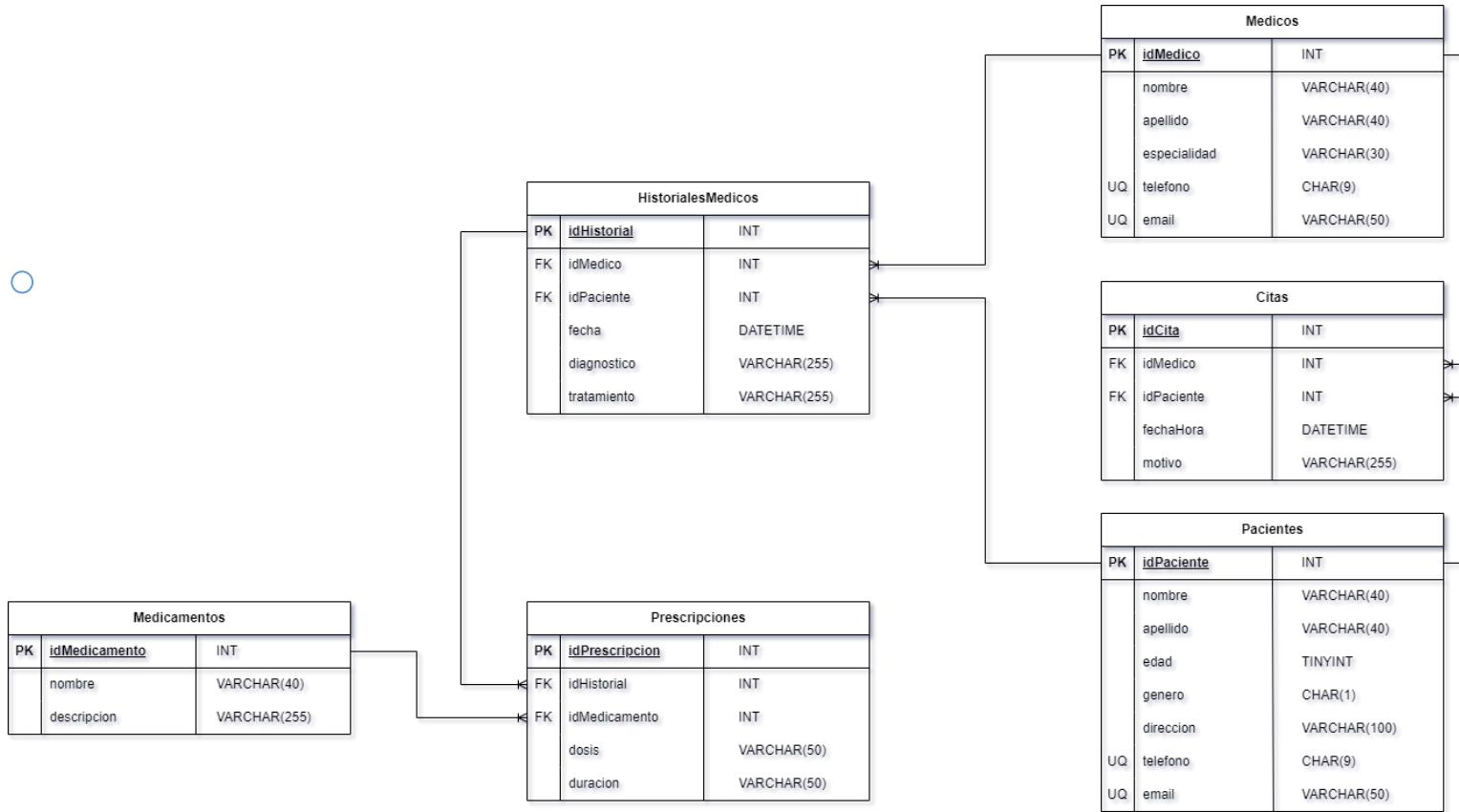
Restricciones

Implementar el Modelo Relacional

Pruebas de Implementación



# Objetivo de la Sección



## Funciones de Metadatos

En SQL Server, las funciones de metadatos son funciones integradas que proporcionan información sobre la base de datos y sus objetos.



Estas funciones permiten a los administradores de bases de datos y a los desarrolladores consultar propiedades y características de la base de datos, sus tablas, columnas, índices, procedimientos almacenados, vistas y otros objetos. Son herramientas esenciales para gestionar y mantener la base de datos de manera eficiente.



## Funciones del Sistema

## Aggregate Functions

(Funciones de Agregación)

## Funciones de Metadatos

@@PROCID	DB_NAME	INDEXKEY_PROPERTY	SCHEMA_ID
APP_NAME	FILE_ID	INDEXPROPERTY	SCHEMA_NAME
APPLOCK_MODE	FILE_INDEX	NEXT VALUE FOR	SCOPE_IDENTITY
APPLOCK_TEST	FILE_NAME	OBJECT_DEFINITION	SERVERPROPERTY
ASSEMBLYPROPERTY	FILEGROUP_ID	OBJECT_ID	STATS_DATE
COL_LENGTH	FILEGROUP_NAME	OBJECT_NAME	TYPE_ID
COL_NAME	FILEGROUPPROPERTY	OBJECT_SCHEMA_NAME	TYPE_NAME
COLUMNPROPERTY	FILEPROPERTY	OBJECTPROPERTY	TYPEPROPERTY
DATABASE_PRINCIPAL_ID	FULLTEXTCATALOGPROPERTY	OBJECTPROPERTYEX	VERSION
DATABASEPROPERTYEX	FULLTEXTSERVICEPROPERTY	ORIGINAL_DB_NAME	
DB_ID	INDEX_COL	PARSENAME	

## Tipos Comunes de Funciones de Metadatos

## 1. INFORMACIÓN SOBRE BASES DE DATOS

**DB\_NAME()**: Devuelve el nombre de la base de datos.

```
SELECT DB_NAME();
```

**DATABASEPROPERTYEX()**: Devuelve propiedades de la base de datos.

```
SELECT DATABASEPROPERTYEX('BaseDeDatosEjemplo', 'Status');
```

## 2. INFORMACIÓN SOBRE OBJETOS

**OBJECT\_ID()**: Devuelve el identificador de un objeto.

```
SELECT OBJECT_ID('dbo.TablaEjemplo');
```

**OBJECT\_NAME()**: Devuelve el nombre de un objeto dado su ID.

```
SELECT OBJECT_NAME(OBJECT_ID('dbo.TablaEjemplo'));
```

## Tipos Comunes de Funciones de Metadatos

## 3. INFORMACIÓN SOBRE COLUMNAS

COLUMNPROPERTY(): Devuelve propiedades de una columna específica.

```
SELECT COLUMNPROPERTY(OBJECT_ID('dbo.TablaEjemplo'), 'ColumnaEjemplo', 'Precision');
```

## 4. INFORMACIÓN SOBRE ÍNDICES

INDEXPROPERTY(): Devuelve propiedades de un índice.

```
SELECT INDEXPROPERTY(OBJECT_ID('dbo.TablaEjemplo'), 'IndiceEjemplo', 'IsUnique');
```

## 5. INFORMACIÓN SOBRE ARCHIVOS

FILEPROPERTY(): Devuelve propiedades de un archivo.

```
SELECT FILEPROPERTY('NombreArchivo', 'IsReadOnly');
```

## 6. INFORMACIÓN SOBRE EL SERVIDOR

SERVERPROPERTY(): Devuelve propiedades de la instancia del servidor.

```
SELECT SERVERPROPERTY('ProductVersion');
```

## Beneficios:

- Automatización
- Administración
- Auditoría

## Beneficios:

Las funciones de metadatos son herramientas poderosas para quienes gestionan y desarrollan bases de datos, proporcionando información detallada y específica de una manera fácil y directa.

## Restricciones

# Restricciones

Son reglas y restricciones predefinidas que se aplican en una sola columna o en varias columnas, relacionados a los valores permitidos en las columnas, para mantener la integridad, precisión y confiabilidad de los datos de esa columna.

- Si los datos insertados cumplen con la regla de restricción, se insertarán con éxito.
- Si los datos insertados violan la restricción definida, la operación de inserción se cancelará.

Las restricciones se pueden considerar a nivel de columna, donde se especifica como parte de la definición de columna y se aplicarán solo a esa columna.

```
CREATE TABLE NOMBRE_TABLA  
(  
    colum1 ... ,  
    colum1 ... ,  
    colum1 ... ,  
    colum1 ...  
);
```

```
ALTER TABLE NOMBRE_TABLA ...
```

Existen 6 tipos de Restricciones que podemos utilizar, las cuales son:

**NOT NULL**

**UNIQUE**

**PRIMARY KEY**

**FOREIGN KEY**

**CHECK**

**DEFAULT**

## NOT NULL

Por defecto, las columnas pueden contener valores NULL. Se

- usa una restricción NOT NULL para evitar insertar valores NULL en la columna especificada, considerándolo entonces como un valor no aceptado para esa columna. Esto significa que debe proporcionar un valor válido NO NULO, ya que la columna siempre contendrá datos.

## NOT NULL

PROFESORES		
	dni	INT
	nombre	VARCHAR(40)
	paterno	VARCHAR(20)
	materno	VARCHAR(20)
	email	VARCHAR(40)
	edad	TINYINT

```
CREATE TABLE PROFESORES
```

```
(
```

```
    dni      INT      NOT NULL,  
    nombre   VARCHAR(40) NOT NULL,  
    paterno  VARCHAR(20) NOT NULL,  
    materno  VARCHAR(20) NOT NULL,  
    email    VARCHAR(40) NOT NULL,  
    edad     TINYINT
```

```
);
```

# Restricciones

## NOT NULL

PROFESORES	
dni	INT
nombre	VARCHAR(40)
paterno	VARCHAR(20)
materno	VARCHAR(20)
email	VARCHAR(40)
edad	TINYINT

```
CREATE TABLE PROFESORES
```

```
(
```

```
    dni      INT      NOT NULL,  
    nombre   VARCHAR(40) NOT NULL,  
    paterno  VARCHAR(20) NOT NULL,  
    materno  VARCHAR(20) NOT NULL,  
    email    VARCHAR(40) NOT NULL,  
    edad     TINYINT
```

```
);
```

dni	nombre	paterno	materno	email	edad
99999999	Fernanda	Torres	Campos	f.torres@example.com	30

# Restricciones

## NOT NULL

PROFESORES	
	INT
nombre	VARCHAR(40)
paterno	VARCHAR(20)
materno	VARCHAR(20)
email	VARCHAR(40)
edad	TINYINT

```
CREATE TABLE PROFESORES
```

```
(
```

```
    dni      INT      NOT NULL,  
    nombre   VARCHAR(40) NOT NULL,  
    paterno  VARCHAR(20) NOT NULL,  
    materno  VARCHAR(20) NOT NULL,  
    email    VARCHAR(40) NOT NULL,  
    edad     TINYINT
```

```
);
```

dni	nombre	paterno	materno	email	edad
99999999	Fernanda	Torres	Campos	f.torres@example.com	30
88888888	Jose	Perez	Garcia	j.perez@example.com	NULL
...	...	...	...	...	...

**UNIQUE**

La restricción UNIQUE se utiliza para garantizar que no se inserten valores duplicados en una columna específica o combinación de columnas que participen en la restricción UNIQUE y no formen parte de la CLAVE PRIMARIA.

# Restricciones

## UNIQUE

PROFESORES	
	INT
dni	VARCHAR(40)
nombre	VARCHAR(20)
paterno	VARCHAR(20)
materno	VARCHAR(20)
email	VARCHAR(40)
edad	TINYINT

```
CREATE TABLE PROFESORES
```

```
(  
    dni      INT      NOT NULL,  
    nombre   VARCHAR(40) NOT NULL,  
    paterno  VARCHAR(20) NOT NULL,  
    materno  VARCHAR(20) NOT NULL,  
    email    VARCHAR(40) UNIQUE NOT NULL,  
    edad     TINYINT  
)
```

dni	nombre	paterno	materno	email	edad
99999999	Fernanda	Torres	Campos	f.torres@example.com	30
88888888	Jose	Perez	Garcia	j.perez@example.com	NULL
...	...	...	...	...	...

# Restricciones

## UNIQUE

PROFESORES	
	INT
dni	VARCHAR(40)
nombre	VARCHAR(20)
paterno	VARCHAR(20)
materno	VARCHAR(20)
email	VARCHAR(40)
edad	TINYINT

```
CREATE TABLE PROFESORES
```

```
(  
    dni      INT      NOT NULL,  
    nombre   VARCHAR(40) NOT NULL,  
    paterno  VARCHAR(20) NOT NULL,  
    materno  VARCHAR(20) NOT NULL,  
    email    VARCHAR(40) UNIQUE NOT NULL,  
    edad     TINYINT  
)
```

dni	nombre	paterno	materno	email	edad
99999999	Fernanda	Torres	Campos	f.torres@example.com	30
88888888	Jose	Perez	Garcia	j.perez@example.com	NULL
77777777	Fatima	Torres	Gallegos	f.torres@example.com	NULL
...	...	...	...	...	...

## PRIMARY KEY

La restricción PRIMARY KEY consta de una columna o varias columnas con valores que identifican de forma única cada fila de la tabla.

La restricción PRIMARY KEY, combina restricciones **UNIQUE** y **NOT NULL**, donde la columna o el conjunto de columnas que participan en PRIMARY KEY no pueden aceptar el valor NULL. Si la PRIMARY KEY(CLAVE PRIMARIA) se define en varias columnas, entonces se puede insertar valores duplicados en cada columna individualmente, pero es importante mencionar que los valores de combinación de todas las columnas de PRIMARY KEY(CLAVE PRIMARIA) deben ser únicos.

Además de proporcionar un acceso rápido a los datos de la tabla, se crea automáticamente el **ÍNDICE** al definir la CLAVE PRIMARIA

# Restricciones

## PRIMARY KEY

PROFESORES		
PK	dni	INT
	nombre	VARCHAR(40)
	paterno	VARCHAR(20)
	materno	VARCHAR(20)
	email	VARCHAR(40)
	edad	TINYINT

```
CREATE TABLE PROFESORES
(
    dni      INT PRIMARY KEY,
    nombre   VARCHAR(40)      NOT NULL,
    paterno  VARCHAR(20)      NOT NULL,
    materno  VARCHAR(20)      NOT NULL,
    email    VARCHAR(40) UNIQUE NOT NULL,
    edad     TINYINT
);
```

dni	nombre	paterno	materno	email	edad
99999999	Fernanda	Torres	Campos	f.torres@example.com	30
88888888	Jose	Perez	Garcia	j.perez@example.com	NULL
...	...	...	...	...	...

## PRIMARY KEY

PROFESORES		
PK	dni	INT
	nombre	VARCHAR(40)
	paterno	VARCHAR(20)
	materno	VARCHAR(20)
	email	VARCHAR(40)
	edad	TINYINT

```
CREATE TABLE PROFESORES
(
    dni      INT PRIMARY KEY,
    nombre   VARCHAR(40)      NOT NULL,
    paterno  VARCHAR(20)      NOT NULL,
    materno  VARCHAR(20)      NOT NULL,
    email    VARCHAR(40) UNIQUE NOT NULL,
    edad     TINYINT
);
```

dni	nombre	paterno	materno	email	edad
99999999	Fernanda	Torres	Campos	f.torres@example.com	30
88888888	Jose	Perez	Garcia	j.perez@example.com	NULL
	Fatima	Torres	Gallegos	fa.torres@example.com	
...	...	...	...	...	...

## PRIMARY KEY

PROFESORES		
PK	dni	INT
	nombre	VARCHAR(40)
	paterno	VARCHAR(20)
	materno	VARCHAR(20)
	email	VARCHAR(40)
	edad	TINYINT

CREATE TABLE PROFESORES

```
(      dni INT PRIMARY KEY,  
        nombre VARCHAR(40)      NOT NULL,  
        paterno VARCHAR(20)      NOT NULL,  
        materno VARCHAR(20)      NOT NULL,  
        email  VARCHAR(40) UNIQUE NOT NULL,  
        edad   TINYINT           );
```

dni	nombre	paterno	materno	email	edad
99999999	Fernanda	Torres	Campos	f.torres@example.com	30
88888888	Jose	Perez	Garcia	j.perez@example.com	NULL
99999999	Fatima	Torres	Gallegos	fa.torres@example.com	
...	...	...	...	...	...

## FOREIGN KEY

Una "*Llave Externa(Foreign Key)*" es una clave de base de datos que se utiliza para vincular dos tablas. La restricción **FOREIGN KEY** identifica las relaciones entre las tablas de la base de datos haciendo referencia a una columna, o conjunto de columnas en la "**TABLA HIJA**" que contiene la clave externa, a la columna PRIMARY KEY o conjunto de columnas, en la "**TABLA PADRE**".

## FOREIGN KEY

PROFESORES		
PK	dni	INT
	nombre	VARCHAR(40)
	paterno	VARCHAR(20)
	materno	VARCHAR(20)
	email	VARCHAR(40)
	edad	TINYINT

Tabla Padre

CURSOS		
PK	cod_cur	SMALLINT
	dni	INT
	nombre	VARCHAR(40)

Tabla Hija

De esta manera, la restricción **FOREIGN KEY**, en la “**TABLA HIJA**” que hace referencia a la PRIMARY KEY en la “**TABLA PADRE**”, impondrá la integridad referencial de la base de datos.

**PRIMARY KEY** garantiza que no se insertarán **valores NULL o duplicados**

**PRIMARY KEY**  
+  
**FOREIGN KEY**

**INTEGRIDAD  
DE LA CLAVE**

## FOREIGN KEY

PROFESORES		
PK	dni	INT
	nombre	VARCHAR(40)
	paterno	VARCHAR(20)
	materno	VARCHAR(20)
	email	VARCHAR(40)
	edad	TINYINT

Tabla Padre

CURSOS		
PK	cod cur	SMALLINT
FK	dni	INT
	nombre	VARCHAR(40)

Tabla Hija

La restricción **FOREIGN KEY** se diferencia de la restricción **PRIMARY KEY** en que puede crear solo una **PRIMARY KEY** por cada tabla, con la capacidad de crear múltiples restricciones **FOREIGN KEY** en cada tabla haciendo referencia a varias **"TABLA PADRE"**.

## FOREIGN KEY

PROFESORES		
PK	dni	INT
	nombre	VARCHAR(40)
	paterno	VARCHAR(20)
	materno	VARCHAR(20)
	email	VARCHAR(40)
	edad	TINYINT

CURSOS		
PK	cod_cur	SMALLINT
FK	dni	INT
	nombre	VARCHAR(40)

```
CREATE TABLE PROFESORES
```

```
(  
    dni      INT PRIMARY KEY ,  
    nombre   VARCHAR(40)      NOT NULL,  
    paterno  VARCHAR(20)      NOT NULL,  
    materno  VARCHAR(20)      NOT NULL,  
    email    VARCHAR(40) UNIQUE NOT NULL,  
    edad     TINYINT  
)
```

```
CREATE TABLE CURSOS
```

```
(  
    cod_cur  SMALLINT PRIMARY KEY ,  
    dni       INT FOREIGN KEY REFERENCES PROFESORES(dni),  
    nombre   VARCHAR(40) NOT NULL  
)
```

ON UPDATE  
ON DELETE

ALTER TABLE NOMBRE\_TABLA ...

## FOREIGN KEY

Las acciones admitidas que se pueden realizar al eliminar o actualizar los valores de la “TABLA PADRE” incluyen:

## NO ACTION

Cuando las cláusulas **ON UPDATE** o **ON DELETE** se establecen en **NO ACTION**, la operación de actualización o eliminación realizada en la “TABLA PADRE” fallará con un error. Están son las cláusulas por “defecto” si no se establece las cláusulas **ON UPDATE** o **ON DELETE** dentro de las instrucciones.

## CASCADE

Al establecer las cláusulas **ON UPDATE** o **ON DELETE** en **CASCADE**, la misma acción realizada en los valores referenciados de la “TABLA PADRE” se reflejará en los valores relacionados en la “TABLA HIJA”. Por ejemplo, si el valor al que se hace referencia se elimina en la “TABLA PADRE”, también se eliminan todas las filas relacionadas en la “TABLA HIJA”.

## SET NULL

Con esta opción de cláusulas **ON UPDATE** y **ON DELETE**, si los valores a los que se hace referencia en la “TABLA PADRE” se eliminan o modifican, todos los valores relacionados en la “TABLA HIJA” se establecen en valor **NULL**.

## SET DEFAULT

El uso de la opción **SET DEFAULT** de las cláusulas **ON UPDATE** y **ON DELETE** especifica que, si los valores a los que se hace referencia en la “TABLA PADRE” se actualizan o eliminan, los valores relacionados en la “TABLA HIJA” con columnas **FOREIGN KEY** se establecerán en su valor predeterminado.

## CHECK Constraint

Un **CHECK Constraint** se define en una columna o conjunto de columnas para limitar el rango de valores que se pueden insertar en estas columnas, utilizando una condición predefinida.

La condición de restricción CHECK utiliza los diferentes operadores de comparación, como **AND**, **OR**, **BETWEEN**, **IN**, **LIKE** e **IS NULL** para escribir su expresión **booleana** que devolverá **VERDADERO**, **FALSO** o **DESCONOCIDO**.

## CHECK Constraint

PROFESORES		
PK	dni	INT
	nombre	VARCHAR(40)
	paterno	VARCHAR(20)
	materno	VARCHAR(20)
	email	VARCHAR(40)
	edad	TINYINT

CURSOS		
PK	cod_cur	SMALLINT
FK	dni	INT
	nombre	VARCHAR(40)

```
CREATE TABLE PROFESORES
(
    dni      INT PRIMARY KEY ,
    nombre   VARCHAR(40)      NOT NULL,
    paterno  VARCHAR(20)      NOT NULL,
    materno  VARCHAR(20)      NOT NULL,
    email    VARCHAR(40) UNIQUE NOT NULL,
    edad     TINYINT
);
```

```
CREATE TABLE CURSOS
(
    cod_cur  SMALLINT PRIMARY KEY ,
    dni       INT FOREIGN KEY REFERENCES PROFESORES(dni)
              ON UPDATE CASCADE
              ON DELETE CASCADE,
    nombre   VARCHAR(40) NOT NULL
);
```

dni	nombre	paterno	materno	email	edad
-9999	FERNANDA	TORRES	GUILLEN	f.torres@example.com	25
...	...	...	...	...	...

## CHECK Constraint

PROFESORES		
PK	<u>dni</u>	INT
	nombre	VARCHAR(40)
	paterno	VARCHAR(20)
	materno	VARCHAR(20)
	email	VARCHAR(40)
	edad	TINYINT

CURSOS		
PK	<u>cod_cur</u>	SMALLINT
	dni	INT
	nombre	VARCHAR(40)

```
CREATE TABLE PROFESORES
(
    dni      INT PRIMARY KEY
    → CHECK(dni >= 10000000 AND dni <= 99999999),
    nombre   VARCHAR(40)      NOT NULL,
    paterno  VARCHAR(20)      NOT NULL,
    materno  VARCHAR(20)      NOT NULL,
    email    VARCHAR(40) UNIQUE NOT NULL,
    edad     TINYINT
);
```

```
CREATE TABLE CURSOS
(
    cod_cur   SMALLINT PRIMARY KEY ,
    dni       INT FOREIGN KEY REFERENCES PROFESORES(dni)
              ON UPDATE CASCADE
              ON DELETE CASCADE,
    nombre   VARCHAR(40) NOT NULL
);
```

dni	nombre	paterno	materno	email	edad
99998888	FERNANDA	TORRES	GUILLEN	f.torres@example.com	25
...	...	...	...	...	...

# Restricciones

## CHECK Constraint

PROFESORES		
PK	dni	INT
	nombre	VARCHAR(40)
	paterno	VARCHAR(20)
	materno	VARCHAR(20)
	email	VARCHAR(40)
	edad	TINYINT

```
CREATE TABLE PROFESORES
(
    dni      INT PRIMARY KEY
    CHECK(dni >= 10000000 AND dni <= 99999999),
    nombre   VARCHAR(40)          NOT NULL,
    paterno  VARCHAR(20)          NOT NULL,
    materno  VARCHAR(20)          NOT NULL,
    email    VARCHAR(40) UNIQUE NOT NULL,
    edad     TINYINT
);
```

**dni char(8)**

**Longitud(dni) = 8**

CURSOS		
PK	cod_cur	SMALLINT
FK	dni	INT
	nombre	VARCHAR(40)

	dni	nombre	paterno	materno	email	edad
	99998888	FERNANDA	TORRES	GUILLEN	f.torres@example.com	25
	...	...	...	...	...	...

# Restricciones

## CHECK Constraint

PROFESORES		
PK	<u>dni</u>	INT
	nombre	VARCHAR(40)
	paterno	VARCHAR(20)
	materno	VARCHAR(20)
	email	VARCHAR(40)
	edad	TINYINT

CURSOS		
PK	<u>cod_cur</u>	SMALLINT
FK	dni	INT
	nombre	VARCHAR(40)

```
CREATE TABLE PROFESORES
(
    dni      INT PRIMARY KEY
        CHECK(dni >= 10000000 AND dni <= 99999999),
    nombre   VARCHAR(40)      NOT NULL,
    paterno  VARCHAR(20)      NOT NULL,
    materno  VARCHAR(20)      NOT NULL,
    email    VARCHAR(40) UNIQUE NOT NULL,
    edad     TINYINT
);
```

```
CREATE TABLE CURSOS
(
    cod_cur   SMALLINT PRIMARY KEY ,
    dni       INT FOREIGN KEY REFERENCES PROFESORES(dni)
              ON UPDATE CASCADE
              ON DELETE CASCADE,
    nombre   VARCHAR(40) NOT NULL
);
```

dni	nombre	paterno	materno	email	edad
99998888	FERNANDA	TORRES	GUILLEN	f.torres@example.com	0
...	...	...	...	...	...



# Restricciones

## CHECK Constraint

PROFESORES		
PK	dni	INT
	nombre	VARCHAR(40)
	paterno	VARCHAR(20)
	materno	VARCHAR(20)
	email	VARCHAR(40)
	edad	TINYINT

CURSOS		
PK	cod_cur	SMALLINT
FK	dni	INT
	nombre	VARCHAR(40)

```
CREATE TABLE PROFESORES
(
    dni      INT PRIMARY KEY
        CHECK(dni >= 10000000 AND dni <= 99999999),
    nombre   VARCHAR(40)      NOT NULL,
    paterno  VARCHAR(20)      NOT NULL,
    materno  VARCHAR(20)      NOT NULL,
    email    VARCHAR(40) UNIQUE NOT NULL,
    edad     TINYINT          CHECK(edad >= 18 AND edad <= 70)
);
```

```
CREATE TABLE CURSOS
(
    cod_cur   SMALLINT PRIMARY KEY ,
    dni       INT FOREIGN KEY REFERENCES PROFESORES(dni)
              ON UPDATE CASCADE
              ON DELETE CASCADE,
    nombre   VARCHAR(40) NOT NULL
);
```

dni	nombre	paterno	materno	email	edad
99998888	FERNANDA	TORRES	GUILLEN	f.torres@example.com	20
...	...	...	...	...	...



# Restricciones

## CHECK Constraint

PROFESORES		
PK	dni	INT
	nombre	VARCHAR(40)
	paterno	VARCHAR(20)
	materno	VARCHAR(20)
	email	VARCHAR(40)
	edad	TINYINT

CURSOS		
PK	cod_cur	SMALLINT
FK	dni	INT
	nombre	VARCHAR(40)

```
CREATE TABLE PROFESORES
(
    dni      INT PRIMARY KEY
        CHECK(dni >= 10000000 AND dni <= 99999999),
    nombre   VARCHAR(40)      NOT NULL,
    paterno  VARCHAR(20)      NOT NULL,
    materno  VARCHAR(20)      NOT NULL,
    email    VARCHAR(40) UNIQUE NOT NULL,
    edad     TINYINT          CHECK(edad >= 18 AND edad <= 70)
);
```

```
CREATE TABLE CURSOS
(
    cod_cur   SMALLINT PRIMARY KEY ,
    dni       INT FOREIGN KEY REFERENCES PROFESORES(dni)
              ON UPDATE CASCADE
              ON DELETE CASCADE,
    nombre   VARCHAR(40) NOT NULL
);
```

dni	nombre	paterno	materno	email	edad
99998888	FERNANDA	TORRES	GUILLEN	f.torres@example.com	20
88888888	Jose	Perez	Garcia	j.perez@example.com	NULL

### CHECK Constraint

Campo

Descuento  
decimal(1,2)

Restricción

0 <= Descuento  
AND  
Descuento <= 0.25

### CHECK Constraint

#### Campo

**Fecha\_Ingreso**

**smalldatetime**

**Fecha\_Salida**

**smalldatetime**

#### Restricción

**Fecha\_Salida >  
Fecha\_Ingreso**

## DEFAULT Constraint

Se utiliza un **DEFAULT Constraint**(restricción DEFAULT) para proporcionar un valor de columna predeterminado para las filas insertadas si no se especifica ningún valor para esa columna en la instrucción **INSERT**.

La restricción Predeterminada ayuda a mantener la **integridad del dominio** al proporcionar valores adecuados para la columna, en caso de que el usuario no proporcione un valor para ella. El valor predeterminado puede ser **un valor constante, un valor de función del sistema o NULL**.

## DEFAULT Constraint

PROFESORES	
PK	dni
	nombre
	paterno
	materno
	email
	edad
	ingreso
	salida

```
CREATE TABLE PROFESORES
(
    dni      INT PRIMARY KEY
              CHECK(dni >= 10000000 AND dni <= 99999999),
    nombre   VARCHAR(40)      NOT NULL,
    paterno  VARCHAR(20)      NOT NULL,
    materno  VARCHAR(20)      NOT NULL,
    email    VARCHAR(40) UNIQUE NOT NULL,
    edad     TINYINT CHECK(edad >= 18 AND edad <= 70),
    ingreso  SMALLDATETIME DEFAULT GETDATE(),
    salida   SMALLDATETIME DEFAULT NULL
);
```

dni	nombre	paterno	materno	email	edad	ingreso	salida
99998888	FERNANDA	TORRES	GUILLEN	f.torres@example.com	25	2024-01-14 11:57:00	NULL
88888888	Jose	Perez	Garcia	j.perez@example.com	NULL	2024-01-14 12:57:00	2024-01-14 18:00:00

### DEFAULT Constraint

#### Campo

orderDate

#### Restricción

GETDATE()

city

‘LIMA’

descuento

0.15

Estas son las restricciones para garantizar la integridad de los datos y definir las relaciones entre tablas. La aplicación adecuada de restricciones es esencial para mantener la consistencia y calidad de los datos en una base de datos.

## Validación de Base de Datos

- La validación de una base de datos en SQL Server se refiere a la verificación de la existencia, integridad y consistencia de la base de datos y sus objetos (como tablas, índices, vistas, procedimientos almacenados, etc.)

Utilizar la Función de Metadato llamado “DB\_ID”, la cual nos va devolver el número de identificación de base de datos (ID) de una base de datos especificada.

`DB_ID( 'databaseName' )` → ID de la Base de Datos

`DB_ID( )` → ID de la Base de Datos que estamos usando

Consultar la Vista del sistema llamado “`DBO.SYSDATABASES`” la cual contiene información de todas las bases de datos de SQL Server.

`SELECT * FROM DBO.SYSDATABASES`

Utilizar la Función de Metadato llamado “DB\_ID”, la cual nos va devolver el número de identificación de base de datos (ID) de una base de datos especificada.

`DB_ID( 'databaseName' )` → ID de la Base de Datos

`DB_ID( )` → ID de la Base de Datos que estamos usando

Consultar la Vista del sistema llamado “`DBO.SYSDATABASES`” la cual contiene información de todas las bases de datos de SQL Server.

```
USE MASTER  
GO
```

```
SELECT * FROM DBO.SYSDATABASES
```

Utilizar la Función de Metadato llamado “DB\_ID”, la cual nos va devolver el número de identificación de base de datos (ID) de una base de datos especificada.

`DB_ID( 'databaseName' )` → ID de la Base de Datos

`DB_ID( )` → ID de la Base de Datos que estamos usando

Consultar la Vista del sistema llamado “`DBO.SYSDATABASES`” la cual contiene información de todas las bases de datos de SQL Server.

```
USE MASTER  
GO
```

```
SELECT * FROM Master.DBO.SYSDATABASES
```

```
IF DB_ID('databaseName') IS NOT NULL  
BEGIN  
    DROP DATABASE databaseName  
END
```

```
IF EXISTS( SELECT DB_ID( 'databaseName' ) )  
BEGIN  
    DROP DATABASE databaseName  
END
```

```
IF (SELECT name FROM master.dbo.sysdatabases WHERE name = 'databaseName') IS NOT NULL  
BEGIN  
    DROP DATABASE DEMO;  
END
```

```
IF EXISTS (SELECT name FROM master.dbo.sysdatabases WHERE name = 'databaseName')  
BEGIN  
    DROP DATABASE DEMO;  
END
```

**DROP DATABASE IF EXISTS databaseName**

## Validación de Tabla

La validación de tablas en SQL Server implica comprobar que las tablas existan.

utilizando la Función de Metadato llamado “**OBJECT\_ID**”, la cual nos va devolver el número de identificación del objeto de base de datos de un objeto con alcance de esquema.

```
SELECT OBJECT_ID('object_type')
```

La segunda forma es consultar la Vista del sistema llamado “**sys.tables**” la cual contiene información de todas las tablas de la Base de Datos en donde estamos ubicados.

```
SELECT * FROM sys.tables
```

```
IF OBJECT_ID('tableName') IS NOT NULL  
BEGIN  
    DROP TABLE tableName  
END
```

```
IF EXISTS( SELECT OBJECT_ID('tableName') )  
BEGIN  
    DROP TABLE tableName  
END
```

```
IF (SELECT name from sys.tables WHERE name = 'tableName') IS NOT NULL  
BEGIN  
    DROP TABLE tableName  
END
```

```
IF EXISTS (SELECT name from sys.tables WHERE name = 'tableName')  
BEGIN  
    DROP TABLE tableName  
END
```

**DROP TABLE IF EXISTS tableName**

# Descripción de la Sección

¿Qué es un Procedimiento Almacenado?

Opciones SET ANSI\_NULLS, SET QUOTED\_IDENTIFIER,  
SET NOCOUNT

Implementar un Procedimiento Almacenado

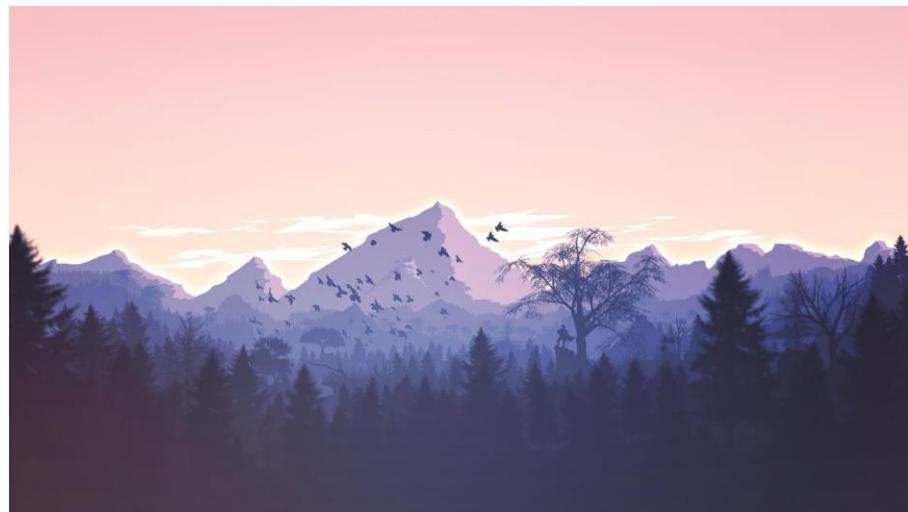
¿Qué es un Procedimiento Almacenado dinámico?

Implementar un Procedimiento Almacenado Dinámico

Parámetro OutPut en un Procedimiento Almacenado

Manejo de Errores con Procedimiento Almacenado

Procedimiento Almacenado VS Funciones



○ **¿Qué es un Procedimiento  
Almacenado(Stored Procedure)?** ○

## ¿Qué es un Procedimiento Almacenado(Stored Procedure)?

Un Procedimiento Almacenado (Stored Procedure) es un conjunto de instrucciones SQL predefinidas que se almacenan en la base de datos y se pueden ejecutar cuando sea necesario. Los procedimientos almacenados permiten encapsular lógica de negocio compleja en una única unidad que puede ser reutilizada, mejorando la eficiencia, seguridad y mantenibilidad del código.

## ¿Qué es un Procedimiento Almacenado(Stored Procedure)?

- ❖ Consultando Datos con Procedimiento Almacenado(Stored Procedure)
- ❖ Pasar Parámetros al Procedimiento Almacenado(Stored Procedure)
- ❖ Crear un Procedimiento Almacenado(Stored Procedure) Simple
- ❖ Trabajar con Procedimiento Almacenado(Stored Procedure) Dinámicos
- ❖ Ejecutar Procedimiento Almacenado(Stored Procedure)
- ❖ Examinar un Procedimiento Almacenado(Stored Procedure)
- ❖ Modificar un Procedimiento Almacenado(Stored Procedure)

## Características :

Reusabilidad

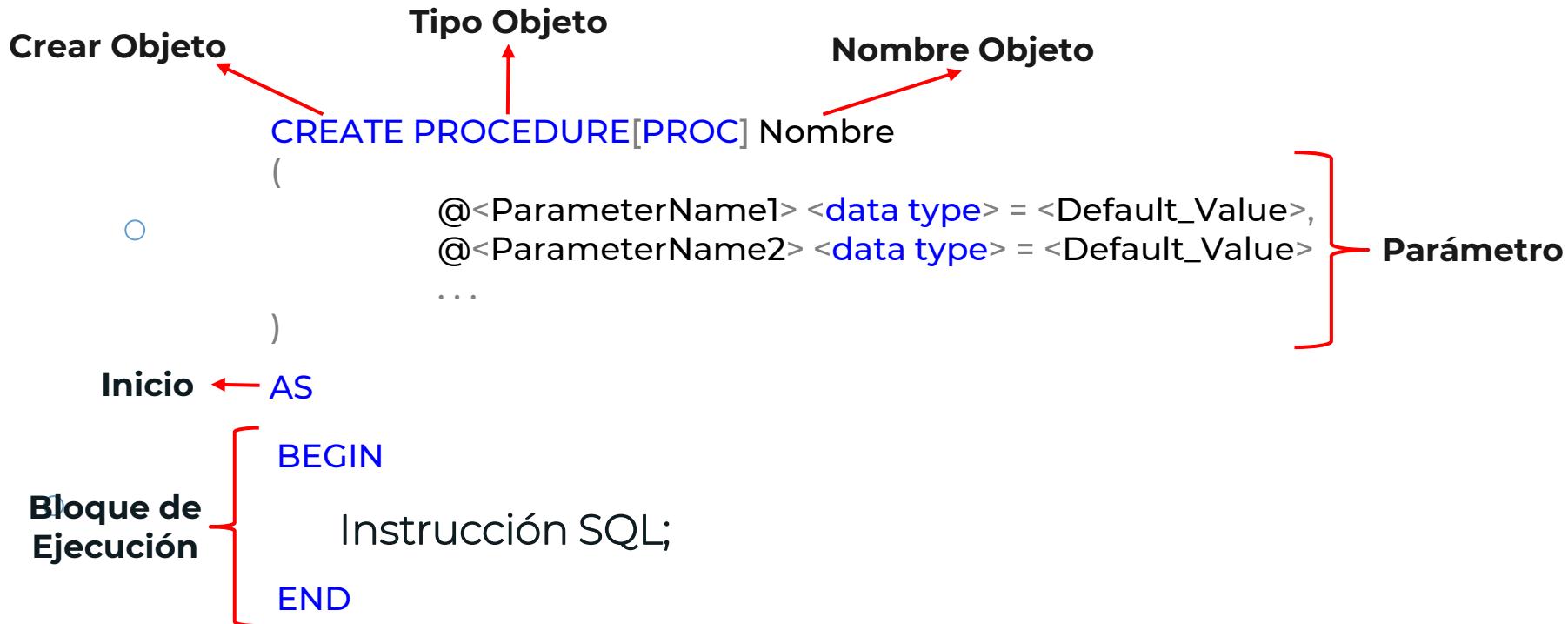
Encapsulación

Rendimiento

Seguridad

Manejo de  
parámetros

Transacciones



```
ALTER PROCEDURE[PROC] Nombre
(
    @<ParameterName1> <data type> = <Default_Value>,
    @<ParameterName2> <data type> = <Default_Value>
    ...
)
AS
BEGIN
    Instrucción SQL;
END
```

```
CREATE OR ALTER PROCEDURE[PROC] Nombre
(
    @<ParameterName1> <data type> = <Default_Value>,
    @<ParameterName2> <data type> = <Default_Value>
    ...
)
AS
BEGIN
    Instrucción SQL;
END
```

## Ejecutar un Procedimiento Almacenado

EXECUTE Nombre

GO

EXEC Nombre

GO

○ **¿Qué es un Procedimiento  
Almacenado dinámico?** ○

## ¿Qué es un Procedimiento Almacenado dinámico?

Un Procedimiento Almacenado dinámico es un procedimiento almacenado que utiliza SQL dinámico dentro de su código. El SQL dinámico implica construir y ejecutar sentencias SQL en tiempo de ejecución en lugar de predefinirlas de manera estática.

## ¿Cómo funciona?

**Construcción de la sentencia SQL:** La sentencia SQL se construye como una cadena de texto basada en parámetros o condiciones que se reciben en el procedimiento almacenado. Esta cadena puede variar en función de los valores de los parámetros, lo que hace que la consulta sea "dinámica".

**Ejecución de la sentencia SQL:** Una vez construida la cadena con la consulta SQL, se utiliza la función `EXEC` o `sp_executesql` para ejecutar la consulta.

### Características y Consideraciones:

- Flexibilidad
- Seguridad
- Rendimiento

## ¿Qué es un Procedimiento Almacenado dinámico?

Este enfoque es útil cuando necesitas construir consultas donde las columnas o las tablas involucradas pueden cambiar en tiempo de ejecución o cuando hay muchas combinaciones posibles de condiciones.

## Procedimiento Almacenado VS Funciones

Tanto los procedimientos almacenados como las funciones en SQL Server son bloques de código reutilizables que encapsulan lógica y pueden ejecutar operaciones en la base de datos.

## Procedimiento Almacenado

- Propósito y Uso

Generalmente se utiliza para realizar una secuencia de operaciones en la base de datos, como insertar, actualizar o eliminar registros.

Puede realizar tareas más complejas, como trabajar con transacciones, llamar a otros procedimientos almacenados, manejar errores, etc.



Se ejecuta con la sentencia **EXEC** o **EXECUTE**.

## Función

- Propósito y Uso

Se utiliza para realizar cálculos o manipulación de datos que devuelve un valor. Las funciones son comúnmente usadas para retornar un valor escalar o una tabla derivada.

Se puede utilizar en consultas SQL dentro de una cláusula **SELECT, WHERE, FROM**, etc.

Se invoca como parte de una expresión o consulta.

Tanto los procedimientos almacenados como las funciones en SQL Server son bloques de código reutilizables que encapsulan lógica y pueden ejecutar operaciones en la base de datos.

## Procedimiento Almacenado

- **Valor de Retorno**

Valor de RetornoNo está obligado a devolver un valor. Puede devolver un valor utilizando parámetros **OUTPUT** o mediante un código de estado con **RETURN**, pero este valor es limitado y no puede ser utilizado directamente en una consulta SQL.

- **Parámetros**

Puede aceptar parámetros de entrada y salida. Los parámetros de salida (**OUTPUT**) permiten que el procedimiento devuelva valores.

## Función

- **Valor de Retorno**

Siempre devuelve un valor. Las funciones escalares devuelven un solo valor (como un número, cadena, fecha), mientras que las funciones con valores de tabla devuelven un conjunto de resultados (tabla).

- **Parámetros**

Solo acepta parámetros de entrada. No puede tener parámetros de salida (**OUTPUT**).

Tanto los procedimientos almacenados como las funciones en SQL Server son bloques de código reutilizables que encapsulan lógica y pueden ejecutar operaciones en la base de datos.

## Procedimiento Almacenado

- Manejo de Errores

Puede manejar errores mediante bloques **TRY...CATCH**, realizar transacciones (**BEGIN TRANSACTION, COMMIT, ROLLBACK**) y llamar a otros procedimientos o funciones.

- Efectos Secundarios

Puede realizar operaciones de modificación de datos (**INSERT, UPDATE, DELETE**), cambiar el estado de la base de datos y tener efectos secundarios.

- Permisos

Los permisos para ejecutar un procedimiento almacenado son más amplios y flexibles.

## Función

- Manejo de Errores

No puede manejar transacciones ni manejar errores de la misma manera que un procedimiento almacenado. No se puede usar **TRY...CATCH** dentro de una función.

- Efectos Secundarios

No puede realizar operaciones de modificación de datos que afecten tablas o realizar operaciones que cambien el estado de la base de datos.

- Permisos

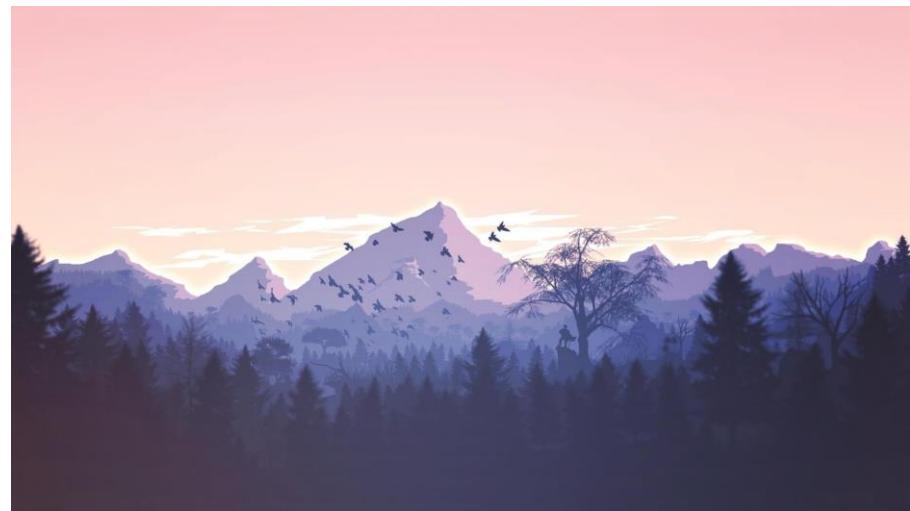
Las funciones pueden tener restricciones más estrictas dependiendo de su uso.

# Descripción de la Sección

¿Qué es una Vista(View)?

Crear una Vista(View) con T-SQL

Crear una Vista(View) con Entorno Gráfico



¿Qué es una Vista(View)?

## ¿Qué es una Vista(View)?

Una vista en SQL Server es un objeto de base de datos que actúa como una tabla virtual que contiene datos de una o varias tablas. No contiene ningún dato y no existe físicamente en la base de datos. De manera similar a una tabla SQL, el nombre de la vista debe ser único en una base de datos.

Una VISTA no requiere almacenamiento en una base de datos porque no existe físicamente. En una VISTA, también podemos controlar la seguridad del usuario para acceder a los datos de las tablas de la base de datos. Podemos permitir que los usuarios obtengan los datos de la VISTA, y el usuario no necesita permiso para cada tabla o columna para obtener datos.

La Vista se define a partir de una consulta SELECT que puede extraer datos de una o más tablas u otras vistas. Aunque una vista no almacena datos por sí misma, proporciona una manera conveniente de encapsular y simplificar consultas complejas, permitiendo que los usuarios accedan a datos filtrados o combinados de una manera más sencilla.

### Características :

Definición

No Almacena Datos

Uso de SELECT

### Ventajas :

implicidad

Seguridad

Reutilización

Abstracción

## ¿Qué es una Vista(View)?

- En resumen, una vista es una poderosa herramienta en SQL Server que permite simplificar el acceso a datos y crear abstracciones sobre la base de datos.

## ¿Qué es una Vista(View)?

Crear Objeto

Tipo Objeto

Nombre Objeto

CREATE VIEW Nombre

Inicio ← AS

Instrucción SQL;

ALTER VIEW Nombre

AS

Instrucción SQL;

CREATE OR ALTER Nombre  
AS  
Instrucción SQL;

## Ejecutar una Vista

```
SELECT * FROM Nombre
```

```
GO
```

```
SELECT Col1, Col2, ... , ColN FROM Nombre
```

```
GO
```

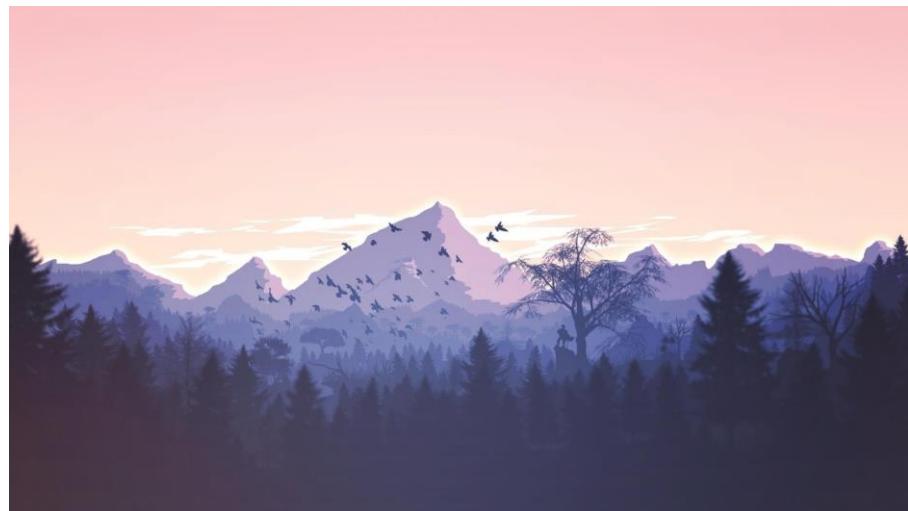
# Descripción de la Sección

Definición sobre la Sentencia Merge

Sentencia Merge de Inserción

Sentencia Merge de Actualización

Sentencia Merge de Eliminación



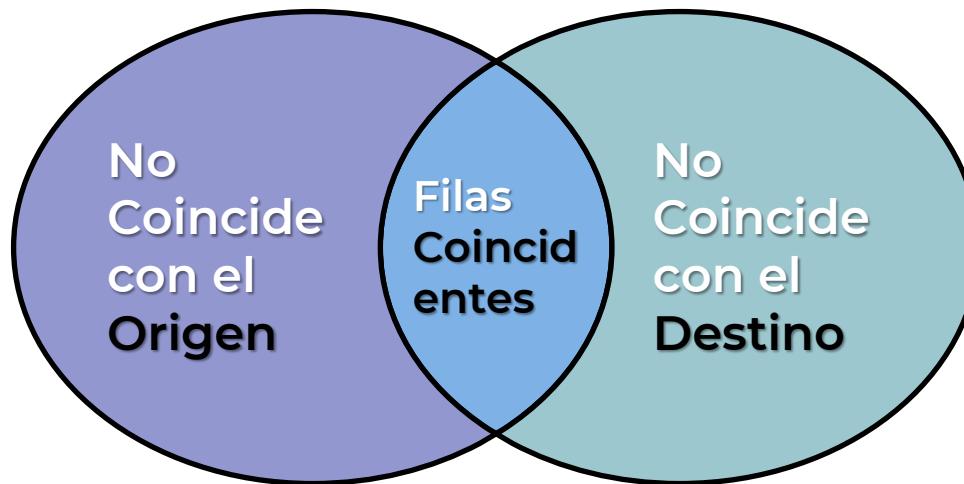
## Sentencia Merge

La sentencia MERGE en SQL es una cláusula muy popular que puede manejar inserciones, actualizaciones y eliminaciones en una sola transacción sin tener que escribir lógica separada para cada una de ellas.

- Puede especificar las condiciones en las que espera que la sentencia MERGE inserte, actualice o elimine.

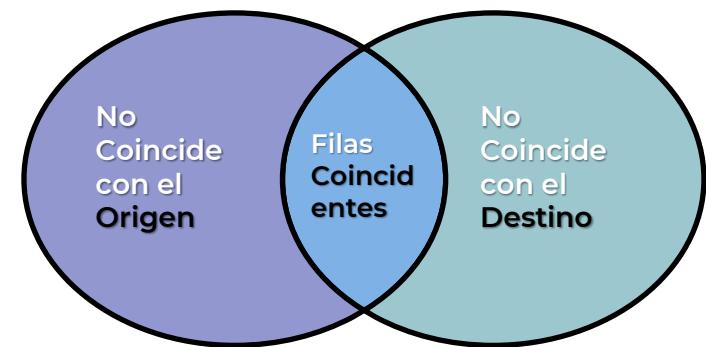
El uso de la sentencia MERGE en SQL le brinda mayor flexibilidad para personalizar sus scripts SQL complejos y también mejora la legibilidad de los mismos. La sentencia MERGE básicamente modifica una tabla existente en función del resultado de la comparación entre los campos clave con otra tabla en el contexto.

## Sentencia Merge



La sentencia MERGE en realidad combina las operaciones INSERT, UPDATE y DELETE en conjunto.

```
MERGE TargetTable AS TTable  
USING SourceTable AS STable  
ON TTable.columnKey = STable.columnKey  
[ WHEN MATCHED THEN ]  
    Statement Update Sql;  
  
[ WHEN NOT MATCHED BY TARGET THEN ]  
    Statement Insert Sql;  
  
[ WHEN NOT MATCHED BY SOURCE THEN ]  
    Statement Delete Sql;
```



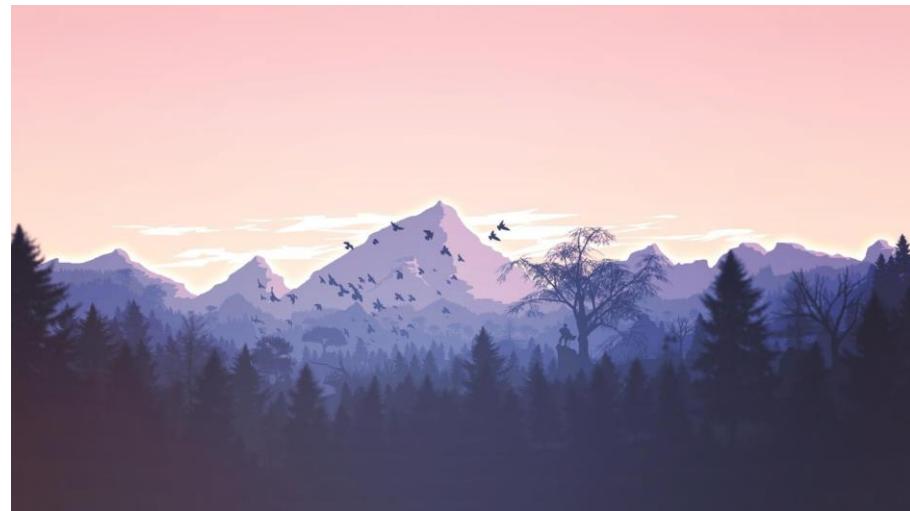
# Descripción de la Sección

¿Qué es CTE(Common Table Expression)?

CTE(Common Table Expression) Simple

CTE(Common Table Expression)  
Múltiple en una Consulta

CTE(Common Table Expression) Anidados



○ **¿Qué es CTE(Common Table Expression)?**

## ¿Qué es CTE(Common Table Expression)?

Una **CTE(Common Table Expression)**, en español **expresiones de tabla comunes** son un conjunto de resultados temporales que se devuelven mediante una única declaración para su posterior uso dentro de la misma declaración. Como se trata de un resultado temporal, no se almacena en ningún lugar y no consume espacio en disco. Sin embargo, se puede hacer referencia a él de la misma manera que a cualquier tabla.

O bien, podemos considerar una **CTE** como una especie de tabla virtual que contiene columnas con registros físicos. Esta tabla virtual se crea como resultado de la ejecución de la consulta, es utilizada por otra consulta y se elimina después de la ejecución de la consulta principal.

Desde la introducción desde la versión de **SQL Server 2005**, las **CTE** se han convertido en un método popular para los especialistas en bases de datos que las aplican para hacer que las consultas complejas sean más fáciles de leer y mantener.

### Características:

Definición Temporal

Facilita la Legibilidad

Uso Recursivo

## Sintaxis:

Comienza con la instrucción

Nombre CTE

Opcional

WITH expression\_name [ ( ColName<sub>1</sub>, ColName<sub>2</sub>, ..., ColName<sub>N</sub> ) ]

Inicio del bloque CTE

AS

Definir CTE ( CTE\_query\_definition )

Consultar

SELECT \* FROM expression\_name;

CTE

SELECT [Column1,Column2,Column3 ....] FROM expression\_name;

### Sintaxis:

#### Creación CTE

```
WITH expression_name [ ( ColName1, ColName2, ..., ColNameN ) ]  
AS  
( CTE_query_definition )
```

```
SELECT * FROM expression_name;
```

```
SELECT [Column1,Column2,Column3 ....] FROM expression_name;
```

#### Consulta CTE

## CTE Anidados

Al igual que las subconsultas, las CTE también se pueden anidar o, en palabras sencillas, podemos usar una definición de CTE dentro de otra definición de CTE.

### Sintaxis:

```
WITH expression_1 [ ( ColName1, ColName2, ..., ColNameN ) ]
```

```
AS (
```

```
    CTE_query_definition_1
```

```
)
```

```
expression_2 [ ( ColName1, ColName2, ..., ColNameN ) ]
```

```
AS(
```

```
    CTE_query_definition_2
```

```
)
```

```
SELECT * FROM expression_name;
```

```
SELECT [Column1,Column2,Column3 ....] FROM expression_name;
```

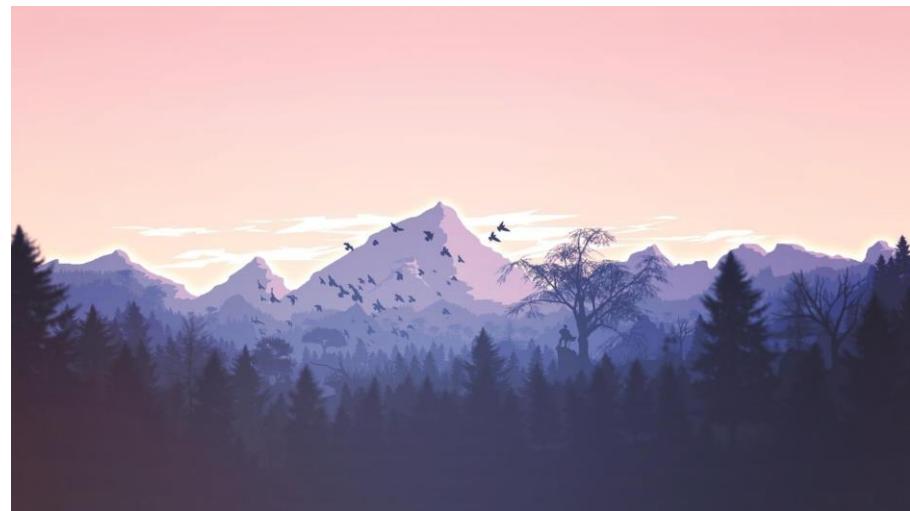
En la cláusula FROM se refiere a cte\_expression\_1

# Descripción de la Sección

Definición de los Operadores PIVOT y UNPIVOT

Implementar el Operador PIVOT

Implementar el Operador UNPIVOT





## Operadores PIVOT y UNPIVOT

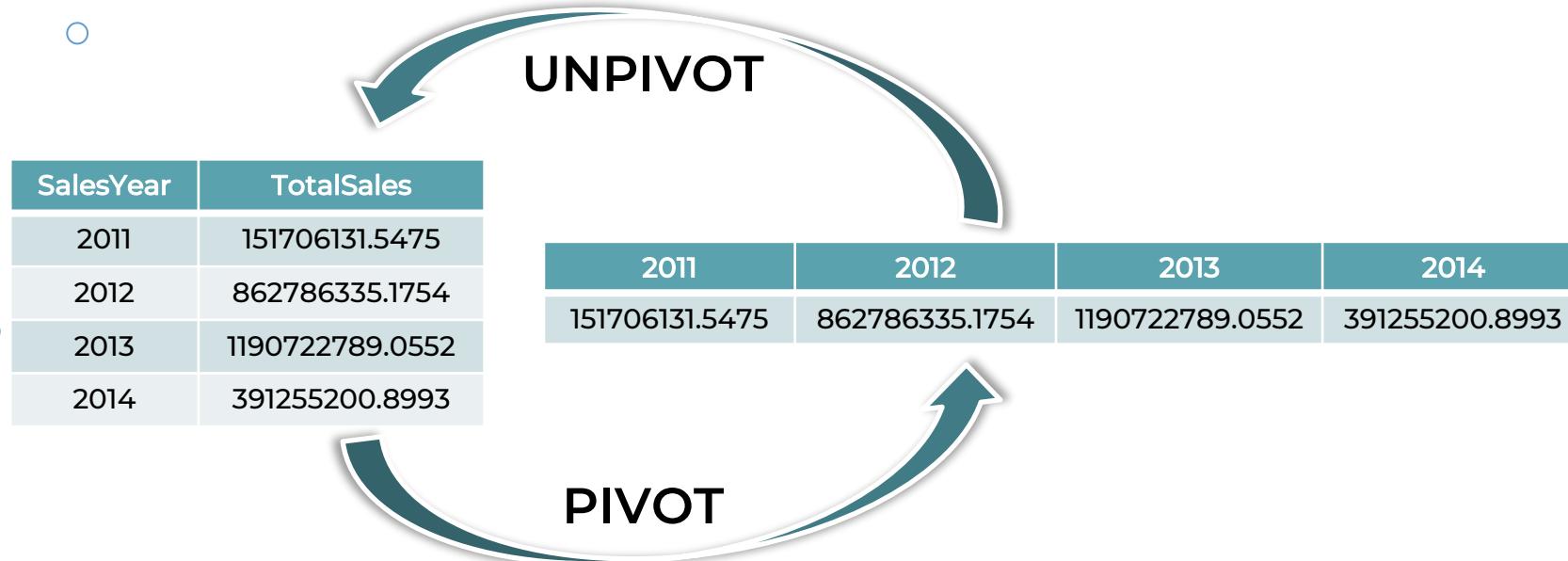
Hay varias formas de transponer e intercambiar un conjunto de datos ya sea de filas a columnas y de columnas a filas.

Podemos observar que el operador PIVOT es una de las técnicas que permite el intercambio y la transposición de filas a columnas y realiza posibles agregaciones en el camino. PIVOT y UNPIVOT son los operadores relacionales para transponer una serie de datos bidimensionales con valores de tabla en otra forma de datos.

El operador PIVOT transpone una expresión con valores de tabla de un conjunto único de valores de una columna a varias columnas en la salida y realiza agregaciones.

Además, el operador UNPIVOT realiza la operación opuesta que el operador PIVOT transformando una serie de columnas de una expresión con valores de tabla en valores de columna.

Podemos utilizar el operador PIVOT para rotar filas en una tabla convirtiendo los valores de fila en varias columnas. El siguiente diagrama ilustra lo que puede hacer PIVOT, donde tomamos 4 filas de datos y las convertimos en 1 fila con 4 columnas. Como puede ver, el proceso PIVOT convierte filas en columnas al pivotar la tabla.



## Sintaxis:

```

SELECT [ <non-pivoted column> [ AS <column name> ] , ]
      ...
      [ <first pivoted column> [ AS <column name> ] ,
        <second pivoted column> [ AS <column name> ] ,
      ...
      [ <last pivoted column> [ AS <column name> ] ]
FROM   ( <SELECT query that produces the data> )
       AS <alias>
PIVOT
(
    <aggregation function> ( <column being aggregated> )
FOR <column that contains the values
    that become column headers>
    IN ( <first pivoted column>
          , <second pivoted column>
          , ... <last pivoted column> )
) AS <alias for the pivot table>
[ <optional ORDER BY clause> ]
[ ; ]
  
```

## Sintaxis:

```

SELECT [ <non-pivoted column> [ AS <column name> ] , ]
      ...
      [ <output column for names of the pivot columns>
        [ AS <column name> ] ,
      [ <new output column created for values
        in result of the source query> [ AS <column name> ] ]
FROM   ( <SELECT query that produces the data> )
       AS <alias>
UNPIVOT
(
    <new output column created for values
    in result of the source query>
FOR <output column for names of the pivot columns>
    IN ( <first pivoted column>
          , <second pivoted column>
          , ... <last pivoted column> )
)
[ <optional ORDER BY clause> ]
[ ; ]
  
```

## ¿Qué problema de codificación resuelve PIVOT y UNPIVOT?

Tanto el operador PIVOT como el operador UNPIVOT se utilizan normalmente para fines de generación de informes cuando el usuario requiere la información en un formato diferente.

Por ejemplo, imaginemos una tabla con datos como éstos:

CustomerID	Phone1	Phone2	Phone3
1	705-491-1111	705-491-1110	NULL
2	613-492-2222	NULL	NULL
3	416-493-3333	416-493-3330	416-493-3339

## ¿Qué problema de codificación resuelve PIVOT y UNPIVOT?

Tanto el operador PIVOT como el operador UNPIVOT se utilizan normalmente para fines de generación de informes cuando el usuario requiere la información en un formato diferente.

Por ejemplo, imaginemos una tabla con datos como éstos:

CustomerID	Phone1	Phone2	Phone3	CustomerID	Phone
1	705-491-1111	705-491-1110	NULL	1	705-491-1111
2	613-492-2222	NULL	NULL	2	613-492-2222
3	416-493-3333	416-493-3330	416-493-3339	3	416-493-3333
				3	416-493-3330
				3	416-493-3339

## ¿Por qué intercambiar filas por columnas y viceversa?

La mayoría de los casos en los que se cambian filas por columnas es para entregar un informe con un formato específico. El analista de datos o

- alguien de la empresa requiere la información de una manera diferente a la estructura de la tabla. Si se trata de un requisito único, una consulta puede resultar de ayuda. De lo contrario, se podría utilizar una vista.

Es importante mencionar que la técnica PIVOT es suficiente para los requisitos comerciales. Por ejemplo, el pronóstico de ventas mensuales, la división de ventas anuales, la agregación de ventas trimestrales, etc.

## ¿Qué implicaciones tienen en el rendimiento PIVOT y UNPIVOT?

- Los operadores PIVOT y UNPIVOT son lentos y consumen muchos recursos.
- Si las tablas no son muy grandes, no tendrás problemas para utilizarlos.

# Descripción de la Sección

Objetivo de la Sección

¿Qué es un Trigger?

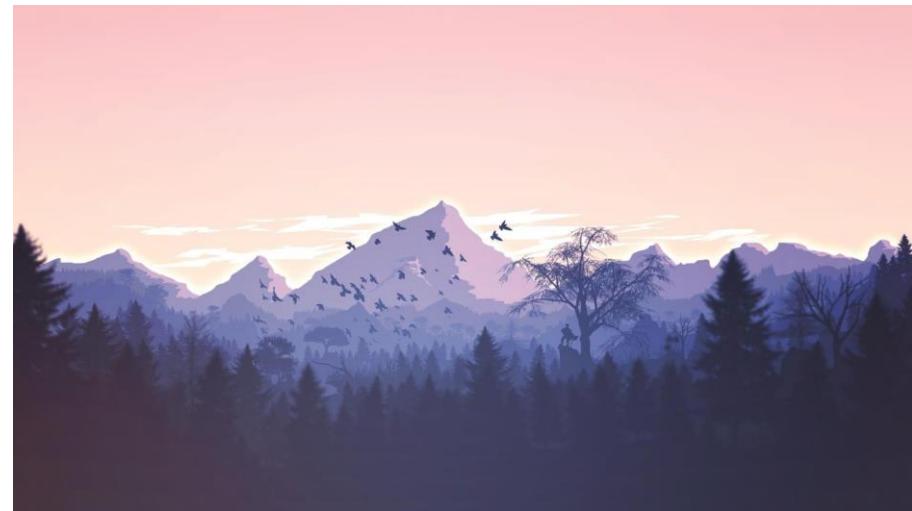
Trigger DDL - ALL SERVER

Trigger DDL - DATABASE

Trigger DML - FOR | AFTER

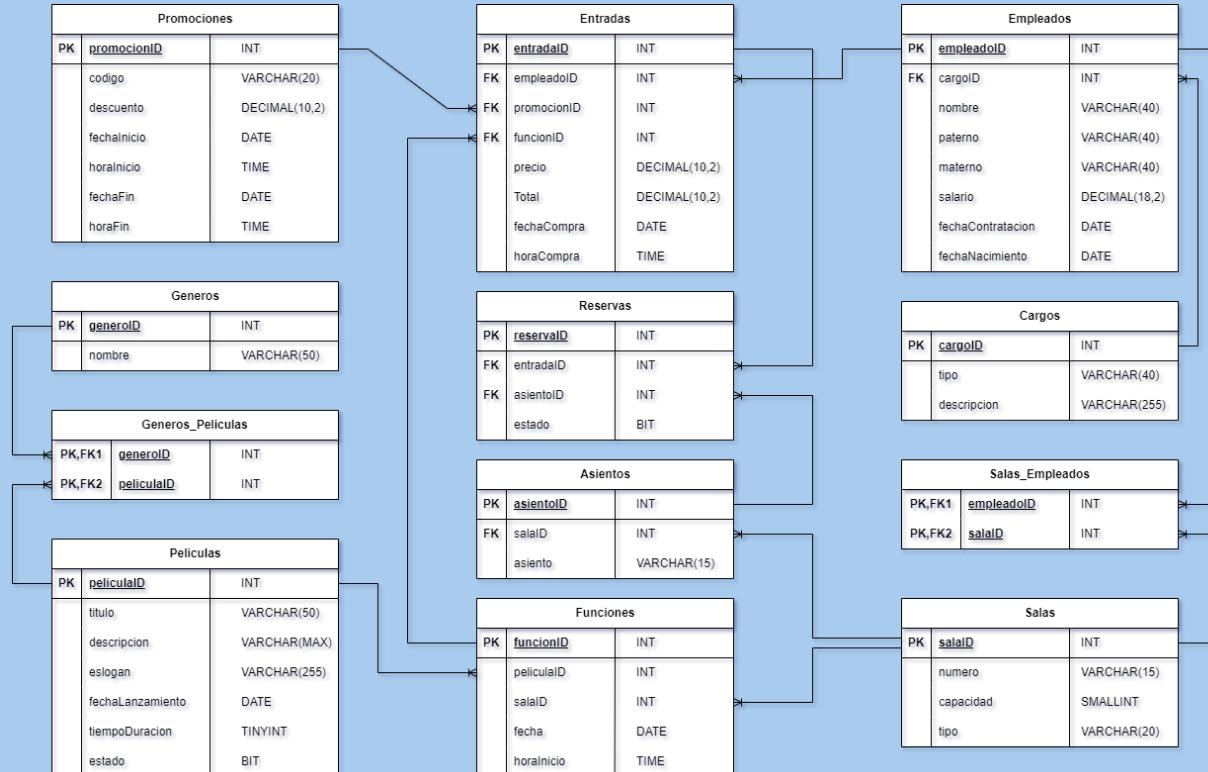
Trigger DML - INSTEAD OF

Implementación de la Base de Datos



## Objetivo de la Sección

# Objetivo de la Sección



¿Qué es un Trigger?

## ¿Qué es un Trigger?

Un trigger es un Objeto de Base de Datos que se ejecuta automáticamente cuando se produce un evento específico en una tabla o vista. Los triggers se utilizan comúnmente para

- implementar lógica empresarial adicional, como validar datos antes de insertarlos o actualizarlos, mantener la integridad referencial o realizar auditorías de cambios en los datos.

Triggers DDL

Triggers DML

LOGON Triggers

Una vez que se activan, los triggers pueden realizar una variedad de funciones, como la modificación de datos en la misma tabla o

- en otras tablas, la aplicación de restricciones y reglas de negocio, y muchas más tareas relacionadas con la gestión de datos.

# Importancia y Aplicaciones de los Triggers

- Automatización de Tareas

- Integridad de Datos

- Auditoría y Registro

- Acción en Cascada

## Trigger DDL

Se ejecutan en respuesta a una variedad de eventos de Lenguaje

- de Definición de Datos (DDL). Estos eventos corresponden principalmente a instrucciones CREATE, ALTER y DROP de Transact-SQL, y a determinados procedimientos almacenados del sistema que ejecutan operaciones de tipo DDL.

## Trigger DDL

Instrucción DDL      Instrucción Alter

CREATE [ OR ALTER ] TRIGGER trigger\_name

ON { ALL SERVER | DATABASE }

Tipo de Objeto

Nombre

El alcance del Trigger DDL será el servidor actual.

El alcance del Trigger DDL será la Base de Datos Actual.

Objeto Referenciado

## Trigger DDL

```
CREATE [ OR ALTER ] TRIGGER trigger_name
```

```
ON { ALL SERVER | DATABASE }
```

```
FOR | AFTER { event_type }
```

```
AS
```

```
[ BEGIN ]
```

Bloque de ejecución

```
Instrucción SQL;
```

```
[ END ];
```

Momento de Ejecución

## Trigger DDL

CREATE [ OR ALTER ] TRIGGER trigger\_name  
ON { ALL SERVER | DATABASE }

Servidor  
Actual

Base de Datos  
Actual

DROP TRIGGER trigger\_name  
ON ALL SERVER;

DROP TRIGGER trigger\_name  
ON DATABASE;

## Trigger DML

Se ejecutan automáticamente cuando tiene lugar un evento de

- lenguaje de manipulación de datos (DML) que afecta a la tabla o la vista definida en el Trigger. Los Triggers DML incluyen las instrucciones INSERT, UPDATE o DELETE. Los Triggers DML pueden usarse para aplicar reglas de negocios y la integridad de datos, consultar otras tablas e incluir instrucciones Transact-SQL complejas.

## Trigger DML

CREATE [ OR ALTER ] TRIGGER trigger\_name  
ON table | view  
FOR | AFTER | INSTEAD OF  
Después de la Acción      En lugar de la Acción

Instrucción DDL      Instrucción Alter      Tipo de Objeto      Nombre  
ON table | view      → Objeto Referenciado  
FOR | AFTER | INSTEAD OF      → Momento de Ejecución

**Trigger DML**

```
CREATE [ OR ALTER ] TRIGGER trigger_name  
ON table | view  
FOR | AFTER | INSTEAD OF  
[ INSERT , UPDATE , DELETE ] → Acción de Disparo
```

Bloque de ejecución

```
AS  
[ BEGIN ]  
Instrucción SQL;  
[ END ];
```

## Trigger DML

CREATE [ OR ALTER ] TRIGGER trigger\_name

ON table | view

FOR | AFTER | INSTEAD OF

Después  
de la Acción

En lugar  
de la Acción

DROP TRIGGER trigger\_name;

**Trigger DML**

En esta Tabla

CREATE TRIGGER nombre\_trigger

ON tabla

FOR INSERT

AS

BEGIN

Instrucción SQL;

Ejecuta la siguiente  
instrucciónSe dispara después de  
insertar algún registro

Después de validar:

- Tipos de Datos
- Restricciones
- Foreign Key
- Funciones

END;

**Trigger DML**

En esta Tabla

CREATE TRIGGER nombre\_trigger

ON tabla

FOR UPDATE, DELETE

AS

BEGIN

Diferentes Acciones

Instrucción SQL;

END;

Trigger DML

```
CREATE TRIGGER tgr_insert_producto_1  
ON tabla_product
```

```
AFTER INSERT
```

```
AS
```

```
BEGIN
```

Instrucción SQL;

```
END;
```

```
CREATE TRIGGER tgr_insert_producto_1  
ON tabla_product
```

```
AFTER INSERT
```

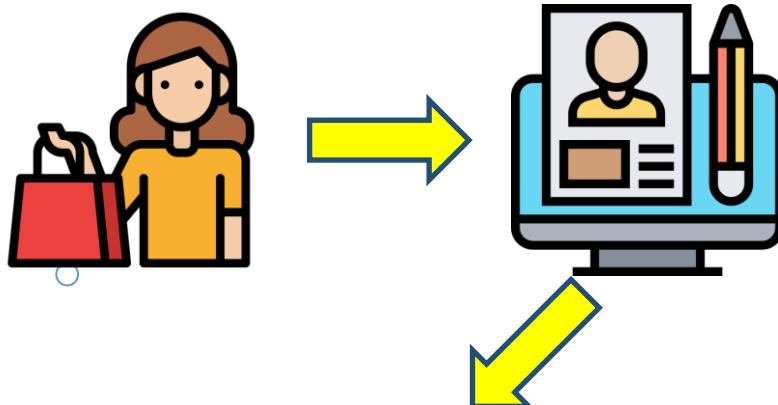
```
AS
```

```
BEGIN
```

Instrucción SQL;

```
END;
```

# Triggers(Disparadores)



id	nombre	paterno	materno	edad
101	Isabel	Campos	Torres	19
102	Marco	Gallegos	Ochoa	35
103	Julia	Gamboa	Gallegos	58
...	...	...	...	...

```
CREATE TRIGGER tgr_insert_clientes  
ON CLIENTES  
FOR INSERT  
AS  
BEGIN
```

*Inserted* → Perdurar durante la ejecución

*Deleted*

```
END;
```

## Triggers(Disparadores)



CLIENTES

id	nombre	paterno	materno	edad
101	Isabel	Campos	Torres	19
102	Marco	Gallegos	Ochoa	35
103	Julia	Gamboa	Gallegos	58
...	...	...	...	...

```
CREATE TRIGGER tgr_insert_clientes  
ON CLIENTES  
FOR INSERT  
AS  
BEGIN
```

*Inserted*

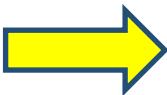
id	nombre	paterno	materno	edad
104	Oscar	Mesías	Guillén	25

*Deleted*

id	nombre	paterno	materno	edad

END;

# Triggers(Disparadores)



CLIENTES

id	nombre	paterno	materno	edad
101	Isabel	Campos	Torres	19
102	Marco	Gallegos	Ochoa	35
103	Julia	Gamboa	Gallegos	58
...	...	...	...	...

```
CREATE TRIGGER tgr_insert_clientes  
ON CLIENTES  
FOR DELETE  
AS  
BEGIN
```

*Inserted*

id	nombre	paterno	materno	edad

*Deleted*

id	nombre	paterno	materno	edad
101	Isabel	Campos	Torres	19

END;

## Triggers(Disparadores)



CLIENTES

id	nombre	paterno	materno	edad
101	Isabel	Campos	Torres	19
102	Marco	Gallegos	Ochoa	35
103	Julia	Gamboa	Gallegos	58
...	...	...	...	...

```
CREATE TRIGGER tgr_insert_clientes  
ON CLIENTES  
FOR UPDATE  
AS  
BEGIN
```

*Inserted*

id	nombre	paterno	materno	edad
101	Karen	Campos	Torres	19

*Deleted*

id	nombre	paterno	materno	edad
101	Isabel	Campos	Torres	19

END;

## Trigger DDL - ALL SERVER

CREATE [ OR ALTER ] TRIGGER nombre\_trigger

ON { ALL SERVER | DATABASE }

Servidor  
Actual

Base de Datos  
Actual

FOR | AFTER { event\_type }

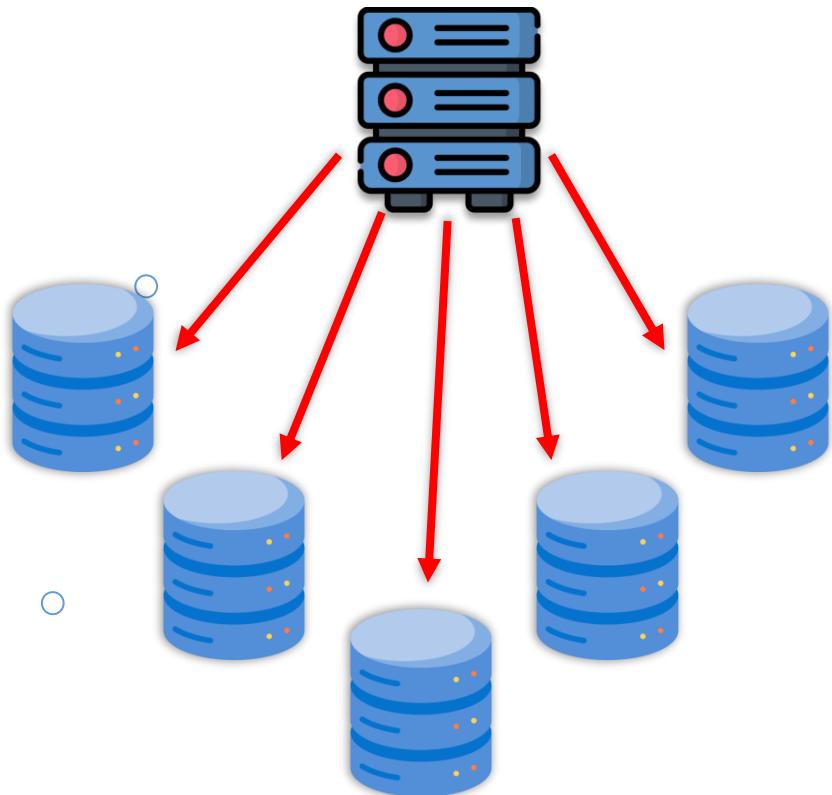
AS

[ BEGIN ]

Instrucción SQL;

[ END ];

## Trigger DDL - ALL SERVER



```
CREATE [ OR ALTER ] TRIGGER nombre_trigger  
ON ALL SERVER  
FOR | AFTER { event_type }  
AS  
[ BEGIN ]  
  
Instrucción SQL;  
[ END ];
```

- CREATE\_DATABASE
- ALTER\_DATABASE
- CREATE\_TABLE
- ALTER\_TABLE
- DROP\_TABLE
- RENAME
- CREATE\_SCHEMA
- CREATE\_VIEW
- ALTER\_VIEW
- DROP\_VIEW
- Etc

## Trigger DDL - DATABASE

CREATE [ OR ALTER ] TRIGGER nombre\_trigger

ON { ALL SERVER | DATABASE }

Servidor  
Actual

Base de Datos  
Actual

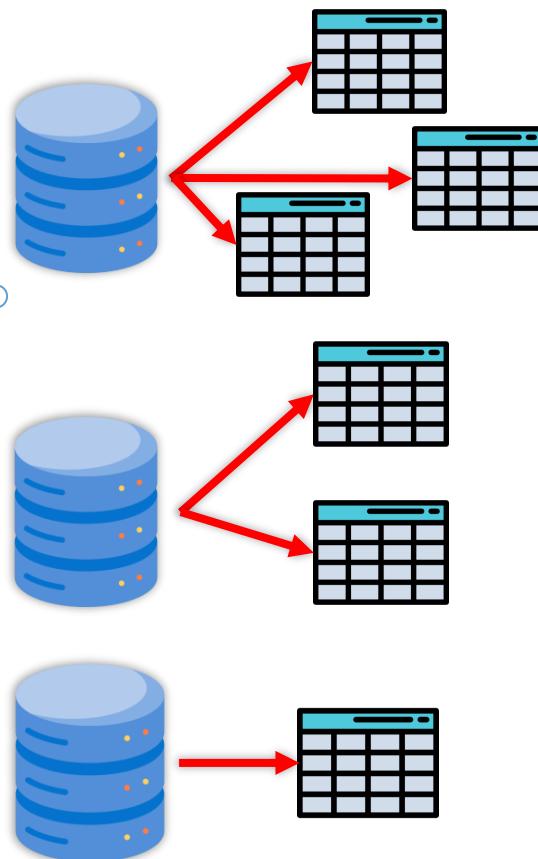
FOR | AFTER { event\_type }

AS

[ BEGIN ]

Instrucción SQL;

[ END ];



```
CREATE [ OR ALTER ] TRIGGER nombre_trigger  
ON DATABASE  
FOR | AFTER { event_type }  
AS  
[ BEGIN ]  
Instrucción SQL;  
[ END ];
```

- CREATE\_DATABASE
- ALTER\_DATABASE
- CREATE\_TABLE
- ALTER\_TABLE
- DROP\_TABLE
- RENAME
- CREATE\_SCHEMA
- CREATE\_VIEW
- ALTER\_VIEW
- DROP\_VIEW
- Etc

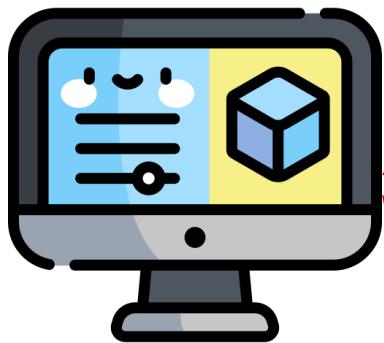
## Trigger DML - FOR | AFTER

CREATE TRIGGER nombre\_trigger  
ON tabla | vista  
1 solo Trigger por  
cada tipo de acción ← **FOR | AFTER | INSTEAD OF**  
Después  
de la Acción      En lugar  
de la Acción  
[ INSERT , UPDATE , DELETE ]  
AS  
BEGIN  
  
Instrucción SQL;  
END;

# Automatización de Tareas

**PRODUCTOS**

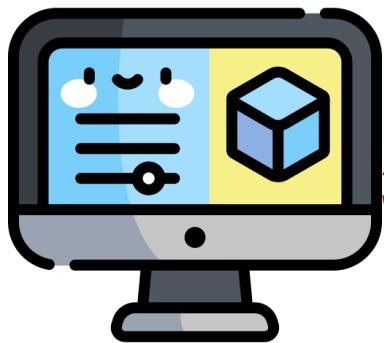
<b>id_producto</b>	<b>nombre</b>	<b>stock</b>
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	100
103	SSD Kingston 960GB	50
...	...	...

**DETALLE\_FACTURA**

<b>codigo</b>	<b>id_producto</b>	<b>precio</b>	<b>cantidad</b>

## PRODUCTOS

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	100
103	SSD Kingston 960GB	50
...	...	...

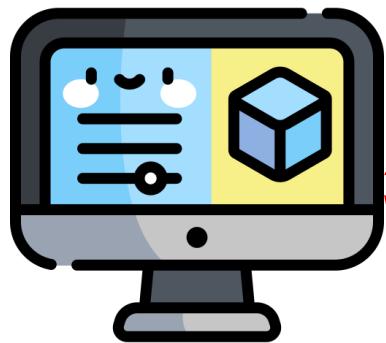


## DETALLE\_FACTURA

codigo	id_producto	precio	cantidad
F101	102	499.90	10

## PRODUCTOS

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	90
103	SSD Kingston 960GB	50
...	...	...



## DETALLE\_FACTURA

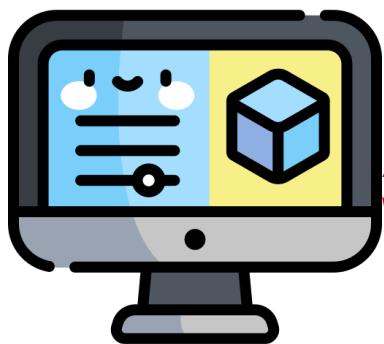
codigo	id_producto	precio	cantidad
F101	102	499.90	10



# Triggers

## PRODUCTOS

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	90
103	SSD Kingston 960GB	30
...	...	...



## DETALLE\_FACTURA

codigo	id_producto	precio	cantidad
F101	102	499.90	10
F101	103	99.90	20



**Triggers**

Insert  
Delete  
Update

## Insert

```
CREATE TRIGGER tgr_stock_insert
```

```
ON DETALLE_FACTURA
```

```
AFTER INSERT
```

```
AS
```

```
BEGIN
```

```
END;
```

## Insert

PRODUCTOS

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	100
103	SSD Kingston 960GB	50
...	...	...

DETALLE\_FACTURA

codigo	id_producto	precio	cantidad

```
CREATE TRIGGER tgr_stock_insert
ON DETALLE_FACTURA
AFTER INSERT
AS
BEGIN
    UPDATE PRODUCTOS
    SET stock = stock - total_cantidad
    FROM (
        SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
        FROM Inserted
        GROUP BY id_producto
    ) T
    WHERE id_producto = T.id;
END;
```

## Insert

PRODUCTOS

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	100
103	SSD Kingston 960GB	50
...	...	...

codigo	id_producto	precio	cantidad
F101	102	99.90	10

DETALLE\_FACTURA

codigo	id_producto	precio	cantidad
F101	102	99.90	10

```
CREATE TRIGGER tgr_stock_insert
ON DETALLE_FACTURA
AFTER INSERT
AS
BEGIN
    UPDATE PRODUCTOS
    SET stock = stock - total_cantidad
    FROM (
        SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
        FROM Inserted
        GROUP BY id_producto
    ) T
    WHERE id_producto = T.id;
END;
```

## Insert

PRODUCTOS

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	100
103	SSD Kingston 960GB	50
...	...	...

id_producto	total_cantidad
102	10

DETALLE\_FACTURA

codigo	id_producto	precio	cantidad
F101	102	99.90	10

```
CREATE TRIGGER tgr_stock_insert
ON DETALLE_FACTURA
AFTER INSERT
AS
BEGIN
    UPDATE PRODUCTOS
    SET stock = stock - total_cantidad
    FROM (
        SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
        FROM Inserted
        GROUP BY id_producto
    ) T
    WHERE id_producto = T.id;
END;
```

## Insert

PRODUCTOS

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	100
103	SSD Kingston 960GB	50
...	...	...

id_producto	total_cantidad
102	10
102	20

DETALLE\_FACTURA

codigo	id_producto	precio	cantidad
F101	102	99.90	10
F102	102	99.90	20

```
CREATE TRIGGER tgr_stock_insert
ON DETALLE_FACTURA
AFTER INSERT
AS
BEGIN
    UPDATE PRODUCTOS
    SET stock = stock - total_cantidad
    FROM (
        SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
        FROM Inserted
        GROUP BY id_producto
    ) T
    WHERE id_producto = T.id;
END;
```

## Insert

PRODUCTOS

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	100
103	SSD Kingston 960GB	50
...	...	...

id_producto	total_cantidad
102	30

DETALLE\_FACTURA

codigo	id_producto	precio	cantidad
F101	102	99.90	10
F102	102	99.90	20

```
CREATE TRIGGER tgr_stock_insert
ON DETALLE_FACTURA
AFTER INSERT
AS
BEGIN
    UPDATE PRODUCTOS
    SET stock = stock - total_cantidad
    FROM (
        SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
        FROM Inserted
        GROUP BY id_producto
    ) T
    WHERE id_producto = T.id;
END;
```

## Insert

PRODUCTOS

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	100
103	SSD Kingston 960GB	50
...	...	...

id_producto	total_cantidad
102	30

DETALLE\_FACTURA

codigo	id_producto	precio	cantidad
F101	102	99.90	10
F102	102	99.90	20

```
CREATE TRIGGER tgr_stock_insert
ON DETALLE_FACTURA
AFTER INSERT
AS
BEGIN
    UPDATE PRODUCTOS
    SET stock = stock - total_cantidad
    FROM (
        SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
        FROM Inserted
        GROUP BY id_producto
    ) T
    WHERE id_producto = T.id;
END;
```

## Insert

PRODUCTOS

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	70
103	SSD Kingston 960GB	50
...	...	...

```
CREATE TRIGGER tgr_stock_insert
```

```
ON DETALLE_FACTURA
```

```
AFTER INSERT
```

```
AS
```

```
BEGIN
```

```
UPDATE PRODUCTOS
```

```
SET stock = stock - total_cantidad
```

```
FROM (
```

```
SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
```

```
FROM Inserted
```

```
GROUP BY id_producto
```

```
) T
```

```
WHERE id_producto = T.id;
```

```
END;
```

DETALLE\_FACTURA

codigo	id_producto	precio	cantidad
F101	102	99.90	10
F102	102	99.90	20

## Delete

PRODUCTOS

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	70
103	SSD Kingston 960GB	50
...	...	...

DETALLE\_FACTURA

codigo	id_producto	precio	cantidad
F101	102	99.90	10
F102	102	99.90	20

```
CREATE TRIGGER tgr_stock_delete
ON DETALLE_FACTURA
AFTER DELETE
AS
BEGIN
    UPDATE PRODUCTOS
    SET stock = stock + total_cantidad
    FROM (
        SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
        FROM Deleted
        GROUP BY id_producto
    ) T
    WHERE id_producto = T.id;
END;
```

## Delete

PRODUCTOS

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	70
103	SSD Kingston 960GB	50
...	...	...

DETALLE\_FACTURA

codigo	id_producto	precio	cantidad
F101	102	99.90	10
F102	102	99.90	20

```
CREATE TRIGGER tgr_stock_delete
ON DETALLE_FACTURA
AFTER DELETE
AS
BEGIN
    UPDATE PRODUCTOS
    SET stock = stock + total_cantidad
    FROM (
        SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
        FROM Deleted
        GROUP BY id_producto
    ) T
    WHERE id_producto = T.id;
END;
```

## Delete

PRODUCTOS

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	70
103	SSD Kingston 960GB	50
...	...	...

DETALLE\_FACTURA

codigo	id_producto	precio	cantidad
F101	102	99.90	10
F102	102	99.90	20

```
CREATE TRIGGER tgr_stock_delete
ON DETALLE_FACTURA
AFTER DELETE
AS
BEGIN
    UPDATE PRODUCTOS
    SET stock = stock + total_cantidad
    FROM (
        SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
        FROM Deleted
        GROUP BY id_producto
    ) T
    WHERE id_producto = T.id;
END;
```

## Delete

PRODUCTOS

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	70
103	SSD Kingston 960GB	50
...	...	...

id_producto	total_cantidad
102	20

DETALLE\_FACTURA

codigo	id_producto	precio	cantidad
F101	102	99.90	10

```
CREATE TRIGGER tgr_stock_delete
```

```
ON DETALLE_FACTURA
```

```
AFTER DELETE
```

```
AS  
BEGIN
```

```
UPDATE PRODUCTOS
```

```
SET stock = stock + total_cantidad
```

```
FROM (
```

```
SELECT id_producto AS id, SUM(cantidad) AS total_cantidad  
FROM Deleted  
GROUP BY id_producto
```

```
) T
```

```
WHERE id_producto = T.id;
```

```
END;
```

## Delete

PRODUCTOS

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	70
103	SSD Kingston 960GB	50
...	...	...

id_producto	total_cantidad
102	20

DETALLE\_FACTURA

codigo	id_producto	precio	cantidad
F101	102	99.90	10

```
CREATE TRIGGER tgr_stock_delete
ON DETALLE_FACTURA
AFTER DELETE
AS
BEGIN
    UPDATE PRODUCTOS
    SET stock = stock + total_cantidad
    FROM (
        SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
        FROM Deleted
        GROUP BY id_producto
    ) T
    WHERE id_producto = T.id;
END;
```

## Delete

PRODUCTOS

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	90
103	SSD Kingston 960GB	50
...	...	...

```
CREATE TRIGGER tgr_stock_delete
```

```
ON DETALLE_FACTURA
```

```
AFTER DELETE
```

```
AS
```

```
BEGIN
```

```
UPDATE PRODUCTOS
```

```
SET stock = stock + total_cantidad
```

```
FROM (
```

```
SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
```

```
FROM Deleted
```

```
GROUP BY id_producto
```

```
) T
```

```
WHERE id_producto = T.id;
```

```
END;
```

DETALLE\_FACTURA

codigo	id_producto	precio	cantidad
F101	102	99.90	10

### Update

**stock = stock - stock\_Insertado + stock\_Eliminado**

## Update

**stock = stock - stock\_Inserted + stock\_Deleted**

```
SELECT id_producto AS id, SUM(cantidad) AS total_cantidad  
FROM Inserted  
GROUP BY id_producto
```

```
SELECT id_producto AS id, SUM(cantidad) AS total_cantidad  
FROM Deleted  
GROUP BY id_producto
```

## Update

```
SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
FROM(
    SELECT id_producto, -cantidad AS cantidad FROM Inserted
    UNION ALL
    SELECT id_producto, cantidad FROM Deleted
) T
GROUP BY id_producto
```

## Update

```
SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
FROM(
    SELECT id_producto, -cantidad AS cantidad FROM Inserted
    UNION ALL
    SELECT id_producto, cantidad FROM Deleted
) T
GROUP BY id_producto
```

## Update

```
SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
FROM(
    SELECT id_producto, -cantidad AS cantidad FROM Inserted
    UNION ALL
    SELECT id_producto, cantidad FROM Deleted
) T
GROUP BY id_producto
```

## Update

**stock = stock + total\_cantidad**

```
SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
FROM(
    SELECT id_producto, -cantidad AS cantidad FROM Inserted
    UNION ALL
    SELECT id_producto, cantidad FROM Deleted
) T
GROUP BY id_producto
```

## Update

```
CREATE TRIGGER tgr_stock_update
ON DETALLE_FACTURA
AFTER UPDATE
AS
BEGIN
    UPDATE PRODUCTOS
    SET stock = stock + total_cantidad
    FROM (
        SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
        FROM(
            SELECT id_producto, -cantidad AS cantidad FROM Inserted
            UNION ALL
            SELECT id_producto, cantidad FROM Deleted
        ) T
        GROUP BY id_producto
    ) A
    WHERE id_producto = A.id;
END;
```

## Update

PRODUCTOS

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	90
103	SSD Kingston 960GB	50
...	...	...

DETALLE\_FACTURA

codigo	id_producto	precio	cantidad
F101	102	99.90	10

```
CREATE TRIGGER tgr_stock_update
ON DETALLE_FACTURA
AFTER UPDATE
AS
BEGIN

    UPDATE PRODUCTOS
    SET stock = stock + total_cantidad
    FROM (
        SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
        FROM(
            SELECT id_producto, -cantidad AS cantidad FROM Inserted
            UNION ALL
            SELECT id_producto, cantidad FROM Deleted
        ) T
        GROUP BY id_producto
    ) A
    WHERE id_producto = A.id;

END;
```

## Update

**PRODUCTOS**

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	90
103	SSD Kingston 960GB	50
...	...	...

id_producto	cantidad
102	20

id_producto	cantidad
102	10

**DETALLE\_FACTURA**

codigo	id_producto	precio	cantidad
F101	102	99.90	20

```

CREATE TRIGGER tgr_stock_update
ON DETALLE_FACTURA
AFTER UPDATE
AS
BEGIN
    UPDATE PRODUCTOS
    SET stock = stock + total_cantidad
    FROM (
        SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
        FROM(
            SELECT id_producto, -cantidad AS cantidad FROM Inserted
            UNION ALL
            SELECT id_producto, cantidad FROM Deleted
        ) T
        GROUP BY id_producto
    ) A
    WHERE id_producto = A.id;
END;

```

## Update

PRODUCTOS

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	90
103	SSD Kingston 960GB	50
...	...	...

id_producto	cantidad
102	20

id_producto	cantidad
102	10

 $-20 + 10$ 

DETALLE\_FACTURA

codigo	id_producto	precio	cantidad
F101	102	99.90	20

```

CREATE TRIGGER tgr_stock_update
ON DETALLE_FACTURA
AFTER UPDATE
AS
BEGIN
    UPDATE PRODUCTOS
    SET stock = stock + total_cantidad
    FROM (
        SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
        FROM(
            SELECT id_producto, -cantidad AS cantidad FROM Inserted
            UNION ALL
            SELECT id_producto, cantidad FROM Deleted
        ) T
        GROUP BY id_producto
    ) A
    WHERE id_producto = A.id;
END;

```

## Update

**PRODUCTOS**

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	90
103	SSD Kingston 960GB	50
...	...	...

id_producto	Total_cantidad
102	-10

**DETALLE\_FACTURA**

codigo	id_producto	precio	cantidad
F101	102	99.90	20

```

CREATE TRIGGER tgr_stock_update
ON DETALLE_FACTURA
AFTER UPDATE
AS
BEGIN

    UPDATE PRODUCTOS
    SET stock = stock + total_cantidad
    FROM (
        SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
        FROM(
            SELECT id_producto, -cantidad AS cantidad FROM Inserted
            UNION ALL
            SELECT id_producto, cantidad FROM Deleted
        ) T
        GROUP BY id_producto
    ) A
    WHERE id_producto = A.id;

END;

```

## Update

PRODUCTOS

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	90
103	SSD Kingston 960GB	50
...	...	...

id_producto	Total_cantidad
102	-10

DETALLE\_FACTURA

codigo	id_producto	precio	cantidad
F101	102	99.90	20

```
CREATE TRIGGER tgr_stock_update
ON DETALLE_FACTURA
AFTER UPDATE
AS
BEGIN
    UPDATE PRODUCTOS
    SET stock = stock + total_cantidad
    FROM (
        SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
        FROM(
            SELECT id_producto, -cantidad AS cantidad FROM Inserted
            UNION ALL
            SELECT id_producto, cantidad FROM Deleted
        ) T
        GROUP BY id_producto
    ) A
    WHERE id_producto = A.id;
END;
```

## Update

PRODUCTOS

id_producto	nombre	stock
101	Monitor LG 27 pulgadas	50
102	Mouse Logitech G502	80
103	SSD Kingston 960GB	50
...	...	...

```
CREATE TRIGGER tgr_stock_update
```

```
ON DETALLE_FACTURA
```

```
AFTER UPDATE
```

```
AS
```

```
BEGIN
```

```
UPDATE PRODUCTOS
```

```
SET stock = stock + total_cantidad
```

```
FROM (
```

```
SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
```

```
FROM(
```

```
SELECT id_producto, -cantidad AS cantidad FROM Inserted
```

```
UNION ALL
```

```
SELECT id_producto, cantidad FROM Deleted
```

```
) T
```

```
GROUP BY id_producto
```

```
) A
```

```
WHERE id_producto = A.id;
```

```
END;
```

DETALLE\_FACTURA

codigo	id_producto	precio	cantidad
F101	102	99.90	20

## Update

```
CREATE TRIGGER tgr_stock_update
ON DETALLE_FACTURA
AFTER UPDATE
AS
BEGIN

    UPDATE PRODUCTOS
    SET stock = stock + total_cantidad
    FROM (
        SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
        FROM(
            SELECT id_producto, -cantidad AS cantidad FROM Inserted
            UNION ALL
            SELECT id_producto, cantidad FROM Deleted
        ) T
        GROUP BY id_producto
    ) A
    WHERE id_producto = A.id;

END;
```

Inserción

Deleted = Vacía

Eliminación

Inserted = Vacía

## Update

```
CREATE TRIGGER tgr_stock_modificados
ON DETALLE_FACTURA
AFTER UPDATE, INSERT, DELETE
AS
BEGIN
    UPDATE PRODUCTOS
    SET stock = stock + total_cantidad
    FROM (
        SELECT id_producto AS id, SUM(cantidad) AS total_cantidad
        FROM(
            SELECT id_producto, -cantidad AS cantidad FROM Inserted
            UNION ALL
            SELECT id_producto, cantidad FROM Deleted
        ) T
        GROUP BY id_producto
    ) A
    WHERE id_producto = A.id;
END;
```

Trigger DML - INSTEAD OF

CREATE TRIGGER nombre\_trigger  
ON tabla | vista  
FOR | AFTER | INSTEAD OF  
[ INSERT , UPDATE , DELETE ]  
AS  
BEGIN  
Instrucción SQL;  
END;

1 solo Trigger por cada tipo de acción

Después de la Acción      En lugar de la Acción



Un "Trigger INSTEAD OF" es un tipo especial de Trigger que se dispara en lugar de la acción desencadenante habitual. Esto significa que en lugar de que la operación de Inserción, Actualización o Eliminación se ejecute directamente en la Tabla, el Trigger INSTEAD OF se ejecutará en su lugar, permitiéndote controlar y modificar la operación según sea necesario.

Este tipo de Trigger es comúnmente utilizado en situaciones donde deseas realizar acciones personalizadas o complejas en lugar de la operación de base de datos estándar. Por ejemplo, puedes utilizarlo en Vistas o Tablas que involucran múltiples tablas para manipular los datos de forma más específica o restringir ciertas operaciones según ciertas condiciones.

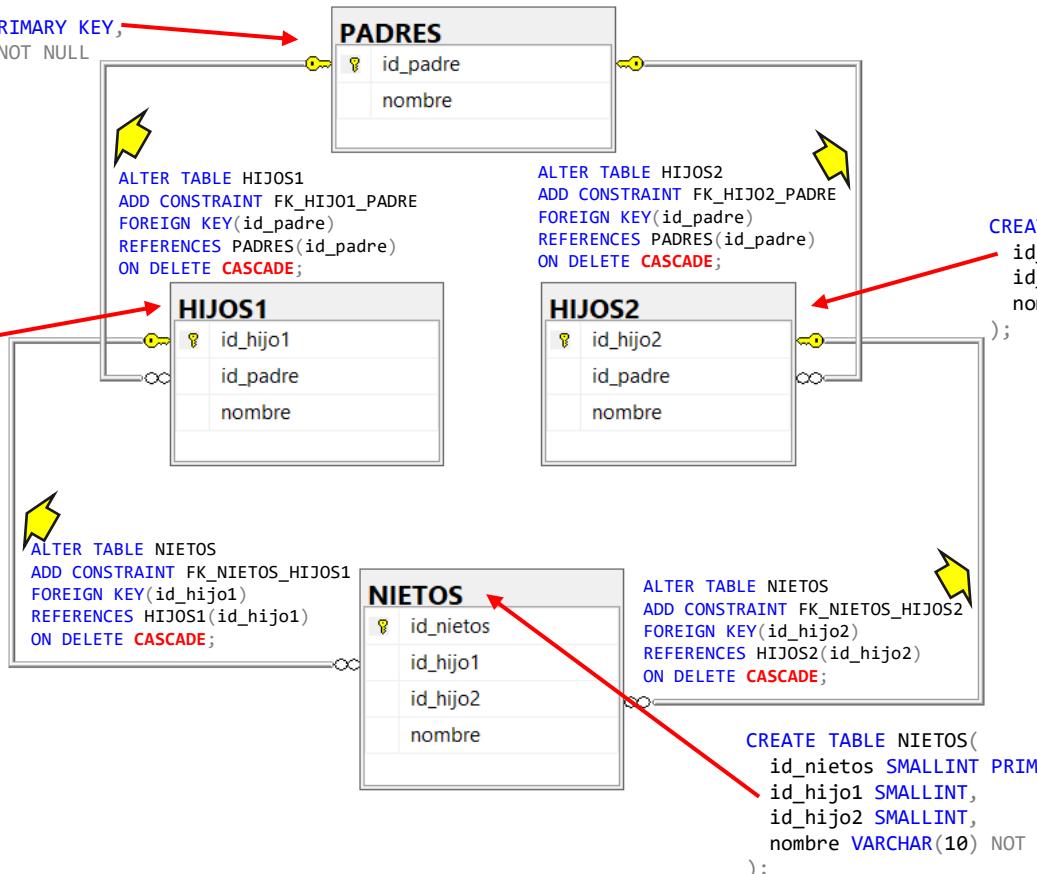
En esta Tabla

```
CREATE TRIGGER nombre_trigger  
ON tabla  
INSTEAD OF INSERT  
AS  
BEGIN  
    Instrucción SQL;  
END;
```

Cancela la acción desencadenadora

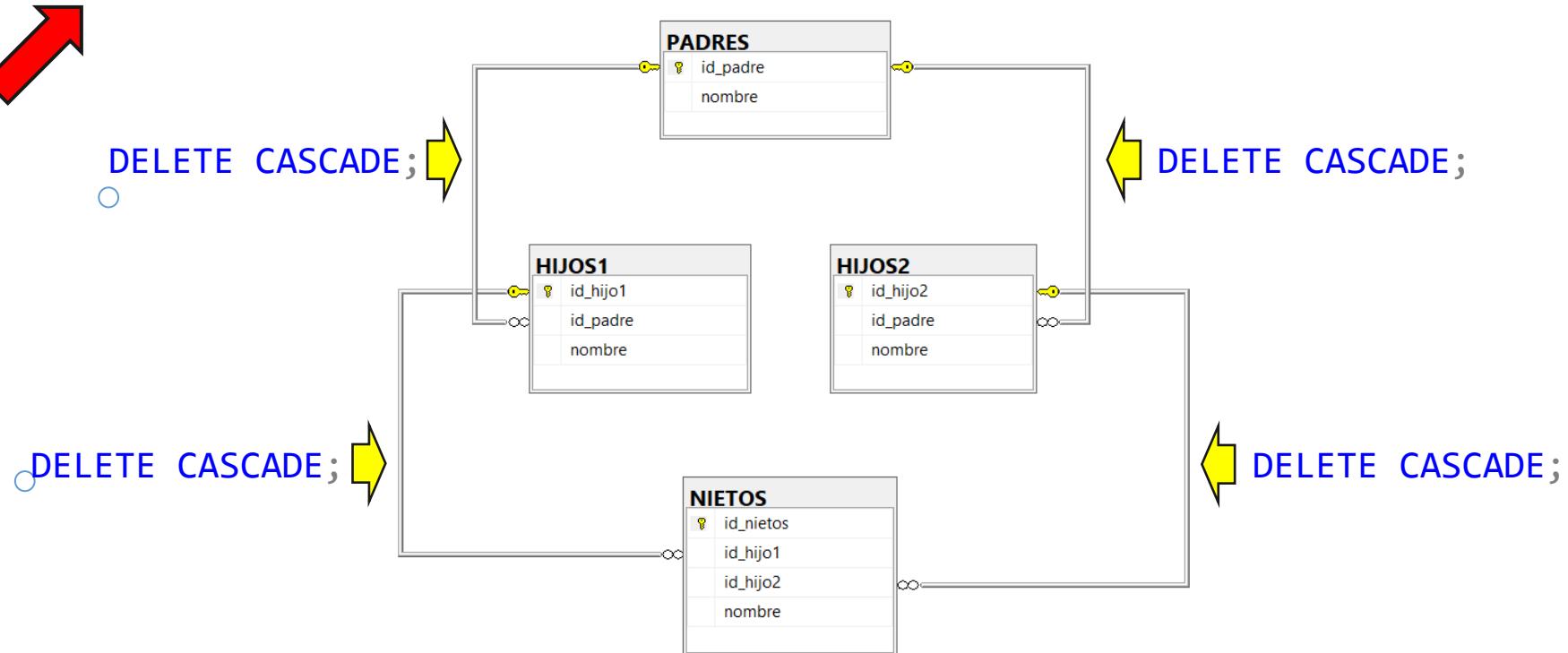
Ejecuta la instrucción SQL.

```
CREATE TABLE PADRES(  
    id_padre SMALLINT PRIMARY KEY,  
    nombre VARCHAR(10) NOT NULL  
);
```



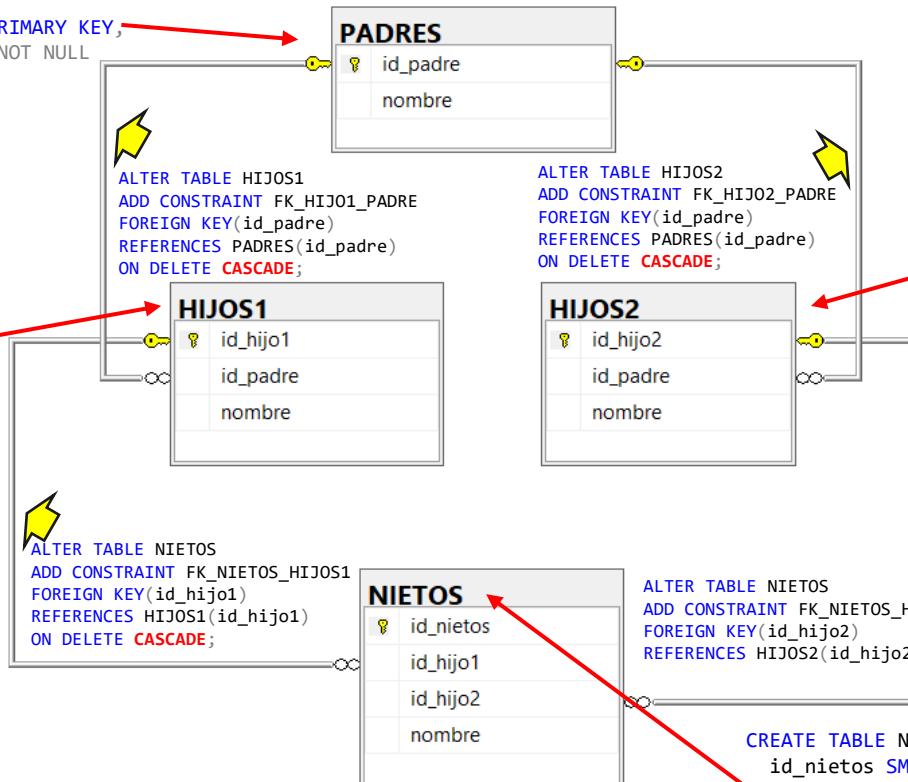
Msg 1785, Level 16, State 0, Line 78

Si especifica la restricción FOREIGN KEY 'FK\_HIJOS\_HIJOS2' en la tabla 'HIJOS', podrían producirse ciclos o múltiples rutas en cascada. Especifique ON DELETE NO ACTION o UPDATE NO ACTION, o bien modifique otras restricciones FOREIGN KEY.



No está permitido en Muchos Manejadores de Bases de Datos

```
CREATE TABLE PADRES(  
    id_padre SMALLINT PRIMARY KEY,  
    nombre VARCHAR(10) NOT NULL  
);
```



```
CREATE TABLE HIJOS1(  
    id_hijo1 SMALLINT PRIMARY KEY,  
    id_padre SMALLINT,  
    nombre VARCHAR(10) NOT NULL  
);
```

```
ALTER TABLE HIJOS1  
ADD CONSTRAINT FK_HIJOS1_PADRE  
FOREIGN KEY(id_padre)  
REFERENCES PADRES(id_padre)  
ON DELETE CASCADE;
```

```
ALTER TABLE HIJOS2  
ADD CONSTRAINT FK_HIJOS2_PADRE  
FOREIGN KEY(id_padre)  
REFERENCES PADRES(id_padre)  
ON DELETE CASCADE;
```

```
CREATE TABLE HIJOS2(  
    id_hijo2 SMALLINT PRIMARY KEY,  
    id_padre SMALLINT,  
    nombre VARCHAR(10) NOT NULL  
);
```



```
ALTER TABLE NIETOS  
ADD CONSTRAINT FK_NIETOS_HIJOS1  
FOREIGN KEY(id_hijo1)  
REFERENCES HIJOS1(id_hijo1)  
ON DELETE CASCADE;
```

```
ALTER TABLE NIETOS  
ADD CONSTRAINT FK_NIETOS_HIJOS2  
FOREIGN KEY(id_hijo2)  
REFERENCES HIJOS2(id_hijo2);
```

```
CREATE TABLE NIETOS(  
    id_nietos SMALLINT PRIMARY KEY,  
    id_hijo1 SMALLINT,  
    id_hijo2 SMALLINT,  
    nombre VARCHAR(10) NOT NULL  
);
```

```
CREATE TABLE PADRES(  
    id_padre SMALLINT PRIMARY KEY,  
    nombre VARCHAR(10) NOT NULL  
)
```



```
ALTER TABLE HIJOS1  
ADD CONSTRAINT FK_HIJO1_PADRE  
FOREIGN KEY(id_padre)  
REFERENCES PADRES(id_padre);
```



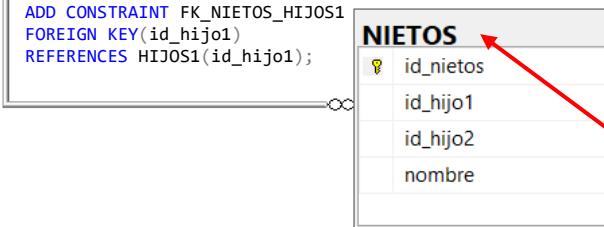
```
CREATE TABLE HIJOS1(  
    id_hijo1 SMALLINT PRIMARY KEY,  
    id_padre SMALLINT,  
    nombre VARCHAR(10) NOT NULL  
)
```

```
ALTER TABLE HIJOS2  
ADD CONSTRAINT FK_HIJO2_PADRE  
FOREIGN KEY(id_padre)  
REFERENCES PADRES(id_padre);
```



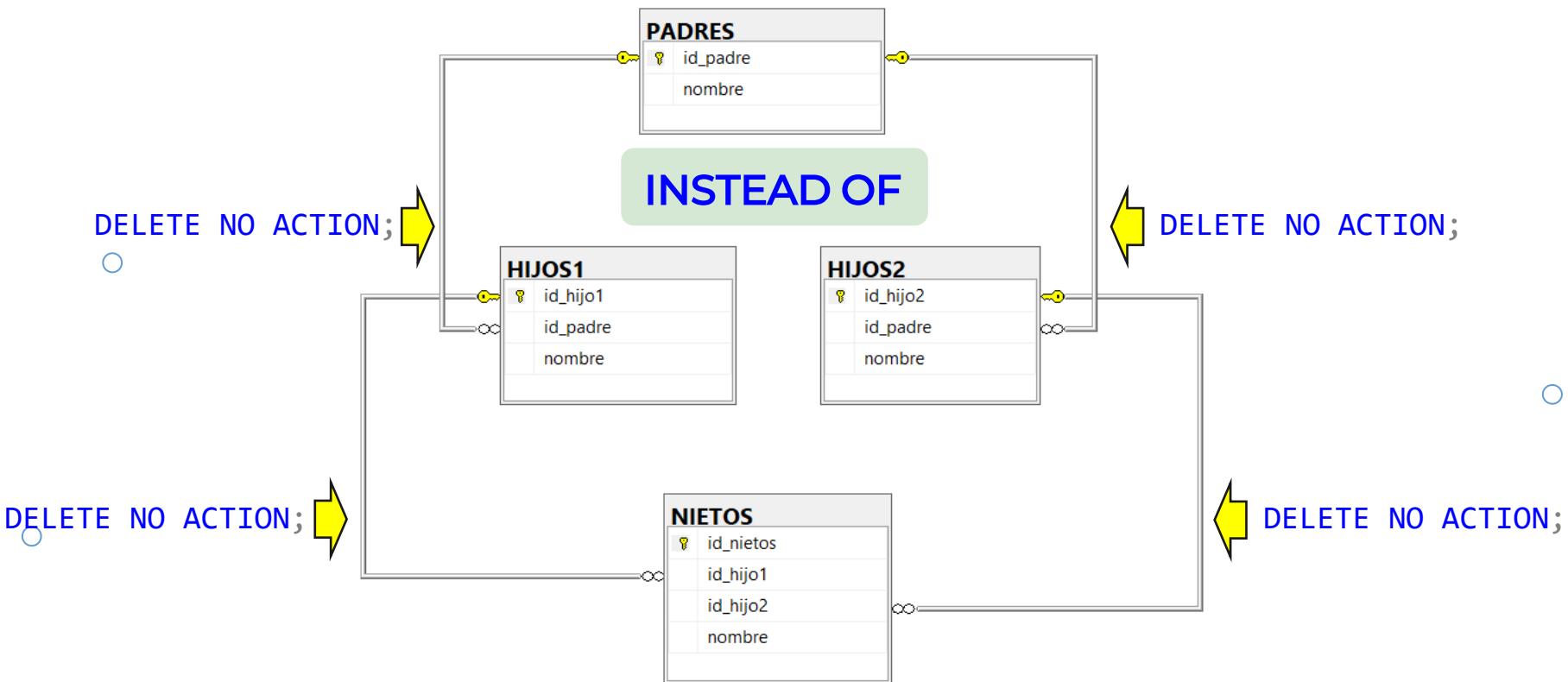
```
CREATE TABLE HIJOS2(  
    id_hijo2 SMALLINT PRIMARY KEY,  
    id_padre SMALLINT,  
    nombre VARCHAR(10) NOT NULL  
)
```

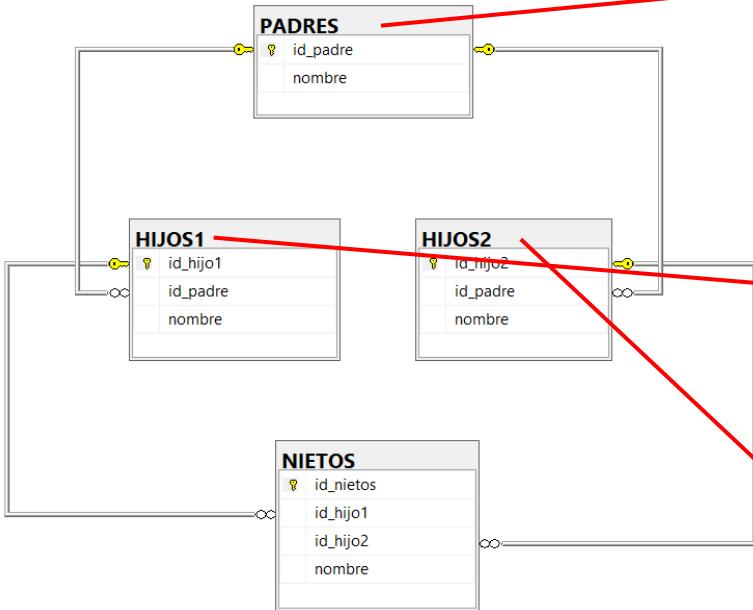
```
ALTER TABLE NIETOS  
ADD CONSTRAINT FK_NIETOS_HIJOS1  
FOREIGN KEY(id_hijo1)  
REFERENCES HIJOS1(id_hijo1);
```



```
ALTER TABLE NIETOS  
ADD CONSTRAINT FK_NIETOS_HIJOS2  
FOREIGN KEY(id_hijo2)  
REFERENCES HIJOS2(id_hijo2);
```

```
CREATE TABLE NIETOS(  
    id_nietos SMALLINT PRIMARY KEY,  
    id_hijo1 SMALLINT,  
    id_hijo2 SMALLINT,  
    nombre VARCHAR(10) NOT NULL  
)
```





```
CREATE TRIGGER tgr_eliminar_padres
ON PADRES
INSTEAD OF DELETE
AS
BEGIN

DELETE FROM HIJOS1 WHERE id_padre IN (SELECT id_padre FROM DELETED)
DELETE FROM HIJOS2 WHERE id_padre IN (SELECT id_padre FROM DELETED)
DELETE FROM PADRES WHERE id_padre IN (SELECT id_padre FROM DELETED)

END;

CREATE TRIGGER tgr_eliminar_HIJOS1
ON HIJOS1
INSTEAD OF DELETE
AS
BEGIN

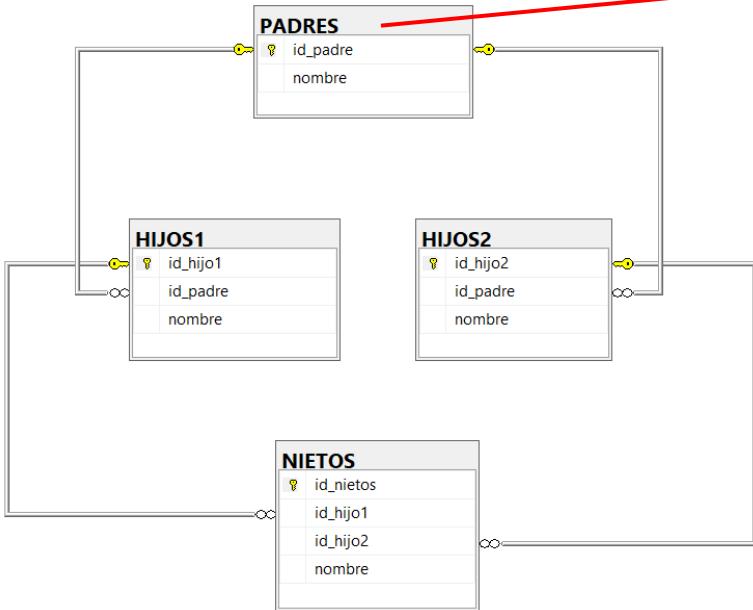
DELETE FROM NIETOS WHERE id_hijo1 IN (SELECT id_hijo1 FROM DELETED)
DELETE FROM HIJOS1 WHERE id_hijo1 IN (SELECT id_hijo1 FROM DELETED)

END;

CREATE TRIGGER tgr_eliminar_HIJOS2
ON HIJOS2
INSTEAD OF DELETE
AS
BEGIN

DELETE FROM NIETOS WHERE id_hijo2 IN (SELECT id_hijo2 FROM DELETED)
DELETE FROM HIJOS2 WHERE id_hijo2 IN (SELECT id_hijo2 FROM DELETED)

END;
```



```
CREATE TRIGGER tgr_eliminar_padres  
ON PADRES  
INSTEAD OF DELETE
```

```
AS  
BEGIN
```

```
DELETE FROM HIJOS1 WHERE id_padre IN (SELECT id_padre FROM DELETED)  
DELETE FROM HIJOS2 WHERE id_padre IN (SELECT id_padre FROM DELETED)  
DELETE FROM PADRES WHERE id_padre IN (SELECT id_padre FROM DELETED)
```

```
END;
```

```
CREATE TRIGGER tgr_eliminar_hijos1  
ON HIJOS1  
INSTEAD OF DELETE
```

```
AS  
BEGIN
```

```
DELETE FROM NIETOS WHERE id_hijo1 IN (SELECT id_hijo1 FROM DELETED)  
DELETE FROM HIJOS1 WHERE id_hijo1 IN (SELECT id_hijo1 FROM DELETED)
```

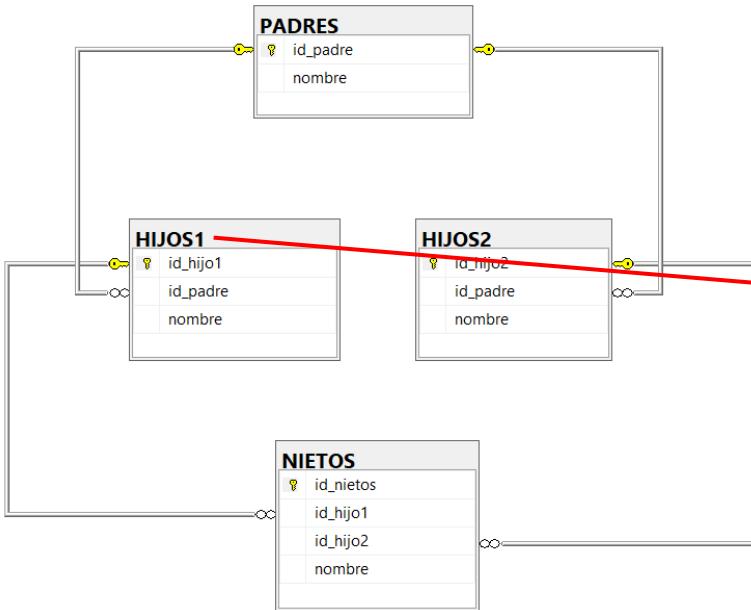
```
END;
```

```
CREATE TRIGGER tgr_eliminar_hijos2  
ON HIJOS2  
INSTEAD OF DELETE
```

```
AS  
BEGIN
```

```
DELETE FROM NIETOS WHERE id_hijo2 IN (SELECT id_hijo2 FROM DELETED)  
DELETE FROM HIJOS2 WHERE id_hijo2 IN (SELECT id_hijo2 FROM DELETED)
```

```
END;
```



```
CREATE TRIGGER tgr_eliminar_padres
  ON PADRES
  INSTEAD OF DELETE
AS
BEGIN

  DELETE FROM HIJOS1 WHERE id_padre IN (SELECT id_padre FROM DELETED)
  DELETE FROM HIJOS2 WHERE id_padre IN (SELECT id_padre FROM DELETED)
  DELETE FROM PADRES WHERE id_padre IN (SELECT id_padre FROM DELETED)

END;

CREATE TRIGGER tgr_eliminar_HIJOS1
  ON HIJOS1
  INSTEAD OF DELETE
AS
BEGIN

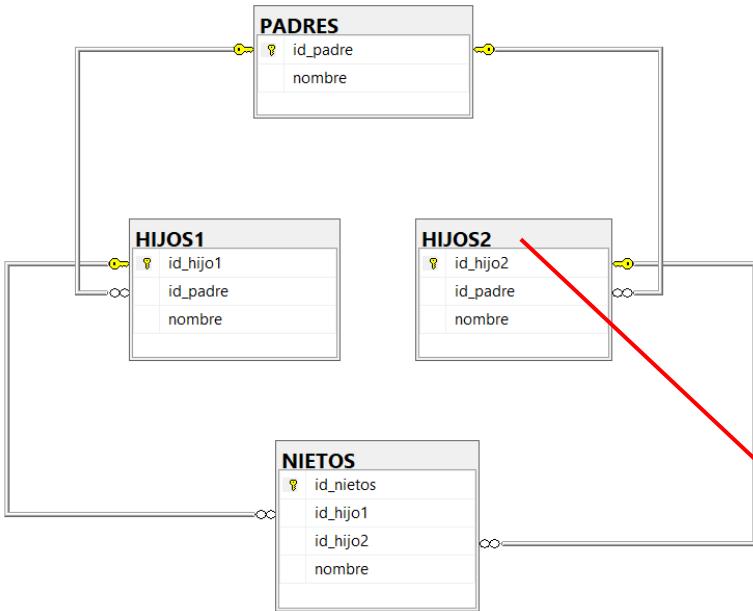
  DELETE FROM NIETOS WHERE id_hijo1 IN (SELECT id_hijo1 FROM DELETED)
  DELETE FROM HIJOS1 WHERE id_hijo1 IN (SELECT id_hijo1 FROM DELETED)

END;

CREATE TRIGGER tgr_eliminar_HIJOS2
  ON HIJOS2
  INSTEAD OF DELETE
AS
BEGIN

  DELETE FROM NIETOS WHERE id_hijo2 IN (SELECT id_hijo2 FROM DELETED)
  DELETE FROM HIJOS2 WHERE id_hijo2 IN (SELECT id_hijo2 FROM DELETED)

END;
```



```
CREATE TRIGGER tgr_eliminar_padres
    ON PADRES
    INSTEAD OF DELETE
AS
BEGIN

    DELETE FROM HIJOS1 WHERE id_padre IN (SELECT id_padre FROM DELETED)
    DELETE FROM HIJOS2 WHERE id_padre IN (SELECT id_padre FROM DELETED)
    DELETE FROM PADRES WHERE id_padre IN (SELECT id_padre FROM DELETED)

END;
```

```
CREATE TRIGGER tgr_eliminar_hijos1
    ON HIJOS1
    INSTEAD OF DELETE
AS
BEGIN

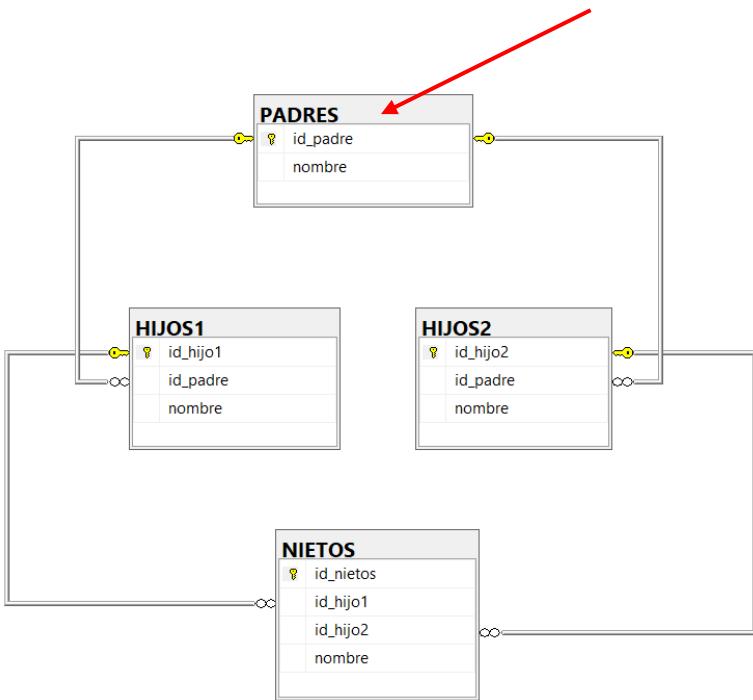
    DELETE FROM NIETOS WHERE id_hijo1 IN (SELECT id_hijo1 FROM DELETED)
    DELETE FROM HIJOS1 WHERE id_hijo1 IN (SELECT id_hijo1 FROM DELETED)

END;
```

```
CREATE TRIGGER tgr_eliminar_hijos2
    ON HIJOS2
    INSTEAD OF DELETE
AS
BEGIN

    DELETE FROM NIETOS WHERE id_hijo2 IN (SELECT id_hijo2 FROM DELETED)
    DELETE FROM HIJOS2 WHERE id_hijo2 IN (SELECT id_hijo2 FROM DELETED)

END;
```



```
CREATE TRIGGER tgr_eliminar_padres  
ON PADRES  
INSTEAD OF DELETE
```

```
AS  
BEGIN
```

```
DELETE FROM HIJOS1 WHERE id_padre IN (SELECT id_padre FROM DELETED)  
DELETE FROM HIJOS2 WHERE id_padre IN (SELECT id_padre FROM DELETED)  
DELETE FROM PADRES WHERE id_padre IN (SELECT id_padre FROM DELETED)
```

```
END;
```

```
CREATE TRIGGER tgr_eliminar_hijos1  
ON HIJOS1  
INSTEAD OF DELETE
```

```
AS  
BEGIN
```

```
DELETE FROM NIETOS WHERE id_hijo1 IN (SELECT id_hijo1 FROM DELETED)  
DELETE FROM HIJOS1 WHERE id_hijo1 IN (SELECT id_hijo1 FROM DELETED)
```

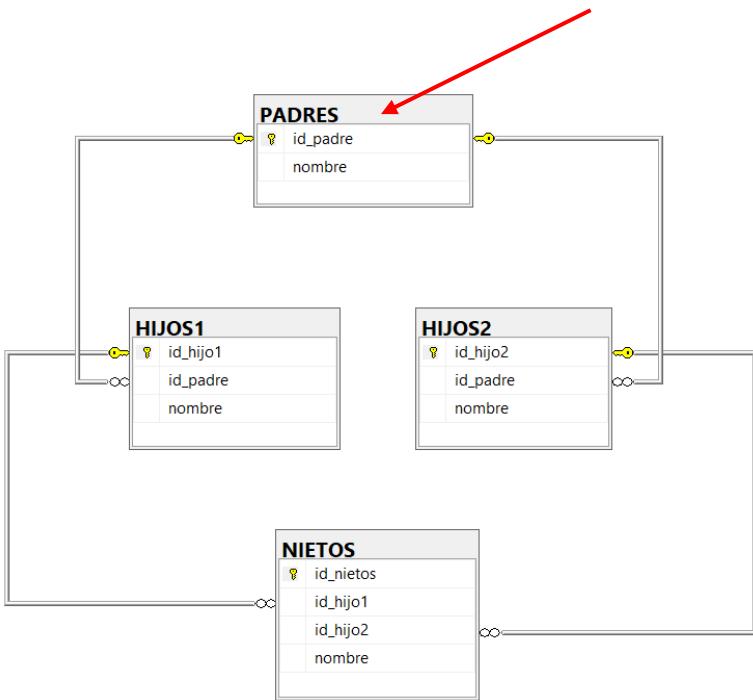
```
END;
```

```
CREATE TRIGGER tgr_eliminar_hijos2  
ON HIJOS2  
INSTEAD OF DELETE
```

```
AS  
BEGIN
```

```
DELETE FROM NIETOS WHERE id_hijo2 IN (SELECT id_hijo2 FROM DELETED)  
DELETE FROM HIJOS2 WHERE id_hijo2 IN (SELECT id_hijo2 FROM DELETED)
```

```
END;
```



```
CREATE TRIGGER tgr_eliminar_padres  
ON PADRES  
INSTEAD OF DELETE
```

```
AS  
BEGIN
```

```
DELETE FROM HIJOS1 WHERE id_padre IN (SELECT id_padre FROM DELETED)  
DELETE FROM HIJOS2 WHERE id_padre IN (SELECT id_padre FROM DELETED)  
DELETE FROM PADRES WHERE id_padre IN (SELECT id_padre FROM DELETED)
```

```
END;
```

---

```
CREATE TRIGGER tgr_eliminar_HIJOS1  
ON HIJOS1  
INSTEAD OF DELETE
```

```
AS  
BEGIN
```

```
DELETE FROM NIETOS WHERE id_hijo1 IN (SELECT id_hijo1 FROM DELETED)  
DELETE FROM HIJOS1 WHERE id_hijo1 IN (SELECT id_hijo1 FROM DELETED)
```

```
END;
```

---

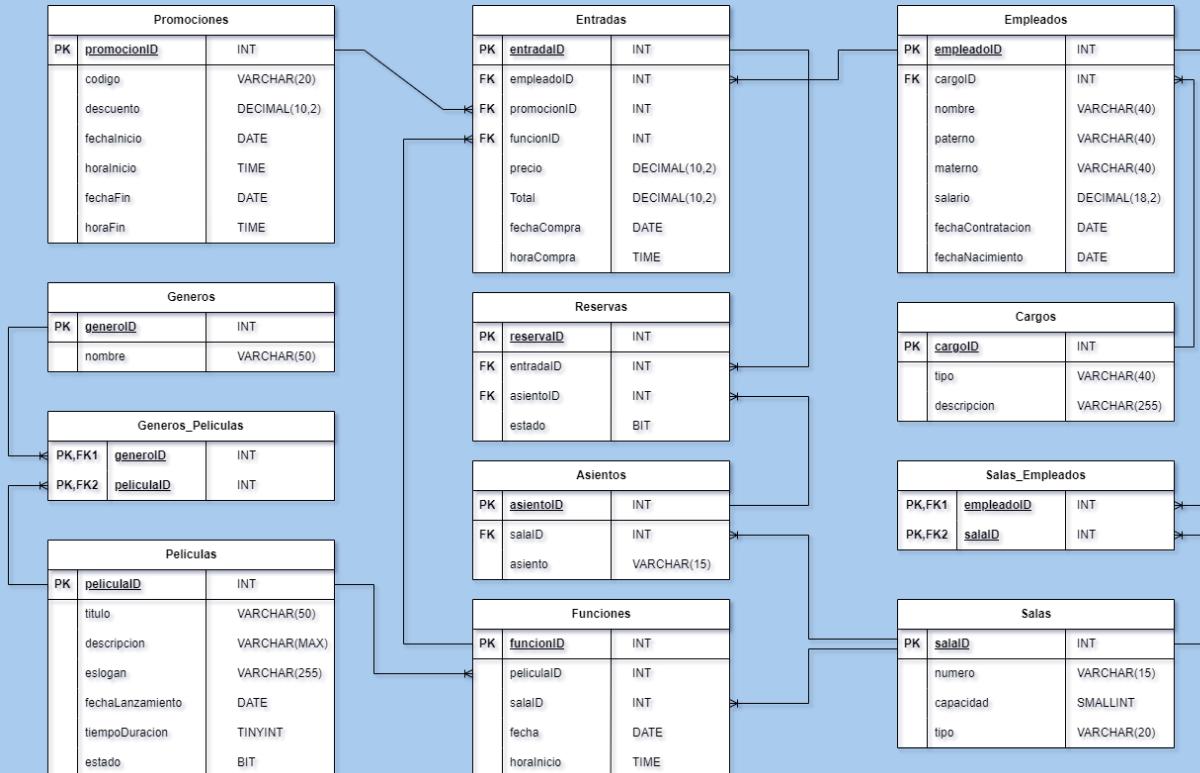
```
CREATE TRIGGER tgr_eliminar_HIJOS2  
ON HIJOS2  
INSTEAD OF DELETE
```

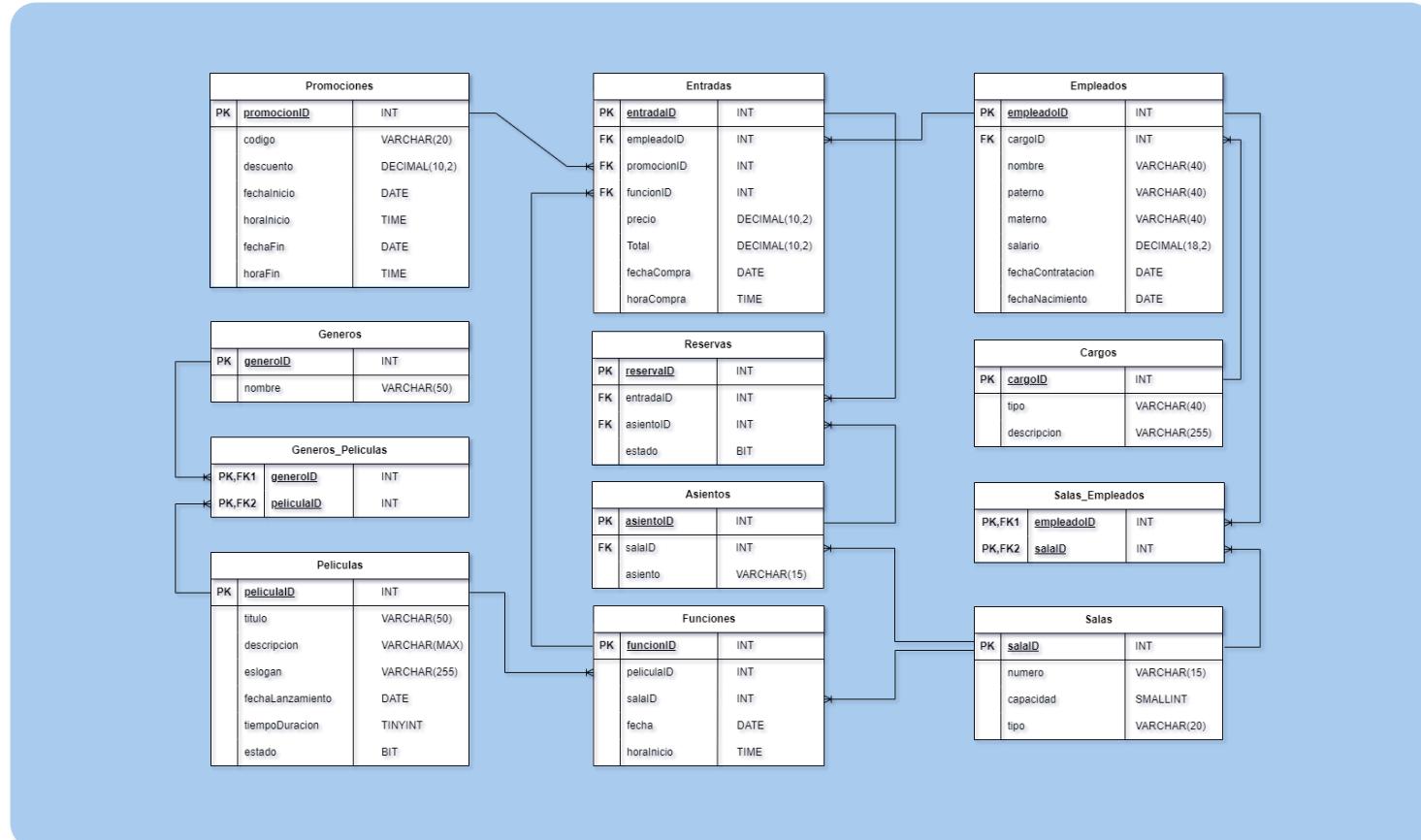
```
AS  
BEGIN
```

```
DELETE FROM NIETOS WHERE id_hijo2 IN (SELECT id_hijo2 FROM DELETED)  
DELETE FROM HIJOS2 WHERE id_hijo2 IN (SELECT id_hijo2 FROM DELETED)
```

```
END;
```

## Implementación de la Base de Datos





# Descripción de la Sección

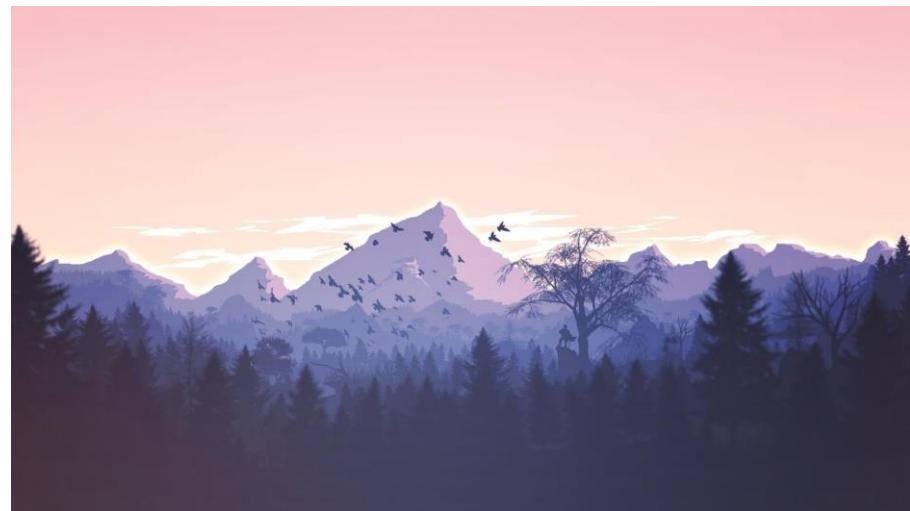
Objetivo de la Sección

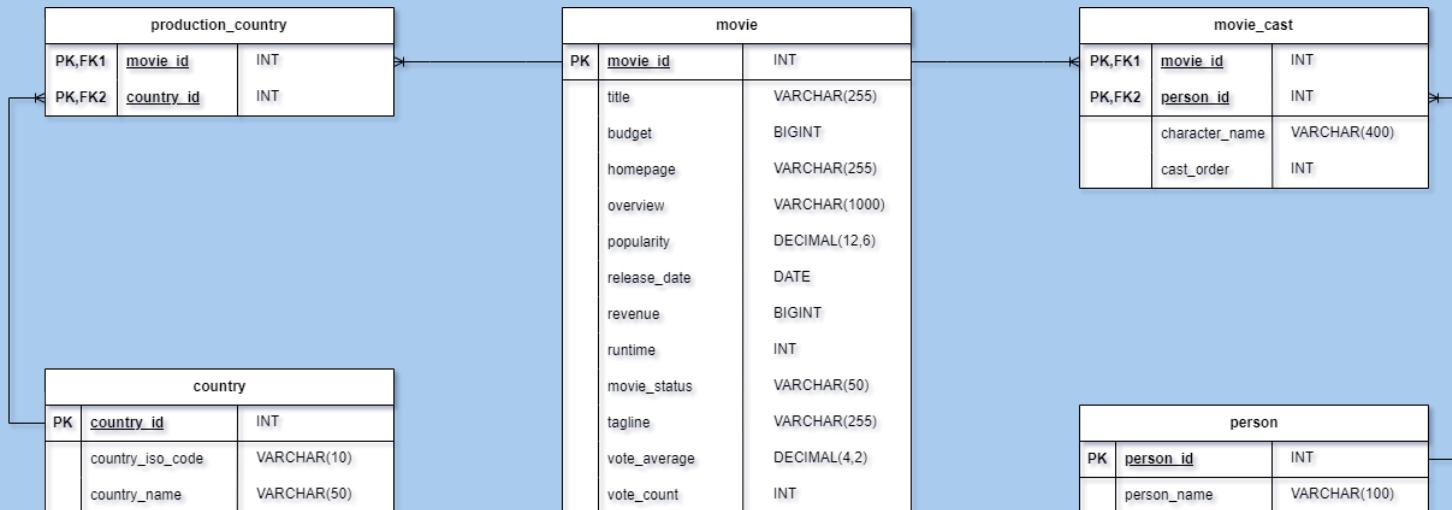
Index(Índice)

Crear Tablas con Registros

Consulta con Clustered Index

Consulta con Non-Clustered Index







**Index(Índice)**

Recuperar datos de una base de datos de una manera más rápida.



El indexar una tabla es sin lugar a dudas, una de las mejores opciones de poder mejorar el rendimiento de las consultas y aplicaciones.

Un índice SQL es una tabla de búsqueda rápida para poder encontrar los registros que los usuarios necesitan buscar con mayor frecuencia. Ya que un índice es pequeño, rápido y optimizado para búsquedas rápidas. Además, que son muy útiles para conectar las tablas relacionales y la búsqueda de tablas grandes.

LIBROS	
PK	<u>ISBN</u>
	titulo
	editorial
	pagina

ISBN = 9783161484100

13 caracteres

13 Bytes

100 Libros = 1 300 Bytes

1000 Libros = 13 000 Bytes

10000 Libros = 130 000 Bytes

70%

LIBROS	
PK	<u>cod_libro</u>
	isbn
	titulo
	editorial
	pagina

Cod\_libro = 101

Número entero

4 Bytes

100 Libros = 400 Bytes

1000 Libros = 4 000 Bytes

10000 Libros = 40 000 Bytes

Oculto al usuario

Mostrar al usuario

# Sin índice

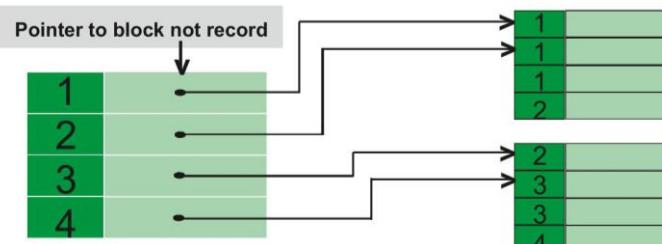
LIBROS				
cod_libro	isbn	titulo	editorial	pagina
103	9988556114250	Don Quijote de la Mancha	Editorial Juventud	312
101	9783161484100	El corazón de la piedra	Hodder Children Book	560
102	9955112035100	Paulo Coelho	Editorial Planeta	208
...	...	...	...	...

Si hacemos una consulta a la tabla, lo que hará es recorrer toda la tabla para devolver todas las filas.

Índice Agrupado  
(CLUSTERED INDEX)

Índice No Agrupado  
(NONCLUSTERED INDEX)

## Índice Agrupado(CLUSTERED INDEX)

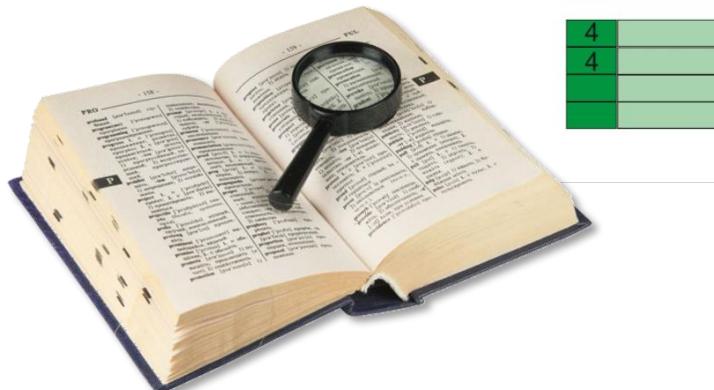


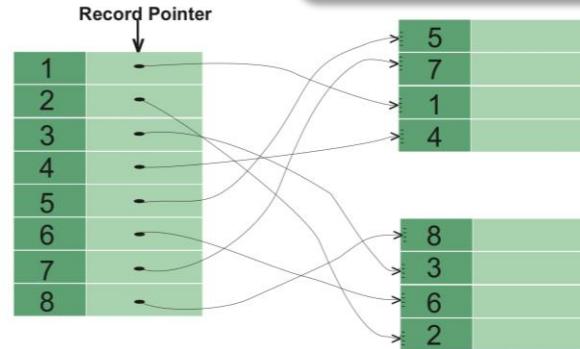
El índice agrupado define el orden en el que se ordenarán y almacenarán los datos de la tabla en el disco.

Cuando define un índice agrupado en una columna, ordenará los datos según los valores de esa columna y los almacenará.

Sólo puede haber un índice agrupado en una tabla.

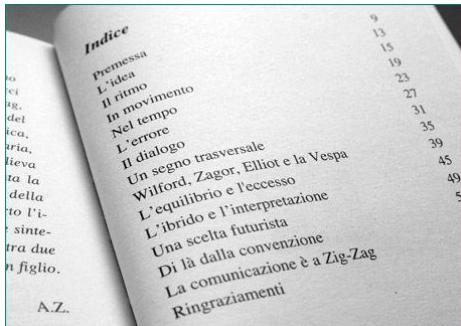
Cuando crea la Llave Primaria en una tabla, se crea automáticamente un índice agrupado único en la tabla.





**El índice no agrupado no ordena físicamente las filas de datos en el disco.**

Crea una estructura clave-valor separada de los datos de la tabla donde la clave contiene los valores de las columnas (en las que se declara un índice no agrupado) y cada valor contiene un puntero a la fila de datos que contiene el valor real.



Puede haber 999 índices no agrupados en una sola tabla.

Cuando crea una restricción UNIQUE, se crea un índice único no agrupado en la tabla.

Instrucción DDL      Tipo de índice      Tipo de Objeto      Nombre

CREATE [ CLUSTERED | **NONCLUSTERED** ] INDEX index\_name  
ON <table\_name> ( <column\_name> [ **ASC** | **DESC** ] )

Tabla      Columna      Orden

Instrucción DDL      Tipo de Objeto      Nombre      Tabla

DROP INDEX index\_name ON <table\_name>

## Factores de un buen rendimiento de Consulta

En la cláusula **SELECT**, utilice únicamente las columnas necesarias. Evite utilizar “\*”.



Use la cláusula **WHERE** para filtrar y devolver solo las filas necesarias.



Trabajar con el Administrador de Base de Datos(DBA) para crear buenos **Índices** para los filtros, joins, ordenamientos, etc.

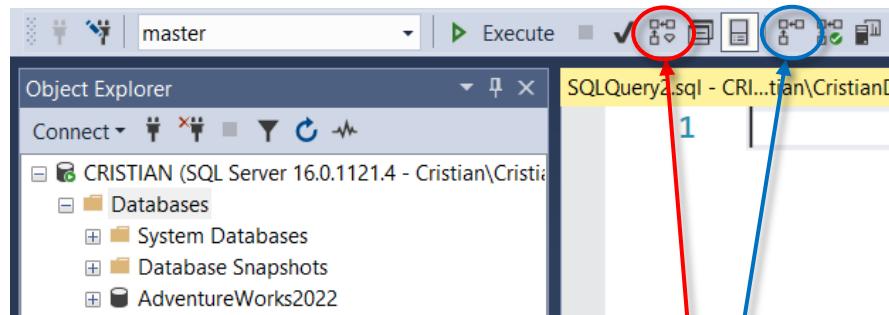


Aprender cómo hacer frente a las tareas con diferentes enfoques de consulta para **comparar el rendimiento**.

## ¿Qué es un Plan de Ejecución?

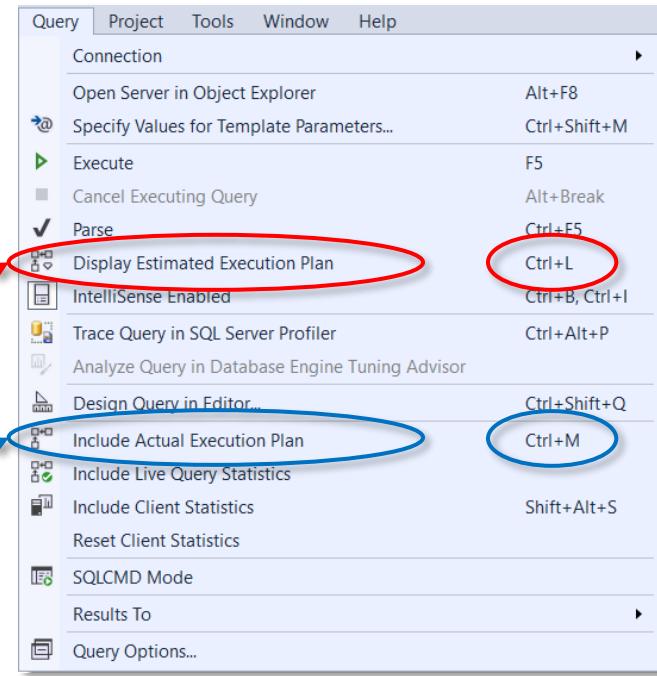
- Un Plan de Ejecución es una representación gráfica de los distintos pasos que intervienen en la obtención de resultados de las tablas de la base de datos. Una vez que se ejecuta una consulta, el motor de procesamiento de consultas genera rápidamente varios planes de ejecución y selecciona el que devuelve los resultados con el mejor rendimiento.

## ¿Qué es un Plan de Ejecución?



Display Estimated Execution Plan

Include Actual Execution Plan



## ¿Qué es un Plan de Ejecución?

```
1 USE NORTHWND
2 GO
3
4 SELECT * FROM Customers
5 WHERE Country = 'Mexico'
6 ORDER BY CompanyName
7 GO
```

144 % ▾

Results Messages Execution plan

Query 1: Query cost (relative to the batch): 100%

SELECT \* FROM [Customers] WHERE [Country]=@1 ORDER BY [CompanyName] ASC

The execution plan diagram illustrates the query's execution flow. It starts with a 'Clustered Index Scan' node, which is a blue rectangle with a circular arrow icon. An arrow points from this node to a 'Sort' node, represented by a blue rectangle with a downward arrow icon. The 'Clustered Index Scan' node has associated statistics: 'Cost: 30 %', '0.000s', '5 of 5 (100%)'. The 'Sort' node also has statistics: 'Cost: 70 %', '0.000s', '5 of 5 (100%)'. A legend at the bottom left identifies the icons: a blue square for 'SELECT Cost: 0 %', a blue rectangle with a circular arrow for 'Clustered Index Scan', and a blue rectangle with a downward arrow for 'Sort'.

## Table Scan



Table Scan  
[Customer]  
Cost: 100 %  
0.000s  
92 of  
92 (100%)



Table Scan  
[Customer]  
Cost: 100 %  
0.000s  
92 of  
92 (100%)

Imaginemos que queremos encontrar un tema específico en un libro sin utilizar el índice. En este caso, tendríamos que revisar cada página del libro de manera secuencial hasta dar con el tema deseado. De manera similar, un Table Scan indica que el motor de base de datos debe leer toda la tabla completa sin utilizar un índice adecuado, ya sea porque no existe o no es útil en ese contexto. Esta operación suele ser menos eficiente la mayor parte del tiempo.

## Clustered Index Scan



Clustered Index Scan (Cluste...  
[Customer].[PK\_Customer]  
Cost: 100 %  
0.000s  
92 of  
92 (100%)

Similar al Table Scan, pero en este caso la tabla tiene un índice Clustered que organiza previamente los datos en un orden específico. Esto significa que ahora se recorren los datos ya ordenados. Sin embargo, a pesar de estar ordenados, sigue siendo un proceso lento recorrer todo el índice.

## NonClustered Index Scan



Index Scan (NonClustered)  
[Customer].[NCIX\_Country]  
Cost: 100 %  
0.000s  
92 of  
92 (100%)

Dado que los datos no pueden ser ordenados físicamente en disco, ya que esto lo ha realizado el índice Clustered, se genera una estructura adicional con las columnas a indexar, y estas columnas se organizan de manera ordenada. Si la consulta solo utiliza esta estructura del índice sin acceder directamente a las páginas de datos, aunque no sea un método perfecto, es más eficiente que un [Table Scan](#) o un [Clustered Index Scan](#).

## Clustered Index Seek



Clustered Index Seek (Clustere...

[Customer].[PK\_CustomerID]

Cost: 100 %

0.000s

1 of

1 (100%)

Este método sí es eficiente. Es similar a buscar en un diccionario que está organizado alfabéticamente. Si queremos encontrar la palabra "Zanahoria", no empezaremos desde la primera página que contiene palabras con la letra "A". En su lugar, iremos directamente hacia el final, donde se encuentran las palabras que empiezan con la letra "Z".

## NonClustered Index Seek



Index Seek (NonClustered)

[Customer].[NIX\_Country]

Cost: 100 %

0.000s

9 of

9 (100%)

Este es el uso ideal de un índice NonClustered. El concepto es parecido a buscar un tema en un libro: acudimos al índice de las primeras páginas para encontrar rápidamente las secciones donde se encuentra la información que necesitamos.

## Stream Aggregate



Stream Aggregate  
(Aggregate)

Cost: 0 %  
0.000s  
21 of  
21 (100%)

## Order By



Sort

Cost: 72 %  
0.000s  
92 of  
92 (100%)

Ocurre cuando organizamos los datos en grupos y utilizamos funciones de agregación como MIN, SUM, AVG, o cuando empleamos la cláusula HAVING.

Cuando el motor de base de datos debe ordenar los resultados por un campo que no está previamente ordenado por un índice, necesita realizar una operación de ordenación, lo que generalmente reduce el rendimiento de la consulta.



## Nested Loops (Inner Join)

Cost: 0 %  
0.000s

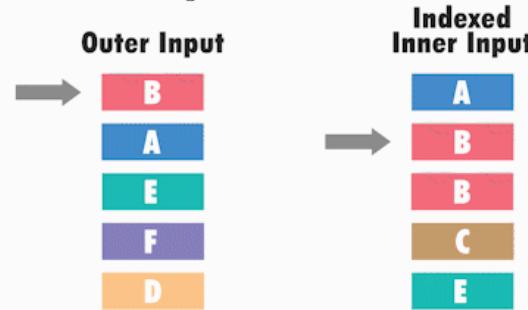
C 9 of  
5 (180%)

## Nested Loop Join



Tenemos dos conjuntos de datos: uno está preordenado y contiene elementos únicos, mientras que el otro no. Usando el conjunto ordenado y de elementos únicos como referencia, se selecciona el primer elemento y se busca una coincidencia en cada elemento del segundo conjunto, recorriendo todos sus elementos porque no está ordenado. Al finalizar, se procede con el segundo elemento del conjunto ordenado. Es posible que se vuelva a encontrar el primer elemento y que ocurra una nueva coincidencia con el segundo, debido a que el conjunto original ya estaba previamente ordenado.

### Nested Loops Join



## Merge Join



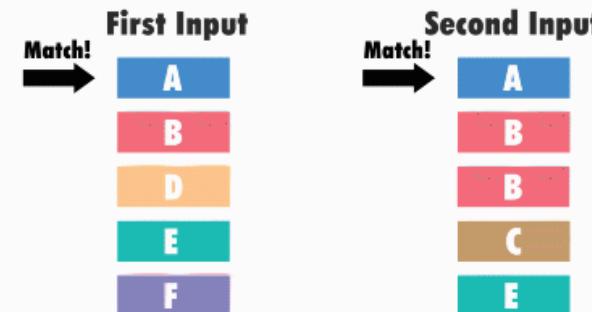
Este operador es el más eficiente de los Joins, ya que ambos conjuntos de datos están ordenados. Esto permite realizar una búsqueda más rápida en el segundo conjunto. Cada elemento del primer conjunto busca coincidir con los ítems del segundo, y una vez que no hay más coincidencias, ya no es necesario recorrer toda la tabla para cada elemento del primer conjunto.



Merge Join  
(Inner Join)

Cost: 46 %  
0.000s  
1 of  
1 (100%)

### Merge Join



## Hash Join



Hash Match  
(Inner Join)

Cost: 33 %

0.000s

9 of  
8 (112%)

Este es el tipo de Join menos eficiente, ya que al no haber un orden válido a través de índices en ninguna de las dos tablas o conjuntos de datos (es decir, no hay índices en ninguna tabla), el sistema debe crear una tabla temporal ordenada para reemplazar uno de los conjuntos y luego simular un Join similar al Nested Loop Join.

### Hash Match Join

#### Build Input      Probe Input

B	A
A	B
E	E
F	B
B	F

Hash Function

## Estadísticas de Distribución

- Las estadísticas de distribución describen la distribución y la singularidad, de la selectividad, de los datos.
- Las estadísticas de distribución de forma predeterminada, se crean y se actualizan automáticamente.
- Las estadísticas son utilizadas por el optimizador de consultas para estimar la selectividad de los datos, incluyendo el tamaño de los resultados.
- Las grandes variaciones entre los valores estimados y los reales podrían indicar un problema con las estimaciones, que pueden ser abordados a través de estadísticas de actualización.

# Descripción de la Sección

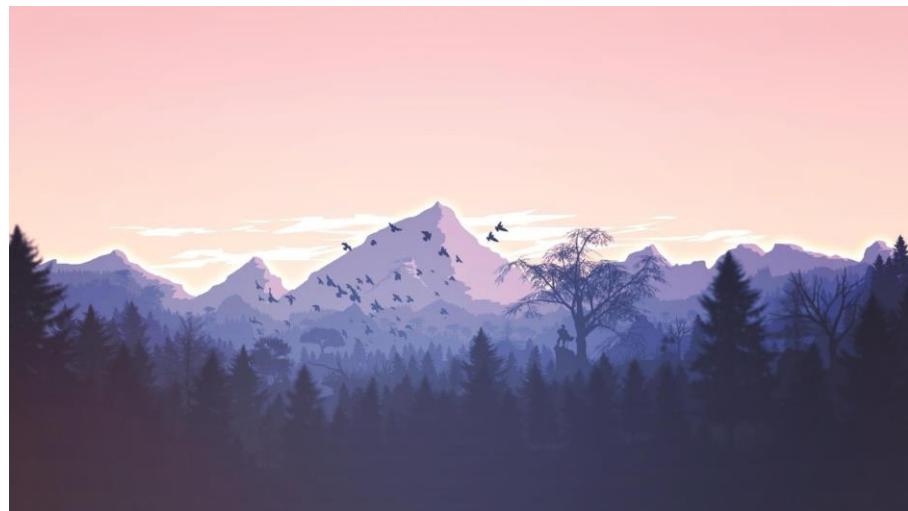
Descripción - Sentencias TCL

Aplicar Transaction(Transacción)

Transacciones con Procedimiento Almacenado

Transacciones con Manejo de Control de Errores

Comando NOLOCK

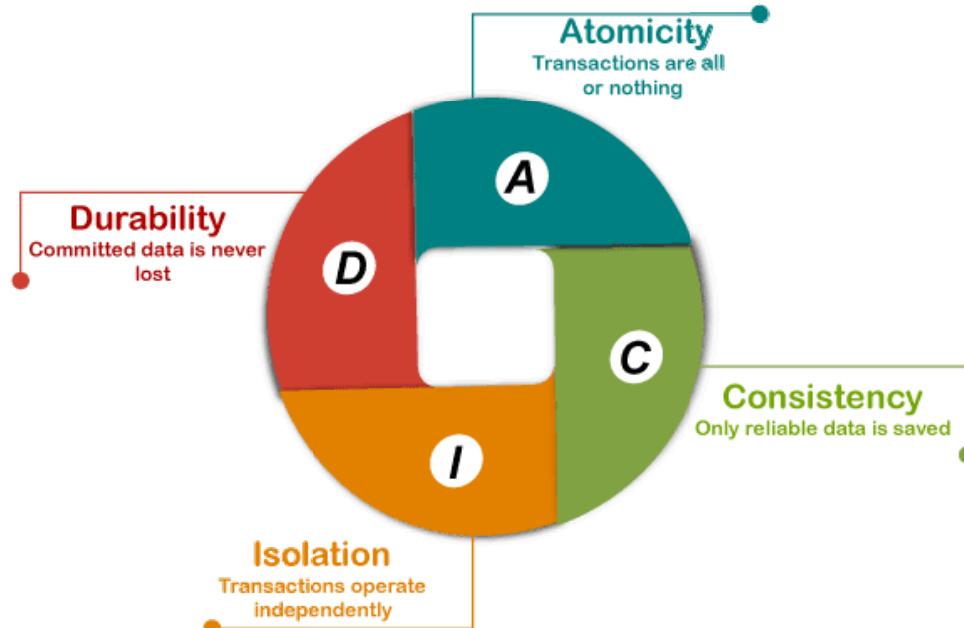


## Transaction(Transacción) - Sentencias TCL

Las sentencias TCL (del inglés: "Transaction Control Language" o en español: "Lenguaje de control de transacciones") es una parte de SQL que se utiliza para gestionar transacciones y garantizar la integridad de los datos en una base de datos.

Una transacción es una unidad lógica de trabajo que consta de una o más operaciones de base de datos que se ejecutan como un todo, lo que significa que todas deben completarse correctamente o, en caso contrario, ninguna de ellas se aplica.

Usar la Transacción es el mejor método debido a que cumple con las cuatro propiedades de todo Sistema de Bases de Datos Relacional, a estas propiedades de les conoce como ACID(atomicity, consistency, isolation y durability)



Algunas de las principales sentencias **TCL** en **SQL** son:

BEGIN TRANSACTION[TRAN]

COMMIT [TRANSACTION | TRAN]

ROLLBACK [TRANSACTION | TRAN]

## Transferencia Bancaria



Envía Dinero

Transacción

Recibe Dinero

## Transferencia Bancaria

BEGIN TRANSACTION;  Inicio de la Transacción

BEGIN TRY

○ UPDATE Cuentas  
SET Saldo = Saldo - 100  
WHERE CuentaID = 1;

Deduces el monto de la cuenta origen

○ UPDATE Cuentas  
SET Saldo = Saldo + 100  
WHERE CuentaID = 2;

Agrega el monto a la cuenta destino



○ COMMIT TRANSACTION;

PRINT 'Transferencia completada exitosamente.';

Si todo va bien,

confirmamos la transacción

END TRY

BEGIN CATCH

ROLLBACK TRANSACTION;

PRINT 'Error en la transferencia. Se han revertido los cambios.';

END CATCH;

Si hay un error,

revertimos la transacción

## Transferencia Bancaria

BEGIN TRANSACTION;  Inicio de la Transacción

BEGIN TRY

    UPDATE Cuentas

    SET Saldo = Saldo - 100  
    WHERE CuentaID = 1;

    UPDATE Cuentas

    SET Saldo = Saldo + 100  
    WHERE CuentaID = 2;

    COMMIT TRANSACTION;

    PRINT 'Transferencia completada exitosamente.';

END TRY

BEGIN CATCH

    ROLLBACK TRANSACTION;

    PRINT 'Error en la transferencia. Se han revertido los cambios.';

END CATCH;



### Corte de energía

### Desastre natural

### Problema de hardware

Si hay un error,  
revertimos la transacción

- Las transacciones permiten ejecutar un grupo de operaciones de manera segura y coherente.
- • Garantizan la integridad de los datos mediante las propiedades ACID.
- Se controlan mediante las sentencias **BEGIN**, **COMMIT** y **ROLLBACK**.
- • Es importante manejar errores adecuadamente para evitar estados inconsistentes en la base de datos.

Las transacciones son esenciales en sistemas de bases de datos para garantizar que las operaciones críticas se ejecuten correctamente o se reviertan en caso de fallo.

# Descripción de la Sección

Descripción - Sentencias DCL

Habilitar el usuario “sa”

Logins

Eliminar un Login bloqueado

Roles a Nivel de Servidor

Roles a Nivel de Servidor Personalizado

Users

Roles a Nivel de Base de Datos

Roles a Nivel de Base de Datos Personalizados



# Sentencias DCL



# Sentencias DCL

En SQL Server, DCL (Data Control Language o Lenguaje de Control de Datos) es un conjunto de instrucciones SQL utilizadas para controlar el acceso a los datos dentro de una base de datos. Su principal objetivo es gestionar los permisos y privilegios que tienen los usuarios o roles para realizar ciertas acciones en los objetos de la base de datos, como tablas, vistas, procedimientos almacenados, etc.



# Sentencias DCL

## GRANT

Otorga permisos específicos a un usuario o rol para realizar acciones en la base de datos.

## REVOKE

Revoca o quita permisos previamente otorgados a un usuario o rol.

## DENY

Niega explícitamente permisos a un usuario o rol, incluso si el permiso fue otorgado anteriormente.

# Sentencias DCL

## Consideraciones:

### Seguridad

Las instrucciones DCL son fundamentales para garantizar la seguridad de la base de datos, ya que limitan qué acciones puede realizar cada usuario o rol.

### Jerarquía

La instrucción DENY tiene prioridad sobre GRANT. Si un permiso es negado con DENY, el usuario no podrá realizar la acción, aunque tenga permisos otorgados.

# Sentencias DCL

Las instrucciones DCL son esenciales para el control de acceso y la administración de la seguridad en SQL Server, asegurando que solo los usuarios autorizados puedan interactuar con los datos de manera adecuada.

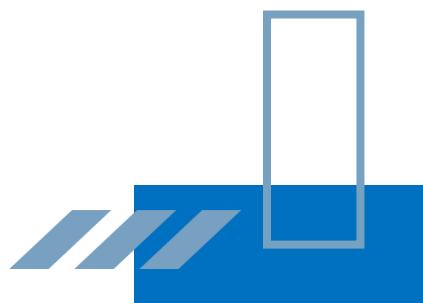


# Usuario “sa”



# Usuario "sa"

El usuario **sa** se refiere al "**System Administrator**". Es un usuario especial y predeterminado que tiene todos los privilegios y permisos posibles en el servidor. Este usuario suele ser creado opcionalmente durante la instalación de SQL Server y se utiliza principalmente para administrar y configurar la instancia de SQL Server. Por ser un usuario con tantos privilegios, se recomienda usarlo con precaución y limitar su acceso cuando sea posible, por motivos de seguridad.



# Logins



# Logins

En SQL Server, un **Login** es una credencial de acceso que permite a un usuario autenticarse en la instancia del servidor de base de datos. Es el primer paso para establecer una conexión con **SQL Server**, y una vez que un usuario se autentica con su **Login**, puede acceder a las bases de datos dentro de esa instancia, siempre que también tenga permisos otorgados a nivel de base de datos.

# Logins

Windows Authentication

SQL Server Authentication

Microsoft Entra MFA

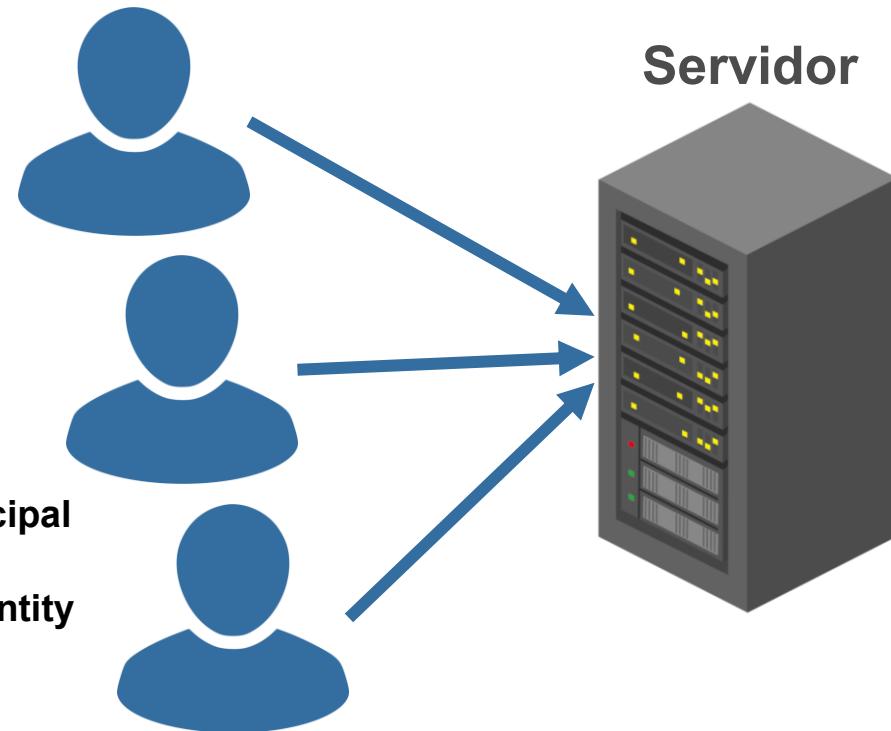
Microsoft Entra Password

Microsoft Entra Integrated

Microsoft Entra Service Principal

Microsoft Entra Managed Identity

Microsoft Entra Default



# Logins

## Sintaxis:

```
USE master;
```

```
CREATE LOGIN login_name WITH PASSWORD = 'password'
```

```
[ MUST_CHANGE ],
```

```
[ DEFAULT_DATABASE = database ],
```

```
[ DEFAULT_LANGUAGE = language ],
```

```
[ CHECK_EXPIRATION = { ON | OFF } ],
```

```
[ CHECK_POLICY = { ON | OFF } ]
```

```
DROP LOGIN login_name;
```

Si la directiva de Windows requiere contraseñas seguras, las contraseñas deben tener al menos tres de las cuatro siguientes características:

- Un carácter en mayúscula (A-Z).
- Un carácter en minúsculas (a-z).
- Un dígito (0-9).
- Uno de los caracteres especiales, como un espacio, \_, @, \*, ^, %, !, \$, #, o &.

# Roles a Nivel de Servidor



# Roles a Nivel de Servidor

SQL Server proporciona roles a nivel de servidor para ayudarlo a administrar los permisos en un servidor. Estos roles son entidades de seguridad que agrupan a otras entidades. Los roles a nivel de servidor abarcan todo el servidor en cuanto a su alcance de permisos.

# Roles a Nivel de Servidor

En la versión de SQL Server 2019 y versiones anteriores se contaban con nueve roles a nivel de servidor.

## sysadmin

- Es el rol con “**mayores privilegios**” en SQL Server.
- Los miembros del rol ‘**sysadmin**’ tienen “**control total**” sobre la instancia de SQL Server. Pueden realizar cualquier operación en el servidor, incluyendo la administración de bases de datos, configuración del servidor y seguridad.

## serveradmin

- Los miembros de este rol pueden “**configurar las opciones del servidor**” y realizar tareas administrativas relacionadas con el servidor, como cambiar las opciones de configuración globales del servidor.
- También pueden iniciar y detener el servidor SQL Server.

## setupadmin

- Los miembros del rol ‘**setupadmin**’ pueden “**administrar los enlaces de servidores remotos**” y “**configurar**” opciones del servidor.
- Este rol tiene permisos para “**agregar y eliminar servidores vinculados**”, que permiten la ejecución de consultas distribuidas entre servidores SQL.

# Roles a Nivel de Servidor

En la versión de SQL Server 2019 y versiones anteriores se contaban con nueve roles a nivel de servidor.

## securityadmin

- Los miembros de este rol pueden “**administrar la seguridad a nivel de servidor**”: pueden crear, modificar y eliminar logins.
- También pueden otorgar y revocar permisos a nivel de servidor y establecer permisos para los usuarios en las bases de datos.

## processadmin

- Los miembros del rol ‘**processadmin**’ pueden “**administrar los procesos que se ejecutan en SQL Server**”, lo que incluye la capacidad de “**ver y terminar procesos**” o conexiones activas.
- Pueden detener cualquier tarea que esté bloqueando otros procesos o consumiendo demasiados recursos.

## dbcreator

- Los miembros de este rol pueden “**crear, modificar, eliminar y restaurar bases de datos**” en la instancia de SQL Server.
- Tienen privilegios para realizar tareas relacionadas con la creación y administración de bases de datos, pero no necesariamente pueden administrar la seguridad de los usuarios de la base de datos.

# Roles a Nivel de Servidor

En la versión de SQL Server 2019 y versiones anteriores se contaban con nueve roles a nivel de servidor.

## bulkadmin

- Los miembros del rol 'bulkadmin' tienen permisos para ejecutar operaciones de “**importación masiva de datos**” (Bulk Insert) en SQL Server.
- Este rol es útil cuando se necesitan insertar grandes cantidades de datos en tablas de SQL Server mediante el comando '**BULK INSERT**'.

## diskadmin

- El rol '**diskadmin**' en SQL Server es responsable de la “**administración de los discos físicos**” en los que se almacenan los archivos de bases de datos.
- Los miembros de este rol pueden realizar tareas relacionadas con la gestión de los archivos del disco que utiliza SQL Server, como agregar, quitar o modificar unidades de disco para almacenamiento.

## public

- El rol '**public**' es un rol especial y está “**disponible para todos los usuarios**” de SQL Server de manera predeterminada. “**Todos los logins**” que acceden a SQL Server automáticamente son miembros del rol '**public**'.
- El rol '**public**' no se puede eliminar ni quitar de ningún login o usuario. Su función es proporcionar un conjunto básico de permisos que todos los usuarios tienen por defecto.

# Roles a Nivel de Servidor

En la versión de SQL Server 2022 se han añadido 10 roles de servidor adicionales que se diseñaron específicamente teniendo en cuenta el principio de privilegio mínimo, que tienen el prefijo **##MS\_** y el sufijo **##** para distinguirlos de otros principales creados por el usuario y roles de servidor personalizados. Esos nuevos roles contienen privilegios que se aplican en el ámbito del servidor, pero también pueden heredarse hasta bases de datos individuales (excepto el rol de servidor **##MS\_LoginManager##**).

**##MS\_DatabaseConnector##**

**##MS\_DatabaseManager##**

**##MS\_DefinitionReader##**

**##MS\_LoginManager##**

**##MS\_PerformanceDefinitionReader##**

**##MS\_SecurityDefinitionReader##**

**##MS\_ServerPerformanceStateReader##**

**##MS\_ServerSecurityStateReader##**

**##MS\_ServerStateManager##**

**##MS\_ServerStateReader##**

# Roles a Nivel de Servidor

## Sintaxis:

```
USE master;
```

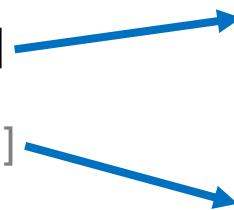
```
ALTER SERVER ROLE server_role_name
```

```
{
```

```
[ ADD MEMBER login ] |
```

Agrega la entidad de seguridad del servidor especificada al rol de servidor. Login es el nombre de inicio de sesión.

```
[ DROP MEMBER login ]
```



Quita la entidad de seguridad del servidor especificada del rol de servidor. Login es un nombre de inicio de sesión.

```
}
```

# Roles a Nivel de Servidor

Login



Servidor



Role



Base de Datos



# Role a Nivel de Servidor Personalizado



# Role a Nivel de Servidor Personalizado

Los roles a nivel de servidor personalizados en SQL Server son roles definidos por el administrador de la base de datos que permiten agrupar y asignar permisos a usuarios o inicios de sesión de acuerdo con las necesidades específicas de la organización o sistema. Estos roles permiten una mayor flexibilidad en la administración de permisos, ya que permiten definir conjuntos de privilegios que no están cubiertos por los roles de servidor predefinidos de SQL Server.

Este tipo de roles personalizados se utiliza para simplificar la administración de permisos cuando necesitas que varios usuarios tengan un conjunto específico de privilegios en el servidor, pero no quieres asignar esos permisos uno por uno.

# Role a Nivel de Servidor Personalizado

## Características:

### Control de permisos

Te permite agrupar permisos específicos para que se asignen a múltiples usuarios de manera eficiente.

### Seguridad

Puedes definir roles con permisos más ajustados a las necesidades de seguridad de tu organización, sin necesidad de otorgar permisos excesivos que podrían poner en riesgo el sistema.

### Facilidad de administración

Al asignar permisos mediante roles, puedes modificar o revocar permisos fácilmente a través del rol en lugar de hacerlo para cada usuario individual.

# Role a Nivel de Servidor Personalizado

## Sintaxis:

```
USE master;
```

```
CREATE SERVER ROLE ServerRoleNameCustom
```

```
ALTER SERVER ROLE [RoleName | Permission ]
```

```
ADD MEMBER ServerRoleNameCustom
```

```
DROP SERVER ROLE ServerRoleNameCustom
```

# Users



# Users

Los **usuarios (users)** son entidades de seguridad que representan cuentas individuales que pueden acceder a una base de datos. Los usuarios se asocian a cuentas de inicio de sesión (**logins**) que permiten la autenticación en el servidor de base de datos. Mientras que los **logins** controlan el acceso al servidor, los **users** controlan el acceso y los permisos dentro de una base de datos específica.

# Roles a Nivel de Servidor

Login



Servidor



Role / User



Base de Datos



# Users

## Sintaxis:

```
USE DataBaseName;
```

```
CREATE USER user_name FOR LOGIN = login_name
```

```
[ DEFAULT_SCHEMA = schema_name ]
```

```
DROP USER user_name;
```

# Roles a Nivel de Base de Datos



# Roles a Nivel de Base de Datos

En SQL Server, los roles a nivel de base de datos son un conjunto de permisos predefinidos que se pueden asignar a los usuarios para gestionar su acceso y las operaciones que pueden realizar en una base de datos específica. Los roles facilitan la administración de permisos, ya que en lugar de asignar permisos de manera individual a cada usuario, puedes agregar usuarios a un rol que ya tenga los permisos necesarios.

# Roles a Nivel de Base de Datos

SQL Server consta de 10 roles a Nivel de Base de datos

## **db\_owner**

Tiene control completo sobre la base de datos. Los miembros de este rol pueden realizar cualquier tarea dentro de la base de datos, incluida la modificación de los permisos de otros usuarios.

## **db\_securityadmin**

Puede modificar los roles y permisos a nivel de base de datos. No tiene todos los permisos que tiene **db\_owner**, pero es el rol responsable de gestionar la seguridad de la base de datos.

## **db\_accessadmin**

Puede agregar o eliminar acceso a la base de datos, es decir, asignar usuarios a la base de datos.

## **db\_backupoperator**

Tiene permisos para realizar respaldos de la base de datos, pero no puede restaurar.

# Roles a Nivel de Base de Datos

SQL Server consta de 10 roles a Nivel de Base de datos

## **db\_datareader**

Permite leer todos los datos de todas las tablas y vistas dentro de la base de datos.

## **db\_datawriter**

Permite modificar los datos (INSERT, UPDATE, DELETE) en todas las tablas y vistas de la base de datos.

## **db\_denydatareader**

Deniega explícitamente permisos de lectura en todas las tablas y vistas.

## **db\_denydatawriter**

Deniega explícitamente permisos de escritura en todas las tablas y vistas.

# Roles a Nivel de Base de Datos

SQL Server consta de 10 roles a Nivel de Base de datos

## db\_ddladmin

Permite ejecutar comandos **DDL (Data Definition Language)**, como CREATE, ALTER, DROP sobre los objetos de la base de datos (tablas, vistas, procedimientos, etc.).

## public

Es un rol a nivel de base de datos especial que incluye a todos los usuarios de una base de datos. Por defecto, el rol public tiene permisos mínimos, generalmente solo de lectura de ciertos metadatos.

# Roles a Nivel de Base de Datos

## Sintaxis:

```
USE DataBaseName;
```

```
ALTER ROLE database_role_name
```

```
{
```

```
[ ADD MEMBER user_name ] |
```

Agregue la entidad de seguridad de base de datos a la pertenencia de un rol de base de datos.

```
[ DROP MEMBER user_name ]
```

Quita la entidad de seguridad de base de datos de la pertenencia de un rol de base de datos.

```
}
```

# Roles a Nivel de Base de Datos Personalizado



# Roles a Nivel de Base de Datos Personalizado

En SQL Server, los roles a nivel de base de datos personalizados son roles que puedes crear manualmente para organizar y gestionar los permisos de los usuarios de forma más flexible y eficiente. Estos roles son muy útiles cuando los roles fijos de base de datos no se ajustan exactamente a tus necesidades, ya que te permiten agrupar permisos específicos y asignarlos a usuarios o grupos.

# Roles a Nivel de Base de Datos Personalizado

## Ventajas:

### Control Granular de Permisos

Puedes asignar permisos específicos a los roles personalizados según las necesidades de la aplicación o del equipo.

### Facilidad de Gestión

En lugar de asignar permisos individualmente a cada usuario, puedes agrupar permisos en roles personalizados y luego asignar usuarios a esos roles, simplificando la administración.

### Reusabilidad

Puedes crear roles personalizados que se ajusten a un perfil común de usuario y asignarlos a múltiples usuarios.

# Roles a Nivel de Base de Datos Personalizado

## Sintaxis:

```
USE DataBaseName;
```

```
CREATE ROLE DataBaseRoleNameCustom
```

```
[ GRANT | DENY | REVOKE ] [ Permission ]
```

```
TO DataBaseRoleNameCustom
```

```
DROP ROLE DataBaseRoleNameCustom
```

## Descripción de la Sección

¿Qué es un Backup?

Estrategias de Backups

Hacer “Backups” a una Base de Datos

Realizar “Restore” a una Base de Datos



# ¿Qué es un Backup?



# ¿Qué es un Backup?

Un backup (o copia de seguridad) es una copia exacta de los datos que se realiza para proteger y preservar la información en caso de pérdida, daño, corrupción o eliminación accidental de los datos originales. El propósito principal de un backup es garantizar la recuperación de la información en caso de fallos, errores humanos, ataques cibernéticos, o desastres.

En el contexto de SQL Server (y otros sistemas de gestión de bases de datos), los backups son fundamentales para mantener la disponibilidad y seguridad de las bases de datos, permitiendo restaurarlas en casos de emergencia o mantenimiento.

# ¿Qué es un Backup?

## Características:

### Copia de datos

Un backup guarda los datos de un sistema, aplicación o base de datos en un lugar seguro, que puede ser un disco duro, una nube o un servidor externo.

### Seguridad

Los backups suelen implementarse con medidas de seguridad adicionales, como cifrado, para proteger los datos sensibles.

### Recuperación de datos

Permite restaurar parcial o totalmente los datos a partir del backup en caso de que la información original esté inaccesible o dañada.

# ¿Qué es un Backup?

## Ventajas:

### Protección ante pérdida de datos

Minimiza el impacto de fallos de hardware, ataques de malware o errores humanos.

### Continuidad del negocio

En caso de desastres, los backups permiten que las empresas sigan funcionando sin una pérdida significativa de datos.

### Flexibilidad en la restauración

Dependiendo del tipo de backup, se puede restaurar el sistema completo o partes específicas de los datos.

# ¿Qué es un Backup?

## Tipos:

**Full Backup**  
(Copia de seguridad Completa)

**Differential Backup**  
(Copia de seguridad Diferencial)

**Transaction log Backup**  
(Copia de seguridad de Registro de Transacción)

**Copy-Only Backup**  
(Copia de seguridad de solo copia)

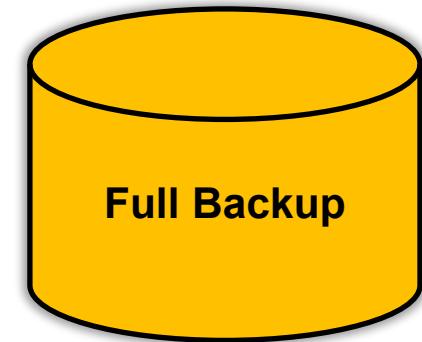
**Filegroup Backup**  
(Copia de seguridad de Grupo de Archivo)

**Tail Log Backup**  
(Copia de seguridad de la Cola de Registro)

# ¿Qué es un Backup?

## Full Backup (Copia de seguridad Completa)

Una copia de seguridad completa(Full Backup), como su nombre lo indica, respalda todo. Es la base de cualquier tipo de copia de seguridad. Se trata de una copia completa, que almacena todos los objetos de la base de datos: [tablas](#), [procedimientos](#), [funciones](#), [vistas](#), [índices](#), [etc](#). Con una copia de seguridad completa, podrá restaurar fácilmente una base de datos exactamente en la misma forma en que estaba en el momento de la copia de seguridad.



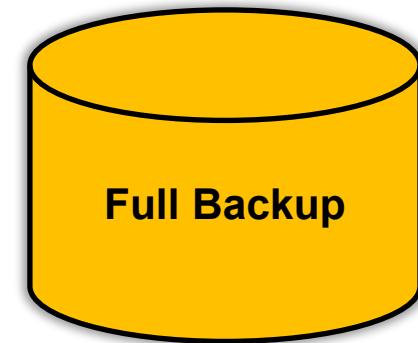
Se debe realizar [una copia de seguridad completa al menos una vez](#) antes de poder ejecutar cualquier otro tipo de copia de seguridad: esta es la base para cualquier otro tipo de copia de seguridad.

# ¿Qué es un Backup?

**Full Backup**  
(Copia de seguridad Completa)

## Sintaxis:

```
BACKUP DATABASE [database_name]  
TO DISK = 'path'  
WITH [ options ];
```



# ¿Qué es un Backup?

## Differential Backup (Copia de seguridad Diferencial)

Una copia de seguridad diferencial de la base de datos(Differential Backup) es el superconjunto de la **última** copia de seguridad completa(**Full Backup**) y contiene todos los **cambios** que se han realizado desde la **última** copia de seguridad completa(**Full Backup**).

Como una copia de seguridad diferencial no realiza una copia de seguridad de todo , **la copia de seguridad suele ejecutarse más rápido que una copia de seguridad completa**. Una copia de seguridad diferencial de la base de datos captura el estado de las extensiones modificadas en el momento en que se creó la copia de seguridad. Si crea una serie de copias de seguridad diferenciales, es probable que una base de datos que se actualiza con frecuencia **contenga datos diferentes en cada diferencial**.

Las copias de seguridad diferenciales **ahorran espacio de almacenamiento** y el **tiempo que lleva realizar una copia de seguridad**. Sin embargo, a medida que los datos cambian con el tiempo, el **tamaño de la copia de seguridad diferencial** también **aumenta**.

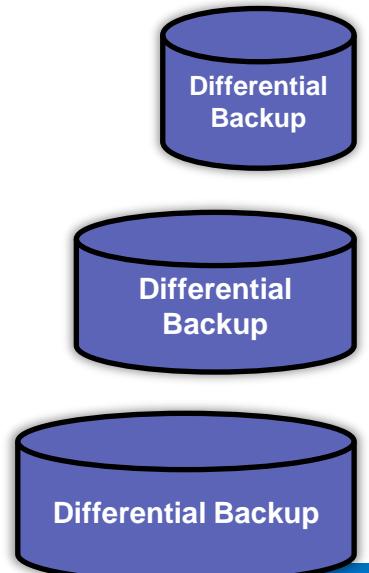


# ¿Qué es un Backup?

**Differential Backup**  
(Copia de seguridad Diferencial)

## Sintaxis:

```
BACKUP DATABASE [database_name]  
TO DISK = 'path'  
WITH DIFFERENTIAL,  
[ options ];
```



# ¿Qué es un Backup?

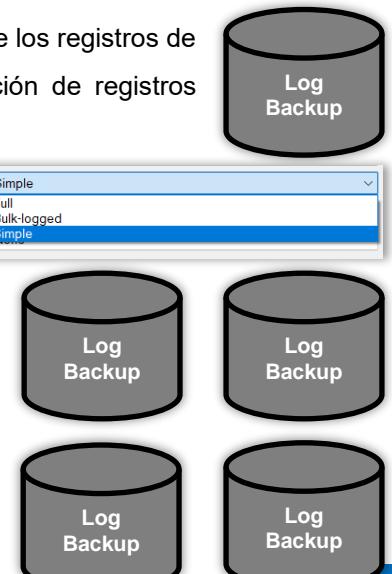
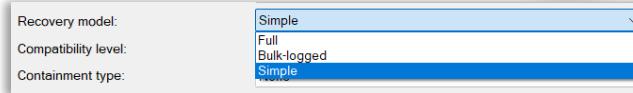
## Transaction log Backup (Copia de seguridad de Log de Transacción)

La copia de seguridad de registros, como su nombre lo indica, realiza una copia de seguridad de los registros de transacciones. Este tipo de copia de seguridad solo es posible con modelos de recuperación de registros completos o masivos.

Este tipo de backup es esencial en bases de datos que utilizan el modelo de recuperación completa o bulk-logged, ya que permite restaurar la base de datos hasta un punto específico en el tiempo.

Un archivo de registro de transacciones almacena una serie de registros que proporcionan el historial de cada modificación de datos en una base de datos.

No se puede hacer un backup del log de transacciones sin haber hecho antes un backup completo de la base de datos. El log backup depende del último full backup para restaurar completamente la base de datos.



# ¿Qué es un Backup?

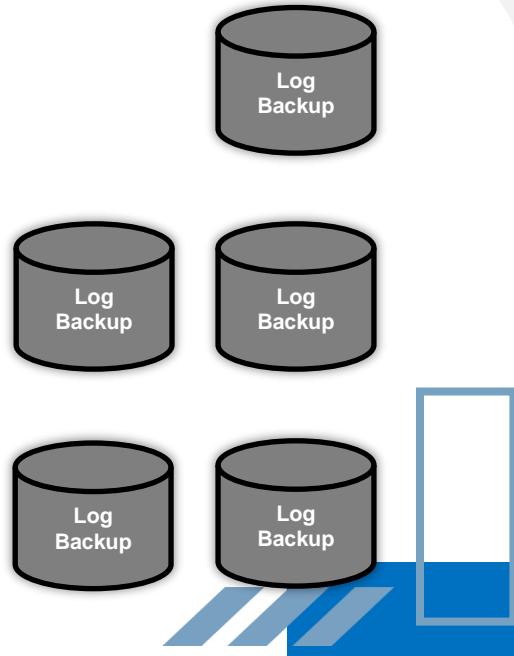
**Transaction log Backup**  
(Copia de seguridad de Log de Transacción)

## Sintaxis:

`BACKUP LOG [database_name]`

`TO DISK = 'path'`

`WITH [ options ];`



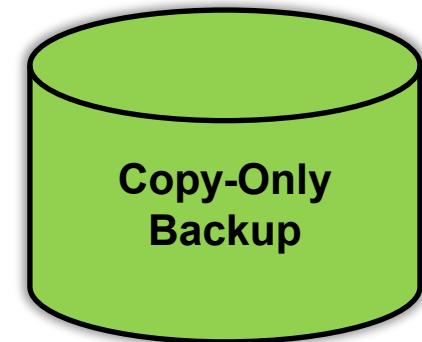
# ¿Qué es un Backup?

## Copy-Only Backup (Copia de seguridad de solo copia)

El **Copy-Only Backup** es un tipo **especial** de backup en SQL Server que se realiza sin afectar la secuencia normal de backups. Esencialmente, es una **copia de seguridad independiente** que **no interfiere** con la estrategia regular de **backups**, lo que significa que no altera los puntos de restauración ni la capacidad de realizar otros tipos de backups (como diferenciales o de transacciones). Se utiliza cuando se necesita hacer una copia de seguridad **temporal** o **puntual**, pero sin alterar el historial de **backups** de la base de datos.

Si necesitamos una copia temporal de la base de datos para moverla o probarla en otro entorno, un **copy-only backup** te permite hacerlo sin **afectar** la estrategia regular de **backups**.

También si estás realizando mantenimiento en la base de datos y necesitas una copia de seguridad adicional como medida de seguridad, el **copy-only backup** te permite hacerlo sin interferir con los otros **backups**.

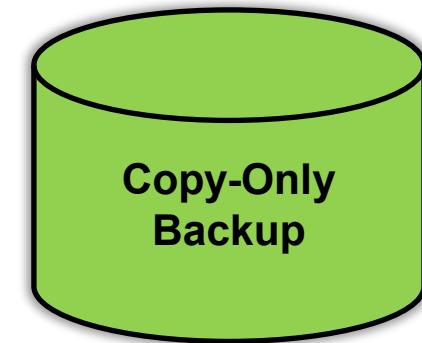


# ¿Qué es un Backup?

**Copy-Only Backup**  
(Copia de seguridad de solo copia)

## Sintaxis:

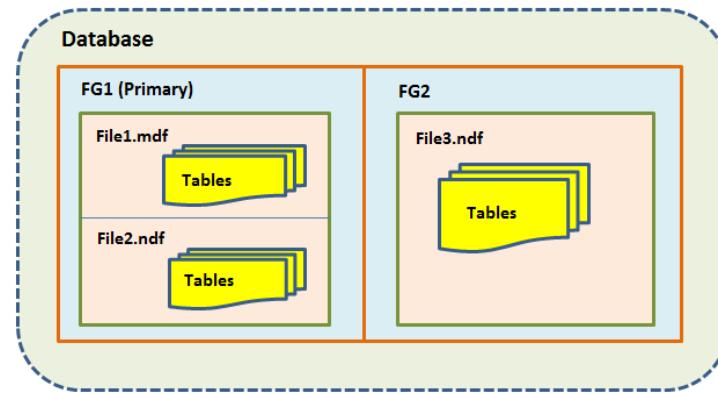
```
BACKUP DATABASE [database_name]  
TO DISK = 'path'  
WITH COPY_ONLY,  
[ options ];
```



# ¿Qué es un Backup?

## Filegroup Backup (Copia de seguridad de Grupo de Archivo)

Un **Filegroup Backup** en SQL Server es un tipo de respaldo que permite hacer copias de seguridad de un conjunto específico de archivos o filegroups en lugar de realizar un respaldo completo de la base de datos. Los **filegroups** son **contenedores lógicos** que **agrupan** archivos dentro de una base de datos y **facilitan** la gestión de grandes bases de datos distribuyendo los datos en múltiples archivos. Con un filegroup backup, puedes respaldar solo los datos que se encuentran en un filegroup particular.



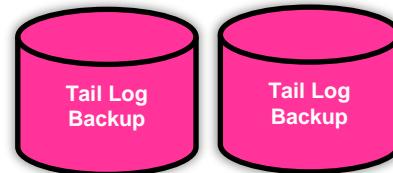
# ¿Qué es un Backup?

## Tail Log Backup

(Copia de seguridad de la Cola de Registro)

El **Tail-Log Backup** en SQL Server es un tipo especial de copia de seguridad que captura las últimas **transacciones que aún no se han respaldado en el log** de transacciones antes de que se restaure la base de datos o cuando la base de datos ha fallado. Este tipo de respaldo garantiza que no se pierdan datos al final de la cadena de transacciones antes de una restauración o antes de que una base de datos se vuelva inaccesible.

El **Tail-Log Backup** es una parte crucial de la estrategia de recuperación ante desastres, ya que permite **capturar los últimos cambios** en la base de datos para asegurar que toda la actividad reciente esté protegida antes de iniciar una restauración.



# Estrategias de Backups



# Estrategias de Backups

Las estrategias de backup en SQL Server son enfoques diseñados para garantizar la seguridad, integridad y disponibilidad de los datos almacenados en las bases de datos. La elección de la estrategia de backup adecuada depende de varios factores, como el tamaño de la base de datos, la frecuencia de los cambios en los datos, los requisitos de recuperación ante desastres y los recursos disponibles (como almacenamiento y tiempo de inactividad permitido).

# Estrategias de Backups

Domingo



Lunes



Martes



Miércoles



Jueves



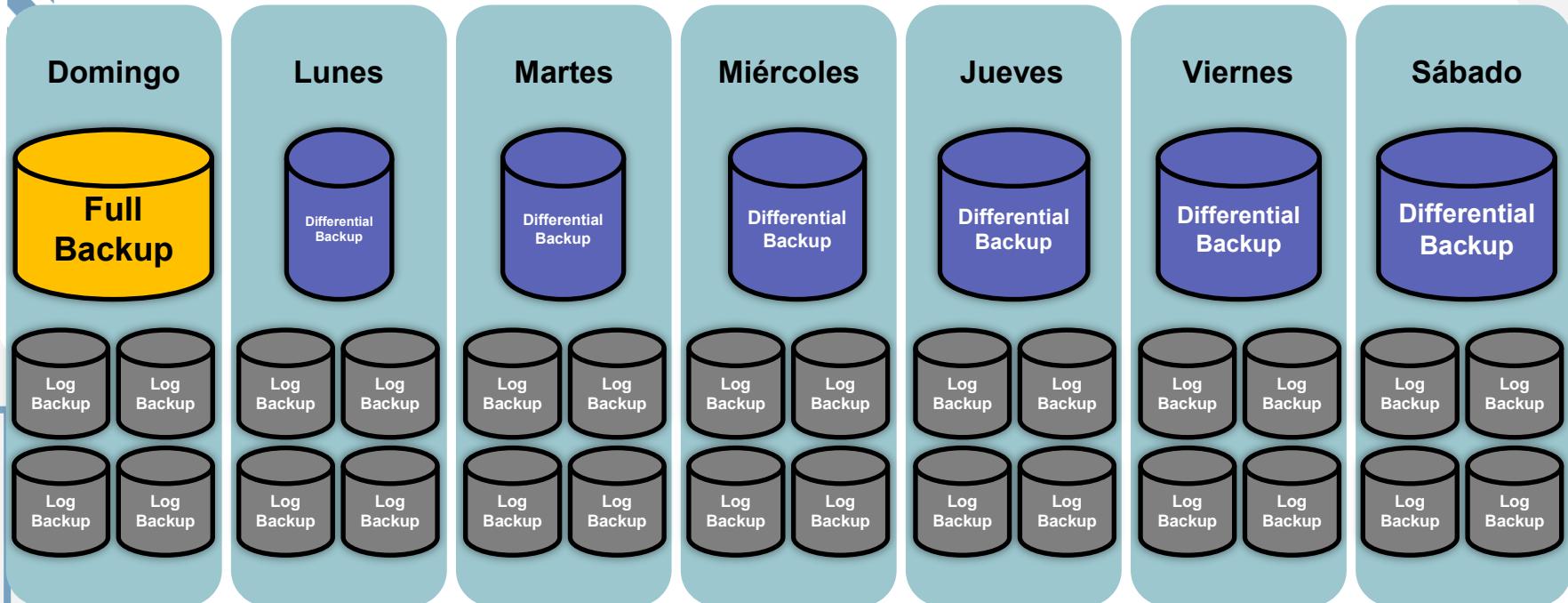
Viernes



Sábado



# Estrategias de Backups



## Descripción de la Sección

¿Qué es un Scheduled JOB?

Iniciar SQL Server Agent

Crear JOB

Crear JOB Múltiple



# ¿Qué es un Scheduled JOB?



# ¿Qué es un Scheduled JOB?

Un Scheduled Job en SQL Server es una tarea automatizada que se programa para ejecutarse en un momento específico o a intervalos regulares. Estos trabajos son manejados por el SQL Server Agent, que es el servicio responsable de ejecutar tareas automatizadas en segundo plano. Los jobs (trabajos) pueden incluir una variedad de tareas, como ejecutar consultas SQL, respaldar bases de datos, restaurar datos, realizar tareas de mantenimiento, etc.

# ¿Qué es un Scheduled JOB?

Ejemplo de Tareas que se Pueden Automatizar con un Job:

Backups automáticos

Tareas de mantenimiento

Tareas ETL (Extract, Transform, Load)

Ejecución de reportes periódicos

# ¿Qué es un Scheduled JOB?

Ventajas:

Automatización

Minimiza errores humanos

Flexibilidad

Monitoreo

# ¿Qué es un Scheduled JOB?

Un Scheduled Job en SQL Server es una forma poderosa de automatizar tareas críticas, como el respaldo de bases de datos, mantenimiento y reportes, permitiendo a los administradores de bases de datos asegurarse de que estas tareas se realicen consistentemente y sin intervención manual. El SQL Server Agent es el componente clave que gestiona estos jobs, y la configuración de estos trabajos puede adaptarse a las necesidades particulares del entorno o de la empresa.



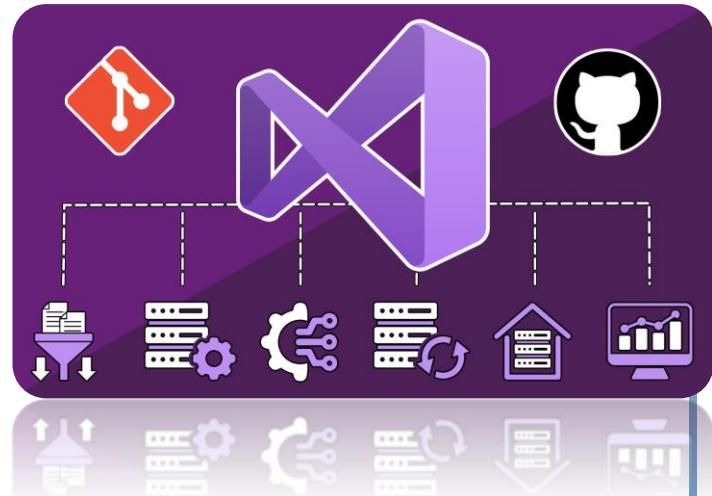
felicitaciones!

# ¡CUAL ES EL SIGUIENTE PASO!

## DISEÑO DE BASES DE DATOS RELACIONALES



## SQL SERVER INTEGRATION SERVICES DESDE CERO



# ¡CUAL ES EL SIGUIENTE PASO!

## AZURE DATA FACTORY (ADF)



## APRENDE TABLEAU DESDE CERO HASTA EXPERTO





felicitaciones!