Module 7

What to learn

Union Clause
Except and Intersect Clauses
Derived Tables
CTE

Practice Exercise

Practice 1

Do the practice exercise from the following link https://docs.microsoft.com/en-us/sql/t-sql/queries/with-common-table-expression-transact-sql?view=sql-server-ver15

Practice 2

Practice Exercise: Declare Variables and Set Values

Create a variable to store a customer's first name and another for their last name. Assign the values 'John' and 'Doe' to these variables, respectively. Use these variables to display the full name of the customer in a query output.

Practice 3

Practice Exercise: Use of IF-ELSE Logic

Create a table named **CustomerOrders** with columns **OrderID** (Primary Key, INT), **CustomerName** (VARCHAR), and **OrderAmount** (DECIMAL). Write an SQL query using **IF-ELSE** logic to check if the **OrderAmount** is greater than 500. Display 'Premium Order' if true, else display 'Standard Order'.

Practice 4

Practice Exercise: Case Statement

Add a column **OrderCategory** to the table **CustomerOrders**, which categorizes orders into 'Low', 'Medium', or 'High' based on the **OrderAmount**:

'Low' for amounts less than 100
'Medium' for amounts between 100 and 500
'High' for amounts above 500

Write a query using a **CASE** statement to populate this column appropriately.

Practice 5

Practice Exercise: Use WHILE Loop

Create an SQL query that uses a **WHILE** loop to generate a series of numbers from 1 to 10. Output each number in a column labeled 'SequenceNumber'.

Practice 6

Practice Exercise: Validate JSON Using ISJSON

Create a column **OrderDetails** (NVARCHAR(MAX)) in the table **CustomerOrders**. Insert a few rows with valid JSON strings, and a few with invalid JSON strings. Use the **ISJSON** function to filter out only the rows containing valid JSON data.

Practice 7

Practice Exercise: Extract Values from JSON using JSON_VALUE

Suppose the **OrderDetails** column contains valid JSON strings with keys such as `{ 'Product': 'Laptop', 'Price': 1200 }`. Write a query to extract and display the 'Product' and 'Price' values separately using the **JSON_VALUE** function.

Practice 8

Practice Exercise: Retrieve Nested JSON Objects using JSON_QUERY

Insert a nested JSON string in the **OrderDetails** column, e.g., `{ 'OrderID': 101, 'Items': [ { 'Product': 'Laptop', 'Price': 1200 }, { 'Product': 'Mouse', 'Price': 50 } ] }`. Write an SQL query to retrieve the array of 'Items' as JSON using the **JSON_QUERY** function.

Practice 9

Practice Exercise: Update JSON Data using JSON_MODIFY

Using the nested JSON data from the previous exercise, write an SQL query to change the price of the 'Laptop' from 1200 to 1100 using the **JSON_MODIFY** function.

Practice 10

Practice Exercise: Convert JSON Collection to Rowset using OPENJSON

Consider the nested JSON data in the **OrderDetails** column. Write a query to use the **OPENJSON** function to transform each item in the 'Items' array into a separate row with columns for 'Product' and 'Price'.

Practice 11

Practice Exercise: Export SQL Server Data to JSON using FOR JSON PATH

Write a query on the **CustomerOrders** table to export all records as a JSON array using the **FOR JSON PATH** clause, including nested details if applicable.

Practice 12

Practice Exercise: Export SQL Server Data to JSON using FOR JSON AUTO

Modify the previous exercise to use the **FOR JSON AUTO** clause instead. Analyze the differences in the structure of the returned JSON data.

Practice 13

Practice Exercise: Export JSON without Array Wrapper

Write a query to export the first record from the **CustomerOrders** table as a JSON object, without wrapping it in an array. Use the **WITHOUT_ARRAY_WRAPPER** option in your query.

Assignment Exercise

Assignment 1

Convert Day4 and Day5 Exercises with CTE and Derived Table.

Assignment 2

Assignment: Order Management System with JSON Handling

Business Scenario: Create an order management system for an e-commerce platform. The system should store customer orders and allow retrieving and updating JSON-formatted order details through SQL operations.

Functional Flow:

Create the following tables:

Customers (CustomerID, CustomerName, ContactInfo)

Orders (OrderID, CustomerID, OrderDate, OrderDetails as a JSON column)

Add at least 5 customers and 10 orders, where **OrderDetails** is formatted as JSON strings. Examples of **OrderDetails**: `{ 'Items': [ { 'Product': 'Laptop', 'Quantity': 1, 'Price': 1200 }, { 'Product': 'Mouse', 'Quantity': 2, 'Price': 25 } ], 'OrderTotal': 1250 }`.

Write a query to validate the JSON structure in the **OrderDetails** column using **ISJSON**.

Write a query to extract specific fields such as 'OrderTotal', 'Product', and 'Quantity' from the JSON using **JSON_VALUE**.

Transform the 'Items' array into individual rows using the **OPENJSON** function, displaying columns for 'Product', 'Quantity', and 'Price'.

Update an order's price for a specific product in the JSON data using **JSON_MODIFY**.
Export all orders as a JSON array using **FOR JSON PATH**, and compare it with the output of **FOR JSON AUTO**.
Export the details of a single order (e.g., OrderID = 1) as a standalone JSON object using **WITHOUT_ARRAY_WRAPPER**.

Business Logic:

Customers can have multiple orders, and each order can consist of multiple items.
Ensure the JSON fields adhere to the correct structure and type requirements.
Provide queries for analyzing sales, such as finding customers who have spent the most on purchases or listing the top products based on sales volume.

Online Reference

No online Reference

- Introduction to Relational Databases
- Introduction to Select Statement
- Filtering Results with WHERE Statements
- Utilizing Joins
- Executing Sub queries and Unions
- Aggregating Data
- Advanced Data Aggregations
- Built in Functions
- Query Optimization
- Modifying Data
- Advanced Data Modification
- Stored Procedure
- Transaction
- Error handling
- Designing Tables
- triggers