Chris Singleton 03/12/2017
PROG 140 – Transactions & Performance

Module 8 Assignment

Turn In:

For this exercise, you will submit one WORD documents (instead of a .sql file) in which you have copied and pasted your entire work from your .sql file including all assignment questions and your SQL queries. The document should contain your name at the top, assignment title, the date and any difficulties encountered.

Submit your file to the instructor using the Canvas Assignment tool.
*/

/*
12 pts
1. From our video (and PPT) on Performance, list 3 QUERY Performance tips and give query examples and explanations
of how/why they will help performance:

*/
-- list your answers here:
/*
--Things to remember on performance:
1.Performance Query tools can put a load on the system like (DETA) Database Engine Tuning Advisor, SQL Server Profiler can take up a lot of resources.

2.One thing that will help to improve performance is to have less named instances running on the server.

3.Named instances have their own resources and can slow the server down because of taking up so much in resources.

4.Dont mix production databases and development databases on the same Server (testing databases or staging databases).

5.Truncate tables instead of using delete, because the log file stores information on delete which takes up resources.

```
6.If other users must access your data, let them access on a
replication server, so they do not create performance issues.

7.Use a consistent style coding format. Make it easy to read for
maintenance.

8.Use Joins instead of subqueries. Joins are much faster and better on
performance.

9.Limit nesting of views and sprocs.

10.Good database design can increase performance. (No repeating
groups, 3rd normal form normalized structures)

11.Use stored procedures and views over SQL scripts because of taking
advantage of cache capabilities, but limit nesting levels of views and
sprocs.

12.Limit the use of triggers, because rollbacks are expensive.

13.Limit the use of functions in large queries to enable caching.
*/
--Try not to use:
--The query below can take up a lot of resources, especially if there
are millions of rows.

--Use a filter and/or key words (TOP) whenever possible to get only
the data that is needed, so you don't interrupt others transactions.

SELECT * FROM Customers

--============== First Query =================--

--Instead of using this multiple times (below):
SELECT COUNT(*) AS [Number of Employees]
FROM Employees

/*
  Use after the first time you do the above query. Below displays the
  most recent count of what was counted the last time the system saw
  and is more optimized because it uses the cached content instead of
  looking while using more resources.
*/
SELECT rows
FROM sysindexes
WHERE id = OBJECT_ID('employees') AND indid > 2
```

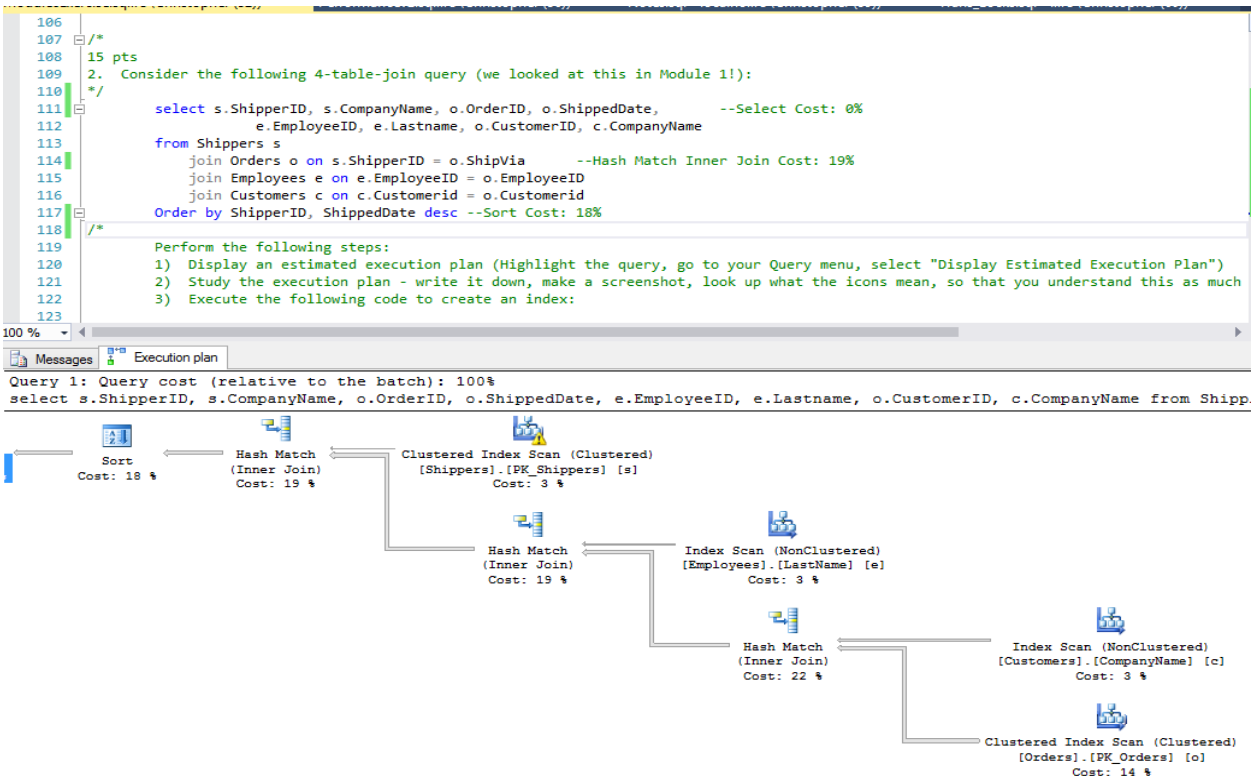
```sql
--============== Second Query ==============--

--Pattern matching is a performance hit.
--Try not to do this: (This is like searching all the indexes for a
descriptive string.)
SELECT ProductName
FROM Products
WHERE ProductName LIKE 'Boston Crab%'

--Instead do a range because in using a range is more efficient on
resources which frees up resources for others and reduces locking:
SELECT TOP (10) ProductName --Use of a range in the select clause.
FROM Products
ORDER BY ProductName ASC

--Or you can do:
SELECT TOP (10) ProductName
FROM Products
WHERE ProductName > 'A' and ProductName  < 'C' --Filter a range.
ORDER BY ProductName DESC --Shows at the Top of the list.

--============== Third Query ==============--

--Use Joins instead of queries whenever possible to speed up
performance. With joins, the system can make the best determination on
how to handle the query to bring the fastest results.
--The SQL Server system is optimized to use joins more efficiently
with all its available resources.
--During a subquery the user specifies how to derive verses the SQL
Server system specifying with deriving results.

SELECT
      C.CustomerID,
      C.CompanyName
FROM
      dbo.Customers AS C
WHERE
      EXISTS (
      SELECT *
      FROM dbo.Orders AS OH
      WHERE OH.CustomerID = C.CustomerID AND OH.OrderDate >= '19980101'
                                         AND OH.OrderDate < '19990101'
      )
ORDER BY CompanyName
```

```
--This one is much more efficient than the one above and the system is
more optimized to process joins.

SELECT DISTINCT
      C.CustomerID,
      C.CompanyName
FROM
      dbo.Customers AS C
      INNER JOIN dbo.Orders AS OH
            ON OH.CustomerID = C.CustomerID
WHERE OH.OrderDate >= '19980101' AND OH.OrderDate < '19990101'
ORDER BY CompanyName


/*
15 pts
2.  Consider the following 4-table-join query (we looked at this in
Module 1!):
*/
            select s.ShipperID, s.CompanyName, o.OrderID, o.ShippedDate,
--Select Cost: 0%
            e.EmployeeID, e.Lastname, o.CustomerID, c.CompanyName
            from Shippers s
                  join Orders o on s.ShipperID = o.ShipVia
--Hash Match Inner Join Cost: 19% --NonClustered Index added Cost: 29%
                  join Employees e on e.EmployeeID = o.EmployeeID
                  join Customers c on c.Customerid = o.Customerid
            Order by ShipperID, ShippedDate desc
          --Sort Cost: 18% --NonClustered Index added Cost:24%
/*
Perform the following steps:
1)  Display an estimated execution plan (Highlight the query, go to
your Query menu, select "Display Estimated Execution Plan")
2)  Study the execution plan - write it down, make a screenshot, look
up what the icons mean, so that you understand this as much as you
can.
3)  Execute the following code to create an index:
*/
      CREATE NONCLUSTERED INDEX idxOrdersShipVia
                  ON [dbo].[Orders] ([ShipVia])
      INCLUDE ([OrderID],[CustomerID],[EmployeeID],[ShippedDate])
/*
4)  Do step #1 again, ie., display the estimated execution plan
```

5) Discuss below how the index changed (or did not change) the execution plan for the query. Discuss whether or not you think this index should be permanently implemented on the Northwind database. Use the course discussion area for this module to get input from other students.
***/**
-- list your answers here:
/*
The Select cost did not change at all, but the Sort cost and Hash Match percentage numbers went up by nearly a third of their original values. Since I did not see anywhere in the discussion, I will take a crack at this.

I think it is not a good idea to permanently keep this index even though it could be used often to gauge when the order was shipped.

Below is what I think is the right answer:
The best option that you can do is to use a view instead of the index, which would be more efficient or even better I would recommend replicating data on another server altogether; then people querying queries like these are not taking up as much resources that would or could interfere with transactional processing.

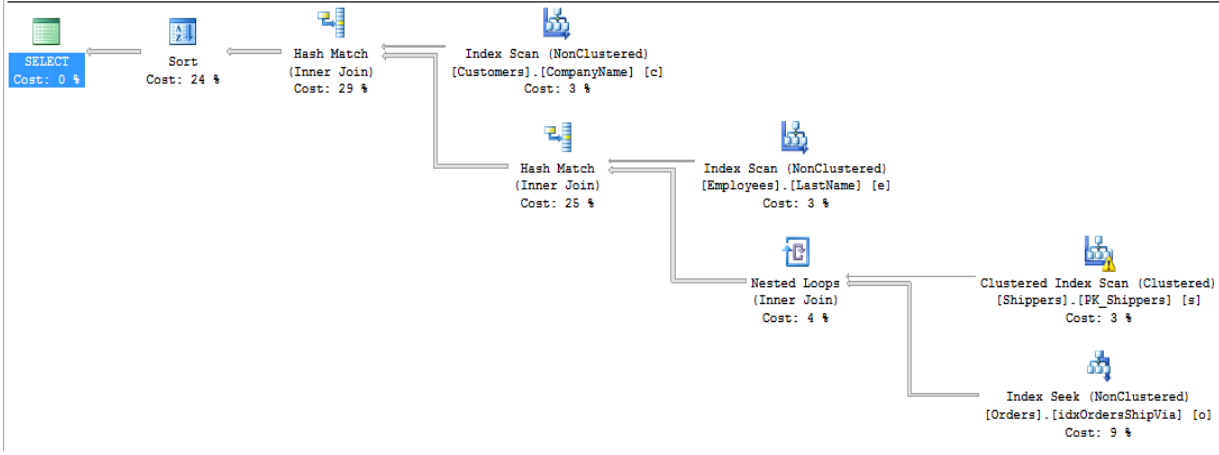


```
2.  Consider the following 4-table-join query (we looked at this in Module 1!):
*/
        select s.ShipperID, s.CompanyName, o.OrderID, o.ShippedDate,       --Select Cost: 0%
                   e.EmployeeID, e.Lastname, o.CustomerID, c.CompanyName
        from Shippers s
            join Orders o on s.ShipperID = o.ShipVia      --Hash Match Inner Join Cost: 19%  --NonClustered Index added Cost: 29%
            join Employees e on e.EmployeeID = o.EmployeeID
            join Customers c on c.Customerid = o.Customerid
        Order by ShipperID, ShippedDate desc --Sort Cost: 18%  --NonClustered Index added Cost:24%
/*
        Perform the following steps:
        1)  Display an estimated execution plan (Highlight the query, go to your Query menu, select "Display Estimated Execution Plan")
        2)  Study the execution plan - write it down, make a screenshot, look up what the icons mean, so that you understand this as much as you ca
        3)  Execute the following code to create an index:
*/
                    CREATE NONCLUSTERED INDEX idxOrdersShipVia
                        ON [dbo].[Orders] ([ShipVia])
                        INCLUDE ([OrderID],[CustomerID],[EmployeeID],[ShippedDate])
/*
        4)  Do step #1 again, ie., display the estimated execution plan
        5)  Discuss below how the index changed (or did not change) the execution plan for the query.  Discuss whether or not you
             think this index should be permanently implemented on the Northwind database. Use the course discussion area for this module
             to get input from other students.
```

100 %

Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
select s.ShipperID, s.CompanyName, o.OrderID, o.ShippedDate, --Select Cost: 0% e.EmployeeID, e.Lastname, o.CustomerID, c.CompanyName…
```




```
*/
/*
15 pts
3.  Write code below using the Northwind database that will:

a)  Create a transaction
b)  Write a query to perform an update on the [order details] table
    that will add 10% to all unit prices.
c)  Create a save point after the query in b)
d)  Write a query to to perform an update on the Products table that
    will add $2.00 to all unit prices
e)  Write the statement to roll back to the save point
f)  Right under the statement in e), Write a statement that rolls
    back to the beginning of the Transaction.

*/
```

```
-- list your answers here:
--Checking...
--SELECT * FROM [Order Details]

--========== Update Order Details ==========--

BEGIN TRANSACTION
      UPDATE [Order Details]
      SET UnitPrice = UnitPrice + (UnitPrice * .1)
      --Updates UnitPrice by 10%
SAVE TRANSACTION Upd_UnitPrice10Perc;
      ROLLBACK TRANSACTION Upd_UnitPrice10Perc

--10248    11    0.2479     12    0 --.02479 +0.2479 = 0.2727 Correct
--10248    11    0.2727     12    0

--============ Update Products ============--

--Checking...
-- SELECT * FROM Products
--Note: Even the nulls will be updated to 2. Nulls need to be handled.

BEGIN TRANSACTION
      UPDATE [Products]
      SET UnitPrice = ISNULL(UnitPrice,0) + 2
       --Adds $2.00 to each Unit Price.
SAVE TRANSACTION Upd_UnitPriceBy2Dol
      ROLLBACK TRANSACTION Upd_UnitPriceBy2Dol

/*
8 pts
4.
You have a colleague that is executing some work in a transaction on
the Customers table. She is running her transaction with an isolation
level of Serializable.

You need to run a quick query for your boss that groups and counts all
customers by their Country.  Write the query below that will execute
(and not be blocked):
```

```
*/

-- list your answers here:
--Some Commands to remember by:
SELECT @@SPID --Session Number.
EXEC sp_lock -- find out about the locks
dbcc useroptions --Option Settings.
EXEC sp_who2 52 --User's session state/ More info
EXEC sp_who 52 --User's session state/In General

/*
 --Checking... SELECT TOP (10) * FROM Customers
 Overrides session level locking (Even serializable).
 Note: This NOLOCK is not good to use with Update statements
 because it may not be dependable or good data.
 Select statements are OK to use when it is needed.
*/

SELECT Country, COUNT(CustomerID) AS Customer_Count
FROM Customers WITH(NOLOCK)
GROUP BY Country
ORDER BY Customer_Count DESC
```