

Base de Datos I

IV. Conceptos de Manipulación de Datos

4.6. Manipulación de Datos – Los comandos (insert, update, delete)

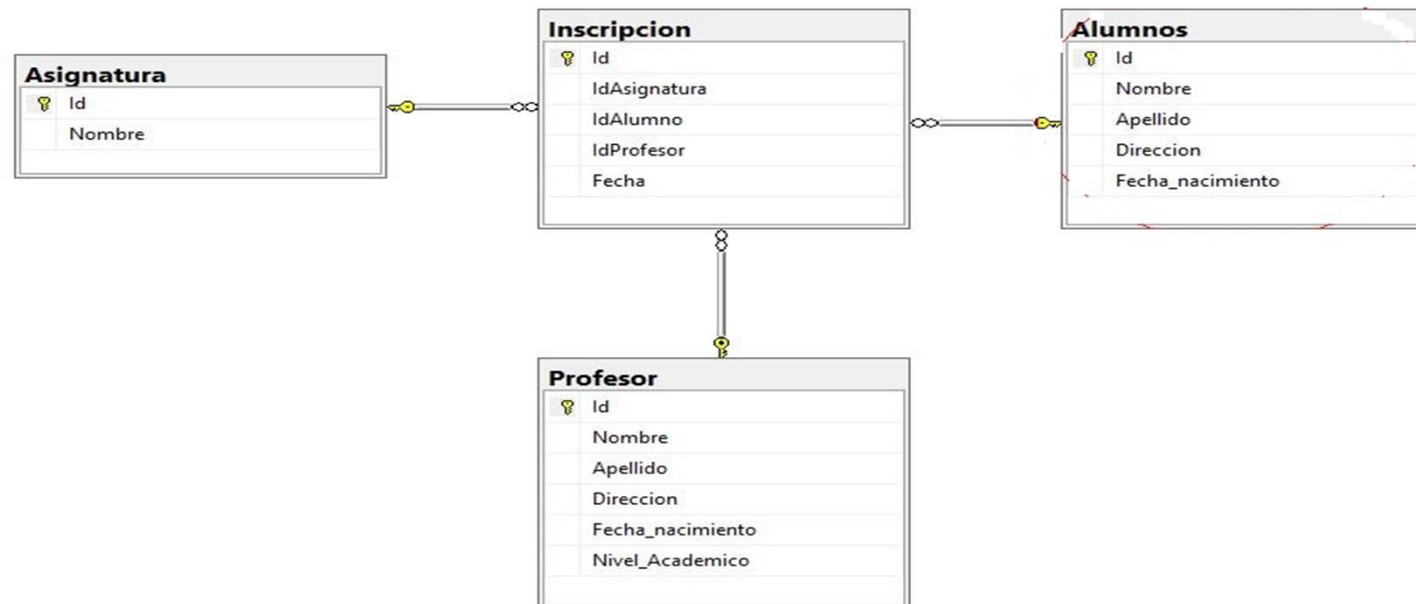
Consideraciones para la Inserción de Datos

INSERT

¿Cual entidad debo llenar o registrar primero?

En esta ocasión tomaremos como referencia la tabla **Alumnos** de nuestra Base de Datos creada, esta tabla tiene 5 columnas, que son: *Id*, *Nombre*, *Apellido*, *Direccion* y *Fecha_nacimiento*.

A continuación, el modelo conceptual.



4.6. Manipulación de Datos – Los comandos (insert, update, delete)

Si revisamos el **diagrama** nos podemos dar cuenta que la tabla Inscripcion es la que recibe las llaves foráneas de las tablas Alumnos, Profesor y Asignatura.

Por lo tanto, la lógica para **agregarles registros** es que primero debemos agregar:

- (1) asignaturas,
- (2) profesores y
- (3) alumnos;
- (4) Posteriormente podemos insertar datos a la tabla (4) Inscripcion, ya que los campos que están en la tabla inscripción (que relacionan las demás tablas), en ella son FK

El concepto antes descritos se conoce con el nombre de “Integridad Referencial” e indica que las tablas principales o de tipo de catálogo deben ser cargadas (tendrán una llave primaria y valores únicos) previas a llenar los valores de las tablas en donde ellas aparecen como llave foránea FK.

Es decir el IdAsignatura debe estar agregado en la tabla Asignatura, después que se han validado la integridad, recordando siempre que tanto para las PK como para las FK, se debe construir el Constraint correspondiente.

4.6. Manipulación de Datos – Los comandos (insert, update, delete)

Existen dos formas : lo podemos hacer de **uno en uno**, o podemos **agregar varios registros** a través de una misma instrucción

```
INSERT INTO «Nombre_Tabla» («columna1», «columna2», ...)  
VALUES («Dato1», «Dato2», ...);
```

```
INSERT INTO “Nombre_Tabla”  
VALUES («Dato1», «Dato2», «Dato3», ,etc);
```

Debemos respetar el orden que especificamos en las columnas y en values enviar los datos exactamente como los hemos especificado en las columnas.

Importante: Si una columna está definida como NOT NULL (No admite valores vacíos), debemos especificar y enviar siempre un dato para insertar, de lo contrario se producirá un error al ejecutar la instrucción INSERT.

4.6. Manipulación de Datos – Los comandos (insert, update, delete)

1. Insertar un registro a la vez con insert

```
INSERT INTO Alumnos (Id, Nombre, Apellido, Direccion, Fecha_nacimiento)  
VALUES ('0101', 'Franklin', 'Garcia', 'avenida 01', '12/01/80')
```

```
INSERT INTO Alumnos  
VALUES ('0102', 'Juan', 'Hernandez', 'avenida 01', '12/01/80')
```

2. Insertar varios registros en un solo insert

```
INSERT INTO Alumnos (Id, Nombre, Apellido, Direccion, Fecha_nacimiento)  
VALUES  
('0104', 'Franklin2', 'Garcia', 'avenida 01', '12/01/80'),  
('0105', 'Franklin3', 'Garcia', 'avenida 02', '12/01/70')  
('0103', 'Ronald3', 'Ponce', 'AltosPanama', '12/02/69')
```

4.6. Manipulación de Datos – Los comandos (insert, update, delete)

Insertar un registro utilizando Identity

Un campo numérico puede tener un atributo extra "identity". Los valores de un campo con este atributo genera valores secuenciales que se inician en 1 y se incrementan en 1 automáticamente.

- Se utiliza generalmente en campos correspondientes a códigos de identificación para generar valores únicos para cada nuevo registro que se inserta.
- Sólo puede haber un campo "identity" por tabla.
- Para que un campo pueda establecerse como "identity", éste debe ser entero (también puede ser de un subtipo de entero o decimal con escala 0

Para que un campo genere sus valores automáticamente, debemos agregar el atributo "identity" luego de su definición al crear la tabla:

```
CREATE TABLE LIBROS(  
codigo int identity,  
titulo varchar(40) not null,  
autor varchar(30),  
editorial varchar(15),  
precio money );
```

4.6. Manipulación de Datos – Los comandos (insert, update, delete)

```
-- INSTRUCCION INSERT BASICA --
```

```
USE BD_IGS116
```

```
GO
```

```
INSERT INTO BD_IGS116..CLIENTES  
VALUES ('RONALD ISAAC',  
'PONCE VELASQUEZ',  
'CIUDAD DE PANAMA,  
ANCON CENTENARIO',  
'6130-1066', 'ronald.ponce@utp.ac.pa',  
GETDATE())
```

```
-- INSERTAR DATOS CON IDENTITY --
```

```
USE BD_IGS116
```

```
GO
```

```
CREATE TABLE BD_IGS116..Persona (  
    Persona_id INT IDENTITY(1,1) PRIMARY KEY,  
    Nombres VARCHAR(50) NOT NULL,  
    Apellidos VARCHAR(50) NOT NULL,  
    Genero CHAR(1) NOT NULL  
);  
  
INSERT INTO BD_IGS116..Persona(Nombres, Apellidos, Genero)  
OUTPUT inserted.Persona_id  
VALUES('Ronald Isaac', 'Ponce Velásquez', 'M');  
  
INSERT INTO BD_IGS116..Persona(Nombres, Apellidos, Genero)  
OUTPUT inserted.Persona_id  
VALUES('Ilka María', 'Barría Zerna', 'F');
```

4.6. Manipulación de Datos – Los comandos (insert, update, delete)

Cuando un campo tiene el atributo "identity" no se puede ingresar valor para él, porque se inserta automáticamente tomando el último valor como referencia, o 1 si es el primero.

Para ingresar registros omitimos el campo definido como "identity", por ejemplo:

INSERT INTO LIBROS

(titulo, autor, editorial, precio)

VALUES ('El aleph','Borges','Emece',23);

Este primer registro ingresado guardará el valor 1 en el campo correspondiente al código, si hemos definido el **IDENTITY** (1,1,).



Cree una tabla llamada DVDS _INVENTARIO con donde podamos guardar la información de los DVDS musicales que tiene un cliente, para tal fin debe guardarse el titulo del DVD, Nombre del grupo o cantante, la descripción musical, fecha del emisión.

Utilice la instrucción **IDENTITY**, para incorporar los primeros cinco (5) DVDS musicales

4.6. Manipulación de Datos – Los comandos (insert, update, delete)

UPDATE

La sentencia **UPDATE** se utiliza en SQL Server para modificar valores en una tabla, generalmente se apoya de la instrucción condicional WHERE.

Si omitimos la cláusula WHERE, por defecto, modificará los valores del campo que se desee actualizar en todas las filas de la tabla.

La sintaxis de SQL UPDATE es:

```
UPDATE nombre_tabla  
SET columna1 = valor1, columna2 = valor2  
WHERE columna3 = valor3
```

Para modificar el valor de fecha y hora actual usando la función de fecha y hora **GETDATE**

4.6. Manipulación de Datos – Los comandos (insert, update, delete)

Si queremos cambiar el apellido2 'DESVENTURA' por 'BUENAVENTURA' ejecutaremos:

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
LUIS	LOPEZ	PEREZ
ANTONIO	DEGRACIA	DESVENTURA
PEDRO	RUIZ	GONZALEZ

```
UPDATE personas  
SET apellido2 = 'BUENAVENTURA'  
WHERE nombre = 'ANTONIO'  
AND apellido1 = 'DEGRACIA'
```

4.6. Manipulación de Datos – Los comandos (insert, update, delete)

DELETE

La instrucción DELETE permite eliminar uno o múltiples registros. Incluso todos los registros de una tabla, dejándola vacía.

La condición, como siempre, define las condiciones que deben cumplir los registros que se desean eliminar.

La instrucción DELETE puede tener un error si infringe un desencadenador o intenta quitar una fila a la que hacen referencia datos de otra tabla con una restricción **FOREIGN KEY**. Si la instrucción DELETE quita varias filas y cualquiera de las filas eliminadas infringe un desencadenador o restricción, se cancela la instrucción, se devuelve un error y no se elimina ninguna fila.

Por ejemplo, la siguiente instrucción eliminará todas las filas de LA TABLA : GRUPO_CLIENTES

```
DELETE FROM [BD_..]. GRUPO_CLIENTES
```

4.6. Manipulación de Datos – Los comandos (insert, update, delete)

1. Eliminar un número de registros aleatoriamente en una tabla SQL Server

La siguiente instrucción DELETE elimina 21 filas aleatorias de la tabla GRUPO_CLIENTES

```
DELETE TOP (21) FROM [BD_..].. GRUPO_CLIENTES
```

Este es el mensaje emitido por el servidor SQL:

(21 row(s) affected)

2. Eliminar un porcentaje de registros aleatoriamente en una tabla SQL Server

La siguiente instrucción DELETE elimina el 5 por ciento de las filas aleatorias de la tabla GRUPO_CLIENTES

```
DELETE TOP (5) PERCENT FROM [BD_..]..GRUPO_CLIENTES
```

SQL Server regresa el siguiente mensaje, en el cual indica que se han eliminado 15 filas ($300 \times 5\% = 15$).

(15 row(s) affected)

4.6. Manipulación de Datos – Los comandos (insert, update, delete)

3. Eliminar algunas filas de una tabla de SQL Server con una condición

La siguiente declaración **DELETE** elimina todos los productos cuyo año de modelo es 2017

```
DELETE FROM [BD_1M3215]..GRUPO_CLIENTES  
WHERE Annio_Ingreso = '2017';
```

Aquí está el mensaje de salida:

(75 row(s) affected)

4. Eliminar todas las filas de una tabla de SQL Server

La siguiente instrucción **DELETE** elimina todas las filas de la tabla GRUPO_CLIENTES :

```
DELETE FROM [BD_1M3215]..GRUPO_CLIENTES
```

Tenga en cuenta que, si desea eliminar todas las filas de una tabla grande, debe usar la instrucción **TRUNCATE TABLE**, que es más rápida y eficiente.

4.6. Manipulación de Datos – Ordenamiento y Agrupamiento

1. **ORDER BY:** Ordenar los resultados.
2. **GROUP BY:** Agrupar registros para realizar operaciones agregadas.
3. **HAVING:** Filtrar después del agrupamiento.

Ejemplos de cómo ordenar y agrupar datos.

Sintaxis:

```
SELECT Columna1, Columna2  
FROM NombreTabla  
ORDER BY Columna1 [ASC | DESC];
```

Ejemplo práctico:

```
SELECT * FROM CLIENTES  
ORDER BY Apellido ASC;
```

4.6. Manipulación de Datos – Ordenamiento y Agrupamiento

GROUP BY y HAVING

Agrupar los resultados y aplicar funciones de agregación.

Sintaxis:

```
SELECT Columna1, COUNT(*)  
FROM NombreTabla  
GROUP BY Columna1  
HAVING COUNT(*) > 1;
```

ID	Nombre	Apellido
1	Juan	Pérez
2	María	Gómez
3	Pedro	López
4	María	Barría
5	Juan	Gómez
NULL	NULL	NULL

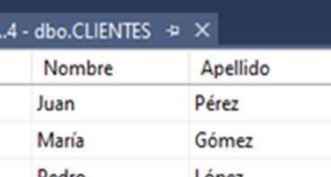
Ejemplo Práctico

```
INSERT INTO CLIENTES (ID, Nombre, Apellido)  
VALUES (5, 'Juan', 'Gómez');
```

```
SELECT Nombre, COUNT(*)  
FROM CLIENTES  
GROUP BY Nombre  
HAVING COUNT(*) > 1;
```

	Nombre	Cuenta
1	Juan	2
2	María	2

4.6. Manipulación de Datos – Ordenamiento y Agrupamiento



11 - SS-RONALD\MSSQLSERVER_202...

MSSQLS...4 - dbo.CLIENTES

Nombre	Apellido
Juan	Pérez
María	Gómez
Pedro	López
María	Barria
Juan	Gómez
NULL	NULL

of 5

Solution1 - S5-RONALD\MSSQLSERVER_2022.BD_1IL124 - dbo...

S5-RONALD\MSSQL...24 - dbo.PEDIDOS

	Numero	ID_cliente	Cod_formaPago	Fecha
▶	20	1	1	2024-01-02
	22	2	1	2024-02-02
	24	3	1	2024-03-02
	27	1	2	2024-05-02
	29	2	2	2024-09-02
	31	3	2	2024-12-02
	33	1	3	2024-12-03
	35	2	3	2024-12-04
	37	3	3	2024-12-04
•	NULL	NULL	NULL	NULL

of 9

Solution1 - S5-RONALD\MSSQLSERVER_2022.BD_1IL124 - dbo.PEDIDO_DETALLE

S5-RONALD\MSSQL....PEDIDO_DETALLE

	ID_Pedido	ID_Cliente	Cod_Producto	Descripcion	Cantidad	Precio
▶	20	1	C1	CELULAR HON...	3	560.00
	20	1	R2	RELOG INT AM...	5	245.00
	20	1	T3	TABLET SAMSU...	2	390.00
	22	1	L5	LENTES PARA E...	10	90.00
	22	1	M4	MODEM INALA...	2	275.00
	22	1	T8	TECLADO PARA...	3	27.00
	27	2	A2	ALMOHADAS	20	12.00
	27	2	S2	JUEGO DE SAB...	4	120.00
	27	2	T9	TRAJE DE BAÑ...	3	59.00
	29	2	J8	JUEGO DE ANIL...	1	996.00
	29	2	L5	LENTES PARA E...	50	60.00
	29	2	Z5	ZAPATOS COLO...	5	158.00
	33	3	L5	LENTES PARA E...	100	40.00
	33	3	R2	RELOG INT AM...	2	245.00
	33	3	T8	TECLADO PARA...	4	27.00
	35	3	A2	ALMOHADAS	10	12.00
	35	3	S2	JUEGO DE SAB...	2	120.00
	35	3	Z5	ZAPATOS COLO...	10	118.00
*	NULL	NULL	NULL	NULL	NULL	NULL

◀ | 1 of 18 ▶▶▶⌂

4. Conceptos de Manipulación de Datos (Distinct, Group By , Having)

SQL: Distinct

La cláusula DISTINCT en SQL es una herramienta muy útil para eliminar los valores duplicados en una consulta. Al utilizar DISTINCT, se garantiza que cada fila en el resultado de la consulta sea única en cuanto a los campos especificados.

SELECT DISTINCT carrera FROM ALUMNO;

Esta consulta devolverá una lista de todas las carreras únicas en la tabla «carreras», sin duplicados.

Por ejemplo, si tienes una tabla «ALUMNO» con un campo «carrera», puedes utilizar la cláusula DISTINCT para seleccionar todas las carreras diferentes de la tabla:

```
SELECT DISTINCT a.idCarrera, c.NombreCarrera
FROM Alumno a
INNER JOIN Carrera c
ON c.idCarrera=a.idCarrera
```

4. Conceptos de Manipulación de Datos (Distinct, Group By , Having)

La cláusula DISTINCT también se puede utilizar con varios campos, si deseas obtener una lista de todos los clientes únicos en la tabla «clientes» (sin importar si un cliente tiene varios registros), puedes utilizar la siguiente consulta:

```
SELECT DISTINCT nombre, apellido FROM clientes;
```

En este caso, se devuelve una lista de nombres y apellidos únicos de clientes en la tabla, sin duplicados.

La cláusula DISTINCT es especialmente útil en combinación con otras cláusulas, como GROUP BY y HAVING.

GROUP BY permite agrupar los resultados de una consulta por uno o varios campos,

HAVING permite filtrar los resultados de una consulta en función de una condición específica.

La cláusula DISTINCT en SQL es una herramienta valiosa para eliminar los valores duplicados en una consulta. Se puede utilizar con uno o varios campos para obtener una lista única de valores, lo que es especialmente útil al trabajar con tablas grandes y complejas. La cláusula DISTINCT también se puede combinar con otras cláusulas para proporcionar un control adicional sobre los resultados de la consulta.

4. Conceptos de Manipulación de Datos (Distinct, Group By , Having)

SQL: Having

La cláusula HAVING se usa a menudo con la cláusula **GROUP BY** para filtrar grupos en una lista específica de condiciones. A continuación se ilustra la sintaxis de la cláusula HAVING:

```
SELECT select_list FROM table_name GROUP BY group_list HAVING conditions;
```

En la sintaxis anterior, la cláusula **GROUP BY** lista las filas que se agruparan y la cláusula **HAVING** aplica una o más condiciones a cualquiera de estos grupos, siempre y cuando los especifiques en el HAVING.

Solo los grupos que cumplen con las condiciones establecidas se evalúan como TRUE y se incluyen en el resultado de la consulta **SELECT**.

En otras palabras, los grupos para los que la condición se evalúa como FALSE o UNKNOWN se filtran y no los regresara en la consulta.

4.6. Manipulación de Datos – Instrucciones Lógicas

AND, OR, LIKE, BETWEEN, IN

AND: Criterio de filtro para coincidencias específicas donde ambos criterios se cumplen

Ejemplo : Seleccionar ventas realizadas por el cliente con ID 1 y producto con ID 1

```
SELECT * FROM VENTAS
```

```
WHERE ClienteID = 1 AND ProductID = 1;
```

OR: Criterio de filtro donde cualquiera de las condiciones de coincidencias se pueden cumplir.

Ejemplo: Seleccionar ventas realizadas por los clientes con ID 1 o 2

```
SELECT * FROM VENTAS
```

```
WHERE ClienteID = 1 OR ClienteID = 2;
```

4.6. Manipulación de Datos – Instrucciones Lógicas

AND, OR, LIKE, BETWEEN, IN

BETWEEN: Se da cuando la condición de búsqueda o criterio se encuentra dentro de un rango especificado

EJEMPLO: Seleccionar productos cuyo precio esté entre 40 y 500

```
SELECT * FROM PRODUCTOS  
WHERE Precio BETWEEN 40 AND 500;
```

IN: Actúa como un filtro, devolviendo únicamente los valores productos de la búsqueda solicitada.

EJEMPLO: : Tienes una tabla llamada PRODUCTOS con las siguientes columnas: ID_PRODUCTO, NOMBRE, y PRECIO.

```
SELECT NOMBRE, PRECIO FROM PRODUCTOS  
WHERE ID_PRODUCTO IN (1, 3, 5);
```

4.6. Manipulación de Datos – Instrucciones Lógicas

AND, OR, LIKE, BETWEEN, IN

LIKE: Buscar patrones dentro de una columna de texto. Permite filtrar registros que coincidan parcial o totalmente con una cadena específica

Ejemplo: Seleccionar productos cuyo nombre comience con la letra "L"

```
SELECT * FROM PRODUCTOS  
WHERE NombreProducto LIKE 'L%';
```

- **% (Porcentaje):** Representa cualquier secuencia de caracteres (incluyendo ninguna). LIKE 'A%';
- **_ (Guión bajo):** Representa exactamente un carácter en una posición específica . LIKE 'J_n%';
- **[] (Corchetes):** Define un conjunto de caracteres posibles para una posición específica. LIKE '[AEIOU]%';
- **(Guion dentro de corchetes):** Se utiliza para definir un rango de caracteres dentro de los corchetes. LIKE '[A-D]%';
- **[^] (Caret dentro de corchetes):** Representa una negación dentro del conjunto de caracteres, es decir, cualquier carácter **excepto** los especificados. LIKE '[^A]%';

4.6. Manipulación de Datos – Funciones Agregadas

SUM, COUNT, AVG, MIN, MAX, STDEV, VAR

SUM: Suma valores dentro de un campo específico de una tabla

Ejemplo: Sumar la cantidad total de productos vendidos

```
SELECT SUM(Cantidad) AS TotalProductosVendidos  
FROM VENTAS;
```

COUNT: Cuenta la cantidad total de productos vendidos

Ejemplo : Contar el número total de ventas

```
SELECT COUNT(*) AS NumeroDeVentas  
FROM VENTAS;
```

4.6. Manipulación de Datos – Funciones Agregadas

SUM, COUNT, AVG, MIN, MAX, STDEV, VAR

MIN y MAX: Encuentra el máximo y el mínimo valor dentro de una columna específica de una Tabla

Ejemplo: Obtener el precio mínimo y máximo de los productos

```
SELECT MIN(Precio) AS PrecioMinimo, MAX(Precio) AS PrecioMaximo  
FROM PRODUCTOS;
```

AVG: Promedia una serie de valores contenidos en una columna de una tabla específica

Ejemplo: Obtener el precio promedio de los productos

```
SELECT AVG(Precio) AS PrecioPromedio  
FROM PRODUCTOS;
```


4.6. Manipulación de Datos – Funciones Agregadas

SUM, COUNT, AVG, MIN, MAX, STDEV, VAR

STDEV (Desviación Estándar): Calcula la desviación estándar de un conjunto de valores, lo que mide cuánto se desvían los datos en promedio de su media.

- **Ejemplo:**

```
SELECT STDEV(MONTO) AS DesviacionEstandar FROM VENTAS;
```

VAR (Varianza): Calcula la varianza de los datos, que es la media de las diferencias al cuadrado respecto a la media. Es una medida de dispersión.

- **Ejemplo:**

```
SELECT VAR(MONTO) AS VarianzaVentas FROM VENTAS;
```

- **Explicación:** Calcula la varianza de los valores en la columna MONTO para medir la dispersión de los datos.

4.6. Manipulación de Datos – Funciones de Fecha

GETDATE(), DATEADD(), DATEDIFF(),

GETDATE(): Función que obtienen y te muestra la fecha actual del servidor

Ejemplo: Obtener la fecha y hora actual

```
SELECT GETDATE() AS FechaActual;
```

DATEADD(): Adiciona una cantidad de días especificados por el usuario

Ejemplo: Agregar 5 días a la fecha actual

```
SELECT DATEADD(DAY, 5, GETDATE()) AS FechaFutura;
```

DATEDIFF(): Obtiene la resta o diferencia de entre dos fechas seleccionadas

Ejemplo: Calcular la diferencia en días entre dos fechas

```
SELECT DATEDIFF(DAY, '2024-09-20', '2024-09-25') AS DiasDeDiferencia;
```

4.6. Manipulación de Datos – Funciones de Fecha

GETDATE(), DATEADD(), DATEDIFF(), DATENAME(), DATEPART()

DATENAME(): Devuelve una cadena que representa la parte especificada de una fecha.

Ejemplo : Devuelve el nombre del mes actual (como una cadena) utilizando la función DATENAME() con el parámetro MONTH. Si la fecha es octubre, por ejemplo, el resultado será "October".

```
SELECT DATENAME(MONTH, GETDATE()) AS MesActual;
```

DATEPART(): Devuelve una parte específica de una fecha como un número entero (día, mes, año, etc.).

Ejemplos: Devuelve el año actual. Por ejemplo, si la fecha es 2024-10-03, el resultado será 2024.

```
SELECT DATEPART(YEAR, GETDATE()) AS AñoActual;
```

Ejemplo: Devuelve el número del mes actual. Por ejemplo, si la fecha es octubre, el resultado será 10.

```
SELECT DATEPART(MONTH, GETDATE()) AS MesActual;
```

Ejemplo: Devuelve el día del mes actual. Por ejemplo, si la fecha es 3 de octubre, el resultado será 3.

```
SELECT DATEPART(DAY, GETDATE()) AS DiaActual;
```

4. Conceptos de Manipulación de Datos - Ejemplo T-SQL

```
USE BD_IGS116
GO
```

```
SELECT Cod_factura, Descrpccion, MAX(Cantidad * Precio) Sub_tot1, MIN((Cantidad * Precio)*.07) ITBM,
AVG((Cantidad* Precio)+ ((Cantidad* Precio)*.07)) Sub_tot2,
SUM((Cantidad * Precio) + ((Cantidad* Precio)*.07)) Total_Final
FROM BD_IGS116..FACTURA_VENTAS_DETALLE
WHERE Descrpccion like 'T%'
GROUP BY Cod_factura, Descrpccion
```

```
SELECT Descrpccion, Cantidad
FROM BD_IGS116..FACTURA_VENTAS_DETALLE
WHERE Cantidad IN (5,3,2)
```

```
SELECT Descrpccion, Cantidad, Precio
FROM BD_IGS116..FACTURA_VENTAS_DETALLE
WHERE Precio BETWEEN 2 AND 5
```

150 %

Results Messages

	Descrpccion	Cantidad	Precio
1	BROCHAS MEDIANAS	7	4.50
2	LIJAS DE DE .49	3	2.95

4. Conceptos de Manipulación de Datos (Distinct, Group By , Having)

HAVING y COUNT

La siguiente declaración usa la cláusula HAVING y COUNT para encontrar todos los clientes que realizaron al menos dos pedidos al año:

```
SELECT customer_id, YEAR (order_date)
fecha, COUNT (order_id) order_count
FROM sales.orders
GROUP BY customer_id, YEAR (order_date)
HAVING COUNT (order_id) >= 2
ORDER BY customer_id;
```

HAVING y SUM

La siguiente declaración encuentra las órdenes de venta cuyos valores netos son mayores a 20,000 usando la función SUM:

```
SELECT order_id,
SUM ( quantity * list_price * (1 -
discount) ) net_value
FROM sales.order_items
GROUP BY order_id
HAVING SUM ( quantity * list_price *
(1 - discount) ) > 20000
ORDER BY net_value;
```

4. Conceptos de Manipulación de Datos (Distinct, Group By , Having)

HAVING Y MAX() y MIN()

La siguiente declaración usa las funciones para encontrar primero los precios de lista máximos y mínimos en cada categoría de producto. Filtra la categoría que tiene el precio de lista máximo superior a 4000 o el precio de lista mínimo inferior a 500:

```
SELECT category_id,  
MAX (list_price) max_list_price,  
MIN (list_price) min_list_price  
FROM production.products  
GROUP BY category_id  
HAVING MAX (list_price) > 4000 OR MIN  
(list_price) < 500;
```

HAVING y AVG()

La siguiente declaración usa la función AVG() para encontrar las categorías de productos cuyos precios de lista promedio están entre 500 y 1,000:

```
SELECT category_id,  
AVG (list_price) avg_list_price  
FROM production.products  
GROUP BY category_id  
HAVING  
AVG (list_price) BETWEEN 500 AND 1000;
```