Analyzing and Optimizing Queries

Objective

After completing this lab, you will be able to:

- Analyze the performance of queries using text-based and graphical analysis tools.
- Perform some common optimizations on SELECT statements.

Required Materials

- SQL Server Management Studio
- Your copy of the Northwind database

Related Reading

- http://www.sql-server-performance.com/
  This site, although geared towards Microsoft SQL Server optimization, also contains helpful pointers for optimizing queries on relational databases in general.
- http://www.bcarter.com/optimsql.htm
  Additional information on SQL query optimization. Coverage of SQL query optimization is not as broad as what is provided by the above site, but it is much more focused and accessible.

Formatting Details

Tasks in this lab are written on a green background and are enclosed by a dashed border.

This is an example of text describing a task.

Tasks in this lab with a deliverable item are written on a yellow background and are enclosed by a solid border.

This is an example of text describing a task with a deliverable.

Turn-in Instructions

The only turn-in is the ANGEL survey title "Lab Questions".

Assignment Details

(1) Query Analysis Tools

SQL Server provides a comprehensive suite of tools to facilitate query analysis and optimization. We will take a brief look at a few of them in this part of the lab.

The tools covered in this section are the execution plan viewer and retrieval mechanisms for query statistics.

A) **The Execution Plan**

The query optimizer is a component of a relational database system that both optimizes a standardized[1] version of a user's SQL query and creates an execution

1 The process of query standardization occurs after parsing. The major action done in query standardization is elimination of redundant clauses.

plan to be executed by the database access routines. It optimizes the query on the basis of CPU and I/O cost, which it measures in seconds.

SQL Server includes two tools to view the execution plan devised by the query optimizer.

B) **The Execution Plan Viewer**

We will first discuss the graphical execution plan viewer in SQL Server Management Studio, and then briefly cover the text-based analysis tool provided by SQL Server

In SQL Server Management Studio, click Query > Include Actual Execution Plan while you are in Query Editor.

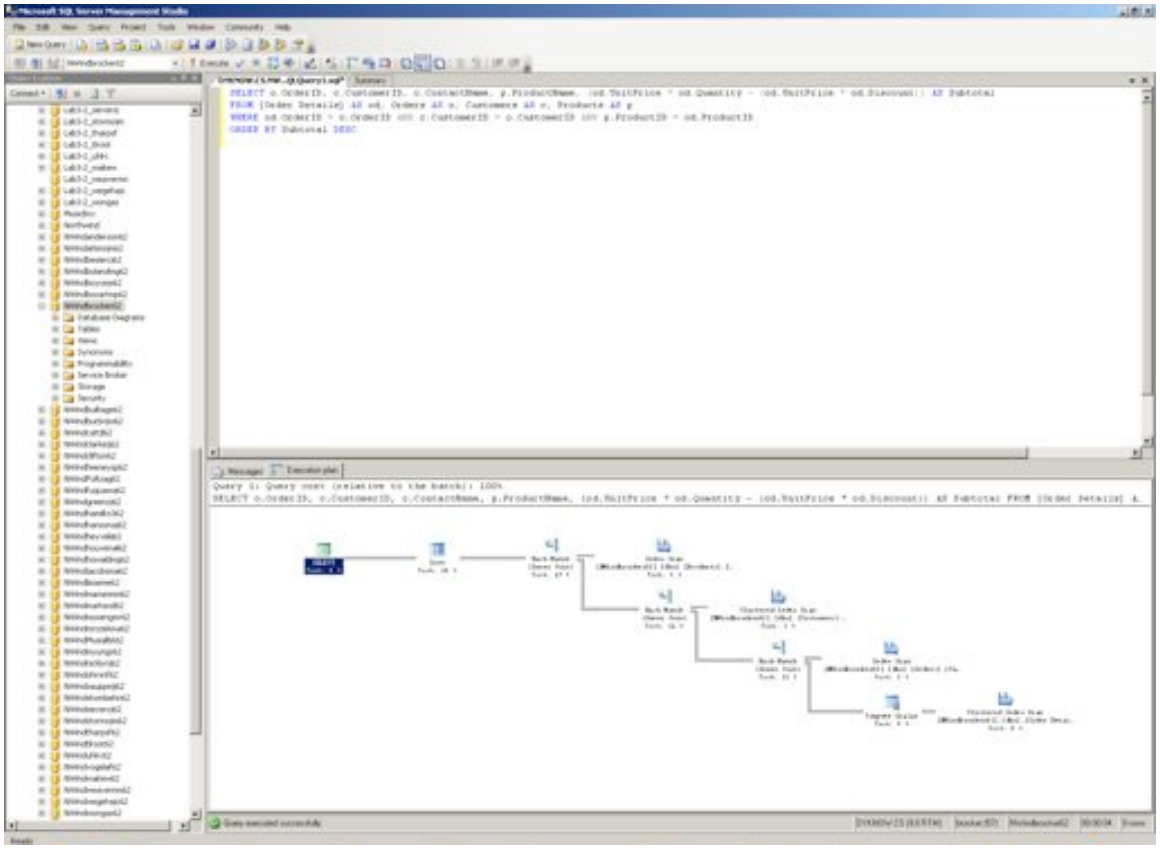
This action will cause the SQL Server Management Studio to request the execution plan for a query from SQL Server and present a graphical representation of the execution plan.

After ensuring that you are using your copy of the Northwind database, input and execute the following query:

```
SELECT o.OrderID, c.CustomerID, c.ContactName, p.ProductName,
    (od.UnitPrice * od.Quantity * (1 - od.Discount)) AS Subtotal
FROM [Order Details] AS od, Orders AS o, Customers AS c, Products AS p
WHERE od.OrderID = o.OrderID AND c.CustomerID = o.CustomerID AND
p.ProductID = od.ProductID
ORDER BY Subtotal DESC
```

This query retrieves information on the products ordered by customers, as well as each order's subtotal. It is ordered by Subtotal in descending order to highlight the customers with the largest subtotals. (Modulo changes you have made to your copy of Northwind in previous labs, this query should return 2155 rows.)

When the query completes, SQL Server Management Studio will have a third tab in its results pane labeled "Execution Plan". This is like the estimated execution plan from the Indexes lab, but shows what query steps were actually used. The execution plan should resemble the screen below.



Your screen will (hopefully) be more legible than this example. You can zoom the Execution Plan display by right-clicking on the display and selecting one of the Zoom options.

The execution plan is designed to be read from right-to-left. Inputs into each node are indicated by arrows, and each node (except left- and right-most nodes) operate on their inputs in some fashion. Hovering the mouse pointer over an item will cause a tooltip resembling the image below to appear.

The tooltip provides information such as the operation being executed, the number of rows being operated on, I/O, CPU, and aggregate costs, and the actual SQL code used to invoke the operation.

C) **Viewing the Execution Plan through SQL Server**

It is also possible to retrieve details on the execution plan directly from SQL Server. This section briefly describes how to do so.

Modify the query you used in part (B) by placing the SQL commands

```
SET SHOWPLAN_ALL ON
GO
```

at the top of the query. Disable the graphical execution plan, and then re-run the query.

The **SHOWPLAN_ALL** flag causes SQL Server to return a table with the same information present in the graphical execution plan viewer. To disable the plan, you must issue the commands

```
SET SHOWPLAN_ALL OFF
GO
```

Set the SHOWPLAN_ALL flag to off by issuing the commands

```
SET SHOWPLAN _ALL OFF
GO
```

D) **Other Analysis Tools**

In addition to a query's execution plan, SQL Server provides aggregate statistics on a query's performance. To see these statistics, you need to set the STATISTICS flag to ON.

Ensure that SHOWPLAN_ALL and the graphical execution plan are disabled, and then set the STATISTICS flags to ON by issuing the commands

```
SET STATISTICS IO ON
SET STATISTICS TIME ON
GO
```

Execute this query, and then re-run the query in part (B). Look at the "Messages" tab to see the statistics.

The **STATISTICS IO** and **STATISTICS TIME** flags cause SQL Server to output aggregate statistics on I/O (disk) usage and processing time respectively in the "Messages" tab.

(2) Indexes and Other Table Optimizations

Indexes are a major part of query optimization. Used correctly, they can drastically reduce I/O load on relational databases, especially on aggregate operations such as COUNT, MAX, MIN, and so forth.

In some cases, data for queries can be retrieved directly from a table index, thus eliminating any need to scan through the table to retrieve data. When the entire query can be satisfied by the data in an index, the index is said to *cover* the query.

However, indexes are not a cure-all for SQL query optimization problems. Indexing every column in a table, for example, often does not improve speed: it merely adds a large amount of bloat to the database. We will see an example of a query that – although benefiting from indexes – may benefit more from other optimizations, such as modifying the data definition.

In this part of the lab, you will investigate the effect of indexes in various situations. Finally, you will be asked to answer some questions about the effectiveness of indexes, and where they should and should not be applied.

A) This part of the lab involves comparison of queries operating on tables with and without indexes. Consequently, you will need to copy some tables from the Northwind database without copying their indexes. This can be done using the Import Wizard as described here:

(1) Open SQL Server Management Studio and right click on your Northwind. Select Tasks> Import Data

(2) In "Choose a Data Source", fill in the appropriate fields with the source "SQL Native Client", server name "dyknow.cs.rose-hulman.edu", your username, and your password. Ensure that the source database is *your copy* of Northwind.

(3) In "Choose a Destination", fill in the appropriate fields with the server name (dyknow.cs.rose-hulman.edu), your username, and your password. Ensure that the destination database is again *your copy* of Northwind.

(4) In "Specify Table Copy or Query", select "Copy data from one or more tables or views".

(5) Select the tables Order Details, Customers, Orders, and Products from Northwind. You do NOT want to copy them over your existing tables, so copy them to a different destination – say, zOrder Details, zCustomers, and so forth. You can do this by selecting the table name in the Destination cell and editing the destination name.

(6) Finish this task with immediate execution.

(7) Verify that your four new tables appear in the Object Explorer in your copy of Northwind and that they have no indexes. (Refresh to see the tables.)

B) The optimizing effect of properly used indexes can be seen most dramatically by running the same query on two different data sets. You will do that here.

(1) Enable the graphical execution plan viewer in SQL Server Management Studio, or set SHOWPLAN_ALL to ON, depending on which you prefer.

(2) Execute the following query:

```
SET STATISTICS IO ON
SET STATISTICS TIME ON
GO
SELECT MAX(OrderID) FROM [zOrder Details] --your new unindexed table
SELECT MAX(OrderID) FROM [Order Details]
GO
```

(3) The questions Q1-Q3 below are to be answered in the ANGEL survey labeled "Lab Questions", in the folder for this lab.

(Q1) What are the differences in the execution plans for these two queries?

(Q2) What is the difference in I/O and CPU cost for these two queries? (Hint: Look at the individual components of the execution plan to determine this; aggregate statistics can be difficult to interpret for multiple queries.)

(Q3) What can you conclude about the relationship between aggregate data and indexes insofar as time optimization is concerned? (Hint: If you need some more examples, run and analyze the following additional queries:

```
SELECT MIN(OrderID) FROM [zOrder Details]
SELECT MIN(OrderID) FROM [Order Details]
SELECT AVG(OrderID) FROM [zOrder Details]
SELECT AVG(OrderID) FROM [Order Details]  )
```

C) The query used in the above section was very contrived: it is not something that one would often, if ever, run. In this section, we will analyze a more realistic query.

(1) Enable the graphical execution plan viewer in SQL Server Management Studio, or set SHOWPLAN_ALL to ON, depending on which you prefer.

(2) Execute the query batch

```
SELECT o.OrderID, c.CustomerID, c.ContactName, p.ProductName,
        (od.UnitPrice * od.Quantity * (1 - od.Discount)) AS Subtotal
FROM [zOrder Details] AS od, zOrders AS o, zCustomers AS c, zProducts AS p
WHERE od.OrderID = o.OrderID AND c.CustomerID = o.CustomerID AND
p.ProductID = od.ProductID

SELECT o.OrderID, c.CustomerID, c.ContactName, p.ProductName,
        (od.UnitPrice * od.Quantity * (1 - od.Discount)) AS Subtotal
AS Subtotal
FROM [Order Details] AS od, Orders AS o, Customers AS c, Products AS p
WHERE od.OrderID = o.OrderID AND c.CustomerID = o.CustomerID AND
p.ProductID = od.ProductID
```

If necessary, replace zProducts, zCustomers, zOrder Details, and zOrders with the names of your indexed tables.

(3) **Questions Q4 and Q5 below are to be answered in the ANGEL survey labeled "Lab Questions".**

(**Q4**) This is the same situation that we examined in part (B), so your results should be similar. However, there should also be some subtle similarities between the two queries. What are these similarities? (Hint: Look at the I/O costs for the index and table scans on the Customers index and table, respectively.)

(**Q5**) Why do you think these similarities exist? (Hint: Think about how the query uses the Customers table and what sort of index is defined for Customers.)

(3) Optimizing SELECT Statements

Although the query optimizer often chooses an extremely fast execution plan, there are times when badly written SELECT statements can limit a query's execution speed.

In this section, pairs of SELECT statements that can accomplish the same task at different speeds are provided. For each SELECT statement, select the faster statement, and explain your choice. Do this in the ANGEL survey for this lab. You may find the links provided in the Related Reading section to be useful.

(Q6)	SELECT count(*) FROM table_name
	SELECT rows FROM sysindexes WHERE id = OBJECT_ID('table_name') AND indid < 2
(Q7)	DECLARE @age int SET @age = "30" DECLARE @service_years int SET @service_years = "10" SELECT employee_id FROM employees WHERE age = @age and service_years = @service_years
	SELECT employee_id FROM employees WHERE age = 30 and service_years = 10
(Q8)	DECLARE @InvoiceTotal money SELECT @InvoiceTotal = sum(UnitPrice*Quantity) FROM [order details] WHERE orderid = 10248 SELECT @InvoiceTotal InvoiceTotal
	DECLARE @LineTotal money DECLARE @InvoiceTotal money SET @LineTotal = 0 SET @InvoiceTotal = 0 DECLARE Line_Item_Cursor CURSOR FOR SELECT UnitPrice*Quantity FROM [order details] WHERE orderid = 10248 OPEN Line_Item_Cursor FETCH NEXT FROM Line_Item_Cursor INTO @LineTotal WHILE @@FETCH_STATUS = 0 BEGIN SET @InvoiceTotal = @InvoiceTotal + @LineTotal FETCH NEXT FROM Line_Item_Cursor INTO @LineTotal END CLOSE Line_Item_Cursor DEALLOCATE Line_Item_Cursor SELECT @InvoiceTotal InvoiceTotal

(Q9)	USE Northwind UPDATE Products SET UnitPrice = UnitPrice * 1.06 WHERE UnitPrice > 5 GO USE Northwind UPDATE Products SET UnitPrice = ROUND(UnitPrice, 2) WHERE UnitPrice > 5 GO
	USE Northwind UPDATE Products SET UnitPrice = ROUND(UnitPrice * 1.06, 2) WHERE UnitPrice > 5 GO
(Q10)	SELECT column_name1, column_name2 FROM table_name1 WHERE column_name1 = some_value UNION SELECT column_name1, column_name2 FROM table_name1 WHERE column_name2 = some_value
	SELECT DISTINCT column_name1, column_name2 FROM table_name1 WHERE column_name1 = some_value OR column_name2 = some_value

(4) Please complete the anonymous lab feedback survey on Angel
under Materials -> Lab Feedback. Your feedback will help us improve the labs for future students."

Turn-in Instructions

The only turn-in is the ANGEL survey title “Lab Questions”.

Revision History

Jan. 23, 2007: Minor clarifications, Curt Clifton.
Jan. 19, 2007: Updated to SQL Server 2005 by Eliza Brock
Jan. 23, 2006: More slight updates, Curt Clifton.
Jan 22, 2006: Slight updates by Steve Chenoweth.
Jan 20, 2005: Completion of indexes section by David Yip; integration of CSSE333 Introduction to Databases – Lab Assignment 9

SELECT material from Jennifer Ford.
Jan 19, 2005: Writing on indexes section begun by David Yip