**EXCEPTION HANDLING – SQL SERVER**

**What is Exception Handling in SQL Server?**

* Exception Handling means dealing with errors that happen while running SQL code.
* It helps to find and catch errors so the program doesn’t just stop suddenly.
* It lets you handle errors in a planned way, like showing a message or fixing the problem.
* This makes your database programs more reliable and easier to maintain.
* In SQL Server, we use TRY and CATCH blocks to catch and respond to errors.
* When an error occurs inside the TRY block, control moves to the CATCH block where you can manage the error.
* You can log errors, roll back transactions, or display custom messages inside the CATCH block.

**Why do we use Exception Handling?**

* To maintain data integrity by preventing unwanted changes during errors.
* To provide meaningful error messages to users or logs.
* To ensure smooth execution flow by catching and managing exceptions.
* To perform rollback or cleanup operations if errors occur.

**When do we Handling use Exception?**

* When executing complex transactions requiring atomicity.
* When dealing with user inputs that can cause errors.
* When performing operations that may fail (e.g., division by zero, constraint violations).
* When calling external resources or dependent stored procedures.

**When don’t we need Exception Handling?**

* Simple queries or scripts where error impact is minimal.
* Read-only queries with no risk to data consistency.
* Quick ad-hoc commands without critical error management.
* When the calling application or client handles errors externally.

**SQL Server Error Functions for Exception Handling**

* ERROR\_NUMBER()
* ERROR\_MESSAGE()
* ERROR\_SERVERITY()
* ERROR\_STATE()
* ERROR\_LINE()
* ERROR\_PROCEDURE()

**1. ERROR\_MESSAGE()**

* **Explanation:**  
  This function returns the full descriptive text of the error that caused the control to jump to the CATCH block. It provides a clear, human-readable explanation of what went wrong during execution, such as syntax errors, constraint violations, or runtime errors.
* **Use:**  
  ERROR\_MESSAGE() is primarily used for displaying meaningful error messages to users or logging detailed information for developers and DBAs. It helps in understanding the nature of the problem quickly and facilitates troubleshooting.
* **Example Message:**  
  For a divide-by-zero error, the message might be:  
  **"Divide by zero error encountered."**  
  For a foreign key violation, it might be:  
  **"The INSERT statement conflicted with the FOREIGN KEY constraint."**

**2. ERROR\_NUMBER()**

* **Explanation:**  
  This function returns a numeric error code unique to the type of error encountered. Each SQL Server error is assigned a number, making it easier to identify and programmatically handle specific error cases.
* **Use:**  
  ERROR\_NUMBER() is useful when you want to take different actions based on the error type. For example, you might want to retry a transaction on deadlock errors but log and abort on data conversion errors.
* **Example Error Numbers and Their Meaning:**
  + **8134**: Divide by zero error
  + **208**: Invalid object name (e.g., referencing a non-existent table)
  + **547**: Foreign key violation (e.g., trying to delete a parent row referenced by child rows)

**3. ERROR\_SEVERITY()**

* **Explanation:**  
  This function returns the severity level of the error on a scale of 0 to 25. The severity indicates how serious the error is, ranging from informational messages to fatal errors.
* **Use:**  
  By examining the severity, you can decide how to respond to errors. For example, low severity errors might just trigger a warning, whereas high severity errors might require rolling back transactions or closing connections.
* **Severity Level Ranges:**
  + **0-10**: Informational messages or warnings (not errors)
  + **11-16**: Errors caused by user mistakes (e.g., syntax errors, constraint violations)
  + **17-19**: Software or hardware-related errors, such as resource problems (memory, disk space)
  + **20-25**: Fatal errors that usually terminate the database connection or require immediate attention

**4. ERROR\_STATE()**

* **Explanation:**  
  The state number is an additional code that indicates the location within the SQL Server engine or user code where the error was raised. It often helps differentiate the context of an error when multiple sources can cause the same error number.
* **Use:**  
  Useful for debugging complex systems where the same error code might appear in different procedures or triggers. It helps pinpoint the exact internal state or condition causing the error.
* **Example:**  
  For a foreign key violation (error 547), the error state might differ if the violation happens in a trigger versus a stored procedure, helping narrow down the source of the problem.

**5. ERROR\_LINE()**

* **Explanation:**  
  Returns the line number inside the current batch, stored procedure, or trigger where the error occurred.
* **Use:**  
  Extremely helpful for debugging because it tells you exactly where the error was triggered in your SQL code, especially in large scripts or procedures.
* **Example:**  
  If an error occurs on line 15 of a stored procedure, ERROR\_LINE() returns 15. This lets you open the procedure and immediately jump to the problematic line.

**6. ERROR\_PROCEDURE()**

* **Explanation:**  
  Returns the name of the stored procedure or trigger where the error occurred. If the error occurs outside of any procedure or trigger (e.g., in a direct batch), it returns NULL.
* **Use:**  
  This is valuable in complex database environments with many stored procedures and triggers, helping developers trace exactly which procedure generated the error.
* **Example:**  
  Suppose an error happens inside a procedure named UpdateOrderStatus. Calling ERROR\_PROCEDURE() will return "UpdateOrderStatus". This helps isolate the source quickly in a multi-procedure system.

#### @@ERROR:

#### Explanation: @@ERROR is a system function that returns the error number of the last T-SQL statement executed. If the last statement was successful, it returns 0.

* **Use:**  
  It is mainly used in **older T-SQL error handling** before TRY...CATCH was introduced. You typically check @@ERROR immediately after each statement to detect if an error occurred.
* **Important Notes:**
  + Must be checked **immediately** after the statement; otherwise, its value gets overwritten.
  + Doesn't provide details like message, line, severity, etc.

**RAISERROR:**

* **Explanation:**  
  RAISERROR is used to **generate a custom error message** and send it to the calling application, trigger a CATCH block, or log the issue.
* **Use:**
  + Raise custom or system-defined errors
  + Trigger CATCH logic
  + Log or return meaningful messages with custom severity levels
* **Syntax:**

RAISERROR ('Message text', severity, state);

* **Severity & State:**
  + Severity: 11–16 for user-defined errors
  + State: A number from 0 to 255 (just used for tracking different locations)

#### THROW:

* **Explanation:**  
  THROW is a newer and **more modern** way to raise errors in SQL Server (introduced in SQL Server 2012). It re-throws the original error or creates a new one, and **preserves the error’s original context**, unlike RAISERROR.
* **Use:**
  + Re-throw caught errors in a CATCH block
  + Create and throw custom errors
  + Cleaner and simpler syntax than RAISERROR
* **Syntax to re-throw:**

THROW;

* **Syntax to raise new error:**

THROW error\_number, message, state;