

Capítulo XII

Control de Errores

En este capítulo aprenderemos a utilizar las instrucciones SQL para controlar los errores que pueden presentarse durante la ejecución de las aplicaciones SQL.

Muchas veces cuando ejecutamos una aplicación, el flujo normal de la misma es interrumpido por la presencia de un error no previsto, lo que obliga al usuario a cancelar y cerrar la aplicación ante la imposibilidad de continuar. En otras ocasiones se presentan situaciones que para el lenguaje de programación no constituyen errores, pero si no controlamos estas situaciones pueden hacer pensar al usuario que se trata de un error de la aplicación; por ejemplo, un usuario ejecuta la búsqueda de un cliente en la base de datos y recibe un formulario vacío. Si no recibe ningún mensaje de la aplicación puede pensar que se trata de un error.

1. SITUACIONES CONSIDERADAS ERRORES

1.1. Error en tiempo de ejecución

Se presenta cuando durante la ejecución de un programa, el motor de ejecución no puede procesar una instrucción que es sintácticamente correcta. Este error interrumpe el flujo normal de ejecución del programa y suele generar un código y mensaje de error propios del lenguaje.

Ejercicio 12.1: Error en tiempo de ejecución

```
USE QhatuPERU
go

INSERT INTO LINEA(CodLinea, NomLinea)
VALUES(17, 'COMPUTO')
go
```

Esta instrucción INSERT es sintácticamente correcta. Además, la tabla LINEA existe en la base de datos **QhatuPERU**, y también las columnas **CodLinea** y **NomLinea**. El resto de columnas de la tabla permite valores NULL.

Cuando la enviamos al servidor se recibe el siguiente mensaje de error:

```
Mens. 544, Nivel 16, Estado 1, Línea 2
```

No se puede insertar un valor explícito en la columna de identidad de la tabla 'LINEA' cuando IDENTITY_INSERT es OFF.

La columna **CodLinea** tiene la propiedad IDENTITY, por lo que su valor es autogenerado por el sistema. Si deseamos insertar para dicha columna un valor explícito, como en este caso el valor 17, se requiere configurar a ON la opción IDENTITY_INSERT.

1.2. Error lógico

Se presenta cuando la ejecución del programa no genera ningún error, pero el resultado al que llegamos no es el esperado según la lógica de programación diseñada por el programador.

Ejercicio 12.2: Error lógico

Una de las muchas fórmulas utilizadas para obtener el valor de PI (π) es la siguiente:

$$\pi = 4 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 + 4/13 - \dots$$

Después de operar los primeros 20 millones de operandos el valor que se obtiene para PI es de aproximadamente 3.1416, dependiendo del lenguaje de programación y de la precisión utilizada para los datos numéricos. El siguiente script SQL utiliza la fórmula para calcular el valor de PI.

```
DECLARE @denom decimal(18,4), @contador integer
DECLARE @pi decimal(18,4), @control bit
SET @denom = 3.0
SET @contador = 0

SET @pi = 4.0
SET @control = 0
WHILE @contador < 20000000
BEGIN
    IF @control = 0
        BEGIN
            SET @pi = @pi + (4.0/@denom)
```

```

        SET @control = 1
        END
    ELSE
        BEGIN
            SET @pi = @pi + (4.0/@denom)
            SET @control = 0
            END
        SET @contador = @contador + 1
        SET @denom = @denom + 2.0
    END
    PRINT CONCAT('PI = ', @pi)

```

En este programa hay un error lógico, ya que si lo ejecutamos el resultado obtenido es $PI = 25.6601$. Le dejo como reto descubrir dónde se produce el error lógico.

1.3. "Error" de ausencia de datos

Se produce cuando el usuario ejecuta una consulta y no recibe dato alguno. Técnicamente no es un error porque una búsqueda puede o no encontrar los datos buscados. Sin embargo, algunos usuarios pueden pensar que ha ocurrido un error, y al no recibir ningún mensaje de la aplicación piensan que algo anda mal.

Ejercicio 12.3; "Error" de ausencia de datos

El siguiente procedimiento entrega el precio unitario del artículo P, donde P es el código del artículo.

```

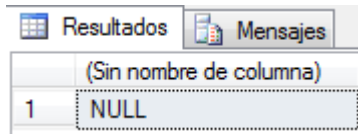
USE QhatuPERU
go

CREATE PROCEDURE usp_PrecioArticulo
    @articulo integer,
    @precio money OUTPUT
AS
SET NOCOUNT ON;
SET @precio = (SELECT PrecioProveedor
                FROM ARTICULO
                WHERE CodArticulo = @articulo);

```

go

```
DECLARE @elPrecio money
EXECUTE usp_PrecioArticulo 217, @elPrecio OUTPUT
SELECT @elPrecio
go
```



(Sin nombre de columna)	
1	NULL

El procedimiento entrega un valor NULL debido a que el precio no está registrado para dicho artículo, o el artículo no existe.

2. GENERACIÓN DE MENSAJES DE ERROR

La sentencia RAISERROR genera un mensaje de error que se le envía al usuario. Puede enviar un mensaje de error del sistema leído desde la vista del catálogo **sys.sysmessages** ó puede enviar un mensaje creado dinámicamente en tiempo de ejecución (mensaje ad hoc).

Sintaxis

```
RAISERROR( msgID | msgCad | @variableLocal ,
           severidad , estado )
           [ WITH LOG | WITH NOWAIT | WITH SETERROR ]
```

- El primer argumento indica que se puede especificar un número de mensaje (**msgID**), una cadena con el mensaje personalizado (**msgCad**), ó una variable conteniendo la cadena con el mensaje. **msgID** es el número del mensaje leído desde **sys.sysmessages**; **msgCad** es la cadena con el mensaje personalizado.
- **severidad** indica el nivel de gravedad del error. Su valor va de 0 a 25. Los niveles del 19 al 25 solo pueden ser especificados por los miembros del rol **sysadmin** y requieren el uso de WITH LOG. Los mensajes con niveles mayor a

19 son considerados muy graves y provocan que la conexión con la aplicación cliente se interrumpa.

- **estado** es un entero de 1 a 127 y su significado es determinado por el programador.
- WITH LOG permite registrar el error en el log de aplicación del sistema.
- WITH NOWAIT envia inmediatamente el mensaje al cliente.
- SETERROR configura el valor de la variable **@@error** al número del error generado independientemente del nivel de severidad. Por defecto, los mensajes de niveles de 0 a 10 no cambian el valor de **@@error**.

Ejercicio 12.4: Mensaje de error ad hoc

Un mensaje de error ad hoc es aquel que se puede utilizar solo en el programa en que fue creado.

El siguiente procedimiento entrega el monto total enviado a las tiendas en la fecha que se le entrega como parámetro.

```
USE QhatuPERU
go

CREATE PROCEDURE usp_MontoGuiasFecha
    @año char(4),
    @mes char(2),
    @dia char(2),
    @monto money OUTPUT
AS
SET NOCOUNT ON;
DECLARE @fecha char(10);
SET @fecha = @dia + '/' + @mes + '/' + @año;
IF NOT EXISTS (SELECT FechaSalida
                FROM GUIA_ENVIO
                WHERE CONVERT(char(10), FechaSalida, 103) =
                    @fecha)
BEGIN
    RAISERROR('No hay guías para la fecha
              indicada.', 16, 1);
    RETURN;
END;
SET @monto = (SELECT SUM(gd.CantidadEnviada *
                        gd.PrecioVenta)
              FROM GUIA_DETALLE gd INNER JOIN
```

```

        GUIA_ENVIO ge
        ON gd.NumGuia = ge.NumGuia
        WHERE CONVERT(char(10), ge.FechaSalida, 103)
            = @fecha);
go

```

Pruebe el procedimiento ejecutándolo para una fecha en la que no se ha emitido guías de envío.

```

DECLARE @elMonto money
EXEC usp_MontoGuiasFecha 2013, 05, 21,
    @elMonto OUTPUT
SELECT @elMonto
go

```

Se recibe el mensaje

```

Mens 50000, Nivel 16, Estado 1, Procedimiento
usp_MontoGuiasFecha, Línea 14
No hay guías para la fecha indicada.

```

2.1. Acerca de los mensajes de RAISERROR

Tenga en cuenta lo siguiente para los mensajes generados por RAISERROR:

- Todos los mensajes generados usando el argumento msgCad de la instrucción se consideran mensajes ad hoc y su número de mensaje asociado es siempre 50000.
- Los mensajes con nivel de severidad 10 se consideran solo informativos y por lo general indican que hay un error en la data ingresada por el usuario.
- Los mensajes con niveles de severidad del 11 al 16 señalan errores que pueden ser corregidos por el usuario.

Ejercicio 12.5: Generación de un mensaje de error del sistema

Usando RAISERROR se puede generar un mensaje de error del sistema, pero está limitado a mensajes cuyo número es mayor a 13000 y menor que 50000.

```

RAISERROR(13191, 10, 1)

```

```
SELECT @@ERROR
go
```

Se genera el mensaje

La entidad de seguridad de base de datos no tiene ninguna asignación a una entidad de servidor

Como se ha especificado el nivel de severidad 10, el error no se registra en la variable @@ERROR. Si desea que se registre se debe ejecutar RAISERROR con la opción SETERROR.

```
RAISERROR(13191, 10, 1) WITH SETERROR
SELECT @@ERROR
go
```

2.2. Las variables @@ROWCOUNT y @@ROWCOUNT_BIG

Ambas variables almacenan el número de filas afectadas por la última instrucción ejecutada. Si el número de filas afectadas es mayor a 2 mil millones utilice @@ROWCOUNT_BIG.

Ejercicio 12.6; Uso de @@ROWCOUNT

El siguiente procedimiento muestra una lista de los proveedores del artículo que se le entrega como parámetro.

```
USE QhatuPERU
go

CREATE PROCEDURE usp_ProveedoresArticulo
    @articulo integer
AS
SET NOCOUNT ON;
SELECT PROVEEDOR.CodProveedor,
        PROVEEDOR.NomProveedor
FROM PROVEEDOR INNER JOIN ARTICULO
ON PROVEEDOR.CodProveedor =
```



```

        ARTICULO.CodProveedor
WHERE ARTICULO.CodArticulo = @articulo;
IF @@ROWCOUNT = 0
    RAISERROR('No existe el artículo o no hay
        proveedores para el artículo', 16, 1);
go

```

Para probar el procedimiento

```

EXECUTE usp_ProveedoresArticulo 217
go

```

Se genera el mensaje

```

Mens 50000, Nivel 16, Estado 1, Procedimiento
usp_ProveedoresArticulo, Línea 11
No existe el artículo o no hay proveedores para
el artículo

```

Ejercicio 12.7: Uso de @@ERROR

```

USE QhatuPERU
go

SELECT DescripcionArticulo
FROM ARTICULO
WHERE CodArticulo = (SELECT CodArticulo
                        FROM GUIA_DETALLE
                        WHERE
CantidadEnviada > 100)
IF @@ERROR <> 0
    PRINT 'Se ha producido un error'

```

La consulta genera error porque la subconsulta entrega más de una fila.

3. BLOQUES TRY...CATCH

TRY...CATCH permite implementar un mecanismo de control de errores similar a los utilizados en otros lenguajes de programación.

Sintaxis

```
BEGIN TRY
    sentencia_SQL | bloque_sentencias_SQL
END TRY
BEGIN CATCH
    sentencia_SQL | bloque_sentencias_SQL
END CATCH
```

- El bloque TRY contiene las sentencias que el programa debe ejecutar para obtener los resultados deseados. El bloque CATCH es el que tiene las instrucciones para el control de errores.
- El bloque TRY detecta los errores que tienen un nivel de severidad mayor a 10 y que no cierran la conexión a la base de datos. Los errores cuyo nivel de severidad es 20 ó superior detienen el procesamiento de las tareas del motor de base de datos para la sesión y cierran la conexión.
- Si en el bloque TRY no se genera ningún error, el control de flujo se transfiere después de la sentencias END CATCH.

3.1. Obtención de información sobre un error

Las siguientes funciones se pueden utilizar en un bloque CATCH para obtener información del error capturado.

- ERROR_NUMBER(): entrega el número del error.
- ERROR_SEVERITY(): entrega el nivel de severidad del error.
- ERROR_STATE(): entrega el valor de estado del error.
- ERROR_PROCEDURE(): entrega el nombre del procedimiento almacenado o desencadenante donde se generó el error.
- ERROR_LINE(): entrega el número de línea que generó el error.
- ERROR_MESSAGE() entrega el texto del mensaje de error.

Ejercicio 12.8: Procedimiento que muestra información de error

El siguiente procedimiento utiliza las funciones para obtener información del error, y la muestra. El procedimiento puede ser invocado en el bloque CATCH de un procedimiento.

```
USE QhatuPERU
go

CREATE PROCEDURE usp_InfoError
AS
SELECT
    ERROR_NUMBER() AS NumeroError,
    ERROR_SEVERITY() AS SeveridadError,
    ERROR_STATE() AS EstadoError,
    ERROR_PROCEDURE() AS FuenteError,
    ERROR_LINE() AS LineaError,
    ERROR_MESSAGE() AS MensajeError;
GO
```

Ejercicio 12.9: Uso de TRY...CATCH

Este ejercicio utiliza el procedimiento usp_InfoError creado en el ejercicio anterior.

```
USE QhatuPERU
go

BEGIN TRANSACTION;

BEGIN TRY
    -- Sentencias SQL que forman parte
    -- de la transacción.
    -- Sentencia que genera un error
    -- de integridsd referencial.
    DELETE FROM GUIA_ENVIO
        WHERE NumGuia = 1;
END TRY

BEGIN CATCH
    EXECUTE usp_InfoError;
    IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;
END CATCH;
```

```
IF @@TRANCOUNT > 0
    COMMIT TRANSACTION;
go
```

Resultados		Mensajes				
	NumeroError	SeveridadError	EstadoError	FuenteError	LineaError	MensajeError
1	547	16	0	NULL	6	Instrucción DELETE en conflicto con la restricci...

4. LA INSTRUCCIÓN THROW

Esta instrucción genera una excepción o error y transfiere el control de flujo al bloque CATCH.

Sintaxis

```
THROW númeroError, mensaje, estado
```

El nivel de severidad de una excepción generada por THROW se establece siempre en 16.

Ejercicio 12.10: Generar una excepción usando THROW

```
THROW 50001, 'El registro no existe', 1
go
```

Se muestra el siguiente mensaje:

```
Mens. 50001, Nivel 16, Estado 1, Línea 1
El registro no existe
```

Ejercicio 12.11: Regenerar la última excepción ocurrida

La ejecución de una instrucción en el bloque TRY del siguiente ejercicio genera un error. En el bloque CATCH se utiliza THROW para propagar la excepción.

```

USE Pruebas
go
-- Esta base de datos la creamos
-- en el capítulo 11 del libro

CREATE TABLE PruebaThrow(
    Coll integer PRIMARY KEY)
go

BEGIN TRY
    SET NOCOUNT ON
    INSERT INTO PruebaThrow VALUES(1)
    -- Forzamos un error
    -- de violación de clave primaria
    INSERT INTO PruebaThrow VALUES(2)
END TRY
BEGIN CATCH
    PRINT 'THROW en el bloque CATCH';
    -- la instrucción anterior a THROW
    -- debe tener el ; de finalización
    -- de sentencia
    THROW
END CATCH

```

Se obtiene la siguiente salida

```

THROW en el bloque CATCH
Mens. 2627, Nivel 14, Estado 1, Línea 3
Infracción de la restricción PRIMARY KEY
'PK__PruebaTh__A259EE54A5045484'. No se puede
insertar una clave duplicada en el objeto
'dbo.PruebaThrow'. El valor de la clave duplicada
es (1).

```

5. MENSAJES DE ERROR CON PARÁMETROS SUSTITUIBLES

En este punto veremos cómo generar mensajes de error personalizados que le proporcionen al usuario detalles del porqué de la situación de error.

5.1. Mensaje RAISERROR con parámetros sustituibles

Al definir el texto del mensaje asociado a RAISERROR es posible utilizar los parámetros sustituibles %s (para definir una cadena), %i (para definir un número entero) para incluir en el texto del mensaje valores personalizados.

Ejercicio 12.12: Mensaje RAISERROR con parámetros sustituibles

Crear un procedimiento que genere un listado de las guías de envío emitidas en la fecha F que contienen al artículo P, donde P indica el tipo de artículo, por ejemplo QUESO, GALLETA, ó LECHE.

```
USE QhatuPERU
go

CREATE PROCEDURE usp_BuscaGuiaArticulo
    @fecha char(10), @articulo varchar(20)
AS
SET NOCOUNT ON
SELECT gd.NumGuia, gd.CodArticulo,
       a.DescripcionArticulo
FROM GUIA_DETALLE gd INNER JOIN ARTICULO a
    ON gd.CodArticulo = a.CodArticulo
INNER JOIN GUIA_ENVIO g
    ON gd.NumGuia = g.NumGuia
WHERE CONVERT(CHAR(10), g.FechaSalida, 103) =
    @fecha AND a.DescripcionArticulo LIKE
    '%' + @articulo + '%'

IF @@rowcount = 0
    RAISERROR('No existe guías para el
              artículo %s en la fecha %s.',
              16, 1, @articulo, @fecha)
go
```

Pruebe el procedimiento ejecutándolo para un fecha en la que no existen guías de envío o para guías en la que no existe el artículo especificado.

```
-- Probando el procedimiento
EXEC usp_BuscaGuiaArticulo '26/05/2013',
    'GALLETA'
go
```

Se recibe el siguiente mensaje de error:

```
Mens 50000, Nivel 16, Estado 1, Procedimiento
usp_BuscaGuiaArticulo, Línea 14
No existe guías para el artículo GALLETA en la
fecha 26/05/2013.
```

La definición de RAISERROR en el procedimiento

```
RAISERROR('No existe guías para el artículo %s en
la fecha %s.', 16, 1, @articulo, @fecha)
```

contiene 2 parámetros sustituibles: el primer **%s** que indica que en esa posición debe ir una cadena, y el segundo **%s** que también indica que en esa posición debe ir otra cadena.

El valor de la variable **@articulo** sustituye al primer **%s** en el mensaje, y el valor de la variable **@fecha** sustituye al segundo **%s**, generándose el siguiente mensaje:

```
No existe guías para el producto GALLETA en la
fecha 26/05/2013.
```

5.2. El procedimiento **sp_addmessage**

La instrucción **THROW** no permite parámetros sustituibles en el texto del mensaje. Para generar un mensaje con parámetros sustituibles con **THROW** es necesario primero almacenar el mensaje personalizado con los parámetros sustituibles en la tabla de sistema **sys.sysmessages** de SQL Server que guarda todos los mensajes del sistema. Luego utilizando la función **FORMATMESSAGE** se

le pasa al mensaje personalizado los valores de los parámetros que THROW se encargará de mostrar.

Veremos el uso del procedimiento **sp_addmessage** para guardar mensajes personalizados en el sistema.

Sintáxis

```
sp_addmessage msgID , severidad , msgCad  
    [ , idioma ] [ , TRUE | FALSE ]  
    [ , 'replace' ]
```

- **msgID** es el número del mensaje de error; para los mensajes de error personalizados este valor debe ser mayor a 50000, y el valor no debe haber sido asignado a otro mensaje.
- **severidad** indica el nivel de gravedad del error. Su valor va de 0 a 25. Los niveles del 19 al 25 solo pueden ser especificados por los miembros del rol **sysadmin** y requieren el uso de WITH LOG. Los mensajes con niveles mayor a 19 son considerados muy graves y provocan que la conexión con la aplicación cliente se interrumpa.
- **msCad** es el texto del mensaje, y que puede incluir parámetros sustituibles.
- **idioma** indica el idioma del mensaje. Cuando se crean cuentas de inicio de sesión de SQL se asigna a la cuenta un valor para idioma; este valor establece en qué idioma recibirá los mensajes el propietario de la cuenta. Para registrar un mensaje personalizado en el sistema, es obligatorio registrar primero el mensaje para el idioma inglés. Su valor predeterminado es NULL.
- **TRUE | FALSE** indica si cuando se produce el mensaje se debe registrar en el log de aplicación de Windows y en el log de errores del motor de base de datos. El valor predeterminado es FALSE.
- **'replace'**, si se especifica esta cadena indica que esta definición del mensaje debe reemplazar a la ya existente en el sistema. El valor predeterminado del parámetro es NULL.

Ejercicio 12.13: Registrar un mensaje de sistema personalizado

En este ejercicio registraremos en la tabla **sys.messages** el mensaje 60001, tanto para el idioma inglés como para español. En la definición del texto del mensaje en el idioma inglés usaremos los parámetros sustituibles; en la definición del

texto del mensaje en el segundo idioma (en este caso español) se usa la posición de los parámetros en el texto del idioma inglés (seguida del signo de exclamación !) debido a las diferencias sintácticas en la construcción de los mensajes en distintos idiomas, o porque se desea mostrarlos en otro orden.

```
sp_addmessage 60001, 16,  
'The status of article %i (%s) has changed to  
DISCONTINUED (valor 0)',  
'us_english'  
go
```

```
sp_addmessage 60001, 16,  
'El estado del artículo %2! (%1!) ha cambiado a  
DESCONTINUADO (valor 0)',  
'spanish'  
go
```

Probamos el mensaje personalizado.

```
RAISERROR(60001, 16, 1, 199, 'Arroz perlado')  
go
```

Se genera el siguiente mensaje:

```
Mens. 60001, Nivel 16, Estado 1, Línea 1  
El estado del artículo Arroz perlado (199) ha  
cambiado a DESCONTINUADO (valor 0)
```

5.3. Mensaje THROW con parámetros sustituibles

Como se ha mencionado arriba, la instrucción THROW requiere que nos ayudemos con la función FORMATMESSAGE para generar un mensaje con parámetros sustituibles con ella.

Ejercicio 12.14: Mensaje THROW con parámetros sustituibles

```
USE QhatuPERU  
go
```

```

DECLARE @msg nvarchar(2048)
DECLARE @articulo varchar(40)
BEGIN TRY
    SET NOCOUNT ON
    UPDATE ARTICULO
        SET StockActual = 0, Descontinuado = 0
        WHERE CodArticulo = 9
    SET @articulo = (SELECT DescripcionArticulo
                     FROM ARTICULO
                     WHERE CodArticulo = 9)

    SET @msg =
        FORMATMESSAGE(60001, 9, @articulo);
    THROW 60001, @msg, 1
END TRY
BEGIN CATCH
    THROW
END CATCH

```

Se genera el siguiente mensaje:

```

Mens. 60001, Nivel 16, Estado 1, Línea 13
El estado del artículo MELLOWS FAMILIAR FIELD (9)
ha cambiado a DESCONTINUADO (valor 0)

```

La función `FORMATMESSAGE` recibe como primer argumento el número del mensaje al que debe pasarle los valores de los parámetros sustituibles, y en los argumentos siguientes debe pasar los valores en el orden en que fueron definidos en el mensaje en inglés.

6. EJERCICIOS PROPUESTOS

Para los ejercicios propuestos del capítulo anterior, se le pide añadirles control de errores.

1. Escriba un procedimiento que entregue una lista de los artículos de la línea L, donde L es el nombre de la línea de artículos.

2. Escriba un procedimiento que entregue una lista de los artículos del proveedor P y de la línea L, donde P y L son el nombre del proveedor y el nombre de la línea respectivamente.
3. Crear un procedimiento que entregue el monto total de los artículos enviados a la tienda T, donde T es el código de la tienda.
4. Crear un procedimiento que entregue el monto total de los artículos enviados a la tienda T durante el mes M del año A.
5. Crear un procedimiento que entregue la lista de los N artículos más enviados a las tiendas; N es un entero.
6. Crear un procedimiento que entregue el balance entrada/salida del artículo A. El procedimiento debe entregar el total de unidades que ingresaron, y el total de unidades que salieron del artículo A.
7. Crear un procedimiento que entregue el monto total adquirido al proveedor P durante el año A, donde P es el código del proveedor.
8. Crear un procedimiento que en la tabla ARTICULO permita actualizar el precio del artículo A, donde A es el código del artículo.
9. Crear un procedimiento que permita registrar los datos de un artículo nuevo.
10. Crear un procedimiento que calcule el monto promedio mensual enviado a la tienda T, donde T es el código de la tienda.