Let's go through each topic with two code examples for each.

Error Handling:

1. TRY...CATCH Blocks:

```sql
-- Example 1: Simple TRY...CATCH block
BEGIN TRY
    -- Some code that might cause an error
    SELECT 1 / 0; -- Division by zero error
END TRY
BEGIN CATCH
    -- Handle the error
    PRINT 'An error occurred: ' + ERROR_MESSAGE();
END CATCH;
```

In this example, the `TRY` block contains code that may cause an error (division by zero). If an error occurs, it's caught by the `CATCH` block, and a message is printed using the `ERROR_MESSAGE()` function.

```sql
-- Example 2: Nested TRY...CATCH blocks
BEGIN TRY
    BEGIN TRY
        -- Some code that might cause an error
        SELECT 1 / 0; -- Division by zero error
    END TRY
    BEGIN CATCH
        -- Handle the error in the inner CATCH block
        PRINT 'Inner Catch Block: ' + ERROR_MESSAGE();
    END CATCH;
END TRY
BEGIN CATCH
    -- Handle the error in the outer CATCH block
    PRINT 'Outer Catch Block: ' + ERROR_MESSAGE();
END CATCH;
```

This example demonstrates nested `TRY...CATCH` blocks. If an error occurs in the inner block, it is caught by the inner `CATCH` block. If an error occurs in the outer block, it is caught by the outer `CATCH` block.

2. RAISEERROR Statement:

```sql
-- Example 1: Using RAISEERROR for a custom error
BEGIN
    DECLARE @ErrorMessage NVARCHAR(200) = 'This is a custom error.';
    DECLARE @ErrorSeverity INT = 16;
    RAISEERROR(@ErrorMessage, @ErrorSeverity, 1);
END;
```

In this example, the `RAISEERROR` statement is used to raise a custom error with a specified message and severity level.

```sql
-- Example 2: Using RAISEERROR with error information
BEGIN TRY
    -- Some code that might cause an error
    SELECT 1 / 0; -- Division by zero error
END TRY
BEGIN CATCH
    -- Raise a custom error with information from the caught error
    DECLARE @ErrorMessage NVARCHAR(200) = ERROR_MESSAGE();
    DECLARE @ErrorSeverity INT = 16;
    RAISEERROR(@ErrorMessage, @ErrorSeverity, 1);
END CATCH;
```

In this example, the `RAISEERROR` statement is used within a `CATCH` block to raise a custom error with information from the caught error.

Working with JSON Data:

1. JSON Functions:

```sql
-- Example 1: Using JSON_VALUE to extract a value from JSON
DECLARE @json NVARCHAR(MAX) = '{"name": "John", "age": 30}';
SELECT JSON_VALUE(@json, '$.name') AS Name;
```

In this example, the `JSON_VALUE` function is used to extract the value of the "name" property from a JSON string.

```sql
-- Example 2: Using JSON_QUERY to extract a JSON object or array
DECLARE @json NVARCHAR(MAX) = '{"employees": [{"name": "Alice"}, {"name": "Bob"}]}';
SELECT JSON_QUERY(@json, '$.employees') AS Employees;
```

Here, the `JSON_QUERY` function is used to extract the JSON array of employees from a JSON string.

2. JSON Data Types:

```sql
-- Example 1: Creating a table with a JSON column
CREATE TABLE EmployeeData
(
    EmployeeID INT PRIMARY KEY,
    EmployeeDetails NVARCHAR(MAX)
);

-- Inserting JSON data into the table
INSERT INTO EmployeeData (EmployeeID, EmployeeDetails)
VALUES (1, '{"name": "Alice", "age": 25, "department": "HR"}');
```

In this example, a table `EmployeeData` is created with a column `EmployeeDetails` of type `NVARCHAR(MAX)` to store JSON data. JSON data is then inserted into the table.

```sql
-- Example 2: Querying JSON data from the table
SELECT EmployeeID, JSON_VALUE(EmployeeDetails, '$.name') AS Name,
JSON_VALUE(EmployeeDetails, '$.age') AS Age
FROM EmployeeData;
```

Here, a `SELECT` statement is used to query specific values from the JSON data stored in the table, demonstrating how to work with JSON data types.

Certainly! Let's continue with more examples for each topic.

Error Handling:

3. TRY...CATCH Blocks:

```sql
```

```
-- Example 3: Handling different types of errors in a TRY...CATCH block
BEGIN TRY
    -- Some code that might cause an error
    SELECT 1 / 0; -- Division by zero error
END TRY
BEGIN CATCH
    -- Check for specific error numbers
    IF ERROR_NUMBER() = 8134
        PRINT 'Divide by zero error occurred.';
    ELSE
        PRINT 'An unexpected error occurred: ' + ERROR_MESSAGE();
END CATCH;
```

In this example, the `ERROR_NUMBER()` function is used within the `CATCH` block to check for a specific error number (8134 in this case) and handle it accordingly.

```sql
-- Example 4: Rethrowing an error in a TRY...CATCH block
BEGIN TRY
    -- Some code that might cause an error
    EXEC non_existent_procedure; -- Calling a non-existent stored procedure
END TRY
BEGIN CATCH
    -- Rethrow the error with additional information
    DECLARE @ErrorMessage NVARCHAR(200) = 'Error calling stored procedure: ' +
ERROR_MESSAGE();
    RAISEERROR(@ErrorMessage, 16, 1);
END CATCH;
```

In this example, if an error occurs, it is caught by the `CATCH` block, and a new error is raised with additional information using the `RAISEERROR` statement.

Working with JSON Data:

3. JSON Functions:

```sql
-- Example 3: Using OPENJSON to parse a JSON array
DECLARE @json NVARCHAR(MAX) = '[{"name": "Alice", "age": 25}, {"name": "Bob", "age":
30}]';
SELECT *
FROM OPENJSON(@json)
```

```
WITH (
    Name NVARCHAR(50) '$.name',
    Age INT '$.age'
) AS Persons;
```

Here, the `OPENJSON` function is used to parse a JSON array, extracting the "name" and "age" properties for each person.

```sql
-- Example 4: Using JSON_MODIFY to update a value in a JSON string
DECLARE @json NVARCHAR(MAX) = '{"name": "Alice", "age": 25}';
SET @json = JSON_MODIFY(@json, '$.age', 26);
SELECT @json AS UpdatedJson;
```

In this example, the `JSON_MODIFY` function is used to update the value of the "age" property in a JSON string, and the updated JSON is displayed.

4. JSON Data Types:

```sql
-- Example 3: Updating JSON data in the table
UPDATE EmployeeData
SET EmployeeDetails = JSON_MODIFY(EmployeeDetails, '$.age', 26)
WHERE EmployeeID = 1;
```

This example demonstrates updating a specific property ("age") within a JSON column in the `EmployeeData` table.

```sql
-- Example 4: Filtering data based on JSON properties
SELECT EmployeeID, EmployeeDetails
FROM EmployeeData
WHERE JSON_VALUE(EmployeeDetails, '$.department') = 'HR';
```

Here, a `SELECT` statement is used to filter rows based on the value of the "department" property in the JSON data stored in the `EmployeeDetails` column.