**1. Nested Loops Join**

The SQL Server optimizer might choose a Nested Loops join when one of the joining tables is small (considered as the outer table) and another one is large (considered as the inner table which is indexed on the column that is in the join) and hence it requires minimal I/O and the fewest comparisons.

* **naive nested loops join** in which case the search scans the whole table or index
* **index nested loops join** when the search can utilize an existing index to perform lookups
* **Temporary index nested loops join** if the optimizer creates a temporary index as part of the query plan and destroys it after query execution completes.

An index nested loops join performs better than a merge join or hash join if a small set of rows are involved. Whereas, if a large set of rows are involved the Nested Loops join might not be an optimal choice. Nested Loops support almost all join types except right and full outer joins, right semi-join and right anti-semi join.

**2. Merge Join**

The first thing that you need to know about a Merge join is that it requires both inputs to be sorted on join keys/merge columns (or both input tables have clustered indexes on the column that joins the tables) and it also requires at least one equijoin (equals to) expression/predicate.

A Merge join is often a more efficient and faster join operator if the sorted data can be obtained from an existing B-tree index and it performs almost all join operations as long as there is at least one equality join predicate involved. It also supports multiple equality join predicates as long as the input tables are sorted on all joining keys involved and are in the same order.

The presence of a Compute Scalar operator indicates the evaluation of an expression to produce a computed scalar value. In the above query I am selecting LineTotal which is a derived column, hence it has been used in the execution plan.

Nested Loop一般在两个集合都很大的情况下效率就相当差了，而Sort-Merge在这种情况下就比它要高效不少，尤其是当两个集合的JOIN字段上都有聚集索引(clustered index)存在时，Sort-Merge性能将达到最好。

**3. Hash Join**

During the build phase, joining keys of all the rows of the build table are scanned. Hashes are generated and placed in an in-memory hash table. Unlike the Merge join, it's blocking (no rows are returned) until this point.  
  
During the probe phase, joining keys of each row of the probe table are scanned. Again hashes are generated (using the same hash function as above) and compared against the corresponding hash table for a match.

A Hash function requires significant amount of CPU cycles to generate hashes and memory resources to store the hash table. If there is memory pressure, some of the partitions of the hash table are swapped to tempdb and whenever there is a need (either to probe or to update the contents) it is brought back into the cache. To achieve high performance, the query optimizer may parallelize a Hash join to scale better than any other join, for more details [click here.](http://blogs.msdn.com/b/craigfr/archive/2006/11/16/parallel-hash-join.aspx)

There are basically three different types of hash joins,

* **In-memory Hash Join** in which case enough memory is available to store the hash table
* **Grace Hash Join** in which case the hash table cannot fit in memory and some partitions are spilled to tempdb
* **Recursive Hash Join** in which case a hash table is so large the optimizer has to use many levels of merge joins. For more details about these different types [click here.](http://technet.microsoft.com/en-us/library/ms189313.aspx)