SQL Server: Transact-SQL Basic Data Retrieval

Module 4: Querying Multiple Data Sources

Joe Sack

Joe@SQLskills.com

pluralsight
hardcore developer training

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**