Report 3

Databases optimization

Attention! Whole project is available at the repository.

Introduction

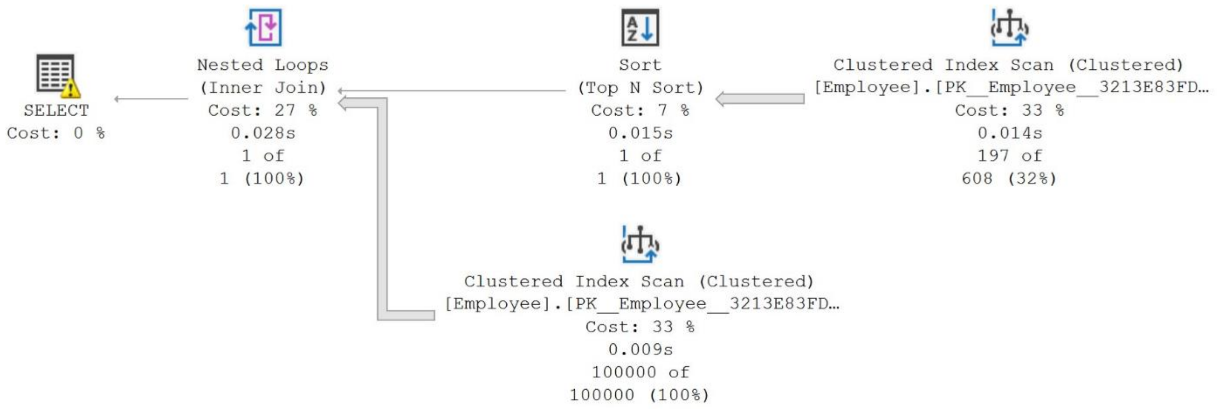
For define the optimization level of the query, I will be using explain plan provided by Microsoft in SQL Server Management Studio. The execution plan shows us which tables were accessed, how they were accessed, how they were joined together, and any other operations that occurred along the way. Included are query costs, which are estimates of the overall expense of any query component. A treasure trove of data is also included, such as row size, CPU cost, I/O cost, and details on which indexes were utilized.

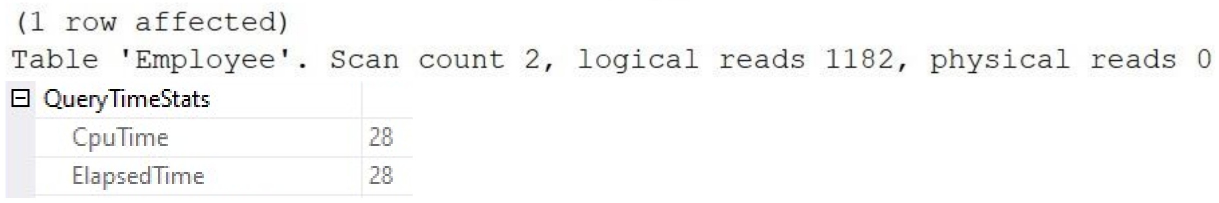
IO statistics will be the main criteria of how the query was optimized. Logical reads tell us how many reads were made from the buffer cache. This is the number that we will refer to whenever we talk about how many reads a query is responsible for, or how much IO it is causing. Physical reads tell us how much data was read from a storage device as it was not yet present in memory. This can be a useful indication of buffer cache/memory capacity problems if data is very frequently being read from storage devices, rather than memory. In general, IO will be the primary cause of latency and bottlenecks when analysing slow queries. The unit of measurement of STATISTICS IO = 1 read = a single 8kb page = 8192 bytes.

1. First

a. Bad

```sql
SELECT Employee.first_name, Employee.last_name FROM Employee WHERE Employee.salary =
(SELECT TOP(1) Employee.salary FROM Employee WHERE YEAR(Employee.birth_date) = 2020
AND MONTH(Employee.birth_date) = 9 ORDER BY Employee.salary DESC)
```



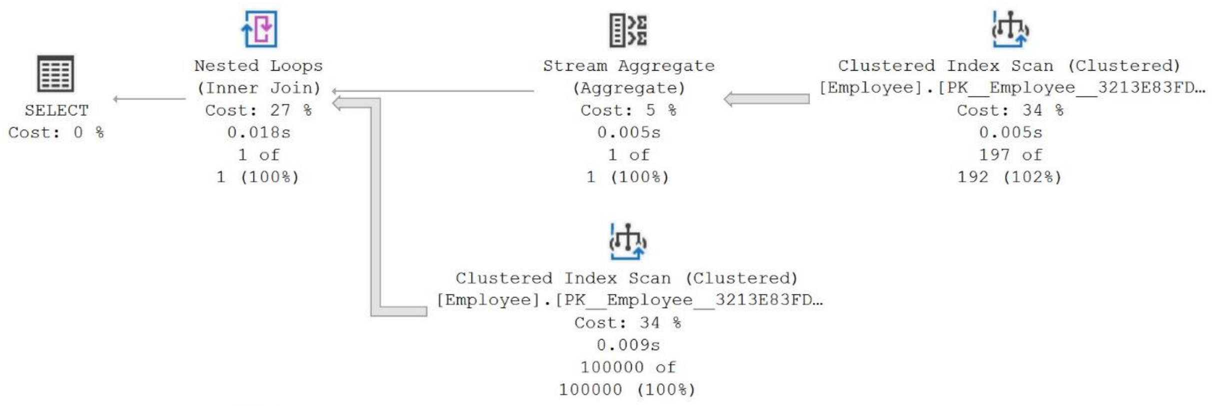


```
(1 row affected)
Table 'Employee'. Scan count 2, logical reads 1182, physical reads 0
```

QueryTimeStats	
CpuTime	28
ElapsedTime	28

b. Better

```sql
SELECT Employee.first_name, Employee.last_name FROM Employee WHERE Employee.salary =
(SELECT MAX(Employee.salary) FROM Employee WHERE Employee.birth_date BETWEEN
'9/1/2020' AND '9/30/2020')
```



```
(1 row affected)
Table 'Employee'. Scan count 2, logical reads 1182, physical reads 0
```

QueryTimeStats	
CpuTime	19
ElapsedTime	19

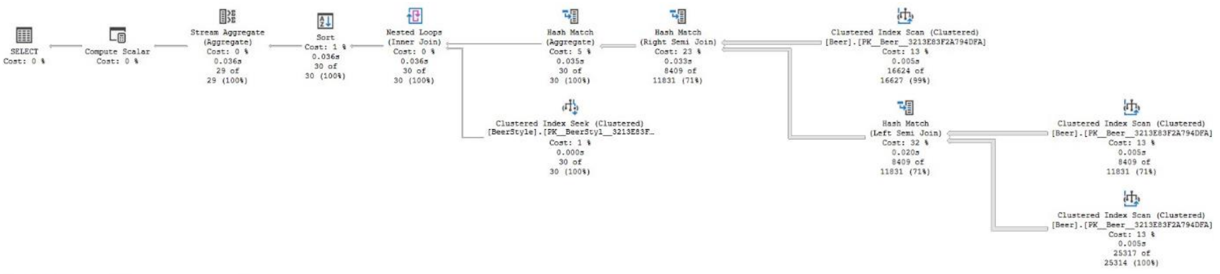
c. Description

These two queries may seem to be the same. They have to very similar execution plan and exactly the same IO statistics. The main goal of this optimization was to decrease execution time. Using aggregation instead of sorting to select maximal value is faster, same for using BETWEEN keyword instead of specifying the individual value of year and month

2. Second

a. Bad

```
SELECT BeerStyle.name, AVG(Beer.price) FROM BeerStyle, Beer WHERE BeerStyle.id =
Beer.beer_style_id AND Beer.price IN (SELECT Beer.price FROM Beer WHERE BEER.price >
500) AND Beer.amount_in_liters IN (SELECT Beer.amount_in_liters FROM Beer WHERE
amount_in_liters > 1000) GROUP BY BeerStyle.name
```



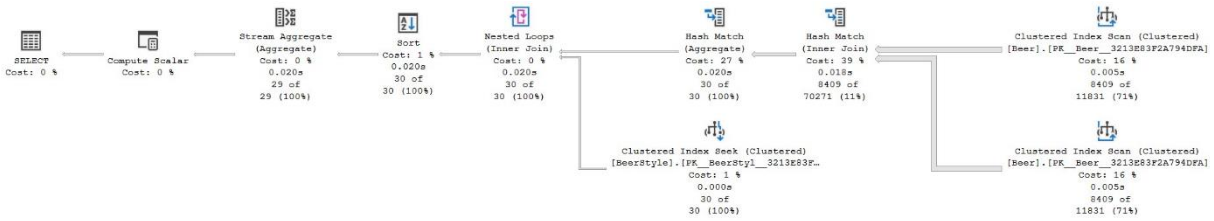
```
(29 rows affected)
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0,
Table 'BeerStyle'. Scan count 0, logical reads 60, physical reads 0,
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, p
Table 'Beer'. Scan count 3, logical reads 600, physical reads 0, pag
```

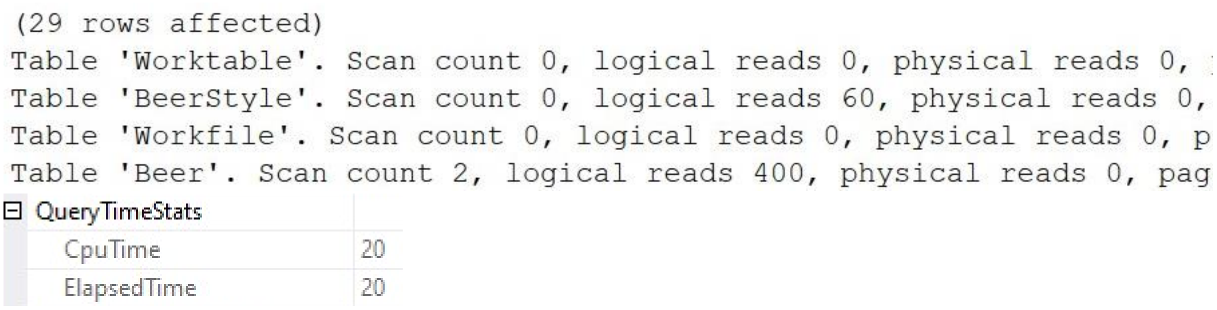
QueryTimeStats	
CpuTime	36
ElapsedTime	36

b. Better

```
SELECT BeerStyle.name, AVG(Beer.price) FROM BeerStyle INNER JOIN Beer ON (BeerStyle.id
= Beer.beer_style_id), (SELECT Beer.price, Beer.amount_in_liters FROM Beer WHERE
BEER.price > 500 AND amount_in_liters > 1000) AS Selection WHERE Beer.price IN
```

```
(Selection.price) AND Beer.amount_in_liters IN (Selection.amount_in_liters) GROUP BY
BeerStyle.name
```





```
 (29 rows affected)
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0,
Table 'BeerStyle'. Scan count 0, logical reads 60, physical reads 0,
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, p
Table 'Beer'. Scan count 2, logical reads 400, physical reads 0, pag
```

⊟ QueryTimeStats	
CpuTime	20
ElapsedTime	20

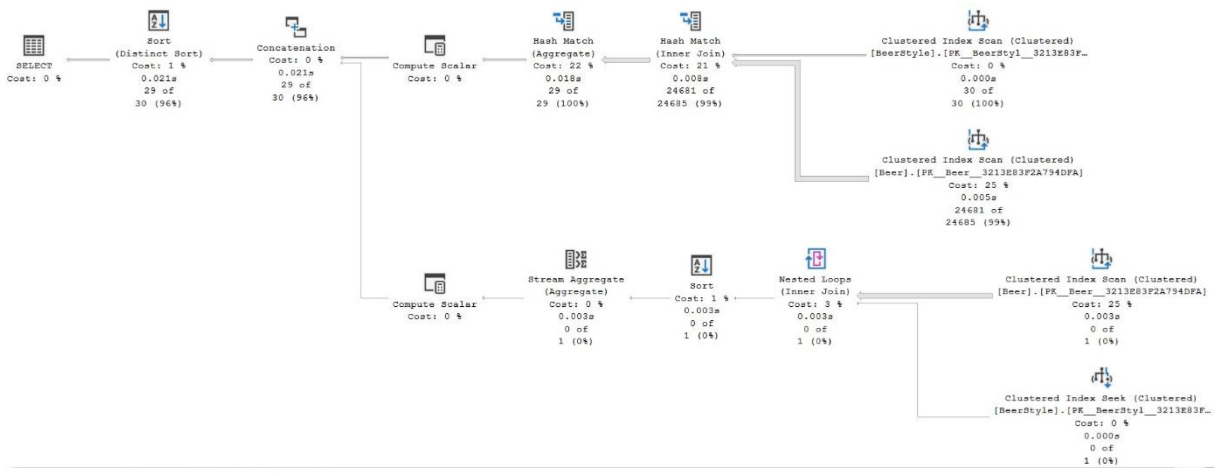
c. Description

Sometimes there may have been more than one subqueries in your main query. Better is minimizing the number of subquery block in your query. Using the WHERE clause creates a Cartesian Join, also called a Cartesian Product or CROSS JOIN. In a Cartesian Join, all possible combinations of the variables are created. In this example, if we had 1,000 customers with 1,000 total sales, the query would first generate 1,000,000 results, then filter for the 1,000 records where CustomerID is correctly joined. This is an inefficient use of database resources, as the database has done 100x more work than required. Cartesian Joins are especially problematic in large-scale databases because a Cartesian Join of two large tables could create billions or trillions of results. To prevent creating a Cartesian Join, it is better to use an INNER JOIN.

3. Third

a. Bad

```
SELECT BeerStyle.name, COUNT(*) FROM BeerStyle INNER JOIN Beer ON (BeerStyle.id =
Beer.beer_style_id) WHERE Beer.price < 500 GROUP BY BeerStyle.name UNION SELECT
BeerStyle.name, COUNT(*) FROM BeerStyle INNER JOIN Beer ON (BeerStyle.id =
Beer.beer_style_id) WHERE Beer.price > 1000 GROUP BY BeerStyle.name
```

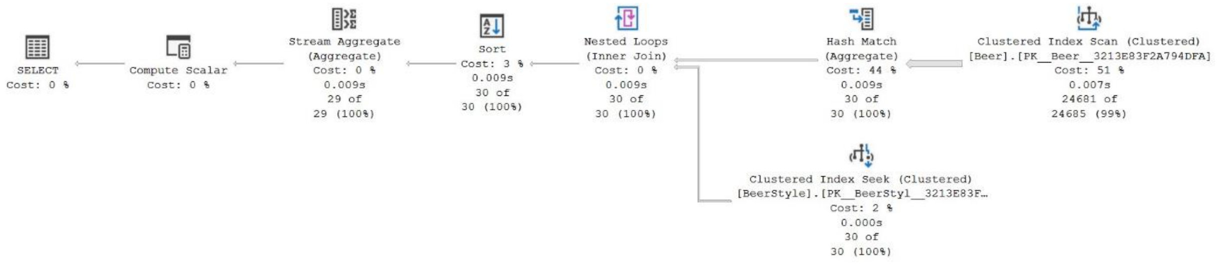


```
(29 rows affected)
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0,
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0,
Table 'Beer'. Scan count 2, logical reads 400, physical reads 0, pa
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, 
Table 'BeerStyle'. Scan count 1, logical reads 2, physical reads 0,
```

⊟ QueryTimeStats	
CpuTime	22
ElapsedTime	22

b. Better

```
SELECT BeerStyle.name, COUNT(*) FROM BeerStyle INNER JOIN Beer ON (BeerStyle.id =
Beer.beer_style_id) WHERE Beer.price < 500 OR Beer.price > 1000 GROUP BY
BeerStyle.name
```



```
(29 rows affected)
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0,
Table 'BeerStyle'. Scan count 0, logical reads 60, physical reads 1,
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, p
Table 'Beer'. Scan count 1, logical reads 200, physical reads 0, pag
```

⊟ QueryTimeStats	
CpuTime	9
ElapsedTime	10

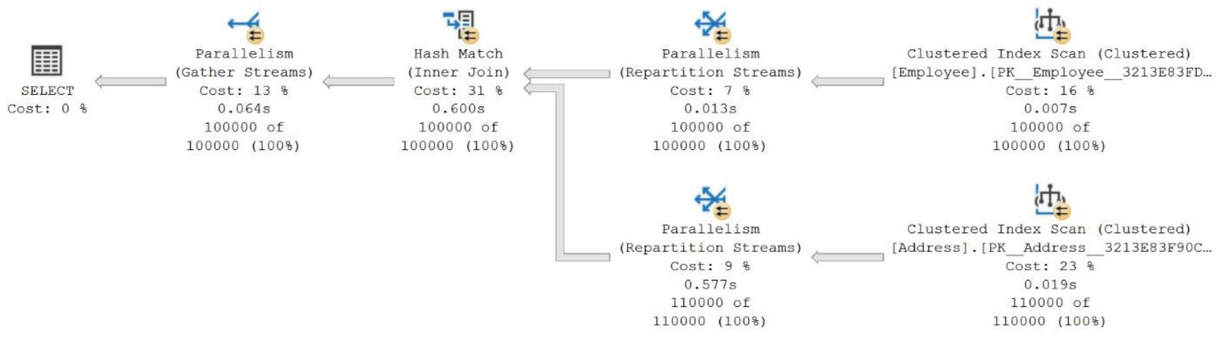
c. Description

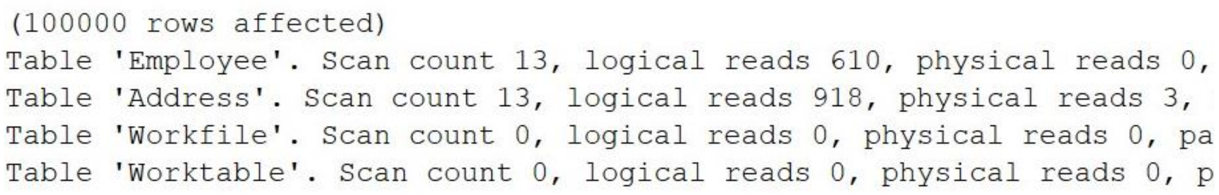
UNION clause makes the server to execute two queries instead of 1 and then merge them. There is often a way to simplify the whole process by using one compressed query.

4. Fourth

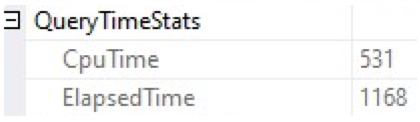
a. Bad

```
SELECT DISTINCT * FROM Employee, Address WHERE (Employee.address_id = Address.id)
```





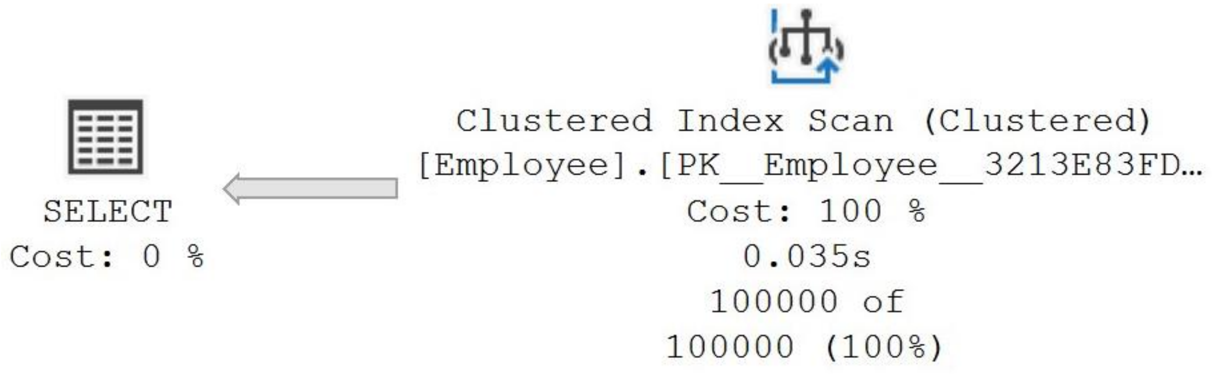
```
(100000 rows affected)
Table 'Employee'. Scan count 13, logical reads 610, physical reads 0,
Table 'Address'. Scan count 13, logical reads 918, physical reads 3,
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, pa
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, p
```

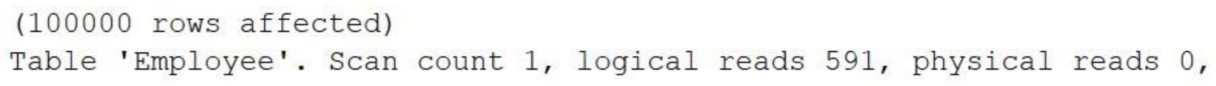


⊟ QueryTimeStats	
CpuTime	531
ElapsedTime	1168

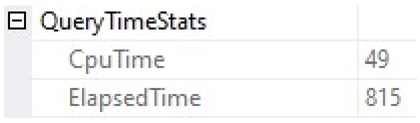
b. Better

```
SELECT * FROM Employee WHERE EXISTS (SELECT 'Y' FROM Address WHERE Employee.address_id
= Address.id)
```


SELECT
Cost: 0 %
Clustered Index Scan (Clustered)
[Employee].[PK__Employee__3213E83FD…
Cost: 100 %
0.035s
100000 of
100000 (100%)



```
(100000 rows affected)
Table 'Employee'. Scan count 1, logical reads 591, physical reads 0,
```



⊟ QueryTimeStats	
CpuTime	49
ElapsedTime	815

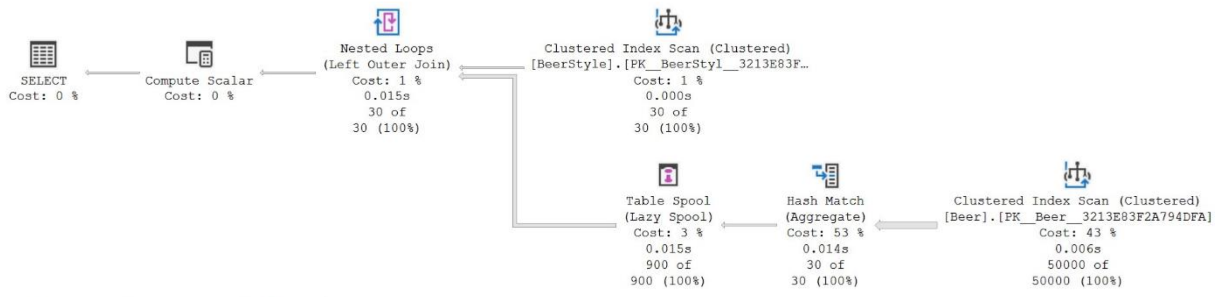
c. Description

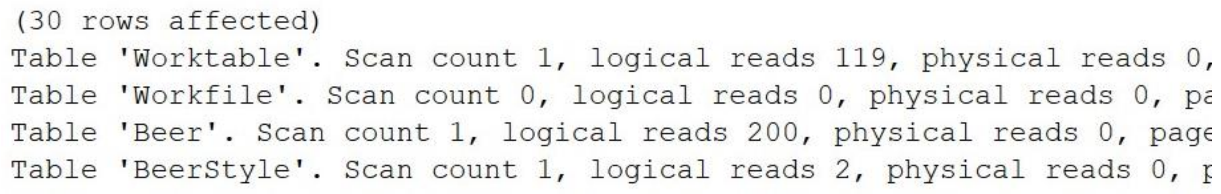
When we use the Distinct keyword it looks as if the query will be automatically optimized by the internal SQL engine, that is true when we use the Distinct for filtering operation when we use the distinct during the table join operations which has one too many relations then it's not advisable. When we use the "Exists" query for fetching some operations during table join then it's better comparing it to the Distinct compares.

5. Fifth

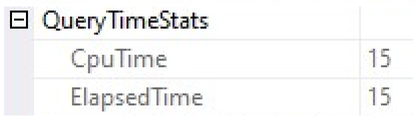
a. Bad

```
SELECT BeerStyle.name, (SELECT MAX(Beer.price) FROM Beer WHERE Beer.beer_style_id =
BeerStyle.id) FROM BeerStyle
```





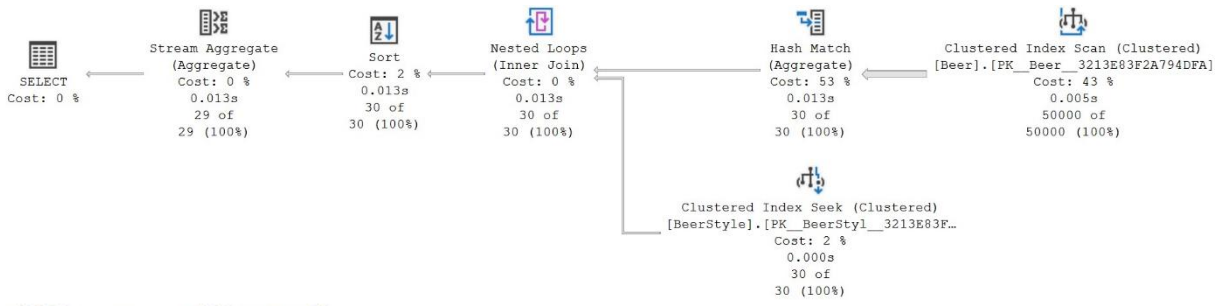
```
(30 rows affected)
Table 'Worktable'. Scan count 1, logical reads 119, physical reads 0,
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, pa
Table 'Beer'. Scan count 1, logical reads 200, physical reads 0, page
Table 'BeerStyle'. Scan count 1, logical reads 2, physical reads 0, p
```

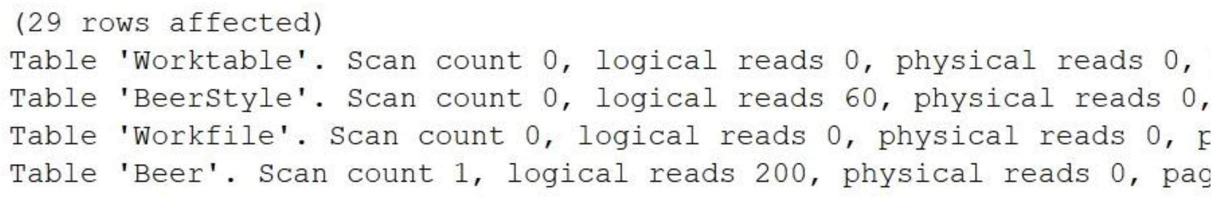


⊟ QueryTimeStats	
CpuTime	15
ElapsedTime	15

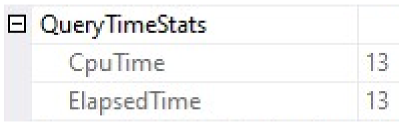
b. Better

```
SELECT BeerStyle.name, MAX(Beer.price) FROM BeerStyle INNER JOIN Beer ON (BeerStyle.id
= Beer.beer_style_id) GROUP BY BeerStyle.name
```





```
(29 rows affected)
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0,
Table 'BeerStyle'. Scan count 0, logical reads 60, physical reads 0,
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, p
Table 'Beer'. Scan count 1, logical reads 200, physical reads 0, pag
```



⊟ QueryTimeStats	
CpuTime	13
ElapsedTime	13

c. Description

Although subqueries are useful, they often can be replaced by a join, which is definitely faster to execute.