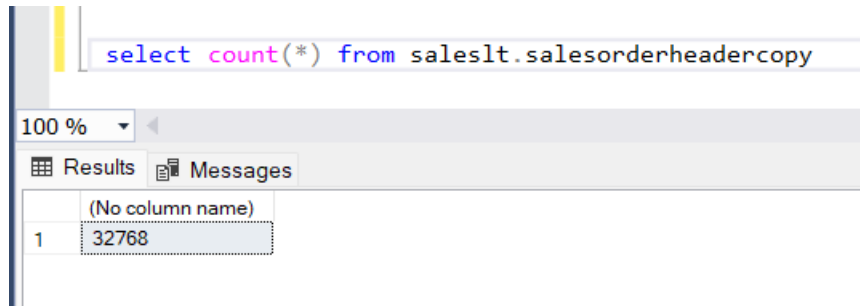


## use of hints for query performance

So, what I am going to do is, we currently have this table, which has now expanded to 32768.



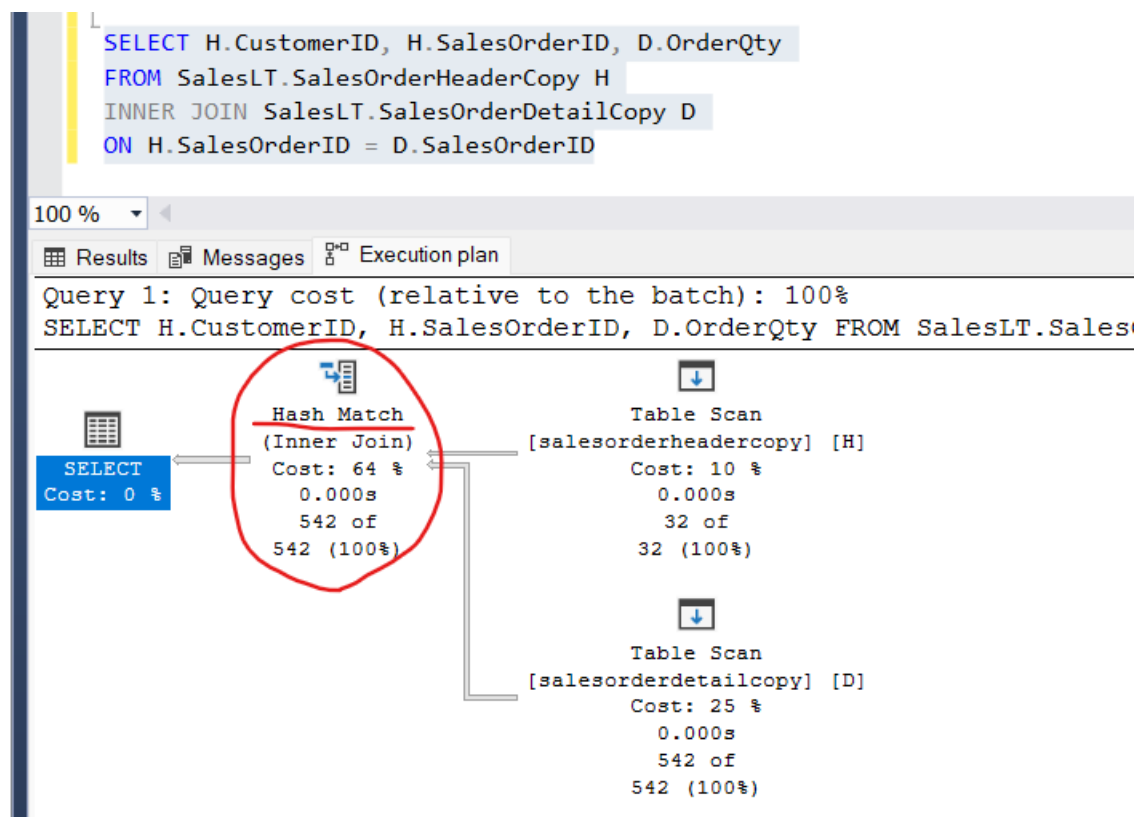
The screenshot shows a SQL query window with the following text: `select count(*) from saleslt.salesorderheadercopy`. Below the query, the 'Results' tab is selected, showing a single row with the value 32768.

| (No column name) |
|------------------|
| 32768            |

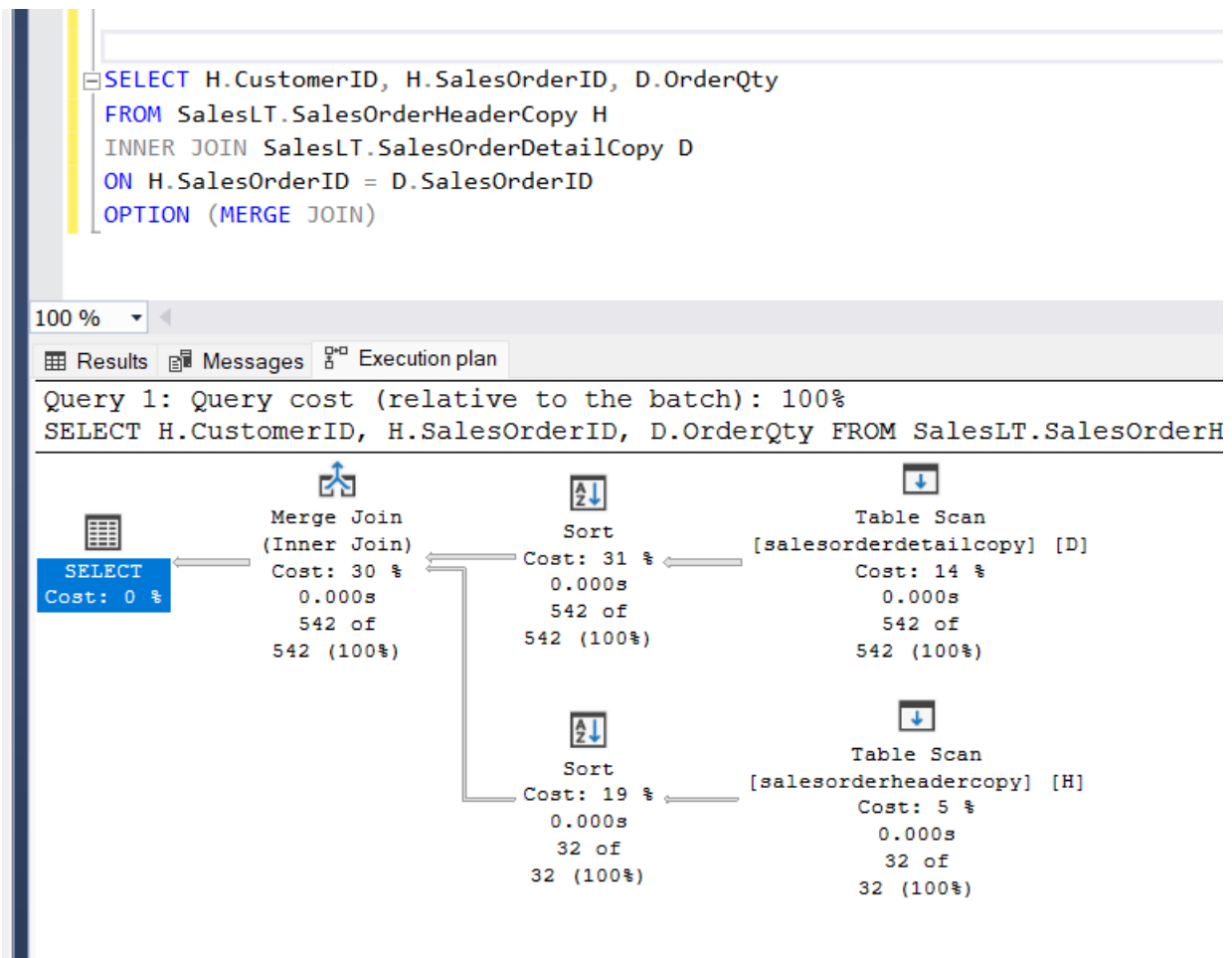
So, I am going to drop this table and then recreate it from the original. So, we're back to 32 rows.

```
drop table saleslt.salesorderheadercopy
```

So, our queries will be lot quicker. So, as a reminder, we currently have an execution plan with a hash match but suppose we did not want to hash match.



suppose we wanted any other type of match, say a merge join. Well, we can do that with the use of query hints and they're fairly easy to use. We write option, and then in brackets, the hint.

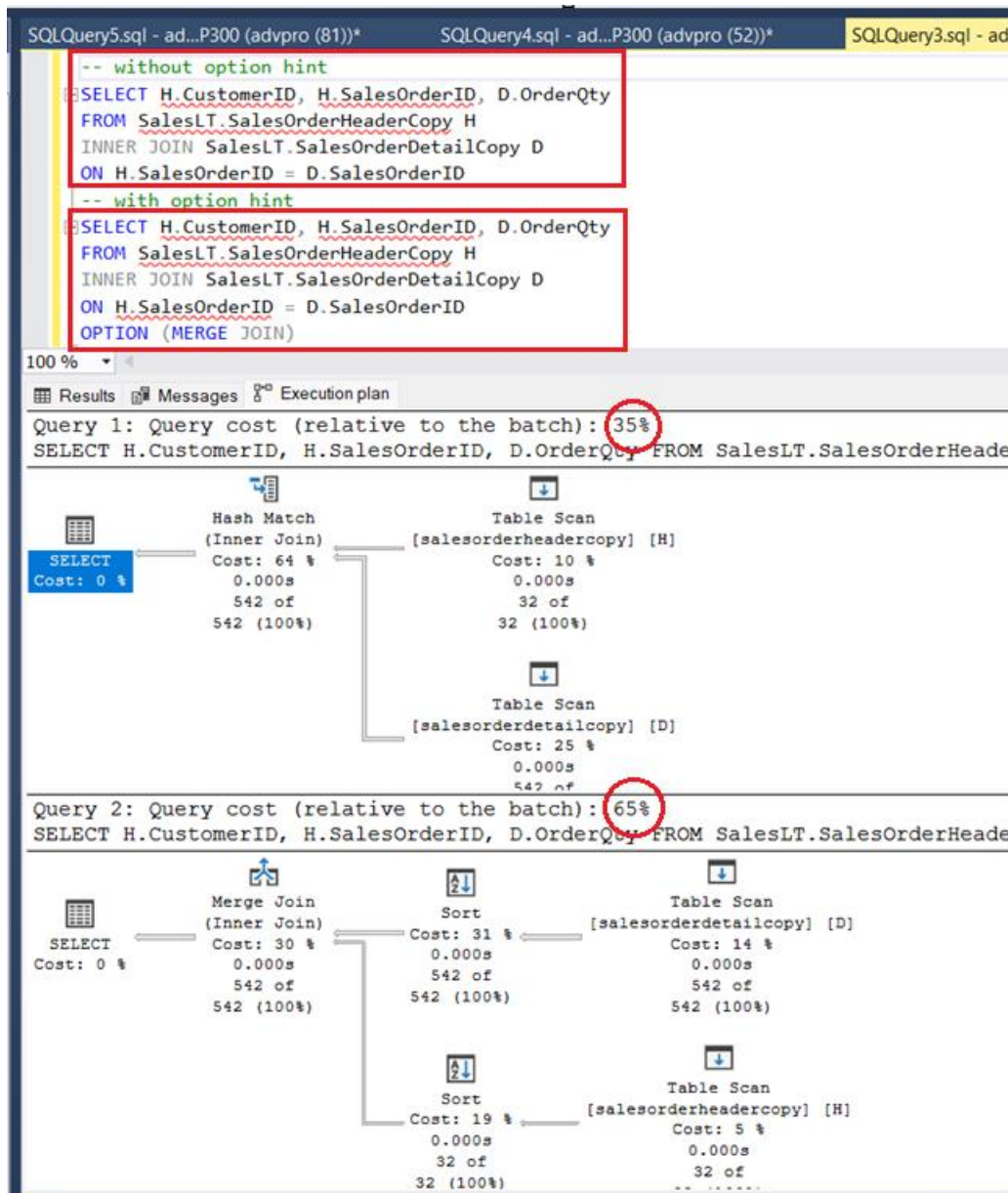


So, in this case, merge join. So, let's run this merger JOIN and have a look at the execution plan.

So, you can now see, we've got these scans and these sorts.

Now remember what I said about SORTS. They take up a lot of time, they're very costly. So, you do need to be aware of that but because of these sorts, we now have this MERGE JOIN.

Now, was this a good thing to have done? What I'm going to do is I'm going to duplicate this query. So now we've got one that's using a MERGE JOIN and one that's using the previous hash match,



You can see that the one with the hash match is about twice as quick as the one with the merge join. So, the first one only uses 35% of the total batch of these two statements, whereas the second uses 65%.

<https://learn.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql-query?view=sql-server-ver16>

Now, if you have a look at the Microsoft documentation regarding hints,

[www.linkedin.com/in/ragasudha-m](http://www.linkedin.com/in/ragasudha-m)

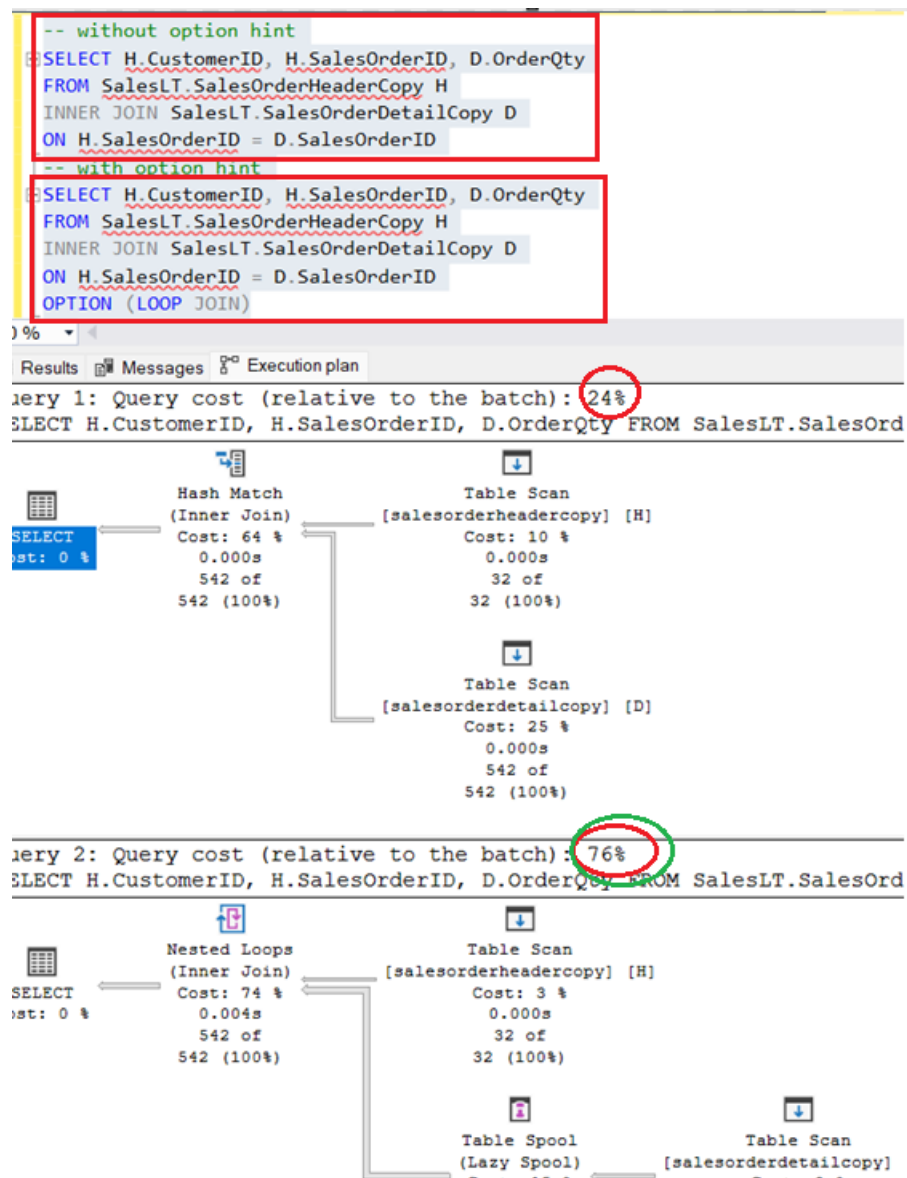
### ⊗ Caution

Because the SQL Server Query Optimizer typically selects the best execution plan for a query, we recommend only using hints as a last resort for experienced developers and database administrators.

I just want to look at some of the main Query hints,

So, we've already seen that you can have option merge join. Well, we can also have other JOINS. We can have a loop join.

So, let's put that in and see how good that is.



And you can see even worse at the loop join, but it is possible to do so.

If we wanted to force the hash JOIN, we could, of course, just write hash join.

```
-- without option hint
SELECT H.CustomerID, H.SalesOrderID, D.OrderQty
FROM SalesLT.SalesOrderHeaderCopy H
INNER JOIN SalesLT.SalesOrderDetailCopy D
ON H.SalesOrderID = D.SalesOrderID
-- with option hint
SELECT H.CustomerID, H.SalesOrderID, D.OrderQty
FROM SalesLT.SalesOrderHeaderCopy H
INNER JOIN SalesLT.SalesOrderDetailCopy D
ON H.SalesOrderID = D.SalesOrderID
OPTION (HASH JOIN)
```

Query 1: Query cost (relative to the batch): 50%

SELECT H.CustomerID, H.SalesOrderID, D.OrderQty FROM SalesLT.Sales

Execution plan for Query 1:

- Hash Match (Inner Join)
  - Cost: 64 %
  - 0.000s
  - 542 of 542 (100%)
- Table Scan [salesorderheadercopy] [H]
  - Cost: 10 %
  - 0.000s
  - 32 of 32 (100%)
- Table Scan [salesorderdetailcopy] [D]
  - Cost: 25 %
  - 0.000s
  - 542 of 542 (100%)

Query 2: Query cost (relative to the batch): 50%

SELECT H.CustomerID, H.SalesOrderID, D.OrderQty FROM SalesLT.Sales

Execution plan for Query 2:

- Hash Match (Inner Join)
  - Cost: 64 %
  - 0.000s
  - 542 of 542 (100%)
- Table Scan [salesorderheadercopy] [H]
  - Cost: 10 %
  - 0.000s
  - 32 of 32 (100%)
- Table Scan [salesorderdetailcopy] [D]
  - Cost: 25 %
  - 0.000s
  - 542 of 542 (100%)

We've also got similar things for groups and unions. So, we've got hash group, order group, merge union, hash union, and concat union, different ways of doing groups and unions.

[www.linkedin.com/in/ragasudha-m](http://www.linkedin.com/in/ragasudha-m)

But again, don't do it unless you absolutely must.

Now, what else do we have?

Now incidentally, if you are putting in more than one option, you can do that. So you can have a comma in between options.

**[KEEP FIXED PLAN]**

Now this query won't be recompiled when its statistics change, it will only be recompiled if the SCHEMA of the underlying tables change.

What's the schema? We're talking about the columns, if you actively run it.

A stored procedure code RECOMPILE,

#EXEC sp\_recompile

That would also change the plan. So, once it is done, it's fixed.

**[KEEP PLAN]**

That recompiles less often when statistics change. So, statistics being, for instance, you've got 10,000 rows for where a particular sale all day is equal to one, but you've only got a hundred rows where it's equal to 100 and you've only got two rows where it's equal to 1000. So that's statistics, but suppose I delete all where the sales order ID = 1,

where there's is no longer these 1000 or 1 million rows that we've got there. So those are statistics.

**[OPTIMIZE FOR UNKNOWN]**

We can also optimize for unknown. So, this uses the average selection of the were, rather than a specific thing. So, I might run this query where the parameter is sales order ID = one. And so, we have got these million rows, but I would say, I just want to do the average number of rows that it would be.

**[ROBUST PLAN]**

So, this creates a plan that works with the maximum potential row size. So, in this case, using the million rows, if you are running a query that doesn't contain these million rows, then the performance may be impaired.

|                         |                                |
|-------------------------|--------------------------------|
| OPTION (KEEPFIXED PLAN) | which doesn't recompile at all |
|-------------------------|--------------------------------|

[www.linkedin.com/in/ragasudha-m](http://www.linkedin.com/in/ragasudha-m)



|                                      |  |
|--------------------------------------|--|
|                                      | unless something major happens                                 |
| <b>OPTION (KEEP PLAN)</b>            | which recompiles left often                                    |
| <b>OPTION (OPTIMIZE FOR UNKNOWN)</b> | which uses the average selectivity, the average number of rows |
| <b>OPTION (ROBUST PLAN)</b>          | which uses the maximum number of rows                          |

inistering-relational-databases-azure-dba/learn/lecture/28211950#anno

```

9  INNER JOIN SalesLT.SalesOrderDetailCopy D
10 ON H.SalesOrderID = D.SalesOrderID
11 GO
12
13 CREATE PROC my_proc(@SalesOrderID int) as
14 SELECT H.CustomerID, H.SalesOrderID, D.OrderQty
15 FROM SalesLT.SalesOrderHeaderCopy H
16 INNER JOIN SalesLT.SalesOrderDetailCopy D
17 ON H.SalesOrderID = D.SalesOrderID
18 WHERE H.SalesOrderID = int
19 OPTION (OPTIMIZE FOR (@SalesOrderID UNKNOWN))
20
21 EXEC sp_recompile
22

```

And then if I was creating a stored procedure, I created a stored procedure. So, my procedure, and I have an input for the sales order ID, and I use it here. Then I could say optimize for, so I can say, create this plan now, where a particular parameter is a particular value, or I could equally say where it is unknown.

If you do not do that, then the stored procedure will be optimize in its first running and we will keep that, so if I call this with a sales order ID of one, then it will be optimized for a million rows. And then when I call it again with sales order, a thousand where there are only a thousand rows, a hundred rows, it will still be using the same plan as if sales order ID were optimized for number one. Oh, I should have an equal sign there, that is why I have the squiggles.

So, optimize for unknown, quite useful. If you want the stored procedure to just optimize for a particular parameter with unknown values. So, these are table hints, so we can use them, but only when necessary, I would suggest. So, we have got merge join, loop join, hash join.

Those are the more widely used hints, but we have a hash group and order group. We have several types of union and then we have, keep fixed plan. So, the plan remains the plan until say, the schema of the table changes.

We have key plan, so that re-optimizes or recompiles left often.

We have optimize for, so we can optimize for known or optimize for a particular value, more often used in stored procedures, I would suggest.

We have a robust plan, creating a plan that works for the maximum potential row size. But the major thing is not use it unless you absolutely must, because quite often, SQL server query optimizer does a fairly good job.

https://learn.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql-query?view=sql-server-ver16

My Drive - Google... Adobe Acrobat

## Syntax

ents

JOIN, APPLY, PIVOT

its

s

```
syntaxsql

<query_hint> ::=
{ { HASH | ORDER } GROUP
| { CONCAT | HASH | MERGE } UNION
| { LOOP | MERGE | HASH } JOIN
| DISABLE_OPTIMIZED_PLAN_FORCING
| EXPAND VIEWS
| FAST <integer_value>
| FORCE ORDER
| { FORCE | DISABLE } EXTERNALPUSHDOWN
| { FORCE | DISABLE } SCALEOUTEXECUTION
| IGNORE_NONCLUSTERED_COLUMNSTORE_INDEX
| KEEP PLAN
| KEEPFIXED PLAN
| MAX_GRANT_PERCENT = <numeric_value>
| MIN_GRANT_PERCENT = <numeric_value>
| MAXDOP <integer_value>
| MAXRECURSION <integer_value>
| NO PERFORMANCE SPOOL
| OPTIMIZE FOR ( @variable_name { UNKNOWN | = <literal_constant> } [ , ...n ] )
| OPTIMIZE FOR UNKNOWN
| PARAMETERIZATION { SIMPLE | FORCED }
| QUERYTRACEON <integer_value>
| RECOMPILE
| ROBUST PLAN
| USE HINT ( <use_hint_name> [ , ...n ] )
| USE PLAN N'<xml_plan>'
| TABLE HINT ( <exposed_object_name> [ , <table_hint> [ [ , ] ...n ] ] )
| FOR TIMESTAMP AS OF '<point_in_time>'
}
```



Question 1:

I want to create an index called "myIndex" on the table dbo.myTable.

The index should use the columns Column1 and Column2.

It needs to re-sort the table, and the columns used need to be unique.

How do I write the T-SQL statement?

☐ **CREATE UNIQUE NONCLUSTERED INDEX myIndex ON dbo.myTable (Column1, Column2)**

☐ **CREATE NONCLUSTERED UNIQUE INDEX myIndex ON dbo.myTable (Column1, Column2)**

☒ **CREATE UNIQUE CLUSTERED INDEX myIndex ON dbo.myTable (Column1, Column2)**

☐ **CREATE CLUSTERED UNIQUE INDEX myIndex ON dbo.myTable (Column1, Column2)**

Question 2:

Which DMV allows you to retrieve the last execution plans?

☐ **sys.dm\_exec\_connections**

☐ **sys.dm\_exec\_query\_stats**

☐ **sys.dm\_exec\_requests**

☒ **sys.dm\_exec\_cached\_plans**

Question 3:

Which DMV allows you to identify long running transactions?

☐ sys.dm\_exec\_active\_transactions

☒ sys.dm\_tran\_active\_transactions

☐ sys.dm\_db\_active\_transactions

☐ sys.tran\_active\_transactions

Question 4:

I want to get information about indexes which could be added so that my queries improve their performance.

Which DMV should I use?

☐ sys.resource\_stats

☐ sys.dm\_exec\_cached\_plans

☒ sys.dm\_db\_missing\_index\_details

☐ sys.dm\_exec\_requests

Question 5:

I want to add a hint in my query, to force my query to use a MERGE JOIN.

What do I add at the end of my query?

☐ HINT (MERGE JOIN)

☐ USING (MERGE JOIN)

☒ OPTION (MERGE JOIN)

☐ WITH (MERGE JOIN)