



ERROR Handling

Database Programming



@@ERROR



Using @@ERROR

- @@ERROR returns the error number for the last SQL statement executed
- @@ERROR Returns
 - 0 if the previous statement encountered no errors or
 - An error number if an error is encountered
- Any T-SQL statement causes @@ERROR to be reset
- @@ERROR value must read immediately after the statement to be tested



Using @@ERROR (cont.)

- If the error is one of the errors in the sys.messages, then @@ERROR will be the value from the sys.messages.message_id column for that error.
- You can view the text (description) associated with the error by:

```
SELECT * FROM sys.messages where message_id = 222
```
- To check the success or failure of any statement
 - Store the value of @@ERROR to a local variable
 - Test the value of the variable and take appropriate action



@@ERROR - Example

```
CREATE PROC spInsertEmployee
    @FName varchar(50),@LName varchar(50),@DepID int
AS
BEGIN
    DECLARE @ErrorNo int, @retVal int

    INSERT Employee (FName, LName, DepID)
    VALUES (@FName , @LName, @DepID)

    SET @ErrorNo = @@ERROR

    IF @ErrorNo = 0
        SET @retVal = 0
    ELSE
        SET @retVal = -1
END
```



@@ERROR & @@ROWCOUNT

- @@ROWCOUNT - Returns the number of rows affected or read by the last statement
 - Statements that make a simple assignment always set the @@ROWCOUNT value to 1
- Both @@ERROR and @@ROWCOUNT are reset with each Transact-SQL statement;
 - Therefore, both must be referenced in the same statement immediately after the one being tested
- To refer to both @@ERROR and @@ROWCOUNT after a statement, they must be referenced in the same statement
 - assigning @@ERROR to a variable will also reset @@ROWCOUNT
 - assigning @@ROWCOUNT to a variable will also reset @@ERROR



@@ERROR & @@ROWCOUNT

```
DECLARE @ErrorVar INT;  
DECLARE @RowCountVar INT;
```

```
DELETE      Employee  
WHERE       DepID = 3
```

```
-- Save @@ERROR and @@ROWCOUNT
```

```
SELECT @ErrorVar = @@ERROR, @RowCountVar = @@ROWCOUNT
```

```
PRINT 'Value of @@ROWCOUNT = ' + CAST(@@ROWCOUNT AS VARCHAR(8));
```

```
IF (@ErrorVar <> 0)  
    PRINT 'Error = ' + CAST(@ErrorVar AS VARCHAR(8));
```

```
PRINT 'Value of @@ROWCOUNT = ' + CAST(@@ROWCOUNT AS VARCHAR(8));  
PRINT 'Rows Deleted = ' + CAST(@RowCountVar AS NVARCHAR(8));  
GO
```



RAISE ERROR



RAISEERROR

- RAISEERROR allows you to raising your own error (custom defined error)
- RAISEERROR allows you to Create an dynamic message to return
 - Raise errors that you have defined and registered in `sys.messages`
- RAISERROR can either reference a user-defined message stored in the `sys.messages` catalog view or build a message dynamically
- Error numbers for user-defined error messages should be greater than 50000



RAISEERROR - Syntax

RAISERROR ({ msg_id | msg_str | @local_variable } { ,severity ,state }

■ msg_id

- ☐ A user-defined error message number stored in the sys.messages catalog view using sp_addmessage
- ☐ Error numbers for user-defined error messages should be greater than 50000
- ☐ When msg_id is not specified, RAISERROR raises an error message with an error number of 50000

■ msg_str

- ☐ Is a user-defined message



RAISEERROR – Syntax (cont.)

■ Severity

- Is the user-defined severity level associated with this message
- Severity Levels (1 – 25)
 - 1 – 10 : Info only
 - 11-16 : Constraint violation because of bad data
 - 17 - 19 : Hardware / software error
 - 20 - 25 : Fatal Error (client connection is disconnected)

■ State

- Is an integer from 0 through 255



RAISEERROR - Examples

■ Example

□ RAISEERROR('Invalid operation, 16, 1)

```
sp_addmessage @msgnum = 50005,  
              @severity = 10,  
              @msgtext = 'Invalid Operation'
```

GO

```
RAISEERROR (50005, 10, 1 )
```

GO

```
sp_dropmessage @msgnum = 50005;
```

GO



RAISEERROR - Examples

- Using a local variable to supply the message text for a RAISEERROR statement.

```
DECLARE @StringVariable NVARCHAR(50);  
SET @StringVariable = N'Invalid Operation';
```

```
RAISERROR (@StringVariable, 10, 1 )  
GO
```



Try Catch



Try ... Catch

- Syntax:

```
BEGIN TRY
    { <sql statement(s)> }
END TRY
BEGIN CATCH
    { <sql statement(s)> }
END CATCH
```



Try ... Catch

- SQL Server will “try” to run anything within the BEGIN...END that goes with your TRY block.
- SQL Server will exit the TRY block immediately and begin with the first line in your CATCH block only when an error condition that has an error level of 11–19 occurs
- Errors trapped by a catch block are not returned to the calling application
 - If error info is to be returned to the application use RAISERROR() function



Try ... Catch - Example

```
DECLARE @ErrorNo int, @Severity tinyint, @State smallint, @LineNo int, @Message nvarchar(4000);
BEGIN TRY
    -- Try and create the table
    CREATE TABLE Employee(Col1 int PRIMARY KEY);
END TRY
BEGIN CATCH
    SELECT @ErrorNo = ERROR_NUMBER(), @Severity = ERROR_SEVERITY(),
           @State = ERROR_STATE(), @LineNo = ERROR_LINE (),
           @Message = ERROR_MESSAGE();

    SELECT @ErrorNo , @Severity, @State, @LineNo, @Message

    IF @ErrorNo = 2714 -- Object exists error
        SELECT 'WARNING: Skipping CREATE as table already exists';
    ELSE -- error not recognized
        RAISERROR(@Message, 16, 1 );
END CATCH
```

Exercise

■ Implement an error handling mechanism for the following

□ Tables

Product(ProductID, Pname, UnitPrice, AmountAvailable)

Sales(SalesID, ProductID, QuantitySold)

Product			
ProdID	ProdName	UnitPrice	qntyAvailable
1	LapTop	18000	10
2	DeskTop	15000	30

ProdSales		
SalesID	ProdID	qntySold
1	1	5

□ Business Rules

1. AmountAvailable $\geq$ QuantitySold
2. ProductID is the PK for Product table
3. SalesID is incremental number



Example (cont.)

```
CREATE PROC uspSelProduct
@prodID int, @qntToSell int
AS
BEGIN
    --check if there is enough quantity to sell

    IF (@qntAvailable < @qntToSell)
        RAISERROR('Not enough quantity available', 16, 1)
    END
    ELSE
        BEGIN
            BEGIN TRAN
                -- deduct quantity to sell
                -- Increment SalesID by 1 to get the new value for the PK
            COMMIT TRAN
        END
    END
END
```



Example 2

--if there is enough quantity to sell

BEGIN TRY

BEGIN TRAN

-- deduct quantity to sell

-- Increment SalesID by 1 to get the new value for the PK

COMMIT TRAN

END TRY

BEGIN CATCH

ROLLBACK TRAN

SELECT

ERROR_NUMBER() AS ErrorNumber

,ERROR_SEVERITY() AS ErrorSeverity

,ERROR_STATE() AS ErrorState

,ERROR_PROCEDURE() AS ErrorProcedure

,ERROR_LINE() AS ErrorLine

,ERROR_MESSAGE() AS ErrorMessage;

END CATCH

END