BASE DE DATOS (INCLUYE TRANSACT SQL)

Ms, Ing. Ricardo Mendoza Rivera
rimenri@hotmail.com .
http://rimenri.blogspot.com
www.premiunnet.com

Sesión 09

EL TRANSACT-SQL

Sesión 09. El Transact-SQL

Este capítulo proporciona un conocimiento integral de las Ordenes SQL y alguna de las herramientas que se pueden usar para programar en SQL Server.

Para ello el SQL Server proporciona el Transact SQL que está basado en el estándar del Lenguaje SQL incorporando una potente funcionalidad propia del producto.

Objetivos

- Describir y utilizar el SQL Server Managment Studio
- Describir los elementos básicos del Transact-SQL.
- Escribir ordenes SELECT básicas y algunas funciones de sistema
- Describir los modos de ejecutar ordenes Transact-SQL

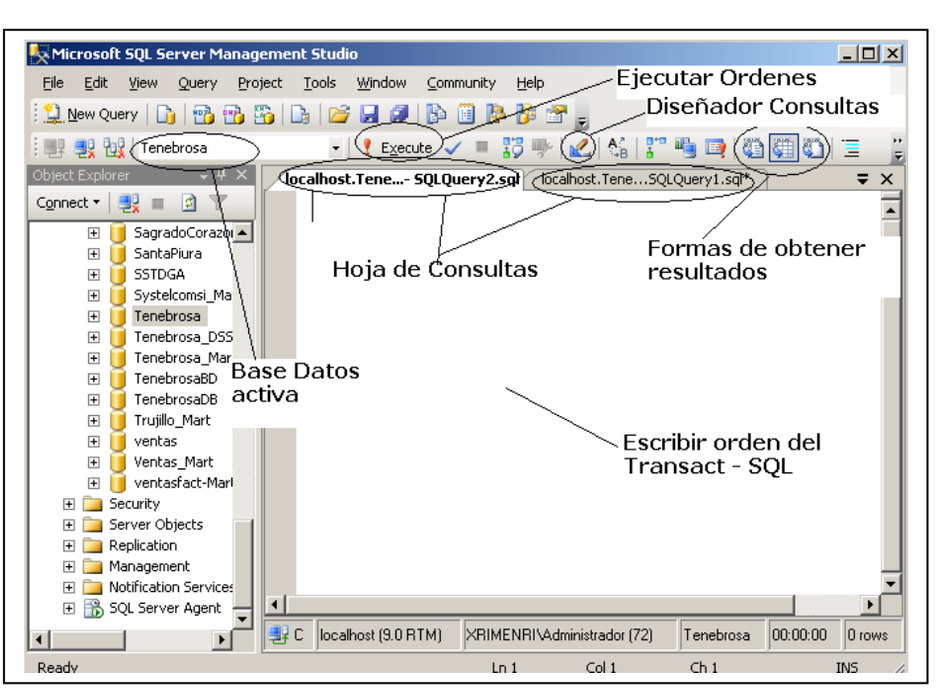
Management Studio

Herramienta general de administración y además que integra la posibilidad del ingreso y ejecución de órdenes del Transact-SQL, batches o scripts iterativamente.
Ventajas:

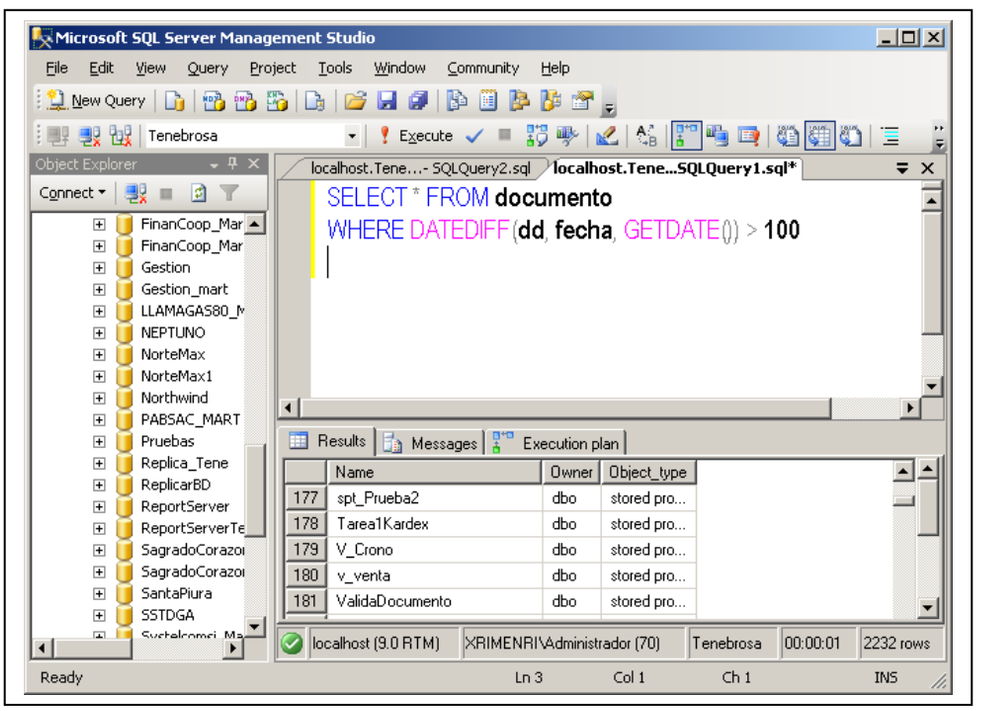
- Editor de Textos free-form. Permite digitar, grabar, reusar y ejecutar órdenes Transact-SQL
- Resultados presentados en un Grid o Texto
- Diagrama gráfico del Plan de Ejecución de una orden SQL. Esta herramienta le permite a los programadores analizar y optimizar sus órdenes indicando los puntos de lenta ejecución.

Formas de Acceso:

- Desde el Explorador de Objetos: Clic derecho sobre la BD deseada: Nueva consulta (New Query)



- Desde el Menu de Opciones: Archivo (File) – Nueva (New) – Consulta Motor de Datos (Database Engine Query)



Ejemplos. Al digitar cada orden, pulsar F5 y estando **Tenebrosa** como base de datos activa:

a. SELECT @@version
b. SELECT * FROM cliente
c. sp_help cliente

Seleccione como BD activa a master
a. SELECT * FROM sys.databases_files
b. SELECT * FROM sys.database_principals

El Lenguaje de Programación Transact-SQL

Está basado en el estándar ANSI SQL-92 ISO y comprende todo las órdenes permitidas por el estándar, pero añade algunas potentes funcionalidades ha alguna de sus órdenes propias del SQL Server.

Se encuentran clasificados en los sgts. grupos:

- Data Control Language: permisos a los objetos de la BD
- Data Definition Language:crear objetos en la BD
- Data Manipulation Language: consultas y modificación de datos
- Elementos adicionales del lenguaje: variables, operadores, funciones, control del flujo del lenguaje y comentarios.

Data Control Language (Lenguaje de Control de Datos)

Usadas para cambiar permisos asociados con una BD de usuario o un Rol, tenemos GRANT y DENY

Ejemplo:
USE tenebrosa
GRANT SELECT ON producto TO Ruben

ORDEN	Descripción
GRANT	Permite a los usuarios trabajar con datos o ejecutar ciertas ordenes del Transact-SQL
DENY	No permite a los usuarios trabajar con datos o ejecutar ciertas ordenes del Transact-SQL
REVOKE	Remueve los permisos GRANT y DENY, normalmente prevalecen los roles definidos a nivel de BD

Estas opciones fueron vistas en el Capítulo de Manejo de la Seguridad.

Data Definition Language (Lenguaje de Definicion de Datos)

Definen la estructura de la BD creando y manejando BD y objetos de la BD como tablas y stored procedures. Las órdenes comúnmente usadas son:

- CREATE nombre_objeto
- ALTER nombre_objeto
- DROP nombre_objeto

Por defecto sólo los miembros de los Roles SysAdmin, db_creator, db_owner o db_ddladmin pueden ejecutar este tipo de órdenes.

Ejemplo: Creando una tabla

USE Tenebrosa CREATE TABLE modelo (modelo char(5), descripcion varchar(40))

Nombrando Objetos:

Identificadores Estándar:

- La primera letra debe ser un carácter alfabético
- La sgte puede ser un carácter, número o los símbolos: @, #, _

Identificadores delimitados:

- Cuando el nombre incluye un espacion en blanco
- Cuando se usan palabras reservadas para nombrar los objetos
- Utilizar corchetes [] ó apóstrofes "
- Ejemplos:

  - SELECT * FROM [unidad de medida]
  - SELECT * FROM 'unidad de medida'

Recomendaciones para nombrar objetos:

- Mantener nombres cortos
- Usar nombres descriptivos
- Usar un identificador que distinga el tipo de objeto.
- Ejemplo
  - Stored Procedure : Venta_CalcularMora
  - Vista : v_DetalleVenta

Nombrando a un Objeto en Forma Completa: Se nombra así:

Servidor.BaseDatos.Owner.Objeto

Ejemplo:

SqlServer1.Tenebrosa.dbo.Cliente

Observe que owner ha sido reemplazado por **dbo**, para ello los objetos de la BD deben ser creados por los que tengan rol de SystemAdministrator o con el usuario sa. Si el objeto pertenece a un Schema, incluir el nombre del schema en lugar de dbo (recuerde que dbo es el esquema por defecto)

Nombrando a un Objeto en Forma Particionada

Tenemos:

Servidor.basedatos.owner. objeto
Servidor. basedatos..objeto
Servidor..owner.objeto
Servidor...objeto
basedatos.owner.objeto
basedatos..objeto owner.objeto
objeto

Data Manipulation Language (Lenguaje de Manipulacion de Datos)

Permite trabajar con los datos de la BD. Usando estas órdenes se puede cambiar datos o recuperar información. Estas órdenes son aplicadas en el desarrollo de sistemas de información operacional.

Estas órdenes son:

- SELECT : consultar datos, leer información
- INSERT : añadir registro(s) de información
- UPDATE : modificar datos que existen en las tablas
- DELETE : eliminar registro(s) de información.

Por defecto, sólo los miembros del Rol: SysAdministrator, db_owner, db_datawriter o db_datareader pueden ejecutar alguno de estos comandos.

Ejemplos:

a.
```
SELECT tipodoc, documento, fecha, observacion
FROM documento
```

b.
```
SELECT d.documento, d.producto, p.descripcion
FROM detadoc d INNER JOIN producto p ON d.producto =p.producto
```

Elementos Adicionales del Lenguaje

Variables

Son de dos tipos:

- **Variables Locales**

  Definidas por el usuario son de la sgte forma:

  DECLARE @variable_local <tipo_dato> [<long>]

  Se les asigna un valor mediante la orden SET

  SET @variable_local = <expression>

  Su nombre debe iniciar con el símbolo: @

  Ejemplos:

```
USE TENEBROSA
DECLARE @tasaigv  numeric(9,2)        -- Declarando la variable local
SET @tasaigv = 18.0                    -- Asignando Valor directamente
Otra forma de asignar directamente desde una orden SELECT
DECLARE @tasaigv  numeric(9,2)
SELECT @tasaigv = tasaigv FROM parametro WHERE activo = 1
```

- **Variables globales**

  Propias del SQL Server

  No se les puede asignar ningún valor, sólo pueden ser leídas.

  Inician con el símbolo: @@

  Ejemplos:

```
/* Variable @@rowcount: devuelve el Nro de registros afectados con
la última instrucción SQL */

DECLARE @ NroRegistros int     -- Se declara variable local
SELECT descripción FROM producto WHERE producto = 'PR01'
SET @NroRegistros = @@rowcount   -- @@rowcount variable global

PRINT @NroRegistros
```

  Importante! Cuando las variables son declaradas no tienen ningún valor o sea son NULL (nulas)

Operadores

Son símbolos que permiten ejecutar operaciones matemáticas. Concatenación de cadenas, combinación entre columnas, constantes y variables. Pueden ser combinadas y usadas como condiciones.

Tipo de Operadores

Asignación (=)

Aritméticos : permite ejecutar cálculos con columnas numéricas o constantes. Estos son multiplicación (*), división (/), adición (+), substracción (-) y modulo (%), el resido entero resultado de una división Comparación : permite comparar dos expresiones. Las comparaciones pueden efectuarse entre variables, columnas y expresiones de igual tipo. Entre ellas tenemos: igual que (=), diferente (<>), mayor que (>), mayor igual que (>=), menor o igual que (<=) y menor que (<).

Concatenación de cadenas: permite concatenar dos valores tipo cadena. Se usa el operador (+).

Lógicos: permite unir condiciones de la clausula WHERE, estos son: AND, OR, y NOT.

- Nivel de prioridad de los operadores (de la más alta a la más baja)

Tipo	**Operador**	**Símbolo**
Agrupa miento	Agrupamiento primario	0
Aritmético	Multiplicación	* , / , %
Aritmético	Adición	- , +
Otro	Concatenar cadenas	+
Lógico	NOT	NOT
Lógico	AND	AND
Lógico	OR	OR

Funciones de Sistema:

Son funciones proporcionadas por el Transact-SQL, normalmente utilizan parámetros para devolver valores.

Son de 3 tipos:

Funciones de Agregamiento

Evalúan una serie de valores y retornan un valor simple sumarizado. Las revisaremos con más detalle en el capítulo de sumarización de datos.

Ejemplo:

a. Imaginemos que queremos conocer el importe al que asciende la factura número 100000017

```
SELECT SUM(precunit * cantidad) AS total FROM detadoc  WHERE
documento = '100000017' AND tipodoc = 'F'
```

b. Suponga que desea conocer a cuánto ascenderá el IGV de la factura 100000017

```
DECLARE @monto numeric(9,2), @igv numeric(9,2)
SET @igv = 19

SELECT @monto = SUM(precunit * cantidad)
FROM detadoc  WHERE documento = '100000017' AND tipodoc = 'F'

PRINT 'El Igv es: ' + CONVERT(char(12), @monto* @igv /100.0 )
```

```
El Igv es: 38.700000000
```

Funciones escalares

Operan un simple valor y retornan un simple valor. Pueden ser agrupadas dentro de las siguientes categorías

Tipo de Función	Descripción
Configuración	Retornan información acerca de la configuración actual
Cursor	Información acerca de cursores
Fecha y Hora	Ejecuta una operación de la fecha y hora
Matemáticas	Ejecuta una operación basado en valores de entrada
Metadata	Información acerca de los objetos de la base de datos
Seguridad	Información acerca de usuarios y roles
Cadena	Ejecuta una operación en una cadena
Sistema	Información acerca de valores, objetos y configuración
Estadísticas Sistema	Información estadística acerca del sistema
Texto e imagen	Información acerca de campos textos e imágenes

Algunas funciones escalares más usadas

DATEPART: retorna un valor entero de acuerdo a la frecuencia especificada

Sintaxis: DATEPART(<frecuencia>, <dato_tipo_fecha>) Donde

frecuencia puede tomar la abreviatura de acuerdo al siguiente cuadro:

Frecuencia	Abreviatura
Año	yy
Trimestre	qq
Mes	mm
Día del año	dy
Día	dd, d
Semanda	ww
Día de semana	dw
Hora	hh
Minuto	mi
Segundo	ss
Milisegundo	ms

Ejemplo: imagine que desea conocer los pedidos ocurridos en Julio del año 2000

```
SELECT pedido,  cliente, fecha
FROM pedido WHERE DATEPART(yy,      fecha) = 2000 AND
DATEPART(mm, fecha) = 7
```

pedido	cliente	fecha
900005	CLI3	02/07/2000 00:00
900006	CLI3	02/07/2000 00:00

DATEDIFF : retorna un entero del tiempo transcurrido - <frecuencia> - entre la

<fecha_referencia> y la <fecha_comparativa>.

Sintaxis:
DATEDIFF(<frecuencia>, <fecha_referencia>, <fecha_comparativa>)

Donde:
<frecuencia> : equivale a los valores de la tabla anterior
<fecha_referencia> : es la fecha desde la cual se quiere medir el tiempo transcurrido.
<fecha_comparativa>: es la fecha final sobre la cual se desea conocer cuantos días han transcurrido desde la <fecha_referencia>

Ejemplo: imagine que desea conocer los días transcurridos desde la emisión del documento con respecto a la fecha de hoy.

```
SELECT documento, tipodoc, DATEDIFF(dd, fecha, GETDATE()) AS DiasTranscurridos, fecha
FROM documento
```

Documento	tipodoc	DiasTranscurridos	Fecha
1	A	2929	01/01/2000 00:00
19	B	2929	01/01/2000 10:20
4	F	2897	02/02/2000 00:00
5	F	2868	02/03/2000 00:00
18	B	2837	02/04/2000 00:00
6	F	2837	02/04/2000 00:00

DATEADD : retorna una nueva fecha basado en un un tiempo - <frecuencia> - agregado <incremento> a una <fecha_referencia>.

Sintaxis:
DATEADD(<frecuencia>, <incremento>, <fecha_referencia>)

Donde:
<frecuencia> : equivale a los valores de la tabla anterior de DATEPART
<incremento> : es la cantidad a ser agregada a la <fecha_referencia>
<fecha_referencia> : es la fecha sobre la cual se produce el incremento

Ejemplo: imagine que desea agregar 1 año a los documentos existentes.

```
SELECT documento, tipodoc, fecha , DATEADD(YY, 1, fecha) AS NuevaFecha
FROM documento
```

Documento	TipoDoc	Fecha	NuevaFecha
1	A	01/01/2000	01/01/2001
19	B	01/01/2000	01/01/2001
4	F	02/02/2000	02/02/2001
21	B	07/02/2000	07/02/2001
5	F	02/03/2000	02/03/2001
18	B	02/04/2000	02/04/2001
6	F	02/04/2000	02/04/2001
100000055	F	04/04/2000	04/04/2001
100000034	F	05/05/2000	05/05/2001
100000044	F	06/06/2000	06/06/2001

CONVERT : transforma una expresión a un tipo de dato requerido

Sintaxis: CONVERT (*tipo_dato* [(*long*)] , *expresión* [, formato])

El formato sólo es aplicado a las expresiones tipo Datetime y el valor de formato a ingresar se resume en la sgte tabla:

Valor de Formato	Estándar	Formato Devuelto
0 or 100	Default	mon dd yyyy hh:miAM (or PM)
101	USA	mm/dd/yyyy
102	ANSI	yyyy.mm.dd
103	British/French	dd/mm/yyyy
104	German	dd.mm.yyyy
105	Italian	dd-mm-yyyy
106	-	dd mon yyyy
107	-	Mon dd, yyyy
108	-	hh:mm:ss
9 or **109 (*)**	Default + milliseconds	mon dd yyyy hh:mi:ss:mmmAM (or PM)
110	USA	mm-dd-yyyy
111	JAPAN	yyyy/mm/dd
112	ISO	yyyymmdd
13 or 113 **(**	Europe default + milliseconds	dd mon yyyy hh:mm:ss:mmm(24h)
114	-	hh:mi:ss:mmm(24h)
20 or **120 (**	ODBC canonical	yyyy-mm-dd hh:mi:ss(24h)
21 or 121 **(*)**	ODBC canonical (with milliseconds)	yyyy-mm-dd hh:mi:ss.mmm(24h)
126(***)	ISO8601	yyyy-mm-dd Thh:mm:ss:mmm(no spaces)

Ejemplo:

a. Imagine que desea mostrar el codigo del producto, su descripción y el precio de venta acompañado del mensaje: "Precio es"

```
SELECT  producto, descripcion, 'Precio es=' +
CONVERT(char(10), precVenta) As precio FROM producto
```

producto	descripcion	Precio
PR01	FRIOL GALON	Precio es=135.97
PR02	FRIOL 3/4 LT	Precio es=285.53
PR03	GLORIA 1LT	Precio es=22.68
PR04	GLORIA CAJA 48	Precio es=39.69
PR05	GLORIA CAJA 24	Precio es=17.01
PR06	GLORIA SIX PACK	Precio es=19.28
PR07	PRIMOR 1LT	Precio es=17.01
PR08	FRIOL 1/2 lt	Precio es=407.91
PR09	CRISOL 1 LT	Precio es=45.36
PR10	COMPASS X 48	Precio es=12.96
PR11	COMPASS x 24	Precio es=24.84

Nótese que para poder concatenar las expresiones debe ser tipo cadena, en el caso de **PrecVenta** es numérico, por lo que se ha incluido la función CONVERT()

b. Suponga que desea conocer los documentos emitidos el día , dos de Julio del año 2000. El formato de la fecha dependerá del idioma de instalación, vamos a suponer que la instalación es en Español (dd/mm/yyyy).

```
SELECT Documento, TipoDoc, Pedido, Cliente, Fecha
FROM   documento
WHERE fecha = '02/07/2000'
```

producto	tipodoc	pedido	cliente	fecha
20	B	900005	CLI3	02/07/2000 00:00
21	B	900006	CLI3	02/07/2000 00:00
8	F	900005	CLI3	02/07/2000 00:00
9	F	900006	CLI3	02/07/2000 00:00

c. Como puede observar los datos obtenidos para la fecha ingresada corresponden a las 00:00:00 de ese día.

Importante! Cada vez que no se registra la hora el sistema internamente le pone a la fecha las 00:00:00

Pero que pasaría si la aplicación desearía controlar la hora ? Definitivamente los resultados no serán los esperados. Vamos a cambiar al documento : 21 B e incluirle la hora con la siguiente instrucción:

```
-----   Vamos cambiar el documento con la orden UPDATE
UPDATE documento SET fecha = '02/07/2000 10:20'
WHERE documento = '21' AND tipodoc = 'B'
```

```
-- Volvamos a consultar nuevamente el documento
SELECT Documento, TipoDoc, Pedido, Cliente, Fecha
FROM   documento
WHERE fecha = '02/07/2000'
```

producto	tipodoc	pedido	cliente	fecha
20	B	900005	CLI3	02/07/2000 00:00
8	F	900005	CLI3	02/07/2000 00:00
9	F	900006	CLI3	02/07/2000 00:00

Observe que no aparece el documento, definitivamente **02/07/2000 00:00** es diferente que **02/07/2000 10:20.** Cuando se maneje la hora como parte de la fecha hay que tener presente en cambiar nuestra condición, una alternativa podría ser aplicar : fecha >= '02/07/2000' AND fecha < '03/07/2000', observe que avanzamos al siguiente día.

```
SELECT Documento, TipoDoc, Proveedor, Pedido, Cliente, Fecha
FROM   documento
WHERE fecha >=  '02/07/2000'  AND fecha < '03/07/2000'
```

Ahora analice los resultados.

Elementos del Control de flujo

Se agrupan en 2 tipos:

Nivel de Orden

BEGIN... END

Si alguno de los elementos de control, que veremos a continuación, incluyen una serie de órdenes Transact-SQL es necesario que estén delimitadas por BEGIN.. END

```
BEGIN
        <órdenes SQL>
END
```

WHILE

Establece una condición que mientras se es cumplida permitirá la ejecución del bloque de instrucciones. Puede ser controlada mediante BREAK: lo cual finaliza la ejecución del bloque y CONTINUE que reinicia la ejecución del bloque.

Syntax

WHILE *Expresión Condicional*

```
BEGIN

        < ordenes SQL >

        [ BREAK ] [
        CONTINUE ]

END
```

IF...ELSE:

Permite evaluar una condición si es verdadera se ejecuta el bloque de órdenes. Opcionalmente se puede incluir ELSE para ejecutar órdenes en caso la condición sea Falsa

```
IF  <condición>
```

```
        BEGIN
            <órdenes SQL>
        END
    [ ELSE BEGIN
                <órdenes SQL> END
        ]
```

Comentarios:

Sirven para documentar un Script
Existen 2 formas

```
/*    el comentario alcanza varias líneas,
    todo lo que está encerrado es ignorado en la ejecución del Script   */
```

< orden SQL > — Sirve para documentar en la misma línea

Nivel de Registro

CASE

Permite evaluar una lista de condiciones y retorna el valor de la condición verdadera.

```
CASE <valor_a_evaluar>
   WHEN <expression_condición> THEN <Resultados> [
      ...n ]
   [
      ELSE <resultado_tomado_sino_cumplen_condiciones_aanteriores>
   ]
END
```

Modos de Ejecutar una Orden Transact-SQL

Se puede ejecutar de varios modos:

- Ejecutar individualmente o en batches
- Almacenados en scripts y ejecutados desde un cualquier cliente
- Pueden ser ejecutados desde un Stored Procedure, Trigger.

A continuación veremos los mecanismos para ejecutar consultas y transacciones

Usando Batches

Definición

Un batch es un conjunto de órdenes Transact-SQL que pueden ser incluídas en SQL Server y ejecutadas como un grupo. Pueden formar parte de un Script. Un script puede incluir uno o más Batches.

Delimitador

El delimitador de finalización de un Batch es la orden GO

Cómo son procesados los Batches

SQL Server optimiza, compila y ejecuta las órdenes en un batch; sin embargo las órdenes no se ejecutan como una unidad de trabajo (como si lo hace una

transacción).
Si existe una error en el batch, ninguna de las órdenes son ejecutadas. La ejecución continua con el sgte. batch.

Reglas Para Batches

Ciertas reglas no pueden ser ejecutadas en conjunto, así

- No puede combinar CREATE DEFAULTM CREATE PROCEDURE, CREATE RULE, CREATE TRIGGER Y CREATE VIEW

Ejemplos de Batches

a. CREATE DATABASE
   CREATE TABLE
   GO
   CREATE VIEW1
   GO
   CREATE PROCEDURE...

b. CREATE DATABASE
   CREATE TABLE
   CREATE TRIGGER
   CREATE TRIGGER

   Habría un error!

c. CREATE DATABASE
   CREATE TABLE
   GO
   CREATE TRIGGER
   GO
   CREATE TRIGGER

Usando Scripts

Definición

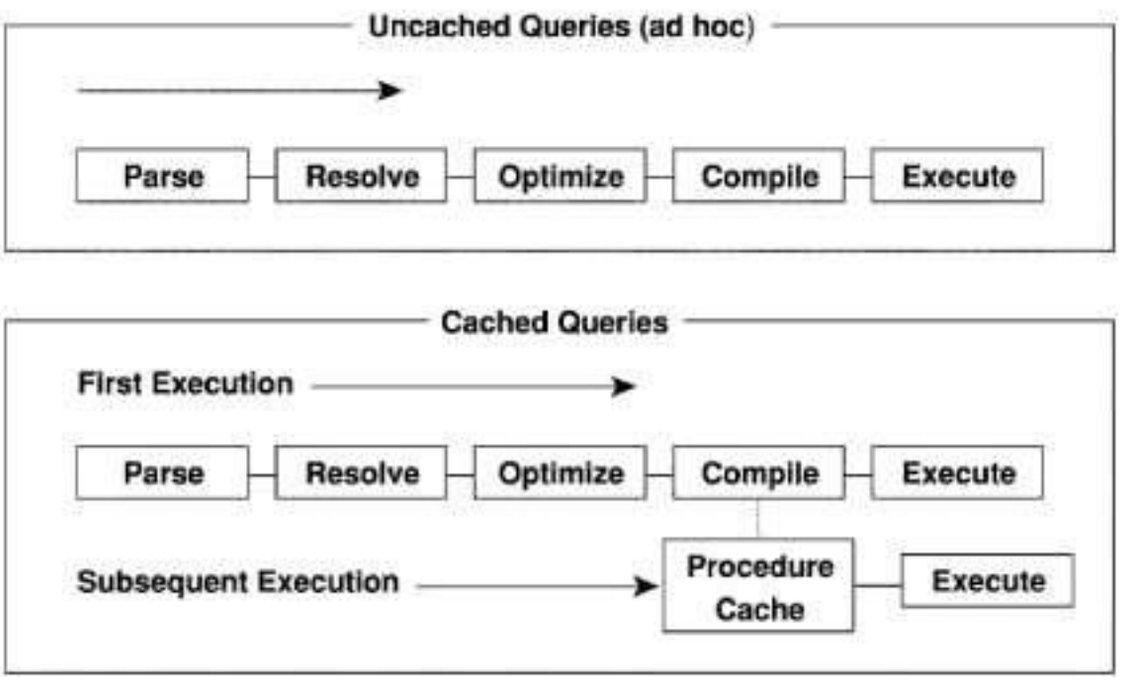
Constituye una de las formas más comunes para ejecutar órdenes Transact-SQL. Un Script es una o más órdenes Transact-SQL que son almacenadas como un archivo.

Se reconocen porque tienen la extensión .SQL y pueden ser ejecutados las veces que sea necesario.

Cómo es procesado una Consulta

Cuando SQL Server recibe una consulta un número de pasos son requeridos para procesar la consulta antes, ver figura sgte.

Paso	**Descripción**
Parse	Cheque sintáxis
Resolve	Valida los nombres de los objetos y los permisos
Optimiza	Determina los índices usados y la estrategia de los JOINs
Compila	Traslada la consulta dentro de una forma ejecutable
Execute	Procesa requerimiento



Construcción Dinámica de Ordenes

Definición

Una orden dinámica es construída mientras otro script es ejecutado. Las cadenas literales son concatenadas con los valores de una o más variables, el resultado de la cadena es una orden Transact-SQL.

Ejecución

Se tienen las sgts. posibilidades

- Usar el stored procedure: sp_executesql
  SP_EXECUTESQL [@orden_sql=]
- EXECUTE y la serie concatenada
  EXECUTE (@orden_sql)

Sesión 10

RECUPERANDO DATOS

La orden SELECT

- ***SELECT : Lista Columnas Deseadas → Que mostrar?***
- ***FROM : Especifica el Nombre de la Tabla → Donde mostrar?***
- ***WHERE : Especifica Registros a Mostrar → Quienes mostrar?***

```
SELECT [ALL / DISTINCT] <lista de columnas>
 FROM {<tabla>} [,n]
WHERE <condicion de busqueda>
```

Especificando Columnas

```
SELECT cliente, Nombre, Saldo, Credito, TipoCliente
FROM cliente
GO
```

Ejemplo 1

Cliente	Nombre	Zona	Saldo	Credito	TipoCliente
CL10	LOS COCOS	Z2	0	1	NULL
CL11	COMERCIAL PEPITA	Z1	2500	0	A
CL12	COMER SAN LORENZO	Z2	4855	0	B
CL13	SAN ANTONIO COM	Z1	500	0	C
CL14	COQUITOS SAC	Z2	2092	0	A
CL15	LOS PARQUES 1254	Z1	0	0	C
CL17	LAS LIRAS	Z1	0	0	C

Usando la Clausula WHERE para Especificar Registros

Ejemplo 2

```
SELECT cliente, Nombre, Saldo, Credito,
TipoCliente
FROM cliente
WHERE  zona  =  'Z1'
GO
```

Cliente	Nombre	Zona	Saldo	Credito	TipoCliente
CLI1	COMERCIAL PEPITA	Z1	2500	0	A
CLI3	SAN ANTONIO COM	Z1	500	0	C
CLI5	LOS PARQUES 1254	Z1	0	0	C
CLI7	LAS LIRAS	Z1	0	0	C

Filtrando Datos

- Usando Operadores de Comparacion
- Usando Comparaciones de Cadena
- Usando Operadores Logicos
- Recuperando de un Rango de Valores
- Usando Lista de Valores como Criterio de Busqueda
- Recuperando Valores Desconocidos

Usando Comparaciones de Cadena

Ejemplo 3

```
SELECT
Descripcion, Stockac, Producto, Marca, Peso
FROM producto
WHERE Marca = 'M1'
```

Descripcion	Stock	Producto	Marca	Peso
aceites	100	PR02	M1	1
FRIOL 1 LT	2764	PR01	M1	1
FRIOL 1/2 lt	2867	PR08	M1	0.5
FRIOL 5 LT	2139	PR13	M1	5

Usando Comparaciones de Cadena Usando Clausula OVER con Row_Number

Ejemplo 4

```
SELECT Row_Number() over (Order By
Descripcion) As Item,
Descripcion, Stockac, Producto, Marca, Peso
FROM producto
WHERE Marca = 'M1'
```

Item	Descripcion	Stock	Producto	Marca	Peso
1	aceites	100	PR02	M1	1
2	FRIOL 1 LT	2764	PR01	M1	1
3	FRIOL 1/2 lt	2867	PR08	M1	0.5
4	FRIOL 5 LT	2139	PR13	M1	5

Usando Operadores de Comparacion

Ejemplo 5

```
SELECT
Nombre, cliente, Zona, Saldo, Credito
FROM Cliente
WHERE NOmbre LIKE '%COM%'
```

Nombre	Cliente	Zona	Saldo
COMERCIAL PEPITA	CLI1	Z1	2500
SAN ANTONIO COM	CLI3	Z1	500
COMER SAN LORENZO	CLI2	Z2	4855

Usando Operadores de Comparacion incluyendo la Clausula OVER con Rank() y Row_Number()

Ejemplo 6

```
SELECT Row_Number() over (Order By Zona) As Item,
Nombre, cliente, Zona, Saldo, Credito,
Rank() Over (Order By Zona) as Ranking
FROM Cliente
WHERE NOmbre LIKE '%COM%'
```

Item	Descripcion	Stock	Zona	Saldo	Ranking
1	COMERCIAL PEPITA	CLI1	Z1	2500	1
2	SAN ANTONIO COM	CLI3	Z1	500	1
3	COMER SAN LORENZO	CLI2	Z2	4855	3

Recuperando Rango de Valores

Ejemplo 7

```
SELECT Nombre, cliente, Zona, Saldo
FROM Cliente
WHERE Saldo BETWEEN 100 AND 5000
```

Nombre	Cliente	Zona	Saldo
COMERCIAL PEPITA	CLI1	Z1	2500
SAN ANTONIO COM	CLI3	Z1	500
COQUITOS SAC	CLI4	Z2	2092
COMER SAN LORENZO	CLI2	Z2	4855
PREMIUN.NET	CLXY	Z3	1340

Recuperando Rango de Valores incluyen OVER con: Dense_Rank(), Rank() y Row_Number()

Ejemplo 8

```
SELECT Row_Number() over (Order By Zona) As Item,
Nombre, cliente, Zona, Saldo,
Rank() Over (Order By Zona) as Ranking,
Dense_Rank() OVER (ORDER BY Zona) AS DenseRank
FROM Cliente
WHERE Saldo BETWEEN 100 AND 5000
```

Item	Descripcion	Stock	Producto	Marca	Ranking	R_Densld
1	COMERCIAL PEPITA	CLI1	Z1	2500	1	1
2	SAN ANTONIO COM	CLI3	Z1	500	1	1
3	COQUITOS SAC	CLI4	Z2	2092	3	2
4	COMER SAN LORENZO	CLI2	Z2	4855	3	2
5	PREMIUN.NET	CLXY	Z3	1340	5	3

Recuperando Rango de Valores incluyendo OVER con: Ntile, Dense_Rank(), Rank() y Row_Number()

Ejemplo 9

```
SELECT Row_Number() over (Order By Zona) As Item,
Nombre, cliente, Zona, Saldo,
Rank() Over (Order By Zona) as Ranking,
Dense_Rank() OVER (ORDER BY Zona) AS DenseRank,
NTile(2) OVER (ORDER BY Zona) AS Grupo2,
NTile(3) OVER (ORDER BY Zona) AS Grupo3
FROM Cliente
WHERE Saldo BETWEEN 100 AND 5000
```

Item	Descripcion	Stock	Producto	Marca	Ranking	DenseRank	Grupo2	Grupo3
1	COMERCIAL PEPITA	CLI1	Z1	2500	1	1	1	1
2	SAN ANTONIO COM	CLI3	Z1	500	1	1	1	1
3	COQUITOS SAC	CLI4	Z2	2092	3	2	1	2
4	COMER SAN LORENZO	CLI2	Z2	4855	3	2	2	2
5	PREMIUN.NET	CLXY	Z3	1340	5	3	2	3

Usando una Lista de Valores como Criterio de Busqueda

Ejemplo 10

```
SELECT cliente, Nombre, Zona, Saldo, Credito, TipoClie
FROM cliente where TipoCliente IN ('A', 'B')
```

Cliente	Nombre	Zona	Saldo	Credito	TipoCliente
CLI1	COMERCIAL PEPITA	Z1	2500	0	A
CLI2	COMER SAN LORENZO	Z2	4855	0	B
CLI4	COQUITOS SAC	Z2	2092	0	A

Uso de Common Table Expresions (CTE)

Ejemplo 11

```
with Tablita(zona, cliente, saldo) as
(
select zona, cliente, saldo
from cliente  )
select * from Tablita
```

Zona	Cliente	Saldo
Z2	CL10	0
Z1	CLI1	2500
Z2	CLI2	4855
Z1	CLI3	500
Z2	CLI4	2092
Z1	CLI5	0
Z1	CLI7	0
Z3	CLXY	1340

Uso de PIVOT con tablas temporales

Ejemplo 12

```
select d.personal, c.zona, d.pagado into #datos
from cliente c INNER JOIN documento d ON d.cliente= c.cliente
```

```
select personal, [z1], [z2]   from #datos
pivot (sum(pagado) FOR zona in ([Z1], [Z2])) as pvt
```

Personal	Z1	Z2
P1	21417	9275
P2	404	7534
P3	3930	1726
P4	1100	9860
P5	NULL	15603
P6	NULL	4438
ZZ	NULL	567

Uso de PIVOT con CTE

Ejemplo 13

```
WITH Tablita (Anual, Trimestral, Cobrado) as
( SELECT datepart(yy, fecha) , datepart(QQ, FECHA),
pagado FROM documento)

SELECT * FROM Tablita
PIVOT (SUM(Cobrado) FOR Trimestral in ([1], [2],[3],
[4])) as pvt
```

Anual	1	2	3	4
2000	5	1935	908	904
2002	338	2581	2	2439
2003	33259	6401	20652	6432

Uso de UNPIVOT

Ejemplo 14

```
select Cliente, zona, cantidad
 from TablaPivote
UNPIVOT (Cantidad  FOR Zona IN ([Z1],[Z2]))  as uPiv
```

Zona	Z1	Z2
CL10	NULL	0
CLI1	2500	NULL
CLI2	NULL	4855
CLI3	500	NULL
CLI4	NULL	2092
CLI5	0	NULL
CLI7	0	NULL
CLXY	NULL	NULL

Cliente	Zona	Saldo
CL10	Z2	0
CLI1	Z1	2500
CLI2	Z2	4855
CLI3	Z1	500
CLI4	Z2	2092
CLI5	Z1	0
CLI7	Z1	0

Recuperando de una Lista de Valores Desconocida

```
SELECT cliente, Nombre, Zona, Saldo, Credito
TipoCliente
 FROM cliente where TipoCliente IS NULL
```

Ejemplo 15

Cliente	Nombre	Zona	Saldo	Credito	TipoCliente
CL10	LOS COCOS	Z2	0	1	NULL
CLXY	PREMIUN.NET	Z3	1340	1	NULL

Recuperando de una Lista de Valores Desconocida

```
SELECT cliente, Nombre, Zona, Saldo, Credito
TipoCliente
 FROM cliente where TipoCliente IS NOT NULL
```

Ejemplo 16

Cliente	Nombre	Zona	Saldo	Credito	TipoCliente
CL10	LOS COCOS	Z2	0	1	A
CLXY	PREMIUN.NET	Z3	1340	1	B

Recuperando con Mensajes Condicionados

```
SELECT Mensaje = CASE WHEN Saldo > 2400 THEN 'MuyMoroso'
WHEN Saldo > 1800 THEN 'Morosos'
            ELSE 'Regulares' END, cliente, nombre, Saldo
FROM cliente
```

Ejemplo 17

Mensaje	Cliente	Nombre	Saldo
Regulares	CL10	LOS COCOS	0
Muy Moroso	CLI1	COMERCIAL PEPITA	2500
Muy Moroso	CLI2	COMER SAN LORENZO	4855
Regulares	CLI3	SAN ANTONIO COM	768.83
Morosos	CLI4	COQUITOS SAC	2092
Regulares	CLI5	LOS PARQUES 1254	0
Regulares	CLI7	LAS LIRAS	0
Regulares	CLXY	PREMIUN.NET	1340
Regulares	CXX1	LOS COCODRILOS	NULL

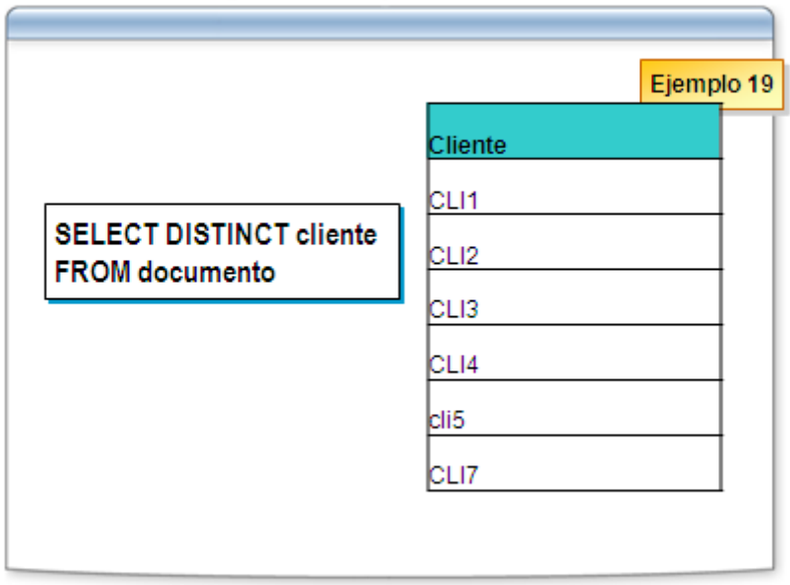
Ordenando Datos

```
USE northwind
SELECT productid, productname, categoryid,
unitprice
FROM products
ORDER BY categoryid, unitprice DESC
```

Ejemplo 18

productid	*productname*	*categoryid*	*unitprice*
38	Cote de Blaye	1	263.5000
43	Ipoh Coffee	1	46.0000
2	Chang	1	19.0000
...	...	...	...
63	Vegie-spread	2	43.9000
8	Northwoods Cranberry Sauce	2	40.0000
61	Sirop d'érable	2	28.5000
...	...	...	...

Eliminando Registros Duplicados



Lab 10: Recuperando Datos

Objetivos

- Cumplir requerimientos utilizando la orden SELECT
- Implementar requerimientos en Tenebrosa usando una sola tabla

1. Se quiere conocer a todo los clientes cuyo nombre empieza con RODRIGUEZ.
2. La administración se encuentra depurando la lista de clientes que tienen crédito, para ello necesita conocer previamente quienes son los que actualmente tienen crédito autorizado.
3. La empresa necesita reponer sus stocks, el Jefe de compras le solicita que productos se van a adquirir mostrando código, descripción, stock actual y stock mínimo.
4. Identificar todos los vendedores cuyo sueldo básico sea superior a 400 y se encuentren activos.
5. Se necesita conocer los pedidos efectuados por el cliente con código CL10 emitidas en Enero, Junio y Diciembre.
6. Se desea tener una calificación del detalle de ventas (detadoc) en función a la tabla siguiente

Cantidad* PrecUnit	Mensaje
> 5000	Alto
2000- 5000.00	Regular
0 - 1999.99	Medio
Si es NULL	Bajo

7. Muestre las ventas entre Enero y Febrero del 2006
8. Se desea conocer los montos vendidos (detadoc) del producto PR01 con montos superiores (Cantidad *precunit) entre 2000 y 10000.

Sesión 11

UNIENDO MULTIPLES TABLAS

Sesión 11. Uniendo Múltiples Tablas

Dentro de una empresa existen una serie de políticas cambiantes que algunas veces no son contempladas como opciones de un sistema; sin embargo existen los datos para satisfacer estos requerimientos de información, que muchas veces se encuentran en más de una tabla.
Dentro de SQL Server podemos utilizar los JOINS como forma de poder compartir información entre las tablas, que son una extensión de la orden SELECT como veremos a continuación.

Objetivos

- Combinar datos desde dos tablas usando JOIN
- Describir y usar INNER JOIN, OUTER JOIN y CROSS JOIN
- Combinando datos desde más de dos tablas
- Utilizando JOIN para generar una consulta a una sola tabla.
- Creando tablas usando SELECT INTO

Introducción a los JOINS

Los JOIN producen un simple resultado que incorpora registros y columnas desde dos o más tablas.

Sintáxis:

SELECT *{nombre campo} [, ...n]* FROM
{tabla_o_vista}
[
[INNER | {{LEFT | RIGHT | FULL} [OUTER] }]
JOIN
tabla o vista ON *condiciones de búsqueda]*
] [-n] ~
[WHERE *condiciones_de_búsqueda]* ORDER BY
{nombre campo} [, ...n] }

Performance

Cuando use JOIN en más de dos tablas, y si estas son largas -con una cantidad considerable de registros-, debería de analizar las consultas utilizando el Plan de Ejecución o las ordenes SET SHOWPLAN_TEXT, SET_SHOWPLAN_ALL estudiadas en el manejo de índices. Tener presente el manejo de índices como una forma de mejorar la performance de las consultas.

- Ejemplo 1 (con alias)

```
USE Tenebrosa
        SELECT documento, tipodoc, nombre, cliente.cliente, personal
        FROM documento  INNER JOIN cliente
                ON documento.cliente = cliente.cliente
```

- Ejemplo 2 (sin alias)

```
USE Tenebrosa
   SELECT d.documento, d.tipodoc, c.nombre, c.cliente, d.personal
   FROM documento d INNER JOIN cliente c
                ON d.cliente = c.cliente
```

El alias es el uso alternativo de un "nombre" que representa a la tabla. Una vez definido el sistema solo reconocerá al alias especificado en cualquier punto de la orden SQL respectiva.

- **SELECT Especifica Columnas desde Múltiples Tablas**
  - JOIN especifica qué tablas y cómo estarán unidas
  - ON especifica condición de unión

- **Para Consultar 2 o Más Tablas: buscar datos comunes en las 2 tablas**
  - Usar primary y foreign key como condición de unión
  - Usar columnas para especificar tablas a unir

Imagine el siguiente requerimiento:
"Conocer las ventas especificando: el documento, tipodoc, el nombre del cliente, el código del cliente y el código del personal"

Observe los datos en CLIENTE y DOCUMENTO

CLIENTE

Cliente	Nombre
CL10	LOS COCOS
CLI1	COMERCIAL PEPITA
CLI2	COMER SAN LORENZO
CLI3	SAN ANTONIO COM
CLI4	COQUITOS SAC
CLI5	LOS PARQUES 1254
CLI7	LAS LIRAS

DOCUMENTO

Document	Tipo	Fecha	Cliente	Person
100000017	F	01/01/2003	CLI1	P1
100000023	F	19/02/2003	CL10	p1
100000024	F	19/02/2003	CLI1	p2
100000026	F	19/02/2003	CLI1	P4
100000030	F	19/02/2003	CLI4	P3
100000031	F	19/02/2003	cli5	P1
100000033	F	27/02/2003	CLI4	P1

Documento	T	Nombre	Clie	Personal
100000017	F	COMERCIAL P	CLI1	P1
100000023	F	LOS COCOS	CL10	p1
100000024	F	COMERCIAL P	CLI1	p2
100000026	F	COMERCIAL P	CLI1	P4
100000030	F	COQUITOS SA	CLI4	P3
100000031	F	LOS PARQUES	cli5	P1
100000033	F	COQUITOS SA	CLI4	P1

Recomendaciones para consultar datos de 2 tablas:

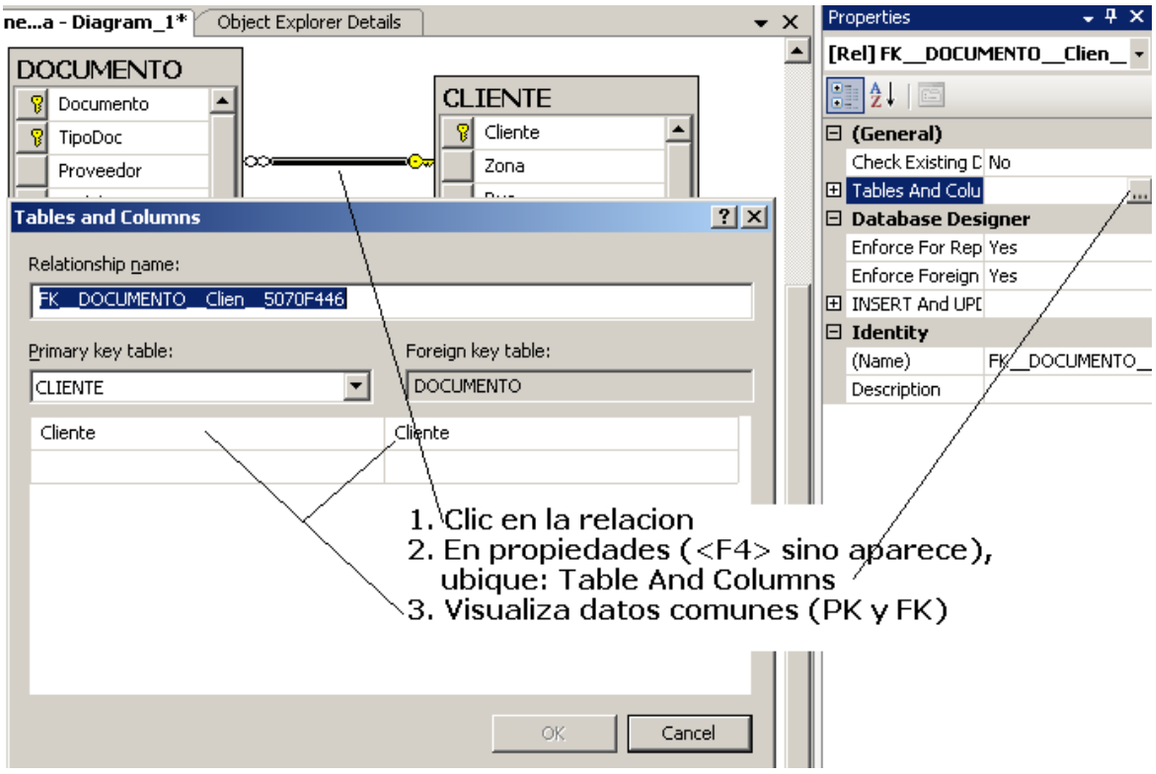
1. Verificar (columnas que se van a mostrar –SELECT,
   Columnas que aparecen en las condiciones –WHERE)
   de que tablas se extraerá la información.
   En el caso expuesto:

Documento	Tipodoc	Nombre	Cliente	Personal
Documento	Documento	Cliente	Documento, Cliente	Documento

   En este caso son las tablas: DOCUMENTO y CLIENTE

2. Verificar si existe relación entre las tablas
   a. Si existe : PK (Clave Primaria) y FK (Clave Foránea) de las tablas relacionadas
   b. Si no existe relación: buscar campos con datos comunes entre las 2 tablas a intercambiar

En el caso expuesto existe una relación directa entre: CLIENTE y DOCUMENTO . Esto puede verlo con el diagramador, una vez ubicadas las tablas, siga los pasos indicados a continuación:



Si no existiera relación lo fundamental es que existan columnas con datos comunes en ambas tablas tal como se puede apreciar a continuación con las columnas cliente en las tablas: CLIENTE y DOCUMENTO

CLIENTE

Cliente	Nombre
CL10	LOS COCOS
CLI1	COMERCIAL PEPITA
CLI2	COMER SAN LORENZO
CLI3	SAN ANTONIO COM
CLI4	COQUITOS SAC
CLI5	LOS PARQUES 1254
CLI7	LAS LIRAS

DOCUMENTO

Document	Tipc	Fecha	Cliente	Person
100000017	F	01/01/2003	CLI1	P1
100000023	F	19/02/2003	CL10	p1
100000024	F	19/02/2003	CLI1	p2
100000026	F	19/02/2003	CLI1	P4
100000030	F	19/02/2003	CLI4	P3
100000031	F	19/02/2003	cli5	P1
100000033	F	27/02/2003	CLI4	P1

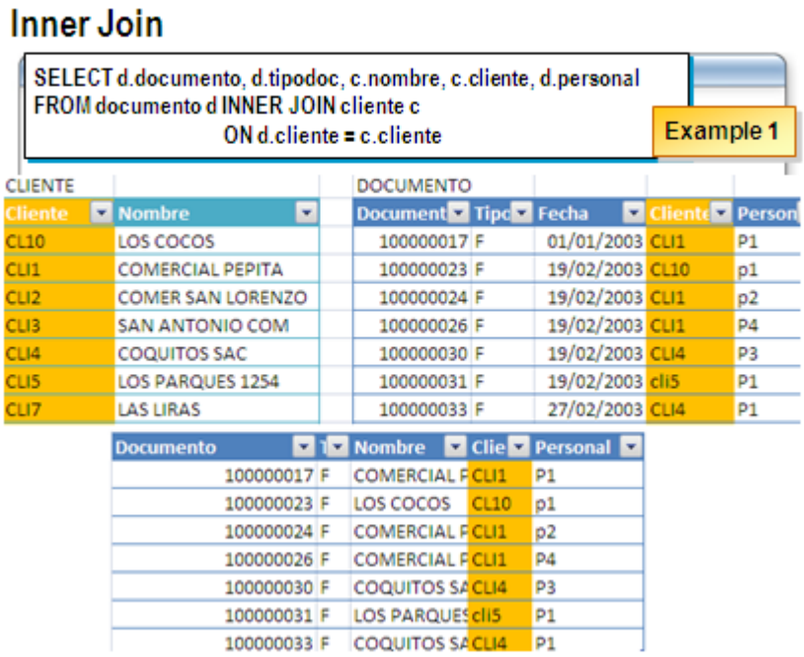
Esto corresponde a la parte del ON (documento.cliente = cliente.cliente)

3. Construir la orden respectiva.
   Para nuestro caso es:

```
USE Tenebrosa
        SELECT documento, tipodoc, nombre, cliente.cliente, personal
        FROM documento  INNER JOIN cliente
                ON documento.cliente = cliente.cliente
```

Note que la palabra JOIN es vital para el enlace de las 2 tablas. INNER es una forma de unión que exige que los datos comunes existan en las tablas unir. En términos del ejemplo con INNER solo aparecerán los clientes que hayan comprado, o sea que existan en DOUMENTO.

Finalmente tenemos:



Como se observó con INNER JOIN la información que brinda es de todos los datos de cliente que existe en ambas tablas.

En algunas ocasiones es necesario mostrar información al 100% de una de las 2 tablas involucradas, esto al margen de que exista la columna común con datos en ambas tablas; por ejemplo si se desea conocer todos los clientes (el 100% de registros) al margen de que tengan documentos se debe recurrir a LEFT JOIN o RIGHT JOIN.

Como saber si debo aplicar LEFT o RIGHT, esto dependerá de la ubicación de la tabla que se desea obtener el 100% respecto a la palabra JOIN. Por ejemplo en el caso de que se quieran todos los clientes

CLIENTE

Client	Nombre
CL10	LOS COCOS
CLI1	COMERCIAL PEPITA
CLI2	COMER SAN LORENZO
CLI3	SAN ANTONIO COM
CLI4	COQUITOS SAC
CLI5	LOS PARQUES 1254
CLI7	LAS LIRAS

DOCUMENTO

Documento	Tip	Fecha	Cliente	Per
100000017	F	01/01/2003	CLI1	P1
100000023	F	19/02/2003	CL10	p1
100000024	F	19/02/2003	CLI1	p2
100000026	F	19/02/2003	CLI1	P4
100000030	F	19/02/2003	CLI4	P3
100000031	F	19/02/2003	cli5	P1
100000033	F	27/02/2003	CLI4	P1
100000034	F	05/05/2000	CLI2	P1

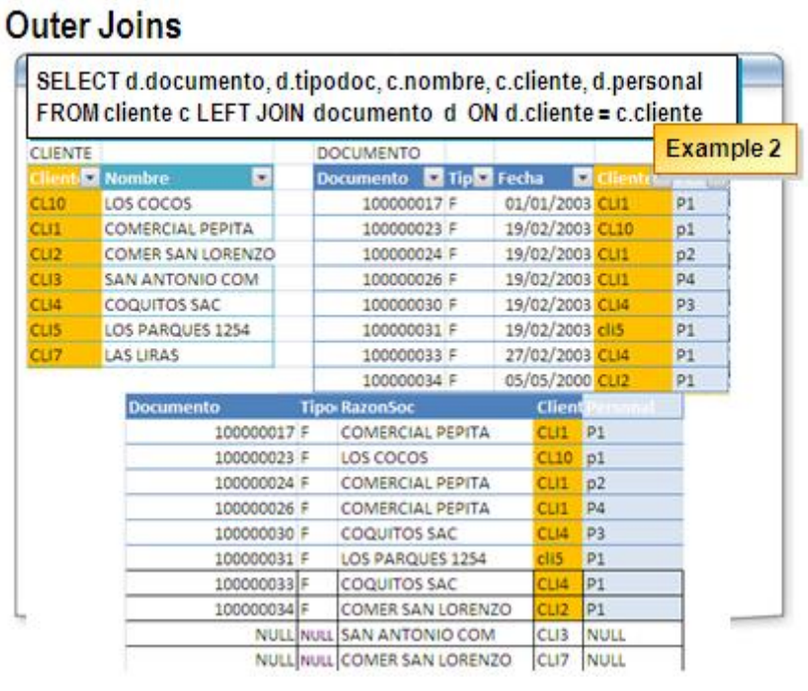
Al aplicar la orden SELECT la tabla CLIENTE se encuentra a la izquierda de la palabra JOIN por lo tanto debe ubicar la palabra LEFT tal como se muestra a continuación

```
SELECT d.documento, d.tipodoc, c.nombre, c.cliente, d.personal
FROM cliente c LEFT JOIN documento  d  ON d.cliente = c.cliente
```

El resultado es:

Documento	Tipo	RazonSoc	Client	Personal
100000017	F	COMERCIAL PEPITA	CLI1	P1
100000023	F	LOS COCOS	CL10	p1
100000024	F	COMERCIAL PEPITA	CLI1	p2
100000026	F	COMERCIAL PEPITA	CLI1	P4
100000030	F	COQUITOS SAC	CLI4	P3
100000031	F	LOS PARQUES 1254	cli5	P1
100000033	F	COQUITOS SAC	CLI4	P1
100000034	F	COMER SAN LORENZO	CLI2	P1
NULL	NULL	SAN ANTONIO COM	CLI3	NULL
NULL	NULL	COMER SAN LORENZO	CLI7	NULL

Finalmente tenemos:



Note que: CLI3 y CLI7 no tienen presencia en DOCUMENTO por lo que el sistema devuelve NULL.

Podemos obtener utilidad con LEFT o RIGHT cuando por ejemplo necesitemos conocer los clientes que no nos han comprado, tendríamos que especificar cualquier columna de documento como NULL, la orden quedaría

```
SELECT d.documento, d.tipodoc, c.nombre, c.cliente, d.personal
FROM cliente c LEFT JOIN documento  d  ON d.cliente = c.cliente
WHERE d.documento IS NULL
```

Verifique los resultados.

Lab 11: Consultando Múltiples Tablas

Objetivos

- Cumplir requerimientos utilizando la orden SELECT incluyendo JOINS

Lista de Requerimientos

1. Se desea conocer las ventas al crédito pendientes de pago mostrando el nro de documento, razón social del cliente y la fecha en que se generó la venta.

2. Se desea conocer las venta de los productos superiores a 20 unidades, mostrando el código del producto, nombre del cliente, nro de documento y cantidad vendida, precunit

3. Se necesita conocer las ventas superiores a 500 unidades de todos los productos de la línea L2. Especificando el nro de documento, nombre del vendedor, monto vendido y la fecha de emisión del documento.

4. Se pretende realizar una depuración de los productos que no han tenido ni una sola venta en lo que va desde el inicio de las operaciones de la empresa. Muestre estos productos.

5. Se desea conocer los pagos efectuados a partir del año 2000, mostrando el nro de pago, el nro del documento, su tipo, el importe pagado, el nombre del cliente y la fecha en que se produjo el pago.

Sesión 12

SUMARIZANDO DATOS

Sesión12. Sumarizando Datos

Existe otra forma en que los usuarios realicen sus requerimientos de información, y esta es en forma resumida. Así por ejemplo un requerimiento podría ser: obtener un resumen de las ventas por cada producto en el mes de Mayo, o un consolidado de ventas por cada cliente en 1999.
A continuación conoceremos una extensión de la orden SELECT aplicando GROUP BY para generar resultados agrupados y HAVING como herramienta de filtro de los grupos generados.

Objetivos

- Generar un simple sumario usando funciones de agregamiento
- Organizar los datos resumidos de una columna usando GROUP BY y la clausula HAVING
- Usando TOP n para recuperar una lista de los n registros top de una tabla. Aplicando ORDER BY

Usando Funciones de Agregamiento

A continuación presentamos un resumen de las principales funciones de agregamiento

Función	Descripción
AVG	Promedio
COUNT	Contar valores de una expresión
COUNT (*)	Número de registros
MAX	Máximo valor de la expresión
MIN	Mínimo valor de la expresión
SUM	Acumula valores de una
STDEV	Desviación estándar
STDEVP	Desviación estádar de toda la
VAR	Varianza
VARP	Varianza estadística de todos los

Sintáxis:

```
SELECT {{AVG | COUNT | MAX | MIN | SUM
        | STDEV | STDEVP | VAR | VARP} (expression | *) } [, . . . n] FROM tabla
 [ WHERE condiciones_de_búsqueda ]
 [ GROUP BY [ALL] lista_de_campos ]
 [ HAVING condioion de grupo ]
```

Consideraciones al usar GROUP BY

1. SQL Server produce un registro con los valores de cada grupo definido en GROUP BY
2. Usando GROUP BY tendremos información resumida.
3. Todas las columnas que se especifican en la lista SELECT deben ser incluidas en GROUP BY.
4. Al incluir WHERE sólo aparecerán los registros que satisfacen la condición en WHERE
5. El tamaño de registro máximo listado por GROUP BY no debe ser mayor a 8060 bytes.
6. Si se encuentran valores NULL estos son procesados como un grupo en el caso de obviar NULL especificar en el WHERE la expresión IS NOT NULL

Lab 12: Sumarizando Datos

Objetivos

- Cumplir requerimientos utilizando la orden SELECT incluyendo JOINS, GROUP BY HAVING

Lista de Requerimientos

1. Se desea tener un resumen de las cantidades vendidas por cada producto desde 2000.
2. Se desea conocer un resumen de ventas al contado y al crédito que han sido canceladas.
3. Se necesita conocer los montos deudores acumuladas por cada cliente a la fecha.
4. Se desea conocer el monto vendido por cada vendedor desde Enero del 2000 a la fecha.
5. Necesitamos conocer el volumen de ventas, en unidades monetarias, por cada línea de producto.
6. Se requiere un resumen de deuda de cada cliente siempre que sean superiores a 5000 unidades monetarias.
7. Se desea conocer los 3 mejores vendedores del año 2000
8. Se pretende tener el total vendido -tanto en soles como en cantidades- por cada producto, mostrando en el detalle: código, descripción, fecha del documento, monto y cantidad
9. Se desea tener un resumen de ventas por línea de productos en forma trimestral.

Sesión 14

MODIFICANDO DATOS

Sesión 14 .Modificando Datos

En todo sistema de información se presentan los cambios continuos en las tablas ya sea porque se tiene que registrar un nuevo pedido, actualizar el stock de un producto o eliminar información aislada.
Esto lo haremos utilizando las ordenes INSERT, UPDATE y DELETE. Es en este momento donde se activarán las reglas de integridad de datos permitiendo o rechazando los cambios que se intenten dar

Objetivos

- Conocer la orden INSERT
- Conocer la orden UPDATE
- Conocer la orden DELETE
-

INSERT:

- Agregar registros de información a las tablas
- Registros en una tabla por vez
- Se activan las reglas de integridad de datos
- Sintaxis:

  INSERT [INTO] <nom_tabla> [<lista_campos>]
  VALUES <lista_valores>

Ejemplos

```
INSERT linea (linea, descripcion)
VALUES ('TU', 'GASES')
--select * from linea
-- 2--  error PK
INSERT linea (linea, descripcion) VALUES ('TU', 'GASES')
-- 3  - NO CORREr FALTAN COLUMAS
INSERT linea (linea)  VALUES ('TX', 'BEBIDAS')
-- 4  -  OK
INSERT linea (linea)   VALUES ('TX')
--   5  --> ERROR FK (LINEA)
SELECT * FROM LINEA
INSERT marca (marca, proveedor, linea, descripcion)
VALUES ('X1', 'PV01', 'KR', 'PRIMAVERA')
--  6 --> OK
INSERT marca (marca, proveedor, linea, descripcion)
VALUES ('X1', 'PV01', 'L2', 'PRIMAVERA')
```

DELETE:

- eliminar registros de información a las tablas
- Registros en una tabla por vez
- Se activan las reglas de integridad de datos
- Sintaxis:

  DELETE [FROM] <nom_tabla>
  [WHERE <condición>]

Ejemplos

```
DELETE linea where linea ='TU'
--- 2  error : FK
DELETE linea where linea ='L1'
```

UPDATE:

- modificar registros de información a las tablas
- Registros en una tabla por vez
- Se activan las reglas de integridad de datos
- Sintaxis:

```
UPDATE <tabla> SET <campo> =<expresión>
  [{,<campo1> = <expresion1>}]
[WHERE <condición>]
```

Ejemplos

```
UPDATE cliente SET direccion ='AV. MARTINEZ 500'
WHERE cliente ='CLI1'
--1 Modificar el saldo con 1000 y ponerle sujeto de credito, CLI2
UPDATE cliente SET saldo =1000,  credito = 1
WHERE cliente ='CLI2'

-- Quitar el status de credito a todos los clientes que tengan 1 documento
-- pendiente con mas de 300 dias de vencido
UPDATE c SET credito = 0
FROM cliente c INNER JOIN documento d ON d.cliente = c.cliente
WHERE d.estado = 'P' AND DATEDIFF(dd, d.fecha, GETDATE()) > 300
```

Sesión 16

MANEJO DE TRIGGERS

Sesión 16. Implementando Triggers

Constituyen una forma avanzada de Integridad de Datos ya que permite asegurar que las reglas de negocio se plasmen automáticamente cuando se ejecuta un cambio en la BD. Por ejemplo cada vez que se realiza un ingreso de productos al almacén automáticamente debe de actualizarse el stock del producto.

Objetivos

- Describir el uso de un Trigger y cuando debe ser usado
- Cómo se crea un Trigger.
- Describir como trabaja un Trigger

Introducción a los Triggers

Qué es *un Trigger ?*

Un Trigger es un tipo especial de Stored Procedure que se ejecuta automáticamente cundo se pretende realizar un cambio a una tabla de la BD, es decir cuando se aplica: UPDATE, DELETE, INSERT.

Características

- Van asociados a una Tabla
- Son invocados automáticamente
- No pueden ser llamados directamente
- Constituyen una transacción

Usos de un Trigger

- Modificaciones en cascada para tablas relacionadas (Padre-Hijo); por ejemplo: Docum y LineaDocum
- Forzar la complejidad de la Integridad de Datos
- Personalizar mensajes de error
- Comparar el estado de un registro antes o después de un cambio

Las Tablas deleted e inserted

Cuando se activa la ejecución de un Trigger automáticamente aparecen 2 tablas: inserted y deleted.
La tabla deleted almacena información de todos los registros afectados a
consecuencia de una orden DELETE ó UPDATE
La tabla inserted almacena información de todos los registros afectados a
consecuencia de una orden INSERT ó UPDATE
Estas dos tablas constituyen una gran utilidad en el manejo de los triggers ya que ser usadas para determinar el estado actual y nuevo del dato a modificar como ser verá a continuación.

Creando Triggers

Se utilizar la orden CREATE TRIGGER. Para lo cual es necesario tener los permisos respectivos

```
CREATE TRIGGER trigger_name ON
table
[WITH ENCRYPTION] {FOR
{[INSERT][,][UPDATE][,][DELETE]}

[WITH APPEND]

[NOT FOR REPLICATION] AS
        sql statement [,...n]}
```

Por ejemplo, cuando se produce un ingreso de un producto al almacén automáticamente debería de actualizar el stock actual del producto

```
CREATE TRIGGER ti_producto
ON producto
FOR INSERT
AS
/* Capturando datos ha actualizar */

UPDATE producto
SET stockac = stockac + inserted.cantidad
FROM lineadocum l INNER JOIN inserted
                ON l.nrodoc = inserted.nrodoc AND l.idproducto = inserted.producto
```

Cómo Trabaja un INSERT Trigger

- Existe una orden INSERT asociada con un INSERT Trigger
- La orden INSERT es ejecutada
- El INSERT Trigger es disparado con todas las órdenes SQL definidas dentro del Trigger

Cuando se dispara el Trigger la nueva información es almacenada en la Tabla en que se produce el INSERT y la tabla inserted. La tabla inserted es una tabla lógica que permanece en memoria y sobre la cual podemos hacer mención en cualquiera de sus campos a fin de disparar alguna acción en otras tablas si deseamos.

Cómo Trabaja un DELETE Trigger

- Existe una orden DELETE asociada con un DELETE Trigger
- La orden DELETE es ejecutada
- El DELETE Trigger es disparado con todas las órdenes SQL definidas dentro del Trigger

Cuando se dispara el Trigger la información eliminada es almacenada en la Tabla en la tabla deleted. La tabla deleted es una tabla lógica que permanece en memoria y

sobre la cual podemos hacer mención en cualquiera de sus campos a fin de disparar

alguna acción en otras tablas si deseamos.

Por ejemplo, cuando se produce la eliminación de la línea de una factura automáticamente debería de actualizar el stock actual del producto deduciendo la cantidad eliminada

```
CREATE TRIGGER td_producto
ON producto
FOR DELETE
AS
/* Capturando datos ha actualizar */

UPDATE producto
SET stockac = stockac - deleted.cantidad
FROM lineadocum l INNER JOIN deleted
                ON l.nrodoc = inserted.nrodoc AND l.idproducto = inserted.producto
```

Cómo Trabaja un UPDATE Trigger

- Existe una orden UPDATE asociada con un UPDATE Trigger
- La orden UPDATE es ejecutada
- El INSERT Trigger es disparado con todas las órdenes SQL definidas dentro del Trigger

Cuando se dispara el Trigger la nueva información es almacenada en la Tabla inserted y la información que se cambia es almacenada en la tabla deleted. La tabla inserted, así como la tabla deleted, son tablas lógica que permanece en memoria y sobre la cual podemos hacer mención en cualquiera de sus campos a fin de disparar alguna acción en otras tablas si deseamos.

Por ejemplo, cuando se produce la modificación de la línea de un documento grabado anteriormente, automáticamente debería de actualizar el stock actual del producto deduciendo la cantidad cambiada y agregando el nuevo valor.

```
CREATE TRIGGER tu_producto
ON producto
FOR UPDATE
AS
/* Capturando datos ha actualizar */

UPDATE producto
SET stockac = stockac - deleted.cantidad + inserted.cantidad
FROM lineadocum l INNER JOIN deleted
                ON l.nrodoc = inserted.nrodoc AND l.idproducto = inserted.producto
```

Ejercicio

- Crear los trigger respectivos para actualizar el saldo de un cliente cada vez que éste realiza una compra al crédito o paga un documento.
- Crear un trigger para actualizar las compras efectuadas.

Lab 16: Implementando Triggers

Objetivos

- Preparar triggers para actualiza el Stock del producto y el Saldo de cliente para una venta al crédito

a. Al realizar una venta el contado o crédito

```
CREATE TRIGGER ti_detadoc_stock ON dbo.DETADOC

FOR INSERT

AS

DECLARE          @valor numeric(9,2), @cliente CHAR(6),   @fp CHAR(1)

                        -- Capturando variables

 SELECT @fp = p.formapago, @cliente =d.cliente, @valor = SUM(i.cantidad*i.precunit) FROM inserted i

 INNER JOIN documento d ON i.documento = d.documento AND i.tipodoc = d.tipodoc

                INNER JOIN pedido p        ON d.pedido = p.pedido

 GROUP BY p.formapago, d.cliente

                        -- Actualizando el stock

 UPDATE producto

 SET stockac =stockac + i.cantidad * t.signo FROM inserted i INNER JOIN

 producto ON i.producto = producto.producto

                INNER JOIN tipodoc t ON t.tipodoc = i.tipodoc

/* Actualizando saldo de cliente, si la venta no es al contado */

IF    @fp  o 'C

      UPDATE cliente SET saldo = saldo + @valor WHERE cliente = @cliente
```

b. Al eliminar una Venta

```
CREATE TRIGGER td_detadoc_stock ON dbo.DETADOC

FOR DELETE

AS

DECLARE @valor numeric(9,2), @cliente CHAR(6),   @fp CHAR(1)

                        -- Capturando variables

 SELECT @fp = p.formapago, @cliente =d.cliente, @valor = SUM(i.cantidad*i.precunit) FROM deleted i

 INNER JOIN documento d ON i.documento = d.documento AND i.tipodoc = d.tipodoc

                INNER JOIN pedido p         ON d.pedido = p.pedido

 GROUP BY p.formapago, d.cliente

                        -- Actualizando el stock

 UPDATE producto

 SET stockac =stockac - i.cantidad * t.signo FROM deleted  i INNER JOIN

 producto ON i.producto = producto.producto

            INNER JOIN tipodoc t ON t.tipodoc = i.tipodoc /*

Actualizando saldo de cliente, si la venta no es al contado */ IF

@fp  o 'C

      UPDATE cliente SET saldo = saldo - @valor WHERE cliente = @cliente
```

c. Prepare el Trigger respective en el caso de una modificación

Sesión 17

IMPLEMENTANDO STORED PROCEDURES

Sesión 17. Implementando Stored Procedures

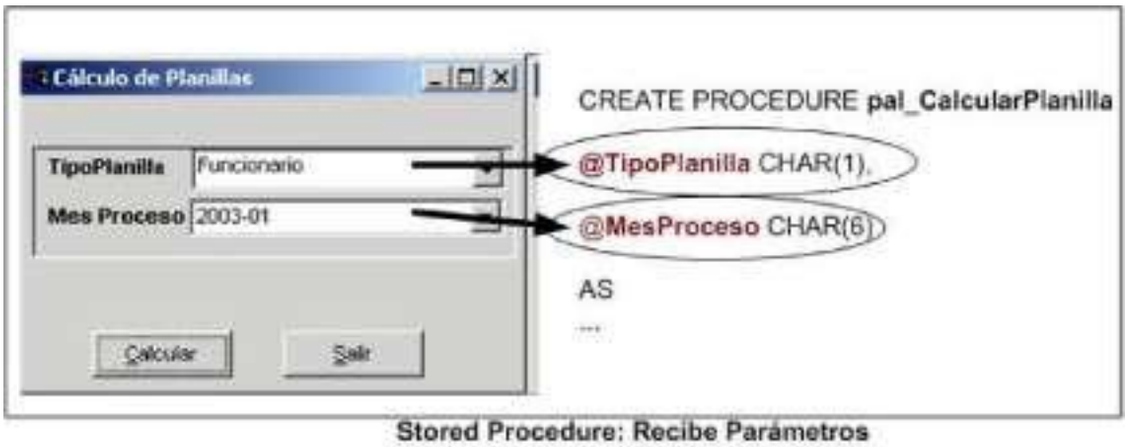
Los SP permiten incrementar la performance de la BD, realzan la seguridad de la BD e incorporan técnicas de programación basadas en el Transact SQL. A continuación veremos los tipos de SP, como crearlos y ejecutarlos..

Objetivos

- Describir el propósito y razones de usar SP.
- Explicar como se procesa un SP en SQL Server.
- Determinar cuando usar SP para completar tareas en SQL Server.

Definición

Es una colección de ordenes Transact-SQL que van orientados a incrementar la performance de las tareas repetitivas. Por ejemplo si desea realizar un cálculo de planillas o realizar un proceso de facturación en Batch. Es recomendable usarlos cuando desde el cliente se van a ejecutar más de una instrucción SQL. Tienen la capacidad de aceptar parámetros y de retornar valores . Por ejemplo imagine el cálculo de planillas teniendo una interfaz desde la aplicación tal como se muestra en el siguiente diagrama:



Stored Procedure: Recibe Parámetros

Tipos

Existen 5 tipos

- System
- Local
- Temporary
- Re mote
- Extended

System SP

System SP son almacenados en la Base de Datos **Master** y llevan típicamente el

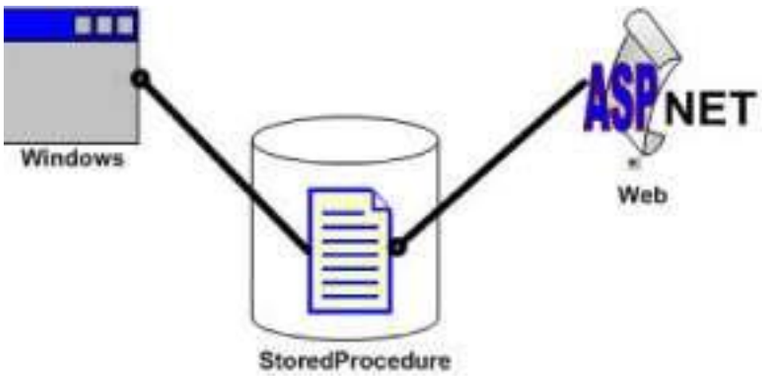
prefijo sp_, pueden invocar el llamado de datos de las tablas del sistema, ejecutar tareas administrativas y tareas de seguridad.
Por ejemplo, si desea conocer la información del catálogo de la tabla de **cliente** puede ejecutar sp_help de la siguiente manera:

```
Sp_help cliente
```

Local SP
Son comúnmente los más usados y corresponden a los Stored Procedures que vamos creando en nuestros servidores locales.

Ventajas de las SP

Compartir lógica de aplicaciones: Un Stored Procedure puede ser invocado desde una plataforma Window o desde el Web.



Proporciona Mecanismos de Seguridad:

Representan cierta ventaja en la simplificación de las tareas de seguridad, dado que sólo es necesario tener permiso de ejecución no importando que tenga acceso a las tablas involucradas dentro del Stored Procedure.
Es una de las formas más recomendadas para el llamado desde aplicaciones Web.

Incrementan la Performance:
Dado que los procesos se realizan en el servidor los tiempos de ejecución tienden a ser mejores.

Reducen el tráfico de red
Dado a que la lógico de proceso se encuentra en el servidor el llamado desde la aplicación (cliente) a la Base de Datos (servidor) se limita al mandado de parámetros.

Creando Stored Procedures

Sixtáxis:

```
CREATE PROCEDURE <Nombre Stored ProO [
  @parámetro tipo_dato [ = default ] [ OUTPUT ]
 ]

AS
<ordenes_SQL>
```

Por ejemplo: vamos a suponer que queremos conocer los clientes que nos adeudan 2 documentos pendientes de pago:

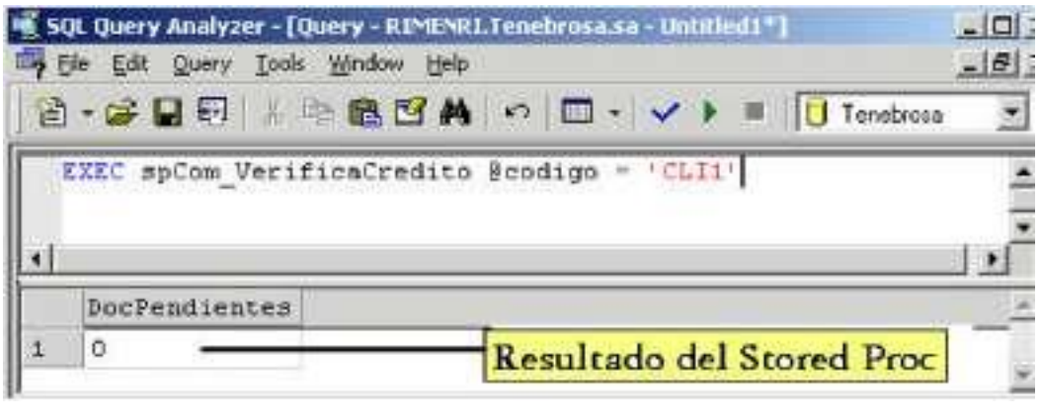
```
CREATE PROCEDURE spCom_VerificaCredito
@codigo CHAR(4)          — es un parámetro de entrada desde la aplicación
AS
      — Imagine que para los clientes tipo 'A' no se produce validación SET NOCOUNT ON IF (SELECT tipocliente FROM cliente
WHERE cliente = @codigo) = 'A
    SELECT 0 As DocPendientes  -- Esto llegaría como RecordSet al cliente ELSE
-- Se produce verificación para cliente en el estado del documento si es diferente de tipo 'A'
  SELECT DocPendientes = COUNT(*) -- Esto llegaría como RecorsSet al cliente
```

El llamado de la interfaz podría ser:

El código muestra desde Visual Basic sería:

```
Private Sub TxtCliente_Validate(Cancel As Boolean)
    Dim rs As adodb.Recordset
    Dim sql As String
        '  Llamado del Stored Procedure
    sql = " EXEC spCom_VerificaCredito @codigo='" & TxtCliente.Text & "'"
    rs.Open sql, cn, adOpenForwardOnly, adLockOptimistic
            '  Nótese DocPendientes viene del Stored Procedure If
    rs!DocPendientes > 1 Then
    MsgBox "Crédito No Procede!!! Nro Doc Pendientes =" & Str(rs! DocPendientes) Else
        MsgBox "Crédito Procede!!! Nro Doc Pendientes =" & Str(rs!DocPendientes) End
    If Set rs = Nothing
```

Pi



Usando parámetros en Stored Procedures

Los SP tienen la capacidad de recibir parámetros y de entregar valores.

Parámetros de Entrada

Son aquellos que reciben los valores mandados desde la aplicación, pueden incluirse valores por defecto los mismos que entrarían a trabajar en el caso no hayan definido al momento de su llamado. En el caso de que hayan sido definidos como cadena, tomarán la longitud del parámetro en el SP.

```
CREATE PROCEDURE spCom_VerificaCredito
@codigo CHAR(4)          -- es un parámetro de entrada desde la aplicación
AS                       -- deben ser definidos con su tipo de dato
respectivo
```

Formas de ejecutar un Stored Procedure desde el Analizador de Consultas:

```
EXEC spCom_VerificaCredito 'CLI1'          --Llamado por posición
ó
EXEC  spCom_VerificaCredito @codigo= 'CLI1'      -- Llamado por nombre de
                          parámetro, recomendada
```

Retornando Valores usando parámetros de salida:

```
CREATE PROCEDURE Formula
    @m1 smallint,
    @m2  smallint,
    @resultado smallint OUTPUT     --     parámetro de salida
AS
SET @resultado = @m1* @m2     -- el último valor de @resultado será
                              es el devuelto

DECLARE @Valor  smaI int
EXECUTE Formula 7,5, @Valor OUTPUT
SELECT 'Resultado = ', @Valor

--observe OUTPUT

        Resultado = 35
```

Manejando Cursores en un Stored Procedure

Existe necesidad de hacer recorridos secuenciales a un grupo de registros, ya sea por ejemplo el realizar un cálculo de planillas, en donde trabajador por trabajador vamos calculando los importes de ingresos, descuentos y cuotas patronales; o de realizar un cálculo de intereses a un grupo de clientes 1 por uno. En este caso SQL Server nos ofrece la posibilidad de realizar estos barridos secuenciales utilizando cursores.

Como se verá a continuación el trabajo con cursores no es muy complicado, pero podría restar tiempos de ejecución del servidor, ya que los cáculos se realizan con los recursos directos de la memoria del servidor, por ello se recomienda trabajar con ellos cuando es necesario.

El manejo de cursores se asemeja a procesar :

- Un recordset en Visual Basic , preguntando por el EOF (fin de archivo), con su bucle Do While para procesarlo y con MoveNext avanzar al sgte registro.
- Una tabla en Visual Fox, verificando el fin de archivo, con un bucle While para procesarlo y un Skip para avanzar al sgte registro.
- Recorrer un Data Window, verificando con la función RowCount() y con el bucle For procesarlo directamente.

Pasos

- Declarar el cursor

```
DECLARE Nombre_Cursor CURSOR
FOR orden_select        — se declara a partir de una orden
```

- Abrir el cursor

```
OPEN  Nombre Cursor
```

- Leer el cursor. Para leer el cursor es necesaria almacenar su información en variables de memoria. La cantidad de variables está en función al número de campos definidos en el SELECT. FETCH también cumple la función de preparar el avance para el siguiente registro del cursor. Por defecto avanza al siguiente registro (NEXT)

```
FETCH [NEXT|PRIOR|FIRST |LAST] Nombre_Cursor INTO <lista_variab_memoria>
```

- Verificar que exista información en el cursor

@@fetch_status = 0 -- cuando es cero significa que existe información por procesar en el

- Procesar el cursor

```
WHILE
    @@fetch_s
    tatus = 0
    BEGIN
        — <procesar el cursor>
        FETCH Nombre_Cursor INTO <lista_variab_memoria> --
    leer, avanzar END
```

- Cerrar el cursor

```
CLOSE  Nombre Cursor
```

- Desactivarlo de memoria

```
DEALLOCATE  Nombre Cursor
```

Por Ejemplo:

Suponga que desea mostrar el detalle de movimientos de un producto sus ingresos y sus salidas, lo que sería el tener un Kardex del producto

Lab 17: Implementando Stored Procedures

Objetivos

- Combinar la potencialidad de programación del lado del servidor que ofrece SQL Server.

a. Generador de Nros a partir de un formato establecido

```
CREATE PROCEDURE sp_siguenumero_doc @doc
CHAR(1),
@tipo  tinyint,     -- Ver valores   para tipo
@number CHAR(9) OUTPUT -- Valores para
tipo
/* devuelve el correlativo segun el tipo de documento y el centro de servicio */
/*    1    ->    Incrementa un valor al tipo documento     */ /*    2    ->
Disminuye     el valor al tipo documento     */ /*    3    ->    Consulta Valor del
tipo de documento actual    */ /*    0    ->    Consulta Valor siguiente al
documento actual */ AS
DECLARE @cuenta tinyint, @suma smallint, @factor tinyint SET
NOCOUNT ON
SELECT @number =serie +REPLICATE('0', 6 - LEN(LTRIM(RTRIM(numero+1)))) +
LTRIM(RTRIM(numero+1)) FROM tipodoc WHERE tipodoc = @doc IF @@rowcount
= 0 BEGIN
          RAISERROR(' No Se han Programado numeracion para esta Actividad ', 16, -
1)
    RETURN END
ELSE
     BEGIN
      IF   @tipo = 1
        BEGIN
           UPDATE tipodoc     SET numero = numero + 1      WHERE tipodoc = @doc SELECT @number
     =serie +REPLICATE('0', 6 - LEN(LTRIM(rtrim(numero))))+LTRIM(rtrim(numero))
            FROM tipodoc       WHERE tipodoc = @doc END
      IF   @tipo = 2
          BEGIN
           UPDATE tipodoc   SET numero = numero - 1     WHERE tipodoc= @doc
  SELECT @number = serie +REPLICATE('0', 6 - LEN(LTRIM(rtrim(numero)))) + LTRIM(rtrim(numero))
           FROM tipodoc cn     WHERE tipodoc = @doc END
     IF    @tipo = 3
         SELECT @number =serie +REPLICATE('0', 6 - LEN(LTRIM(rtrim(numero))))+LTRIM(rtrim(numero))
             FROM tipodoc      WHERE tipodoc = @doc END
SELECT @number AS numero GO
```

b. Generando Cronograma de Pagos:

El presente ejemplo supone una venta realizada al Crédito, en donde se necesitan como parámetros el Nro de Documento, Tipodoc y Nro de Cuotas a financiar el crédito

Creando Tabla Cronograma

```
CREATE TABLE cronograma (idCronograma INT NOT NULL IDENTITY(1,1) PRIMARY
KEY,
Documento CHAR(9), TipoDoc CHAR(1), Importe numeric(9,2),
Interes numeric(9,2), IgvInteres numeric(9,2), feVence  datetime, Fepago datetime,
estado CHAR(1) NOT NULL DEFAULT 'P')
```

Preparando el Stored Procedure

```
CREATE PROCEDURE sp_Genera_Cronograma @Doc
CHAR(9), @td CHAR(1), @NroCuotas int AS

DECLARE @igv numeric(9,2),   @Interes numeric(9,2), @Deuda numeric(10,2),
@Cuenta smallint

SET NOCOUNT ON
-- Capturando IGV
SELECT @igv = Igv/100, @Interes = TasaInt /100 FROM parametro WHERE activo =
1
—   Calculando Deuda para documento ingreado
SELECT @Deuda = SUM(cantidad * precUnit)   FROM detadoc
WHERE documento = @doc AND tipodoc = @td

SET @Cuenta = 0 -- Empieza
Generación de Cuotas WHILE
@Cuenta < @NroCuotas BEGIN
        SET @Cuenta = @Cuenta + 1
INSERT cronograma ( Documento , TipoDoc , Importe ,Interes, IgvInteres, feVence )
VALUES (@doc, @td, @Deuda / @NroCuotas,  @Deuda * @Interes , @Deuda* @Interes
*@Igv,
                        DATEADD( mm, @Cuenta, GETDATE())  )
        —Observese el uso de DATEADD incrementa:   @Cuenta a la fecha de referencia
getdate() ,
        — Un tiempo : mm-meses
   END

-- mostrado datos al cliente
SELECT * FROM cronograma   WHERE documento = @doc AND tipodoc = @td

GO
```

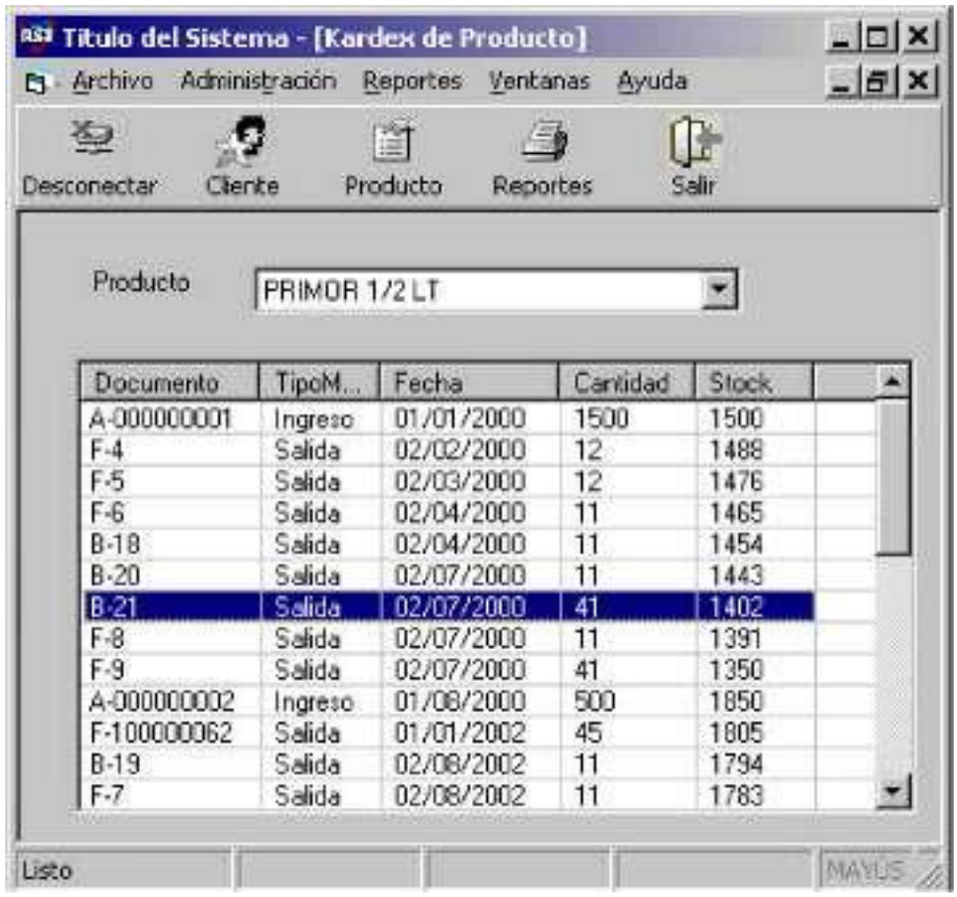
Ejemplo de ejecución desde el Query Analyzer exec sp_Genera_Cronograma

@doc = '100000017', @td = 'F', @NroCuotas = 10

c. Reportando un Kardex de Producto:

Uno de los reportes tradicionales en un Sistema de Comercialización lo constituye el conocer los movimientos de un producto determinado. Es decir mostrar las compras y ventas. En este sentido nuestro modelo propuesto incluye tanto compras como ventas en las tabla documento y detadoc. Veamos la interfaz propuesta desde el cual se haría el llamado respectivo:

a. Interfaz posible que invoque al Store Procedure


Titulo del Sistema - [Kardex de Producto]
Archivo Administración Reportes Ventanas Ayuda
Desconectar
Cliente
Producto
Reportes
Salir
Producto
PRIMOR 1/2 LT
Documento TipoM... Fecha Cantidad Stock
A-000000001 Ingreso 01/01/2000 1500 1500
F-4 Salida 02/02/2000 12 1488
F-5 Salida 02/03/2000 12 1476
F-6 Salida 02/04/2000 11 1465
B-18 Salida 02/04/2000 11 1454
B-20 Salida 02/07/2000 11 1443
B-21 Salida 02/07/2000 41 1402
F-8 Salida 02/07/2000 11 1391
F-9 Salida 02/07/2000 41 1350
A-000000002 Ingreso 01/08/2000 500 1850
F-100000062 Salida 01/01/2002 45 1805
B-19 Salida 02/08/2002 11 1794
F-7 Salida 02/08/2002 11 1783
Listo
MAYUS

Es fundamental la presentación de los movimientos ordenados en forma cronológica y la idea es ir calculando el Stock por cada movimiento procesado secuencialmente. Para ello prepararemos una tabla temporal la cual la recorreremos con Cursores y línea a línea iremos calculando el Stock respectivo

Preparando el Store Procedure para reportar el Kardex de un Producto

```
CREATE PROCEDURE spt_Kardex
@producto CHAR(4) AS

SET NOCOUNT ON
— Preparando Tabla Temporal
DECLARE @stockac numeric(9,2)
SET @stockac = 0  — Asignando Valor
— Preparando Tabla Temporal con Movimientos para producto requerido
SELECT d.documento , d.tipodoc, d.fecha, dd.cantidad, t.signo, @stockac AS Stock—stock se genera con valor 0
INTO #Kardex  — Permite crear una nueva tabla , en este caso #Kardex - que es temporal
FROM    documento d INNER JOIN detadoc dd ON d.documento = dd.documento AND d.tipodoc = dd.tipodoc
                    INNER JOIN tipodoc t ON t.tipodoc = d.tipodoc
WHERE dd.producto  = @producto -- Paso 1 Declarando Cursor
DECLARE c_Kardex CURSOR
FOR SELECT documento, tipodoc, cantidad , signo  FROM #Kardex ORDER BY fecha, signo DESC     -- Aparece desde el más antiguo al mas nuevo, venta mismo día 1ro ingresos

-- Paso 2. Abriendo Cursor
OPEN c_Kardex

DECLARE @doc char(9), @td char(1),  @can numeric(9,2), @signo smallint

— Paso 3. Leyendo Cursor y preparando para sgte registro
FETCH c_Kardex INTO @doc, @td, @can, @signo

— Paso 4. Procesando Cursor mientras existan datos
WHILE @@fetch_status   = 0
   BEGIN
       SET @Stockac = @stockac  +  @can * @signo

       UPDATE #Kardex SET stock = @stockac WHERE
       documento = @doc AND tipodoc = @td

       FETCH c_Kardex INTO @doc, @td, @can, @signo
   END
-- Paso 5. Cerrando Cursor
CLOSE c_Kardex
-- Paso 6 . Desactivando de memoria
DEALLOCATE c_Kardex
— Preprando Datos para devolver a la aplicación
SELECT tipodoc  +'-' + documento  AS documento,
tipomov = CASE WHEN signo = 1 THEN 'Ingreso' ELSE 'Salida' END,
fecha, cantidad, stock
FROM #kardex
ORDER BY fecha, signo DESC
GO
```

Ejemplo de ejecución desde el Query Analyzer

```
EXEC spt_Kardex @producto= 'PR02'
```

Sesión 18

IMPLEMENTANDO FUNCIONES

DEFINIDAS POR USUARIO

Sesión 18. Implementando Funciones Definidas por Usuario

A partir de la versión 2000 Sql Server proporciona la capacidad de que uno mismo pueda crear sus propias funciones

Algunas de estas funciones pueden ejecutarse desde una orden SELECT u otra orden del lenguaje de modificación de datos

Objetivos

- Describir funciones escalares
- Describir funciones multistatement Table-valued
- Describir funciones In-Line Tabled-Value

Sintaxis

```
CREATE FUNCTION <nombre_function>
 (<lista_de_parámetros>)
RETURNS (tipo_dato_esperado_que _retorne)

AS
BEGIN
(<ordenes_sql>
RETURN <valor_devuelto_funcion>)
END
```

Ejemplo:

Suponga que desea calcular el interés que debe tener un documento pendiente en función a los días transcurridos desde su emisión

Lab 18: Implementando Funciones Definida por Usuario

Objetivo

- Aplicar una función escalar y determinar su utilidad a partir de una orden SELECT

Función que calcula la mora

```
CREATE FUNCTION f_CalculaInteres
(@f1 datetime, @f2 datetime)
RETURNS numeric(9,2)
AS
BEGIN

        DECLARE @mora numeric(9,2)
        IF @f2 - @f1 > 0
         BEGIN
                DECLARE @TasaLegal numeric(9,2)
                SELECT @TasaLegal = TasaLegal /100   FROM parametro WHERE activo =
1
              — DATEDIFF toma los <dd -dias > transcurridos SET @mora
      =    DATEDIFF(dd, @f1, @f2) * @tasaLegal END ELSE SET @mora = 0

       RETURN  @mora

END
```

Llamando a la función desde una orden SELECT

```
SELECT *, DBO. f_CalculaInteres (FECHA, GETDATE())    fROM documento
```