BASE DE DATOS AVANZADO I

Unidad 3: Introducción a la programación Transact-SQL y Cursores

Tema 6: Programación en SQL

Tema 6: Programación en SQL

3.1. Tema 6: Programación en SQL
3.1.1. Fundamentos de la programación con Transact-SQL
3.1.2. Identificadores
3.1.3. Variables: declaración, asignación
3.1.4. Elementos de flujo de control
- Estructuras de control IF
- Estructura condicional CASE
- Estructura de control WHILE

3.1.5. Control de errores con TRY / CATCH, uso de @@Error, uso del RaisError
3.1.6. Uso de transacción: Commit y RollBack

Capacidades

1. Identifica las sentencias y estructuras de programación del lenguaje Transact-SQL
2. Diseña e implementa bloques de programas para optimizar las operaciones con la base de datos.
3. Implementa cursores

1. Fundamentos de la programación Transact- SQL

- Transact-SQL (T-SQL) es la extensión patentada de Microsoft y Sybase al SQL (Structured Query Language) utilizado para interactuar con bases de datos relacionales. T-SQL amplía el estándar SQL para incluir programación de procedimientos, variables locales, diversas funciones de soporte para el procesamiento de cadenas, procesamiento de fechas, matemáticas, etc. y cambios en las instrucciones DELETE y UPDATE.
- Transact-SQL es fundamental para usar Microsoft SQL Server. Todas las aplicaciones que se comunican con una instancia de SQL Server lo hacen mediante el envío de instrucciones de Transact-SQL al servidor, independientemente de la interfaz de usuario de la aplicación.

2. Manejo de variables

- Una variable es una entidad a la que se asigna un valor. Este valor puede cambiar durante el proceso donde se utiliza la variable. SQL Server tiene dos tipos de variables: locales y globales.
- Las variables locales están definidas por el usuario, mientras que las variables globales las suministra el sistema y están predefinidas.

2.1. Variables globables

- Las variables globales son variables predefinidas suministradas por el sistema. Se distinguen de las variables locales por tener dos símbolos “@”.

Variable	Contenido
@@ERROR	Contiene o si la última transacción se ejecutó de forma correcta; en caso contrario, contiene el último número de error generado por el sistema. La variable global @@error se utiliza generalmente para verificar el estado de error de un proceso ejecutado.
@@IDENTITY	Contiene el último valor insertado en una columna IDENTITY mediante una instrucción insert
@@VERSION	Devuelve del SQL Server
@@SERVERNAME	Devuelve el Nombre del Servidor
@@LANGUAGE	Devuelve el nombre del idioma en uso
@@MAX_CONNECTIONS	Retorna la cantidad máxima de conexiones permitidas

2.1. Variables globables

```
/*Variables Globales*/
Use Negocios
go

--La version del SQL Server
Print 'Versión:' + @@version
--Lenguaje configurado en el Servidor
Print 'Lenguaje:' + @@language
--Nombre del servidor
Print 'Servidor:' + @@servername
--Nro. de conexiones permitidas
Print 'Conexiones:' + str(@@max_connections)
Go
```

2.2. Variables locales

- Las variables locales se declaran, nombran y escriben mediante la palabra clave declare, y reciben un valor inicial mediante una instrucción Select o Set.
- Los nombres de las variables locales deben empezar con el símbolo “@”. A cada variable local se le debe asignar un tipo de dato definido por el usuario o un tipo de dato suministrado por el sistema distinto de text, image o sysname.

2.2. Variables locales

- Sintaxis:

```
--Declara una variable
Declare @variable <tipo de dato>
-- Asigna valor a una variable
Set @variable= valor
```

2.2. Variables locales

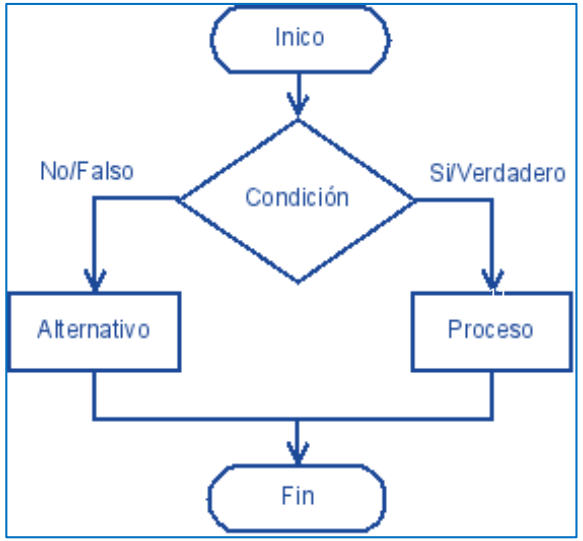
```
--Ejemplo 1
Declare @v_precio decimal
Set @v_precio = 50
Select * From Compras.Productos P
where P.precioUnidad > @v_precio
Go
```

```
--Ejemplo 2
Declare @mx decimal
Select @mx=MAX(precioUnidad) from Compras.productos
--imprimir los valores de las variables
print 'Mayor Precio mas alto es: '+str(@mx)
go
```

```
--Ejemplo 3
Declare @v_mn decimal
Select @v_mn = MIN(PrecioUnidad) From Compras.productos
Print 'El precio mas bajo es:'+str(@v_mn)
go
```

3. Estructura de control IF

- IF se utiliza para definir una condición que determina si se ejecutará la instrucción siguiente. La instrucción SQL se ejecuta si la condición se cumple, es decir, si devuelve TRUE (verdadero).
- La palabra ELSE introduce una instrucción SQL alternativa que se ejecuta cuando la condición IF devuelva FALSE.



3. Estructura de control IF

- Sintaxis:

```
IF(<expression>)
        BEGIN
            …
        END
ELSE IF (<expression>)
        BEGIN
            …
        END
ELSE
        BEGIN
            …
        END
```

4. Estructura de control CASE

```
Estructura IF*/
Use Negocios
go

/*Estructura*/
DECLARE @idemp int, @cantidad int
SET @idemp = 6
--Recuperar la cantidad de pedidos del empleado de codigo 6
SELECT @cantidad = COUNT(*)
FROM Ventas.pedidoscabe WHERE IdEmpleado = @idemp
--Evalua el valor de cantidad
IF    @cantidad =  0
      PRINT 'El empleado no ha realizado algún pedido'
ELSE IF @cantidad =  1
      PRINT 'Ha registrado 1 pedido, continúe trabajando'
 ELSE
      PRINT 'Ha registrado muchos pedidos'
 GO
```

4. Estructura de control CASE

- La estructura CASE evalúa una lista de condiciones y devuelve una de las varias expresiones de resultado posibles.
- La expresión CASE tiene dos formatos:
  - La expresión CASE sencilla compara una expresión con un conjunto de expresiones sencillas para determinar el resultado.
  - La expresión CASE buscada evalúa un conjunto de expresiones booleanas para determinar el resultado.
- Ambos formatos admiten un argumento ELSE opcional.

4. Estructura de control CASE

- Sintaxis:

```
CASE <expresión>
        WHEN <valor_expresion> THEN <valor_devuelto>
        WHEN <valor_expresion1> THEN <valor_devuelto1>
        ELSE <valor_devuelto2>  -- Valor por defecto
END
```

4. Estructura de control CASE

```
 /*CASE: Evaluando valores*/
Begin
    Declare @v_nroMensaje Tinyint =1
    Declare @v_txtMensaje varchar (max )
    Set @v_txtMensaje = Case     @v_nroMensaje

                        When 1Then  'Hello world'
                        When 2 Then  'Vas a lograrlo'
                        When 3 Then  'Eres un(a) Campeón(a)'
                        Else 'Mensaje no implementado'
                    End Print
    @v_txtMensaje
 End


 go
```

4. Estructura de control CASE

```
 /*CASE: Evaluando resultado de una expresión de comparación*/
Begin
      Set Dateformat DMY
      Declare @v_fna DATE = '10/04/90'
      Declare @v_edad Smallint
      Declare @v_etapaGeneracional varchar(50)
      Set @v_edad = DateDiff(yy,@v_fna,getdate())
      Set @v_etapaGeneracional =  Case
                                      When @v_edad<1 Then 'Bebe'
                                      When @v_edad<=5 Then
                                      'Infante' When @v_edad<=12
                                      Then 'Niño' When @v_edad<=14
                                      Then 'Pubeda' When @v_edad<18 Then 'Ad olescente
                                      '
                                      When @v_edad<=30 Then 'Joven'
                                      When @v_edad<65 Then 'Ad ulto
                                      ' Else 'Adulto Mayor'
                                  End
      Print 'La etapa generacional es: '+@v_etapaGeneracional
 End
 go
```

4. Estructura de control CASE

```
 /*Utilizando CASE dentro de un SELECT*/
*Ejemplo 1*/


 --Optimizando con CASE
select  IdPedido,
            FechaPedido,
            Destinatario,
            Case EnvioPedido
                When 0 Then  'Envio Pendiente'
                When 1 Then  'Ya Enviado'
            End As  [Estado del Envio]
 from Ventas.pedidoscabe
 go
```

4. Estructura de control CASE

```
/*Ejemplo 2*/
DECLARE @stock int
SET @stock=100
SELECT  NomProducto,
        PrecioUnidad,
        UnidadesEnExistencia,
        'Estado'= (CASE
                    WHEN UnidadesEnExistencia >@stock THEN 'Stockeado'
                    WHEN UnidadesEnExisten cia=@stock THEN 'Limite'
                    WHEN UnidadesEnExistencia <@stock THEN 'Haga una Solicitud'
                   END)
FROM Compras.productos
GO
```

5. Estructura de control WHILE

- Ejecuta en forma repetitiva un conjunto o bloque de instrucciones SQL 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.
- Sintaxis:

```
WHILE <expresion>
   BEGIN
      ...
   END
```

Estructura de control WHILE

```
Use Negocios
go

/*Crear estructura WHILE*/
Begin
    Set DateFormat DMY
    Declare @v_fechalnicial Date='01/01/90.
    Declare @v_fechaFinal Date='31/12/00'
    While @v_fechalnicial <= @v_fechaFinal
        Begin
            Print @v_fechainicial
            Set @v_fechainicial=DateAdd(dd,1,@v_fechainicial)
        End
End
go
```

5. Estructura de control WHILE

```
/*Crear estructura WHILE*/
Begin
    Set DateFormat DMY
    Declare @v_fechalnicial Date='01/01/90'
    Declare @v_fechaFinal Date='31/12/00'
    Declare @v_Flag  bit =1
    While @v_Flag = 1
        Begin
            Print @v_fechalnicial
            Set @v_fechainicial=DateAdd(ddJ1J@v_fechainicial)
            If @v_fechalnicial > @v_fechaFinal
                Break
            Else
                Continue
        End
End
go
```

6. Control de errores

- SQL Server proporciona el control de errores a través de las instrucciones TRY y CATCH. Estas nuevas instrucciones suponen un gran paso adelante en el control de errores en SQL Server.
- La sintaxis de TRY CATCH es la siguiente:

```
BEGIN TRY
        EXPRESION_SQL
END TRY
BEGIN CATCH
        EXPRESION_SQL
END CATCH
```

6. Control de errores

TRY / CATCH

- Si tratamos de eliminar un registro Cliente que está siendo usado en la tabla Pedidos, nos va a provocar un error de Integridad Referencial.

```
/*Eliminar un registro*/
Delete From Ventas.clientes
where IdCliente = 'ALFKI'
go
```

Messages

```
Msg 547, Level 16, State 0, Line 181
Instrucción DELETE en conflicto con la restricción REFERENCE 'FK__pedidosca__IdCli__4CA06362'.
Se terminó la instrucción.
```

6. Control de errores

TRY / CATCH

- La misma sentencia, pero con Excepción de Errores.

```
/*Manejando excepción de errores*/
Begin Try
    Delete From Ventas.clientes
    where IdCliente = 'ALFKI'
End Try
Begin Catch
    Print 'Error excepcionado'
End Catch
go
```

Messages

```
(0 rows affected)
Error excepcionado
```

6. Control de errores

TRY / CATCH

```
/*Manejando excepción de errores Usando @@Error*/
Begin Try
    Delete From Ventas.clientes
    where IdCliente = 'ALFKI'
End Try
Begin Catch
    If @@error = 547
            Print 'No se puede eliminar por integridad referencial'
End Catch
go
```

Messages

```
(0 rows affected)
No se puede eliminar por integridad referencial
```

6. Control de errores

Funciones especiales de errores

- Las funciones especiales de error, están disponibles únicamente en el bloque CATCH para la obtención de información detallada del error:

Error	Descripción
ERROR_NUMBER()	Devuelve el numero de error
ERROR_SEVERITY()	Devuelve la severidad del error
ERROR_STATE()	Devuelve el estado del error
ERROR_PROCEDURE()	Devuelve el nombre del procedimiento almacenado que ha provocado el error
ERROR_LINE()	Devuelve el número de línea en la que se ha producido el error.
ERROR_MESSAGE()	Devuelve el mensaje de error

6. Control de errores

Variable @@ERROR

- Devuelve el número de error de la última instrucción TRANSACT-SQL ejecutada; si la variable devuelve 0, la TRANSACT-SQL anterior no encontró errores.

```
DELETE FROM Ventas.clientes
WHERE IdCliente = 'ALFKI'
IF @@ERROR<>0
    PRINT 'No se puede eliminar'
GO
```

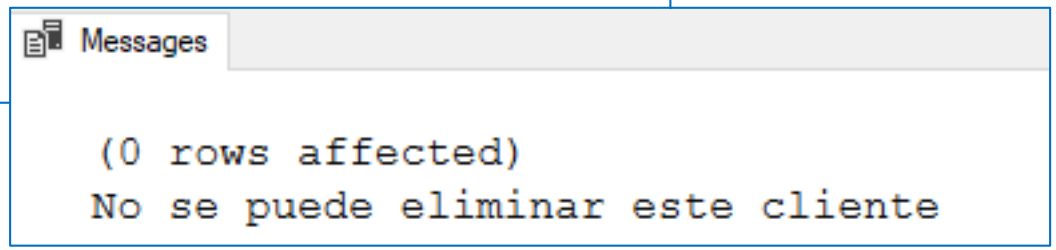
```
Messages
Msg 547, Level 16, State 0, Line 206
Instrucción DELETE en conflicto con la restricción REFERENCE 'FK__pedidosca__IdCli__4CA06362'.
Se terminó la instrucción.
No se puede eliminar
```

6. Control de errores

Variable @@ERROR

- Otra forma de implementar este proceso, es controlándolo a través del bloque TRY CATCH.

```
BEGIN TRY
    Delete From Ventas.clientes
    where IdCliente = 'ALFKI'
END TRY
BEGIN CATCH
    If @@ERROR=547
    PRINT 'No se puede eliminar este cliente'
END CATCH
GO
```



6. Control de errores

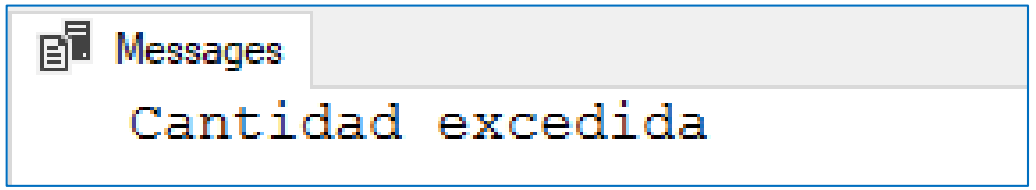
RaisError

- En ocasiones es necesario provocar voluntariamente un error, por ejemplo nos puede interesas que se genere un error cuando los datos incumplen una regla de negocio.
- Podemos provocar un error en tiempo de ejecución a través de la función RAISERROR.
- La función RAISERROR recibe tres parámetros, el mensaje del error (o código de error predefinido), la severidad y el estado.

6. Control de errores

RaisError

```
 /*Personalizando mensajes, con severidad Leve (10)*/
Declare @v_CantitadPedida smallint =110
If @v_CantitadPedida>=100
    RaisError('Cantidad excedida',10,1)
go
```



6. Control de errores

RaisError

```
 /*Personalizando mensajes con severidad Grave (16)*/
Declare @v_CantitadPedida smallint =110
If @v_CantitadPedida>=100
     RaisError('Cantidad excedida',16,1)
go
```

Messages

```
Msg 50000, Level 16, State 1, Line 14
Cantidad excedida
```

6. Control de errores

RaisError.-

```
/*Personalizando mensajes con severidad 16 controlando con TRY CATCH*/
Declare @v_CantitadPedida smallint =110
Begin Try
    If @v_CantitadPedida>=100
        RaisError('Cantidad excedida',16,1)
End Try
Begin Catch
        Print error_message()
        Print 'Error controlado con Catch'
End Catch
go
```

Messages

```
Cantidad excedida
Error controlado con Catch
```

7. Transacciones

- Una transacción es un conjunto de operaciones TRANSACT SQL que se ejecutan como un único bloque, es decir, si falla una operación TRANSACT SQL fallan todas.
- Si una transacción tiene éxito, todas las modificaciones de los datos realizadas durante la transacción se confirman y se convierten en una parte permanente de la base de datos.
- Si una transacción encuentra errores y debe cancelarse o revertirse, se borran todas las modificaciones de los datos.

7. Transacciones

Método de Transacciones

- Para agrupar varias sentencias TRANSACT SQL en una única transacción, disponemos de los siguientes métodos:
  - Transacciones explícitas: Cada transacción se inicia explícitamente con la instrucción BEGIN TRANSACTION y se termina explícitamente con una instrucción COMMIT o ROLLBACK.
  - Transacciones implícitas: Se inicia automáticamente una nueva transacción cuando se ejecuta una instrucción que realiza modificaciones en los datos, pero cada transacción se completa explícitamente con una instrucción COMMIT o ROLLBACK.

7. Transacciones

Sintaxis para el control de las transacciones

```
BEGIN TRAN NombreTransaccion --Inicio de transacción con su nombre.
   /*Bloque de instrucciones a ejecutar en la Transacción
   ------------------------------------
   ------------------------------------*/
COMMIT TRAN NombreTransaccion   --Confirmación de la transacción.
ROLLBACK TRAN NombreTransaccion --Reversión de la transacción.
```

7. Transacciones

```
USE NEGOCIOS
GO

DECLARE @V_IDPAIS CHAR (3) = '99'
DECLARE @V_NOMPAIS VARCHAR (50) = 'UGANDA'
BEGIN TRAN MITRANSACCION
        UPDATE VENTAS.PAISES
        SET NOMBREPAIS = @V_NOMPAIS
        WHERE IDPAIS = @V_IDPAIS
        IF @V_IDPAIS > 80
                ROLLBACK TRAN MITRANSACCION
        ELSE
                COMMIT TRAN MITRANSACCION
GO
```

Ejercicio

- Efectúe una transacción explícita para registrar nuevos Clientes, en caso el nombre del Cliente no se repita, se confirma la transacción, de lo contrario, genere un error de severidad grave, controlándolo con Try Catch y deshacer.
- Controle los errores de duplicidad de llave primaria y llave foránea..

Ejercicio: Solución

- Solución:

```
Set Nocount On
Begin Try
    Begin Transaction TR001
        Declare @v_IdCli char(5)        = 'AOOOI'
        Declare @v_nomCli varchar(max)  = 'Alfredo Kimball'
        Declare @v_dirCli varchar(max)  = 'Ca Paujile 123'
        Declare @v_idPai char(3)        = '001'
        Declare @v_fonCli varchar(25)   = '98765567'
        Declare @v_tabNom Table (nomCli varchar(455) )
        Insert into @v_tabNom
        Select NomCliente from Ventas.clientes
        Insert into Ventas.clientes
        Values
        (@v_IdCli, @v_nomCli, @v_dirCli, @v_idPai, @v_fonCli)
        If not exists (Select * from @v_tabNom where nomCli = @v_nomCli)
            Commit Transaction TR001
        Else
            RaisError ('Datos de nombre ya existe',16,1)
End Try
Begin Catch
    If ERROR_NUMBER() = 2627
        Print 'Error de IdCliente Repetido'
    If ERROR_NUMBER() = 547
        Print 'Error de NomCli  no existe'
    Else
        Begin
            Rollback Tran TR001
            Print error_message()
        End
End Catch
GO
```

Conclusiones

- Transact-SQL es fundamental para usar Microsoft SQL Server. Todas las aplicaciones que se comunican con una instancia de SQL Server lo hacen mediante el envío de instrucciones de Transact-SQL al servidor.
- Una transacción es un conjunto de operaciones **Transact SQL** que se ejecutan como un único bloque.
- RaisError genera un mensaje de error e inicia el procesamiento de errores para la sesión. RAISERROR puede hacer referencia a un mensaje definido por el usuario.

Bibliografías

- Microsoft (2017) RAISERROR (Transact-SQL). Recuperado de: https://docs.microsoft.com/en-us/sql/t-sql/language- elements/raiserror-transact-sql?view=sql-server-2017

- Microsoft. (2017). Transacciones en Transact SQL (Transact- SQL). Recuperado de: http://www.devjoker.com/contenidos/catss/292/Transacciones- en-Transact-SQL.aspx

GRACIAS

SEDE LIMA CENTRO
Av. Uruguay 514
Cercado – Lima
Teléfono: 419-2900

SEDE INDEPENDENCIA
Av. Carlos Izaguirre 233
Independencia – Lima
Teléfono: 633-5555

SEDE BREÑA
Av. Brasil 714 – 792
(CC La Rambla – Piso 3)
Breña – Lima
Teléfono: 633-5555

SEDE TRUJILLO
Calle Borgoño 361
Trujillo
Teléfono: (044) 60-2000

SEDE SAN JUAN DE LURIGANCHO
Av. Próceres de la Independencia 3023-3043
San Juan de Lurigancho – Lima
Teléfono: 633-5555

SEDE BELLAVISTA
Av. Mariscal Oscar R. Benavides 3866 – 4070
(CC Mall Aventura Plaza)
Bellavista – Callao
Teléfono: 633-5555

SEDE AREQUIPA
Av. Porongoche 500
(CC Mall Aventura Plaza)
Paucarpata - Arequipa
Teléfono: (054) 60-3535