Taller	SQL SERVER PROGRAMACIÓN
Docente	Mg. Ing. Eric Gustavo Coronel Castillo
Tema	**CONTROL DE ERROES**

Control de Errores

Variable: @@ROWCOUNT

Devuelve el número de filas afectadas por la última instrucción. Si el número de filas es mayor de 2 mil millones, use ROWCOUNT_BIG.

```
@@ROWCOUNT
```

Más información en: http://technet.microsoft.com/es-pe/library/ms187316.aspx

Función: ROWCOUNT_BIG ( )

Devuelve el número de filas afectadas por la última instrucción ejecutada. Esta función actúa como @@ROWCOUNT, pero el tipo de valor devuelto de ROWCOUNT_BIG es bigint.

```
ROWCOUNT_BIG ( )
```

Más información en: http://technet.microsoft.com/es-es/library/ms181406.aspx

Variable: @@ERROR

Devuelve el número de error de la última instrucción Transact-SQL ejecutada.

```
@@ERROR
```

Más información en: http://technet.microsoft.com/es-es/library/ms188790.aspx

Función: RAISERROR ( )

Genera un mensaje de error e inicia el procesamiento de errores de la sesión. RAISERROR puede hacer referencia a un mensaje definido por el usuario almacenado en la vista de catálogo **sys.messages** o puede generar un mensaje dinámicamente. El mensaje se devuelve como un mensaje de error de servidor a la aplicación que realiza la llamada o a un bloque CATCH asociado de una construcción TRY…CATCH.

Las nuevas aplicaciones deben utilizar THROW en su lugar.

```
RAISERROR ( { msg_id | msg_str | @local_variable }
   { ,severity ,state }
   [ ,argument [ ,...n ] ] );
```

Más información en: http://msdn.microsoft.com/es-es/library/ms178592.aspx

Manejo de Excepciones

Estructura TRY/CATCH

Una mejora importante que tenemos en SQL Server es el manejo de errores que ahora es posible en T-SQL con los bloques TRY/CATCH sin olvidar la sintaxis que utilizamos para las transacciones.

Sintaxis:

```
BEGIN TRY
   BEGIN TRANSACTION;

   -- Bloque de código SQL a proteger

    COMMIT TRANSACTION;
END TRY
BEGIN CATCH
   ROLLBACK TRANSACTION;
   -- Código para mostrar el mensaje de la  excepción
END CATCH;
```

Si ocurre un error dentro de la transacción en un bloque TRY inmediatamente se dirige al bloque CATCH.

Para poder acceder a la información del error tenemos las siguientes funciones:

- **ERROR_NUMBER()** : Devuelve el número de error.
- **ERROR_MESSAGE()** : Devuelve el texto completo del mensaje de error. El texto incluye los valores suministrados para los parámetros sustituibles, como longitudes, nombres de objeto u horas.
- **ERROR_SEVERITY()** : Devuelve la gravedad del error.
- **ERROR_STATE()** : Devuelve el número de estado de error.
- **ERROR_LINE()** : Devuelve el número de línea donde se produjo el error.
- **ERROR_PROCEDURE()** : Devuelve el nombre del procedimiento donde se produjo el error.

Pasemos a crear un ejemplo bastante sencillo en donde vamos a provocar una excepción en una división por cero para observar el resultado de estas funciones.

Más información en: http://technet.microsoft.com/es-pe/library/ms175976.aspx

Ejemplo 1: Captura de error

```
BEGIN TRY
   DECLARE @TOTAL INT;
   SET @TOTAL = 20;
   SELECT @TOTAL/0
END TRY
BEGIN CATCH
   SELECT
      ERROR_NUMBER() AS Numero_de_Error,
      ERROR_SEVERITY() AS Gravedad_del_Error,
      ERROR_STATE() AS Estado_del_Error,
      ERROR_PROCEDURE() AS Procedimiento_del_Error,
      ERROR_LINE() AS Linea_de_Error,
      ERROR_MESSAGE() AS Mensaje_de_Error;
END CATCH;
GO
```

Un bloque CATCH no controla los siguientes tipos de errores cuando se producen en el mismo nivel de ejecución que la construcción TRY…CATCH:

- Errores de compilación, como errores de sintaxis, que impiden la ejecución de un lote.
- Errores que se producen durante la recompilación de instrucciones, como errores de resolución de nombres de objeto que se producen después de la compilación debido a una resolución de nombres diferida.

Estos errores se devuelven al nivel de ejecución del lote, procedimiento almacenado o desencadenador.

En el **Ejemplo 2** se muestra cómo la construcción TRY…CATCH no captura un error de resolución de nombre de objeto generado por una instrucción SELECT, sin embargo, el bloque CATCH si captura el error cuando la misma instrucción SELECT se ejecuta dentro de un procedimiento almacenado, tal como se puede apreciar en el **Ejemplo 3**.

Ejemplo 2: Error no capturado

```
BEGIN TRY
    SELECT * FROM TablaNoExiste;
END TRY
BEGIN CATCH
    SELECT ERROR_NUMBER() AS ErrorNumber,
           ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO

Mens. 208, Nivel 16, Estado 1, Línea 2
El nombre de objeto 'TablaNoExiste' no es válido.
```

El error no se captura y el control se transfiere fuera de la construcción TRY…CATCH, al siguiente nivel superior.

Al ejecutar la instrucción SELECT dentro de un procedimiento almacenado, el error se produce en un nivel inferior al bloque TRY. La construcción TRY…CATCH controlará el error.

Ejemplo 3: Captura de error de un nivel inferior

```
USE EDUCA;
GO

IF OBJECT_ID ( N'usp_ProcEjemplo', N'P' ) IS NOT NULL
    DROP PROCEDURE usp_ExampleProc;
GO

CREATE PROCEDURE usp_ProcEjemplo
AS
    SELECT * FROM TablaNoExiste;
GO

BEGIN TRY
    EXECUTE usp_ProcEjemplo;
END TRY
BEGIN CATCH
    SELECT ERROR_NUMBER() AS ErrorNumber,
           ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO

ErrorNumber ErrorMessage
----------- ------------------------------------------------------------
208         El nombre de objeto 'TablaNoExiste' no es válido.
```

```
(1 filas afectadas)
```

Más información en: http://technet.microsoft.com/es-pe/library/ms175976.aspx

Sentencia: THROW

Genera una excepción y transfiere el control a un bloque CATCH de una estructura TRY...CATCH..

Sintaxis:

```
THROW [ { error_number | @local_variable },
        { message | @local_variable },
    { state | @local_variable }
] [ ; ]
```

Ejemplo 4: Generar una nueva excepción

```
THROW 51000, 'The record does not exist.', 1;
```

Ejemplo 5: Propagar la última excepción

```
BEGIN TRY

    -- Bloque a controlar

END TRY
BEGIN CATCH

    -- Proceso de control

    THROW; -- Propaga la última excepción
END CATCH;
```

Más información en: http://technet.microsoft.com/es-pe/library/ee677615.aspx

Requerimientos a Resolver

Requerimiento 1

Base de Datos: EDUTEC

Se necesita un procedimiento que permita generar un nuevo ciclo, las condiciones son las siguientes:

1. Por cada año existen doce ciclos, uno por mes, por ejemplo: 2013-01, 2013-02, 2013-03, etc.
2. El procedimiento debe leer el último ciclo y generar el que continua.
3. La fecha de inicio es el primer día del mes.
4. La fecha de fin es el último día del mes.
5. El nuevo ciclo se debe grabar en la tabla CICLO.

Requerimiento 2

Base de Datos: EDUTEC

Se necesita un procedimiento para registrar una nueva matricula, las condiciones son las siguientes:

1. Se debe verificar que el curso programado exista y se encuentre vigente.
2. Se debe verificar que el alumno exista.
3. Se debe verificar que el alumno no se encuentre matriculado.
4. Se debe verificar que existan vacantes disponibles.
5. La matrícula se registra en la tabla MATRICULA.
6. Se deben actualizar las columnas **vacantes** y **matriculados** en la tabla CURSOPROGRAMADO.