Avoiding Optimizer Hints

**SQL Server’s cost-based optimizer dynamically determines the processing strategy for a query based on the current table/index structure and data. This dynamic behavior can be overridden using optimizer hints, taking some of the decisions away from the optimizer by instructing it to use a certain processing strategy. This makes the optimizer behavior static and doesn’t allow it to dynamically update the processing strategy as the table/index structure or data changes.**

**Since it is usually difficult to outsmart the optimizer, the usual recommendation is to avoid optimizer hints. Generally, it is beneficial to let the optimizer determine a cost-effective processing strategy based on the data distribution statistics, indexes, and other factors. Forcing the optimizer (with hints) to use a specific processing strategy hurts performance more often than not, as shown in the following examples for these hints:**

**JOIN hint**

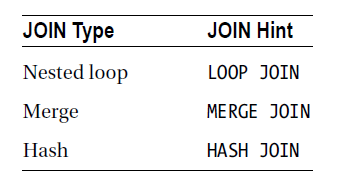
**INDEX hint**

**FORCEPLAN hint**

**Join Hints**

**The optimizer dynamically determines a cost-effective JOIN strategy between two data**

**sets based on the table/index structure and data. Table below presents a summary of the JOIN types supported by SQL Server.**

****

You can instruct SQL Server to use a specific JOIN type by using the JOIN hints in Table above. To understand how the use of JOIN hints can affect performance, consider the following SELECT statement.

**---query without a join hint**

SELECT s.[Name] AS StoreName,

p.[LastName] + ', ' + p.[FirstName]

FROM [Sales].[Store] s

JOIN [Sales].SalesPerson AS sp

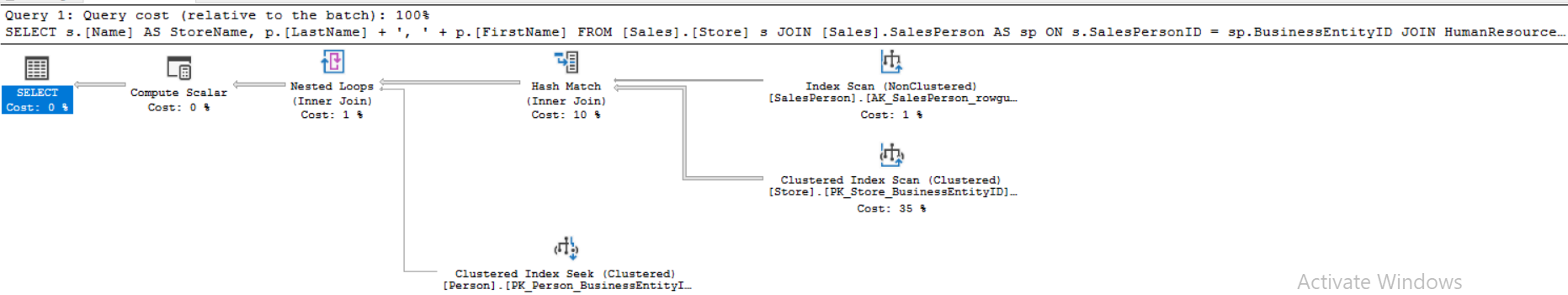
ON s.SalesPersonID = sp.BusinessEntityID

JOIN HumanResources.Employee AS e

ON sp.BusinessEntityID = e.BusinessEntityID

JOIN Person.Person AS p

ON e.BusinessEntityID = p.BusinessEntityID ;

****

**As you can see, SQL Server dynamically decided to use a LOOP JOIN to add the data from the Person.Person table and to add a HASH JOIN for the Sales.Salesperson and Sales.Store tables. For simple queries affecting a small result set, a LOOP JOIN generally provides better performance than a HASH JOIN or MERGE JOIN. Since the number of rows coming from the Sales.Salesperson table is relatively small.**

**You can force the JOIN to be a LOOP like this:**

**---query with a join hint**

SELECT s.[Name] AS StoreName,

p.[LastName] + ', ' + p.[FirstName]

FROM [Sales].[Store] s

JOIN [Sales].SalesPerson AS sp

ON s.SalesPersonID = sp.BusinessEntityID

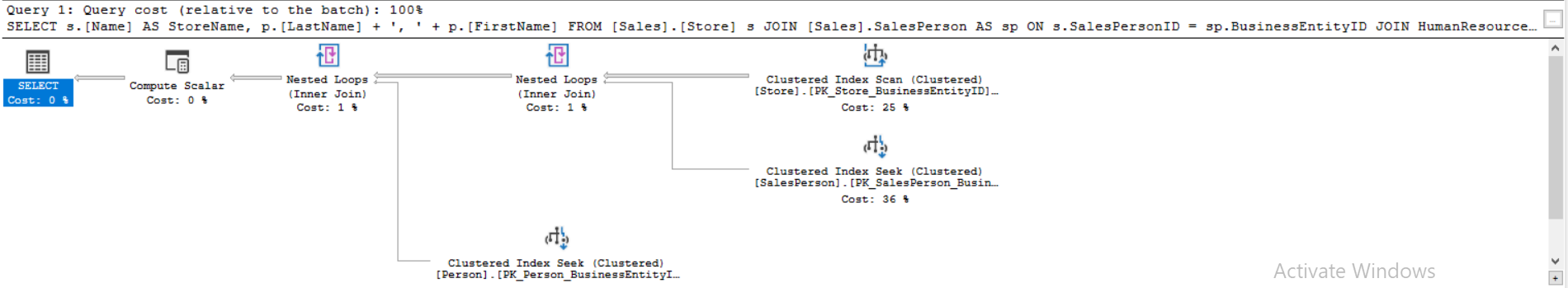
JOIN HumanResources.Employee AS e

ON sp.BusinessEntityID = e.BusinessEntityID

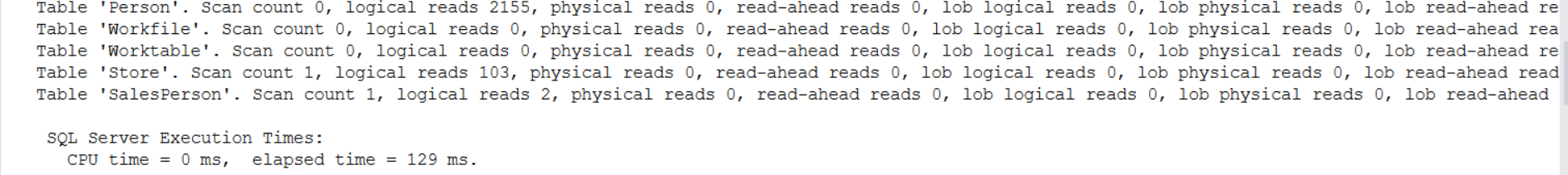
JOIN Person.Person AS p

ON e.BusinessEntityID = p.BusinessEntityID

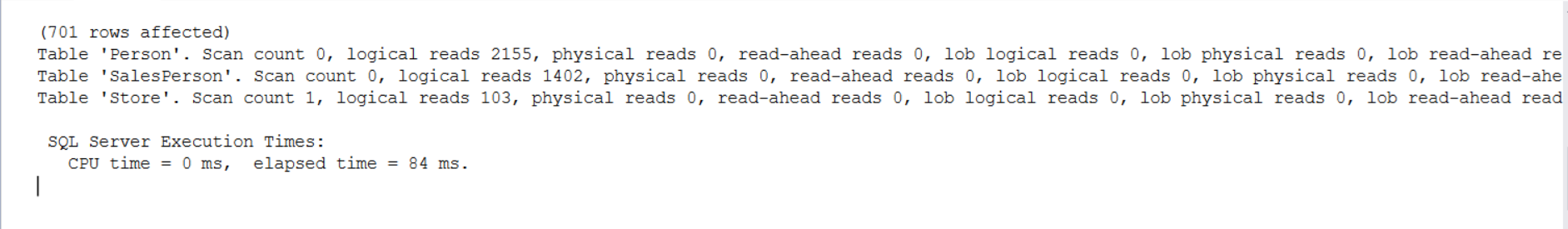
OPTION (LOOP JOIN) ;



**--- With no JOIN hint:**



**--- With JOIN hint:**



**Bad hint practice for join**

set statistics io on

set statistics time on

SELECT s.[Name] AS StoreName,

p.[LastName] + ', ' + p.[FirstName]

FROM [Sales].[Store] s

INNER LOOP JOIN [Sales].SalesPerson AS sp

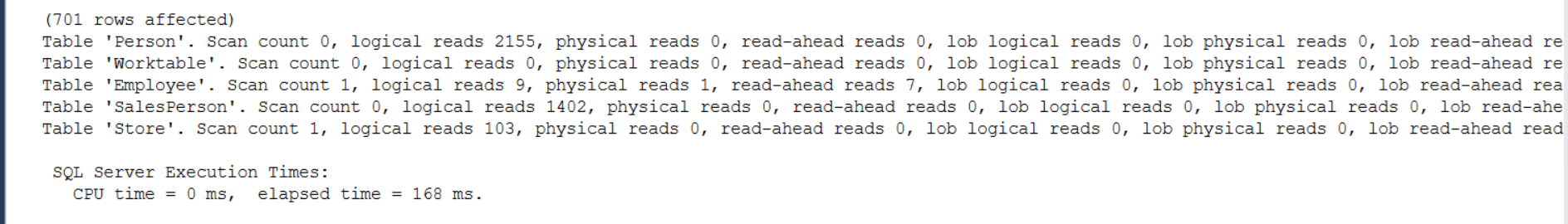
ON s.SalesPersonID = sp.BusinessEntityID

JOIN HumanResources.Employee AS e

ON sp.BusinessEntityID = e.BusinessEntityID

JOIN Person.Person AS p

ON e.BusinessEntityID = p.BusinessEntityID ;



As you can see, there are now four tables referenced in the query plan. There have been four tables referenced through all the previous executions, but the optimizer will eliminate one table from the query through the simplification process of optimization when no hint will be specified as in the below query .The hint has forced the optimizer to make different choices than it otherwise might have and removed simplification from the process.

The reads and execution time suffered as well.

**Above Query without join hint**

set statistics io on

set statistics time on

SELECT s.[Name] AS StoreName,

p.[LastName] + ', ' + p.[FirstName]

FROM [Sales].[Store] s

INNER JOIN [Sales].SalesPerson AS sp

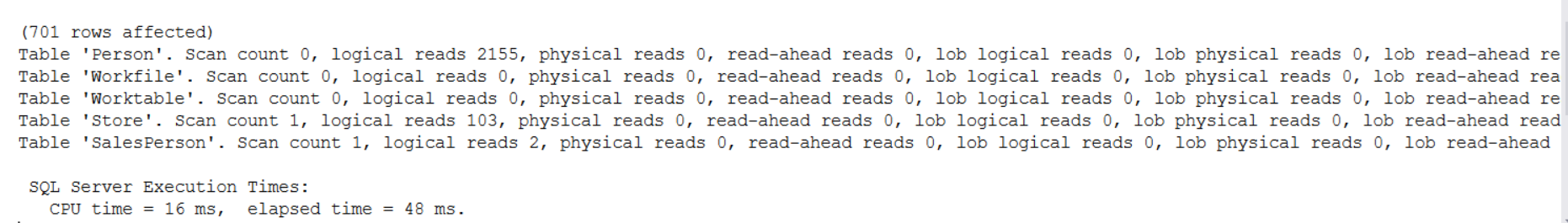
ON s.SalesPersonID = sp.BusinessEntityID

JOIN HumanResources.Employee AS e

ON sp.BusinessEntityID = e.BusinessEntityID

JOIN Person.Person AS p

ON e.BusinessEntityID = p.BusinessEntityID ;



**JOIN hints force the optimizer to ignore its own optimization strategy and use instead the strategy specified by the query. JOIN hints generally hurt query performance because of the following factors:**

**Hints prevent autoparameterization.**

**The optimizer is prevented from dynamically deciding the joining order of the tables.**

**Therefore, it makes sense to not use the JOIN hint but to instead let the optimizer dynamically determine a cost-effective processing strategy.**

**INDEX Hints**

**As mentioned earlier, using an arithmetic operator on a WHERE clause column prevents the optimizer from choosing the index on the column. To improve performance, you can rewrite the query without using the arithmetic operator on the WHERE clause, as shown in the corresponding example. Alternatively,**

**You can force the optimizer to use the index on the column with an INDEX hint (a type of**

**optimizer hint). However, most of the time, it is better to avoid the INDEX hint and let the optimizer be have dynamically.**

**To understand the effect of an INDEX hint on query performance, consider the example:**

SELECT \*

FROM Purchasing.PurchaseOrderHeader AS poh ---WITH (INDEX (PK\_PurchaseOrderHeader\_PurchaseOrderID))

WHERE poh.PurchaseOrderID \* 2 = 3400 ;

**The multiplication operator on the PurchaseOrderID column prevented the optimizer from choosing the index on the column. You can use an INDEX hint to force the optimizer to use the index on the OrderID column as follows:**

SELECT \*

FROM Purchasing.PurchaseOrderHeader AS poh WITH (INDEX (PK\_PurchaseOrderHeader\_PurchaseOrderID))

WHERE poh.PurchaseOrderID \* 2 = 3400 ;

---query without Index hint and use of arithmetic operator on the right side of the comparison operator not left side on the column

SELECT \*

FROM Purchasing.PurchaseOrderHeader AS poh ---WITH (INDEX (PK\_PurchaseOrderHeader\_PurchaseOrderID))

WHERE poh.PurchaseOrderID = 3400/2 ;

Here is the comparison of all three variations of queries:

**No hint (with the arithmetic operator on the WHERE clause column):**

Table 'PurchaseOrderHeader'. Scan count 1, logical reads 11

CPU time = 0 ms, elapsed time = 153 ms.

**No hint (without the arithmetic operator on the WHERE clause column):**

Table 'PurchaseOrderHeader'. Scan count 0, logical reads 2

CPU time = 16 ms, elapsed time = 76 ms.

**INDEX hint:**

Table 'PurchaseOrderHeader'. Scan count 1, logical reads 44

CPU time = 16 ms, elapsed time = 188 ms.