**UNIVERSIDAD NACIONAL AUTÓNOMA DE HONDURAS**

**FACULTAD DE INGENIERÍA**

**DEPTO. DE INGENIERÍA EN SISTEMAS**

**GUIA DE APRENDISAJE SQL**

**PARA SQL SERVER**

**PRESENTADO POR:**

**RICHARD ANDERZON GARCIA RIVERA**

**17 DE AGOSTO DEÑ 2025**

Tabla de contenido

**[INDICE DE COMANDOS DDL 4](#_Toc4372)**

[CREATE 4](#_Toc30898)

[ALTER 4](#_Toc72)

[DROP 4](#_Toc23525)

[OPERADORES 4](#_Toc10460)

[TIPOS DE CONSTRAINTS (RESTRICCIONES) 4](#_Toc15807)

[RESTRICCIONES PARA FOREIGN KEY () 4](#_Toc31276)

[OTROS 5](#_Toc18519)

[TIPOS DE DATOS EN SQL SERVER 5](#_Toc13687)

**[INDICE DE COMANDOS DML 6](#_Toc357)**

[COMANDOS 6](#_Toc21794)

[OPERADORES DML 6](#_Toc7941)

[JOINS 6](#_Toc28412)

[OPERADORES SQL 6](#_Toc16228)

**[FUNCIONES 7](#_Toc29106)**

[FUNCIONES AGREGADAS 7](#_Toc743)

[FUNCIONES DE CADENA 7](#_Toc24277)

[FUNCIONES DE FECHA Y HORA 7](#_Toc17560)

**[INDICE DE COMANDOS DCL 8](#_Toc28189)**

[COMANDOS BASICOS 8](#_Toc19154)

[COMANDOS TCL 8](#_Toc11224)

[CONTROL DE SESIÓN Y CONEXIÓN 8](#_Toc23176)

[DEFINICIÓN AVANZADA DE OBJETOS 8](#_Toc29596)

[GESTIÓN DE USUARIOS Y ROLES 8](#_Toc10660)

[CONTROL AVANZADO DE OBJETOS 8](#_Toc29108)

[MANTENIMIENTO Y ADMINISTRACIÓN 8](#_Toc20797)

**[PERMISOS Y ROLES CON GRANT 9](#_Toc26857)**

[PERMISOS SOBRE TABLAS 9](#_Toc9922)

[PERMISOS SOBRE VISTAS 9](#_Toc10127)

[PERMISOS SOBRE BD 9](#_Toc23376)

[PERMISOS SOBRE PROCEDIMIENTOS 9](#_Toc11776)

[PERMISOS DE ADMINISTRACIÓN 9](#_Toc31428)

**[COMADOS CURSORES 10](#_Toc23021)**

**[SECUENCIAS 11](#_Toc28743)**

**[EJEMPLOS DEL USO DE DDL 12](#_Toc17703)**

[CREATE 12](#_Toc9326)

[DROP 14](#_Toc15316)

**[EJEMPLOS DEL USO DE DML 17](#_Toc10530)**

[SELECT 17](#_Toc3372)

[ORDER BY 18](#_Toc18914)

**[SUBCONSULTAS 19](#_Toc21294)**

**[EJEMPLOS DEL USO DE TCL 20](#_Toc19248)**

[COMIT AND ROLLBACK 20](#_Toc28910)

**[GESTIÓN DE USUARIOS 21](#_Toc14869)**

**[ENCRIPTACIÓN DE DATOS 22](#_Toc17518)**

[HAHS 22](#_Toc29991)

**[CREACION DE ESQUEMAS 23](#_Toc12318)**

**[CURSORES 24](#_Toc8832)**

[SINTAXIS: 24](#_Toc374)

[EXEC 24](#_Toc8820)

**[BACKUPS 25](#_Toc9543)**

[FULL BACKUP (Respaldo completo) 25](#_Toc19736)

[DIFFERENTIAL BACKUP (Respaldo diferencial) 25](#_Toc24504)

[TRANSACTION LOG BACKUP (Respaldo del log de transacciones) 26](#_Toc30928)

[COPY-ONLY BACKUP 26](#_Toc11532)

[FILE / FILEGROUP BACKUP 26](#_Toc31359)

**[SECUENCIAS 27](#_Toc22940)**

**[EJERCICIOS 28](#_Toc5140)**

**[AUDITORIA DE BDs 29](#_Toc4786)**

**[TEMAS POR CUBRIR 32](#_Toc11388)**

**[PROYECTOS POR CUBRIR 33](#_Toc32546)**

**[Python como aliado en tu camino (explicado en cada punto) 34](#_Toc17486)**

**[Lo que estamos haciendo hoy 35](#_Toc9694)**

[Módulo 1 – Introducción a SQL Server Agent 35](#_Toc29865)

[Módulo 2 – Crear tu primer Job 35](#_Toc22985)

[Módulo 3 – Jobs con múltiples Steps 35](#_Toc29204)

[Módulo 4 – Monitoreo y Alertas 36](#_Toc23967)

[Módulo 5 – Buenas Prácticas 36](#_Toc7019)

[Módulo 6 – Proyecto Final 36](#_Toc18351)

**[REQUISITOS TRABAJOS 37](#_Toc7448)**

# INDICE DE COMANDOS DDL

## **CREATE**

* CREATE DATABASE → crea una nueva base de datos.
* CREATE TABLE → crea una nueva tabla.
* CREATE VIEW → crea una vista.
* CREATE INDEX → crea un índice para optimizar búsquedas.
* CREATE SCHEMA → crea un esquema para organizar objetos.
* CREATE TRIGGER → crea un disparador asociado a eventos (INSERT, UPDATE, DELETE)

## ALTER

* ALTER DATABASE → cambia configuraciones de la base de datos
* ALTER TABLE → agrega, modifica o elimina columnas, constraints, claves foráneas, etc.
* ALTER INDEX → modifica propiedades de un índice (habilitar/deshabilitar)
* ALTER SCHEMA → Modificar o mover objetos entre esquemas

## DROP

* DROP DATABASE → elimina una base de datos completa
* DROP TABLE → elimina una tabla (estructura y datos)
* DROP VIEW → elimina una vista
* DROP INDEX → elimina un índice.
* DROP SCHEMA → elimina un esquema

## OPERADORES

* PRIMARY KEY → Define una clave primaria en una tabla.
* FOREIGN KEY → Define una clave foránea para relacionar tablas.
* UNIQUE → Garantiza valores únicos en una columna o conjunto de columnas.
* NOT NULL → Evita que una columna acepte valores nulos.
* CHECK → Define restricciones de validación en columnas. Se puede usar IN
* DEFAULT → Define un valor por defecto para una columna.
* REFERENCES → Especifica la tabla y columna referenciada para claves foráneas.
* IDENTITY(n,n) → autoincrementar los valores de n en n
* ADD → usada con alter para agregar caracteristicas
* REFERENCES → para referencias otra tabla cunado usamos el FOREING KEY

## **TIPOS DE CONSTRAINTS (RESTRICCIONES)**

* PRIMARY KEY → Clave primaria
* FOREIGN KEY → Clave foránea
* UNIQUE → Valores únicos
* CHECK → Validación de datos
* DEFAULT → Valor por defecto
* NOT NULL → No nulos

## RESTRICCIONES PARA FOREIGN KEY ()

* 1. CASCADE Actualiza las claves foránea en los hijos cuando se actualiza la clave primaria del padre
* 2. SET NULL Establece NULL en la clave foránea de los hijos al actualizar la clave del padre
* 3. SET DEFAULT Establece el valor por defecto en la clave foránea de los hijos
* 4. NO ACTION Genera error si se intenta actualizar una clave primaria que está siendo referenciada
* 5. RESTRICT Similar a NO ACTION pero con verificación inmediata
* ON DELETE CASCADE : Elimina todas los resgistros relacionados
* UN UPDATE CASCADE : actualiza los registros relacionados

## OTROS

* **TRUNCATE** → Vaciar todos los registros de una tabla sin eliminar su estructura
* **COMMENT** → Agregar comentarios descriptivos a objetos
* **RENAME** → Renombrar objetos de la base de datos ( tablas, columnas)
* USE name\_bd →sleleccionar la BD con la que vamos a trabajar

## TIPOS DE DATOS EN SQL SERVER

**Numéricos**

* INT: Enteros (±2.147 mil millones)
* BIGINT: Enteros grandes (±9.2 trillones)
* SMALLINT: Enteros pequeños (±32,767)
* TINYINT: Enteros 0-255
* DECIMAL(p,s): Números exactos (precisión, escala)
* FLOAT: Números aproximados
* REAL: Números flotantes simples
* MONEY: Datos monetarios

**Texto**

* VARCHAR(n): Texto variable (1-8000 chars)
* CHAR(n): Texto fijo (1-8000 chars)
* TEXT: Texto largo (hasta 2GB)
* NVARCHAR(n): Texto Unicode variable

**Fecha/Hora**

* DATE: Fecha (año, mes, día)
* TIME: Hora (hora, minuto, segundo)
* DATETIME: Fecha y hora (1753-9999)
* DATETIME2 (n): Fecha y hora extendida deciamales con n presicion
* SMALLDATETIME: Fecha y hora compacta
* SYSUTCDATETIME(): Devuelve la fecha y hora actual del servidor en horario UTC

**Binarios**

* BINARY: Datos binarios fijos
* VARBINARY: Datos binarios variables
* IMAGE: Datos binarios grandes
* VARBINARY(MAX) almacena (doc, pdf, imágenes, videos, audio) máx 2 GB por registro
* VARCHAR(MAX) → almacena texto variable hasta caracteres (2 GB).
* NVARCHAR(MAX) → almacena texto variable en Unicode hasta 2^30-1 caracteres ( 1 GB).
* FILESTREAM: Almacena en sistema de archivos, gestiona con SQL

**Otros**

* BIT: Booleanos (0/1)
* UNIQUEIDENTIFIER: GUID (128 bits)
* XML: Datos XML
* JSON: Datos JSON (SQL Server 2016+)

# INDICE DE COMANDOS DML

## COMANDOS

* **SELECT** → Recupera datos de una o varias tablas o vistas.
* **INSERT INTO** →Inserta nuevos registros en una tabla.
* **UPDATE** →Modifica datos existentes en una tabla.
* **DELETE**→ Elimina registros específicos de una tabla.
* **MERGE** → combina operaciones de INSERT, UPDATE y DELETE en una sola instrucción

## OPERADORES DML

* FROM → Indica la tabla o vista de donde se consultan los datos.
* ON → Condición de unión entre tablas.
* USING → Otra forma de indicar la columna de unión.
* GROUP BY → Agrupar filas por columnas para funciones agregadas.
* HAVING → Filtrar grupos (después de GROUP BY).
* ORDER BY → Ordenar resultados.
* LIMIT / OFFSET / FETCH → Limitar número de filas devueltas.

## JOINS

* INNER JOIN → Devuelve solo las filas que **tienen coincidencias en ambas tablas**.
* LEFT JOIN → Devuelve **las filas de la tabla izquierda** y las coincidencias de la tabla derecha
* RIGHT JOIN → Devuelve **las filas de la tabla derecha** y las coincidencias de la tabla izquierda.
* FULL JOIN → Devuelve **ambas tablas**, combinando coincidencias, llenado con NULL donde no haya
* CROSS JOIN → Devuelve el producto cartesiano de ambas tablas
* SELF JOIN → La tabla se une **a sí misma**.
* NATURAL JOIN → Une tablas usando columnas con el mismo nombre y tipo de dato
* JOIN … USING → Filas coincidentes usando columna específica

## OPERADORES SQL

* =, <>, !=, <, >, <=, >= → Comparación básica.
* AND, OR, NOT→ Lógica booleana.
* IN → Comparar con un conjunto de valores.
* BETWEEN→ Rango de valores.
* LIKE → Comparación con patrones
* IS NULL / IS NOT NULL→ Verificar valores nulos.
* EXISTS → Verifica existencia de filas en subconsulta.
* ANY / ALL → Comparaciones con subconsultas
* VALUES → crusiar para asignar los valores usando INSERT INTO

# FUNCIONES

## FUNCIONES AGREGADAS

* SUM () → Suma los valores de una columna numérica.
* AVG ( )→ Calcula el promedio de los valores de una columna.
* COUNT () → Cuenta el número de filas.
* MAX () → Obtiene el valor máximo de una columna.
* MIN () → Obtiene el valor mínimo de una columna.
* STDDEV ()/VARIANCE () → Desviación estándar o varianza (estadística).
* GROUP\_CONCAT () STRING\_AGG () → Concatenar valores de un grupo en una sola cadena

## FUNCIONES DE CADENA

* CONCAT() → Une dos o más cadenas.
* SUBSTRING()/SUBSTR() → Extrae una parte de una cadena.
* LENGTH()/LEN()→ Longitud de una cadena.
* UPPER()/LOWER()→ Convierte a mayúsculas o minúsculas.
* TRIM() / LTRIM / RTRIM → Elimina espacios en blanco de una cadena.
* REPLACE() → Reemplaza un valor dentro de la cadena.

## FUNCIONES DE FECHA Y HORA

* NOW()/CURRENT\_TIMESTAMP → Fecha y hora actual
* CURDATE()/CURRENT\_DATE → Solo la fecha actual
* DATEADD() / ADDDATE() → Sumar días, meses, años a una fecha
* DATEDIFF() → Diferencia entre dos fechas
* EXTRACT() → Extrae parte de una fecha (año, mes, día)
* TO\_DATE()/TO\_CHAR() → Convertir entre texto y fecha
* @@IDENTITY → devuelve el último IDENTITY generado en la sesión
* IDENT\_CURRENT('Tabla') → devuelve el último IDENTITY generado en una tabla específica,
* SCOPE\_IDENTITY() → el más seguro para obtener el último IDENTITY generado en tu misma sesión

# INDICE DE COMANDOS DCL

## COMANDOS BASICOS

* **GRANT**→ Otorga permisos o privilegios a usuarios o roles sobre objetos de la base de datos.
* **REVOKE**→ Quita permisos o privilegios previamente otorgados a usuarios o roles.

## COMANDOS TCL

* COMMIT → Confirma los cambios realizados en la transacción actual.
* ROLLBACK → Revierte los cambios realizados en la transacción actual.
* SAVEPOINT → Define un punto dentro de una transacción al que se puede volver con ROLLBACK.
* SET TRANSACTION → Configura propiedades de la transacción (aislamiento, modo, etc.)

## CONTROL DE SESIÓN Y CONEXIÓN

* SET → Configura variables de sesión, parámetros o opciones de la base de datos.
* USE → Selecciona la base de datos activa en la sesión.
* ALTER SESSION (Oracle) → Cambia parámetros de la sesión actual.

## DEFINICIÓN AVANZADA DE OBJETOS

* CREATE SEQUENCE → Crea una secuencia para generar valores únicos automáticamente.
* CREATE SYNONYM → Crea un alias para un objeto de la base de datos (Oracle).
* CREATE MATERIALIZED VIEW → Crea una vista materializada que almacena datos físicamente (Oracle, PostgreSQL).
* ALTER SEQUENCE → Modifica propiedades de una secuencia existente.
* DROP SEQUENCE / SYNONYM / MATERIALIZED VIEW → Elimina estos objetos.

## GESTIÓN DE USUARIOS Y ROLES

* CREATE USER / ALTER USER / DROP USER → Crear, modificar o eliminar usuarios.
* CREATE ROLE / ALTER ROLE / DROP ROLE → Gestionar roles de permisos.
* SET PASSWORD → Cambiar contraseña de un usuario.
* GRANT ROLE / REVOKE ROLE → Asignar o quitar roles a usuarios.

## CONTROL AVANZADO DE OBJETOS

* ANALYZE / DBCC → Comandos de diagnóstico o mantenimiento (DBCC en SQL Server).
* EXPLAIN / EXPLAIN PLAN → Muestra el plan de ejecución de una consulta (optimización).
* LOCK TABLE / SELECT … FOR UPDATE → Bloquea filas o tablas para control de concurrencia.

## MANTENIMIENTO Y ADMINISTRACIÓN

* BACKUP / RESTORE → Respaldo y recuperación de la base de datos.
* CHECKPOINT → Fuerza la escritura de datos pendientes en disco (SQL Server, PostgreSQL).
* SHUTDOWN / STARTUP → Encender o apagar instancias de la base de datos (Oracle).
* TRUNCATE / PURGE → Eliminación rápida de datos, liberando espacio.

# PERMISOS Y ROLES CON GRANT

## PERMISOS SOBRE TABLAS

* **SELECT** → **Permite leer datos de la tabla.**
* **INSERT**→ **Permite agregar nuevos registros a la tabla.**
* **UPDATE**→ **Permite modificar datos existentes en la tabla.**
* **DELETE**→ **Permite eliminar registros de la tabla.**
* **REFERENCES**→ **Permite crear claves foráneas que apunten a la tabla.**
* **TRIGGER**→ **Permite crear triggers sobre la tabla**

## PERMISOS SOBRE VISTAS

* SELECT→ Permite consultar datos de la vista.
* INSERT →Permite insertar datos a través de la vista (si es actualizable).
* UPDATE →Permite actualizar datos a través de la vista.
* DELETE →Permite eliminar datos a través de la vista.

## PERMISOS SOBRE BD

* CREATE→ Permite crear tablas, vistas u otros objetos dentro de la base de datos o esquema.
* ALTER → Permite modificar objetos existentes dentro de la base de datos o esquema.
* DROP→ Permite eliminar objetos dentro de la base de datos o esquema.
* BACKUP/RESTORE → Permite hacer respaldo o restaurar la base de datos (SQL Server)

## PERMISOS SOBRE PROCEDIMIENTOS

* EXECUTE→ Permite ejecutar procedimientos almacenados o funciones.
* ALTER→ Permite modificar la definición del objeto.
* DROP→ Permite eliminar el objeto.
* PROCEDURE → Crear procedimientos almacenados

## PERMISOS DE ADMINISTRACIÓN

* GRANT OPTION→ Permite al usuario otorgar a otros usuarios los permisos que él posee.
* ALL PRIVILEGESO→ torga todos los permisos posibles sobre el objeto (tabla, base de datos, etc.)
* CONTROL → para asiganar permisos de transacciones

# COMADOS CURSORES

## GENERAL

* DECLARE name\_c OPTION CURSOR FOR → Define un cursor con su consulta.
* OPEN CURSOR → Abre el cursor para lectura.
* FETCH → Recupera la siguiente fila del cursor.
* CLOSE CURSOR → Cierra el cursor.
* OPEN → Abre el cursor y ejecuta la consulta definida en DECLARE
* DECLARE @varible1 TIPO, @varible2 TIPO; → decalra variables
* SET @variable = “valor” → Asignar valores obtenidos de filas del cursor a variables.
* FETCH NEXT FROM name\_c INTO @varible1, @varible2;
* @@FETCH\_STATUS → indica si el FETCH tuvo éxito
* CLOSE → Cierra el cursor
* DEALLOCATE → Libera completamente el cursor de memoria

Fetch

* FETCH FIRST → Primera fila (alternativa).
* FETCH PRIOR → Fila anterior.
* FETCH LAST → Última fila.
* FETCH ABSOLUTE n → Fila n-ésima.
* FETCH NEXT → siguiente fila

Fetch status

* 0 → Se recuperó la fila correctamente.
* -1 → No hay más filas.
* -2 → Fila eliminada.

## CONTROL DE FLUJO ASOCIADO A CURSORES

* WHILE … END / LOOP → Permite iterar sobre las filas mientras se cumpla una condición.
* IF … ELSE → Permite tomar decisiones basadas en los valores de las filas.
* BREAK / LEAVE → Salir de un bucle o ciclo antes de que termine (según motor).
* CONTINUE → Saltar a la siguiente iteración del bucle.
* WHILE → Itera mientras haya filas.

OPCIONES DE CURSOR

* STATIC→ No refleja cambios posteriores
* DYNAMICC → refleja cambios en la tabla (inserciones, actualizaciones, eliminaciones)
* KEYSET → los cambios en los valores no claves se reflejan, pero filas nuevas no aparecen.
* FAST\_FORWARD → permite recorrido hacia adelante, no permite actualizar ni retroceder
* FORWARD\_ONLY → Similar a FAST\_FORWARD, solo permite avanzar fila por fila.
* READ\_ONLY → No permite actualizar los datos a través del cursor.
* SCROLL Permite moverse hacia adelante y atrás (por ejemplo FETCH PRIOR, FETCH ABSOLUTE)
* INSENSITIVE → Los cambios en la tabla no afectan el cursor una vez abierto.

OPCIONES ADICIONALES EN ORACLE Y POSTGRESQL

* FOR UPDATE → Bloquea las filas seleccionadas para que otros no las modifiquen mientras se recorre
* WITH HOLD (PostgreSQL) → Cursor permanece abierto después de un COMMIT.
* WITHOUT HOLD → Cursor se cierra automáticamente al hacer COMMIT.

TRANSACCIONES Y CURSORES

* COMMIT → Confirmar cambios realizados dentro de un cursor si se usan operaciones DML.
* ROLLBACK → Deshacer cambios dentro de un cursor en caso de error.

# SECUENCIAS

## GENERAL

* CREATE SEQUENCE → Crear una secuencia.
* ALTER SEQUENCE → Modificar parámetros de la secuencia
* DROP SEQUENCE → elimina una sequencia
* START WITH n → El primer número que generará la secuencia será “n”
* INCREMENT BY n → Cada vez que se use la secuencia, se incrementa en “n
* NOCACHE → No almacena números en memoria
* CACHE → Permite almacenar en memoria varios valores de la secuencia para optimizar el rendimiento.
* START WITH → Define el valor inicial de la secuencia.
* MINVALUE → Establece el valor mínimo que puede generar la secuencia.
* MAXVALUE → Establece el valor máximo que puede generar la secuencia.
* CYCLE → Permite que la secuencia vuelva al valor inicial cuando alcanza el máximo (si aplica).
* NOCYCLE → Evita que la secuencia se reinicie al alcanzar el máximo.

COMANDOS PARA OBTENER VALORES

* NEXTVAL → Devuelve el siguiente valor de la secuencia y lo incrementa según el valor definido.
* CURRVAL → Devuelve el último valor generado de la secuencia en la sesión actual
* SETVAL → Permite establecer manualmente un valor actual de la secuencia (reiniciarla o ajustarla).

OPERADORES Y RESTRICCIONES ASOCIADOS A SECUENCIAS

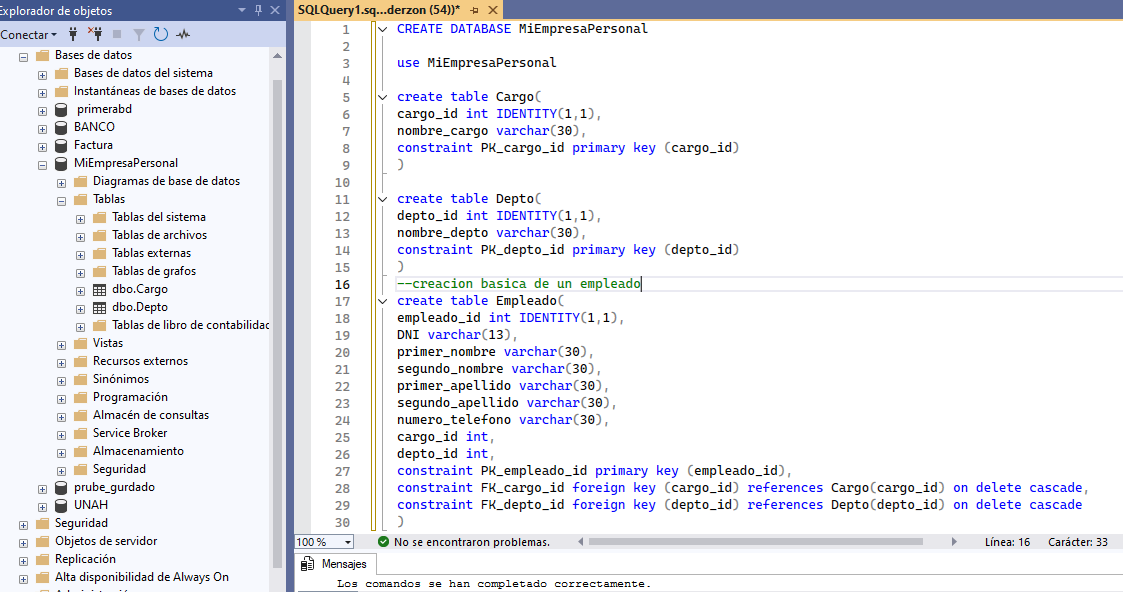
* PRIMARY KEY → Usar la secuencia para generar valores únicos para una clave primaria.
* DEFAULT → Asignar automáticamente el siguiente valor de la secuencia a una columna REFERENCES → se puede usar en combinación con claves foráneas si la secuencia genera identificadores que se relacionan

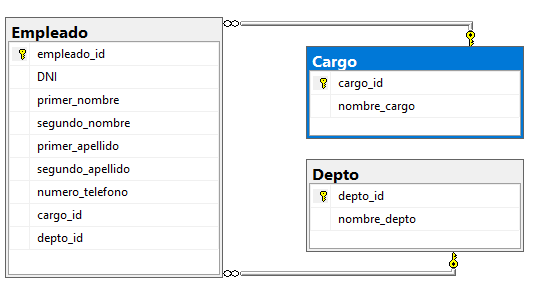
**MODULO 2**

# EJEMPLOS DEL USO DE DDL

## CREATE

* CREATE TABLE nombre\_tabla --sintaxis
* CREATE DATABASES nombre\_BD --sintaxis

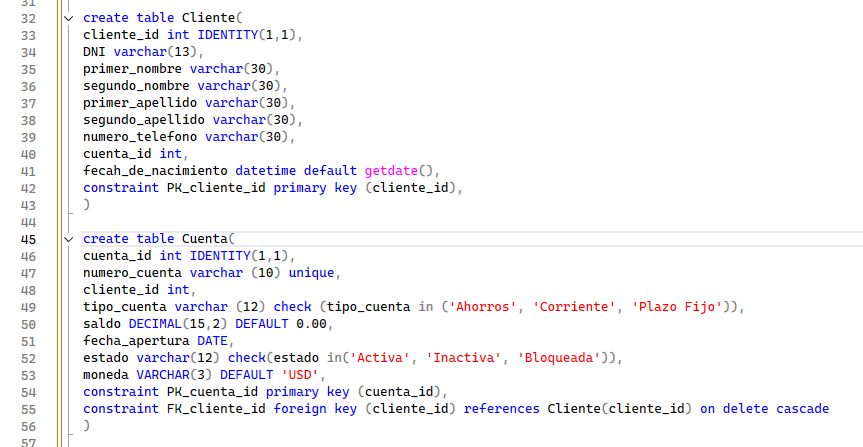




**Descripción:**

Creamos una BD (MiEmpresaPersonal) básica con 3 tablas (Cargo, Depto, Empleado) agregamos restricciones para sus llaves PK y FK para crear la relación entre las tablas como se ve en el diagrama, usamos IDENTITY (1,1) para que la PK se autoincremente en cada registro nuevo desde 1 y va de 1en 1 y agregamos un ON DELETE CASCADE por si se elimina un registro que támbien lo hagan sus referencias

Comandos usados (CREATE, USE, IDENTITY,CONSTRAINT, REFERENCES, PK,FK, ON DELETE CASCADE)



**Descripción:**

Usamos comando como **UNIQUE** esta es una restriccion que no perimite que el valor creado se repita en otro registro, Tambien usamos **GETDATE()** para obtener la fecha del registro, en la tabla “Cuenta” usamos un **CHECK** que verifica que el campo se llene con un valor ya definido y agregamos **IN** para decir que puede ser cualqueria de los valores que le siguen **()**, usamos **DEFALUL** para asignar un valor automatico ya predefindo

## DROP

### BORRAS TABLAS

* DROP TABLE nombre\_tabla --sintaxis
* **Caso**: Tenemos 2 tablas (Cargos y Empleados), la tabla Empleados depende de una FK que proviene de la tabla Cargo entonces, borrar la tabla Cargos

drop table Cargos -- esto no eliminara la tabal!! ya que esa tabla tiene una PK de la cual depende la tabla empleados entonces habria que borrar Empleados primero para luego borrar Cargos)

--Si no quieres borrar toda la tabla dependiente, elimina solo la restricción FK de la tabla dependiete

ALTER TABLE Empleados

drop constraint Nombre\_FK;

--Después ya puedes borrar la tabla Cargo ya que no esta la restriccion de FK lo que le impedia que se borrase, la tabla Empleados conserva los datos

drop table Cargo;

--Definir la FK con ON DELETE CASCADE --borra datos de forma automática al eliminar registros de la tabla principal (al crear la tabla o modificando el CONSTRAINT con AlTER)

Y lo mismo para objetos que estan relacionados a la tabla

### BORRAR OBJETOS

* DROP Tipo\_objeto NombreOjeto --sintaxis

Tipos\_objetos:

* Vista

DROP VIEW NombreVista;

--Si la Vw depende o es dependecia de otra Vw Quitar dependencias

* Procedimiento almacenado

DROP PROCEDURE NombreSP;

--Si tiene o es dependecia de otra Vw Quitar dependencias

* Función

DROP FUNCTION NombreFuncion;

--Si la f depende o es dependecia de otra f Quitar dependencias

* Trigger

DROP TRIGGER NombreTrigger;

-- Si un trigger está activando otro trigger Deshabilitarlo primero:

DISABLE TRIGGER MiTrigger ON MiTabla;

DROP TRIGGER MiTrigger;

* Índices

DROP INDEX NombreIndice ON MiTabla;

--Si es PK o Unique, Constraint, quitar la restricción.

ALTER TABLE MiTabla DROP CONSTRAINT PK\_MiTabla;

-- o para índices simples

DROP INDEX MiIndice ON MiTabla;

* Roles y usuarios

DROP ROLE MiRol;

--No se puede borrar si tiene miembros asignados, solucion: sacarlos

DROP USER MiUsuario;

--No se puede borrar un usuario si tiene objetos propios en la BD

* Synonyms

DROP SYNONYM MiSinonimo

* Schemas

DROP SCHEMA MiEsquema

--No se puede borrar si contiene obj dentro, mover o borrarlos todos

--Foreign Key / Primary Key / Unique / Check / Default

ALTER TABLE MiTabla DROP CONSTRAINT NombreConstraint

### BORRAR BDs

* DROP DATABASE nombre\_BD --sintaxis
* **Caso**: Tenemos una BD cual tiene usuarios conectados u objetos en uso

DROP DATABASE Mi\_BD;

ALTER DATABASE Mi\_BD SET SINGLE\_USER WITH ROLLBACK IMMEDIATE;

DROP DATABASE Mi\_BD; -- si usamos DROP con usuarios conectados dara error usamos este comando para evitar eso

ALTER DATABASE Mi\_BD

--cambiamos la configuración de la BD

SET SINGLE\_USER (＃)

-- Cierra el acceso a todos los demás usuarios, te reserva la BD para ti solo

WITH ROLLBACK IMMEDIATE (＃)

-- Desconecta a todos los usuarios de inmediato, cancelando sus transacciones abiertas, (si no pones esto, te obliga a esperar a que cierren sesión).

DROP DATABASE Mi\_BD;-- Ahora si puedes eliminar la BD sin error

* **Caso**: Jobs de SQL Agent, Tienes que editar o eliminar el Job en SQL Server Agent antes de hacer DROP DATABASE.

EXEC msdb.dbo.sp\_delete\_job @job\_name = 'NombreDelJob';

* **Caso**: Linked Servers, Si la BD está referenciada en un Linked Server, debes eliminar o modificar el Linked Server.

EXEC sp\_dropserver 'NombreLinkedServer', 'droplogins';

* **Caso**: Replication Si la BD es Publisher, Distributor o Subscriber, primero debes deshabilitar la replicación

EXEC sp\_removedbreplication @dbname = 'Mi\_BD';

* **Caso**: Mirroring Si está en Database Mirroring, debes quitar el mirroring:

ALTER DATABASE MiBD SET PARTNER OFF;

* **Caso** :Availability Groups (AlwaysOn) Si la BD está en un AG, primero quítala del grupo:

ALTER AVAILABILITY GROUP MiAG REMOVE DATABASE MiBD;

## ALTER

ALTER TABLE name\_table --modifica una tabla

ALTER COLUMN name\_column -- modifica una columna

ADD constraint -- modifica un constraint

ADD campo tipo y condiciones -- agrega un campo y su tipo

### EJEMPLOS

Ejemplo\_1:

Se creo la tabla empleados se desea agregar un constrint a la pimary key:

create table Empleado (

empleado\_id nvarchar (5),

nombre\_empleado nvarchar (100),

sueldo\_empleado money

)

-- el constraint no acepta campos donde pueda caer un null

ALTER TABLE Empleado --le agregamos no null con alter colum

ALTER COLUMN empleado\_id NVARCHAR(5) NOT NULL ;

-- agregamos el constraint con ADD

alter table Empleado add constraint empleadoPK primary Key (empleado\_id)

-- alter table edita la tabla

ALTER TABLE empleado

ADD empleado\_id INT IDENTITY(1,1) UNIQUE;

# EJEMPLOS DEL USO DE DML

## SELECT

sintaxis básica de una consulta select:

select campos from name\_table

go

-- go = final de un lote de consulta

-- from = tabla de procedencia

Ejemplo:

select \* from Factura

go

-- \* = todo

sintaxis de un select mostrando campos seleccionados:

select name\_campo1,name\_campo2,name\_campon\_n from name\_table k

go

-- k es una alias para la tabla, puede ser cualquier otra letra o palbra

-- es como un objeto y se accede a ella con un punto ‘.’

Ejemplo:

cualquiera de las 2 formas dara el mismo resultado

-- de forma directa

select FacturaId, total\_pagar from Factura f

go

-- usando el alias f.

select f.FacturaId, f.total\_pagar from Factura f

go

## ORDER BY



“OFFSET 5 ROWS FETCH NEXT 10 ROWS ONLY” no mostrara los registros apartir de la fila 6 y nos traera los siguientes 10

# SUBCONSULTAS

### Sintaxis

-- una sub conculta es una consulta dentro de otra consulta

-- agg 'as total' para asignar un alias al campo creado por la subconsulta

select f.FacturaId, f.fecha\_pago,

(select AVG(f.total\_MO) from factura f) as total -- esta es una subsconsulta

from Factura f

go

**Subconsulta** + where

select \* from Cliente c

where c.clienteID in (select c.clienteID from cliente c where c.P\_nombre like '%a%')

-- esta consulta me busca todo de los clientes cuya ID son de aquello usuarios

-- que tienen una A en su nombre

### EXISTS

* se usa para verificar si una **subconsulta** devuelve al menos una fila.
* Si la subconsulta devuelve **una o más filas**, EXISTS devuelve **TRUE**.
* Si la subconsulta no devuelve nada, EXISTS devuelve **FALSE**
* **Ejemplo**

-- mostrar los clientes que han hecho almenos una orden

SELECT cl.P\_nombre , clienteID, 1 as n\_compras

-- imprimir un campo con unos por cada resgistro

FROM cliente cl

WHERE EXISTS (SELECT 1 FROM dbo.ORDEN\_SERVICIO fa WHERE fa.clienteID = Cl.ClienteID);-- exists se asegura de que existe al menos una fila relacionada

-- mostrar los clientes que no han hecho ni una orden

SELECT cl.P\_nombre , clienteID

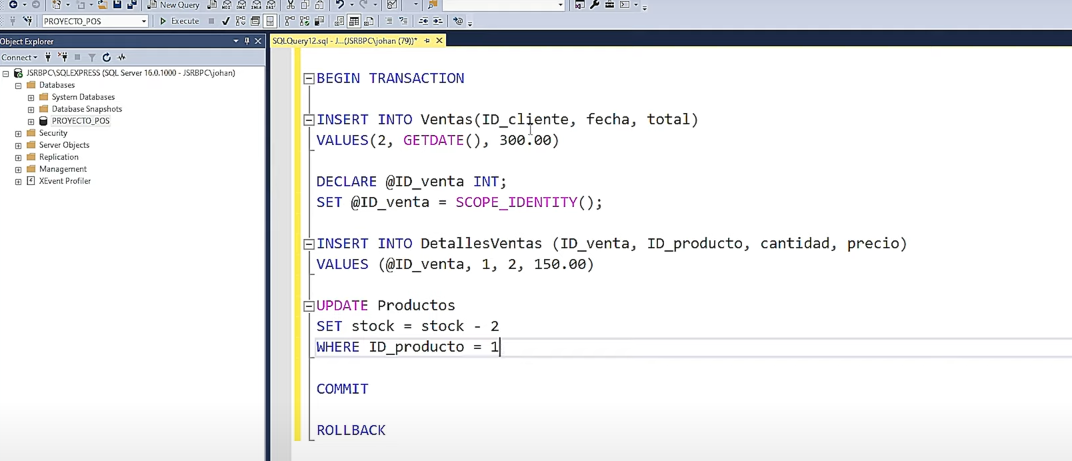
FROM cliente cl

WHERE not EXISTS (SELECT 1 FROM dbo.ORDEN\_SERVICIO os where cl.clienteID = os.clienteID );

Go -- not exists se asegura de que no exista una fila relacionada

# EJEMPLOS DEL USO DE TCL

## COMIT AND ROLLBACK



Sintaxis de una transsacción, en este caso hacemo uns transaccion donde modficiamos tablas asi como agregamos nuevos registros, primero agregamos un registro a la tabla ventas, decalaramos una variable a la cual le vamos a asignar el ultimo el ultimo valor insertado a una identidad usando **SCOPE\_IDENTUTY(),** ingresmaos los valores a la tabla DetallesVenta y con opdate modificamos la tabla producton restando lo que se vendio e la tabla DetallesVentas

finalmente ejecutamos **COMIT** si queremos guardar la transacion o **ROLLBACK** si queremos desh

# GESTIÓN DE USUARIOS

## CREAR USUARIOS

-- Crear un login en la BD, Un mismo login puede tener usuarios distintos en varias bases de datos del servidor (EL LOGIN SE CREA A NIVEL SERVIDOR)

create login name\_login with password = 'password'

-- Crear un usuario en la BD

create user name\_usuario for login name\_login

-- Asignar permisos al rol a una BD

GRANT permisos ON database::name\_BD TO name\_user

-- Asignar permisos al rol a una tabla a un usuario

GRANT permisos ON esquema.name\_table TO name\_user

-- Asignar permisos todo un esquema a un usuario

grant permisos on schema::name\_esquema to name\_user

-- Quitar permiso a un usuario

REVOKE SELECT, UPDATE ON dbo.Empleados FROM usuario\_reportes;

-- Dar control total del esquema a un usuario

GRANT CONTROL ON SCHEMA::nombre\_esquema TO usuario\_reportes;

## CREAR ROLES

-- Crear un rol

CREATE ROLE name\_role

-- Asignar permisos al rol a una BD

GRANT SELECT ON database::name\_BD TO name\_role

-- Asignar permisos al rol a una tabla a un usuario

GRANT SELECT ON esquema.name\_table TO name\_role

-- Asignar permisos todo un esquema a un role

grant delete on schema::name\_esquema to name\_role

-- Agregar un usuario al rol ALTER ROLE name\_role ADD MEMBER name\_user

--sacar a un usuario del rol

ALTER ROLE name\_role DROP MEMBER name\_user;

-- Quitar permiso a un rol

REVOKE INSERT ON dbo.AuditoriaCambiosSalario FROM name\_role;

-- Dar control total del esquema al rol

GRANT CONTROL ON SCHEMA::nombre\_esquema TO name\_role;

## AUDITORIA A ROLES Y USUARIOS

-- ver tablas de usuarios

select \* from sys.sysusers

-- Ver roles existentes

SELECT \* FROM sys.database\_principals WHERE type = 'R';

-- Ver permisos de un rol específico

SELECT dp.permission\_name, dp.state\_desc, ob.name AS objeto

FROM sys.database\_permissions dp

JOIN sys.objects ob ON dp.major\_id = ob.object\_id

WHERE dp.grantee\_principal\_id = USER\_ID('rol\_gerencia');

-- Ver roles asignados a cada usuario

SELECT dp1.name AS Usuario, dp2.name AS Rol

FROM sys.database\_role\_members drm

JOIN sys.database\_principals dp1 ON drm.member\_principal\_id = dp1.principal\_id

JOIN sys.database\_principals dp2 ON drm.role\_principal\_id = dp2.principal\_id;

-- Ver permisos directos de cada usuario

SELECT dp.class\_desc, dp.permission\_name, dp.state\_desc, ob.name AS Objeto, usr.name AS Usuario

FROM sys.database\_permissions dp

JOIN sys.database\_principals usr ON dp.grantee\_principal\_id = usr.principal\_id

LEFT JOIN sys.objects ob ON dp.major\_id = ob.object\_id

ORDER BY usr.name;

## EJEMPLOS

Ejemplo1

Dar permisos de lectura sobre todas las tablas a un usuario

CREATE LOGIN usuario\_reportes WITH PASSWORD = 'Reportes2025!';

CREATE USER usuario\_reportes FOR LOGIN usuario\_reportes;

GO

-- Declaramos variable para el script dinámico

DECLARE @sql NVARCHAR(MAX) = '';

-- Generamos GRANT SELECT para todas las tablas en dbo

SELECT @sql = @sql + 'GRANT SELECT ON dbo.' + QUOTENAME(name) + ' TO usuario\_reportes;' + CHAR(13)

FROM sys.tables

WHERE schema\_id = SCHEMA\_ID('dbo');

-- Ejecutamos el script

EXEC sp\_executesql @sql;

(Resumido)

-- asigna el perimiso a toda la BD al usuario

GRANT SELECT ON database::FacturaCajero TO usuario\_reportes;

# ENCRIPTACIÓN DE DATOS

## HAHS

create table cuentas (

cuenta\_id int identity(1,1),

usuario varchar(20),

pass VARBINARY(64) NOT NULL,--el tipo de datos debe ser VARNINARY(n)

constraint PK\_cuenta\_id\_1 primary key (cuenta\_id)

)

-- Insertar usuario con contraseña hasheada

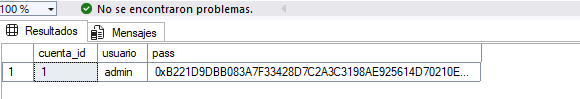
INSERT INTO cuentas(usuario, pass)

VALUES ('admin', HASHBYTES('SHA2\_256', 'hola'));

-- ver la tabla

select \* from cuentas

Esta es la sintaxis basica para sifrar datos en sql server con los **HASH,** se vera asi:



# CREACION DE ESQUEMAS

--crear esquema

CREATE SCHEMA name\_esquema;

-- Insertar en la tabla clientes del esquema ventas

INSERT INTO name\_equema.name\_table (campo1, campo2)

VALUES ('value 1', 'value 2')

-- ver tabla clientes del esquema ventas

SELECT c.Nombre, c.ClienteID FROM ventas.Clientes c

--Esto crea el esquema rrhh y asigna como propietario al usuario/rol usr\_Richard.

CREATE SCHEMA rrhh AUTHORIZATION usr\_Richard;

--mover tablas entre esquemas

ALTER SCHEMA esquema\_destino TRANSFER esquema\_origen.nombre\_tabla

--Eliminar un esquema

DROP SCHEMA rrhh;

--Permitir SELECT a todos los objetos del esquema ventas para el usuario

GRANT SELECT ON SCHEMA::ventas TO User1;

-- Permitir CRUD completo (DML)

GRANT SELECT, INSERT, UPDATE, DELETE ON SCHEMA::ventas TO User1;

-- Quitar permisos de inserción, actualización y eliminación

REVOKE INSERT, UPDATE, DELETE ON SCHEMA::ventas FROM User1;

-- Bloquear SELECT aunque el usuario tenga permisos en otra parte

DENY SELECT ON SCHEMA::ventas TO User1;

-- Quitar permiso de SELECT a usuario sobre una tala que pertenece a un esquema

REVOKE SELECT ON ventas.Clientes FROM User1;

--Renombrar un objeto dentro de un esquema

EXEC sp\_rename 'ventas.Clientes', 'Clientes\_Nuevo';

SELECT \* FROM sys.schemas; --ver tabla de esquemas existentes

--Usar sinónimos apuntando a objetos en otro esquema

CREATE SYNONYM ventas.OrdenesExternas FOR otroSchema.Ordenes;

## EJEMPLOS

Ejemplo 1:

--crear esquema

CREATE SCHEMA ventas;

--Crear tabla Clientes en el esquema ventas

CREATE TABLE ventas.Clientes (

ClienteID INT PRIMARY KEY IDENTITY(1,1),

Nombre NVARCHAR(100) NOT NULL,

Correo NVARCHAR(100) UNIQUE,

FechaRegistro DATE DEFAULT GETDATE()

)

--insertar datos a la tabla del esquema ventas

INSERT INTO ventas.Clientes (Nombre, Correo)

VALUES ('value 1', 'value 2')

Ejemplo 2:

Mover todas la tablas de un esquema a otro de forma dinaminca

-- Script dinámico para mover todas las tablas de dbo a ventas

DECLARE @sql NVARCHAR(MAX) = ''

SELECT @sql = @sql + 'ALTER SCHEMA ventas TRANSFER dbo.' + QUOTENAME(name) + ';' + CHAR(13)

FROM sys.tables

WHERE schema\_id = SCHEMA\_ID('dbo')-- Solo tablas del esquema dbo

-- Ejecutamos el script generado

EXEC sp\_executesql @sql

Ejemplo 3:

Mover ciertas tablas de un esquema a otro de forma dinaminca pero no todas (filtradas)

DECLARE @sql NVARCHAR(MAX) = ''

--Creamos una tabla temporal con los nombres de las tablas que queremos mover

DECLARE @TablasMover TABLE (NombreTabla NVARCHAR(128))

INSERT INTO @TablasMover (NombreTabla)

VALUES ('Tabla1'), ('Tabla3'), ('Tabla7');--tablas que quieras mover

-- Generamos el script dinámico

SELECT @sql = @sql + 'ALTER SCHEMA ventas TRANSFER dbo.' + QUOTENAME(NombreTabla) + ';' + CHAR(13)

FROM @TablasMover

-- Ejecutamos el script

EXEC sp\_executesql @s

# CURSORES

## SINTAXIS:

DECLARE @variable\_1 TIPO, @variable\_2 TIPO;

DECLARE name\_cursor CURSOR FOR

SELECT t.campo\_1, t.campo\_1

FROM name\_table t;

OPEN name\_cursor;

FETCH NEXT FROM name\_cursor INTO @variable\_1, @variable\_2;

WHILE @@FETCH\_STATUS = 0

BEGIN

Cuerpo de alguna función

END;

CLOSE name\_cursor;

DEALLOCATE name\_cursor;

EJEMPLO:

--Consultar

DECLARE @id INT, @nombre NVARCHAR(50);

DECLARE cursor\_clientes CURSOR FOR

SELECT c.clienteID, c.P\_nombre

FROM Cliente c;

OPEN cursor\_clientes;

FETCH NEXT FROM cursor\_clientes INTO @id, @nombre;

WHILE @@FETCH\_STATUS = 0

BEGIN

PRINT 'Cliente: ' + CAST(@id AS NVARCHAR) + ' - ' + @nombre;

FETCH NEXT FROM cursor\_clientes INTO @id, @nombre;

END;

CLOSE cursor\_clientes;

DEALLOCATE cursor\_clientes;

## EXEC

# BACKUPS

* Full Backup: Diariamente (noche)
* Differential Backup: Varias eces al día (cada 4-6 horas)
* Transaction Log Backup: Cada 15-30 minutos (en bancos puede ser cada 5 min)

## FULL BACKUP (Respaldo completo)

-- Copia toda la base de datos en un archivo .bak

BACKUP DATABASE MiBase

TO DISK = 'C:\Backups\MiBase\_Full.bak'

WITH FORMAT, INIT;

--WITH FORMAT → crea un nuevo medio de backup

--INIT → sobrescribe cualquier backup anterior en ese archivo

RESTAURARLA

RESTORE DATABASE MiBase

FROM DISK = 'C:\Backups\MiBase\_Full.bak'

WITH REPLACE;

--WITH REPLACE → sobrescribe la base existente si ya existe

## DIFFERENTIAL BACKUP (Respaldo diferencial)

BACKUP DATABASE MiBase

TO DISK = 'C:\Backups\MiBase\_Diff.bak'

WITH DIFFERENTIAL;

--Solo copia los cambios que están en el log de transacciones.

--Permite restaurar la base de datos hasta un punto específico en el tiempo.

--“diferential” → significa solo los cambios desde el último Full Backup

RESTAURARLA

--Se hace en dos pasos:

-- Primero restaurar Full Backup, pero no finalizamos

RESTORE DATABASE MiBase

FROM DISK = 'C:\Backups\MiBase\_Full.bak'

WITH NORECOVERY;

-- Luego restaurar Differential Backup

RESTORE DATABASE MiBase

FROM DISK = 'C:\Backups\MiBase\_Diff.bak'

WITH RECOVERY;

--NORECOVERY → mantiene la BD para poder aplicar más backups.

--RECOVERY → finaliza la restauración y deja la base lista para usar

## TRANSACTION LOG BACKUP (Respaldo del log de transacciones)

BACKUP LOG MiBase

TO DISK = 'C:\Backups\MiBase\_Log.trn';

--BACKUP LOG → indica que estamos haciendo backup del log de transacciones

--.trn → extensión común para backups de logs (puede ser .bak también)

RESTAURARLA

--Necesitas al menos un Full Backup antes, y opcionalmente un Differential.

-- Restaurar Full Backup (siempre)

RESTORE DATABASE MiBase

FROM DISK = 'C:\Backups\MiBase\_Full.bak'

WITH NORECOVERY; --aplicar mas backups

-- Restaurar Differential Backup (si se nesecita)

RESTORE DATABASE MiBase

FROM DISK = 'C:\Backups\MiBase\_Diff.bak'

WITH NORECOVERY; --aplicar mas backups

-- Restaurar Log Backup

RESTORE LOG MiBase

FROM DISK = 'C:\Backups\MiBase\_Log.trn'

WITH RECOVERY; --finalizar

## COPY-ONLY BACKUP

BACKUP LOG MiBase

TO DISK = 'C:\Backups\MiBase\_Log\_CopyOnly.trn'

WITH COPY\_ONLY;

--WITH COPY\_ONLY → indica que el backup es independiente de la secuencia

habitual.

RESTAURARLA

--se puede hacer como el full backop o igual para log backup

## FILE / FILEGROUP BACKUP

--Permite hacer backup solo de archivos específicos o grupos de archivos

BACKUP DATABASE MiBase

FILE = 'MiArchivo1' -- o FILEGROUP = 'Primario', 'DataGroup1'

TO DISK = 'C:\Backups\MiBase\_File.bak'

-- FILE = 'NombreArchivo' → indica qué archivo de la base de datos respaldar

-- FILEGROUP → indica los grupos de archivos a respal

RESTAURARLA

-- Restaurar solo un archivo específico -- Restaurar otro ach si es necesario

RESTORE DATABASE MiBase RESTORE DATABASE MiBase

FILE = 'MiArchivo1' FILE = 'MiArchivo2'

FROM DISK = 'C:\Backups\MiBase\_File.bak' FROM DISK ='C:\Backups\MiBd\_File2.ba'

WITH NORECOVERY; WITH RECOVERY

# SECUENCIAS

**SINTAXIS BÁSICA PARA CREAR UNA SECUENCIA**

CREATE SEQUENCE esquema.name\_secuencia

AS INT

START WITH 1 -- valor inicial

INCREMENT BY 1 -- incremento

MINVALUE 1 -- valor mínimo

MAXVALUE 1000 -- valor máximo

CYCLE; -- reinicia cuando llega al máximo

NEXT VALUE FOR NombreSecuencia --obtener el siguiente valor de la secuencia

-- tabla de sequencia

select \* from sys.sequences

## EJEMPLOS

-- creamos la sequencia

CREATE SEQUENCE SecuenciaFactura

AS INT

START WITH 1000

INCREMENT BY 1

go

--solo si no esta creada la tabla sobre la que vamos a itera

CREATE TABLE Facturas (

IdFactura INT PRIMARY KEY,

Cliente NVARCHAR(100),

Total DECIMAL(10,2)

)

Go

-- ingresar los valores de la secuencias en la primaryKey de una table

INSERT INTO Facturas (IdFactura, Cliente, Total)

VALUES (NEXT VALUE FOR SecuenciaFactura, 'Juan Pérez', 150.75),

(NEXT VALUE FOR SecuenciaFactura, 'Ana López', 299.99)

go

DIFERENCIA CON IDENTITY:

* IDENTITY está ligado a una tabla/columna.
* SEQUENCE es independiente y se puede usar en varias tablas o donde lo necesites

# TRIGGERS

## General

create trigger name\_trigger -- creaar un triguer

on table -- espesificamos sobre la tabla que se va a ejecutar

after update(delete, insert) -- tipo de triguer

as

begin

-- cuerpo de la accion

End

drop trigger name\_trigger -- eliminar un triguer

## EJEMPLOS

Ejemplos\_1:

Crea un trigger que se dispare después de una actualización en la tabla Agencia, y registre en AuditoriaCambiosSalario cualquier cambio en el campo Salario

create table registro ( --creamos la tabla donde iran los registros

registro\_id int identity primary key,

cliente\_id nvarchar(5),

usuario\_id nvarchar(5),

)

go

create trigger ActualizacionAgencia

on Agencia

after update

as

begin

declare @codigo\_cliente nvarchar(5);

-- mismo tipo de varianles que la tabla

select @codigo\_cliente = d.angencia\_id from deleted d;

-- asignamos un valor a la varibale

print 'Se actualizo tal agencia: ' + @codigo\_cliente;

-- mensage en consola

insert into registro(usuario\_id,cliente\_id)

-- ingresamos los valores a la tabla creada

values (@codigo\_cliente, user) -- por eso debe ser del mismo tipo XD

end

go

select \* from registro -- confirmamos los registros

go

update Agencia -- probar si funciona el triguer

set descripcion = 'adios'

where Agencia.angencia\_id = 110

go

# EJERCICIOS

**Con cursores**

Para la base de datos NORTHWND cree una nueva tabla, que permita puntuar a los customersID con más órdenes un TOP 5, la nueva tabla deberá tener los siguientes campos CustomerID, Nombre, QuantityOrders, para realizar la inserción en la nueva tabla, deberá utilizar un cursor para contabilizar e insertar en la tabla

use Northwind

go

create table TopCustomers( -- 1) creamos la tabla

CustomerID NVARCHAR(5),

Nombre varchar (100),

QuantityOrders int

)

DECLARE @CustomerID NVARCHAR(5), -- 2) declaramos las variables

@Nombre NVARCHAR(100),

@QuantityOrders INT;

DECLARE customer\_cursor CURSOR FOR -- 3) declaramos el cursor

select top(5) ord.CustomerID, cu.ContactName ,count (ord.OrderID) as n\_ordenes from Orders ord

inner join Customers cu on cu.CustomerID = ord.CustomerID

group by ord.CustomerID, cu.ContactName

ORDER BY COUNT(ord.OrderID) DESc;

-- 4) hacemos una especie de tabla temporal (no es, solo es un resultado)

OPEN customer\_cursor; -- 5) abrimos el cursor que declaramos en (3)

FETCH NEXT FROM customer\_cursor INTO @CustomerID, @Name, @QuantityOrders;

-- 6) interamos en el resultado (supuesta tabla) y con **INTO** asignamos los valores a las variables que creamos en (2)

WHILE @@FETCH\_STATUS = 0 -- 7) recoremos la supuesta tabla de (4)

BEGIN

INSERT INTO TopCustomers (CustomerID, Nombre, QuantityOrders)

VALUES (@CustomerID, @Name, @QuantityOrders);

-- 8) ingresamos los vaores de las variables a la tabla creada en (1)

FETCH NEXT FROM customer\_cursor INTO @CustomerID, @Name, @QuantityOrders;

-- 9) recorremos la siguinete fila del resultado (4)

END

CLOSE customer\_cursor; -- 10) cerramos el cursor que abrimos en (5)

DEALLOCATE customer\_cursor; -- 11) borramos de memoria el resultado (4)

# AUDITORIA DE BDs

--esto revisa los login que existena nivel servidor

SELECT \* FROM sys.server\_principals WHERE name = 'userDD1';

--Establece base de datos por defecto o modificar la base de datos a la que tiene acceso

-- asi solo tendra accesos a la BD que definamos

ALTER LOGIN userDD1 WITH DEFAULT\_DATABASE = Usuarios;

--usuarios creados a nivel de servidor

SELECT name, type\_desc, is\_disabled

FROM sys.server\_principals

WHERE type IN ('S', 'U'); -- S=SQL Login, U=Windows Login

SELECT \*

FROM sys.database\_principals

WHERE type IN ('S','U','G'); -- Usuarios SQL, Windows, Roles

-- Roles disponibles en la base actual

SELECT name, type\_desc

FROM sys.database\_principals

WHERE type = 'R';

-- Qué usuarios pertenecen a qué roles

SELECT r.name , m.name

FROM sys.database\_role\_members drm

JOIN sys.database\_principals r ON drm.role\_principal\_id = r.principal\_id

JOIN sys.database\_principals m ON drm.member\_principal\_id = m.principal\_id;

-- Permisos de usuarios sobre tablas

SELECT dp.name AS Usuario, obj.name AS Objeto, perm.permission\_name, perm.state\_desc

FROM sys.database\_permissions perm

JOIN sys.objects obj ON perm.major\_id = obj.object\_id

JOIN sys.database\_principals dp ON perm.grantee\_principal\_id = dp.principal\_id

ORDER BY Usuario, Objeto;

--Ver esquemas

SELECT name AS SchemaName, schema\_id, principal\_id

FROM sys.schemas;

-- Permisos a nivel de base de datos

SELECT dp.name, dp.type\_desc , p.permission\_name, p.state\_desc

FROM sys.database\_principals dp

LEFT JOIN sys.database\_permissions p ON p.grantee\_principal\_id = dp.principal\_id

WHERE dp.type IN ('S','U','G') ORDER BY dp.name;

-- Permisos a nivel de servidor

SELECT sp.name, sp.type\_desc, perm.permission\_name, perm.state\_desc

FROM sys.server\_principals sp

LEFT JOIN sys.server\_permissions perm ON perm.grantee\_principal\_id = sp.principal\_id;

-- Triggers a nivel de base de datos

SELECT name, parent\_class\_desc, create\_date, modify\_date, is\_disabled

FROM sys.triggers;

-- Triggers a nivel de servidor

SELECT name, parent\_class\_desc, create\_date, modify\_date, is\_disabled

FROM sys.server\_triggers;

-- Listado de bases de datos y sus configuraciones principales

SELECT name, database\_id, create\_date, compatibility\_level, collation\_name, state\_desc, recovery\_model\_desc

FROM sys.databases;

SELECT \*

FROM sys.databases;

-- Listar todas las tablas

SELECT t.name AS Tabla, s.name AS Esquema, t.create\_date

FROM sys.tables t

JOIN sys.schemas s ON t.schema\_id = s.schema\_id;

SELECT \*

FROM sys.tables

-- Columnas de cada tabla

SELECT t.name AS Tabla, c.name AS Columna, ty.name AS Tipo, c.max\_length, c.is\_nullable

FROM sys.columns c

JOIN sys.tables t ON c.object\_id = t.object\_id

JOIN sys.types ty ON c.user\_type\_id = ty.user\_type\_id

ORDER BY t.name;

-- Procedimientos almacenados

SELECT name, create\_date, modify\_date FROM sys.procedures;

-- Funciones

SELECT name, type\_desc, create\_date FROM sys.objects

WHERE type IN ('FN','TF','IF'); -- Escalares, tabla, inline

-- Vistas

SELECT name, create\_date FROM sys.views;

-- Cuentas de SQL deshabilitadas o bloqueadas

SELECT name, is\_disabled

FROM sys.sql\_logins

WHERE is\_disabled = 1;

-- Usuarios huérfanos (logins sin user en DB o viceversa)

EXEC sp\_change\_users\_login 'Report';

--Dependencias (qué llama a qué)

-- Qué objetos dependen de otros (ejemplo: SP que usan tablas)

SELECT referencing\_object\_name = o1.name,

referencing\_object\_type = o1.type\_desc,

referenced\_object\_name = o2.name,

referenced\_object\_type = o2.type\_desc

FROM sys.sql\_expression\_dependencies d

JOIN sys.objects o1 ON d.referencing\_id = o1.object\_id

JOIN sys.objects o2 ON d.referenced\_id = o2.object\_id;

--Últimos inicios de sesión fallidos (si tienes login auditing activado)Revisar en el Error Log:

EXEC xp\_readerrorlog 0, 1

EXEC xp\_readerrorlog 0, 1, '18456', NULL, NULL, NULL, 'DESC';

EXEC xp\_readerrorlog 0, 1, 'Login failed', NULL, NULL, NULL, 'DESC'

--0 = Archivo de error actual (0=actual, 1=anterior, etc.)

--1 = Tipo de log (1=SQL Server, 2=SQL Agent)

--'18456' = Texto a buscar

--NULL = Segundo texto a buscar (opcional)

--NULL, NULL = Fechas (opcional)

-- Intentos fallidos

EXEC xp\_readerrorlog 0, 1,NULL, NULL, NULL

-- Logins exitosos

EXEC xp\_readerrorlog 0, 1, Login\_succeeded, NULL, NULL, NULL;

-- Todos los eventos de autenticación

EXEC xp\_readerrorlog 0, 1, Login, NULL, NULL, NULL, 'DESC';

EXEC xp\_readerrorlog 0, 1, 'Login succeeded for user''', NULL, NULL, NULL, 'DESC';

-- Método más robusto

CREATE TABLE #Logins (

LogDate datetime,

ProcessInfo varchar(100),

Text varchar(4000)

);

INSERT INTO #Logins

EXEC xp\_readerrorlog 0, 1, NULL, NULL, NULL, NULL, 'DESC';

SELECT \* FROM #Logins

WHERE Text LIKE '%Login%succeeded%' OR Text LIKE '%Login%failed%'

ORDER BY LogDate DESC;

DROP TABLE #Logins;

-- ver los logins de los usuarios y a que hora

SELECT login\_name, host\_name, program\_name, login\_time

FROM sys.dm\_exec\_sessions

WHERE login\_name in ('UserDD1','Usr\_Richard'

## METADATOS

-- obtener metada de los objetos

SELECT

OBJECT\_ID('Agencia') AS ID\_Objeto,

OBJECT\_NAME(OBJECT\_ID('Agencia')) AS Nombre\_Objeto,

TYPE\_NAME(OBJECTPROPERTY(OBJECT\_ID('Agencia'), 'BaseType')) AS Tipo

# TEMAS POR CUBRIR

* **Backups y recuperación (Disaster Recovery)**
* Integración con Python, PowerShell o herramientas de monitoreo ( backups )
* Backups en la nube (Azure, AWS, etc.).
* Alta disponibilidad y replicación (AlwaysOn, Log Shipping, Mirroring, Clustering).
* Monitoreo de rendimiento (tuning de queries, índices, planes de ejecución).
* Automatización con SQL Server Agent (jobs, alertas, mantenimiento).
* Auditoría y seguridad avanzada.

**Bloque 2: Administración intermedia (lo que debes aprender ahora)**

* Backups diferenciales, completos, transaccionales.
* Restauración en diferentes escenarios (point-in-time recovery).
* SQL Server Agent: automatización de respaldos, limpiezas, envíos de correos.
* Índices: tipos de índices, cuándo usarlos, mantenimiento.
* Monitoreo de performance con DMVs (Dynamic Management Views)

**Bloque 3: Administración avanzada (para entornos empresariales)**

* Replicación, AlwaysOn, Log Shipping.
* Auditoría y cifrado de datos.
* Partitioning y compresión de tablas.
* Optimización de queries con planes de ejecución.
* Diseño de arquitecturas de alta disponibilidad

**Auditoría de base de datos**

* Implementar un log que registre quién inserta/actualiza/elimina datos.
* Usar triggers y tablas de auditoría.

**Sistema de respaldo y recuperación**

* Crear scripts que hagan backups automáticos diarios.
* Restaurar en otro servidor de pruebas.

**Dashboard de rendimiento**

* Usar DMVs para medir queries más lentas, índices usados/no usados, sesiones activas.
* Guardar en tablas históricas y mostrar en Power BI/Excel.

**Alta disponibilidad en un laboratorio**

* Configurar AlwaysOn en dos instancias de SQL Server con máquinas virtuales.
* Simular caídas y ver recuperación automática.

**Orden sugerido de estudio inmediato**

* Backups y restauración
* Automatización con Jobs
* Índices y performance básico
* Monitoreo y seguridad avanzada
* Alta disponibilidad (laboratorio)

# PROYECTOS POR CUBRIR

**BACKUS**

* Restaurar un backup completo.
* Restaurar backups en cadena (completo + diferencial + log).
* Restaurar hasta un punto en el tiempo.
* Automatizar backups con SQL Server Agent.
* **Proyecto**: crea un plan de respaldo automatizado (ejemplo: respaldo completo cada domingo, diferenciales diarios, logs cada hora). Documenta cómo restaurar en distintos escenarios

**SQL Server Agent y Jobs (automatización)**

* Crear jobs programados (respaldos, limpieza de logs, envío de reportes).
* Crear alertas para errores críticos (por ejemplo, si falla un backup).
* Monitoreo de ejecución de jobs.
* **Proyecto:** automatiza un job que respalde y otro que elimine logs viejos

**Índices y optimización de consultas (Performance Tuning básico)**

* Tipos de índices: clustered, non-clustered, unique, covering, filtered.
* Cómo leer un plan de ejecución.
* Detectar consultas lentas con DMVs.
* Uso de STATISTICS para ver cómo SQL Server decide el plan.
* **Proyecto:** crea un sistema con tablas grandes, ejecuta consultas sin índices y con índices, mide la diferencia con SET STATISTICS IO/TIME

**Monitoreo y seguridad avanzada**

* Monitoreo con DMVs
* Consultas más costosas.
* Índices usados/no usados.
* Sesiones bloqueadas.
* Seguridad
* Roles fijos de servidor vs. roles personalizados.
* Permisos a nivel de columna.
* Auditoría con triggers o SQL Server Audit.
* **Proyecto:** implementa una tabla de auditoría que registre usuario, fecha, acción en cada UPDATE/DELETE.

**Alta disponibilidad y recuperación avanzada (cuando domines lo anterior)**

* Log Shipping.
* Replicación.
* AlwaysOn Availability Groups.
* Mirroring (legacy, pero bueno conocerlo).
* **Proyecto:** montar un laboratorio con 2 máquinas virtuales y probar un failover.

# Python como aliado en tu camino (explicado en cada punto)

Python es muy poderoso porque conecta perfecto con **bases de datos, automatización y seguridad**. Veamos cómo:

### 1️⃣ Automatización

Con Python puedes crear scripts para **automatizar tareas repetitivas** de un DBA o sysadmin.

* Script que haga **backups automáticos** de la base de datos cada noche.
* Script que **revise logs de errores** y te mande un correo/alerta.
* Automatizar la creación de **usuarios y roles** en una BD.

**Trabajar con bases de datos**

**P**ython se conecta fácilmente con **MySQL, PostgreSQL, Oracle, SQL Server, MongoDB** y más.

* Con librerías como **SQLAlchemy, psycopg2, pyodbc,**.Esto es oro para un DBA, porque puedes **administrar y analizar bases de datos con código**.

### Análisis de logs

Los **logs** son archivos que guardan todo lo que pasa en un sistema o BD (errores, accesos, consultas, intentos de login).

Python puede leer esos archivos y:

Buscar intentos de **hackeo o accesos sospechosos**.

* Detectar errores frecuentes.
* Resumir actividad de usuarios.
* Esto conecta tu rol de **DBA con ciberseguridad**.

**ETL (Extract, Transform, Load)**

Python para ETL maneja datos de muchos formatos (**CSV, JSON, APIs, BD, Excel**).

* **Extract:** sacar datos de una BD de ventas.
* **Transform:** limpiar duplicados, convertir fechas.
* **Load:** cargar en un Data Warehouse para BI.

👉 Aquí se conecta con **Business Intelligenc**

# Lo que estamos haciendo hoy

## ****Módulo 1 – Introducción a SQL Server Agent****

Aprende:

Qué son los **Jobs** (tareas programadas).

Qué son los **Steps** (pasos dentro de un Job).

Qué son los **Schedules** (horarios para ejecutar).

Ejercicio:

Verifica que el servicio **SQL Server Agent** esté iniciado en tu servidor.

## ****Módulo 2 – Crear tu primer Job****

En SSMS → **SQL Server Agent → Jobs → New Job**.

Ponle un nombre: Backup\_Full\_MiBase.

Agrega un **Step** con este código:

BACKUP DATABASE MiBaseTO DISK = 'C:\Backups\MiBase\_Full.bak'WITH INIT;

Crea un **Schedule** para ejecutarlo todos los días a las 12:00 am.

Ejercicio: Ejecuta el job manualmente y revisa el archivo .bak.

## ****Módulo 3 – Jobs con múltiples Steps****

Objetivo: automatizar backups completos.

Crea un job con **varios Steps**:

Step 1: Backup Full (noche).

Step 2: Backup Differential (cada 4 horas).

Step 3: Backup Log (cada 15 min).

Ejemplo de código para el Log:

BACKUP LOG MiBaseTO DISK = 'C:\Backups\MiBase\_Log.trn';

Ejercicio: programa los tres y revisa que se creen en los horarios.

## ****Módulo 4 – Monitoreo y Alertas****

Objetivo: aprender a detectar errores en Jobs.

Revisa el historial de ejecución de un Job.

Configura notificaciones:

Enviar correo si un backup falla (requiere Database Mail configurado).

O revisar manualmente en SSMS → SQL Server Agent → Job History.

Ejercicio: provoca un error (ej. cambia la ruta de backup a una carpeta inexistente) y verifica la alerta.

## ****Módulo 5 – Buenas Prácticas****

Objetivo: aplicar lo que hacen los DBAs en empresas reales.

Guarda backups en un **disco distinto** al de SQL Server.

Usa nombres dinámicos con fecha en los backups:

DECLARE @FileName NVARCHAR(200);SET @FileName = 'C:\Backups\MiBase\_Full\_' +

CONVERT(VARCHAR(10), GETDATE(), 112) + '.bak';

BACKUP DATABASE MiBaseTO DISK = @FileNameWITH INIT;

Programa limpieza de backups viejos (ej. mantener solo 7 días).

Documenta tu estrategia de backups.

## ****Módulo 6 – Proyecto Final****

Objetivo: simular lo que harías como DBA en una empresa.

Crea una estrategia de backups automática para MiBase:

**Full Backup** → diario a medianoche.

**Differential** → cada 4 horas

**Transaction Log** → cada 15 min.

Monitorea los jobs una semana y revisa los archivos generados. Haz una **prueba de restauración** con los backups creados.

# REQUISITOS TRABAJOS

* Implementar/aplicar políticas de auditoría y seguridad en Bases de Datos.
* Aseguramiento de la integridad de los datos
* Brindar soporte para configuración e implementación de componentes de Bases de Datos.
* Velar por la continuidad y disponibilidad de todos los sistemas de información productivos de la Institución.
* Definir y Asegurar esquema de seguridad, permisos, perfiles, políticas, quotas, límites, etc. Control de la evolución de espacio en bases de datos.
* Atender Incidentes o tickets de servicio relacionados a Infraestructura o Base de datos a través de mesa de servicio

****CONOCIMIENTOS REQUERIDOS:****

* Oracle SOA Suite, Weblogic, Aplication Server.
* Conocimientos de administración de herramientas de Business Intelligence.
* Conocimientos de manejo de respaldo de la información y procesos de recuperación de desastres.
* Sistemas Operativos: Windows. Linux, Unix / Medio.
* Conocimiento avanzado en manejadores de Base de Datos: Oracle, SQL Server.
* Conocimiento intermedio de la metodología ITIL.
* Conocimientos de herramientas de integración y transformación de Datos.
* Conocimientos de TABLEAU.
* Conocimientos de DMBOK.