T-SQL Programming with AdventureWorks2022: Class Note

Course Information

- **Course**: Introduction to Transact-SQL Programming
- **Session**: T-SQL Language Elements (Including Batching), Sets, Predicate Logic, and Logical Order of Operators
- **Database**: AdventureWorks2022
- **Tool**: SQL Server Management Studio (SSMS) or compatible SQL client
- **Date**: May 29, 2025

Learning Objectives

By the end of this session, students will be able to:

1. Declare and use T-SQL variables, operators, predicates, expressions, comments, and batches to construct dynamic and organized queries.
2. Apply batching to manage execution scope and handle multiple operations in a single script effectively.
3. Use set operations (`UNION` , `INTERSECT` , `EXCEPT`) and predicate logic to manipulate and filter data.
4. Understand and apply the logical order of operators in a T-SQL `SELECT` statement to predict query behavior and optimize performance.
5. Write and debug T-SQL scripts using the AdventureWorks2022 database, incorporating best practices like commenting and batch separation.

Delivery Expectations

- **Participation**: Actively engage in class discussions and ask questions to clarify concepts.
- **Hands-On Practice**: Complete all classwork exercises during or after the session to build practical skills.
- **Submission**: Save all T-SQL scripts with specified filenames (e.g., `LanguageElements.sql`) and submit as instructed.

- **Testing**: Run queries in SSMS to verify results, using `SELECT TOP 10` for large datasets to avoid performance issues.
- **Documentation**: Include single-line (`--`) and multi-line (`/* */`) comments in all scripts to explain purpose and logic.
- **Batching**: Use `GO` to separate batches where appropriate to demonstrate understanding of execution scope.
- **Environment**: Ensure access to the AdventureWorks2022 database in SSMS or a compatible tool before starting.

Session Outline

Sub-Session 1: Transact-SQL Language Elements (Including Batching)

Definition

Transact-SQL (T-SQL) language elements are the fundamental components used to build queries and scripts in SQL Server. These include:

- **Variables**: Store temporary data (e.g., `DECLARE @Count INT;`).
- **Operators**: Perform operations like arithmetic (`+` , `-`), comparison (`=` , `>`), and logical (`AND` , `OR`).
- **Predicates**: Conditions that evaluate to `TRUE` or `FALSE` (e.g., `WHERE ListPrice > 100`).
- **Expressions**: Combinations of values, operators, and functions (e.g., `ListPrice * 1.1`).
- **Comments**: Documentation within code using `--` for single-line or `/* */` for multi-line.
- **Batches**: Groups of T-SQL statements executed as a single unit, separated by the `GO` keyword.

Explanation

- **Variables**: Declared with `DECLARE` and assigned with `SET` or `SELECT` . They are scoped to the batch in which they are declared.
- **Operators**: Enable calculations (e.g., `ListPrice + 50`) and comparisons (e.g., `Color = 'Red'`).
- **Predicates**: Filter data in `WHERE` or `HAVING` clauses based on conditions.
- **Expressions**: Produce computed values, used in `SELECT` , `WHERE` , or other clauses.
- **Comments**: Improve code readability and maintainability, essential for collaborative development.
- **Batches**:
  - A batch is a set of T-SQL statements sent to SQL Server for execution as a single unit.
  - The `GO` keyword signals the end of a batch, resetting variable scope and certain session settings.

- Batches are useful for organizing scripts, controlling variable scope, and executing multiple operations sequentially.
- Variables declared in one batch are not accessible in subsequent batches unless redefined.

Examples

1. **Using Variables, Predicates, and Batching**

```
USE AdventureWorks2022;
-- Batch 1: Filter products by price
DECLARE @MinPrice DECIMAL(10,2) = 500.00;
/* Query products with price above the threshold
   and calculate a 10% price increase */
SELECT ProductID, Name, ListPrice, (ListPrice * 1.10) AS IncreasedPrice
FROM Production.Product
WHERE ListPrice > @MinPrice;
GO
-- Batch 2: Update price and repeat query
SET @MinPrice = 1000.00; -- Error: @MinPrice is out of scope
SELECT ProductID, Name, ListPrice, (ListPrice * 1.10) AS IncreasedPrice
FROM Production.Product
WHERE ListPrice > @MinPrice;
```

Explanation: The second batch fails because `@MinPrice` is not defined (out of scope). Variables are batch-specific.

2. **Correct Batching with Variables**

```
USE AdventureWorks2022;
-- Batch 1: Filter products by stock level
DECLARE @StockLevel INT = 200;
SELECT ProductID, Name, SafetyStockLevel
FROM Production.Product
WHERE SafetyStockLevel >= @StockLevel;
GO
-- Batch 2: Redefine variable and repeat query
DECLARE @StockLevel INT = 400;
SELECT ProductID, Name, SafetyStockLevel
FROM Production.Product
WHERE SafetyStockLevel >= @StockLevel;
GO
```

Explanation: Each batch redeclares `@StockLevel` to demonstrate batch scope and proper variable management.

3. **Complex Expression with Batching and Comments**

   ```sql
   USE AdventureWorks2022;
   -- Batch 1: Filter by price and color
   DECLARE @MaxPrice DECIMAL(10,2) = 1000.00;
   DECLARE @Color NVARCHAR(15) = 'Black';
   /* Calculate discounted price for high-value black products */
   SELECT ProductID, Name, ListPrice, (ListPrice * 0.90) AS DiscountedPrice
   FROM Production.Product
   WHERE ListPrice <= @MaxPrice AND Color = @Color;
   GO
   -- Batch 2: Change color and repeat
   DECLARE @MaxPrice DECIMAL(10,2) = 1000.00;
   DECLARE @Color NVARCHAR(15) = 'Red';
   /* Same query with different color filter */
   SELECT ProductID, Name, ListPrice, (ListPrice * 0.90) AS DiscountedPrice
   FROM Production.Product
   WHERE ListPrice <= @MaxPrice AND Color = @Color;
   GO
   ```

   Explanation: Demonstrates batching to separate queries with different parameters, using variables, expressions, and comments.

Class Work

1. **Identification Activity**:
   - **Task**: Analyze the following script and identify variables, operators, predicates, expressions, comments, and batches:

     ```sql
     USE AdventureWorks2022;
     -- Batch 1: Set threshold for product weight
     DECLARE @MinWeight DECIMAL(10,2) = 10.00;
     /* Filter heavy products and calculate shipping cost */
     SELECT ProductID, Name, Weight, (Weight * 0.05) AS ShippingCost
     FROM Production.Product
     WHERE Weight > @MinWeight AND ProductSubcategoryID = 1;
     GO
     -- Batch 2: Update weight threshold
     DECLARE @MinWeight DECIMAL(10,2) = 20.00;
     SELECT ProductID, Name, Weight, (Weight * 0.05) AS ShippingCost
     FROM Production.Product
     WHERE Weight > @MinWeight AND ProductSubcategoryID = 1;
     ```

- **Deliverable**: List each element with examples from the script.
- **Expected Output**:
  - Variables: `@MinWeight`
  - Operators: `>` , `AND` , `*`
  - Predicate: `Weight > @MinWeight AND ProductSubcategoryID = 1`
  - Expression: `(Weight * 0.05)`
  - Comments: `-- Batch 1: Set threshold for product weight` , `/* Filter heavy products and calculate shipping cost */` , `-- Batch 2: Update weight threshold`
  - Batches: Two batches separated by `GO`

2. **Lab Exercise**:
   - **Task**: Write a T-SQL script with two batches:
     - Batch 1: Declare a variable `@MaxPrice` (type `DECIMAL(10,2)`) set to 1500.00. Query `Production.Product` for `ProductID` , `Name` , `ListPrice` , and a computed column `TaxedPrice` (add 8% tax: `ListPrice * 1.08`). Filter where `ListPrice <= @MaxPrice` .
     - Batch 2: Update `@MaxPrice` to 2000.00 and repeat the query.
     - Include single-line and multi-line comments.
     - Save as `LanguageElements.sql` .
   - **Deliverable**: Submit the `.sql` file with both batches.
3. **Challenge Exercise**:
   - **Task**: Write a script with three batches:
     - Batch 1: Declare `@MinStock` (type `INT`) set to 100 and query `Production.Product` for `ProductID` , `Name` , and `SafetyStockLevel` where `SafetyStockLevel > @MinStock` .
     - Batch 2: Update `@MinStock` to 300 and repeat the query.
     - Batch 3: Declare `@MinStock` and `@Color` (type `NVARCHAR(15)`) set to 'Black', and query `Production.Product` for products matching both conditions.
     - Include comments and save as `BatchExample.sql` .
   - **Deliverable**: Submit the `.sql` file with all batches.

Sub-Session 2: Sets and Predicate Logic

Definition

- **Sets**: Collections of rows (e.g., table or query results) manipulated using T-SQL set operations like `UNION` , `INTERSECT` , and `EXCEPT` .
- **Predicate Logic**: Logical conditions in `WHERE` or `HAVING` clauses that filter data based on `TRUE` or `FALSE` evaluations.

Explanation

- **Sets**:
  - SQL Server treats tables and query results as sets, enabling operations on groups of rows.
  - **Set Operations**:
    - `UNION` : Combines rows from two queries, removing duplicates.
    - `UNION ALL` : Combines rows without removing duplicates (faster).
    - `INTERSECT` : Returns rows common to both queries.
    - `EXCEPT` : Returns rows in the first query but not the second.
- **Predicate Logic**:
  - Predicates are conditions (e.g., `ListPrice > 200`).
  - Logical operators (`AND` , `OR` , `NOT`) combine predicates for complex filtering.
  - Used in `WHERE` (row-level filtering) and `HAVING` (group-level filtering).

Examples

1. **UNION ALL Set Operation with Batching**

```sql
USE AdventureWorks2022;
-- Batch 1: Combine red products
DECLARE @Color1 NVARCHAR(15) = 'Red';
SELECT ProductID, Name, Color
FROM Production.Product
WHERE Color = @Color1;
GO
-- Batch 2: Combine with blue products using UNION ALL
DECLARE @Color2 NVARCHAR(15) = 'Blue';
SELECT ProductID, Name, Color
FROM Production.Product
WHERE Color = @Color1
UNION ALL
SELECT ProductID, Name, Color
FROM Production.Product
WHERE Color = @Color2;
```

Explanation: Uses batching to separate a single-color query from a combined query with `UNION ALL` .

2. **INTERSECT Set Operation**

```
USE AdventureWorks2022;
-- Find products that are both expensive and in a specific subcategory
SELECT ProductID, Name
FROM Production.Product
WHERE ListPrice > 1000
INTERSECT
SELECT ProductID, Name
FROM Production.Product
WHERE ProductSubcategoryID = 1;
```

Explanation: Returns products that satisfy both conditions.

3. **EXCEPT Set Operation**

```
USE AdventureWorks2022;
-- Find products in subcategory 2 but not silver
SELECT ProductID, Name
FROM Production.Product
WHERE ProductSubcategoryID = 2
EXCEPT
SELECT ProductID, Name
FROM Production.Product
WHERE Color = 'Silver';
```

Explanation: Excludes silver products from subcategory 2.

4. **Complex Predicate Logic with Batching**

```
USE AdventureWorks2022;
-- Batch 1: High-value orders in 2019
DECLARE @MinTotalDue MONEY = 3000;
SELECT SalesOrderID, OrderDate, TotalDue
FROM Sales.SalesOrderHeader
WHERE TotalDue > @MinTotalDue AND OrderDate BETWEEN '2019-01-01' AND '2019-12-31';
GO
-- Batch 2: Include specific customer
DECLARE @MinTotalDue MONEY = 3000;
DECLARE @CustomerID INT = 11001;
SELECT SalesOrderID, OrderDate, TotalDue
FROM Sales.SalesOrderHeader
WHERE TotalDue > @MinTotalDue AND (OrderDate BETWEEN '2019-01-01' AND '2019-12-31' OR Custome
```

Explanation: Uses batching to separate queries with different predicate logic.

Class Work

1. **Set Operation Lab**:
   - **Task**: Write a script with two batches:
     - Batch 1: Query `Production.Product` for `ProductID`, `Name`, and `Color` where `Color = 'Black'`.
     - Batch 2: Use `UNION` to combine products where `Color = 'Black'` and `Color = 'Silver'`. Include comments and save as `SetOperationUnion.sql`.
     - **Bonus**: Rewrite Batch 2 using `UNION ALL` and compare performance with `SET STATISTICS TIME ON;`.
2. **Predicate Logic Exercise**:
   - **Task**: Write a query to retrieve `Sales.SalesOrderHeader` records where `TotalDue > 1500` and `OrderDate` is between January 1, 2020, and June 30, 2020. Include `SalesOrderID`, `OrderDate`, and `TotalDue`. Use `AND` and `BETWEEN`. Save as `PredicateLogicSales.sql`.
3. **Challenge Exercise**:
   - **Task**: Write a script with two batches:
     - Batch 1: Use `INTERSECT` to find products in `Production.Product` with `ListPrice > 800` and `SafetyStockLevel > 400`.
     - Batch 2: Use `EXCEPT` to find products with `ListPrice > 800` but not in `ProductSubcategoryID = 2`.
     - Save as `SetOperationsAdvanced.sql`.

Sub-Session 3: Logical Order of Operators in the SELECT Statement

Definition

The logical order of operators in a T-SQL `SELECT` statement is the sequence in which SQL Server processes clauses to produce results, distinct from the written order.

Explanation

- **Logical Order**:
  i. **FROM**: Specifies the table(s) or view(s) to query, including joins.
  ii. **WHERE**: Filters individual rows based on predicates.
  iii. **GROUP BY**: Groups rows by specified columns.
  iv. **HAVING**: Filters groups based on aggregate conditions.
  v. **SELECT**: Specifies columns or computed expressions to return.

vi. **ORDER BY**: Sorts the final result set.

- **Key Points**:
    - The written order (`SELECT` , `FROM` , etc.) is for readability; logical order determines execution.
    - Aliases in `SELECT` cannot be used in `WHERE` or `GROUP BY` because `SELECT` is processed later.
    - Batches can affect query execution by resetting session state (e.g., variable scope).

Examples

1. **Basic SELECT with Logical Order**

```
USE AdventureWorks2022;
SELECT ProductID, Name, ListPrice
FROM Production.Product
WHERE ListPrice > 200
ORDER BY ListPrice DESC;
```

Logical Processing:

- FROM: Access `Production.Product` .
- WHERE: Filter rows where `ListPrice > 200` .
- SELECT: Retrieve `ProductID` , `Name` , `ListPrice` .
- ORDER BY: Sort by `ListPrice` descending.

2. **GROUP BY and HAVING with Batching**

```
USE AdventureWorks2022;
-- Batch 1: Group by subcategory with threshold
DECLARE @MinPrice DECIMAL(10,2) = 0;
SELECT ProductSubcategoryID, COUNT(*) AS ProductCount, AVG(ListPrice) AS AvgPrice
FROM Production.Product
WHERE ListPrice > @MinPrice
GROUP BY ProductSubcategoryID
HAVING COUNT(*) > 5
ORDER BY AvgPrice DESC;
GO
-- Batch 2: Increase price threshold
DECLARE @MinPrice DECIMAL(10,2) = 100;
SELECT ProductSubcategoryID, COUNT(*) AS ProductCount, AVG(ListPrice) AS AvgPrice
FROM Production.Product
WHERE ListPrice > @MinPrice
GROUP BY ProductSubcategoryID
HAVING COUNT(*) > 5
ORDER BY AvgPrice DESC;
```

Logical Processing:

- FROM: Access `Production.Product`.
- WHERE: Filter rows where `ListPrice > @MinPrice`.
- GROUP BY: Group by `ProductSubcategoryID`.
- HAVING: Keep groups with more than 5 products.
- SELECT: Compute `COUNT(*)` and `AVG(ListPrice)`.
- ORDER BY: Sort by `AvgPrice` descending.

  Explanation: Batches separate queries with different `@MinPrice` values.

3. **Join with Logical Order**

```sql
USE AdventureWorks2022;
SELECT p.ProductID, p.Name, sc.Name AS Subcategory
FROM Production.Product p
JOIN Production.ProductSubcategory sc ON p.ProductSubcategoryID = sc.ProductSubcategoryID
WHERE p.ListPrice > 600
ORDER BY p.ListPrice DESC;
```

Logical Processing:

- FROM: Join `Production.Product` and `Production.ProductSubcategory`.
- WHERE: Filter rows where `ListPrice > 600`.
- SELECT: Retrieve `ProductID`, `Name`, and `Subcategory`.
- ORDER BY: Sort by `ListPrice` descending.

Class Work

1. **Logical Order Analysis**:
   - **Task**: For the following query, describe the logical order of processing:

   ```sql
   USE AdventureWorks2022;
   SELECT CustomerID, COUNT(*) AS OrderCount, SUM(TotalDue) AS TotalSales
   FROM Sales.SalesOrderHeader
   WHERE OrderDate >= '2020-01-01'
   GROUP BY CustomerID
   HAVING SUM(TotalDue) > 5000
   ORDER BY TotalSales DESC;
   ```

   - **Deliverable**: List each step (FROM, WHERE, etc.) with a brief explanation.
2. **Lab Exercise**:
   - **Task**: Write a script with two batches:
     - Batch 1: Query `Production.Product` for `ProductSubcategoryID` and total `ListPrice` (as `TotalPrice`) where `ListPrice > 300`. Group by `ProductSubcategoryID`, include groups

with total `ListPrice > 8000`, and sort by total price descending.
    - Batch 2: Repeat with `ListPrice > 500` and total `ListPrice > 10000`.
    - Save as `LogicalOrderQuery.sql`.
3. **Challenge Exercise**:
    - **Task**: Write a script with two batches:
        - Batch 1: Join `Sales.SalesOrderHeader` and `Sales.SalesOrderDetail` to compute total `LineTotal` per `SalesOrderID` for orders in 2020. Use `GROUP BY`, `HAVING` (total `LineTotal > 12000`), and `ORDER BY`.
        - Batch 2: Repeat for orders in 2021 with `LineTotal > 15000`.
        - Save as `OrderTotalQuery.sql`.

Additional Notes

- **Setup**: Ensure the AdventureWorks2022 database is installed in SSMS. Download from Microsoft's official source if needed.
- **Best Practices**:
    - Use `SELECT TOP 10` for testing queries on large tables to avoid performance issues.
    - Include descriptive comments in all scripts.
    - Use `GO` to separate batches clearly, especially when variables need to be redefined.
    - Verify data types when declaring variables (e.g., `DECIMAL(10,2)` for prices).
- **Debugging**:
    - If a query fails, check for syntax errors, invalid column names, or variable scope issues (e.g., using a variable outside its batch).
    - Use `PRINT` statements to debug variable values (e.g., `PRINT @MaxPrice;`).
- **Batching Tips**:
    - Variables are scoped to their batch; redeclare them in each batch if needed.
    - Use batches to separate logical operations or test different parameter values.
- **Resources**: Refer to Microsoft's SQL Server documentation for AdventureWorks2022 schema details.

Assessment Criteria

- **Correctness**: Queries produce expected results with no syntax errors.
- **Documentation**: Scripts include clear single-line and multi-line comments.
- **Batching**: Scripts use `GO` appropriately to demonstrate batch separation.
- **Completeness**: All classwork tasks are completed and saved with correct filenames.

- **Understanding**: Students can explain the logical order, batching, and purpose of their queries.

Next Steps

- Review query results in SSMS to understand data patterns in AdventureWorks2022.
- Explore advanced T-SQL features like stored procedures and transactions in the next session.
- Practice combining batching with set operations and joins for complex scripts.