**Join Fundamentals**

By using joins, you can retrieve data from two or more tables based on logical relationships between the tables. Joins indicate how Microsoft SQL Server should use data from one table to select the rows in another table.

A join condition defines the way two tables are related in a query by:

* Specifying the column from each table to be used for the join. A typical join condition specifies a foreign key from one table and its associated key in the other table.
* Specifying a logical operator (for example, = or <>,) to be used in comparing values from the columns.

Inner joins can be specified in either the FROM or WHERE clauses. Outer joins can be specified in the FROM clause only. The join conditions combine with the WHERE and HAVING search conditions to control the rows that are selected from the base tables referenced in the FROM clause.

Specifying the join conditions in the FROM clause helps separate them from any other search conditions that may be specified in a WHERE clause, and is the recommended method for specifying joins. A simplified ISO FROM clause join syntax is:

FROM *first\_table* *join\_type* *second\_table* [ON (*join\_condition*)]

*join\_type* specifies what kind of join is performed: an inner, outer, or cross join. *join\_condition* defines the predicate to be evaluated for each pair of joined rows. The following is an example of a FROM clause join specification:

FROM Purchasing.ProductVendor JOIN Purchasing.Vendor

ON (ProductVendor.VendorID = Vendor.VendorID)

The following is a simple SELECT statement using this join:

SELECT ProductID, Purchasing.Vendor.VendorID, Name

FROM Purchasing.ProductVendor JOIN Purchasing.Vendor

ON (Purchasing.ProductVendor.VendorID = Purchasing.Vendor.VendorID)

WHERE StandardPrice > $10

AND Name LIKE N'F%'

GO

The select returns the product and supplier information for any combination of parts supplied by a company for which the company name starts with the letter F and the price of the product is more than $10.

When multiple tables are referenced in a single query, all column references must be unambiguous. In the previous example, both the ProductVendor and Vendor table have a column named VendorID. Any column name that is duplicated between two or more tables referenced in the query must be qualified with the table name. All references to the Vendor columns in the example are qualified.

When a column name is not duplicated in two or more tables used in the query, references to it do not have to be qualified with the table name. This is shown in the previous example. Such a SELECT statement is sometimes difficult to understand because there is nothing to indicate the table that provided each column. The readability of the query is improved if all columns are qualified with their table names. The readability is further improved if table aliases are used, especially when the table names themselves must be qualified with the database and owner names. The following is the same example, except that table aliases have been assigned and the columns qualified with table aliases to improve readability:

SELECT pv.ProductID, v.VendorID, v.Name

FROM Purchasing.ProductVendor pv JOIN Purchasing.Vendor v

ON (pv.VendorID = v.VendorID)

WHERE StandardPrice > $10

AND Name LIKE N'F%'

The previous examples specified the join conditions in the FROM clause, which is the preferred method. The following query contains the same join condition specified in the WHERE clause:

SELECT pv.ProductID, v.VendorID, v.Name

FROM Purchasing.ProductVendor pv, Purchasing.Vendor v

WHERE pv.VendorID = v.VendorID

AND StandardPrice > $10

AND Name LIKE N'F%'

The select list for a join can reference all the columns in the joined tables, or any subset of the columns. The select list is not required to contain columns from every table in the join. For example, in a three-table join, only one table can be used to bridge from one of the other tables to the third table, and none of the columns from the middle table have to be referenced in the select list.

Although join conditions usually have equality comparisons (=), other comparison or relational operators can be specified, as can other predicates. For more information, see [Using Operators in Expressions](http://msdn.microsoft.com/en-us/library/ms189123.aspx) and [WHERE (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms188047.aspx).

When SQL Server processes joins, the query engine chooses the most efficient method (out of several possibilities) of processing the join. The physical execution of various joins can use many different optimizations and therefore cannot be reliably predicted.

Columns used in a join condition are not required to have the same name or be the same data type. However, if the data types are not identical, they must be compatible, or be types that SQL Server can implicitly convert. If the data types cannot be implicitly converted, the join condition must explicitly convert the data type using the CAST function. For more information about implicit and explicit conversions, see [Data Type Conversion (Database Engine)](http://msdn.microsoft.com/en-us/library/ms191530.aspx).

Most queries using a join can be rewritten using a subquery (a query nested within another query), and most subqueries can be rewritten as joins. For more information about subqueries, see [Subquery Fundamentals](http://msdn.microsoft.com/en-us/library/ms189575.aspx).

|  |
| --- |
| **ms191517.note(en-us,SQL.100).gifNote:** |
| Tables cannot be joined directly on **ntext**, **text**, or **image** columns. However, tables can be joined indirectly on **ntext**, **text**, or **image** columns by using SUBSTRING. For example, SELECT \* FROM t1 JOIN t2 ON SUBSTRING(t1.*textcolumn*, 1, 20) = SUBSTRING(t2.*textcolumn*, 1, 20) performs a two-table inner join on the first 20 characters of each text column in tables **t1** and **t2**. In addition, another possibility for comparing **ntext** or **text** columns from two tables is to compare the lengths of the columns with a WHERE clause, for example: WHERE DATALENGTH(p1.pr\_info) = DATALENGTH(p2.pr\_info) |

**Using Joins**

Join conditions can be specified in either the FROM or WHERE clauses; specifying them in the FROM clause is recommended. WHERE and HAVING clauses can also contain search conditions to further filter the rows selected by the join conditions.

Joins can be categorized as:

* Inner joins (the typical join operation, which uses some comparison operator like = or <>). These include equi-joins and natural joins.   
  Inner joins use a comparison operator to match rows from two tables based on the values in common columns from each table. For example, retrieving all rows where the student identification number is the same in both the **students** and **courses** tables.
* Outer joins. Outer joins can be a left, a right, or full outer join.   
  Outer joins are specified with one of the following sets of keywords when they are specified in the FROM clause:
  + LEFT JOIN or LEFT OUTER JOIN   
    The result set of a left outer join includes all the rows from the left table specified in the LEFT OUTER clause, not just the ones in which the joined columns match. When a row in the left table has no matching rows in the right table, the associated result set row contains null values for all select list columns coming from the right table.
  + RIGHT JOIN or RIGHT OUTER JOIN   
    A right outer join is the reverse of a left outer join. All rows from the right table are returned. Null values are returned for the left table any time a right table row has no matching row in the left table.
  + FULL JOIN or FULL OUTER JOIN   
    A full outer join returns all rows in both the left and right tables. Any time a row has no match in the other table, the select list columns from the other table contain null values. When there is a match between the tables, the entire result set row contains data values from the base tables.
* Cross joins   
  Cross joins return all rows from the left table. Each row from the left table is combined with all rows from the right table. Cross joins are also called Cartesian products.

For example, here is an inner join retrieving the employees who are also sales persons:

USE AdventureWorks;

GO

SELECT e.EmployeeID

FROM HumanResources.Employee AS e

INNER JOIN Sales.SalesPerson AS s

ON e.EmployeeID = s.SalesPersonID

The tables or views in the FROM clause can be specified in any order with an inner join or full outer join. However, the order of tables or views specified when using either a left or right outer join is important.