T-SQL Class Notes: Variables, Expressions, and Control-of-Flow (Sections 4.5.3 - 4.5.5)

4.5.3 Variables

Overview

A variable in Transact-SQL (T-SQL) is an object that holds a data value. Variables are categorized into local and global types. Local variables, prefixed with '@', are used for temporary storage during statement execution and can pass data to SQL statements. Global variables, prefixed with '@@', are system-defined functions in SQL Server 2019, providing metadata or configuration settings.

Examples

Local Variable Example

```sql
-- Declare a local variable to store employee count
DECLARE @EmployeeCount INT;
-- Set the value by querying AdventureWorks2022
SET @EmployeeCount = (SELECT COUNT(*) FROM AdventureWorks2022.HumanResources.Employee);
-- Display the result
SELECT @EmployeeCount AS TotalEmployees;
-- Comment: This stores and retrieves the total number of employees.
```

Global Variable Example

```sql
-- Retrieve the current server language
SELECT @@LANGUAGE AS CurrentLanguage;
-- Comment: Returns the language setting, e.g., 'us_english' at 11:43 AM WAT on June 24, 2025.
```

Common Global Variables

- `@@DATEFIRST` : Returns the first day of the week (e.g., 1 for Monday).
- `@@LANGUAGE` : Returns the current language.
- `@@LOCK_TIMEOUT` : Returns the current lock timeout setting.
- `@@MAX_CONNECTIONS` : Returns the maximum number of connections allowed.
- `@@MAX_PRECISION` : Returns the maximum precision for numeric data.
- `@@SERVERNAME` : Returns the name of the server.
- `@@VERSION` : Returns the SQL Server version.

Analogy

Think of local variables as temporary storage bins you use during a cooking session (e.g., measuring ingredients), while global variables are like pre-set kitchen timers or labels (e.g., oven temperature) provided by the kitchen itself.

4.5.4 Expressions

Overview

An expression is a combination of identifiers, values, and operators evaluated by SQL Server to produce a result. Expressions are versatile, used in SELECT, WHERE, and UPDATE statements for data access or modification.

Examples

Simple Expression

```sql
-- Calculate total sales with a tax rate
SELECT SalesOrderID, TotalDue * 1.15 AS TotalWithTax
FROM AdventureWorks2022.Sales.SalesOrderHeader;
-- Comment: Multiplies TotalDue by 1.15 (15% tax) to estimate total with tax.
```

Complex Expression

```sql
-- Combine date parts into a custom format
SELECT BusinessEntityID,
       YEAR(OrderDate) * 10000 + MONTH(OrderDate) * 100 + DAY(OrderDate) AS CustomDateCode
FROM AdventureWorks2022.Sales.SalesOrderHeader;
-- Comment: Creates a numeric date code (e.g., 20250624 for June 24, 2025).
```

Use Case

Expressions are like mathematical formulas in a spreadsheet, allowing dynamic calculations (e.g., profit margins) without storing intermediate results.

4.5.5 Control-of-Flow, Errors, and Transactions

Overview

T-SQL supports control-of-flow statements to manage execution flow, handle errors, and manage transactions, extending beyond data retrieval.

Control-of-Flow Statements

Statement	Description
`IF...ELSE`	Branches execution based on a condition.
`WHILE`	Repeats statements while a condition is true.
`BEGIN...END`	Groups statements into a block.
`TRY...CATCH`	Handles exceptions and errors.
`BEGIN TRANSACTION`	Marks a block as a transaction.

Examples

IF...ELSE Example

```
-- Check if today is a weekend
IF DATENAME(weekday, GETDATE()) IN (N'Saturday', N'Sunday')
    SELECT 'It is a Weekend';  -- Executes if condition is true
ELSE
    SELECT 'It is a Weekday';  -- Executes if condition is false
-- Comment: At 11:43 AM WAT on June 24, 2025, this returns 'It is a Weekday' (Tuesday).
```

WHILE Loop Example

```
-- Count down from 5
DECLARE @Counter INT = 5;
WHILE @Counter > 0
BEGIN
    PRINT 'Counter: ' + CAST(@Counter AS VARCHAR(2));
    SET @Counter = @Counter - 1;
END;
-- Comment: Loops 5 times, printing 5, 4, 3, 2, 1.
```

TRY...CATCH Example

```
-- Attempt division that might cause an error
BEGIN TRY
    SELECT 10 / 0 AS Result;  -- This will raise a divide-by-zero error
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
-- Comment: Catches the error and returns details instead of failing.
```

Transaction Example with Batching

```sql
-- Start a transaction to update sales data
BEGIN TRANSACTION;
    UPDATE AdventureWorks2022.Sales.SalesOrderHeader
    SET TotalDue = TotalDue * 1.10  -- Increase by 10%
    WHERE OrderDate >= '2025-06-01';
    -- Batch concept: Process in chunks to manage large datasets
    IF @@ROWCOUNT = 0
        BEGIN
            PRINT 'No rows updated.';
            ROLLBACK TRANSACTION;  -- Undo if no changes
        END
    ELSE
        COMMIT TRANSACTION;  -- Confirm changes
-- Comment: Uses batching to handle updates safely; @@ROWCOUNT tracks affected rows.
```

Concepts and Use Cases

- **Batching**: Splitting large operations into smaller chunks to avoid performance issues or timeouts. Analogy: Packing a suitcase in batches rather than all at once to fit everything.
- **Transactions**: Ensure data integrity (e.g., bank transfers where both debit and credit must succeed). Use `BEGIN TRANSACTION`, `COMMIT`, and `ROLLBACK`.
- **Error Handling**: Like a safety net, `TRY...CATCH` prevents crashes and logs issues for debugging.

Classwork

1. **Variables Task**:
   Declare a local variable `@TotalSales` and set it to the sum of `TotalDue` from `Sales.SalesOrderHeader` for 2025. Display the result.
2. **Expression Task**:
   Write an expression to calculate the profit (TotalDue - SubTotal) for each order in `Sales.SalesOrderHeader`.
3. **Control-of-Flow Task 1 (IF...ELSE)**:
   Use `IF...ELSE` to check if the current hour (from `DATEPART(hour, GETDATE())`) is before noon and print 'Morning' or 'Afternoon'.

4. **Control-of-Flow Task 2 (WHILE)**:
   Use a `WHILE` loop to print the first 10 numbers, incrementing a local variable `@Number` starting from 1.
5. **Transaction Task**:
   Create a transaction to update `ListPrice` by 5% in `Production.Product` where `ListPrice` > 1000, and roll back if no rows are affected.

Additional Notes

- Global variables like `@@VERSION` provide system insights, useful for auditing.
- Expressions enhance flexibility, e.g., dynamic pricing calculations.
- Control-of-flow statements are like programming logic, enabling conditional actions or loops in SQL.