UNIVERSIDAD FEDERICO VILLAREAL

FACULTAD DE INGENIERIA INDUSTRIAL Y SISTEMAS

ESCUELA DE INGENIERIA DE SISTEMAS

GUIA DE LABORATORIO Nº 1

NOMBRE DE LA PRÁCTICA : CREACION DE ARCHIVOS DE BASE DE DATOS

CURSO : SISTEMA DE BASE DATOS

DOCENTE : ING. MAIKE JAUREGUI

CICLO : 2018-II

I. OBJETIVOS

1. Crear la base de datos en SQL SERVER
2. Describir los tipos de archivos que lo conforman

II. INTRODUCCION TEORICA

Crear una base de datos se puede reducir a una simple instrucción como esta:

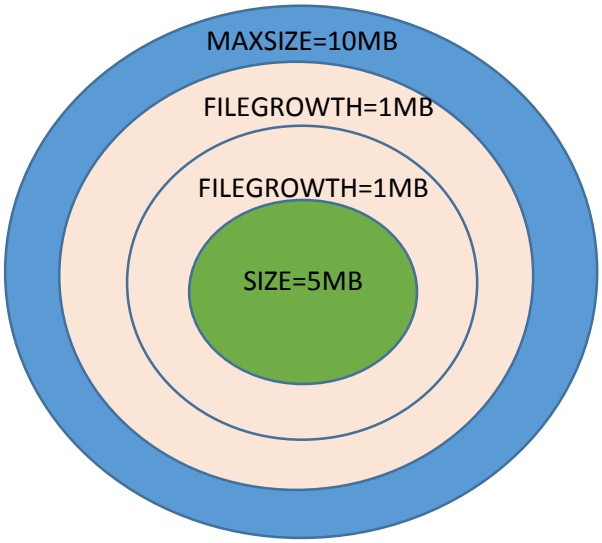
```
create database bd1
```

En una versión más extendida de la instrucción CREATE DATABASE se pueden indicar los detalles de los archivos:

```
create database dbcurso on primary
(name='dbcurso',
filename='D:\Datos\dbcurso.mdf',
size=5 MB, maxsize=10 MB, filegrowth=1 MB)
log on
(name='dbcurso_log',
filename='D:\Datos\dbcurso_log.ldf',
size=5 MB, maxsize=10 MB, filegrowth=1 MB)
```

Parametro	Descripcion
Name	Nombre lógico del archivo de datos. Facilita posteriormente el tratamiento del archivo a través de instrucciones T-SQL
Filename	Ubicación física del archivo. La cuenta de servicio del SQL SERVER debe tener permisos de escritura en la carpeta elegida
Size	SQL SERVER reserva este espacio en disco para que no sea utilizado por otras aplicaciones
Filegrowth	Cuando el tamaño inicial sea totalmente ocupado con información, SQL SERVER reservara nuevamente en disco la cantidad indicada en este parámetro, que puede venir dada en KB, MB o en porcentaje
Maxsize	Tamaño máximo del archivo. Si no se indica, el archivo crecerá hasta llenar el disco

En el ejemplo que se acaba de presentar se crea un archivo principal de extensión mdf de tamaño inicial de 5 MB, cuando este espacio se agote, este se expandirá en 1 MB más para tener espacio libre y meter más registros o tablas y cuando se agote nuevamente el archivo físico de 1MB, éste se expandirá en 1 MB nuevamente y así en lo sucesivo hasta alcanzar el máximo 10 MB ya de ahí no crecerá más.



Para almacenar una base de datos se emplean tres tipos de archivos:

- ✓ El archivo de datos primario: contiene la información de inicio de la base de datos. El archivo principal se utiliza también para almacenar datos. Cada base de datos tiene un único archivo de datos primario. Por convenio el nombre de un archivo de datos primario tiene una extensión mdf.

- ✓ El archivo de datos secundario almacena todos los datos que no caben en el archivo principal de datos. Las bases de datos no necesitan archivos de datos secundarios si el archivo principal es suficientemente grande para contener todos los datos de la base de datos.

- ✓ El archivo de registro de transacciones: contiene la información del registro que se utiliza para recuperar la base de datos. Debe haber al menos un archivo de registro de transacciones para cada base de datos, aunque puede haber más de uno. Después de cada modificación de la base de datos – ocurrencia de transacción, un registro es escrito en el registro de transacciones

TIPO DE ARCHIVO	EXTENSION DE NOMBRE DE ARCHIVO RECOMENDADA
Archivo de datos principal	.mdf
Archivo de datos secundario	.ndf
Archivo de registro de transacciones	.ldf

III. REQUERIMIENTOS

- SQL SERVER
- GUIA DE LABORATORIO Nº 1

IV. PROCEDIMIENTOS

Parte 1: Iniciando sesión desde SQL Server Managment Studio

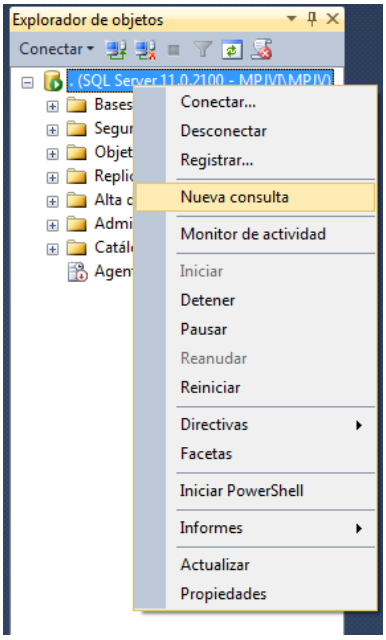
1. Hacer clic en el botón **Inicio**
2. Hacer clic en la opción **Todos los programas** y hacer clic en **Microsoft SQL Server 2012**

Para conectarse con el servidor de base de datos elija los siguientes parámetros de autenticación:

Tipo de servidor : Database Engine
Nombre del servidor : local
Nota : Se puede colocar punto (.)
Autenticación : Autenticación Windows

Parte 2: Manipulando el explorador de objetos

Ejecutar una nueva consulta



Ejemplo 1

Crear la base de datos DBEjemplo_01 que especifique los archivos de registro de datos y de transacciones. El archivo de datos con tamaño 5MB, máximo tamaño 20MB, aumento de 5MB, y de transacciones 2MB, 15MB, 5MB respectivamente, con una ubicación física en la unidad D, carpeta Datos.

```
create database dbejemplo_01 on primary
(name='dbejemplo_01_dat',
filename='D:\Datos\db_01dat.mdf',
size=5 MB, maxsize=20 MB, filegrowth=5 MB)
log on
(name='dbejemplo_01_log',
filename='D:\Datos\db_01log.ldf',
size=2 MB, maxsize=15 MB, filegrowth=5 MB)
```

Ejemplo 2

Crear la base de datos dbejemplo_02 mediante la especificación de múltiples archivos de registro de de datos y de transacciones

```
create database dbejemplo_02 on primary
(name='logi1',
filename='D:\Datos\logidat1.mdf',
size=5 MB, maxsize=10 MB, filegrowth=10 MB),
(name='arch2',
filename='D:\Datos\logidat2.ndf',
size=5 MB, maxsize=10 MB, filegrowth=10 MB),
(name='arch3',
filename='D:\Datos\logidat3.ndf',
size=5 MB, maxsize=10 MB, filegrowth=10 MB)

log on
(name='logilog1',
filename='D:\Datos\logilog1.ldf',
size=5 MB, maxsize=10 MB, filegrowth=10 MB),
(name='archlog2',
filename='D:\Datos\logilog2.ldf',
size=5 MB, maxsize=10 MB, filegrowth=10 MB)
```

Ejemplo 3

Crear una base de datos dbejemplo_03 especificando un único archivo

```
create database dbejemplo_03 on primary
(name='archidat1',
filename='D:\Datos\archidat1.mdf',
size=5 MB, maxsize=10 MB, filegrowth=10 MB)
```

Ejemplo 4

Crear una base de datos sin especificar los archivos

create database dbejemplo_04

V. EJERCICIO COMPLEMENTARIO

1. **Crear la base de datos dbejemplo_05 con un único archivo con las siguientes características:**
   - ✓ **Nombre de la base de datos : dbejemplo_05**
   - ✓ **Nombre de archivo lógico : dbejemplo_05_dat**
   - ✓ **Nombre de archivo físico : D:\Datos\db_05dat.mdf**
   - ✓ **Tamaño inicial : 5 MB**
   - ✓ **Tamaño máximo : 20 MB**
   - ✓ **Porcentaje incremento archivo : 30%**

2. **Crear la base de datos dbejemplo_06 que especifique los archivos de registro de datos y de transacciones con las siguientes características:**

   Nombre de la base de datos : dbejemplo_06

   Para los archivos de registros de datos
   - ✓ **Nombre de archivo lógico : dbejemplo_06_dat**
   - ✓ **Nombre de archivo físico : D:\Datos\db_06dat.mdf**
   - ✓ **Tamaño inicial : 10 MB**
   - ✓ **Tamaño máximo : 30 MB**
   - ✓ **Porcentaje incremento archivo : 25%**

   Para los archivos de registro de transacciones:
   - ✓ **Nombre de archivo lógico : dbejemplo_06_log**
   - ✓ **Nombre de archivo físico : D:\Datos\db_06dat.ldf**
   - ✓ **Tamaño inicial : 10 MB**
   - ✓ **Tamaño máximo : 30 MB**
   - ✓ **Porcentaje incremento archivo : 25%**

3. **Crear la base de datos dbejemplo_07 mediante la especificación de múltiples archivos de registro de datos y de transacciones. La base de datos debe tener 3 archivos de datos de 15 MB cada uno y 2 archivos de registro de transacciones de 10 MB con las siguientes características:**

   Nombre de la base de datos : dbejemplo_07

   Para el archivo principal
   - ✓ **Nombre de archivo lógico : dbejemplo_07**
   - ✓ **Nombre de archivo físico : D:\Datos\db_07dat.mdf**
   - ✓ **Tamaño inicial : 15 MB**
   - ✓ **Tamaño máximo : 50 MB**
   - ✓ **Porcentaje incremento archivo : 10%**

Para el archivo secundario 1

- ✓ **Nombre de archivo lógico : dbejemplo_07_1**
- ✓ **Nombre de archivo físico : D:\Datos\db_07_1dat.ndf**
- ✓ **Tamaño inicial : 15 MB**
- ✓ **Tamaño máximo : 50 MB**
- ✓ **Porcentaje incremento archivo : 10%**

Para el archivo secundario 2

- ✓ **Nombre de archivo lógico : dbejemplo_07_2**
- ✓ **Nombre de archivo físico : D:\Datos\db_07_2dat.ndf**
- ✓ **Tamaño inicial : 15 MB**
- ✓ **Tamaño máximo : 50 MB**
- ✓ **Porcentaje incremento archivo : 10%**

Para los archivos de registros de transacciones 1

- ✓ **Nombre de archivo lógico : dbejemplo_07_1_log**
- ✓ **Nombre de archivo físico : D:\Datos\db_07_1log.ldf**
- ✓ **Tamaño inicial : 10 MB**
- ✓ **Tamaño máximo : 30 MB**
- ✓ **Porcentaje incremento archivo : 10%**

Para los archivos de registros de transacciones 1

- ✓ **Nombre de archivo lógico : dbejemplo_07_2_log**
- ✓ **Nombre de archivo físico : D:\Datos\db_07_2log.ldf**
- ✓ **Tamaño inicial : 10 MB**
- ✓ **Tamaño máximo : 30 MB**
- ✓ **Porcentaje incremento archivo : 10%**

UNIVERSIDAD FEDERICO VILLAREAL

FACULTAD DE INGENIERIA INDUSTRIAL Y SISTEMAS

ESCUELA DE INGENIERIA DE SISTEMAS

GUIA DE LABORATORIO Nº 2

NOMBRE DE LA PRÁCTICA : MODIFICACIONES A LOS ARCHIVOS BASE DE DATOS

CURSO : SISTEMA DE BASE DATOS

DOCENTE : ING. MAIKE JAUREGUI

CICLO : 2018-II

I. OBJETIVOS

1. Modificar los archivos de la base de datos SQL SERVER
2. Cambiar el nombre de la base de datos
3. Eliminar la base de datos

II. INTRODUCCION TEORICA

Si se desea cambiar el tamaño de la base de datos, puede usar la instrucción ALTER DATABASE.

III. REQUERIMIENTOS

- SQL SERVER
- GUIA DE LABORATORIO Nº 2

IV. PROCEDIMIENTOS

Crear una base de datos dbejemplo_03 especificando un único archivo

```
create database dbejemplo_03 on primary
(name='archidat1',
filename='D:\Datos\archidat1.mdf',
size=5 MB, maxsize=10 MB, filegrowth=10 MB)
```

Ejemplo 1

Modificar la base de datos dbejemplo_03 para agregarle un archivo de datos de 5MB de extensión ndf

```
alter database dbejemplo_03
add file
(
name=test1dat2,
filename='D:\Datos\t1dat2.ndf',
size=5MB,
maxsize=10MB,
filegrowth=5MB)
```

Ejemplo 2

Agregar un grupo de archivos a la base de datos dbejemplo_03. Dos archivos de extensión ndf , de 5MB al grupo de archivos.

```
alter database dbejemplo_03
add file
(
name=test1dat3,
filename='D:\Datos\t1dat3.ndf',
size=5MB,
maxsize=10MB,
filegrowth=5MB),
(
name=test1dat4,
filename='D:\Datos\t1dat4.ndf',
size=5MB,
maxsize=10MB,
filegrowth=5MB)
```

Expansión de la base de datos

Ejemplo 3

Aumentar el tamaño de test1dat3 de la base de datos dbejemplo_03 a 20 MB

```
alter database dbejemplo_03
modify file
(name=test1dat3,
size=20MB)
```

Ejemplo 4

Cambiar el nombre de la base de datos dbejemplo_03 por dbejemplo_03c

```
exec sp_renamedb 'dbejemplo_03', 'dbejemplo_003c'
```

Ejemplo 5

Eliminar la base de datos `dbejemplo_003c`

```
use master
go
drop database dbejemplo_003c
go
```

V. Ejercicio Complementario

1. Modificar la base de datos dbejemplo_07 para agregarle archivos de datos de la siguiente manera:

   Para el archivo principal
   - ✓ **Nombre de archivo de datos : adicional_dat**
   - ✓ **Nombre de archivo físico : D:\Datos\adicionaldat.ndf**
   - ✓ **Tamaño inicial : 3 MB**
   - ✓ **Tamaño máximo : 6 MB**
   - ✓ **Porcentaje incremento archivo : 5%**

2. Modificar la base de datos dbejemplo_07 para agregarle 2 archivos de datos, los archivos de datos tienen las siguientes características:

   Para el archivo de datos 1
   - ✓ **Nombre de archivo de datos : adicional2_dat**
   - ✓ **Nombre de archivo físico : D:\Datos\adicional2dat.mdf**
   - ✓ **Tamaño inicial : 5 MB**
   - ✓ **Tamaño máximo : 10 MB**
   - ✓ **Porcentaje incremento archivo : 2 MB**

   Para el archivo de datos 2
   - ✓ **Nombre de archivo de datos : adicional3_dat**
   - ✓ **Nombre de archivo físico : D:\Datos\adicional3dat.mdf**
   - ✓ **Tamaño inicial : 5 MB**
   - ✓ **Tamaño máximo : 10 MB**
   - ✓ **Porcentaje incremento archivo : 2 MB**

3. Cambiar el tamaño de la base de datos dbejemplo_07, aumentándole el tamaño del archivo de datos de la siguiente manera:
   Nombre de archivo de datos : adicional_dat
   Aumentar tamaño : 20 MB
4. Cambiar el nombre de la base de datos dbejemplo_07 por el de dbejemplo_cambio
5. Eliminar la base de datos dbejemplo_cambio

UNIVERSIDAD FEDERICO VILLAREAL

FACULTAD DE INGENIERIA INDUSTRIAL Y SISTEMAS

ESCUELA DE INGENIERIA DE SISTEMAS

GUIA DE LABORATORIO Nº 3

NOMBRE DE LA PRÁCTICA : MANEJO DE SENTENCIAS SQL SERVER

CURSO : SISTEMA DE BASE DATOS

DOCENTE : ING. MAIKE JAUREGUI

CICLO : 2018-II

I. OBJETIVOS

1. Conocer las instrucciones IF-ELSE en SQL SERVER
2. Conocer las instrucciones CASE en SQL SERVER
3. Conocer las instrucciones WHILE en SQL SERVER

II. INTRODUCCION TEORICA

1. IF ... ELSE

Impone condiciones en la ejecución de una instrucción Transact-SQL.
Su condición se ejecuta si la condición se cumple (cuando la expresión devuelve true).

Sintaxis para IF ... ELSE

```
IF ( Boolean_expression ) --SI (Expresion_Booleana)

  BEGIN --EMPEZAR
    Sentencia de instrucciones SQL
    END --FIN

ELSE

  BEGIN --EMPEZAR
    Sentencia de instrucciones SQL END –FIN
```

Ejemplo 1

Se comprueba si el valor ingresado es número o letra

```
DECLARE @NUM
INT
SET @NUM = 1

IF (IsNumeric(@NUM)=1) --Devuelve un valor de 1 (verdadero) si la
        --expresión es un valor numérico, devuelve un
        --valor de 0 (falso)de otra manera.
BEGIN
SELECT 'Ingreso Numero' as Rpta
END
ELSE
BEGIN
SELECT 'Ingreso    Letra' as Rpta
END
```

	RPTA
1	Ingreso Numero

Ejemplo 2

Muestra un mensaje de acuerdo a la nota

```
DECLARE @NOTA INT
SET @NOTA=18 --Ingresando una Nota

IF(@NOTA<= 10) BEGIN
PRINT 'MALO' END
ELSE IF(@NOTA<=14) BEGIN
PRINT 'REGULAR' END
ELSE IF(@NOTA<=17) BEGIN
PRINT 'BUENO' END
ELSE IF(@NOTA<=20) BEGIN
PRINT 'EXCELENTE' END
ELSE BEGIN
PRINT 'INGRESO NOTA INCORRECTA' END
```

```
EXCELENTE
```

Ejemplo 3

Generar un número aleatorio de 2 dígitos, y si el número es mayor a 50, mostrarlo en forma negativa.

```
DECLARE @num decimal(7,2)
SET @num = RAND()*100

IF @num > 50
	BEGIN
		SET @num = @num * (-1)
		PRINT CONCAT('Valor cambiado = ', @num)
	END
ELSE
	PRINT CONCAT('Valor sin cambio = ', @num)
```

Ejemplo 4

Implementar un Script que permita insertar un nuevo registro en la tabla país, en caso se registre duplicidad en el nombre de un país mostrar un mensaje de "país ya registrado", caso contrario insertar dicho registro y mostrar un mensaje de "País registrado Correctamente

Crear la tabla País

Nombre de columna	Tipo de datos	Permitir val...
idpais	char(10)	☐
nombre	varchar(50)	☐

Ejecutar el siguiente script:

```
declare @idpais char(4)='0011',
@nombre varchar(30)='Argelia'
if exists(select * from pais where nombre=@nombre)
begin
	print 'País ya Registrado'
end
else
begin
	insert into pais
	values (@idpais,@nombre)
	print 'País Registrado Correctamente'
end
```

Tarea

Realizar un programa que usando la función rand, genere 3 números enteros de 1 digito. Luego usando la instrucción print debe imprimirlos ordenados de mayor a menor.

2. CASE

La expresión CASE buscada evalúa un conjunto de expresiones booleanas para determinar el resultado

Sintaxis para CASE

```
CASE <expresion>
   WHEN <valor_expresion> THEN <valor_devuelto>
   WHEN<valor_expresion> THEN <valor_devuelto> ELSE
   <valor_devuelto>
END
```

Ejemplo 1

De acuerdo al operador, muestra un resultado

```
DECLARE @OPERADOR CHAR(1)
DECLARE @NUM1 INT,@NUM2
INT

SET @OPERADOR ='*' --Ingresando el operador
SET @NUM1=15 --Ingresando el primer numero
SET @NUM2= 8--Ingresando el segundo numero


   SELECT CASE
      WHEN @OPERADOR ='+' THEN @NUM1 + @NUM2
      WHEN @OPERADOR ='-' THEN @NUM1 - @NUM2
      WHEN @OPERADOR ='*' THEN @NUM1 * @NUM2
      WHEN @OPERADOR ='/' THEN @NUM1 / @NUM2 ELSE 0
       --Nos devuelve cero siempre cuando
       --no ingresamos el Operador Correcto
   END
   AS CALCULO
```

	CALCULO
1	120

Ejemplo 2

Implementar un script que permita mostrar la fecha en texto registrada en la tabla Reserva

Crear la tabla Reserva

Nombre de columna	Tipo de datos	Permitir val...
idreserva	int	☐
costo	money	☐
fecha	date	☐
observacion	varchar(200)	☑

Insertar los siguientes registros:

idreserva	costo	fecha	observacion
1	90.0000	2013-01-27	*NULL*
2	50.0000	2013-01-01	*NULL*
3	300.0000	2014-03-04	*NULL*
4	800.0000	2014-04-05	*NULL*
5	250.0000	2014-06-17	*NULL*
6	1650.0000	2016-09-18	*NULL*
7	700.0000	2015-10-20	*NULL*

Ejecutar el siguiente script:

```
select *, cast(day(fecha) as char(2)) +
case month(fecha)
when 1 then 'enero'
when 2 then 'febrero'
when 3 then 'marzo'
when 4 then 'abril'
when 5 then 'mayo'
when 6 then 'junio'
when 7 then 'julio'
when 8 then 'agosto'
when 9 then 'setiembre'
when 10 then 'octubre'
when 11 then 'noviembre'
when 12 then 'diciembre'
end
+ cast(year(fecha)as char(4)) as [Fecha en Texto]
from reserva
go
```

3. WHILE

Establece una condición para la ejecución repetida de una instrucción o bloque de instrucciones SQL. Las instrucciones se ejecutan repetidamente siempre que la condición especificada sea verdadera. Se puede controlar la ejecución de instrucciones en el bucle WHILE con las palabras clave BREAK y CONTINUE.

```
WHILE
Boolean_expression
BEGIN
    { sql_statement |
    statement_block } [ BREAK
    ]
    { sql_statement |
    statement_block } [
    CONTINUE ]
    { sql_statement |
statement_block } END
```

Argumentos

Boolean_expression

Es una expresión que devuelve TRUE o FALSE. Si la expresión booleana contiene una instrucción SELECT, la instrucción SELECT debe ir entre paréntesis.

{sql_statement | statement_block}

Se trata de cualquier instrucción o grupo de instrucciones Transact-SQL definidas con un bloque de instrucciones. Para definir un bloque de instrucciones, utilice las palabras clave de control de flujo BEGIN y END.

BREAK

Produce la salida del bucle WHILE más interno. Se ejecutan las instrucciones que aparecen después de la palabra clave END, que marca el final del bucle.

CONTINUE

Hace que se reinicie el bucle WHILE y omite las instrucciones que haya después de la palabra clave CONTINUE

Ejemplo 1

Muestra los pares del 2 al 10 y su suma

```
DECLARE @PARES int,@SUMAPARES int SET
@PARES = 0
SET @SUMAPARES = 0
WHILE (@PARES < 10)
BEGIN
SET @PARES = @PARES + 1
            IF (@PARES % 2 = 0)
       BEGIN
          PRINT 'Numeros Pares: '+ str(@PARES)
       SET @SUMAPARES=@SUMAPARES+@PARES


     END
END
PRINT '_____'

PRINT 'Suma de Pares: '+ str(@SUMAPARES)
```

```
Numeros Pares:          2
Numeros Pares:          4
Numeros Pares:          6
Numeros Pares:          8
Numeros Pares:         10
    _
Suma de Pares:         30
```

Ejemplo 2

Muestra la suma de la cifras de un numero

```
DECLARE @N int,@SUMA int,@RESTO int


SET @SUMA = 0
SET @N = 45623 --Ingresando un Numero Positivo


WHILE (@N <> 0)
 BEGIN
   SET @RESTO = @N %10
   SET @SUMA = @SUMA + @RESTO
   SET @N      = @N / 10
END

SELECT 'La Suma de Cifras: '+ str(@SUMA) as solucion
```

	solucion
1	La Suma de Cifras: 20

Ejemplo 3

Muestra los números del 1 al 4 con la instrucción while y break

```
DECLARE @empid as int=1
while @empid <= 5
begin
print @empid
set @empid+=1
if @empid=5
break
end
```

```
1
2
3
4
```

Como vemos solo nos deja ver 4 números ya que al sumar ya 5, le decimos que se salga del bucle.

Ejemplo 4
Muestra los números del 1 al 5 con while y continue

```
DECLARE @empid as int=1
while @empid <= 5
begin
print @empid
set @empid+=1
if @empid=5
continue
end
```

Como aquí vemos el continue pasa de nosotros y nos deja seguir ejecutando el bucle.

```
1
2
3
4
5
```

III. REQUERIMIENTOS

- SQL SERVER
- GUIA DE LABORATORIO Nº 3

IV. EJERCICIO COMPLEMENTARIO

1. Escribir un programa que calcule el factorial de N
2. Escribir un programa que usando la función RAND genere un numero entero de 2 cifras y verifique si el número es primo
3. Escribir un programa que usando la función RAND genere 10 números de 2 dígitos, que identifique al menor del grupo y que lo imprima

UNIVERSIDAD FEDERICO VILLAREAL

FACULTAD DE INGENIERIA INDUSTRIAL Y SISTEMAS

ESCUELA DE INGENIERIA DE SISTEMAS

GUIA DE LABORATORIO Nº 4

NOMBRE DE LA PRÁCTICA : CREACIÓN DE BASE DE DATOS RELACIONAL CON TRANSACT – SQL

CURSO : SISTEMA DE BASE DATOS

DOCENTE : ING. MAIKE JAUREGUI

CICLO : 2018-II

I. Objetivos

1. Crear una base de datos con Lenguaje SQL
2. Crear tablas y definir tipos de datos con lenguaje SQL
3. Implementar la integridad referencial en una base de datos

II. Introducción Teórica

SQL (Structured Query Language), Lenguaje Estructurado de Consulta es el lenguaje utilizado para definir, controlar y acceder a los datos almacenados en una base de datos relacional.

1. Instrucción CREATE

La primera parte de CREATE será siempre igual:

CREATE <tipo de objeto> <nombre del objeto>

A esta parte le seguirán los detalles, que variarán según la naturaleza del objeto que estemos creando. A continuación se presenta un listado de sintaxis más completa de CREATE:

1.1 CREATE DATABASE

Crea una nueva base de datos.

Sintaxis:

```
CREATE DATABASE nombre_base_de_datos
```

1.2 CREATE TABLE

Crea una nueva tabla en SQL Server

Sintaxis:

```
CREATE TABLE nombre_tabla
(
nombre_campo_1 tipo_1
nombre_campo_2 tipo_2
nombre_campo_n tipo_n
Key(campo_x,...)
)
```

2. Restricciones en una tabla

Limitar el tipo de dato que puede ingresarse en una tabla. Dichas restricciones pueden especificarse cuando la tabla se crea por primera vez a través de la instrucción CREATE TABLE, o luego de crear la tabla a través de la instrucción ALTER TABLE.

Los tipos comunes de restricciones incluyen las siguientes:

- NOT NULL
- UNIQUE
- CHECK
- DEFAULT
- PRIMARY KEY
- FOREIGN KEY

2.1 NOT NULL

De forma predeterminada, una columna puede ser NULL. Si no desea permitir un valor NULL en una columna, deberá colocar una restricción NOT NULL en esta columna especificando que en esa columna es obligatorio introducir un dato.

Por ejemplo, en la siguiente instrucción,

```
CREATE TABLE Cliente
(Codigo integer NOT NULL,
Apellido varchar (30) NOT NULL,
Nombre varchar (30));
```

Con la instrucción NOT NULL en las columnas “Codigo” y “Apellido”, estas no aceptan valores nulos (vacíos), mientras que el campo “Nombre” si puede contener valores nulos.

2.2 UNIQUE

La restricción UNIQUE asegura que todos los valores en una columna sean únicos.

Por ejemplo, en la siguiente instrucción,

```
CREATE TABLE Cliente
(Codigo integer UNIQUE,
Apellido varchar (30),
Nombre varchar(30));
```

La columna “Codigo” no puede incluir valores duplicados, dicha restricción no se aplica para columnas “Apellido” y “Nombre”

Nota: esta restricción no sustituye la clave primaria

2.3 CHECK

La restricción CHECK asegura que todos los valores en una columna cumplan ciertas condiciones.

Por ejemplo, en la siguiente instrucción:

```
CREATE TABLE Customer
(Codigo integer CHECK (Codigo > 0),
Apellido varchar (30),
Nombre varchar(30))
```

La columna “Codigo” sólo acepta datos enteros mayores que cero.

2.4 DEFAULT

La restricción DEFAULT establece un valor por defecto a un campo de la tabla. Un valor por defecto se inserta cuando no está presente al ingresar un registro

Por ejemplo, en la siguiente instrucción:

```
CREATE TABLE libros(
  codigo int not null,
  titulo varchar(40),
  autor varchar(30) not null DEFAULT 'Desconocido',
  editorial varchar(20),
  precio decimal(5,2),
  cantidad tinyint DEFAULT 0
 )
```

Queremos que el valor por defecto del campo "autor" de la tabla "libros" sea "Desconocido" y el valor por defecto del campo "cantidad" sea "0":

2.5 PRIMARY KEY

La clave primaria, PRIMARY KEY, identifica de manera única cada fila de una tabla

Por ejemplo, en la siguiente instrucción:

```
CREATE TABLE usuarios(
  nombre varchar(20),
  clave varchar(10),
  primary key(nombre)
 )
```

Al campo "nombre" no lo definimos "not null", pero al establecerse como clave primaria, SQL Server lo convierte en "not null".

La clave primaria (PRIMARY KEY) puede estar compuesta por varias columnas, por ejemplo por las columnas 'identificador' y 'nombre', entonces se define así:

```
CREATE TABLE personas (
identificador int NOT NULL,
nombre varchar(255) NOT NULL,
apellido1 varchar(255) NOT NULL,
CONSTRAINT pers PRIMARY KEY (identificador, nombre) )
```

2.6 FOREIGN KEY

La clave externa o **FOREIGN KEY**, es una columna o varias columnas, que sirven para señalar cual es la clave primaria de otra tabla.

La columna o columnas señaladas como **FOREIGN KEY**, solo podrán tener valores que ya existan en la clave primaria PRIMARY KEY de la otra tabla.

Ejemplo de **FOREIGN KEY**

Tabla "departamentos", con la clave primaria "dep"

Dep	Departamento
1	ADMINISTRACION
2	INFORMATICA
3	COMERCIAL

Tabla personas, con una clave externa **FOREIGN KEY** 'dep', que hace referencia a la clave primaria 'dep' de la tabla anterior 'departamentos' y por tanto, solo puede tener un valor de los que tiene en la tabla

per	nombre	apellido1	apellido2	dep
1	ANTONIO	PEREZ	GOMEZ	1

2	ANTONIO	GARCIA	RODRIGUEZ	2
3	PEDRO	RUIZ	GONZALEZ	4

```
CREATE TABLE departamentos
{
dep int NOT NULL PRIMARY KEY,
departamento varchar(255),
}

CREATE TABLE personas
{
per int NOT NULL PRIMARY KEY,
nombre varchar(255),
apellido1 varchar(255),
dep int FOREIGN KEY REFERENCES departamentos (dep)
}
```

Si la clave externa o foránea (**FOREIGN KEY**) está compuesta por varias columnas o queremos ponerle un nombre, utilizaremos la fórmula siguiente:

```
CONSTRAINT fkpersonas FOREIGN KEY (dep, id) REFERENCES departamentos(dep,id).
```

Si se tuviera varias llaves foráneas, utilizaremos lo siguiente:

```
create table tabla_origen (
  idnumber int  primary key,
 id01 varchar(5),
 id02 varchar(5),
 id03 varchar(5),
 constraint fk_id01 foreign key (id01) references tabla_destino1 (clave),
 constraint fk_id02 foreign key (id02) references tabla_destino2 (clave),
 constraint fk_id03 foreign key (id03) references tabla_destino3 (clave)
  )
```

3. **ALTER TABLE**

Modifica una definición de tabla al alterar, agregar o quitar columnas y restricciones, reasignar particiones, o deshabilitar o habilitar restricciones y desencadenadores.
Algunas instrucciones pueden ser:

3.1 Agregar un nuevo campo

Para agregar un nuevo campo a la tabla digitamos la siguiente sentencia:

```
ALTER TABLE CONTACTOS
ADD estado varchar(8)
```

Esta sentencia nos permite agregar el campo **estado** con un tipo de dato VARCHAR de 8 caracteres a la tabla contactos

3.2 Eliminar un campo

Para eliminar un campo de la tabla digitamos la siguiente sentencia:

```
ALTER TABLE CONTACTOS
DROP COLUMN estado
```

4. **AGREGAR UNA RESTRICCIÓN**

Con la instrucción ALTER TABLE se pueden agregar las diferentes tipos de restricciones mencionadas anteriormente, por ejemplo:

4.1 Restricción Check

```
ALTER TABLE CONTACTOS
ADD CONSTRAINT exd_check CHECK (codigo> 1)
```

4.2 Restricción Default

```
ALTER TABLE VENTA
ADD CONSTRAINT default_fecha
DEFAULT getdate() FOR fechaventa
```

Nota: getdate= fecha actual del sistema

4.3 Restricción Llave Primaria (PRIMARY KEY)

```
ALTER TABLE CONTACTOS
ADD CONSTRAINT primary_codigo PRIMARY KEY (Codigo);
```

4.4 Restricción Llave Foránea (FOREIGN KEY)

```
ALTER TABLE VENTA
ADD CONSTRAINT fK_venta FOREIGN KEY (IdVenta)
REFERENCES CONTACTOS (Codigo)
```

5. Tipos de datos en SQL SERVER

<table>
<tr><th>Tipo de dato</th><th>Intervalo</th><th>Almacenamiento</th></tr>
<tr><td>Bigint</td><td>De -2^63 (-9.223.372.036.854.775.808) a 2^63-1 (9.223.372.036.854.775.807)</td><td>8 bytes</td></tr>
<tr><td>Bit</td><td>Tipo de datos entero que puede aceptar los valores 1, 0 o NULL.
SQL Server Database Engine (Motor de base de datos de SQL Server) optimiza el almacenamiento de las columnas de tipo bit.
Si una tabla contiene 8 columnas o menos de tipo bit, éstas se almacenan como 1 byte. Si hay entre 9 y 16 columnas de tipo bit, se almacenan como 2 bytes, y así sucesivamente.
Los valores de cadena TRUE y FALSE se pueden convertir en valores de tipo bit: TRUE se convierte en 1 y FALSE en 0.</td><td></td></tr>
<tr><td>Decimal</td><td>decimal [(p, [(s)])] Cuando se utiliza la precisión máxima, los valores válidos se sitúan entre - 10^38 +1 y 10^38 - 1.

p (precisión)
El número total máximo de dígitos decimales que se puede almacenar, tanto a la izquierda como a la derecha del separador decimal. La precisión debe ser un valor comprendido entre 1 y la precisión máxima de 38. La precisión predeterminada es 18.
s (escala)
El número máximo de dígitos decimales que se puede almacenar a la derecha del separador decimal. La escala debe ser un valor comprendido entre 0 y p. Sólo es posible especificar la escala si se ha especificado la precisión. La escala predeterminada es 0; por lo tanto, 0 <= s <= p. Los tamaños de almacenamiento máximo varían, según la precisión.
<table>
<tr><th>Precisión</th><th>Bytes de almacenamiento</th></tr>
<tr><td>1 - 9</td><td>5</td></tr>
<tr><td>10-19</td><td>9</td></tr>
<tr><td>20-28</td><td>13</td></tr>
<tr><td>29-38</td><td>17</td></tr>
</table></td><td></td></tr>
<tr><td>Int</td><td>De -2^31 (-2.147.483.648) a 2^31-1 (2.147.483.647)</td><td>4 bytes</td></tr>
<tr><td>Money</td><td>De -922,337,203,685.477,5808 a 922,337,203,685.477,5807</td><td>8 bytes</td></tr>
<tr><td>Numeric</td><td>Se emplea al igual que el tipo decimal. Numeric[(p, s)]</td><td></td></tr>
<tr><td>Smallint</td><td>De -2^15 (-32.768) a 2^15-1 (32.767)</td><td>2 bytes</td></tr>
<tr><td>Smallmoney</td><td>De - 214.748,3648 a 214.748,3647</td><td>4 bytes</td></tr>
<tr><td>Tinyint</td><td>De 0 a 255</td><td>1 byte</td></tr>
</table>

Tabla 5.1 Numéricos exactos.

<table>
<tr><th>Tipo de dato</th><th>Intervalo</th><th>Almacenamiento</th></tr>
<tr><td>Float</td><td>De – 1,79E+308 a -2,23E-308, 0 y de 2,23E-308 a 1,79E+308

float [(n)]
Donde n es el número de bits que se utilizan para almacenar la mantisa del número float en notación científica y, por tanto, dicta su precisión y el tamaño de almacenamiento. Si se especifica n, debe ser un valor entre 1 y 53. El valor predeterminado de n es 53.
<table>
<tr><th>Valor de n</th><th>Precisión</th><th>Tamaño de almacenamiento</th></tr>
<tr><td>1-24</td><td>7 dígitos</td><td>4 bytes</td></tr>
<tr><td>25-53</td><td>15 dígitos</td><td>8 bytes</td></tr>
</table></td><td>Depende de n.</td></tr>
<tr><td>Real</td><td>De – 3,40E + 38 a -1,18E – 38, 0 y de 1,18E – 38 a 3,40E + 38</td><td>4 Bytes</td></tr>
</table>

Tabla 5.2 Numéricos aproximados.

Tipo de dato	Salida
Date	2007-05-08
Datetime	2007-05-08 12:35:29.123
Datetime2	2007-05-08 12:35:29. 1234567
Datetimeoffset	2007-05-08 12:35:29.1234567 +12:15
Smalldatetime	2007-05-08 12:35:00
Time	12:35:29. 1234567

Tabla 5.3 Fecha y hora.

Tipo de dato	Intervalo
Char	**char** [(*n*)] Datos de caracteres no Unicode de longitud fija, con una longitud de *n* bytes. *n* debe ser un valor entre 1 y 8.000. El tamaño de almacenamiento es *n* bytes.
Text	**text** Datos no Unicode de longitud variable de la página de códigos del servidor y con una longitud máxima de 2^31-1 (2.147.483.647) caracteres. Cuando la página de códigos del servidor utiliza caracteres de doble byte, el almacenamiento sigue siendo de 2.147.483.647 bytes. Dependiendo de la cadena de caracteres, el espacio de almacenamiento puede ser inferior a 2.147.483.647 bytes.
Varchar	**varchar** [(*n* | **Max**)] Datos de caracteres no Unicode de longitud variable. *n* puede ser un valor entre 1 y 8.000. **Max** indica que el tamaño de almacenamiento máximo es de 2^31-1 bytes. El tamaño de almacenamiento es la longitud real de los datos especificados + 2 bytes. Los datos especificados pueden tener una longitud de 0 caracteres.

Tabla 5.4 Cadenas de caracteres.

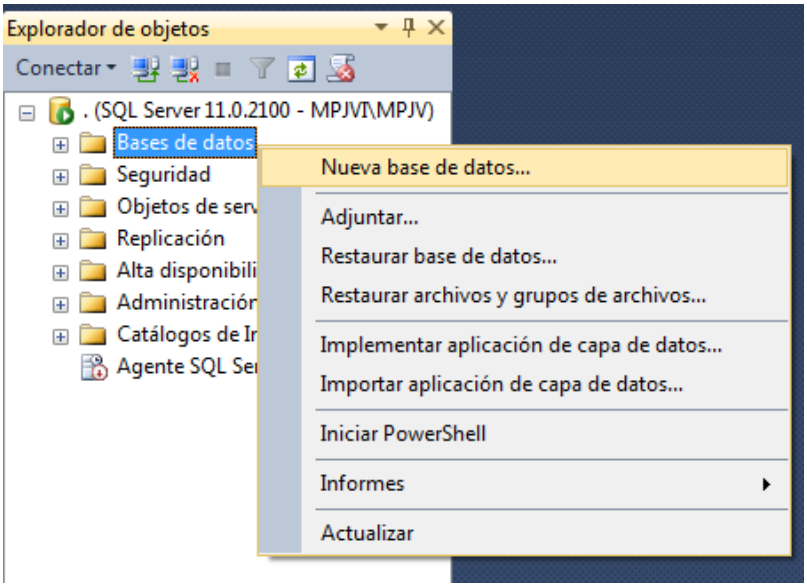
III. Requerimientos

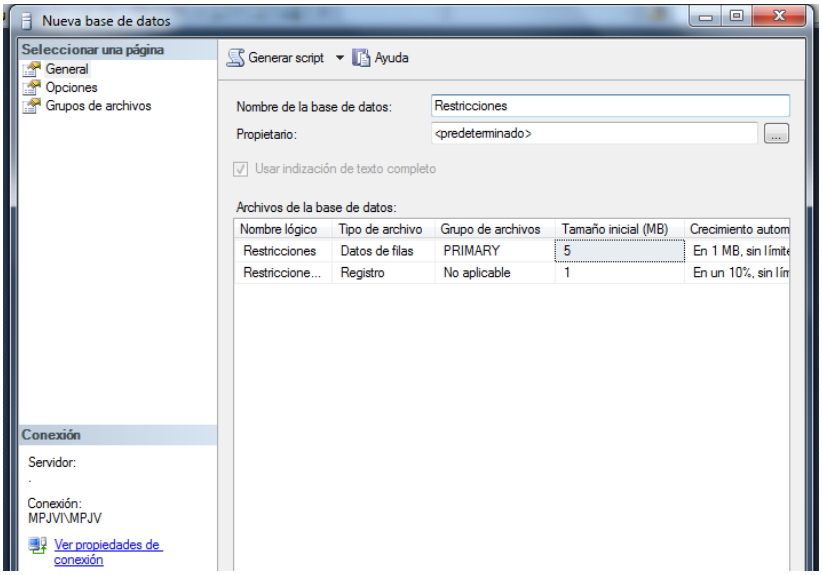
- Máquina con SQL Server
- Guía Número 4 de base de datos

IV. Procedimiento

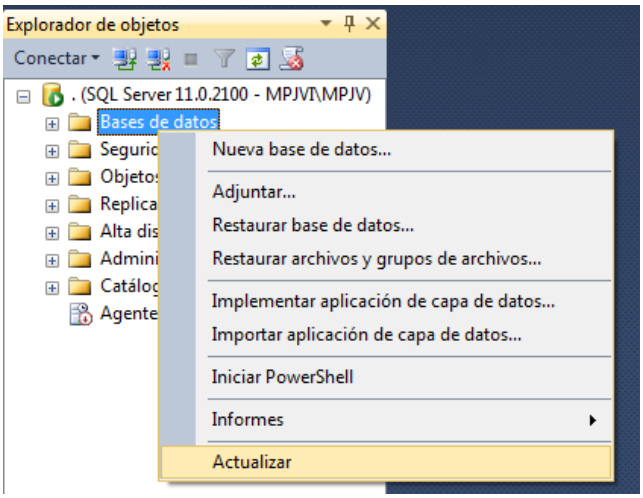
Ejercicio 1. Crear la base de datos

1. Ir al programa SQL SERVER. Crear la base de datos con el nombre de Restricciones





2. Ir a la carpeta Base de Datos. Seleccionar la opción actualizar



Ejercicio 2. Crear las tablas de la base de datos

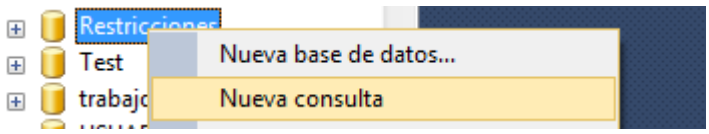
Las tablas a crear en este ejercicio son:

Tablas	**Campos**
Distritos	Iddistrito Nombre_dist
Pagos	Idpago Descripción_pago
Proveedor	idproveedor nombre_prov iddistrito idpago
Productos	idproducto nombre_prod precio cantidad idproveedor

Nota:

- A la tabla Proveedor se agregó el campo iddistrito por la relación que existe entre esta tabla y la tabla Distritos
- A la tabla Productos se agregó el campo idproveedor por la relación que existe entre esta tabla y la tabla Proveedor

1. Seleccionar la base de datos **Restricciones**. Clic derecho. Nueva consulta



2. Crear la tabla **Distritos**

```
--creando las tablas
--tabla Distritos

create table distritos(
iddistrito int primary key,
nombre_dist varchar(100)
)
go
```

3. Dar clic en Ejecutar

4. Crear la tabla **Pagos**

```
--creando las tablas
--tabla Pagos
create table pagos(
idpago int primary key,
descripcion_pago  varchar(200)
)
go
```

Seleccionar desde el comando CREATE hasta la instrucción GO, hacer clic en la opción Ejecutar

Ejercicio 3. Creando restricciones

Relaciones a crear:

Claves primarias	**Claves foráneas**
Tabla Distritos iddistrito	**Tabla Proveedor** iddistrito Hace referencia a la clave primaria de la tabla Distritos
Tabla Pagos idpago	Idpago Hace referencia a la clave primaria de la tabla Pagos

1. En el editor de consultas digitar la consulta la cual va a crear la relación entre la tabla proveedor y distritos, proveedor y pagos

2. Crear la tabla **Proveedor**

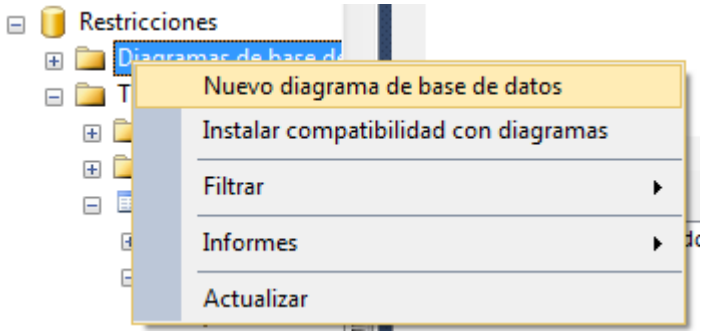
```
    --creación de llaves foráneas
    --relacion entre proveedor y distritos
    --relacion entre proveedor y pagos

 create table proveedor(
 idproveedor int primary key,
 nombre_prov varchar(100),
 iddistrito int foreign key references distritos(iddistrito),
 idpago int foreign key references pagos(idpago)
          )
go
```

3. Crear la tabla **Productos**

```
    --creación de llaves foráneas
    --relacion entre productos y proveedor

create table productos(
 idproducto int primary key,
 nombre_prod varchar(100),
 precio money,
 cantidad int,
 idproveedor int foreign key references proveedor(idproveedor)
 )
 go
```

Ejercicio 3. Creando el diagrama de base de datos

1. Seleccionar la base de datos **Restricciones**. Ubicarse dentro de la **carpeta Diagrama de Base de Datos. Nuevo diagrama de base de datos.**

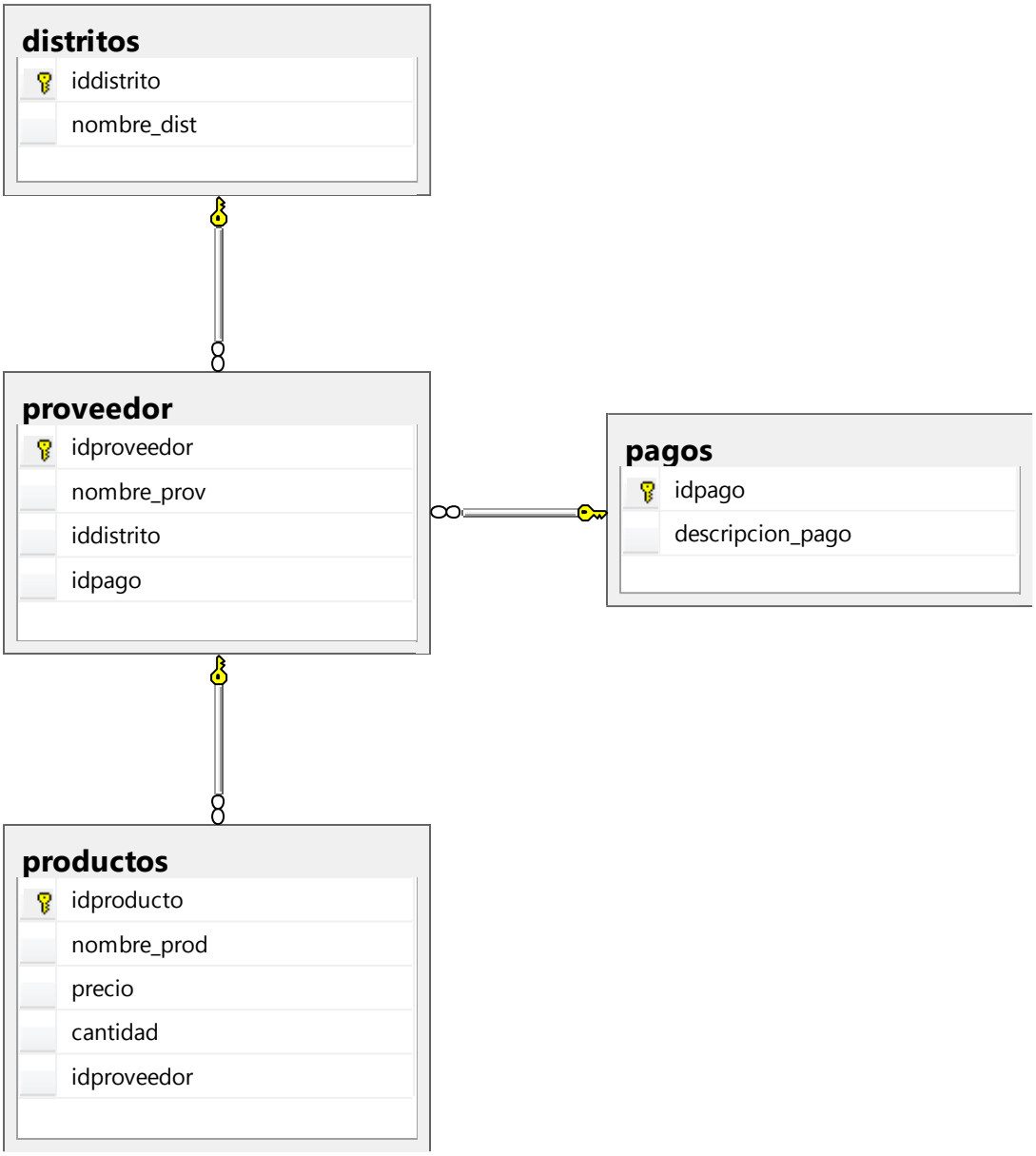


Nota: Si no se crea el diagrama de base de datos, ejecutar la siguiente consulta:

```
ALTER AUTHORIZATION ON DATABASE::Restricciones TO sa;
```

```
GO
```

2. Visualizar el diagrama de base de datos



UNIVERSIDAD FEDERICO VILLAREAL

FACULTAD DE INGENIERIA INDUSTRIAL Y SISTEMAS

ESCUELA DE INGENIERIA DE SISTEMAS

GUIA DE LABORATORIO Nº 5

NOMBRE DE LA PRÁCTICA : CREACIÓN DE BASE DE DATOS RELACIONAL CON TRANSACT – SQL

CURSO : SISTEMA DE BASE DATOS

DOCENTE : ING. MAIKE JAUREGUI

CICLO : 2018-II

I. Objetivos

1. Crear una base de datos con Lenguaje SQL
2. Crear tablas y definir tipos de datos con lenguaje SQL
3. Implementar la integridad referencial en una base de datos

II. Requerimientos

- Máquina con SQL Server
- Guía Número 5 de base de datos

III. Procedimiento

Ejercicio 1: Crear la base de datos y las tablas

1. **Crear la base de datos Biblio**

```
--creando la base de datos
--colocando un comentario
use master--hacer uso de la base de datos master
go--comando que indica el final de un lote de instrucciones transact-sql
create database biblio
go
use biblio--hacer uso de la base de datos biblio
go
```

Ejercicio 2. Crear las tablas de la base de datos

Tablas	Campos
Autor	CodigoAutor Nombre (PrimerNombre y Primer Apellido) FechaNacimiento Nacionalidad
Libro	CodigoLibro

	Titulo ISBN AñoEdicion CodigoEditorial
Editorial	CodigoEditorial Nombre País
Detalle_AutorLibro	CodigoAutor CodigoLibro Fecha

Nota:

- A la tabla Libro se agregó el campo CodigoEditorial por la relación que existe entre esta tabla y la tabla Editorial

- Se creó la tabla Detalle_AutorLibro ya que esta tabla intermedia rompe la relación de muchos a muchos que existe entre Autor y Libro, se le agregaron los campos CodigoAutor y CodigoLibro para crear la relación de uno a muchos.

1. Crear la **tabla Autor**, digitar después de la última instrucción GO, la siguiente consulta

```
--creando las tablas
--tabla autor
create table autor
(CodigoAutor char(5) not null,
PrimerNombre varchar(25),
SegundoNombre varchar(30),
FechaNacimiento date,
Nacionalidad varchar(35),
)
go
```

2. Ahora se tiene que crear la tabla **Libro**, digite la siguiente consulta después de la última instrucción GO:

```
--tabla libro
create table libro
(CodigoLibro char(10) not null,
Titulo varchar(max),
ISBN varchar(20) not null,
AnoEdicion char(4) not null,
CodigoEditorial char(5),
)
go
```

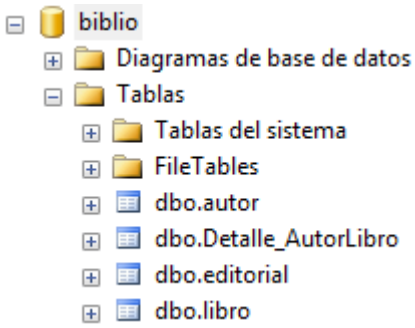
3. Crear la tabla **Editorial,** digite la siguiente consulta después de la última instrucción GO:

```
--tabla editorial
create table editorial
(CodigoEditorial char(5) not null,
Nombre varchar(45),
Pais varchar(50),
)
go
```

4. Ejecutar la consulta y actualice la base de datos
5. Crear la tabla **Detalle_AutorLibro**, después de la consulta anterior, digitar la siguiente:

```
create table Detalle_AutorLibro
(CodigoAutor char(5) not null,
CodigoLibto char(10) not null,
Fecha date
)
```

6. Ejecutar la consulta y actualice la carpeta Tables, al final deberá tener en la base de datos Biblio, las cuatro tablas creadas



Ejercicio 3. Creando restricciones con la instrucción ALTER TABLE

Ahora se creará las relaciones entre las tablas

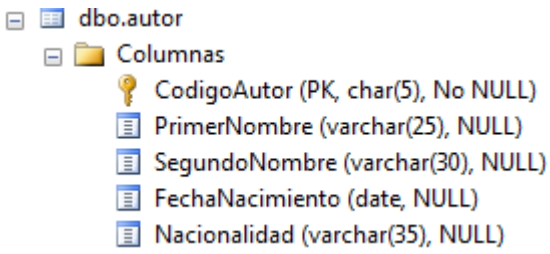
Estableciendo las claves principales o primarias:

1. Después de la última consulta SQL de la parte de creación de tablas, digite la siguiente consulta:

```
--creando las restricciones
--creando las llaves primarias
--tabla autor

alter table autor
add constraint pk_CodigoAutor
primary key (CodigoAutor)
go
```

2. Seleccione la consulta desde la instrucción ALTER hasta el comando GO y ejecútela, si no hay ningún error actualice su tabla

3. Expanda la carpeta dbo.Autor y luego la carpeta Columns, observará los campos de la tabla y la asignación de la clave principal en el campo CodigoAutor



4. Asigne la clave principal a las siguientes tablas:

Tabla	Campo
Libro	CodigoLibro
Editorial	CodigoEditorial

5. Digite las siguientes consultas

```
--tabla libro
alter table libro
add constraint pk_CodigoLibro
primary key(CodigoLibro)
go


--tabla editorial
alter table editorial
add constraint pk_CodigoEditorial
primary key (CodigoEditorial)
go
```

6. Ejecute cada consulta y verifique que se han creado las claves principales en las tablas

Estableciendo las claves externas o foráneas

Claves primarias	Claves foráneas
Tabla : Autor **Campo: CodigoAutor**	Tabla:Detalle_AutorLibro Campo: CodigoAutor Hace referencia a la clave primaria CodigoAutor de la tabla Autor
Tabla: Libro **Campo: CodigoLibro**	Tabla:Detalle_AutorLibro Campo: CodigoLibro Hace referencia a la clave primaria CodigoLibro de la tabla Libro

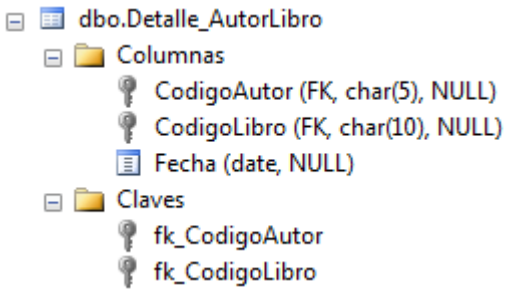
7. En el editor de consultas digite la consulta la cual va a crear la relación entre la tabla **Autor** y **Detalle_AutorLibro**

```
--creacion de llaves foraneas
--relacion entre Autor y Detalle_AutorLibro

alter table Detalle_AutorLibro
add constraint fk_CodigoAutor
foreign key (CodigoAutor)
references Autor(CodigoAutor)
```

8. Seleccione la consulta y ejecútele
9. Digite la siguiente consulta, la cual creará la relación de Libro y Detalle_AutorLibro

```
--relacion entre Libro y Detalle_AutorLibro

alter table Detalle_AutorLibro
add constraint fk_CodigoLibro
foreign key (CodigoLibro)
references Libro(CodigoLibro)
go
```

10. Seleccione la consulta y ejecútele
11. Ahora despliegue las carpetas **Columns** y **Keys** de la tabla **Detalle_AutorLibro** y observará la creación de las llaves foráneas en la tabla y las cuales están se han utilizado para crear la relación entre la tabla:



12. Crear la relación entre las **tablas Editorial y Libros**
13. Digite después de la última consulta:

```
--relacion entre editorial y libro

alter table libro
add constraint fk_LibroEditorial
foreign key (CodigoEditorial)
references Editorial(CodigoEditorial)
go
```

14. Actualice la base de datos y observará los cambios en las tablas Libro y Editorial

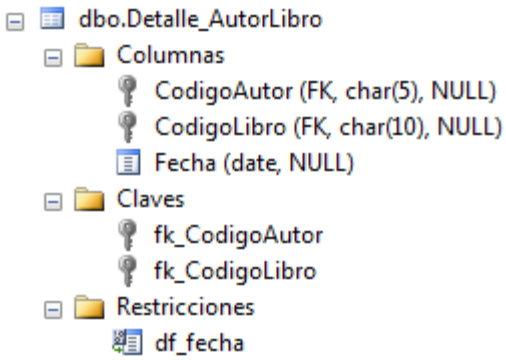
Estableciendo restricciones: Default, Check y Unique

Restricción Default

1. Se creará una restricción Default en el campo Fecha para la tabla Detalle_AutorLibro, en la cual si el usuario no digita nada para esta fecha que se introduzca la fecha del sistema.

```
--restriccion default
alter table Detalle_AutorLibro
add constraint df_fecha
default getdate() for fecha;
go
```

2. Actualice su tabla Detalle_AutorLibro y verifique en la carpeta Constraints la creación de la restricción Default

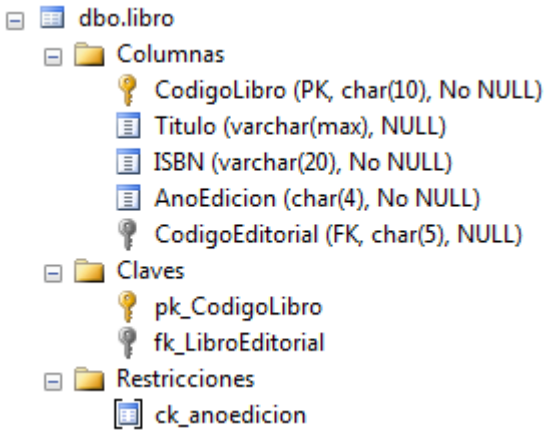


Restricción Check

3. Digite la siguiente consulta:

```
--restriccion check
alter table libro
add constraint ck_anoedicion
check (anoedicion > 2010);
go
```

4. La cual agrega una restricción Check , para el campo AñoEdicion de la tabla Libro, los datos que se introduzcan para este campo deberán ser mayores del 2010
5. Ejecute la consulta y verifique los cambios que le hizo a la columna



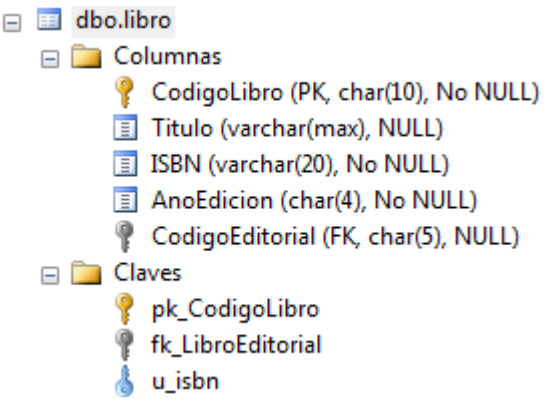
Restricción Unique

6. La consulta que se digitará a continuación, crea una restricción Unique para el campo ISBN de la tabla Libro, el cual se puede tomar ese campo como dato único, pero no es una clave principal ya que ese campo no se utiliza para crear relaciones entre tablas

7. Digite la siguiente consulta:

```
--restriccion unique
alter table libro
add constraint u_isbn
unique(isbn)
go
```

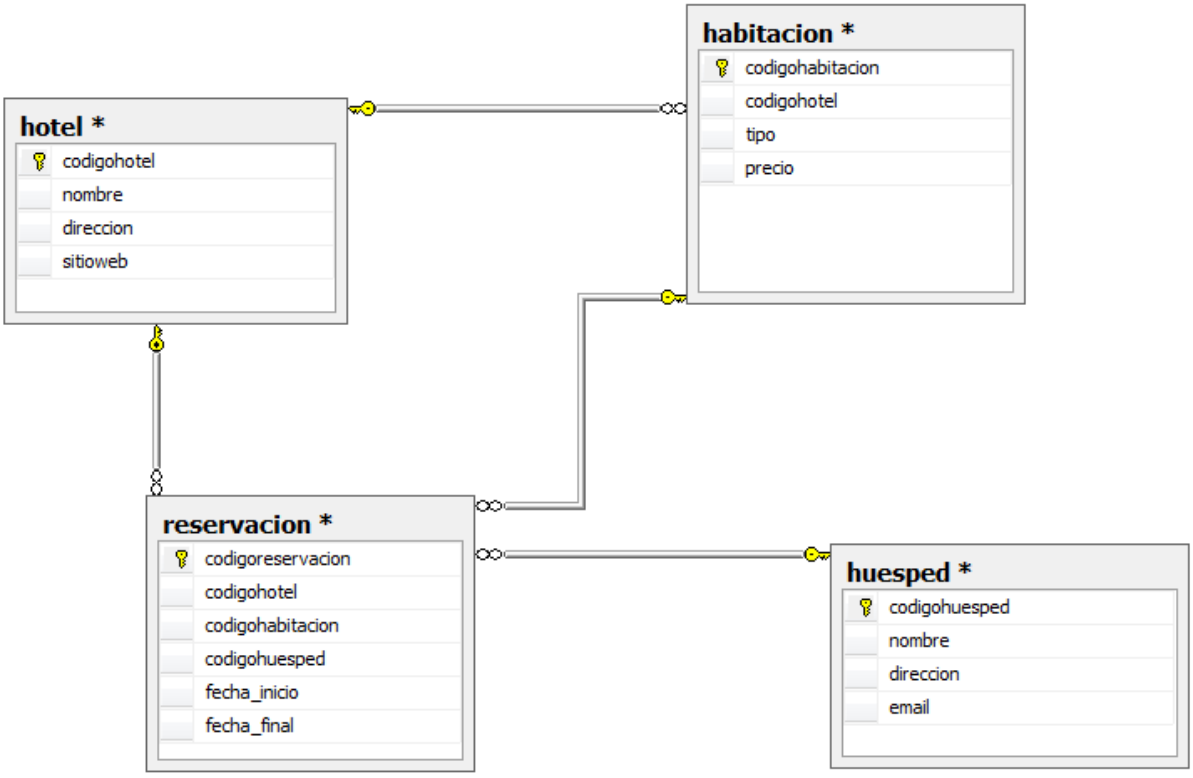
8. Actualice la tabla Libro y vea que agrego la restricción Unique



9. Hasta aquí ha creado las relaciones entre las tablas y las restricciones asegurando la calidad de los datos

IV. Ejercicio Complementario

Nombre de la base de datos: **Hotel**



Crear:

1. La base de datos
2. Las tablas con sus correspondientes campos y propiedades
3. Las relaciones entre las tablas
4. Crear las siguientes restricciones en los campos:

a. **Unique:**

i. Tabla Hotel (Nombre, SitioWeb)

ii. Tabla Huesped (E-mail)

b. **Check:**

i. Tabla Habitacion (Precio mayor que 25, Tipo: Doble, individual)

ii. Tabla Reservacion (Fecha final tiene que ser mayor a la fecha de inicio)

c. **Default:**

i. Tabla Reservacion (Fecha inicio por defecto puede ser la fecha actual del sistema)

5. Crear el diagrama de la base de datos

V. Ejercicio Complementario

Nombre de la base de datos: **Institución**

Ejercicio 1: Crear la base de datos y las tablas

1. **Crear la base de datos Institución**
2. **Crear la siguientes tablas:**
   - **ALUMNOS** (**dni** (12 caracteres), nombre (40 caracteres), apellido1 (40 caracteres), apellido2 (40 caracteres), domicilio (100 caracteres), teléfono (15 caracteres), fecha_nac (fecha/hora), nombre_padre (50 caracteres), nombre_madre (50 caracteres))
   - **ESPECIALIDADES** (**código** (3 caracteres), denominación (30 caracteres))
   - **PROFESORES** (**dni**(12 caracteres), nombre (25 caracteres), apellido1 (25 caracteres), apellido2 (25 caracteres), cod_especialidad (3 caracteres))
   - **MÓDULOS** (**código** (5 caracteres), nombre (40 caracteres), curso (número entero), cod_especialidad (3 caracteres))
   - **GRUPOS** (**código** (6 caracteres), curso (número entero), cod_ciclo (3 caracteres), letra (1 carácter))
   - **CICLOS** (**código**(3 caracteres), denominación (80 caracteres))
   - **MATRÍCULAS** (**dni_alumno(**12 caracteres), **cod_módulo** (5 caracteres))
   - **DOCENCIA** (**cod_grupo** (6 caracteres), **cod_módulo** (5 caracteres), **día** (1 carácter), **hora_inicio** (fecha/hora), hora_fin (fecha/hora), dni_profesor (12 caracteres))

Las llaves foráneas son entre las tablas :

Profesores	Especialidades
Módulos	Especialidades
Grupos	Ciclos
Matriculas	Alumnos
Matriculas	Módulos
Docencia	Grupos
Docencia	Módulos

3. Crear las tablas. Las claves primarias y foráneas deben crearse como restricciones (constraints). Todos los campos que contengan el nombre (o denominación) y los apellidos (de personas, departamentos, áreas, etc.) son obligatorios.
4. Agregar a la tabla **GRUPOS** el campo "observaciones" de 100 caracteres (que sea obligatorio)
5. Agregar a la tabla **MÓDULOS** el campo "horas_semanales", de tipo número decimal con un decimal. Se le asignara por defecto el valor 8. Añadir la restricción de que no se pueda introducir un valor negativo (ni tampoco cero).
6. Añadir la llave foránea **DOCENCIA** (dni_profesor) con **PROFESORES**(dni)
7. Añadir la restricción de que en DOCENCIA el campo “día” sólo pueda tomar los valores 'L', 'M', 'X', 'J' y 'V'.
8. Desactivar la restricción NOT NULL en el campo "denominación" de la tabla ESPECIALIDADES

```
use master
go
create database institucion1
go
use institucion1
go

CREATE TABLE Alumnos
(
dni VARCHAR(12) NOT NULL UNIQUE,
nombre VARCHAR(40) NOT NULL,
apellido1 VARCHAR(40) NOT NULL,
apellido2 VARCHAR(40) NOT NULL,
domicilio VARCHAR(100),
telefono VARCHAR(15),
fecha_nac DATE,
nombre_padre VARCHAR(50),
nombre_madre VARCHAR(50),
CONSTRAINT alumno_pk PRIMARY KEY (dni)
)
go
CREATE TABLE Especialidades
(
codigo VARCHAR(3) NOT NULL UNIQUE,
denominacion VARCHAR(30) NOT NULL,
CONSTRAINT especialidades_pk PRIMARY KEY (codigo)
)
go
CREATE TABLE Profesores
(
dni VARCHAR(12) NOT NULL UNIQUE,
nombre VARCHAR(25) NOT NULL,
apellido1 VARCHAR(25) NOT NULL,
apellido2 VARCHAR(25) NOT NULL,
cod_especialidad VARCHAR(3) NOT NULL,
CONSTRAINT profesores_pk PRIMARY KEY (dni),
CONSTRAINT profesores_fk
FOREIGN KEY (cod_especialidad) REFERENCES Especialidades(codigo)
)
go

CREATE TABLE Modulos
(
codigo VARCHAR(5) NOT NULL UNIQUE,
nombre VARCHAR(40) NOT NULL,
curso INTEGER,
cod_especialidad VARCHAR(3) NOT NULL,
CONSTRAINT modulos_pk PRIMARY KEY (codigo),
CONSTRAINT modulos_fk FOREIGN KEY (cod_especialidad) REFERENCES Especialidades(codigo)
)
go

CREATE TABLE Ciclos
(
codigo VARCHAR(3) NOT NULL UNIQUE,
denominacion VARCHAR(80) NOT NULL,
CONSTRAINT ciclos_pk
PRIMARY KEY (codigo)
)
go
CREATE TABLE Grupos
```

```
(
codigo VARCHAR(6) NOT NULL UNIQUE,
curso INTEGER,
cod_ciclo VARCHAR(3) NOT NULL,
letra VARCHAR(1),
CONSTRAINT grupos_pk PRIMARY KEY (codigo),
CONSTRAINT grupos_fk FOREIGN KEY (cod_ciclo) REFERENCES Ciclos(codigo)
)
go

CREATE TABLE Matriculas
(
dni_alumno VARCHAR(12) NOT NULL,
cod_modulo VARCHAR(5) NOT NULL,
CONSTRAINT matriculas_pk PRIMARY KEY (dni_alumno, cod_modulo),
CONSTRAINT matriculas_fk1 FOREIGN KEY (dni_alumno) REFERENCES Alumnos(dni),
CONSTRAINT matriculas_fk2 FOREIGN KEY (cod_modulo) REFERENCES Modulos(codigo)
)
go

CREATE TABLE Docencia
(
cod_grupo VARCHAR(6) NOT NULL,
cod_modulo VARCHAR(5) NOT NULL,
dia VARCHAR(1) NOT NULL,
hora_inicio TIME NOT NULL,
hora_fin TIME NOT NULL,
dni_profesor VARCHAR(12) NOT NULL,
CONSTRAINT docencia_pk PRIMARY KEY (cod_grupo, cod_modulo, dia),
CONSTRAINT docencia_fk1 FOREIGN KEY (cod_grupo) REFERENCES Grupos(codigo),
CONSTRAINT docencia_fk2 FOREIGN KEY (cod_modulo) REFERENCES Modulos(codigo)
)
go



ALTER TABLE Grupos
ADD observaciones VARCHAR(100) NOT NULL;

ALTER TABLE Modulos
ADD horas_semanales DECIMAL(3,1);

ALTER TABLE Modulos
ADD CONSTRAINT comprobar_horas
CHECK (horas_semanales > 0.0);

ALTER TABLE Modulos
ADD constraint df_2
default  8.0 for horas_semanales

alter table modulos
drop constraint df_1

ALTER TABLE Modulos
ADD CONSTRAINT comprobar_horas
CHECK (horas_semanales > 0.0);


ALTER TABLE Docencia
ADD CONSTRAINT docencia_fk3
FOREIGN KEY (dni_profesor) REFERENCES Profesores(dni);
```

```sql
ALTER TABLE Docencia
ADD CONSTRAINT comprobar_dia
CHECK (dia IN ('L', 'M', 'X', 'J' ,'V') )

ALTER TABLE Especialidades
DROP column denominacion;

ALTER TABLE Especialidades
ADD denominacion VARCHAR(30)



alter table especialidades
drop constraint especialidades_pk

ALTER TABLE Especialidades
ADD CONSTRAINT espec_cl_primaria
PRIMARY KEY (codigo);
```

UNIVERSIDAD FEDERICO VILLAREAL

FACULTAD DE INGENIERIA INDUSTRIAL Y SISTEMAS

ESCUELA DE INGENIERIA DE SISTEMAS

GUIA DE LABORATORIO Nº 6

NOMBRE DE LA PRÁCTICA		**: USO DE CONSULTAS UTILIZANDO LA INSTRUCCIÓN SELECT**
CURSO	**:**	**SISTEMA DE BASE DATOS**
DOCENTE	**:**	**ING. MAIKE JAUREGUI**
CICLO	**:**	**2018-II**

I. Objetivos

1. Mostrar datos almacenados en una base de datos
2. Ejecutar sentencias básicas SELECT
3. Implementar diferentes clausulas con la sentencia SELECT para seleccionar datos tomando ciertas decisiones

II. Introducción Teórica

Lenguaje de manipulación de datos

Lenguaje de cierta complejidad que permite el manejo y procesamiento del contenido de la base de datos.
Sentencias DML:

- Select
- Insert
- Update
- Delete

SELECT. Permite recuperar datos de una o varias tablas. Esta sentencia es de la más compleja y potente de las sentencias SQL.

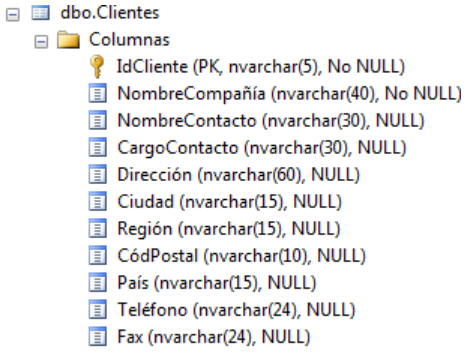
Sintaxis:
SELECT lista_de_campos
[**FROM** nombre_tabla]
[**WHERE** condicion_individual]
[**GROUP BY** campos_a_agrupar]
[**HAVING** condicion_grupo]
[**ORDER BY** campo_a_ordenar [**ASC** | **DESC**]]

FROM

Se utiliza en conjunto con la **cláusula FROM** con la cual se indica en qué tabla o tablas se tiene que buscar la información.

Ejemplo 1

El ejemplo siguiente retorna todas las columnas y filas de la tabla Clientes que se encuentran en la base de datos nw




```
use nw
go
select * from clientes
go
```

Nota: El * significa que se quiere seleccionar todos los campos de la tabla Clientes

Resultado

	IdCliente	NombreCompañía	NombreContacto	CargoContacto	Dirección	Ciudad	Región	CódPostal	País	Teléfono	Fax
1	ALFKI	Alfreds Futterkiste	Maria Anders	Representante de ventas	Obere Str. 57	Berlín	NULL	12209	Alemania	030-0074321	030-0076545
2	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Propietario	Avda. de la Constitución 2222	México D.F.	NULL	05021	México	(5) 555-4729	(5) 555-3745
3	ANTON	Antonio Moreno Taquería	Antonio Moreno	Propietario	Mataderos 2312	México D.F.	NULL	05023	México	(5) 555-3932	NULL
4	AROUT	Around the Horn	Thomas Hardy	Representante de ventas	120 Hanover Sq.	Londres	NULL	WA1 1DP	Reino Unido	(71) 555-7788	(71) 555-6750
5	BERGS	Berglunds snabbköp	Christina Berglund	Administrador de pedidos	Berguvsvägen 8	Luleå	NULL	S-958 22	Suecia	0921-12 34 65	0921-12 34 67
6	BLAUS	Blauer See Delikatessen	Hanna Moos	Representante de ventas	Forsterstr. 57	Mannheim	NULL	68306	Alemania	0621-08460	0621-08924

Ejemplo 2
El ejemplo siguiente retorna las columnas idproducto, nombreproducto, y preciounidad de la tabla Productos

```
select idproducto, nombreproducto, preciounidad
from productos
```

	idproducto	nombreproducto	preciounidad
1	1	Té Dharamsala	18.00
2	2	Cerveza tibetana Barley	19.00
3	3	Sirope de regaliz	10.00
4	4	Especias Cajun del chef Anton	22.00
5	5	Mezcla Gumbo del chef Anton	21.35
6	6	Mermelada de grosellas de la abuela	25.00
7	7	Peras secas orgánicas del tío Bob	30.00
8	8	Salsa de arándanos Northwoods	40.00
9	9	Buey Mishi Kobe	97.00
10	10	Pez espada	31.00
11	11	Queso Cabrales	21.00

Ejemplo 3
El ejemplo siguiente muestra las columnas idpedido, idcliente y fechapedido de la tabla Pedidos. Además se ha agregado encabezados de columna

```
select idpedido as "nro de pedido", idcliente as "codigo", fechapedido
from Pedidos
```

	nro de pedido	codigo	fechapedido
1	10248	WILMK	1994-08-04 00:00:00.000
2	10249	TOMSP	1994-08-05 00:00:00.000
3	10250	HANAR	1994-08-08 00:00:00.000
4	10251	VICTE	1994-08-08 00:00:00.000
5	10252	SUPRD	1994-08-09 00:00:00.000
6	10253	HANAR	1994-08-10 00:00:00.000
7	10254	CHOPS	1994-08-11 00:00:00.000
8	10255	RICSU	1994-08-12 00:00:00.000
9	10256	WELLI	1994-08-15 00:00:00.000
10	10257	HILAA	1994-08-16 00:00:00.000
11	10258	ERNSH	1994-08-17 00:00:00.000

WHERE
La **cláusula WHERE** permite seleccionar únicamente las filas que cumplan con una condición de selección especificada. Sólo se mostrarán las filas para las cuales la evaluación de la condición sea verdadera (TRUE). Los campos con valores NULL no se incluirán en las filas de resultado.
La condición de la **cláusula WHERE** puede ser cualquier condición válida o combinación de condiciones utilizando operadores lógicos (NOT, AND, OR) y relacionales (=, <, >, <=, >=).
Si quisiera unir más condiciones utilizare los operadores lógicos AND u OR, esto será de acuerdo a la información de lo que se desea obtener.

Ejemplo 1

Seleccionar los datos de la tabla Productos en donde el dato almacenado en el campo precio sea mayor a 15

```
SELECT idproducto, nombreproducto, preciounidad
FROM productos
WHERE preciounidad > 15
```

Resultado

	idproducto	nombreproducto	preciounidad
1	1	Té Dharamsala	18.00
2	2	Cerveza tibetana Barley	19.00
3	4	Especias Cajun del chef Anton	22.00
4	5	Mezcla Gumbo del chef Anton	21.35
5	6	Mermelada de grosellas de la abuela	25.00
6	7	Peras secas orgánicas del tío Bob	30.00
7	8	Salsa de arándanos Northwoods	40.00
8	9	Buey Mishi Kobe	97.00
9	10	Pez espada	31.00
10	11	Queso Cabrales	21.00
11	12	Queso Manchego La Pastora	38.00

Ejemplo 2

Seleccionar los datos de la tabla Productos donde el dato almacenado en el campo preciounidad sea mayor o igual a 15 y menor o igual a 50

```
SELECT idproducto, nombreproducto, preciounidad
FROM productos
WHERE preciounidad >=15 and preciounidad <=50
```

Resultado

	idproducto	nombreproducto	preciounidad
1	1	Té Dharamsala	18.00
2	2	Cerveza tibetana Barley	19.00
3	4	Especias Cajun del chef Anton	22.00
4	5	Mezcla Gumbo del chef Anton	21.35
5	6	Mermelada de grosellas de la abuela	25.00
6	7	Peras secas orgánicas del tío Bob	30.00
7	8	Salsa de arándanos Northwoods	40.00
8	10	Pez espada	31.00
9	11	Queso Cabrales	21.00
10	12	Queso Manchego La Pastora	38.00
11	14	Cuajada de judías	23.25

Operador Between

Cuando utilizamos varias condiciones con 2 de un AND utilizaremos la sentencia BETWEEN.

Pero para utilizarlo debe cumplir:

- Debe tener dos condiciones, que uno sea mayor igual (>=) y el otro menor igual (<=)
- Tiene que estar unidas con AND
- Que el campo sea el mismo

Ejemplo 3

Crear la consulta anterior usando la instrucción BETWEEN

```
SELECT idproducto, nombreproducto, preciounidad
FROM productos
WHERE preciounidad between 15 and 50
```

Ejemplo 4

Usando el operador NOT, obtener los registros de la tabla Productos en donde el dato almacenado en el campo preciounidad sea menor a 15

```
SELECT idproducto, nombreproducto, preciounidad
FROM productos
WHERE not preciounidad >= 15
```

Ejemplo 5

Seleccionar los registros de la tabla productos en donde el dato almacenado en el campo idproducto sea mayor a 50 y el dato almacenado en el campo preciounidad sea menor a 10

```
SELECT idproducto, nombreproducto, preciounidad
FROM productos
WHERE idproducto  > 15 or preciounidad < 10
```

	idproducto	nombreproducto	preciounidad
1	13	Algas Konbu	6.00
2	16	Postre de merengue Pavlova	17.45
3	17	Cordero Alice Springs	39.00
4	18	Langostinos tigre Camarvon	62.50
5	19	Pastas de té de chocolate	9.20
6	20	Mermelada de Sir Rodney's	81.00
7	21	Bollos de Sir Rodney's	10.00
8	22	Pan de centeno crujiente estilo Gustaf's	21.00
9	23	Pan fino	9.00
10	24	Refresco Guaraná Fantástica	4.50
11	25	Crema de chocolate y nueces NuNuCa	14.00

Del resultado tomamos el registro 1 y vemos que el dato de ProductID es igual 13 y no cumple la condición, pero el dato almacenado en el campo UnitPrice si es menor que 10 y se cumple la condición, y como se está utilizando el operador OR con uno que sea verdadero es suficiente para que el registro se muestre como un resultado de la consulta SELECT

Ejemplo 6

El siguiente ejemplo retorna las columnas idopedido, idproducto, cantidad y el campo calculado Subtotal de la tabla Detalles de pedidos

```
SELECT idpedido, preciounidad, cantidad, preciounidad * cantidad as subtotal
FROM [dbo].[Detalles de pedidos]
```

Resultado

	idpedido	preciounidad	cantidad	subtotal
1	10248	14.00	12	168.00
2	10248	9.80	10	98.00
3	10248	34.80	5	174.00
4	10249	18.60	9	167.40
5	10249	42.40	40	1696.00
6	10250	7.70	10	77.00
7	10250	42.40	35	1484.00
8	10250	16.80	15	252.00
9	10251	16.80	6	100.80
10	10251	15.60	15	234.00
11	10251	16.80	20	336.00

Operador IN

Cuando tenemos varias condiciones con más de un OR utilizamos la sentencia IN.
IN: Reduce el código, asignado a consultas con condiciones OR. Comando que sirve para búsquedas en varios campos.

Pero para utilizar esta sentencia IN tiene que cumplir:

- Que todas las condiciones estén unidas con el OR.
- Que pertenezcan al mismo campo.

Ejemplo 7

Retornar las filas de la tabla Productos que pertenezcan a las categorías 1, 2, o 3. Usar IN.

```
SELECT * FROM PRODUCTOS
WHERE IdCategoría in(1,3,5)
```

	IdProducto	NombreProducto	IdProveedor	IdCategoría
1	1	Té Dharamsala	1	1
2	2	Cerveza tibetana Barley	1	1
3	16	Postre de merengue Pavlova	7	3
4	19	Pastas de té de chocolate	8	3
5	20	Mermelada de Sir Rodney's	8	3
6	21	Bollos de Sir Rodney's	8	3
7	22	Pan de centeno crujiente estilo Gustaf's	9	5
8	23	Pan fino	9	5
9	24	Refresco Guaraná Fantástica	10	1
10	25	Crema de chocolate y nueces NuNuCa	11	3

El operador like

Se utiliza para búsqueda basada en cadena de caracteres.

Sintaxis

```
select *
from nombre_tabla
where columa LIKE expresión_cadena_a_buscar
```

Los comodines en el operador LIKE

COMODIN	DESCRIPCION
%	Representa cualquier cadena en la posición que sea colocada
_	Representa cualquier carácter en la posición que sea colocada
[abc]	Representa un conjunto de caracteres validos en la posición que sea colocada
[a-b]	Representa un rango de caracteres validos en la posición que sea colocada
^	Excluir. Indica que el carácter, conjunto de caracteres, o rango de caracteres que sigue al símbolo ^ no debe figurar en el resultado de la consulta

Ejemplo 1

Retornar las filas de la tabla Clientes que tengan la letra “F” como carácter inicial en el nombre de la compañía

```
SELECT * FROM Clientes
WHERE NombreCompañía LIKE 'F%'
```

Ejemplo 2

Retornar todas las columnas de la tabla Clientes que tengan los caracteres “ia” al final del de la columna País

```
SELECT * FROM Clientes
WHERE País  LIKE '%ia'
```

Ejemplo 3

Retornar las filas de la tabla Productos que comienzan con la letra “M” y que el segundo carácter excluya la letra “ä”

```
SELECT * FROM Productos
WHERE NombreProducto LIKE 'M[^a]%'
```

Ejemplo 4

Mostrar todos los campos de la tabla Empleados, EXCEPTO aquellos donde el dato almacenado en Apellidos comience con la letra D

```
SELECT * FROM Empleados
WHERE Apellidos NOT  LIKE 'D%'
```

Ejemplo 5

Retornar todos los detalles de pedidos donde el dato almacenado en idpedido termine con los dígitos 0248

```
SELECT * from  [dbo].[Detalles de pedidos]
WHERE IdPedido LIKE '_0248'
```

ORDER BY

Se puede hacer uso de la cláusula ORDER BY para mostrar los datos de forma ordenada.
Si se utiliza en conjunto la cláusula ASC, los registros se mostrarán en orden ascendente (de menor a mayor) según el campo(s) especificado(s) en la cláusula ORDER BY. Si se utiliza la cláusula DESC, los registros serán mostrados en orden descendente (de mayor a menor.

Ejemplo 1
Ordenar de forma ascendente los campos de la tabla Productos por medio del campo idproducto

```
SELECT * FROM PRODUCTOS
ORDER BY IDPRODUCTO ASC
```

Ejemplo 2
Ordenar de forma descendente los campos de la tabla Productos por medio del campo idproducto

```
SELECT  * FROM PRODUCTOS
ORDER BY IDPRODUCTO DESC
```

DISTINCT

La cláusula DISTINCT especifica que los registros con ciertos datos duplicados sean ignorados en el resultado.

Ejemplo 1
Seleccionar todos los registros no repetidos almacenados en el campo idpedido de la tabla Detalles de Pedidos

```
SELECT DISTINCT Idpedido
FROM [dbo].[Detalles de pedidos]
```

TOP N
TOP n, especifica que solo se mostrará el primer conjunto de filas del resultado de la consulta.
El conjunto de filas puede ser un número o un porcentaje de las filas (TOP n PERCENT)
TOP n WITH TIES: Esta cláusula permite incluir en la selección, todos los registros que tengan el mismo valor del campo por el que se ordena.

Para estos ejercicios se ha calculado la venta (unidades vendidas por el precio unitario) de cada orden y se ha renombrado a la columna con el nombre Venta Total

Ejemplo 1
Seleccionar las primeras 3 mejores ventas de la tabla Detalles de Pedidos

```
SELECT TOP 3 idpedido, preciounidad * cantidad as [Venta Total]
FROM [dbo].[Detalles de pedidos]
ORDER BY [Venta Total] DESC
```

Ejemplo 2
Seleccionar las tres mejores ventas, utilizando la instrucción WITH TIES se permitirá incluir en los resultados los registros que tengan el mismo valor en el cálculo de la Venta Total

```
SELECT TOP 3 WITH TIES idpedido, preciounidad * cantidad as [Venta Total]
FROM [dbo].[Detalles de pedidos]
ORDER BY [Venta Total] DESC
```

Ejemplo 3
Seleccionar el 25% del total de registros de las ventas las cuales están almacenadas en la tabla Order Details

```
SELECT TOP 25 PERCENT idpedido, preciounidad * cantidad as [Venta Total]
FROM [dbo].[Detalles de pedidos]
ORDER BY [Venta Total] DESC
```

III. Requerimientos

- Máquina con SQL Server
- Guía Número 6 de base de datos

IV. Ejercicio Complementario

I. Realizar las siguientes consultas

1. Seleccionar todos los campos de la tabla Pedidos ordenados por la fecha de pedido
2. Obtener todos los productos cuyo nombre comienza con la letra P y tienen un precio unitario comprendido entre 10y 120
3. Obtener todos los clientes de los países de Estados Unidos, Francia y España
4. Obtener todos los productos discontinuados (suspendido) y sin stock(unidades en existencia), que pertenezcan a las categorías 2 y 6
5. Obtener todos los pedidos hechos por el empleado con el código 2, 5 y 7 en el año 1996(fechapedido) Usar la función year
6. Seleccionar todos los clientes que cuenten con fax

II. Crear la base de datos Ejercicio

Crear la tabla Empleados:

```
Create Table Empleados
(
Id INT PRIMARY KEY,
Name VARCHAR(100),
Score decimal(8,2)
);
```

Id	Name	Score
1	Shailesh A	98.00
2	Atul K	90.00
3	Vishal P	89.00
4	Naryan N	88.00
5	Rohit G	88.00
6	Varsha K	85.00
7	Sangram K	83.00
8	Vish K	79.00

1. Mostrar los 3 registros que tengan mas alto score
2. Mostrar los 4 registros que tengan más alto score usando TOP n WITH TIES
3. Se muestra el 20% de todos los datos almacenados en la tabla

```sql
select * from pedidos
order by FechaPedido

select * from productos
where nombreproducto like 'P%' and preciounidad between 10 and 120

select * from clientes where País in('Estados Unidos', 'Francia', 'España')

select * from productos where (suspendido=1) and (UnidadesEnExistencia=0) and IdCategoría in
(2,6)

select * from pedidos where idempleado in(2,5,7) and year(fechapedido)=1996

select * from pedidos where idempleado in(2,5,7) and datepart(YYYY, fechapedido)=1996

select * from clientes where fax <> 'isnull'
select * from clientes where fax is not null



create database mjv2
go
use mjv2
go
Create Table SSCResults1
(
Id INT PRIMARY KEY,
Name VARCHAR(100),
Score decimal(8,2)
);


INSERT INTO SSCResults1 VALUES (1, 'Shailesh A',98.0);
INSERT INTO SSCResults1 VALUES (2, 'Atul K',90.0);
INSERT INTO SSCResults1 VALUES (3, 'Vishal P',89.0);
INSERT INTO SSCResults1 VALUES (4, 'Naryan N',88.0);
INSERT INTO SSCResults1 VALUES (5, 'Rohit G',88.0);
INSERT INTO SSCResults1 VALUES (6, 'Varsha K',85.0);
INSERT INTO SSCResults1 VALUES (7, 'Sangram K',83.0);
INSERT INTO SSCResults1 VALUES (8, 'Vish K',79.0);

SELECT TOP 3 *
FROM   SSCResults1
ORDER  BY Score  DESC

SELECT TOP 4 WITh TIES *
FROM   SSCResults1
ORDER  BY Score  DESC


SELECT TOP 20 PERCENT *
from SSCResults1


SELECT TOP 20 PERCENT *
from SSCResults1
ORDER  BY Score  ASC
```

V. Ejercicio Complementario

Tomando la base de datos **AdventureWorks2** crear las siguientes consultas de selección:

a. Seleccionar todos los datos de la tabla **Sales.SalesPerson**

b. Seleccionar todos los registros de los primeros 4 campos de la tabla **Production.Product**

c. Mostrar los 10 productos con el costo (StandardCost) más alto almacenados en la tabla **Production.ProductCostHistory**

d. Seleccionar los campos Name, ProductNumber, ListPrice de la tabla **Production.Product**, debe renombrar cada campo en el siguiente orden: Nombre Producto, Numero de Producto y Precio, los registros se deben ordenar de forma ascendente

e. Seleccionar los primeros tres campos de la tabla **Purchansing.Vendor** donde el dato almacenado en el campo AccountNumber comience con cualquier letra del rango de la G a la T

f. Mostrar los datos de la tabla **Person.CountryRegion** donde el campo CountryRegionCode contenga cualquiera de los siguientes datos: AR, BO, CO, ES, SV y VN

g. Seleccionar los campos CountryRegionCode de la tabla **Person.StateProvince**, pero en el resultado los datos no tienen que repetirse

h. Seleccionar los campos SalesOrderID, OrderQty de la tabla **Sales.SalesOrderDetail** en donde los datos del campo UnitPrice se encuentre entre los valores de 200 y 1000

i. Mostrar los campos ProductID, ListPrice de la tabla **Production.Product** y una columna más que muestre un aumento del 15% del dato almacenado en ListPrice renombrar esta nueva columna con el nombre Aumento de Precio

j. Seleccionar el 20% de los registros almacenados en la tabla **Sales.SalesOrderDetail**

```sql
SELECT * FROM
[Sales].[SalesPerson]


SELECT PRODUCTID, NAME, PRODUCTNUMBER, MAKEFLAG
FROM [Production].[Product]

SELECT TOP 10 *
FROM Production.ProductCostHistory
ORDER BY StandardCost DESC

SELECT Name AS NOMBREPRODUCTO, ProductNumber AS 'NUMERO DE PRODUCTO', ListPrice AS PRECIO
FROM Production.Product
ORDER BY 1 ASC

SELECT BUSINESSENTITYID, ACCOUNTNUMBER, NAME
FROM Purchasing.Vendor
WHERE AccountNumber LIKE '[G-T]%'

SELECT * FROM Person.CountryRegion
WHERE CountryRegionCode LIKE '%AR%' OR CountryRegionCode LIKE '%BO%' OR CountryRegionCode LIKE
'%CO%' OR CountryRegionCode LIKE '%ES%'
OR CountryRegionCode LIKE '%SV%'OR CountryRegionCode LIKE '%VN%'

SELECT  DISTINCT CountryRegionCode
FROM Person.StateProvince

SELECT * FROM Sales.SalesOrderDetail
WHERE UnitPrice BETWEEN 200 AND 1000

SELECT ProductID, ListPrice, ListPrice * 1.15 as  'Aumento de Precio' FROM Production.Product

SELECT SalesOrderID,LineTotal
FROM Sales.SalesOrderDetail
ORDER BY LineTotal DESC
```

Base de datos: **AdventureWorks2012**

a. Seleccionar todos los datos de la tabla **HumanResources.Department**
b. Seleccionar los campos BusinessEntityID, NationalIDNumber y JobTitle de la tabla **HumanResources.Employee** en donde en el campo JobTitle se encuentre la palabra Production
c. Seleccionar los datos de la tabla **Sales.Customer** donde en el campo CustomerID se encuentren los siguientes datos: 2,4, 7 y 10
d. Seleccionar los campos DepartmenID, Name de la tabla **HumanResources.Department** en donde los datos del DepartmenID se encuentre entre los valores 5 y 12
e. Seleccionar los campos AddressID, City y StateProvinceID de la tabla **Person.Address** donde en el campo City el dato comienza con la letra B
f. Seleccionar los datos de la tabla **Production.Culture**, donde el dato almacenado en el campo Name se encuentre entre los valores: English o Spanish
g. Seleccionar el 50% de los datos de la tabla **Sales.CreditCard**
h. Mostrar las 10 mejores ventas (LineTotal) de la tabla **Sales.SalesOrderDetail**
i. Seleccionar el campo JobTitle de la tabla **HumanResources.Employee**, pero no deben mostrarse datos duplicados, ordenar los datos de forma descendente
j. Mostrar los campos Name, ProductNumber y ListPrice y renombrar este campo como Price de la tabla **Production.Product** donde la línea de productos (ProductLine) sea igual a R y el valor correspondiente a los días para fabricar (DaysToMafacture) sea inferior a 4

```sql
SELECT * FROM HumanResources.Department

SELECT BusinessEntityID, NationalIDNumber, JobTitle
FROM HumanResources.Employee
WHERE JobTitle LIKE '%Production%'

SELECT * FROM Sales.Customer
WHERE CustomerID IN (2,4,7,10)

SELECT DepartmentID, Name FROM HumanResources.Department
WHERE DepartmentID BETWEEN 5 AND 12


SELECT AddressID, City, StateProvinceID
FROM Person.Address
WHERE City LIKE 'B%'


SELECT * FROM Production.Culture
WHERE Name  BETWEEN 'ENGLISH' AND 'SPANISH'

SELECT TOP 50* FROM Sales.CreditCard

SELECT TOP 10 *
FROM Sales.SalesOrderDetail
ORDER BY LineTotal DESC

SELECT DISTINCT JobTitle FROM HumanResources.Employee
ORDER BY 1 DESC

SELECT Name, ProductNumber, ListPrice, ProductLine, DaysToManufacture
FROM Production.Product
WHERE ProductLine = 'R' and  DaysToManufacture < 4
```

Base de datos: **Pubs**

a. Seleccionar de la tabla **employee** los datos en donde el campo lname contenga la letra K
b. Seleccionar de la tabla **employee** los datos del campo emp_id que comience con cualquiera de las letras que se encuentren en el rango de la F a la M
c. Mostrar de la tabla **sales** los datos donde la fecha de pedido (ord_date) sean mayores o iguales a 01/01/1994
d. Seleccionar de la tabla **stores** los datos donde el campo state sea igual a CA
e. Seleccionar de la tabla **title** los 5 libros más caros (price) que se encuentran almacenados

```
SELECT * FROM employee
WHERE lname like 'k%'

SELECT * FROM employee
WHERE emp_id LIKE '[F-M]%'

SELECT * FROM SALES
WHERE ord_date >='01/01/1994'

SELECT * FROM stores
WHERE STATE='CA'

SELECT TOP 5 * FROM TITLES
ORDER BY PRICE DESC
```

UNIVERSIDAD FEDERICO VILLAREAL

FACULTAD DE INGENIERIA INDUSTRIAL Y SISTEMAS

ESCUELA DE INGENIERIA DE SISTEMAS

GUIA DE LABORATORIO Nº 8

NOMBRE DE LA PRÁCTICA		**: USO DE INDICES**
CURSO	**:**	**SISTEMA DE BASE DATOS**
DOCENTE	**:**	**ING. MAIKE JAUREGUI**
CICLO	**:**	**2018-II**

I. Objetivos

1. Determinar la necesidad de empleo de índices en una base de datos
2. Emplear los comandos create y drop para el uso de índices

II. Introducción Teórica

Un índice sobre un campo determinado es una estructura de datos que permite encontrar rápidamente las tuplas que poseen un valor fijo en este campo.
Un índice adecuadamente situado en la tabla, ayudara a la base de datos a recuperar más rápidamente los datos, sobre todo cuando las tablas son muy grandes. Los índices se asocian a única tabla.

Tipos de Indices

Índice Agrupado (Clustered)
Un índice agrupado clasifica físicamente los datos. Acceder a los datos por medio de un índice agrupado es más rápido que utilizar un índice no agrupado. Una tabla solo puede tener un índice agrupado debido a que los datos están clasificados físicamente en el orden de la clave del índice.

Índice no agrupado (nonclustered)
En un índice no agrupado el nivel inferior del árbol (el nivel de hoja) contiene una marca que le indica a SQL SERVER donde encontrar la fila de datos correspondiente a la clave del índice.

Sintaxis

CREATE [CLUSTERED NONCLUSTERED] **INDEX** index_name **on** [table]

III. Requerimientos

- Máquina con SQL Server
- Guía Número 8 de base de datos

IV. Ejercicio Complementario

Crear la Base de Datos Indices y las siguientes tablas:

Tabla Conductores

Nombre de columna	Tipo de datos	Permitir val...
codigoc	varchar(5)	☐
nombre	varchar(25)	☑
localidad	varchar(25)	☑
categoria	varchar(25)	☑

codigoc	nombre	localidad	categoria
5	juan montes	quito	5
3	juan perez	ambato	8
4	juan salomon	manta	7
2	luis garro	chambo	10
1	memo dumas	robamba	9

Tabla Proyectos

Nombre de columna	Tipo de datos	Permitir val...
codigop	varchar(5)	☐
descripcion	varchar(25)	☑
localidad	varchar(25)	☑
cliente	varchar(25)	☑
telefono	varchar(9)	☑

codigop	descripcion	localidad	cliente	telefono
1	ecuador	quito	1	444455555
2	ecuador	ambato	2	333344444
3	ecuador	robamba	3	222222222
4	ecuador	cuenca	4	111111111
5	ecuador	chambo	5	666666666

1. En la tabla conductores cree un índice clusterado por el campo nombre
2. En la tabla proyectos cree un índice no clusterado por el campo localidad
3. Crear una nueva tabla llamada ciudad que tenga el código de ciudad y nombre y cree un índice clusterado y otro no clusterado en la misma. Ingrese al menos 10 registros.

```
create clustered index a
on conductores(nombre)

select * from conductores


create nonclustered index  b
on  proyectos(localidad)

select localidad  from proyectos
go

create clustered index c
on ciudad(cod)
go

create nonclustered index d
on ciudad(nombre)
go

select * from ciudad
```

UNIVERSIDAD FEDERICO VILLAREAL

FACULTAD DE INGENIERIA INDUSTRIAL Y SISTEMAS

ESCUELA DE INGENIERIA DE SISTEMAS

GUIA DE LABORATORIO Nº 9

NOMBRE DE LA PRÁCTICA		**: USO DE CONSULTAS DE INSERCIÓN, ACTUALIZACIÓN Y ELIMINACIÓN**
CURSO	**:**	**SISTEMA DE BASE DATOS**
DOCENTE	**:**	**ING. MAIKE JAUREGUI**
CICLO	**:**	**2018-II**

I. Objetivos

1. Agregar información a una o varias tablas almacenadas en una Base de datos.
2. Actualizar datos almacenados en una Base de datos tomando ciertos criterios
3. Eliminar datos o información innecesaria almacenada en una Base de datos

II. Introducción Teórica

Consultas de acción

Las consultas de acción son aquellas consultas que no devuelven ningún registro, sino que se encargan de acciones como:

- Agregar registros
- Actualizar registros
- Eliminar registros

Para comprender este tipo de consultas vamos a utilizar como ejemplo el siguiente diagrama de base de datos:

Creando la base de datos **Ventas**:
Tabla: Marca

```
CREATE TABLE marca(
CodigoMarca int NOT NULL,
NombreMarca varchar(50)
CONSTRAINT pk_marca PRIMARY KEY(CodigoMarca)
)
```

Tabla Producto

```
CREATE TABLE Producto
(CodigoProducto int NOT NULL,
NombreProducto varchar(50) NOT NULL,
PrecioUnitario decimal(18,2),
CodigoMarca int
CONSTRAINT pk_producto PRIMARY KEY (CodigoProducto)
CONSTRAINT fk_marca FOREIGN KEY (Codigomarca)
REFERENCES Marca(CodigoMarca)
)
```

Sentencia INSERT

INSERT. Permite agregar, adicionar o insertar uno o más registros a una (y solo una) tabla en una base de datos relacional.
Debe proporcionar en una sentencia INSERT igual número de campos o columnas.

Ejemplo 1:
Insertando sin colocar los nombres de los campos esto indica que se debe agregar datos a todas las columnas NOT NULL y se debe tomar en cuenta el orden de los campos de la tabla

```
INSERT INTO MARCA VALUES
(1,'HP')
```

Si se van a proporcionar valores para todos los campos de una tabla pueden omitirse los nombres de dichos campos en la instrucción.

Ejemplo 2:
Colocando los nombres de los campos, no importa el orden de como estén los campos en la tabla

```
INSERT INTO MARCA (CodigoMarca, NombreMarca) VALUES
(2, 'IBM')
INSERT INTO MARCA (CodigoMarca, NombreMarca) VALUES
(3, 'SAMSUNG')
```

Ejemplo 3:
Agregando varios registros al mismo tiempo en la tabla

```
INSERT INTO MARCA VALUES
(4,'APPLE'),
(5,'TOSHIBA'),
(6, 'ACER')
```

Ejemplo 4:
En el siguiente ejemplo se agrega datos a la tabla Producto en donde se respeta la relación entre las tablas, quiere decir que no se puede agregar un código de marca si este no ha sido ingresado previamente en la tabla donde se encuentre la clave primaria (en este caso en la tabla Marca)

```
INSERT INTO PRODUCTO VALUES
(1,'MONITOR LCD',500,1),
(2, 'TECLADO',150,2),
(3,'EQUIPO DE SONIDO',600,6),
(4,'PARLANTES',200,3),
(5,'AUDIFONOS',120,1),
(6,'MONITOR LED',700, 2),
(7,'IMPRESORA',400, 1),
(8,'MICRO SD',40,4),
(9,'USB',20,4),
(10,'CPU',1000,5)
```

Sentencia SELECT - INTO
SELECT - INTO se utiliza para crear una tabla a partir de los valores de otra tabla existente en la base de datos.

Ejemplo:
Se desea crear una tabla con los datos de la tabla producto que pertenezcan a la marca HP
Al hacer un SELECT a la tabla Producto se tienen los siguientes resultados:
`SELECT * FROM Producto`
Resultado:

	CodigoProducto	NombreProducto	PrecioUnitario	CodigoMarca
1	1	MONITOR LCD	500.00	1
2	2	TECLADO	150.00	2
3	3	EQUIPO DE SONIDO	600.00	6
4	4	PARLANTES	200.00	3
5	5	AUDIFONOS	120.00	1
6	6	MONITOR LED	700.00	2
7	7	IMPRESORA	400.00	1
8	8	MICRO SD	40.00	4
9	9	USB	20.00	4
10	10	CPU	1000.00	5

Ahora ejecutamos la siguiente consulta SELECT – INTO

```
SELECT * INTO [Producto Marca]
FROM Producto
WHERE CodigoMarca=1
```

Después de ejecutar la sentencia SELECT INTO y al hacer un SELECT a la tabla Producto Marca, se obtienen los siguientes resultados

```
SELECT * FROM [Producto Marca]
```

CodigoProducto	NombreProducto	PrecioUnitario	CodigoMarca
1	MONITOR LCD	500.00	1
5	AUDIFONOS	120.00	1
7	IMPRESORA	400.00	1

Sentencia INSERT INTO – SELECT

La consulta SELECT de la instrucción INSERT se puede utilizar para agregar valores a una tabla de la base de datos.

Ejemplo:

Crear la tabla Producto MarcaA, con las mismas propiedades de la tabla Producto

```
CREATE TABLE [Producto MarcaA]
(CodigoProducto int NOT NULL,
NombreProducto varchar(50),
PrecioUnitario decimal(18,2),
CodigoMarca int
CONSTRAINT pk_producto1 PRIMARY KEY (CodigoProducto)
CONSTRAINT fk_marca1 FOREIGN KEY (CodigoMarca)
REFERENCES Marca(CodigoMarca)
)
```

Al hacer un SELECT a la tabla Producto MarcaA esta no tiene datos

```
SELECT * FROM [Producto MarcaA]
```

En el siguiente ejemplo, la instrucción INSERT agregar en la tabla Producto MarcaE, los datos de la tabla Producto donde el valor del campo CodigoMarca sea igual a 4

```
INSERT INTO [Producto MarcaA]
SELECT CodigoProducto,NombreProducto,PrecioUnitario,CodigoMarca
FROM Producto
WHERE CodigoMarca=4
```

Al hacer un SELECT a la tabla esta debe tener los datos de los productos que pertenecen a la marca con código igual a 4

```
SELECT * FROM [Producto MarcaA]
```

CodigoProducto	NombreProducto	PrecioUnitario	CodigoMarca
8	MICRO SD	40.00	4
9	USB	20.00	4

Nota: a diferencia con la sentencia SELECT - INTO, aquí debe de crearse la tabla previamente

Sentencia UPDATE

UPDATE. Permite la actualización o modificación de uno o varios registros de una única tabla.
Se debe utilizar en conjunto con la cláusula **SET** con la cual se indicará(n) el(los) campo(s) a actualizar con el valor indicado.
Una segunda cláusula WHERE, opcional, permite indicar qué registros deben ser actualizados.
Si se omite la cláusula WHERE la ejecución de la consulta modificará todos los registros de la tabla.

Ejemplo 1. Actualizando datos a varios registros

Al hacer un SELECT a la tabla se obtiene los siguientes resultados:

```
SELECT * FROM [Producto MarcaA]
```

CodigoProducto	NombreProducto	PrecioUnitario	CodigoMarca
8	MICRO SD	40.00	4
9	USB	20.00	4

Con este ejemplo se actualiza el dato almacenado en el campo PrecioUnitario de cada registro de la tabla Producto MarcaA

```
UPDATE [Producto MarcaA] SET PrecioUnitario=45
```

Ahora hacer un SELECT a la tabla se obtiene los siguientes resultados:

CodigoProducto	NombreProducto	PrecioUnitario	CodigoMarca
8	MICRO SD	45.00	4
9	USB	45.00	4

Ejemplo 2. Actualizando datos donde se cumpla una o varias condiciones
Una condición:

Actualiza el precio del producto donde el Codigo del producto sea igual 8

```
UPDATE [Producto MarcaA] SET PrecioUnitario=40
WHERE CodigoProducto=8
```

Ahora hacer un SELECT a la tabla se obtiene los siguientes resultados:

```
SELECT * FROM [Producto MarcaA]
```

CodigoProducto	NombreProducto	PrecioUnitario	CodigoMarca
8	MICRO SD	40.00	4
9	USB	45.00	4

Varias condiciones:
Actualiza el precio del producto donde el Codigo del producto sea igual 9 y el codigo de la marca sea igual 4

```
UPDATE [Producto MarcaA] SET PrecioUnitario=42
WHERE CodigoProducto=9 AND CodigoMarca=4
```

Ahora hacer un SELECT a la tabla se obtiene los siguientes resultados:

```
SELECT * FROM [Producto MarcaA]
```

CodigoProducto	NombreProducto	PrecioUnitario	CodigoMarca
8	MICRO SD	40.00	4
9	USB	42.00	4

Sentencia DELETE

DELETE. Permite eliminar o borrar todos los registros de una tabla.
La sentencia DELETE no borra la estructura física de la tabla únicamente elimina los datos.

Ejemplo 1:
Elimina todos los registros de la tabla Producto MarcaA

```
DELETE FROM [Producto MarcaA]
```

Ahora hacer un SELECT a la tabla se obtiene los siguientes resultados:

```
SELECT * FROM [Producto MarcaA]
```

	CodigoProducto	NombreProducto	PrecioUnitario	CodigoMarca

Ejemplo 2:
La tabla productos tiene los siguientes datos:

CodigoProduc...	NombreProducto	PrecioUnitario	CodigoMarca
1	MONITOR LCD	500.00	1
2	TECLADO	150.00	2
3	EQUIPO DE SONIDO	600.00	6
4	PARLANTES	200.00	3
5	AUDIFONOS	120.00	1
6	MONITOR LED	700.00	2
7	IMPRESORA	400.00	1
8	MICRO SD	40.00	4
9	USB	20.00	4
10	CPU	1000.00	5

Eliminar los registros de la tabla Producto donde el código de la marca sea igual a 2

```
DELETE FROM Producto
WHERE CodigoMarca=2
```

Ahora hacer un SELECT a la tabla se observa que Ya no se encuentran los productos con código de marca 2 en los registros obtenidos

III. Requerimientos

- Máquina con SQL Server
- Guía Número 9 de base de datos

IV. Procedimiento

Ejercicio 1. Creación de la base de datos

Crear la base de datos Control Alumno

Ejercicio 2. Creación de tablas

Tabla Alumno

Nombre del Campo	Tipo de Dato	Tamaño	Permite valores nulos	Tipo de restricción
Carnet	char	8	no	Llave primaria
Nombre completo	varchar	50	si	

Tabla Materia

Nombre del Campo	Tipo de dato	Tamaño	Permite valores nulos	Tipo de restricción
Codigo	char	5	no	Llave primaria
Nombre	varchar	30	si	Valor único, el nombre de la materia no se puede repetir
UV	int		si	Check, en donde se aceptan valores entre 2 y 5

Tabla Inscripcion

Nombre del Campo	Tipo de dato	Tamaño	Permite valores nulos	Tipo de restricción
Carnet	char	**8**	No	Llave foránea la cual hace referencia a la tabla Alumno
CodigoMateria	char	**5**	No	Llave foránea la cual hace referencia a la tabla Materia
Ciclo	char	**5**	No	

La unión de los tres campos se crea una llave única

```sql
--tabla alumno
create table alumno(
carnet char(8) not null,
nombrecompleto varchar(50),
constraint pk_alumno primary key(carnet)
)
go

--tabla materia
create table materia(
codigo char(5) not null,
nombre varchar(30),
uv int,
constraint pk_materia primary key(codigo),
constraint u_nombre unique(nombre),
constraint ck_uv check(uv between 2 and 5)
)
go

--tabla inscripcion
create table inscripcion(
carnet char(8),
codigomateria char(5),
ciclo char(5),
constraint pk_inscripcion primary key (carnet, codigomateria, ciclo)
 )
 go


--llave foranea entre inscripcion y alumno
alter table inscripcion
add constraint fk_alumno_ins foreign key(carnet) references alumno (carnet)
on update cascade
on delete cascade

--llave foranea entre inscripcion y materia
alter table inscripcion
add constraint fk_materia_ins foreign key(codigomateria) references materia (codigo)
on update cascade
on delete cascade
```

Ejercicio 3

Insertar registros a la tabla Alumno

```
insert into alumno values
 ('GH121214', 'Gerardo Hierro'),
 ('VN121415', 'Veronica Nunez'),
 ('CD121515', 'Cesar Deras'),
 ('HL130334', 'Helen Lara'),
 ('GM119056', 'Gricelda Martinez')
```

Agregar registros a la tabla Materia

```
insert into materia values
 ('BD01', 'Base de Datos I', 4),
 ('IP01', 'Introduccion a la Programacion', 4),
 ('AL01', 'Algebra Lineal', 3),
 ('RD02', 'Redes de area amplia', 5),
 ('GE01', 'Gestion Empresarial', 2),
 ('HM02', 'Humanista II', 3)
```

Agregar registros a la tabla Inscripcion

```
insert into inscripcion values
('GH121214','BD01','C1-15'),
('GH121214', 'GE01', 'C1-15'),
('GH121214', 'HM02', 'C1-15')
```

Ejercicio 4. Uso de la instrucción UPDATE

1. El alumno que tiene el carnet GH121214 se debe cambiar el nombre de Gerardo Hierro a Gerardo Hernández, digitar la siguiente consulta:

   ```
   UPDATE ALUMNO SET NOMBRECOMPLETO='Gerardo Hernandez'
   WHERE CARNET='GH121214'
   ```

2. En el siguiente ejemplo se actualizará el carnet del alumno Gerardo Hernández y se verificará el funcionamiento de la instrucción ON UPDATE CASDADE

   Primero se realizará un SELECT a las dos tablas para verificar la información

   ```
   SELECT * FROM ALUMNO
   SELECT * FROM INSCRIPCION
   ```

	carnet	nombrecompleto
1	CD121515	Cesar Deras
2	GH121214	Gerardo Hernandez
3	GM119056	Gricelda Martinez
4	HL130334	Helen Lara
5	VN121415	Veronica Nunez

	carnet	codigomateria	ciclo
1	GH121214	BD01	C1-15
2	GH121214	GE01	C1-15
3	GH121214	HM02	C1-15

3. Realizar la actualización

```
UPDATE ALUMNO SET CARNET='GH111214'
WHERE NOMBRECOMPLETO='Gerardo Hernandez'
```

4. Realizar un select a las tablas y verificar la actualización de los datos

	carnet	nombrecompleto
1	CD121515	Cesar Deras
2	GH111214	Gerardo Hernandez
3	GM119056	Gricelda Martinez
4	HL130334	Helen Lara
5	VN121415	Veronica Nunez

	carnet	codigomateria	ciclo
1	GH111214	BD01	C1-15
2	GH111214	GE01	C1-15
3	GH111214	HM02	C1-15

Como se observa en los resultados se realizó la actualización de los datos en el campo carnet de la tabla Inscripcion al mismo tiempo que se ejecutó la consulta UPDATE en la tabla Alumno

Ejercicio 5. Uso de la instrucción DELETE

1. Crear la siguiente consulta de eliminación de registros

```
DELETE FROM ALUMNO WHERE CARNET= 'GH111214'
```

2. Comprobar los datos de las tablas Alumno e Inscripcion por medio de una consulta SELECT

	carnet	nombrecompleto
1	CD121515	Cesar Deras
2	GM119056	Gricelda Martinez
3	HL130334	Helen Lara
4	VN121415	Veronica Nunez

	carnet	codigomateria	ciclo

V. Ejercicio Complementario

Con la base de datos creada en el procedimiento realice las siguientes consultas

1. Agregar los siguientes registros a la tabla Alumnos

Carnet	NombreCompleto
MC1219854	Mauricio Campos
IP110943	Ignacio Perez
MU127895	Mikel Urrutia
OH132390	Oscar Hernandez
ML1390032	Mayra Lopez

2. Agregar los siguientes registros a la tabla Inscripcion

Carnet	CodigoMateria	Ciclo
CD121515	AL01	C1-14
CD121515	GE01	C1-14
CD121515	HM02	C1-15
GM119056	IP01	C2-14
GM119056	RD02	C2-14
HL130334	BD01	C1-15
VN121415	BD01	C1-15
VN121415	RD02	C1-15
MC129854	AL01	C1-14
MC129854	GE01	C1-14
IP110943	GE01	C1-15
IP110943	HM02	C1-15

3. Con la instrucción SELECT INTO, crear una tabla con el nombre MateriaUV que tenga los datos de la materia donde las unidades valorativas sean mayores o iguales a 4
4. Con la instrucción INSERT INTO – SELECT, crear una tabla con el nombre Alumno2012 en donde se almacenen aquellos alumnos que sean del año 2012
5. Crear las siguientes consultas de actualización de datos
   i. Modificar el nombre completo del alumno con carnet GM119056 a Martínez
   ii. Cambiar el carnet del alumno Martínez a GM119156
   iii. Modificar el ciclo de la inscripción de C1-14 a C1-15
   iv. Modificar el código de la materia HM02 a HM01
   v. Modificar el nombre completo del alumno con carnet IP110943 a Pereira
6. Crear las siguientes consultas de eliminación de datos
   I. Eliminar el alumno con el carnet GM119056
   II. Eliminar los alumnos en donde el carnet comience con letra M
   III. Eliminar la materia Introducción a la Programación
   IV. Eliminar el alumno Oscar Hernández
   V. Eliminar la inscripción donde el código de la materia es igual RD02 y el ciclo es igual C1-15

UNIVERSIDAD FEDERICO VILLAREAL

FACULTAD DE INGENIERIA INDUSTRIAL Y SISTEMAS

ESCUELA DE INGENIERIA DE SISTEMAS

GUIA DE LABORATORIO Nº 10

NOMBRE DE LA PRÁCTICA		**: USO DE FUNCIONES SQL, AGRUPANDO Y SUMARIZANDO DATOS**
CURSO	**:**	**SISTEMA DE BASE DATOS**
DOCENTE	**:**	**ING. MAIKE JAUREGUI**
CICLO	**:**	**2018-II**

I. Objetivos

1. Implementar las diferentes tipos de funciones en la selección de datos almacenados en una base de datos.
2. Combinar las funciones de agregado con las diferentes cláusulas de agrupación de datos

II. Introducción Teórica

1. Funciones de Agregación

Una función de agregación permite efectuar una operación aritmética que consolida los valores de una columna para toda la tabla o para los mismos valores agrupados según determinado criterio. La función devuelve un solo valor que es el consolidado para la tabla o para cada uno de los grupos.

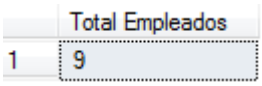
El siguiente cuadro muestra las funciones de agregación más utilizadas:

COMODIN	DESCRIPCION
AVG()	Retorna el promedio de los valores de una columna o expresión
COUNT()	Retorna la cuenta del número de valores distintos a null en una columna o expresión. Devuelve un valor de tipo int
COUNT(ALL expresion)	Evalúa la expresión en todas las filas del grupo y devuelve el número de valores no NULL
COUNT(DISTINCT expresion)	Evalúa la expresión en todas las filas del grupo y devuelve el número de valores no NULL únicos
MAX()	Retorna el valor máximo de una columna o expresión
MIN()	Retorna el valor mínimo de una columna o expresión
SUM()	Retorna la suma de los valores de una columna o expresión

Ejercicio 1. Uso de la función count()

1. Mostrar el número de empleados de la tabla empleados

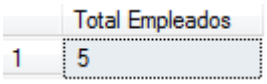
```
select count(*) as [Total Empleados]
from Empleados
```



	Total Empleados
1	9

2. Presentar el número de empleados de la tabla Employees en donde el dato almacenado en el campo Region sea diferente a NULL.

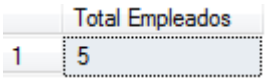
```
select count (ALL Región)as [Total Empleados]
from Empleados
```



	Total Empleados
1	5

Al final solo cuenta los empleados que tienen asignado una región en este caso WA, el mismo resultado, se obtendría a ejecutar la siguiente consulta:

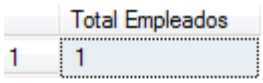
```
select count (Región)as [Total Empleados]
from Empleados
```



	Total Empleados
1	5

3. Se quiere contar a los empleados que tienen asignado una región, la cual esta no debe repetirse

```
select count (DISTINCT Región)as [Total Empleados]
from Empleados
```

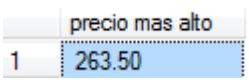


	Total Empleados
1	1

Ejercicio 2. Uso de la función max() y min()

1. Mostrar cómo obtener el precio mayor de un producto almacenado en el campo PrecioUnidad de la tabla Productos

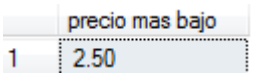
```
select max(PrecioUnidad) as [precio mas alto]
from Productos
```



	precio mas alto
1	263.50

2. Mostrar cómo obtener el precio menor de un producto almacenado en el campo PrecioUnidad de la tabla Productos

```
select min(PrecioUnidad) as [precio mas bajo]
from Productos
```



	precio mas bajo
1	2.50

Ejercicio 3. Uso de la función sum()

1. Sumar todos los datos almacenados en la columna Cantidad de la tabla **Detalles de Pedidos**.

```
select sum(cantidad) as [suma cantidad]
from [Detalles de pedidos]
```

	suma cantidad
1	51317

2. **La cláusula GROUP BY**

Se utiliza para agrupar los registros en base a determinado criterio y luego ejecutar cálculos sobre las columnas para consolidar los datos para cada uno de los grupos obtenidos.

Ejercicio 1.

En este ejercicio se devuelve información acerca de los pedidos de la tabla **Detalles de Pedidos**. La consulta agrupa y presenta cada identificador de producto (IdProducto) y calcula la cantidad total de pedido por cada producto. La cantidad total se calcula con la función de agregado SUM y presenta un valor para cada producto del conjunto de resultados, y al final se ordena la información con el ORDER BY.

```
select idproducto, sum(cantidad) as [Total de Productos]
from [Detalles de pedidos]
group by idproducto
order by idproducto
```

	idproducto	Total de Productos
1	1	828
2	2	1057
3	3	328
4	4	453
5	5	298
6	6	301
7	7	763
8	8	372
9	9	95
10	10	742

Ejercicio 2

Este ejemplo agrega una cláusula WHERE a la consulta del ejercicio anterior. Esta consulta restringe las filas al producto cuyo identificador (IDProducto), está dentro del rango 10 y 25 y, después, agrupa dichas filas y calcula la cantidad total del pedido.

```
select idproducto, sum(cantidad) as [Total de Productos]
from [Detalles de pedidos]
where idproducto between 10 and 25
group by idproducto
order by idproducto
```

	idproducto	Total de Productos
1	10	742
2	11	706
3	12	344
4	13	891
5	14	404
6	15	122
7	16	1158
8	17	978
9	18	539
10	19	723
11	20	313

Ejercicio 3

Calcular y mostrar la cantidad de clientes que se encuentran por cada país (Country) y región (Región) los cuales están almacenados en la tabla Cliente, los resultados se ordenaran por país (País)

```
 select país, región, count(*) as [Total de Clientes]
from clientes
group by país, región
order by país
```

	país	región	Total de Clientes
1	Alemania	NULL	11
2	Argentina	NULL	3
3	Austria	NULL	2
4	Bélgica	NULL	2
5	Brasil	RJ	3
6	Brasil	SP	6
7	Canadá	BC	2
8	Canadá	Québec	1
9	Dinamarca	NULL	2
10	España	NULL	5
11	Estados Unidos	AK	1

3. La cláusula HAVING

Utilizar la cláusula HAVING en columnas o expresiones para establecer condiciones en los grupos incluidos en un conjunto de resultados. La cláusula HAVING establece condiciones en la cláusula GROUP BY de una forma muy similar a como interactúa la cláusula WHERE con la instrucción SELECT. Cuando utilice la cláusula HAVING, considere los hechos e instrucciones siguientes:

- Utilice la cláusula HAVING sólo con la cláusula GROUP BY para restringir los agrupamientos. El uso de la cláusula HAVING sin la cláusula GROUP BY no tiene sentido.
- En una cláusula HAVING puede haber hasta 128 condiciones. Cuando utilice varias condiciones, tiene que combinarlas con operadores lógicos (AND, OR o NOT).
- Puede hacer referencia a cualquiera de las columnas que aparezcan en la lista de selección.

- No utilice la palabra clave ALL con la cláusula HAVING, porque la cláusula HAVING pasa por alto la palabra clave ALL y sólo devuelve los grupos que cumplen la cláusula HAVING.

Ejercicio 1

En este ejemplo se calcula el total de productos (Cantidad) que se han realizado en los pedidos almacenados en la tabla **Detalles de Pedidos** y donde esa cantidad tiene que ser mayor o igual a 100 unidades y las filas se ordenan en forma ascendente según la suma de las cantidades

```
select idproducto, sum(cantidad) as [Total de Productos]
from [Detalles de pedidos]
group by IdProducto
having sum(cantidad) >=100
order by sum(cantidad)
```

	idproducto	Total de Productos
1	15	122
2	37	125
3	48	138
4	67	184
5	50	235
6	66	239
7	73	293
8	32	297
9	74	297
10	5	298
11	6	301

4. El operador ROLL UP

Permite resumir los valores de grupo y se usa normalmente para producir promedios acumulados o sumas acumuladas. Puede hacer esto aplicando la función de agregación en la lista de columnas de SELECT para cada columna en la cláusula GROUP BY moviéndose de izquierda a derecha.

Ejercicio 1

```
select idproducto, idpedido, sum(cantidad) as [total cantidad]
from [Detalles de pedidos]
group by idproducto, idpedido
with rollup
```

	idproducto	idpedido	total cantidad
1	1	10285	45
2	1	10294	18
3	1	10317	20
4	1	10348	15
5	1	10354	12
6	1	10370	15
7	1	10406	10
8	1	10413	24
9	1	10477	15
10	1	10522	40

	idproducto	idpedido	total cantidad
35	1	11031	45
36	1	11035	10
37	1	11047	25
38	1	11070	40
39	1	NULL	828

2230	77	11068	28
2231	77	11077	2
2232	77	NULL	791
2233	NULL	NULL	51317

Se muestra el total general para cada idproducto (el idproducto 1 tiene un total en cantidad de 828)
Se muestra el total general (la suma de todos los idproductos en cantidad es 51317)

5. El operador CUBE

Permite crear y resumir todas las posibles combinaciones de grupos basadas en la cláusula GROUP BY.

Ejercicio 1

```
select idproducto, idpedido, sum(cantidad) as [total cantidad]
from [Detalles de pedidos]
group by idproducto, idpedido
with cube
```

	idproducto	idpedido	total cantidad
1	11	10248	12
2	42	10248	10
3	72	10248	5
4	NULL	10248	27
5	14	10249	9
6	51	10249	40
7	NULL	10249	49

	idproducto	idpedido	total cantidad
2986	NULL	NULL	51317
2987	23	NULL	580
2988	46	NULL	548
2989	69	NULL	714
2990	29	NULL	746
2991	75	NULL	1155
2992	15	NULL	122
2993	9	NULL	95
2994	3	NULL	328
2995	52	NULL	500
2996	72	NULL	806
2997	66	NULL	239
2998	32	NULL	297

	idproducto	idpedido	total cantidad
3004	55	NULL	903
3005	43	NULL	580
3006	49	NULL	520
3007	67	NULL	184
3008	27	NULL	365
3009	21	NULL	1016
3010	58	NULL	534
3011	64	NULL	740
3012	38	NULL	623
3013	7	NULL	763
3014	44	NULL	601
3015	1	NULL	828

El total general en cantidad para la orden 10248 es 27.
Se muestra el total general (la suma de todos los idproductos en cantidad es 51317)
En la fila 3015 se muestra el total en cantidad de idproducto

III. Requerimientos

- Máquina con SQL Server
- Guía Número 10 de base de datos

IV. Ejercicio Complementario

Haciendo uso de la base de datos **Northwind** realice las siguientes consultas:

1. Mostrar cuantos Clientes hay por cada Compañía (CompanyName)
2. Se desea conocer cuántos empleados hay por cada Territorio, utilice la tabla EmployeeTerritories
3. Tomando como base de la consulta del punto 3, crear una consulta donde implemente la instrucción ROLLUP.
4. Se desea conocer cuántos territorios hay por cada región (RegionID), utilice la tabla Territories
5. Mostrar de la tabla Order Details aquellos pedidos en donde las unidades sumen más de 50 y ordenar los datos en forma descendente según la suma de esas cantidades

Haciendo uso de la base de datos **AdventureWorks2012** realice las siguientes consultas:
Utilizando la tabla **Sales.SalesOrderDetail** realice las siguientes consultas:

1. Mostrar la suma de las unidades vendidas (OrderQty) por cada orden (SalesOrderID)
2. Mostrar el promedio de ventas (LineTotal) por cada orden (SalesOrderID)
3. Mostrar la venta (LineTotal) máxima por cada orden (SalesOrderID)
4. Mostrar la venta (LineTotal) mínima por cada orden (SalesOrderID)

V. Actividad Complementaria

1. Creación de base de datos: Nombre de la base de datos: **Control_de_libros**
2. Crear las tablas tomando en cuenta: Crear las relaciones entre las tablas (llaves primarias y llaves foráneas).
3. El formato de las tablas son:

Tabla Autor

CodigoAutor	PrimerNombre	PrimerApellido	FechaNacimiento	Nacionalidad	Edad
PL001	**Pablo**	**López**	**19/08/1960**	**Colombiana**	**54**
CM002	**Claudia**	**Martínez**	**10/06/1970**	**Salvadoreña**	**45**
PM003	**Patricio**	**Murry**	**12/12/1967**	**Española**	**47**
NH004	**Nuria**	**Hernández**	**03/09/1980**	**Colombiana**	**34**
HM005	**Helen**	**Martínez**	**22/11/1980**	**Española**	**34**
JR006	**José**	**Roldan**	**13/09/1967**	**Colombiana**	**54**

Tabla Editorial

CodigoEditorial	Nombre	Pais
ED001	Omega 2000	Colombia
ED002	Anaya Multimedia	España
ED003	McGrawHill	Inglaterra
ED004	Reyes	México
ED005	Prentice Hall	Inglaterra

Tabla Libro

CodigoLibro	Titulo	ISBN	AñoEdicion	CodigoEditorial
BDCOL00001	Fundamentos de Base de datos	12333-8999988	2004	ED001
BDESP00002	La Biblia de SQL Server 2008	3444-99888-88	2008	ED002
PRCOL00002	Programación orientada a objetos	8999-9999444	2011	ED001
DWING00003	Diseño Web y Hojas de estilo	300096-99999	2010	ED003
PRING00004	Programación en C/C++	45667-87878	2009	ED005
HJMEX00005	Uso de hojas de estilo con JavaScript	0990-87878787	2008	ED004
ABESP00006	Administración de Base de datos	585885-88484848	2010	ED002

Tabla DetalleAutorLibro

CodigoAutor	CodigoLibro
PL001	BDCOL00001
NH004	BDCOL00001
CM002	PRCOL00002
PM003	BDESP00002
PM003	DWING00003
HM005	PRING00005
CM002	ABESP00006
NH004	HJMEX00005
JR006	DWING00003

4. Agregar los registros a las tablas

5. Realizar las siguientes consultas, implementando funciones de agregado o funciones de cadenas:

a. Mostrar cuantos autores hay por cada nacionalidad
b. Calcular cuántos libros hay por cada editorial
c. Mostrar la cantidad de editoriales que hay por cada país
d. Mostrar el libro donde el año de edición sea el más actual
e. Mostrar el libro donde el año de edición sea el menos actual
f. Calcular el promedio de las edades de los autores
g. Mostrar cuántos libros ha escrito cada autor
h. Mostrar cuantos autores nacieron en el mismo mes
i. Mostrar el nombre y apellido del autor, haciendo uso de la función CONCAT
j. Contar cuantos libros tienen en su título la palabra PROGRAMACION
k. Mostrar las iniciales del Primer apellido de cada AUTOR y ordenarlos de forma descendente
l. Mostrar los autores que nacieron antes del año 1980
m. Se desea conocer aquellos autores que han escrito más de un libro, mostrar el código de autor y la cantidad de libros
n. Mostrar cuantos autores comienzan con la inicial P en su primer nombre
o. Mostrar cuantos libros finalizan con la palabra Base de datos en su título

UNIVERSIDAD FEDERICO VILLAREAL

FACULTAD DE INGENIERIA INDUSTRIAL Y SISTEMAS

ESCUELA DE INGENIERIA DE SISTEMAS

GUIA DE LABORATORIO Nº 11

NOMBRE DE LA PRÁCTICA		**: CONSULTAS A MÚLTIPLES TABLAS. USO DE JOIN Y SUBCONSULTAS.**
CURSO	**:**	**SISTEMA DE BASE DATOS**
DOCENTE	**:**	**ING. MAIKE JAUREGUI**
CICLO	**:**	**2018-II**

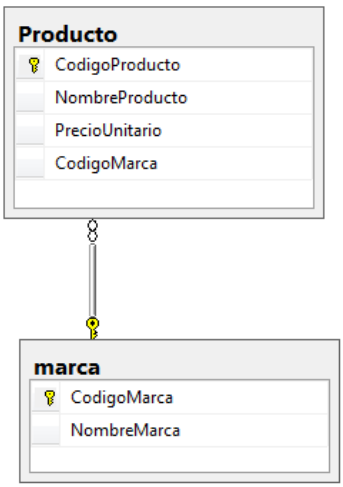
I. Objetivos

1. Combinar datos de dos o más tablas por medio de JOINS.
2. Seleccionar información de varias tablas utilizando SUBCONSULTAS

II. Introduccion Teorica

COMBINACIÓN DE TABLAS

Una combinación es una operación que permite consultar dos o más tablas para producir un conjunto de resultados que incorpore filas y columnas de cada una de las tablas en cuestión. Las tablas se combinan en función de las columnas que son comunes a ambas tablas.



La combinación de campos de tablas distintas sólo es posible cuando se han definido campos relacionados entre tablas. Esto es si existe un campo clave primaria en una tabla que aparece como clave foránea en la otra tabla.
La sentencia JOIN en SQL permite combinar registros de dos o más tablas en una base de datos relacional.

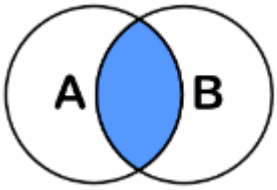
Tipos de Combinaciones

9. **Combinación interna INNER JOIN**
10. **Combinación Cruzada CROSS JOIN**
11. **Combinación externa OUTER JOIN**
    - **i.** LEFT OUTER JOIN o LEFT JOIN
    - **ii.** RIGHT OUTER JOIN o RIGHT JOIN
    - **iii.** FULL OUTER JOIN o FULL OUTER JOIN

La palabra OUTER es opcional y no añade ninguna función

A. **INNER JOIN**

Se utiliza para mostrar los datos coincidentes entre las tablas de donde se quiere mostrar la información:




```
SELECT <lista_campos>
FROM <TablaA A>
INNER JOIN <TablaB B>
ON A.Key=B.Key
```

Nota: **ON** se utiliza para colocar los nombres de los campos con los cuales se ha realizado la relación entre las tablas

Ejemplo 1

Obtener el nombre de los proveedores y los productos que estos suministran ordenados por el nombre de proveedor

```
SELECT PV.NombreCompañía , P.NombreProducto
FROM Proveedores AS PV INNER JOIN Productos AS P
ON PV.IdProveedor = P.IdProveedor
ORDER BY 1
```

Ejemplo 2

Obtener el idproducto, nombreproducto y nombrecategoria. Utilice una combinación de las tablas Productos y Categorias.

```
SELECT P.IdProducto,P. NombreProducto, C.NombreCategoría
FROM Productos AS P INNER JOIN Categorías AS C
ON P.IdCategoría=C.IdCategoría
```

Ejemplo 3
Retorna el idpedido, fechapedido, y todas las columnas de la tabla Cliente. Esta consulta responde a la pregunta "Mostrar todos los números de pedidos, fecha y los datos del cliente que hizo el pedido.

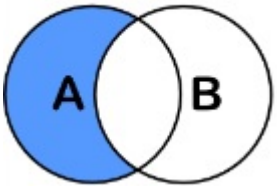
```
SELECT P.IdPedido, P.FechaPedido, C.*
FROM Pedidos AS P INNER JOIN Clientes AS C
ON P.IdCliente = C.IdCliente
ORDER BY P.IdCliente
```

Ejemplo 4
Obtener el idproducto, nombreproducto, nombrecategoria y nombrecompania donde idproveedor es igual a 1. Utilice una combinación de las tablas Productos, Categorias y Proveedores. (combinación de tres tablas)

```
SELECT P.IdProducto,P. NombreProducto, C.NombreCategoría, PV.NombreCompañía
FROM Productos AS P INNER JOIN Categorías AS C
ON P.IdCategoría=C.IdCategoría  INNER JOIN Proveedores AS PV
ON P.IdProveedor= PV.IdProveedor
WHERE P.IdProveedor= 1
```

B. LEFT JOIN

Muestra los registros de la tabla izquierda más los registros coincidentes con la tabla derecha




```
SELECT <lista_campos>
FROM <TablaA A>
LEFT JOIN <TablaB B>
ON A.KEY=B.KEY
```

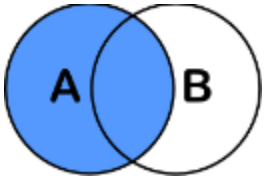
Ejemplo 1
Retornar las columnas idcliente, nombrecompania de la tabla Clientes utilizando una combinación externa izquierda (LEFT OUTER JOIN) con la tabla Pedidos. De esta tabla obtiene las columnas idpedido y fechapedido

```
SELECT   C.IdCliente, C.NombreCompañía, P.IdPedido, P.FechaPedido
FROM Clientes AS C  LEFT OUTER JOIN Pedidos AS P
ON P.IdCliente = C.IdCliente
ORDER BY P.IdCliente
```

Ejemplo 2
Retornar todos los clientes que no realizaron pedidos. Utilizar una combinación externa izquierda LEFT OUTER JOIN

```
SELECT C.IdCliente, C.NombreCompañía, P.IdPedido,
P.FechaPedido
FROM Clientes AS C  LEFT OUTER JOIN Pedidos AS P
ON P.IdCliente = C.IdCliente
WHERE P.Idpedido IS NULL
```

LEFT JOIN (IS NULL)
Muestra los registros de la tabla izquierda menos los registros coincidentes con la tabla derecha




```
SELECT <lista_campos>
FROM <TablaA A>
LEFT JOIN <TablaB B>
ON A.Key=B.Key
WHERE B.Key IS NULL
```

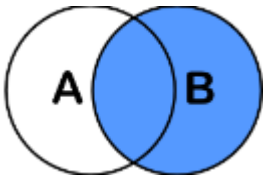
Ejemplo 1

Se desea conocer los empleados que no han atendido ningún pedido

```
select idpedido, e.idempleado, apellidos
from empleados e
left join pedidos p
on e.IdEmpleado=p.IdEmpleado
where p.IdEmpleado is null
```

C. **RIGHT JOIN**

Muestra los registros de la tabla derecha más los registros coincidentes con la tabla izquierda




```
SELECT <lista_campos>
FROM <TablaA A>
RIGHT JOIN <TablaB B>
ON A.KEY=B.KEY
```

Ejemplo 1
El ejemplo siguiente inserta 3 registros a la tabla Compania de Envios. Estos datos son necesarios para crear los ejercicios siguientes

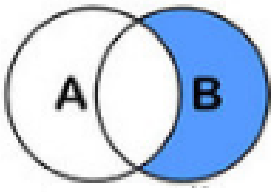
```
INSERT [Compañías de Envíos]VALUES
(4, 'Aero Condor', '(503) 555-8831')
INSERT [Compañías de Envíos] VALUES
 (5, 'American Express', '(503) 555-9761')
INSERT [Compañías de Envíos] VALUES
 (6, 'Amazonas S.A.', '(503) 555-2231')
```

Ejemplo 2
Retorna las columnas idpedido de la tabla Pedidos y nombrecompania de la tabla compañía de envios utilizando una combinación externa derecha RIGHT OUTER JOIN

```
SELECT P.IdPedido, CE.NombreCompañía
FROM Pedidos AS P
RIGHT OUTER JOIN [Compañías de Envíos] AS CE
ON P.FormaEnvío = CE.IdcompañíaEnvíos
ORDER BY 2
GO
```

RIGHT JOIN (IS NULL)

Muestra los registros de la tabla derecha menos los registros coincidentes con la tabla izquierda



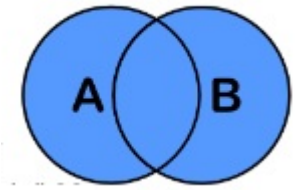

```
SELECT <lista_campos>
FROM <TablaA A>
RIGHT JOIN <TablaB B>
ON A.Key=B.Key
WHERE A.Key IS NULL
```

Ejemplo 1
Mostrar que proveedor no ha ofrecido productos

```
select nombreproducto
from productos p
right join Proveedores pr
on p.IdProveedor=pr.IdProveedor
where p.IdProveedor is null
```

D. FULL JOIN

Muestra los registros de la tabla izquierda y la tabla derecha más los registros coincidentes entre ambas



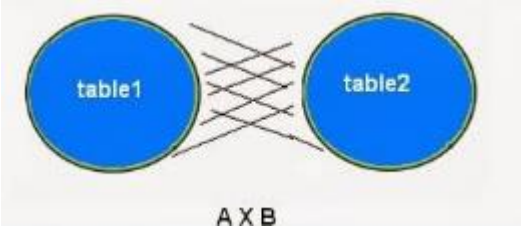

```
SELECT <lista_campos>
FROM <TablaA A>
FULL JOIN <TablaB B>
ON A.KEY=B.KEY
```

Ejemplo 1
Retornar las columnas idcliente de la tabla Pedidos y nombrecompania de la tabla compañía de envios utilizando una combinación externa completa FULL OUTER JOIN

```
SELECT C.IdCliente, C.NombreCompañía,P.IdPedido
FROM Clientes AS C FULL OUTER JOIN Pedidos AS P
ON   C.IdCliente =  P.IdCliente
ORDER BY 3
```

E. <u>CROSS JOIN</u>

Una combinación cruzada que no tenga una cláusula WHERE genera el producto cartesiano de las tablas involucradas en la combinación.
El tamaño del conjunto de resultados de un producto cartesiano es igual al número de filas de la primera tabla multiplicado por el número de filas de la segunda tabla.



Ejemplo 1
Usar la base de datos northwind

Ejecutamos las siguientes consultas para conocer la cantidad de filas o registros tienen las siguientes tablas:

```
select * from products
```

```
Northwind | 00:00:00 | 77 filas
```

```
select * from suppliers
```

```
Northwind | 00:00:00 | 29 filas
```

```
select productname, companyname, contactname
from products p
cross join suppliers s
```

```
Northwind | 00:00:00 | 2233 filas
```

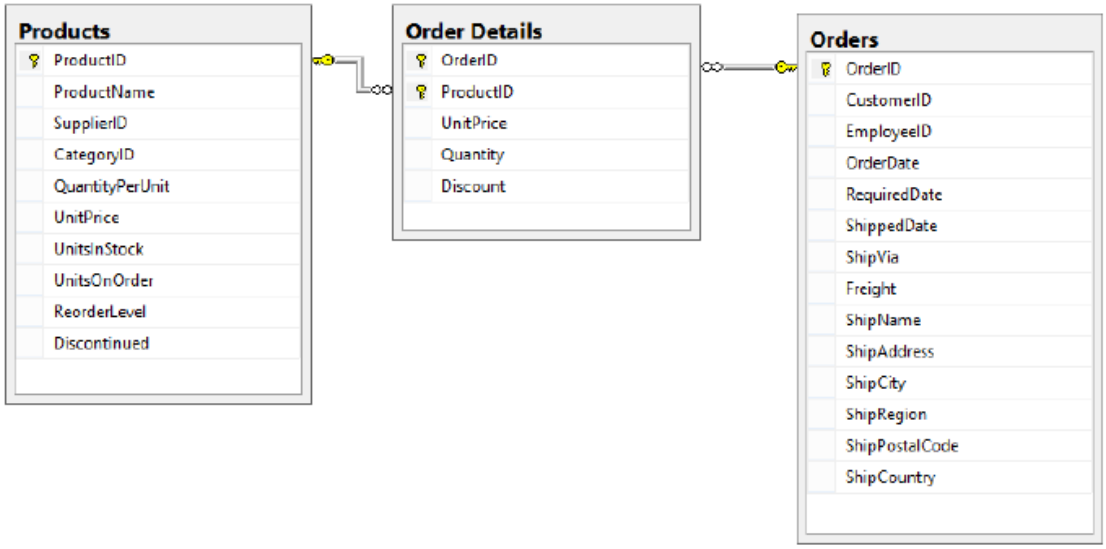
Como resultado tenemos 2233 filas o registros, ya que si multiplicamos las 77 filas de la primera tabla por las 29 filas de la segunda obtenemos ese resultado

III. Requerimientos

- Guia de Laboratorio
- Maquina con SQL SERVER

IV. Ejercicios Complementarios

1. Haciendo uso de INNER JOIN mostrar los campos OrderDate y ProductID de las tablas Orders y Order Details donde el dato almacenado en el campo OrderDate sea igual 8 de Julio de 1996
2. Se desea mostrar cuantas cantidades de cada producto se han vendido y la fecha de la venta de cada uno de ellos, se debe tomar en cuenta el siguiente diagrama relacional



Campos a mostrar: ProductID, ProductName, Quantity y OrderDate

3. Crear la siguiente base de datos

Insertar registros a la tabla Empleados

ID	NOMBRES	APELLIDOS	EDAD	FECHAINICIO
01234567-8	CARLOS FIDEL	ARGUETA MIRANDA	45	2006-08-21
12345678-9	JUAN FRANCISCO	VILLALTA ALVARADO	32	2010-02-27
78901234-5	RAUL ALEJANDRO	PONCIO VALLADARES	32	2010-02-27
89012345-6	MIGUEL EDUARDO	MORALES CLAROS	26	2010-08-21
90123456-7	FABRICIO DAVID	ALAS FLORES	30	2008-12-01

Insertar registros a la tabla Maquina

COD_MAQUINA	DESCRIPCION	MARCA	MODELO	FECHAINGRESO
M00001	TALADORA DE ELEMENTOS VARIOS	CATERPILLAR	EVO2000	2006-01-31
M00002	APLANADORA DE SUELOS Y OTROS	CATERPILLAR	FLU5000	2006-01-31
M00003	PULVERIZADORA DE ELEMENTOS	CATERPILLAR	ASD2001	2006-01-31
M00004	CONCRETERA	MG	EDS	2006-05-31
M00005	MAQUINA ESPECIAL PARA PROYECTO 10	MG	SFD	2006-05-31
M00006	MAQUINA ESPECIAL PARA PROYECTO 30	MG	SFD	2010-12-01

Insertar registros a la tabla Bitacora

CORRELATIVO	ID	COD_MAQUINA	TIEMPO_USO	LUGAR
1	12345678-9	M00001	250	SANTIAGO NONUALCO
2	01234567-8	M00002	300	SANTIAGO NONUALCO
3	90123456-7	M00003	500	ALEGRIA USULATAN
4	89012345-6	M00004	300	ALEGRIA USULATAN
5	90123456-7	M00005	250	SANTIAGO NONUALCO
6	01234567-8	M00002	125	SANTIAGO NONUALCO
7	12345678-9	M00003	375	ALEGRIA USULATAN
8	12345678-9	M00004	200	ALEGRIA USULATAN

Crear las siguientes consultas:

a. Mostrar los empleados que hayan o estén haciendo uso de una maquina
i. Campos a mostrar: Nombres del empleado, Marca, Modelo y Descripción de la maquina

b. Mostrar los empleados que todavía no tienen asignada una maquina
i. Campos a mostrar: Nombres y Apellidos del empleado y Código de la maquina

c. Mostrar las maquinas que no están asignadas a un proyecto
i. Campos a mostrar: Nombres y Apellidos del empleado y Descripción de la maquina

1. SUBCONSULTAS

Una subconsulta (subquery) es una sentencia "select" anidada en otra sentencia "select", "insert", "update" o "delete" (o en otra subconsulta).

Las subconsultas se emplean cuando una consulta es muy compleja, entonces se la divide en varios pasos lógicos y se obtiene el resultado con una única instrucción y cuando la consulta depende de los resultados de otra consulta.

Hay tres tipos básicos de subconsultas:

- las que retornan un solo valor escalar que se utiliza con un operador de comparación o en lugar de una expresión.
- las que retornan una lista de valores, se combinan con "in", o los operadores "any", "some" y "all".
- los que testean la existencia con "exists".

1.1 Subconsultas como expresión

Una subconsulta puede reemplazar una expresión. Dicha subconsulta debe devolver un valor escalar (o una lista de valores de un campo).

Las subconsultas que retornan un solo valor escalar se utiliza con un operador de comparación o en lugar de una expresión:
La sintaxis es la siguiente:

```
select CAMPOS
 from TABLA
 where CAMPO OPERADOR (SUBCONSULTA);
```

Ejemplo:
Mostrar el título, autor y precio del libro más costoso:

```
select titulo,autor, precio
 from libros
 where precio=
  (select max(precio) from libros
```

1.2 Subconsultas con in

El resultado de una subconsulta con "in" (o "not in") es una lista. Luego que la subconsulta retorna resultados, la consulta exterior los usa.

La sintaxis es la siguiente:

```
...where EXPRESION in (SUBCONSULTA);
```

Ejemplo:

Mostrar los nombres de las editoriales que han publicado libros de un determinado autor:

```
select nombre
from editoriales
where codigo in
(select codigoeditorial
 from libros
 where autor='Richard Bach');
```

La subconsulta (consulta interna) retorna una lista de valores de un solo campo (codigo) que la consulta exterior luego emplea al recuperar los datos.

1.3 Subconsultas any, some, all

Any

Chequean si alguna fila de la lista resultado de una subconsulta se encuentra el valor especificado en la condición.

La sintaxis es:

```
...VALORESCALAR OPERADORDECOMPARACION
 ANY (SUBCONSULTA);
```

Ejemplo:

Mostrar los títulos de los libros de "Borges" que pertenecen a editoriales que han publicado también libros de "Richard Bach", es decir, si los libros de "Borges" coinciden con ALGUNA de las editoriales que publicó libros de "Richard Bach":

```
select titulo
from libros
where autor='Borges' and
codigoeditorial = any
(select e.codigo
 from editoriales as e
 join libros as l
 on codigoeditorial=e.codigo
  where l.autor='Richard Bach');
```

La consulta interna (subconsulta) retorna una lista de valores de un solo campo (puede ejecutar la subconsulta como una consulta para probarla), luego, la consulta externa compara cada valor de "codigoeditorial" con cada valor de la lista devolviendo los títulos de "Borges" que coinciden.

All

También compara un valor escalar con una serie de valores. Chequea si TODOS los valores de la lista de la consulta externa se encuentran en la lista de valores devuelta por la consulta interna.

Sintaxis:

```
VALORESCALAR OPERADORDECOMPARACION all (SUBCONSULTA);
```

Ejemplo:

Mostrar si TODAS las editoriales que publicaron libros de "Borges" coinciden con TODAS las editoriales que publicaron libros de "Richard Bach":

```
select titulo
from libros
where autor='Borges' and
codigoeditorial = all
(select e.codigo
from editoriales as e
join libros as l
on codigoeditorial=e.codigo
 where l.autor='Richard Bach');
```

La consulta interna (subconsulta) retorna una lista de valores de un solo campo (puede ejecutar la subconsulta como una consulta para probarla), luego, la consulta externa compara cada valor de "codigoeditorial" con cada valor de la lista, si TODOS coinciden, devuelve los títulos.

1.4 Subconsultas Exists

Los operadores "exists" y "not exists" se emplean para determinar si hay o no datos en una lista de valores.

Cuando se coloca en una subconsulta el operador "exists", SQL Server analiza si hay datos que coinciden con la subconsulta, no se devuelve ningún registro, es como un test de existencia; SQL Server termina la recuperación de registros cuando por lo menos un registro cumple la condición "where" de la subconsulta.

EXISTS simplemente verifica si la consulta interna arroja alguna fila. Si lo hace, entonces la consulta externa procede. De no hacerlo, la consulta externa no se ejecuta, y la totalidad de la instrucción SQL no arroja nada.

La sintaxis es la siguiente:

```
... where exists (SUBCONSULTA);
```

Ejemplo

Se usa una subconsulta correlacionada con un operador "exists" en la cláusula "where" para devolver una lista de clientes que compraron el artículo "lapiz":

```
select cliente,numero
from facturas as f
where exists
(select *from Detalles as d
 where f.numero=d.numerofactura
 and d.articulo='lapiz');
```

ANY o **SOME**	Compara con cualquier registro de la subconsulta. La instrucción es válida si hay un registro en la subconsulta que permite que la comparación sea cierta. Se suele utilizar la palabra ANY (SOME es un sinónimo)
ALL	Compara con todos los registros de la consulta. La instrucción resulta cierta si es cierta toda comparación con los registros de la subconsulta
IN	No usa comparador, ya que sirve para comprobar si un valor se encuentra en el resultado de la subconsulta
NOT IN	Comprueba si un valor no se encuentra en una subconsulta
EXISTS	Este operador devuelve verdadero si la consulta que le sigue devuelve algún valor. Si no, devuelve falso. Se utiliza normalmente mediante consultas correlacionadas

Ejercicio Complementario

Usar la base de datos subconsultas

1. Selecciona los datos de los empleados que trabajan en el departamento de Marketing
2. Selecciona los empleados cuyo salario sea inferior al salario medio (avg(salary))
3. Selecciona los países que están en el mismo continente que Argentina
4. Selecciona los empleados cuyo salario sea inferior al salario medio de los empleados que son representantes de ventas.(job_id='SA_MAN')
5. Obtener todos los empleados con el mismo oficio que David Austin
6. Obtener todos los empleados cuyo puesto de trabajo es Sales Manager
7. Obtener todos los empleados que no trabajan en el departamento que Steven King
8. Obtener los datos de los empleados que ganan más que cualquiera de los empleados del departamento 30
9. Visualiza los departamentos que están en Seattle
10. Selecciona el nombre, apellidos y el nombre del departamento de los empleados que trabajan en Seatle
11. Visualiza aquellos empleados que no trabajen en el departamento de Marketing ni el de Ventas
12. Visualiza aquellos empleados que trabajen en el departamento de Marketing o el de Ventas
13. Listar los países de Asia o Europa.
14. Obtener los datos de los empleados cuyo nombre empieza por H y cuyo salario es mayor que el de algún empleado del departamento 100
15. Encuentra el nombre y los apellidos del jefe de David Austin
16. Selecciona todos los empleados que trabajan en USA.
17. Encuentra los empleados que ganan el salario mínimo correspondiente a su puesto de trabajo.
18. Obtener los datos de los empleados que tienen el salario más alto de su departamento.
19. Seleccionar los departamentos que empiecen por R que no tengan empleados
20. Selecciona los departamentos en los que haya personas cuyo nombre comienza por la letra A

UNIVERSIDAD FEDERICO VILLAREAL

FACULTAD DE INGENIERIA INDUSTRIAL Y SISTEMAS

ESCUELA DE INGENIERIA DE SISTEMAS

GUIA DE LABORATORIO Nº 12

NOMBRE DE LA PRÁCTICA		**: CREACION DE VISTAS**
CURSO	**:**	**SISTEMA DE BASE DATOS**
DOCENTE	**:**	**ING. MAIKE JAUREGUI**
CICLO	**:**	**2018-II**

I. Objetivos

1. Diseñar y crear Vistas para la selección de información
2. Implementar la programación con comandos SQL

II. Introducción Teórica

Una vista es una tabla virtual cuyo contenido está definido por una consulta. Al igual que una tabla real, una vista consta de un conjunto de columnas y filas de datos con un nombre. Sin embargo, la vista no existe como conjunto de valores de datos almacenados en una base de datos. Las filas y las columnas de datos proceden de tablas a las que se hace referencia en la consulta que define a la vista y se producen de forma dinámica cuando se hace referencia a la vista.

Sintaxis

CREATE VIEW *nom_vista* **AS** *instruccion_select*

Ejemplo 1
Crear una vista que devuelva el idcliente, nombrecompania, dirección y país de la tabla clientes

```
CREATE VIEW Vista_01
AS
SELECT IdCliente, NombreCompañía, Dirección, País
FROM Clientes
GO

SELECT * FROM Vista_01
```

Ejemplo 2
Crear la vista vista_02 que permita visualizar solo a los clientes que residen en Brasil

```
CREATE VIEW vista_02
AS
SELECT *
```

```
FROM Clientes
WHERE País = 'Brasil'
GO
--Visualizamos los datos
SELECT * FROM Vista_02
GO
```

Ejemplo 3
Crear la vista ClientesProveedores_Ciudad que presenta información de los clientes y proveedores (utilizar unión)

```
CREATE VIEW ClientesProveedores_Ciudad AS
SELECT Ciudad, NombreCompañía, NombreContacto, 'Clientes' AS Relationship
FROM Clientes
UNION SELECT Ciudad, NombreCompañía, NombreContacto, 'Proveedores'
FROM Proveedores
GO
SELECT * FROM  ClientesProveedores_Ciudad
```

Ejemplo 4
Crear la vista ListaAlfabetica_ProductosCategoria donde se presenta el idproducto, nombreproducto y nombrecategoria. Utilizar las tablas productos y categorías de la base de datos nw

```
CREATE VIEW ListaAlfabetica_ProductosCategoria AS
SELECT IDProducto,NombreProducto, Categorías.NombreCategoría
FROM Categorías INNER JOIN Productos  ON Categorías.Idcategoría =
Productos.IdCategoría
WHERE (((Productos.Suspendido)=0))
GO

SELECT * FROM  ListaAlfabetica_ProductosCategoria
ORDER BY NombreCategoría,NombreProducto
```

Ejemplo 5
Crear la vista ProductosPorCategoria, retorna el nombre de categoría y los productos pertenecientes a dicha categoría. Utilizar las tablas categorias y productos de la base de datos nw

```
CREATE VIEW ProductosPorCategoria AS
SELECT Categorías.NombreCategoría, Productos.NombreProducto,
Productos.CantidadPorUnidad, Productos.UnidadesEnExistencia, Productos.Suspendido
FROM Categorías INNER JOIN Productos ON Categorías.IdCategoría =
Productos.IdCategoría
WHERE Productos.Suspendido <> 1
GO
SELECT * FROM ProductosPorCategoria
```

Ejemplo 6
Crear la vista Subtotal_Pedidos, calcula el subtotal por cada pedido realizado

```
CREATE VIEW Subtotal_Pedidos AS
SELECT [Detalles de Pedidos].IdPedido,
Sum(CONVERT(money,([Detalles de Pedidos].PrecioUnidad
*Cantidad * (1-Descuento)/100))*100) AS SubTotal
FROM [Detalles de Pedidos]
GROUP BY [Detalles de Pedidos].IdPedido
```

```
GO
SELECT * FROM Subtotal_Pedidos
```

Modificación y eliminación de vistas

A menudo, las vistas se alteran como respuesta a las peticiones de información adicional por parte de los usuarios o a causa de cambios en la definición de las tablas subyacentes. Para alterar una vista puede quitarla y volverla a crear, o bien puede ejecutar la instrucción **ALTER VIEW**

Eliminación de vistas

Si ya no necesita una vista, puede quitar su definición de la base de datos con la instrucción DROP VIEW.

Crear la siguiente vista

```
create view ShipStatusView
as
select idpedido, FechaEnvío, destinatario
from clientes c inner join pedidos p
on c.IdCliente=p.IdCliente
where fechapedido < FechaEntrega
GO

SELECT * FROM ShipStatusView
```

Ejemplo 1
Agregar un nuevo campo (ciudad destinatario de la tabla Pedidos) a la consulta ShipStatusView

```
alter view ShipStatusView
as
select idpedido, FechaEnvío, destinatario, CiudadDestinatario
from clientes c inner join pedidos p
on c.IdCliente=p.IdCliente
where fechapedido < FechaEntrega
GO
```

Ejemplo 2
Eliminar la vista ShipStatusView

```
drop view ShipStatusView
```

III. Requerimientos

- Guia de Laboratorio
- Maquina con SQL SERVER

IV. Ejercicio Complementario

1. Usar la base de datos Northwind. Seleccionar los campos ProductID, ProductName, SupplierID, CategoryID, QuantityPerUnit, UnitPrice, Discontinued. Definir una restricción WHERE la cual es: SupplierID = 14
2. Crear una vista donde se muestren las ventas de un producto las cuales se han realizado en el año 1997, los campos que podría mostrar son los siguientes:
   a. Nombre del producto (ProductName)
   b. Fecha de pedido (OrderDate)
   c. Fecha de envió (ShippedDate)
   d. Calcular el subtotal (UnitPrice*Quantity)
   3. Mostrar el código del producto, el nombre del producto y el precio por unidad de todos los productos de la empresa
   4. Mostrar todos los productos cuya categoría sea Beverages
   5. Mostrar los datos del cliente y las fechas de las ordenes que estos han realizado
   6. Cuantos productos existen por cada categoría
   7. Mostrar el promedio de los precios unitarios de las categorías: Produce y Confections

UNIVERSIDAD FEDERICO VILLAREAL

FACULTAD DE INGENIERIA INDUSTRIAL Y SISTEMAS

ESCUELA DE INGENIERIA DE SISTEMAS

GUIA DE LABORATORIO Nº 13

NOMBRE DE LA PRÁCTICA		**: CREACION DE PROCEDIMIENTOS ALMACENADOS**
CURSO	**:**	**SISTEMA DE BASE DATOS**
DOCENTE	**:**	**ING. MAIKE JAUREGUI**
CICLO	**:**	**2018-II**

I. Objetivos

1. Diseñar y crear procedimientos almacenados para la selección de información
2. Implementar la programación con comandos SQL

II. Introducción Teórica

Un procedimiento almacenado es una colección de sentencias SQL que están guardadas en el servidor.

¿Para qué se utilizan?

Los procedimientos almacenados se utilizan para agrupar las instrucciones de T-SQL y cualquier lógica asociada necesaria para llevar a cabo una tarea.

¿Cómo se ejecutan?

Cuando un procedimiento almacenado es ejecutado por primera vez se compila, se crea y se guarda en memoria su plan de ejecución., luego SQL utiliza ese plan de ejecución cuando se vuelve a llamar al procedimiento sin volver a compilarlo nuevamente.

¿Qué nos proporcionan?

Nos proporcionan a nosotros los usuarios un acceso fácil a la base de datos, se puede tener acceso a la base de datos sin tener que conocer los detalles de la arquitectura de tablas - simplemente se ejecutan los procedimientos almacenados que llevan a cabo las tareas solicitadas.

Procedimientos de Entrada

Se esperan parámetros. Condicionan la ejecución del mismo a procedimiento almacenado

Sintaxis

```
create procedure nombreprocedimiento
@parametro1 tipo,
@parametro2 tipo
as
sentencia
```

Ejemplo

Crear la tabla Usuario

Nombre de columna	Tipo de datos	Permitir val...
idusuario	char(10)	☐
nombre	varchar(20)	☐
usuario	varchar(20)	☐
contrasena	varchar(20)	☐
edad	int	☐
sexo	varchar(20)	☐

Insertar los siguientes registros:

idusuario	nombre	usuario	contrasena	edad	sexo
011	alex	nick	alexjau	45	M
012	romina	romi	rm101	14	F

Crear el siguiente procedimiento almacenado

```
create procedure seleccion1--nombre procedimiento
@edad int,--parametro entrada
@sexo varchar(20)--parametro entrada
as
select * from usuario
where edad>=@edad and sexo=@sexo --contenido del procedimiento
GO

exec seleccion1 18, 'M' --ejecucion pasando los valores
GO
```

Procedimientos de Salida

Cuando se ejecuta el procedimiento, que me devuelva una salida con una variable

Sintaxis

```
create procedure nombreprocedimiento
@parametro1 tipo output,
as
sentencia
```

Crear el siguiente procedimiento almacenado: cuantos varones mayores o iguales a 18 años hay en la tabla

```
create procedure seleccion2 --nombre procedimiento
@edad int,--parametro entrada
@sexo varchar(20), --parametro entrada
@count int output --parametro salida
as
set @count=(select count(idusuario) from usuario where edad>=@edad and sexo=@sexo)
GO

declare @total int --declaramos la variable
exec seleccion2 18, 'M', @total output --ejecutamos el procedimiento
select @total--mostramos el resultado
```

III. REQUERIMIENTOS

- SQL SERVER
- GUIA DE LABORATORIO

IV. PROCEDIMIENTOS

Usar la base de datos Nwind

1. **Crear un procedimiento almacenado utilizando un parámetro de entrada que devuelva solo los clientes de un país especificado**

```
CREATE PROCEDURE spejemplo_01
@pais varchar(15)
AS
    SELECT  *
    FROM Clientes
   WHERE País=@pais
GO


EXEC spejemplo_01 'brasil'
```

2. **Crear un procedimiento que contiene tres parámetros N1, N2 y N3 de salida, que devuelve la suma**

```
Create Procedure spejemplo_02
@N1 int,
@N2 int,
@N3 int Output
As
   Set @N3 = @N1+@N2
   return (@N3)
Go

/* EJECUTAR EL PROCEDIMIENTO */
Declare @a Int, @b Int, @c Int
Set @a = 5
Set @b = 3
Exec spejemplo_02 @a, @b, @c output
Select @c
```

3. **Crear un procedimiento almacenado que permita insertar, actualizar y eliminar registros de una tabla**

```
CREATE DATABASE SPTutorial
GO
USE SPTutorial
GO

CREATE TABLE employees(
        employee_id numeric(18, 0) IDENTITY(1,1) PRIMARY KEY NOT NULL,
        full_name varchar(75) NULL,
        gender varchar(6) NULL,
        department varchar (25) NULL,
        position varchar(50) NULL,
        salary money  NULL)
GO

--procedimiento almacenado para insertar registros
CREATE PROCEDURE usp_employee_insert
        @full_name varchar(75),@gender varchar(6), @department varchar(25), @position
varchar(50),@salary money
AS
BEGIN
        INSERT INTO employees (full_name,gender,department,position,salary)
        VALUES  (@full_name,@gender,@department,@position,@salary)
END
GO

exec usp_employee_insert 'kevin', 'mas', 'sistemas', 'programador', 3000
go
--procedimiento almacenado para actualizar registros
CREATE PROCEDURE usp_employee_update
        @full_name varchar(75),@gender varchar(6), @department varchar(25), @position
varchar(50),@salary money,@employee_id numeric(18,0)
AS
BEGIN
        UPDATE employees SET full_name = @full_name
        ,gender = @gender
        ,department = @department
        ,position = @position
        ,salary = @salary
        WHERE employee_id = @employee_id
END
GO

exec usp_employee_update  'alex', 'mas', 'doctor', 'gastro', 5000, 1

select * from employees
go
```

```
--procedimiento almacenado para eliminar registros

CREATE PROCEDURE usp_employee_delete
        @employee_id numeric(18,0)
AS
BEGIN
        DELETE FROM employees WHERE employee_id = @employee_id
END

GO
exec usp_employee_delete 1
```

4. Crear un procedimiento que permita realizar el mantenimiento de una tabla

```
CREATE TABLE employee(
id INTEGER NOT NULL PRIMARY KEY,
first_name VARCHAR(10),
last_name VARCHAR(10),
salary DECIMAL(10,2),
city VARCHAR(20),
)

INSERT INTO employee VALUES (2, 'Monu', 'Rathor',4789,'Agra')
INSERT INTO employee VALUES (4, 'Rahul' , 'Saxena', 5567,'London');
INSERT INTO employee VALUES (5, 'prabhat', 'kumar', 4467,'Bombay');
INSERT INTO employee VALUES (6, 'ramu', 'kksingh', 3456, 'jk');
go
select * from employee

go
create PROCEDURE MasterInsertUpdateDelete
(
@id INTEGER,
@first_name VARCHAR(10),
@last_name VARCHAR(10),
@salary DECIMAL(10,2),
@city VARCHAR(20),
@StatementType nvarchar(20) = ''
)
AS
BEGIN
IF @StatementType = 'Insert'
BEGIN
insert into employee (id,first_name,last_name,salary,city) values( @id, @first_name, @last_name, @salary,
@city)
END
IF @StatementType = 'Select'
BEGIN
```

```
select * from employee
END
IF @StatementType = 'Update'
BEGIN
UPDATE employee SET
First_name = @first_name, last_name = @last_name, salary = @salary,
city = @city
WHERE id = @id
END
else IF @StatementType = 'Delete'
BEGIN
DELETE FROM employee WHERE id = @id
END
end

exec MasterInsertUpdateDelete  7, 'sachin', 'tendukar', 23457,'bombay', 'insert'
exec  MasterInsertUpdateDelete   2,  'mike',  'jauregui',  1234,  'peru',  'update'
exec  MasterInsertUpdateDelete   2,  'mike',  'jauregui',  1234,  'peru',  'delete'
```

V. EJERCICICIO COMPLEMENTARIO

Usar la base de datos ventas

1. Crear un P.A q reciba el código de un cliente como input y muestre el número de comprobantes de pago que ha recibido.
2. Crear un P.A. que tenga como input DNI de un personal y muestre el número de comprobantes emitidos por dicho personal entre una fecha inicial y una fecha final
3. Crear un P.A. que reciba como input el cod de un distrito y que devuelva como output el nomdist
4. Crear un P.A. que reciba como input el nombre de una marca y devuelva como output el número de productos q tiene dicha marca.

```sql
--5) crear un P.A q reciba el codigo de un cliente  como imput y muestre el numero de
comprobanets de pago quqe ah recibido.

alter proc comprobantes
    @codcli CHAR(4)
    as
    select  COUNT (CODCOMP)"NRO DE COMPROBANTES" FROM COMPROBANTE_CABECERA
    WHERE CODCLI= @CODCLI
     GO
     EXEC comprobantes 'cl03'
     go

    --creAr un p.a que tenga como imput DNI DE UN PERSONAL Y MUESTRE EL NUMERO DE COMPROBANTES
EMITIDOS  POR DICHO PERSONAL
     --ENTRE UNA FECHA INICIAL Y UNA FECHA FINAL.

     Alter PROC NCOMP
     @DNI CHAR(8),
     @FECHAINI DATE ,
     @FECHAFIN DATE
     AS
     SELECT  COUNT(CODCOMP) "NRO DE COMPROBANTES" FROM COMPROBANTE_CABECERA cc,personal p
     WHERE p.codper=cc.codper and  DNI=@DNI AND FECHACOMP between @FECHAINI and @FECHAFIN
     GO
     EXEC NCOMP '03296565','16/06/10','12/10/11'
     GO


--PROCEDIMIENTOS ALMACENADOS CON OUMPUST

--1)CREAR UN PA QUE RECIBA COMO IMPUT EL COD DE UN DIST Y QUE DEVUELVA COMO OUTPUTS EL NOMDIST

CREATE PROC  NOMDIST

@COD CHAR(4), @NOMDIST VARCHAR(30) OUTPUT

AS
SET @NOMDIST =(SELECT NOMDIST FROM DISTRITOS WHERE CODDIST=@COD)
GO
--Ejecutar el P.A con output

Declare  @x varchar(30)
exec NOMDIST 'D005',@X OUTPUT
PRINT @X

--2)CREAR UN P.A QUE RECIBA COMO IMPUT EL OMBRE DE UNA MARCA Y DEVUELVA COMO OUTPUT EL NRO DE
PRODUCTOS Q TIENE DICHA MARCA.

CREATE PROC NROPRODUCTOS1

@MARCA VARCHAR(30) , @N int OUTPUT
AS
SET @N=(SELECT COUNT (CODPROD)FROM PRODUCTOS P,MARCAS M  WHERE P.CODMAR = m.CODMAR AND
NOMMAR=@MARCA)
GO

Declare  @n varchar(30)
exec NROPRODUCTOS1 'IBM',@N OUTPUT
PRINT @N
```

UNIVERSIDAD FEDERICO VILLAREAL

FACULTAD DE INGENIERIA INDUSTRIAL Y SISTEMAS

ESCUELA DE INGENIERIA DE SISTEMAS

GUIA DE LABORATORIO Nº 14

NOMBRE DE LA PRÁCTICA		**: FUNCIONES EN SQL SERVER**
CURSO	**:**	**SISTEMA DE BASE DATOS**
DOCENTE	**:**	**ING. MAIKE JAUREGUI**
CICLO	**:**	**2018-II**

I. Objetivos

Identificar los tipos de funciones que hay en SQL SERVER

II. Introducción Teórica

Una función es un conjunto de sentencias que operan como una unidad lógica, una rutina que retorna un valor. Una función tiene un nombre, acepta parámetros de entrada y retorna un valor escalar o una tabla.

SQL Server admite 3 tipos de funciones definidas por los usuarios clasificados según el valor retornado:
1) escalares: retornan un valor escalar;
2) con valores de tabla en linea
3) con valores de tabla de declaración múltiple
Las funciones definidas por el usuario se crean con la instrucción "create function" y se eliminan con "drop function".

b. **Funciones Escalares:** Una función escalar retorna un único valor. Como todas las funciones, se crean con la instrucción "create function".
La sintaxis básica es:

```
create function NOMBRE
@PARAMETRO TIPO=VALORPORDEFECTO)
returns TIPO
begin
INSTRUCCIONES
return VALOR
end;
```

La cláusula "returns" indica el tipo de dato retornado.
El cuerpo de la función, se define en un bloque "begin...end" que contiene las instrucciones que retornan el valor

Ejemplo 1
Crear función denominada "f_promedio" que recibe 2 valores y retorna el promedio

```
create function f_promedio
 (@valor1 decimal(4,2),
  @valor2 decimal(4,2)
 )
 returns decimal (6,2)
 as
 begin
 declare @resultado decimal(6,2)
set @resultado=(@valor1+@valor2)/2
return @resultado
end
go
```

Al hacer referencia a una función escalar, se debe especificar el propietario y el nombre de la función:
select dbo.f_promedio(5.5,8.5);

Ejemplo 2
Crear una función que devuelva la cantidad de ordenes realizadas por un determinado cliente (campo customerid)

Usar la base de datos northwind

```
create function getnumorders(@customerid char(5))
returns int
as begin
declare @ret int
select @ret= count(*)
from orders
where customerid=@customerid
return @ret
end
go

select dbo.getnumorders('HILAA') AS CANTIDAD;
```

Otra forma:

```
declare @numorders int
exec @numorders=getnumorders 'HILAA'
print @numorders
```

2. Funciones con valores de tabla en línea

Una función con valores de tabla en línea retorna una tabla que es el resultado de una única instrucción "select".

Es similar a una vista, pero más flexible en el empleo de parámetros. En una vista no se puede incluir un parámetro, lo que hacemos es agregar una cláusula "where" al ejecutar la vista. Las funciones con valores de tabla en línea funcionan como una vista con parámetros.

Sintaxis

```
create function NOMBREFUNCION
 (@PARAMETRO TIPO=VALORPORDEFECTO)
 returns table
 as
 return (
  select CAMPOS
  from TABLA
  where CONDICION
 )
```

Ejemplo 1

Creamos una función con valores de tabla en línea que recibe un valor de autor como parámetro:

```
create database  f1
go
use f1
go

create table libros(
  codigo int identity,
  titulo varchar(40),
  autor varchar(30),
  editorial varchar(20),
  precio decimal(6,2)
 );


 insert into libros values('Uno','Richard Bach','Planeta',15);
 insert into libros values('Ilusiones','Richard Bach','Planeta',10);
 insert into libros values('El aleph','Borges','Emece',25);
 insert into libros values('Aprenda PHP','Mario Molina','Siglo XXI',55);
```

```
insert into libros values('Alicia en el pais','Lewis Carroll','Paidos',35);
insert into libros values('Matematica estas ahi','Paenza','Nuevo siglo',25);

--se crea la funcion
 create function f_libros
(@autor varchar(30)='Borges')
returns table
as
return (
 select titulo,editorial
 from libros
 where autor like '%'+@autor+'%'
)

--se genera la funcion
select *from f_libros('Bach');
```

Ejemplo 2

```
create database  f2
go
use f2
go


create table empleado(
id int primary key,
name varchar(20),
birth datetime,
gender varchar(20),
departmentid int)
go

set dateformat ymd
insert into empleado values(1, 'sam', '1980/12/30', 'masculino', 1)
insert into empleado values(2, 'pam', '1982/01/09', 'femenino', 2)
insert into empleado values(3, 'jhon', '1985/08/22', 'masculino', 1)
insert into empleado values(4, 'sara', '1979/11/29', 'femenino', 3)
insert into empleado values(5, 'todd', '1978/11/29', 'masculino', 1)

select * from empleado
go
```

```
create function fn_ILTVF_GetEmployees()
returns table
as
return (select id, name, cast(birth as date) as dob from empleado)
go

select * from fn_ILTVF_GetEmployees()
```

Ejemplo 3
Crear una función que devuelva la cantidad de ordenes realizadas por un determinado cliente (campo customerid)

Usar la base de datos notrhwind

```
create function getnumorders1(@customerid nchar(5))
returns table
as
return select *
from Orders
where CustomerID=@customerid
go

select * from getnumorders1('HILAA')
go
```

3. Funciones con valores de tabla de declaración múltiple

Son similares a funciones con valores de tabla en línea, con pequeñas diferencias.

Ejemplo 1

```
create function fn_MSTVF_GetEmployees()
returns @table table (id int, name varchar(20), dob date)
as
begin
insert into @table
select id, name, cast(birth as date) as dob from empleado
return
end
go

select * from fn_MSTVF_GetEmployees()
go
```

Ejemplo 2

```
create function getnumorders2(@customerid nchar(5))
returns @mytable table(orderid int, orderdate date)
as
begin
insert into @mytable
select orderid, orderdate
from Orders
where CustomerID=@customerid
return
end
go
select * from getnumorders2('HILAA')
```

Diferencias entre la segunda y tercera función

a. En la segunda función, la cláusula RETURNS no puede contener la estructura de la tabla. En la tercera función, se especifica la estructura de la tabla que retorna
b. En la segunda función, no puede tener el bloque BEGIN END, como la tercera función si la tiene.

III. Requerimientos

- **Maquina SQL SERVER**
- **Guía de Laboratorio**

IV. Ejercicio Complementario

Funciones Escalares

1. Crear una función que retorne el número de productos vendidos en un año determinado de una marca determinada
2. Crear una función que retorne el factorial de un número entero

Funciones de Tabla

3. Crear una función que retorne una tabla con los siguientes campos: codprod, nomprod, nommarca y precio teniendo como input la función el nombre de la marca
4. Crear una función que reciba como input el nombre de un distrito y retorne una tabla con las siguientes columnas: codper, nomper, dni, dirección y nombre de distrito.
5. Crear una función que reciba como input nombre de un personal y retorne una tabla con los siguientes campos: codcomp ,nomcli,nomper,fechacomprobante de todos los comprobantes emitidos por ese personal
6. Crear una función que reciba como input el nomcar, y devuelva una tabla con los siguientes campos: nomcar, número de personas que tienen dicho cargo, suma de sueldos por cargo y promedio de sueldo por cargo

UNIVERSIDAD FEDERICO VILLAREAL

FACULTAD DE INGENIERIA INDUSTRIAL Y SISTEMAS

ESCUELA DE INGENIERIA DE SISTEMAS

GUIA DE LABORATORIO Nº 15

NOMBRE DE LA PRÁCTICA		**: TRIGGERS EN SQL SERVER**
CURSO	**:**	**SISTEMA DE BASE DATOS**
DOCENTE	**:**	**ING. MAIKE JAUREGUI**
CICLO	**:**	**2018-II**

I. Objetivos

Qué el estudiante sea capaz de:

- Utilizar desencadenadores para activar mensajes de advertencia
- Implementar los diferentes tipos de desencadenadores

II. Introducción Teórica

En general, un disparador es un tipo especial de procedimiento almacenado que se ejecuta automáticamente cuando ocurre un evento en el servidor de la base de datos.
DML significa lenguaje de manipulación de datos. Las instrucciones INSERT, UPDATE y DELETE son declaraciones DML. Los disparadores DML se activan cuando se modifican los datos utilizando eventos INSERT, UPDATE y DELETE.

Los disparadores de DML se pueden clasificar de nuevo en 2 tipos.
1. After triggers(Después de los disparadores , a veces llamados como activadores FOR)
2. Instead of triggers (En lugar de desencadenantes)

2.1 After Triggers

After triggers, como su nombre lo indica, se dispara después de la acción de disparo. Las instrucciones INSERT, UPDATE, y DELETE, hacen que se active un desencadenador posterior después de que las instrucciones respectivas completen la ejecución.

Por otro lado, como su nombre lo dice, Instead of triggers, se dispara en lugar de la acción de activación. Las instrucciones INSERT, UPDATE y DELETE pueden hacer que un desencadenador INSTEAD OF se active en lugar de la ejecución de la instrucción respectiva.

Ejemplo 1

Ejemplo para AFTER TRIGGER para el evento INSERT en la tabla tblEmployee:

```
--crear la base de datos t01
create database t01
go
--usar la base de datos t01
use t01
go
--crear la tabla tblEmployee
CREATE TABLE tblEmployee
(
  Id int Primary Key,
  Name nvarchar(30),
  Salary int,
  Gender nvarchar(10),
  DepartmentId int
)
go
--Insertar data a la tabla  tblEmployee
Insert into tblEmployee values (1,'John', 5000, 'Male', 3)
Insert into tblEmployee values (2,'Mike', 3400, 'Male', 2)
Insert into tblEmployee values (3,'Pam', 6000, 'Female', 1)
```

tblEmployee

Id	Name	Salary	Gender	DepartmentId
1	John	5000	Male	3
2	Mike	3400	Male	2
3	Pam	6000	Female	1
4	Todd	4800	Male	4
5	Sara	3200	Female	1
6	Ben	4800	Male	3

```
--crear la tabla  tblEmployeeAudit
CREATE TABLE tblEmployeeAudit
(
  Id int identity(1,1) primary key,
  AuditData nvarchar(1000)
)
```

Cuando se agrega un nuevo empleado, queremos capturar la ID y la fecha y la hora, el nuevo empleado se agrega en la tabla tblEmployeeAudit. La forma más fácil de lograr esto, es tener un AFTER INSERT para el evento INSERT.

```
--Ejemplo para  AFTER TRIGGER para el evento  INSERT en la tabla tblEmployee:
CREATE TRIGGER tr_tblEMployee_ForInsert
ON tblEmployee
FOR INSERT
AS
BEGIN
 Declare @Id int
 Select @Id = Id from inserted

 insert into tblEmployeeAudit
 values('New employee with Id  = ' + Cast(@Id as nvarchar(5)) + ' is added at ' + cast(Getdate() as nvarchar(20)))
END
```

En el activador, obtenemos el ID de la tabla insertada. Entonces, ¿qué es esta tabla insertada? La tabla INSERTADA, es una tabla especial utilizada por los activadores DML. Cuando agrega una nueva fila a la tabla tblEmployee, también se realizará una copia de la fila en la tabla insertada, a la que solo puede acceder un activador. No puede acceder a esta tabla fuera del contexto del activador. La estructura de la tabla insertada será idéntica a la estructura de la tabla tblEmployee.

Entonces, ahora si ejecutamos la siguiente instrucción INSERT en tblEmployee. Inmediatamente, después de insertar la fila en la tabla tblEmployee, el disparador se activa (se ejecuta automáticamente), y también se inserta una fila en tblEmployeeAudit.

```
Insert into tblEmployee values (7,'Tan', 2300, 'Female', 3)
```

Ejemplo 2

Ejemplo para AFTER TRIGGER para el evento DELETE en la tabla tblEmployee:

Ahora capturemos información de auditoría, cuando se elimina una fila de la tabla, tblEmployee.

```
CREATE TRIGGER tr_tblEMployee_ForDelete
ON tblEmployee
FOR DELETE
AS
BEGIN
 Declare @Id int
 Select @Id = Id from deleted
insert into tblEmployeeAudit
 values('An existing employee with Id  = ' + Cast(@Id as nvarchar(5)) + ' is deleted at ' + Cast(Getdate() as
nvarchar(20)))
END
```

La única diferencia aquí es que, estamos especificando, el evento desencadenante como DELETE y recuperando el ID de fila eliminado de la tabla DELETED. DELETED table, es una tabla especial utilizada por los activadores DML.

Cuando eliminas una fila de la tabla tblEmployee, una copia de la fila eliminada estará disponible en la tabla DELETED, a la que solo puede acceder un activador. Al igual que en la tabla INSERTED, no se puede acceder a la tabla DELETED, fuera del contexto del activador y, la estructura de la tabla DELETED será idéntica a la estructura de la tabla tblEmployee.

```
delete from tblEmployee where id=1
```

After update trigger

Los disparadores hacen uso de 2 tablas especiales, INSERTADAS y ELIMINADAS. La tabla insertada contiene los datos actualizados y la tabla eliminada contiene los datos antiguos. El desencadenante posterior para el evento UPDATE, utiliza tablas insertadas y eliminadas.

Ejemplo 3

Visualizar el contenido de la tabla INSERTED y DELETED

```
--Create AFTER UPDATE trigger script:
Create trigger tr_tblEmployee_ForUpdate
on tblEmployee
for Update
as
Begin
 Select * from deleted
 Select * from inserted
End

Update tblEmployee set Name = 'Tods', Salary = 2000,
Gender = 'Female' where Id = 3
```

Inmediatamente después de la ejecución de la instrucción UPDATE, se activa el desencadenador AFTER UPDATE, y debería ver los contenidos de las tablas INSERTED y DELETED.

Ejemplo 4

El siguiente desencadenante AFTER UPDATE, audita la información de los empleados al ACTUALIZAR, y almacena los datos de auditoría en la tabla tblEmployeeAudit.

```
Alter trigger tr_tblEmployee_ForUpdate
on tblEmployee
for Update
as
Begin

    -- Declare variables to hold old and updated data
    Declare @Id int
    Declare @OldName nvarchar(20), @NewName nvarchar(20)
    Declare @OldSalary int, @NewSalary int
    Declare @OldGender nvarchar(20), @NewGender nvarchar(20)
    Declare @OldDeptId int, @NewDeptId int

    -- Variable to build the audit string
    Declare @AuditString nvarchar(1000)
```

```
    -- Load the updated records into temporary table
    Select *
    into #TempTable
    from inserted

    -- Loop thru the records in temp table
    While(Exists(Select Id from #TempTable))
    Begin
        --Initialize the audit string to empty string
        Set @AuditString = ''

        -- Select first row data from temp table
        Select Top 1 @Id = Id, @NewName = Name,
        @NewGender = Gender, @NewSalary = Salary,
        @NewDeptId = DepartmentId
        from #TempTable

        -- Select the corresponding row from deleted table
        Select @OldName = Name, @OldGender = Gender,
        @OldSalary = Salary, @OldDeptId = DepartmentId
        from deleted where Id = @Id

    -- Build the audit string dynamically
        Set @AuditString = 'Employee with Id = ' + Cast(@Id as nvarchar(4)) + ' changed'
        if(@OldName <> @NewName)
            Set @AuditString = @AuditString + ' NAME from ' + @OldName + ' to ' + @NewName

        if(@OldGender <> @NewGender)
            Set @AuditString = @AuditString + ' GENDER from ' + @OldGender + ' to ' + @NewGender

        if(@OldSalary <> @NewSalary)
            Set @AuditString = @AuditString + ' SALARY from ' + Cast(@OldSalary as nvarchar(10))+ ' to ' +
Cast(@NewSalary as nvarchar(10))

    if(@OldDeptId <> @NewDeptId)
            Set @AuditString = @AuditString + ' DepartmentId from ' + Cast(@OldDeptId as nvarchar(10))+ ' to ' +
Cast(@NewDeptId as nvarchar(10))
        insert into tblEmployeeAudit values(@AuditString)

        -- Delete the row from temp table, so we can move to the next row
        Delete from #TempTable where Id = @Id
    End
End

update tblEmployee set name='mike', salary=2000, gender='male'
where id=2

select * from tblEmployee
select * from tblEmployeeAudit
```

a. **Triggers Varios Eventos**

Un disparador puede definirse para más de una acción; en tal caso, deben separarse con comas.

Ejemplo 5
Un club almacena los datos de sus socios en una tabla denominada "socios", las inscripciones en "inscritos" y en otra tabla "morosos" guarda los documentos de los socios que deben matrículas.

Crear un trigger para evitar que se inscriban socios que deben matrículas y no permitir que se eliminen las inscripciones de socios deudores. El trigger se define para ambos eventos en la misma sentencia de creación.

```
create database te
go
use te
go
create table socios(
  documento char(8) not null,
  nombre varchar(30),
  domicilio varchar(30),
  constraint PK_socios primary key(documento)
 )
 go

 create table inscritos(
  numero int identity,
  documento char(8) not null,
  deporte varchar(20),
  matricula char(1),
  constraint FK_inscritos_documento
   foreign key (documento)
   references socios(documento),
  constraint CK_inscritos_matricula check (matricula in ('s','n')),
  constraint PK_inscritos primary key(documento,deporte)
 )
 go

  create table morosos(
  documento char(8) not null
 )
 go
 insert into socios values('22222222','Ana Acosta','Avellaneda 800');
 insert into socios values('23333333','Bernardo Bustos','Bulnes 345');
 insert into socios values('24444444','Carlos Caseros','Colon 382');
 insert into socios values('25555555','Mariana Morales','Maipu 234');

 insert into inscritos values('22222222','tenis','s');
 insert into inscritos values('22222222','natacion','n');
 insert into inscritos values('23333333','tenis','n');
 insert into inscritos values('24444444','futbol','s');
 insert into inscritos values('24444444','natacion','s');

 insert into morosos values('22222222');
 insert into morosos values('23333333');
 go
```

```
 --Crear un trigger para evitar que se inscriban socios que deben matrículas y
 --no permitir que se eliminen las inscripciones de socios deudores.
 --El trigger se define para ambos eventos en la misma sentencia de creación.

create trigger dis_inscriptos_insert_delete
  on inscritos
  for insert,delete
  as
   if exists (select *from inserted join morosos
              on morosos.documento=inserted.documento)
   begin
     raiserror('El socio es moroso, no puede inscribirse en otro curso', 16, 1)
     rollback transaction
   end
   else
     if exists (select *from deleted join morosos
                on morosos.documento=deleted.documento)
     begin
      raiserror('El socio debe matriculas, no puede borrarse su inscripcion', 16, 1)
       rollback transaction
     end
     else
      if (select matricula from inserted)='n'
       insert into morosos select documento from inserted
```

El trigger controla:

- si se intenta ingresar una inscripción de un socio moroso, se deshace la transacción;
- si se intenta eliminar una inscripción de un socio que está en "morosos", se deshace la transacción;
- si se ingresa una nueva inscripción y no se paga la matrícula, dicho socio se ingresa a la tabla "morosos".

Ingresamos una inscripción de un socio no deudor con matrícula paga:

```
insert into inscritos values('25555555','tenis','s');
```

El disparador se activa ante el "insert" y permite la transacción.

Ingresamos una inscripción de un socio no deudor con matrícula 'n':

```
insert into inscritos values('25555555','natacion','n')
```

El disparador se activa ante el "insert", permite la transacción y agrega al socio en la tabla "morosos".
Verifiquémoslo consultando las tablas correspondientes:

```
select *from inscritos
 select *from morosos
```

Ingresamos una inscripción de un socio deudor:

```
insert into inscritos values('25555555','basquet','s')
```

El disparador se activa ante el "insert" y deshace la transacción porque encuentra su documento en la tabla "morosos".

Eliminamos una inscripción de un socio no deudor:

```
delete inscritos where numero=4
```

El disparador se activa ante la sentencia "delete" y permite la transacción. Verificamos que la inscripción nº 4 fue eliminada de "inscriptos":

```
select *from inscritos;
```

Intentamos eliminar una inscripción de un socio deudor:

```
delete inscritos where numero=6;
```

El disparador se activa ante el "delete" y deshace la transacción porque encuentra su documento en "morosos".

III. Actividad Complementaria

I. Crear la base de datos Practica
Crear las siguientes tablas:

```
CREATE TABLE EMPLEADOS
(ID INT NOT NULL IDENTITY,
DOCUMENTO VARCHAR(30) NOT NULL,
APELLIDO VARCHAR(30) NOT NULL,
NOMBRE VARCHAR(30) NOT NULL,
SECCION VARCHAR(20) NOT NULL,
SUELDO FLOAT NULL,
FECHAINGRESO DATETIME NULL)
GO

CREATE TABLE CONTROLES
(USUARIO VARCHAR(5),
FECHA DATETIME)
GO
```

```
CREATE TABLE COPIAEMPLEADOS
(ID INT NOT NULL,
DOCUMENTO VARCHAR(30) NOT NULL,
APELLIDO VARCHAR(30) NOT NULL,
NOMBRE VARCHAR(30) NOT NULL,
SECCION VARCHAR(20) NOT NULL,
SUELDO FLOAT NULL,
FECHAINGRESO DATETIME NULL)
GO


CREATE TABLE PRODUCTOS
(ID_PRODUCTO CHAR(8) PRIMARY KEY NOT NULL,
NOMBREPRODUCTO VARCHAR(25) NOT NULL,
EXISTENCIA INT NULL,
PRECISO DECIMAL(10,2) NOT NULL,
PRECIOVENTA DECIMAL (10,2))
GO

CREATE TABLE PEDIDO
(ID_PEDIDO INT IDENTITY,
ID_PRODUCTO CHAR(8) NOT NULL,
CANTIDAD_PEDIDO INT
CONSTRAINT PK_ID_PRODUCTO FOREIGN KEY (ID_PRODUCTO)REFERENCES PRODUCTOS(ID_PRODUCTO))
GO
```

1. CREAR UN TRIGGER QUE REGISTRA EL USUARIO Y LA FECHA EN LA QUE SE REALIZA UN INSERT EN LA TABLA EMPLEADOS Y LOS ALMACENA EN LA TABLA CONTROLES
2. CREAR UN TRIGGER QUE DESCUENTA LA EXISTENCIA DE LA TABLA PRODUCTOS SEGUN EL PEDIDO

II. Usar la base de datos Northwind

1. CREAR UN TRIGGER QUE PERMITA ELIMINAR UN SOLO REGISTRO DE LA TABLA ORDER DETAILS
2. CREAR UN TRIGGER QUE CUANDO SE INSERTE UN REGISTRO EN LA TABLA ORDER DETAILS, LA CANTIDAD PEDIDA SE DEBE RESTAR EN EL CAMPO UNITSINSTOCK DE LA TABLA PRODUCTS
3. CREAR UN TRIGGER QUE CUANDO SE BORRE UN DETALLE DE LA ORDEN, EL PRODUCTO QUE ESTABA PEDIDO EN LA ORDEN SE ACTUALIZARA A DESCONTINUADO EN LA TABLA PRODUCTS.
4. CREAR UN TRIGGER QUE CUANDO SE QUIERA ACTUALIZAR EL NOMBRE DE LA COMPANIA DEL PROVEEDOR, NO DEBE PERMITIRLO
5. CREAR UNA TABLA HISTORICO QUE VAYA GUARDANDO EL NOMBRE DE LA COMPANIA (ACTUAL NOMBRE Y NUEVO NOMBRE) DE LOS PROVEEDORES. TABLA: HISTORICO (ID, ACTUAL, NUEVO) EL ID ES IDENTITY.

III. Crear las siguientes tablas:

```
create table sucursales(
codigo int identity,
domicilio varchar(30),
constraint PK_sucursales primary key (codigo)
);

create table empleados(
documento char(8) not null,
nombre varchar(30),
domicilio varchar(30),
```

```
  sucursal int not null,
  constraint PK_empleados primary key (documento),
  constraint FK_empleados_sucursal foreign key(sucursal)
   references sucursales(codigo)
 )

-- Ingrese algunos registros en las dos tablas:
 insert into sucursales values ('Colon 123');
 insert into sucursales values ('Sucre 234');
 insert into sucursales values ('Rivadavia 345');

 insert into empleados values ('22222222','Ana Acosta','Avellaneda 1258',1);
 insert into empleados values ('23333333','Betina Bustos','Bulnes 345',2);
 insert into empleados values ('24444444','Carlos Caseres','Caseros 948',3);
 insert into empleados values ('25555555','Fabian Fuentes','Francia 845',1);
 insert into empleados values ('26666666','Gustavo Garcia','Guemes 587',2);
 insert into empleados values ('27777777','Maria Morales','Maipu 643',3);
```

1. CREAR UN TRIGGER DE INSERCIÓN, ELIMINACIÓN Y ACTUALIZACIÓN QUE NO PERMITA MODIFICACIONES EN LA TABLA "EMPLEADOS" SI TALES MODIFICACIONES AFECTAN A EMPLEADOS DE LA SUCURSAL DE 1.

3. Instead Of

Los activadores INSTEAD OF se activan en lugar del evento activador (eventos INSERT, UPDATE o DELETE). En general, los activadores INSTEAD OF generalmente se usan para actualizar correctamente las vistas que se basan en varias tablas.

Usando los triggers Instead Of, se puede agregar lógica que permita ejecutar operaciones inserts, deletes o updates a cualquier clase de Vista SQL.

Instead Of Insert

```
--SQL Script to create tblEmployee table:
CREATE TABLE tblEmployee
(
 Id int Primary Key,
 Name nvarchar(30),
 Gender nvarchar(10),
 DepartmentId int
)

--SQL Script to create tblDepartment table
CREATE TABLE tblDepartment
(
 DeptId int Primary Key,
 DeptName nvarchar(20)
)

--Insert data into tblDepartment table
Insert into tblDepartment values (1,'IT')
Insert into tblDepartment values (2,'Payroll')
```

```
Insert into tblDepartment values (3,'HR')
Insert into tblDepartment values (4,'Admin')

--Insert data into tblEmployee table
Insert into tblEmployee values (1,'John', 'Male', 3)
Insert into tblEmployee values (2,'Mike', 'Male', 2)
Insert into tblEmployee values (3,'Pam', 'Female', 1)
Insert into tblEmployee values (4,'Todd', 'Male', 4)
Insert into tblEmployee values (5,'Sara', 'Female', 1)
Insert into tblEmployee values (6,'Ben', 'Male', 3)
```

Como ahora tenemos las tablas requeridas, vamos a crear una vista basada en estas tablas. La vista debe devolver las columnas Id. De empleado, Nombre, Género y Nombre del departamento. Por lo tanto, la vista está obviamente basada en varias tablas.

```
--creamos la vista
Create view vWEmployeeDetails
as
Select Id, Name, Gender, DeptName
from tblEmployee
join tblDepartment
on tblEmployee.DepartmentId = tblDepartment.DeptId
go
```

Cuando ejecute, select * from vWEmployeeDetails, los datos de la vista, deben ser como se muestra a continuación:

```
Select * from vWEmployeeDetails
```

Id	Name	Gender	DeptName
1	John	Male	HR
2	Mike	Male	Payroll
3	Pam	Female	IT
4	Todd	Male	Admin
5	Sara	Female	IT
6	Ben	Male	HR

Ahora, intentemos insertar una fila en la vista, vWEmployeeDetails, ejecutando la siguiente consulta. En este punto, se generará un error que indica que "Ver o función vWEmployeeDetails no se puede actualizar porque la modificación afecta a varias tablas base".

Por lo tanto, al insertar una fila en una vista basada en tablas de varios tipos, se genera un error de forma predeterminada. Ahora, entendamos, en qué medida INSTEAD OF TRIGGERS puede ayudarnos en esta situación. Como estamos recibiendo un error, cuando intentamos insertar una fila en la vista, vamos a crear un activador INSTEAD OF INSERT en la vista vWEmployeeDetails.

Ejemplo 1

```
--Script para crear INSTEAD OF INSERT trigger:
```

```
Create trigger tr_vWEmployeeDetails_InsteadOfInsert
on vWEmployeeDetails
Instead Of Insert
as
Begin
 Declare @DeptId int

 --Check if there is a valid DepartmentId
 --for the given DepartmentName
 Select @DeptId = DeptId
 from tblDepartment
 join inserted
 on inserted.DeptName = tblDepartment.DeptName

 --If DepartmentId is null throw an error
 --and stop processing
 if(@DeptId is null)
 Begin
  Raiserror('Invalid Department Name. Statement terminated', 16, 1)
  return
 End

 --Finally insert into tblEmployee table
 Insert into tblEmployee(Id, Name, Gender, DepartmentId)
 Select Id, Name, Gender, @DeptId
 from inserted
End

--Vamos a ejecutar la consulta de insercion a la vista:
Insert into vWEmployeeDetails values(7, 'Valarie', 'Female', 'IT')

--ver las tablas
select * from tblEmployee
select * from vWEmployeeDetails
```

The instead of trigger inserta correctamente, el registro en la tabla tblEmployee. Como estamos insertando una fila, la tabla insertada (tabla inserted) contiene la fila recién agregada, donde la tabla eliminada (tabal deleted)estará vacía.

En el activador, usamos la función Raiserror () para generar un error personalizado, cuando el nombre de departamento proporcionado en la consulta de inserción no existe. Estamos pasando 3 parámetros al método Raiserror (). El primer parámetro es el mensaje de error, el segundo parámetro es el nivel de gravedad. El nivel de gravedad 16 indica errores generales que pueden ser corregidos por el usuario. El parámetro final es el estado.

Instead of Update

Un desencadenador INSTEAD OF UPDATE se activa en lugar de un evento de actualización, en una tabla o en una vista. Por ejemplo, digamos que tenemos un activador INSTEAD OF UPDATE en una vista o una tabla, y luego, cuando intenta actualizar una fila en esa vista o tabla, en lugar de la ACTUALIZACIÓN, el activador se

activa automáticamente. Un desencadenador INSTEAD OF UPDATE, son de gran ayuda para actualizar correctamente una vista, que se basa en varias tablas.

```
--SQL Script para crear la tabla tblEmployee:
CREATE TABLE tblEmployee
(
  Id int Primary Key,
  Name nvarchar(30),
  Gender nvarchar(10),
  DepartmentId int
)

--SQL Script para crear la tabla  tblDepartment
CREATE TABLE tblDepartment
(
 DeptId int Primary Key,
 DeptName nvarchar(20)
)

--Insert data into tblDepartment table
Insert into tblDepartment values (1,'IT')
Insert into tblDepartment values (2,'Payroll')
Insert into tblDepartment values (3,'HR')
Insert into tblDepartment values (4,'Admin')

--Insert data into tblEmployee table
Insert into tblEmployee values (1,'John', 'Male', 3)
Insert into tblEmployee values (2,'Mike', 'Male', 2)
Insert into tblEmployee values (3,'Pam', 'Female', 1)
Insert into tblEmployee values (4,'Todd', 'Male', 4)
Insert into tblEmployee values (5,'Sara', 'Female', 1)
Insert into tblEmployee values (6,'Ben', 'Male', 3)
```

Como ahora tenemos las tablas requeridas, vamos a crear una vista basada en estas tablas. La vista debe devolver las columnas Id. De empleado, Nombre, Género y Nombre del departamento. Por lo tanto, la vista obviamente se basa en múltiples tablas.

```
--creamos la vista
Create view vWEmployeeDetails1
as
Select Id, Name, Gender, DeptName
from tblEmployee
join tblDepartment
on tblEmployee.DepartmentId = tblDepartment.DeptId
```

Cuando ejecute, select * from vWEmployeeDetails1, los datos de la vista, deben ser como se muestra a continuación:

```
Select * from vWEmployeeDetails1
```

Id	Name	Gender	DeptName
1	John	Male	HR
2	Mike	Male	Payroll
3	Pam	Female	IT
4	Todd	Male	Admin
5	Sara	Female	IT
6	Ben	Male	HR

Ejemplo 1

```
Create Trigger tr_vWEmployeeDetails_InsteadOfUpdate
on vWEmployeeDetails1
instead of update
as
Begin
 -- if EmployeeId is updated
 if(Update(Id))
 Begin
  Raiserror('Id cannot be changed', 16, 1)
  Return
 End

 -- If DeptName is updated
 if(Update(DeptName))
 Begin
  Declare @DeptId int

  Select @DeptId = DeptId
  from tblDepartment
  join inserted
  on inserted.DeptName = tblDepartment1.DeptName

  if(@DeptId is NULL )
  Begin
   Raiserror('Invalid Department Name', 16, 1)
   Return
  End

  Update tblEmployee set DepartmentId = @DeptId
  from inserted
  join tblEmployee
  on tblEmployee.Id = inserted.id
 End

 -- If gender is updated
```

```
 if(Update(Gender))
 Begin
  Update tblEmployee set Gender = inserted.Gender
  from inserted
  join tblEmployee
  on tblEmployee.Id = inserted.id
 End
 -- If Name is updated
 if(Update(Name))
 Begin
  Update tblEmployee set Name = inserted.Name
  from inserted
  join tblEmployee
  on tblEmployee.Id = inserted.id
 End
End

--Now, let's try to update JOHN's Department to IT.
Update vWEmployeeDetails1
set DeptName = 'IT'
where Id = 1
select * from vWEmployeeDetails1
select * from tblDepartment
select * from tblEmployee
```

La consulta ACTUALIZACIÓN funciona como se esperaba. El activador INSTEAD OF UPDATE, se actualiza correctamente, el ID de Departamento de JOHN a 1, en la tabla tblEmployee.

Ahora, intentemos actualizar Nombre, Género y Nombre de Dept. La consulta ACTUALIZACIÓN, funciona como se esperaba, sin generar el error: "Ver o funcionar vWEmployeeDetails1 no es actualizable porque la modificación afecta a varias tablas base".

```
Update vWEmployeeDetails1
set Name = 'Johny', Gender = 'Female', DeptName = 'IT'
where Id = 1
```

Instead of delete trigger

Un disparador INSTEAD OF DELETE se activa en lugar de un evento DELETE, en una tabla o en una vista. Por ejemplo, digamos que tenemos un desencadenador INSTEAD OF DELETE en una vista o tabla, y luego, cuando intenta actualizar una fila desde esa vista o tabla, en lugar del evento DELETE real, el desencadenador se activa automáticamente. INSTEAD OF DELETE TRIGGERS, se utilizan para eliminar registros de una vista, que se basa en varias tablas.

```
CREATE TABLE tblEmployee
(
 Id int Primary Key,
 Name nvarchar(30),
 Gender nvarchar(10),
 DepartmentId int
)

--SQL Script to create tblDepartment table
```

```
CREATE TABLE tblDepartment
(
 DeptId int Primary Key,
 DeptName nvarchar(20)
)

--Insert data into tblDepartment3 table
Insert into tblDepartment values (1,'IT')
Insert into tblDepartment values (2,'Payroll')
Insert into tblDepartment values (3,'HR')
Insert into tblDepartment values (4,'Admin')



--Insert data into tblEmployee table
Insert into tblEmployee values (1,'John', 'Male', 3)
Insert into tblEmployee values (2,'Mike', 'Male', 2)
Insert into tblEmployee values (3,'Pam', 'Female', 1)
Insert into tblEmployee values (4,'Todd', 'Male', 4)
Insert into tblEmployee values (5,'Sara', 'Female', 1)
Insert into tblEmployee values (6,'Ben', 'Male', 3)
```

Como ahora tenemos las tablas requeridas, vamos a crear una vista basada en estas tablas. La vista debe devolver las columnas Id. De empleado, Nombre, Género y Nombre del departamento. Por lo tanto, la vista está obviamente basada en varias tablas

```
--crear la vista
Create view vWEmployeeDetails2
as
Select Id, Name, Gender, DeptName
from tblEmployee
join tblDepartment
on tblEmployee.DepartmentId = tblDepartment.DeptId
```

Cuando ejecute, select * from vWEmployeeDetails2, los datos de la vista, deben ser como se muestra a continuación

```
select  * from  vWEmployeeDetails2
```

Id	Name	Gender	DeptName
1	John	Male	HR
2	Mike	Male	Payroll
3	Pam	Female	IT
4	Todd	Male	Admin
5	Sara	Female	IT
6	Ben	Male	HR

Tratamos de insertar una fila en la vista y obtuvimos un error que decía: 'Ver o función vWEmployeeDetails no es actualizable porque la modificación afecta a varias tablas base'. A lo largo de las mismas líneas, en la Parte 46, cuando intentamos actualizar una vista que se basa en varias tablas, obtuvimos el mismo error. Para obtener el error, la ACTUALIZACIÓN debe afectar ambas tablas base. Si la actualización afecta solo a una tabla base, no obtenemos el error, pero la ACTUALIZACIÓN no funciona correctamente, si la columna DeptName se actualiza.

Ahora, intentemos eliminar una fila de la vista y obtenemos el mismo error.

```
Delete from vWEmployeeDetails2 where Id = 1
```

Ejemplo 1

```
Create Trigger tr_vWEmployeeDetails_InsteadOfDelete
on vWEmployeeDetails2
instead of delete
as
Begin
 Delete tblEmployee
 from tblEmployee
 join deleted
 on tblEmployee3.Id = deleted.Id

 --Subquery
 --Delete from tblEmployee
 --where Id in (Select Id from deleted)
End
```

Observe que, el activador tr_vWEmployeeDetails_InsteadOfDelete, utiliza la tabla DELETED. La tabla BORRADO contiene todas las filas que intentamos BORRAR de la vista. Por lo tanto, estamos uniendo la tabla DELETED con tblEmployee, para eliminar las filas. También puedes usar sub-consultas para hacer lo mismo. En la mayoría de los casos, los JOIN son más rápidos que los SUB-QUERIEs. Sin embargo, en los casos en que solo necesita un subconjunto de registros de una tabla a la que se une, las consultas secundarias pueden ser más rápidas.

Al ejecutar la siguiente instrucción DELETE, la fila se SUPRIME como se espera de la tabla tblEmployee.

```
Delete from vWEmployeeDetails2 where Id = 1
```

Trigger	INSERTED or DELETED?
Instead of Insert	DELETED table is always empty and the INSERTED table contains the newly inserted data.
Instead of Delete	INSERTED table is always empty and the DELETED table contains the rows deleted
Instead of Update	DELETED table contains OLD data (before update), and inserted table contains NEW data(Updated data)