Name: Justin Baskaran, Caleb Boraby

**Before and after was run with Linux.**

**Homework Response 1:**

The script starts off deleting all entries in the Sales.SalesOrderHeader that have a SalesOrderID greater then 75123.

Then the script declare a variable called ‘MYCOUNTER’ and initializes it to 0.

Thirdly, the script begins a while statement that iterates eight times, copying the Sales.SalesOrderHeader table to itself, while modifying the ‘Comment’ table.

The ‘Comment’ column takes the CustomerID adds 4,2,1000,3000, or 300. This value is then used to calculate a random integer value that can be used to produce a reproducable value. This pseudo-random value is then multiplied the value by 2000. The resulting value is then rounded to one decimal place value.

**Homework Response 2:**

8055040 rows

Query set 1:

|  |  |  |  |
| --- | --- | --- | --- |
| Query ID | Before Index Average | After Index Average | Comments |
| 1A | 00:00:01.523 | 00:00:12.038 | Indexing and ordering the columns helps speed up the query a lot. |
| 1B | 00:00:01.001 | 00:00:00.004 | Since the column that is being selected has been sorted since it has been indexed, the time has been sped up. |
| 1C | 00:00:01.099 | 00:00:00.008 | Since one of the columns being selected hasn’t been sorted, this one will be a little slower than 1B but still way faster after being indexed. |

For Query set 1: Enter the command used to create the index:

CREATE INDEX idx\_cccr

on sales.SalesOrderHeader (CreditCardId , CurrencyRateId);

Query set 2:

|  |  |  |  |
| --- | --- | --- | --- |
| Query ID | Before Index Average | After Index Average | Comments |
| 2A | 00:00:01.325 | 00:00:00.155 | The after index time was an improvement from the before indexing time because the row being selected, TerritoryID, has been sorted |
| 2B | 00:00:01.198 | 00:00:01.129 | This didn’t have much of an improvement because the rows being selected weren’t indexed or sorted. |
| 2C | 00:00:01.427 | 00:00:02.134 | This time was worse since the query didn’t select any indexed rows and an index was still used. |
| 2D | 00:00:01.981 | 00:00:00.888 | There was a speedup here because one of the rows being selected had been indexed and was also in the GROUP BY clause, so it was already sorted which sped up the group by part. |

For Query set 2: Enter the command used to create the index:

CREATE INDEX **idx\_cccti** ON sales.SalesOrderHeader(TerritoryID);

Query set 3:

|  |  |  |  |
| --- | --- | --- | --- |
| Query ID | Before Index Average | After Index Average | Comments |
| 3A | 00:00:28.888 | 00:00:27.806 | The indexed time is better than the before indexed time because having the comment column indexed and sorted makes searching through it for the WHERE clause faster. |
| 3B | 00:00:01.550 | 00:00:01.461 | This is faster after the index too because the comment column has been sorted which makes it faster for the WHERE clause to run. |

For Query set 3: Enter the command used to create the index:

CREATE INDEX **idx\_cmt** ON sales.SalesOrderHeader(comment);

Query set 4:

|  |  |  |  |
| --- | --- | --- | --- |
| Query ID | Before Index Average | After Index Average | Comments |
| 4A | 00:00:00.013 | 00:00:00.014 | Since none of the indexed columns are used anywhere except the SELECT statement, sorting the columns didn’t help speed up the runtime. |
| 4B | 00:00:01.901 | 00:00:00.017 | This takes a little longer since it needs to select from more rows but after indexing is still better than before. |

For Query set 4: Enter the command used to create the index:

Index:

CREATE INDEX idx\_oddd ON sales.SalesOrderHeader(OrderDate, DueDate);

Before Index Average:

A)

SELECT [RevisionNumber]

,[OrderDate]

,[DueDate]

FROM [AVtest].[Sales].[SalesOrderHeader]

WHERE [SalesOrderID] >43000

AND [SalesOrderID] BETWEEN 0 AND 100000;

B)

SELECT [SalesOrderID]

,[RevisionNumber]

,[OrderDate]

,[DueDate]

FROM [AVtest].[Sales].[SalesOrderHeader]

WHERE [SalesOrderID] >43000

OR [SalesOrderID] < 100000 ;