SQL Server: Transact-SQL Basic Data Retrieval

Module 2: Setting up a Transact-SQL Learning Environment

Joe Sack

Joe@SQLskills.com

pluralsight

hardcore developer training

Module Introduction

- **The best way to learn Transact-SQL is to do hands-on practice as you follow along in this course**

- **In this module:**
  - Installing SQL Server 2012 Express with Tools
  - Installing SQL Server Data Tools
  - Downloading and attaching the sample database
  - Executing a basic data retrieval query in SSMS and SSDT

SQL Server 2012 Express with Tools

- **In order to practice the concepts we discuss in this course, I recommend installing an instance of the SQL Server 2012 Express database engine**
- **SQL Server 2012 Express is a *free* edition that can be used for learning the product and features**
- **We'll be using SQL Server 2012 Express with Tools**
  - SQL Server 2012 Database Engine
  - SQL Server Management Studio Express

SQL Server 2012 Express (Abridged) Requirements

- **Choose a laptop, desktop or test server with the following:**
  - An internet connection from the computer you’re installing it on
  - A minimum of 6 GB of hard-disk space
  - Minimum 512 MB RAM
    - Recommended 1 GB
  - Minimum of 1.0 GHz (x86) or 1.4 GHz (x64)
    - Recommended 2.0 GHz or faster
  - Storage types supported
    - Local disk
    - Shared storage (SAN, for example)
    - SMB File Share

SQL Server 2012 Express (Abridged) Requirements

- **OS versions (see http://bit.ly/zKHAKL for full details):**
  - Windows Server 2008 R2 SP1 64-bit (Standard, Enterprise and more)
  - Windows 7 SP1 32-bit and 64-bit (Home Basic, Home Premium, Enterprise, Professional and more)
  - Windows Server 2008 SP2 64-bit and 32-bit (Standard, Enterprise and more)
  - Windows Vista SP2 32-bit and 64-bit (Home Basic through Ultimate and more)

- **You can download from http://bit.ly/KJIAMR or search on the keywords "Download Microsoft SQL Server 2012 Express with Tools"**
  - Tip: SQLEXPRWT_x64_ENU.exe is the "Express with Tools" version (and there is an x86 version as well)

SQL Server Management Studio Express

- **SQL Server Management Studio (SSMS) Express is a free graphical management tool that is commonly used both by Database Administrators *and* Developers**
  - Manage the database engine plus Transact-SQL query execution
- **SQL Server Data Tools (SSDT) does offer more developer-centric functionality not provided in SSMS**

SQL Server Data Tools

- As an alternative or supplement to SQL Server Management Studio (SSMS), you can also use SQL Server Data Tools (SSDT) in order to familiarize yourself with SQL Server 2012 Transact-SQL
- Whereas SSMS bridges both the administrator and developer worlds, SSDT is specific to the database developer audience, allowing for database design and query development
- If you've worked with Visual Studio before, SSDT will already be familiar to you
- SSDT is also *free* and you can download from http://bit.ly/bIGImY or search on the keywords "SQL Server Data Tools"

SQL Server Data Tools OS Requirements

- **OS versions (see http://bit.ly/zKHAKL for details):**
  - Windows Server 2008 R2 SP1 64-bit (Standard, Enterprise and more)
  - Windows 7 SP1 32-bit and 64-bit (Home Basic, Home Premium, Enterprise, Professional and more)
  - Windows Server 2008 SP2 64-bit and 32-bit (Standard, Enterprise and more)
  - Windows Vista SP2 32-bit and 64-bit (Home Basic through Ultimate and more)

AdventureWorks2012 Sample Database

- **CodePlex (http://www.codeplex.com/) provides various sample databases which you can use to follow along with the Transact-SQL examples in this course**
- **AdventureWorks2012 database**
- **You can download from http://msftdbprodsamples.codeplex.com/ or search on the key words "Adventure Works for SQL Server 2012"**
  - □ Download from the "AdventureWorks2012 Data File" link
- **I'll show you how to attach the database in the next demo**

Basic data retrieval query in SSMS and SSDT

- **SSMS or SSDT for data retrieval – which one should you use?**

- **The next two demos will show you a basic SELECT query using both tools, and then you can decide for yourself**
  - Don’t worry about understanding all the mechanics of the data retrieval examples – we’ll be walking through the key data retrieval concepts throughout this course

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**

SQL Server: Transact-SQL Basic Data Retrieval

Module 4: Querying Multiple Data Sources

Joe Sack

Joe@SQLskills.com

Module Introduction

- **Build on previous module fundamentals by showing the various ways you can reference multiple data sources in a single query**
- **We'll cover:**
  - INNER, OUTER, CROSS Joins
  - Self Joins
  - CROSS APPLY
  - UNION
  - INTERSECT/EXCEPT
  - Common Table Expressions

Inner Joins

- **Given two data sources, inner joins return all matching pairs of rows**
  - Unmatched rows are discarded
- **INNER JOIN is the default join type if you just specify “JOIN” keyword**
  - Optional, but be consistent
- **Use the ANSI SQL-92 standard syntax**
  - Join condition is used to filter rows in the ON clause
  - ANSI SQL-89 syntax specifies the join condition in the WHERE clause

Outer Joins

- **LEFT OUTER returns all rows from the left table that match the right table and also rows from the left table that do NOT match**
  - Output columns for unmatched rows are returned as NULL
- **RIGHT OUTER returns all rows from the right table that match the left table and also rows from the right table that do NOT match**
  - Output columns for unmatched rows are returned as NULL
- **FULL OUTER returns all rows from the right and left table that match and also any unmatched rows from either table**
  - Output columns for unmatched rows are returned as NULL

Outer Joins (2)

- **OUTER keyword is optional, but may help with readability of the query**
  - Choose a standard and then be consistent
- **Put the predicate that determines the join condition in the ON clause**
- **Put the predicate that specifies the *outer* row filter in the WHERE clause**

Cross Joins

- **Cross-product of two data sources**
  - Also referred to as a Cartesian product
  - Each row from a data source is matched with ALL rows in the other data source
    - Data Source 1 multiplied by Data Source 2
- **Tip: you’ll often see cross joins used in order to generate “number tables”**
  - Number sequences
  - Test data sets
- **Cross joins may also be the result of poor join construction**

Self Joins

- **You can join a data source to itself**
- **Use table aliases to allow data sources to be referenced separately**
- **Supported for INNER, OUTER and CROSS joins**
- **Example:**
  - Recursive hierarchy, such as a Manager/Employee relationship

Equi vs. Non-Equi Joins

- **Examples of join conditions that you've seen so far in this course have been for "equi-joins"**
  - Join condition uses an equality operator
    - Table1.column1 = Table2.column1
- **Non-equi join involves non-equality join conditions – for example:**
  - <>, >, <, >=, <=
- **Non-equi join conditions can be coupled with equi join conditions – for example:**
  - An equi join on a primary key and foreign key relationship and…
    - A non-equi join based on dates in the left table being greater than dates in the right

Multi-Attribute Joins

- Your join condition can involve more than one column from each input
- Multi-attribute joins are commonly used for primary key/foreign key associations between data sources involving more than one attribute on each side

Joining More than Two Tables

- **When joining more than 2 tables, multiple JOIN operators are used in the query**
- **When multiple tables are joined, they are processed *logically* in ordinal position**
  - The cost-based query optimizer may physically re-order your joins from a physical perspective, but the intended result set will be correct
- **When using OUTER joins, the order in which you write the various JOIN operators matters, as I'll demonstrate next**

CROSS APPLY Operator

- **Execute a table-valued function (TVF) or sub-query for each row returned by the outer (left) data source input**
  - Tip: common to use CROSS APPLY with Dynamic Management Objects introduced in SQL Server 2005
- **CROSS APPLY only return left-input rows that produce TVF results**
- **OUTER APPLY returns matched and unmatched left-input rows**
  - Unmatched TVF columns set to NULL

Joining Sub-queries

- Defined in the FROM clause within parentheses
- Aliased (named) via the AS clause
- Can be joined like any other data source

Defining the Sub-query

- **Not persisted physically**
- **ORDER BY not permitted**
- **Requires explicit, unique column names**
  - For example, an aggregated column must be given a column alias name
- **A sub-query can be “correlated”**
  - This is when the sub-query refers to columns from the *outer* query
- **When a sub-query is not correlated, it can be executed on its own**

UNION Operator

- **Combines two or more SELECT statements into a single result set**
    - UNION eliminates duplicates
    - UNION ALL retains each data set, including duplicates
        - Tip: When UNION ALL is acceptable, specify it (eliminating a potentially unnecessary de-duplication and thus helping query performance)
- **Requirements of UNION:**
    - Same number of expressions
        - Columns / expressions / aggregates
    - Data types can differ, but aligned columns must allow for implicit data conversion, for example:
        - decimal and numeric = OKAY
        - datetime2 and varbinary = NOT OKAY

INTERSECT and EXCEPT Operators

- **INTERSECT and EXCEPT compare the results of two SELECT statements**
  - INTERSECT returns distinct values returned by both the left and right sides, eliminating unmatched values
  - EXCEPT returns distinct values from the left SELECT that are not found in the right
- **Tip: You can use these to help find data source discrepancies**
  - Test whether one query has the same logical results as another
  - Find missing rows – for example – if a data extraction process is suspected to be faulty

Data Types and Joining Tables

- **Be very aware of the data types of attributes you're joining between data sources**
  - Table column data types, parameters, variables
- **Ideally the data types are identical**
- **When they are NOT identical, but still compatible, implicit data type conversion occurs**
  - Warning! Implicit data type conversion adds processing overhead and often is overlooked when troubleshooting SQL Server query performance
- **Tip: Be strict and consistent when specifying data types for new objects**
- **Tip: Use consistent naming conventions so you can more easily compare data types based on identically named attributes within a database**

Introduction to Common Table Expressions

- **Similar to a derived table but can allow for clear query construction versus a derived table approach**
  - Minimizes nesting of data sets
  - Allows you to isolate more complicated logic
- **You define one or more CTEs and then use them for the scope of a specific statement**
- **CTEs can also be recursive, meaning that it can reference itself in its own definition**

SQL Server: Transact-SQL Basic Data Retrieval

Module 5: Using Functions

Joe Sack

Joe@SQLskills.com

pluralsight
hardcore developer training

Module Introduction

- **This module covers a variety of functions which can be used within your data retrieval queries to meet application and reporting result set requirements**
- **You'll learn what the more commonly used functions are and *how* to use them, including coverage of:**
  - □ Aggregate functions
  - □ Mathematical functions
  - □ Ranking functions
  - □ Conversion functions
  - □ Date and Time functions
  - □ Logical functions
  - □ NULL handling
  - □ String functions
  - □ Analytic functions

Aggregate Functions

- **AVG**
  - Average of values in a group
- **CHECKSUM_AGG**
  - Checksum of values in a group
- **COUNT and COUNT_BIG**
  - Number of items in a group
    - COUNT returns int and COUNT_BIG returns bigint
  - If adding the DISTINCT keyword, COUNT and COUNT_BIG return the number of unique non-null values
  - COUNT(*) or COUNT_BIG(*) specifies all rows – including null values
- **MIN and MAX**
  - Minimum and maximum values in an expression
  - Both ignore null values

Aggregate Functions (2)

- **SUM**
  - Sum of all values
  - With the DISTINCT keyword – returns SUM of unique values
  - Null values are ignored
- **STDEV and STDEVP**
  - STDEV returns the standard deviation of all values in the expression
    - Assumes partial sampling of the whole population
  - STDEVP returns the standard deviation for the entire population of all values in the expression
- **VAR and VARP**
  - VAR is the statistical variance of all values in the specified expression
    - Assumes a partial sampling
  - VARP is the same as VAR, but assumes a entire population of all values

Mathematical Functions

- **More than 20 mathematical functions included natively in SQL Server 2005 – SQL Server 2012**
- **Common mathematical functions include:**
  - CEILING – smallest integer greater than or equal to the numeric expression
  - FLOOR – largest integer less than or equal to the numeric expression
  - PI – returns the constant value of PI
  - POWER – raises the numeric expression to a specified power
  - SQRT – square root of the numeric expression
  - ROUND – rounds number to the specified number of digits
  - RAND – generates a pseudo-random float value from 0 through 1
    - Has an optional “seed” integer value

OVER Clause

- **Applicable to ranking, aggregate and analytic functions**
- **OVER clause defines a “window” within a specific query result set**
  - Think of a window like a user-defined row set within the total result set
- **Allows you to apply aggregations to rows without a GROUP BY**
- **Window functions can then compute values for individual rows within the window**
  - Several examples of OVER in this module

Ranking Functions

- **ROW_NUMBER**
  - Sequential number of a row within a result set partition
- **RANK**
  - Returns rank of a row within the partition of the result set
    - Rank is calculated as one plus the number of ranks before the row
  - Tied rows based on logical order get the same rank
- **DENSE_RANK**
  - Same as RANK but with no gaps in ranking values
- **NTILE**
  - Map rows into equally sized row groups
    - You specify the number of groups
    - When the rows are not evenly divisible by groups, the group sizes will differ

Conversion Functions

- **PARSE (new in SQL Server 2012)**
  - Convert from a string data type to a date/time or number data type
- **TRY_PARSE (new in SQL Server 2012)**
  - Same as PARSE, but if the convert fails, returns NULL
- **TRY_CONVERT (new in SQL Server 2012)**
  - Converts from one data type to another and returns NULL if unsuccessful
- **CAST / CONVERT**
  - Both functions convert an expression from one data type to another
  - For CAST you designate the expression to be converted, data type and optional data type length
  - For CONVERT, you additionally can designate a style argument to determine output formats

Validating Data Types

- **ISDATE**
  - Validates if an input expression is a valid date or time
- **ISNUMERIC**
  - Validates if an input expression is a valid numeric data value

System Time Functions

- **SYSDATETIME**
  - Date and time of the SQL Server instance server
  - Returns datetime2(7) data type
- **SYSDATETIMEOFFSET**
  - Date and time of the SQL Server instance server
  - Includes time zone offset
  - Returns datetimeoffset(7) data type
- **SYSUTCDATETIME**
  - Date and time of the SQL Server instance server returned as Coordinated Universal Time (UTC)
  - Returns datetime2(7) data type
- **Lower precision functions are available – returning datetime**
  - CURRENT_TIMESTAMP, GETDATE, GETUTCDATE

Returning Date and Time Parts

- **DAY**
  - Returns integer representing day part of a provided date
- **MONTH**
  - Returns integer representing month part of a provided date
- **YEAR**
  - Returns integer representing year part of a provided date
- **DATEPART**
  - Returns integer value representing datepart of a date
  - Datepart examples include year (yy, yyyy), quarter (qq, q), month (mm, m), day (dd, d), hour (hh), minute (mi, n), second (ss, s), millisecond (ms)
- **DATENAME**
  - Returns string representing datepart of a date

Constructing Date and Time Values

- **DATEFROMPARTS**
  - Returns **date** data type value for a specified year, month and day
- **DATETIMEFROMPARTS**
  - Returns **datetime** value based on a specified year, month, day, hour, minute, seconds and milliseconds
- **DATETIME2FROMPARTS**
  - Returns **datetime2** value based on a specified year, month, day, hour, minute, seconds, fractions and precision

Constructing Date and Time Values (2)

- DATETIMEOFFSETFROMPARTS
  - Returns **datetimeoffset** value based on a specified year, month, day, hour, minute, seconds, fractions, hour_offset, minute_offset and precision
- SMALLDATETIMEFROMPARTS
  - Returns **smalldatetime** value based on a specified year, month, day, hour and minute
- TIMEFROMPARTS
  - Returns **time** value for specified hour, minute, seconds, fractions and precision

Calculating Time Differences

- **DATEDIFF returns the date/time difference between two input dates**
  - Return data type is integer
- **Takes a datepart argument followed by the start and end date**
  - Same datepart values accepted as would be provided for DATEPART

Modifying Dates

- **DATEADD**
  - Returns a new datetime value based on a datepart interval
- **EOMONTH (new in SQL Server 2012)**
  - Returns the last day of the month for a specified date
  - Optional integer that specifies months to add to the start date
- **SWITCHOFFSET**
  - Changes input datetimeoffset value to a new time zone offset
- **TODATETIMEOFFSET**
  - Converts a datetime2 value to a datetimeoffset value

Logical Functions

- **CHOOSE (new in SQL Server 2012)**
  - Choose an item from a list of values
  - First parameter is the 1-based index and the consecutive arguments are the list of values of any data type
- **IIF (new in SQL Server 2012)**
  - Return one of two values based on a Boolean expression
  - First parameter is the Boolean expression
  - Second parameter is the “true” value
  - Third parameter is the “false” value

Logical Functions (2)

- **Simple CASE expression**
    - Compare an expression to a set of expressions in order to determine the result
- **Searched CASE expression**
    - Evaluate a set of Boolean expressions in order to determine the result

Working with NULL

- **COALESCE**
  - Returns the first non-null expression from a list of arguments
- **ISNULL**
  - Replaces NULL with another value
  - Value is the same data type as the input expression
- **Understanding CONCAT_NULL_YIELDS_NULL**
  - When ON, concatenating a null value and a non-null string yields a NULL result

String Functions

- **ASCII**
  - Returns the ASCII integer code for the leftmost character of the expression
- **CHAR**
  - Converts an ASCII integer code to the char(1) data type equivalent
- **NCHAR**
  - Returns the Unicode character based on the input integer expression
- **UNICODE**
  - Returns the integer value representing the first character of the Unicode expression based on the Unicode standard

String Functions (2)

- **FORMAT (new in SQL Server 2012)**
  - Formats an input expression into an nvarchar value based on a format argument and optional culture argument
- **LEFT**
  - Outputs the left part of an input argument character string based on the positive integer specifying the number of characters to be returned
- **RIGHT**
  - Outputs the right part of an input argument character string based on the positive integer specifying the number of characters to be returned

String Functions (3)

- **LEN**
  - Outputs an int or bigint value representing the number of characters in the input string expression
  - Number of characters excludes trailing blanks
- **DATALENGTH**
  - Outputs number of bytes representing the number of characters in the input string expression
  - Number of characters excludes trailing blanks

String Functions (4)

- **LOWER**
  - Outputs character expression to lowercase
- **UPPER**
  - Outputs character expression to uppercase
- **LTRIM**
  - Removes leading blanks from the character expression
- **RTRIM**
  - Removes trailing blanks from the character expression

String Functions (5)

- **CHARINDEX and PATINDEX**
  - Both return the start position of a pattern within a character expression
  - PATINDEX allows for wildcard characters and CHARINDEX does not
- **REPLACE**
  - Replaces occurrences of a string pattern within a string expression with a replacement string
- **STUFF**
  - Inserts character data into a character expression
  - The point of insertion is determined by the start argument
  - The number of characters to delete is designated by the length argument
- **SUBSTRING**
  - Returns part of a character expression
  - Position argument determines the starting point and length argument determines the number of characters to return

String Functions (6)

- **REPLICATE**
  - Repeats a string expression a specified number of times
- **REVERSE**
  - Reverses the order of characters within a string expression
- **SPACE**
  - Returns a string of repeated spaces based on an integer expression
- **STR**
  - Converts numeric data into non-Unicode character data
- **CONCAT (new in SQL Server 2012)**
  - Returns a character string that is the result of two or more strings
- **QUOTENAME**
  - Delimits an input value, returning a Unicode string that produces a valid SQL Server identifier

Analytic Functions

- **LAG**
  - Compare values in current row with values in the previous row
- **LEAD**
  - Compares values in current row with values in a following row
- **FIRST_VALUE**
  - Return the first value in a result set
  - Result set may or may not be logically partitioned
- **LAST_VALUE**
  - Returns the last value in a result set
  - Result set may or may not be logically partitioned

Analytic Functions (2)

- **CUME_DIST**
  - Returns a value greater than 0 and less than 1 representing the number of rows less than or equal to the current row value divided by the number of rows in the partition/result set
- **PERCENT_RANK**
  - Returns a value greater than 0 and less than 1 representing the relative rank of a row within the partition/result set
- **PERCENTILE_CONT**
  - Calculates a percentile based on a continuous distribution of values
  - Result is interpolated, so the returned value may not actually be in the result set
- **PERCENTILE_DISC**
  - Similar to PERCENTILE_CONT, but the result will choose a number from the result set
    - Smallest CUME_DIST value greater than or equal to the percentile value

Course Summary

- **We've covered:**
  - Setting up your own Transact-SQL learning environment
  - Writing a basic SELECT statement
  - Writing a query that accesses multiple data sources
  - Using functions to meet application and business requirements
- **To solidify what you've learned, keep *applying* your knowledge in real life scenarios**
- **Thanks for watching!**