Base de Datos I

Manual de Laboratorios

Universidad Don Bosco

Facultad de Ingeniería.
Escuela de computación.

Sentencias complementarias + Disparadores

Objetivos:

- Elaborar sentencias especiales con diferentes usos y componentes.

Introducción:

Siempre hay tipos de consultas o transacciones especiales que se usan para realizar copiados de datos, referencias especiales y manejo de estructuras de control y otros.

COPIADO DE TABLAS

Una actividad muy común es el copiado de tablas, tomando todos los datos de una tabla o una parte de ella, hay que tener en cuenda que el tipo de datos de origen y destino deben ser del mismo tipo y tamaño.

SENTENCIA IF

Uno de los usos más conocidos para modificar el flujo de un programa es el **IF** y su modificar ELSE.

SENTENCIA CASE

La sentencia case de SQL es una estructura condicional que ejecuta ciertas instrucciones dependiendo de la validez de alguna expresión. Normalmente se usa cuando surgen múltiples condiciones donde se compara una expresión con una constante literal. Por lo que se considera una alternativa a la sentencia if en estos casos.

Procedimiento:

PARTE - A

Realiza este ejemplo de copiado total y parcial.

```
CREATE DATABASE CopiadoA
GO

USE CopiadoA

-- TABLAS

CREATE TABLE ALUMNOFUENTE(
        NOTA1 INT,
        NOTA2 INT,
        NOTA3 INT)

CREATE TABLE ALUMNOCPYTOTAL(
        NOTA1 INT,
        NOTA2 INT,
        NOTA3 INT)

CREATE TABLE ALUMNOCPYPARCIAL(
        NOTA1 INT,
        NOTA2 INT,)


--INSERCIÓN EN TABLA FUENTE

INSERT INTO ALUMNOFUENTE VALUES (6,7,8)
INSERT INTO ALUMNOFUENTE VALUES (5,5,10)
INSERT INTO ALUMNOFUENTE VALUES (6,0,9)
INSERT INTO ALUMNOFUENTE VALUES (9,3,4)
```

Ahora visualice las tres tablas y vera que una está completa y las otras dos están vacías, ejecute las siguientes consultas para realizar un copiado de datos.

```
--COPIADO DE TABLAS

--COPIADO TOTAL
INSERT INTO ALUMNOCPYTOTAL SELECT * FROM ALUMNOFUENTE

--COPIADO PARCIAL
INSERT INTO ALUMNOCPYPARCIAL (NOTA1,NOTA2) SELECT NOTA1, NOTA3 FROM ALUMNOFUENTE
```

Nuevamente visualice las tres tablas y vera que se han realizado la copia total y parcial de datos.

Por ultimo otra forma de copiado de tablas es la forma de copiar tanto los datos como la estructura que tiene esta, creando una tabla exactamente igual.

```
SELECT * INTO <TABLA A CREAR> FROM <TABLA A COPIAR>

SELECT * INTO ALUMNOS FROM ALUMNOFUENTE

SELECT * FROM ALUMNOS
```

A continuación un ejemplo del uso del **IF** para evitar que una tabla se coloque una fecha posterior menor a una inicial, por lógica la segunda fecha debería ser mayor que la primera.

```
CREATE TABLE agenda(
id int IDENTITY(1,1),
evento varchar(80),
fecini date,
fecfin date
)

CREATE PROCEDURE SumaEvento
@evento varchar(80),
@ini date,
@fin date
AS
BEGIN
IF @fin > @ini
  INSERT INTO agenda VALUES (@evento,@ini,@fin)
else
 PRINT 'El intervalo de fechas esta erroneo'
END

execute SumaEvento 'Reunion','1-4-2016','1-2-2016'
execute SumaEvento 'Reunion','1-1-2016','1-4-2016'
```

El siguiente ejemplo muestra como actualizar una tabla con el uso de case.

```
CREATE TABLE vendedor (
vendedor varchar(25),
sueldo int,
tot_ventas int,
sucursal int
)

INSERT INTO vendedor VALUES ('Efrain Burgos',200,345.56,1),
('Carlos Olmos',200,2345.00,1),('Efrain Burgos',200,509.87,2),
('Karla Alas',200,3123.05,3),('Carlos Olmos',200,323.76,3),
('Efrain Burgos',200,800.00,2),('Carlos Olmos',200,654.43,2),
 ('Karla Alas',200,123.34,4),('Carlos Olmos',200,450.34,5)


UPDATE vendedor
SET sueldo = CASE
WHEN tot_ventas BETWEEN 100.00 AND 700.00 THEN
 sueldo + tot_ventas * 0.02
WHEN tot_ventas BETWEEN 700.00 AND 2000.00 THEN
 sueldo + tot_ventas * 0.03
WHEN tot_ventas >=2001.00 THEN
 sueldo +tot_ventas * 0.07
END

SELECT * FROM vendedor
```

Otro ejemplo del uso del CASE es:

```
SELECT vendedor, sueldo, sucursal, CASE sucursal
 WHEN 1 THEN 'Santa Ana'
 WHEN 2 THEN 'Sonsonate'
 WHEN 3 THEN 'San Salvador'
 WHEN 4 THEN 'La Libertad'
 WHEN 5 THEN 'Ahuchapan'
 ELSE 'No asignado'
END as Sucursal
 FROM vendedor
```

PARTE - B

Introducción:

Un disparador (trigger o desencadenador) es una clase especial de procedimiento almacenado que se ejecuta automáticamente cuando se produce un evento en el servidor de bases de datos.

SQL Server proporciona los siguientes tipos de triggers:

- **Trigger DML**, se ejecutan cuando un usuario intenta modificar datos mediante un evento de lenguaje de manipulación de datos (DML). Los eventos DML son instrucciones INSERT, UPDATE o DELETE de una tabla o vista.

- **Trigger DDL**, se ejecutan en respuesta a una variedad de eventos de lenguaje de definición de datos (DDL). Estos eventos corresponden principalmente a instrucciones CREATE, ALTER y DROP de Transact-SQL, y a determinados procedimientos almacenados del sistema que ejecutan operaciones de tipo DDL.

Trigger DDL

Los trigger DDL se ejecutan en respuesta a una variedad de eventos de lenguaje de definición de datos (DDL). Estos eventos corresponden principalmente a instrucciones CREATE, ALTER y DROP de Transact-SQL, y a determinados procedimientos almacenados del sistema que ejecutan operaciones de tipo DDL

La sintaxis general de un trigger es la siguiente.

```
CREATE TRIGGER <trigger_name, sysname, table_alter_drop_safety>

ON DATABASE

FOR <data_definition_statements, , DROP_TABLE, ALTER_TABLE>

AS

BEGIN



END
```

Manejo de datos

Cuando se quiere agregar, actualizar o eliminar datos de una base de datos, se ejecutan sentencias DML. Una colección de sentencias DML que forman una unidad lógica de trabajo es llamada transacción.
Considere una base de datos de un banco. Cuando un cliente del banco transfiere dinero de su cuenta de ahorros a su cuenta de cheques, la transacción puede consistir de tres

operaciones separadas:

1. Decrementar la cuenta de ahorros.
2. Incrementar la cuenta de cheques.
3. Registrar la transacción en la bitácora de transacciones

Sentencias DML	
Insert	Añade filas de datos en una tabla
Delete	Eliminar filas de datos de una tabla
Update	Modificar los datos de una tabla
Select	Recuperar datos de una tabla
Commit	Confirmar como permanentes las modificaciones realizadas.
Rollback	Deshacer todas las modificaciones realizadas desde la última confirmación.

Transacción

Una transacción es un conjunto de operaciones que van a ser tratadas como una única unidad. Estas transacciones deben cumplir 4 propiedades fundamentales comúnmente conocidas como ACID (atomicidad, coherencia, asilamiento y durabilidad).

La transacción más simple en SQL Server es una única sentencia SQL. Por ejemplo:

UPDATE Products SET UnitPrice=20 WHERE ProductName ='Chai'

Esta es una transacción 'autocommit', una transacción autocompletada(commint implícito).

Cuando enviamos esta sentencia al SQL Server se escribe en el fichero de transacciones lo que va a ocurrir y a continuación realiza los cambios necesarios en la base de datos. Si hay algún tipo de problema al hacer esta operación el SQL Server puede leer en el fichero de transacciones lo que se estaba haciendo y si es necesario puede devolver la base de datos al estado en el que se encontraba antes de recibir la sentencia.

Por supuesto este tipo de transacciones no requieren de nuestra intervención puesto que el sistema se encarga de todo. Sin embargo si hay que realizar varias operaciones y queremos que sean tratadas como una unidad tenemos que crear esas transacciones de manera explícita.

Sentencias para una transacción

Una transacción es un conjunto de operaciones tratadas como una sola. Este conjunto de operaciones debe marcarse como transacción para que todas las operaciones que la conforman tengan éxito o todas fracasen.

La sentencia que se utiliza para indicar el comienzo de una transacción es 'BEGIN TRAN'. Si alguna de las operaciones de una transacción falla hay que deshacer la transacción en

su totalidad para volver al estado inicial en el que estaba la base de datos antes de empezar. Esto se consigue con la sentencia 'ROLLBACK TRAN'.

Si todas las operaciones de una transacción se completan con éxito hay que marcar el fin de una transacción para que la base de datos vuelva a estar en un estado consistente con la sentencia 'COMMIT TRAN'.

Procedimiento:

PARTE - B

A continuación digitaremos las siguientes instrucciones y las ejecutaremos

```
CREATE DATABASE TIENDA
go
USE TIENDA
    --Creación de tablas
    Create table informacion_tienda
    (
    tienda  varchar(50),
    venta   float,
    fecha datetime,
    informacion varchar(50),
    tipopago varchar(25)
    )

    Create table ventas_tienda
    (
    tienda  varchar(50),
    venta   float,
    fecha datetime
    )
```

```sql
--INSERTANDO DATOS
INSERT INTO informacion_tienda(tienda,venta,fecha,informacion,tipopago)
VALUES('San Salvador',200,'1998-02-02','Linea blanca','efectivo')
INSERT INTO informacion_tienda(tienda,venta,fecha,informacion,tipopago)
VALUES('Soyapango',300,'1998-03-22','granos basicos','tarjeta')
INSERT INTO informacion_tienda(tienda,venta,fecha,informacion,tipopago)
VALUES('Mejicanos',220,'2011-01-01','medicinas','efectivo')
INSERT INTO informacion_tienda(tienda,venta,fecha,informacion,tipopago)
VALUES('Mejicanos',100,'2010-06-18','productos para el hogar','tarjeta')
```

Digite: `Select * from información_tienda`, para verificar que los datos están ingresados

Digite: `Select * from ventas_tienda`, verifique que no existe ningún valor en esa tabla

Ejecute el siguiente script y luego revise la tabla ventas_tiendas

```sql
INSERT INTO ventas_tienda (tienda,venta,fecha) SELECT tienda,venta,fecha
FROM informacion_tienda
```

ACTUALIZAR

Agreguemos un campo ventaiva

```sql
ALTER TABLE ventas_tienda
ADD ventainva float
```

Inicialmente ese campo aparece vacio en todas las columnas y queremos que ese campo contenga la venta + el 13% del iva, para esto utilizamos la siguiente sentencia.

```sql
UPDATE ventas_tienda SET ventaiva=venta*1.13 OUTPUT INSERTED.*
```

Por qué se utiliza la opción OUTPUT INSERTED?

USO DE VARIABLES

Una variable es un valor identificado por un nombre (identificador) sobre el que podemos realizar modificaciones.

En Transact SQL los identificadores de variables deben comenzar por el carácter @, es decir, el nombre de una variable debe comenzar por @. Para declarar variables en Transact SQL debemos utilizar la palabra clave declare, seguido del identificador y tipo de datos de la variable.

Ejemplo

```
declare @nombre varchar(50)
set @nombre = 'Utilizando variables en SQL'
print @Nombre -- Imprime por pantalla el valor de @nombre.
```

Ejemplo 2. Asignando variables en T-SQL, podemos utilizar SET para asignar el resultado de una consulta (la variable solo podrá contener un único valor)

```
declare @venta float
SET @venta = (SELECT venta from ventas_tienda where tienda='Soyapango')
PRINT @venta
```

Ejemplo 3. Podemos utilizar SELECT para asignar valores

```
DECLARE @tienda VARCHAR(100),@venta float,@fecha datetime
SELECT @tienda=tienda,@venta=venta,@fecha=fecha
FROM ventas_tienda where tienda='Soyapango'
PRINT 'el resultado de la asignacion es el siguiente'
PRINT @tienda
PRINT @venta
PRINT @fecha
```

Disparadores

Digite la siguientes instrucciones y ejecútelas

```
CREATE TRIGGER Disp_SEGURIDAD

ON DATABASE FOR DROP_TABLE, ALTER_TABLE

AS

BEGIN

      RAISERROR ('No está permitido borrar ni modificar tablas !' , 16, 1)

      ROLLBACK TRANSACTION

END
```

Estas instrucciones crean un disparador que se activará en el momento que el usuario desee: eliminar o modificar una tabla.
Intente eliminar la tabla ventas_tienda
Para eliminar un disparador se utiliza la siguiente instrucción

```
DROP  TRIGGER nombretrigger ON DATABASE
```

TRANSACCIONES

Como se mencionó en la introducción una transacción es un conjunto de operaciones tratadas como una sola. Este conjunto de operaciones debe marcarse como transacción para que todas las operaciones que la conforman tengan éxito o todas fracasen.
La sentencia que se utiliza para indicar el comienzo de una transacción es 'BEGIN TRAN'. Si alguna de las operaciones de una transacción falla hay que deshacer la transacción en su totalidad para volver al estado inicial en el que estaba la base de datos antes de empezar. Esto se consigue con la sentencia 'ROLLBACK RAN'. Si todas las operaciones de una transacción se completan con éxito hay que marcar el fin de una transacción para que la base de datos vuelva a estar en un estado consistente con la sentencia 'COMMIT TRAN'
Ejemplo. Cree la siguiente tabla prueba e inserte el dato que aparece

```
--TABLA PRUEBA
CREATE TABLE PRUEBA
(nombre varchar(25), apellido varchar(25))
--DATOS
insert into PRUEBA values('carlos','alfaro')
```

Al realizar una consulta aparece efectivamente el dato

Ahora insertaremos 2 nuevos datos, pero esta vez dentro de una transacción

```
BEGIN TRAN
insert into PRUEBA values('Mario','Mendez')
insert into PRUEBA values('Marcela','Abarca')
```

Estos datos si realizamos un select * from prueba, se mostraran como insertados en la tabla, pero si deseamos revertir la inserción digitamos

ROLLBACK TRAN

Verificamos nuevamente la tabla y vemos que la inserción de datos se ha revertido, si deseamos que los datos se inserten digitamos COMMIT TRAN, pero si anteriormente habíamos utilizado ROLLBACK nos enviará un mensaje de error: The COMMIT TRANSACTION request has no corresponding BEGIN TRANSACTION.

Si desea que los datos sean insertados de una sola vez tendrá que digitarlo de la siguiente manera

```
BEGIN TRAN
insert into PRUEBA values('Mario','Mendez')
insert into PRUEBA values('Marcela','Abarca')
COMMIT TRAN
```

Ejemplo 2

```
declare @venta float, @tienda varchar(25),@buscar varchar(25)
set @buscar='Soyapango'
Begin tran
select @venta=venta,@tienda=tienda from ventas_tienda where
tienda=@buscar
--Insertando datos utilizando variables
insert into informacion_tienda values(@tienda,@venta+100,'2011-02-
21','Secreto','credito')
--concatenando valores
print 'Tienda'+ ' '+@tienda
--no se puede concatenar texto con numeros hay que hacer conversión
print 'venta'+ ' '+CAST(@venta as varchar)
COMMIT TRAN
```

Ejemplo 3

```
--insertando datos en tabla ventas tienda utilizando variables
declare @tienda varchar(25), @venta float, @fecha datetime, @ventaiva
float
set @tienda='Cuscatancingo'
set @venta=100
set @fecha='2010-02-04'
set @ventaiva=@venta*1.13
begin tran
INSERT INTO ventas_tienda values(@tienda,@venta,@fecha,@ventaiva)
COMMIT TRAN
PRINT 'VALORES INSERTADOS CON EXITO'
PRINT 'TIENDA'+@tienda
```

Análisis de resultados:

- Cree una base de datos de nombre ALMACEN con las siguientes tablas:

Productos

Column Name	Data Type
codproducto	char(6)
nombreproducto	varchar(25)
existencia	int
precio	float

pedidos

Column Name	Data Type
codpedido	int
cantidad	int
fechapedido	datetime
codproducto	char(6)

Inserte los siguientes datos a la tabla productos

	codproducto	nombreproducto	existencia	precio
1	prod01	jabon	100	4.5
2	prod02	lejia	200	2.5
3	prod03	paste	400	6.5
4	prod04	shapoo	211	9
5	prod05	japon de traste	50	3.5
6	prod06	pasta de zapato	400	2.4
7	prod07	pelota	50	3.5
8	prod08	tomates	100	0.25

- Cree un disparador que se active al momento de eliminar o modificar la base de datos almacen
- Cree una transacción parecida al ejemplo 3 para insertar productos