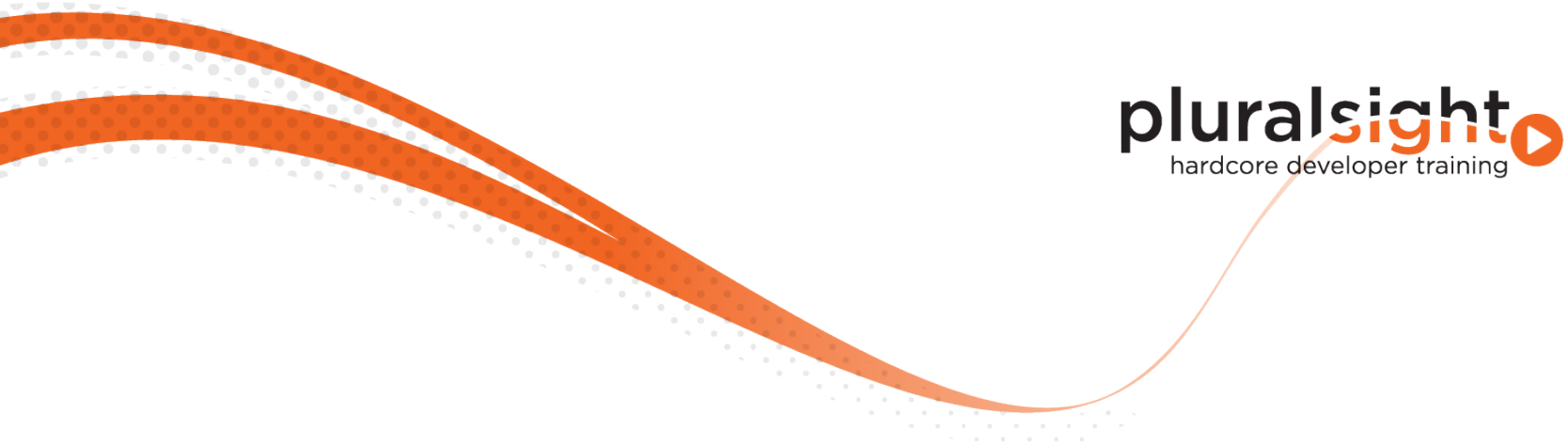
SQL Server: Transact-SQL Basic Data Retrieval

Module 3: Writing a Basic SELECT Statement

Joe Sack

Joe@SQLskills.com



Module Introduction

- **Microsoft SQL Server 2012 = Microsoft's flagship RDBMS**
- **Transact-SQL is Microsoft's version of SQL (based on ANSI-standard SQL)**
- **Transact-SQL is, *for the most part*, a declarative language – describing what is wanted but not how it should be processed**
  - There are exceptions – such as using query hints
- **This module introduces you to the key elements of a SELECT statement**

Thinking in “Sets”

- **If you’re coming in from a programmer’s background, or even if this is your first language you’re learning, start thinking in “sets”**
  - Set = a collection of objects/elements
- **Thinking in “sets” means you’re not focusing on the handling of specific, individual elements**
- **Coming from a programmer’s background, you may be tempted to work with one element (row) at a time, but you’ll learn that Transact-SQL is optimized for set-based operations**
  - Tip: Avoid “RBAR” (Row By Agonizing Row) Transact-SQL programming! This is an efficient method and is likely to create performance bottlenecks as your applications grow over time

Where to Begin?

```
<SELECT statement> ::=
    [WITH <common_table_expression> [,...n]]
    <query_expression>
    [ ORDER BY { order_by_expression | column_position [ ASC | DESC ] }
  [ ,...n ] ]
    [ <FOR Clause>]
    [ OPTION ( <query_hint> [ ,...n ] ) ]
<query_expression> ::=
    { <query_specification> | ( <query_expression> ) }
    [  { UNION [ ALL ] | EXCEPT | INTERSECT }
        <query_specification> | ( <query_expression> ) [...n ] ]
<query_specification> ::=
SELECT [ ALL | DISTINCT ]
    [TOP ( expression ) [PERCENT] [ WITH TIES ] ]
    < select_list >
    [ INTO new_table ]
    [ FROM { <table_source> } [ ,...n ] ]
    [ WHERE <search_condition> ]
    [ <GROUP BY> ]
    [ HAVING < search_condition > ]
```

Source: SQL Server 2012 Books Online
http://bit.ly/cOtADI

SELECT Clause

- **Allows you to define which columns are returned in a query**
- **While you can specify "SELECT *", it is recommend you explicitly define the columns whenever possible**
  - Significant performance impacts in some cases
  - Application resiliency to base table/view changes
- **SELECT can reference columns from a table, view and more**
- **SELECT can also reference an expression**
  - Constant / function / column combination connected by operators
- **SELECT can reference a sub-query**
- **SELECT can reference a literal or scalar value (a constant)**

Column Aliases

- **You can define an alias to replace the original column name or provide a column name for an expression where none already exists**
- **Use aliases to simplify, clarify or reduce the size of the original data source column name**
- **Use "AS" clause to define the alias**
  - Other alias methods exist, but AS is a popular method

```
SELECT    '1' AS [BatchCode],
          [Name] AS [DepartmentName]
FROM [HumanResources].[Department];
```

Identifiers

- **Database objects are referred to in queries using their object name**
- **Object names are also referred to as identifiers**
- **"Regular" identifiers refer to names that comply with specific rules**
  - First character must be a letter, (_), (@), #
  - Must not be a Transact-SQL reserved word (upper or lower case)
  - No embedded spaces, special or supplementary characters
- **"Delimited" identifiers are enclosed in double quotation marks or square brackets**
  - They must be used for identifiers that don't comply with all rules
  - They can still be used for those that DO comply

Semicolon Statement Terminator

- **Transact-SQL statement terminator**
  - Not required for most statements – BUT – it will be required in the future
- **Get in the habit of using it with all statements, because it will eventually be required**
- **Already required for Common Table Expressions and Service Broker Commands**
  - Common Table Expressions, in the context of a batch, the statement before it must be followed by a semicolon
  - Service Broker commands

FROM Clause

- **Defines the data source for your SELECT**
- **Examples of data sources include:**
  - Tables
  - Views
  - Derived tables (sub-query)
  - Table variables
  - Functions (table-valued function)
- **Maximum of 256 data sources per statement, but we'll be starting off with just one in this module**
  - Tip: 256 is a limit – not a goal! You'll want to make the Query Optimizer's task easier.

Table Aliases

- **You can designate an alias for the original data source**
- **This allows you to use a more compact name**
- **Allows for simplification of the name (ORD001T -> orders)**
- **Can be used for self-join scenarios**
  - dbo.Employee AS e1
  - dbo.Employee AS e2
- **Tip: Use alias names that have a logical association with the table name. For example, using an alias of "Q" doesn't make sense for a table named "Employees"**

Expressions, Operators, Predicates

- **Expression**
  - Symbols and operators evaluated to produce a single data value
- **Operator**
  - Symbol specifying an action applied to an expression (or expressions)
- **Predicate is an expression that can evaluate to**
  - TRUE
  - FALSE
  - UNKNOWN
- **Predicates can be a search condition in WHERE or HAVING, or a join condition in FROM**
  - Play a big role in your indexing strategy and query performance

WHERE Clause

- **Define which rows are returned by the statement**
- **One or more predicates**
- **Predicates can use logical operators between predicates**
  - AND
    - When both conditions are TRUE, evaluates to TRUE
  - OR
    - When either condition is TRUE, evaluates to TRUE
  - NOT
    - Negates a boolean expression
- **Multiple conditions?**
  - You can define the evaluation order of expressions using parentheses
  - When nested, inner-most expressions are evaluated first

DISTINCT

- Adding a DISTINCT to a SELECT clause removes duplicate rows from the final result set
- NULL values are treated as equal
- While DISTINCT can be useful for reporting, be sure that such an operation is actually needed, as there is performance overhead associated with the clause

TOP

- **Limits rows returned for your query**
  - Commonly used in conjunction with ORDER BY for predictability purposes
  - Replaces deprecated SET ROWCOUNT command
- **You can specify by rows or percentage**
- **If percentage, then fractional values get rounded up**
- **WITH TIES returns additional rows that match the values of the last row returned based on ORDER BY clause**
- **Can also be used for INSERT / UPDATE / MERGE / DELETE**
  - Recommended you use parentheses for SELECT for consistency's sake, as it is required for the other DML statements

GROUP BY

- **Groups rows into a summarized set, with one row per group**
- **Typically used in conjunction with aggregate functions in the SELECT clause**
- **Grouping is by one or more non-aggregated columns or expressions**
- **Can be used in conjunction with GROUPING SETS**
  - To specify multiple groups of data in one query
- **Predicates can be applied to groups with HAVING**

HAVING Clause

- **HAVING applies a predicate to a group**
- **Used just with SELECT statements**
  - Tip: WHERE predicates can often be “pushed down” the query execution path, improving performance
  - If logically your predicate can be applied in the WHERE clause instead of a GROUP BY, err on the side of the WHERE clause

ORDER BY Clause

- Sorts the data for the SELECT statement
- ORDER BY can specify columns and/or expressions
- Can reference ordinal column position – but this is NOT recommended due to readability problems
- Can define an ascending (ASC) or descending (DESC) ordering for each of the referenced sorted columns
- You can also designate collations (Windows collation or SQL collation) for char, varchar, nchar, and nvarchar columns
- Tip: Don't rely on "natural" sorting based on the referenced table indexes – as this order is *not* guaranteed (common myth)!

Query Paging

- **Often your application will need to "page" through a range of rows given a specific row count offset in the results**
- **SQL Server 2012 extended the ORDER BY clause to allow for simpler to code paging**
  - OFFSET specifies the number of rows to skip before starting
  - FETCH specifies the number of rows after the OFFSET to return
- **Note: While OFFSET…FETCH is easier to construct than other methods, it is unlikely to perform better than other methods used prior to SQL Server 2012**

Binding Order

- **SQL Server honors a binding order for the SELECT statement and objects surfaced at one step are available to reference in consecutive steps**
  1. FROM
  2. ON
  3. JOIN
  4. WHERE
  5. GROUP BY
  6. WITH CUBE or WITH ROLLUP (deprecated)
  7. HAVING
  8. SELECT
  9. DISTINCT
  10. ORDER BY
  11. TOP

Best Practice: Commenting Your Code

- **When creating scripts, comment your code**
  - /* … */
    - "…" represents the text of the comment and is not evaluated by SQL Server
    - Often used for multi-line comments
  - --
    - Two hyphens for single-line comments
- **Commenting enhances the supportability of your Transact-SQL code for those who inherit it later (or even for yourself months or years from now)**
- **Especially important to comment code that is non-standard, allowing you to explain WHY you wrote it the way you did**