Eugene Borts

Applied Database II

Dr. Ron Eaglin

Assignment 12

Introduction

This report details the performance analysis and optimization of an SQL query using a specified optimization technique. The report is divided into three parts: the first part is an analysis of the test query prior to the introduction of an optimization technique, the second part is a summary of the optimization technique used, and the third part is the post-optimization performance analysis.

The database used to perform the optimization analysis is the Florida Housing Database, which includes the primary table FLHousing, which is a set of raw data, and the supplemental tables ACR, Categories, Codes, FS, Region, and Type. The test query used to perform both the pre-optimization and post-optimization analysis is shown below.

Test Query

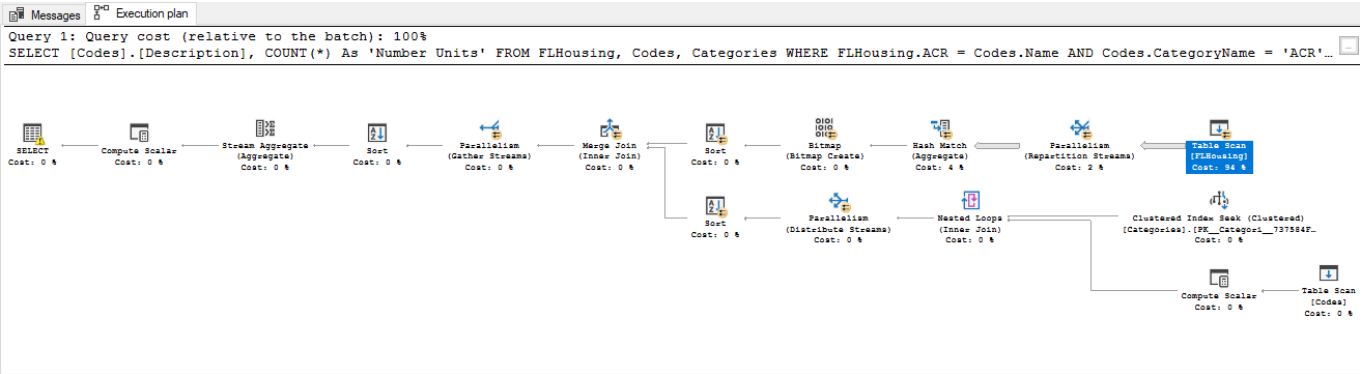
```
SELECT   [Codes].[Description],  COUNT(*) As 'Number Units'
FROM     FLHousing, Codes, Categories
WHERE    FLHousing.ACR = Codes.Name
  AND    Codes.CategoryName = 'ACR'
  AND    Categories.Name = Codes.CategoryName
GROUP BY [Codes].[Description]
```

Part I: Pre-Optimization Performance

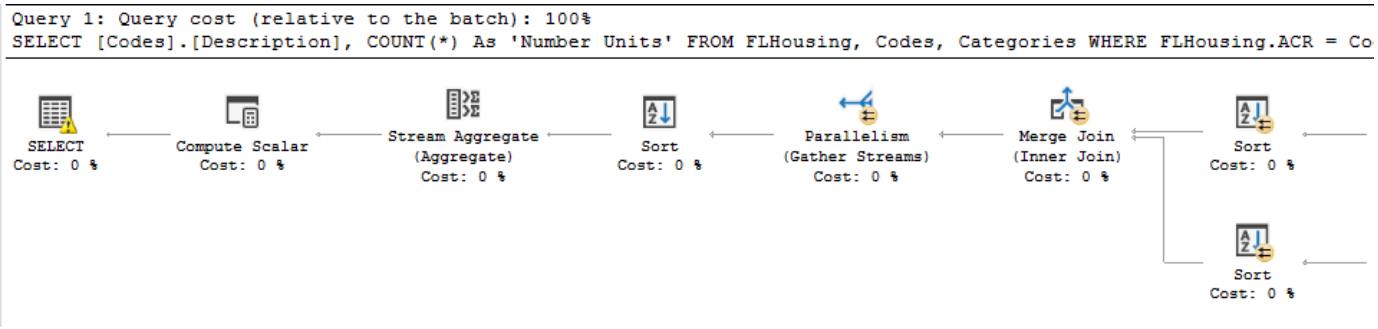
An analysis of query performance by finding the amount of time it takes for the query to run. As shown below, the test query took 7 seconds to run.

LAPTOP-N4N80KJ6 (14.0 RTM) | LAPTOP-N4N80KJ6\Yevge ... | FloridaHousingDB | 00:00:07 | 0 rows

Execution Plan - Full



Execution Plan – Left Side



Execution Plan – Right Side

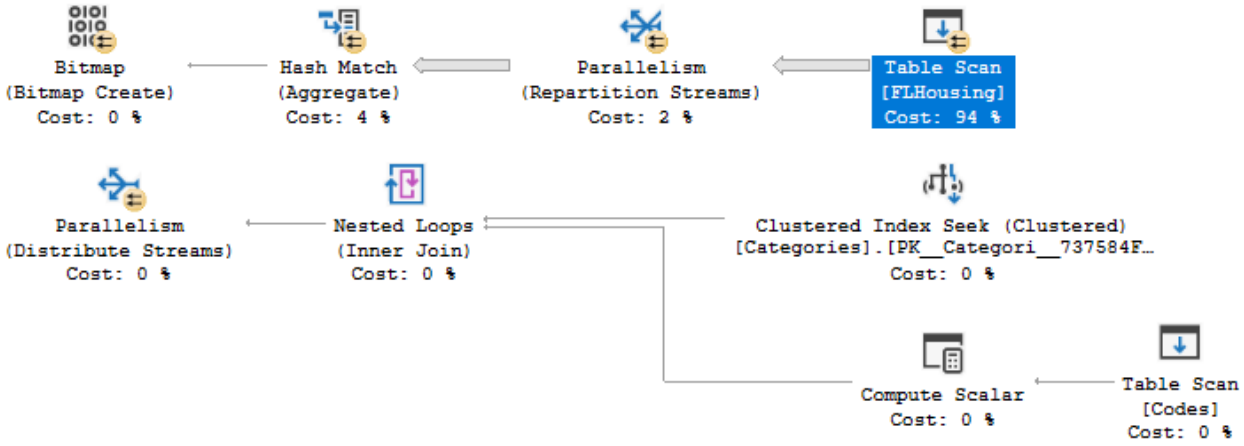


Table Scan

Table Scan

Scan rows from a table.

Physical Operation	Table Scan
Logical Operation	Table Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated I/O Cost	20.2291
Estimated Operator Cost	20.3794 (94%)
Estimated CPU Cost	0.150222
Estimated Subtree Cost	20.3794
Estimated Number of Executions	1
Estimated Number of Rows	273060
Estimated Number of Rows to be Read	273060
Estimated Row Size	11 B
Ordered	False
Node ID	10

Object
[FloridaHousingDB].[dbo].[FLHousing]
Output List
[FloridaHousingDB].[dbo].[FLHousing].ACR

SQL Server Profile Trace

EventClass	TextData	DatabaseID	DatabaseName	ObjectID	ObjectName	ServerName	BinaryData	SPID	StartTime
Trace Start									2019-04-13 15:31:37...
ExistingConnection	-- network protocol: LPC set quote...	16	FloridaHo...			LAPTOP-...	0X60080...	53	2019-04-13 15:21:40...
SQL:BatchStarting	SELECT @@SPID;	16	FloridaHo...			LAPTOP-...		53	2019-04-13 15:32:01...
SQL:BatchStarting	SELECT [Codes].[Description], CO...	16	FloridaHo...			LAPTOP-...		53	2019-04-13 15:32:01...

Part II: Optimization Used

The method of optimization used to increase the performance of this query is indexing. Indexing is an optimization technique in which indexes are created for each table to quickly retrieve data from the database. These indexes work to speed up searches and queries while staying invisible to users.

There are two types of indexes that can be used for a table or view, clustered and non-clustered. Clustered indexes sort and store data rows based on their key values, while non-clustered indexes have a structure that is separate from the data rows. Non-clustered indexes contain non-clustered index key values, and each key value has a pointer to the data row that contains the key value, called a row locator.

The query used to create the indexes is shown below.

Query for Indexes

```
USE FloridaHousingDB

CREATE INDEX HousingIndex
ON FLHousing (ACR, FS, REGION, TYPE)

CREATE INDEX ACRIndex
ON ACR (Code, Description1)

CREATE INDEX CodesIndex
ON Codes (Name, CategoryName, Description)

CREATE INDEX FSIndex
ON FS (Code, Description1)

CREATE INDEX TypeIndex
ON Type (Code, Description1)

CREATE INDEX CategoriesIndex
ON Categories (Name, Description)
```

```
CREATE INDEX RegionIndex
ON REGION (Code, Description1)
```

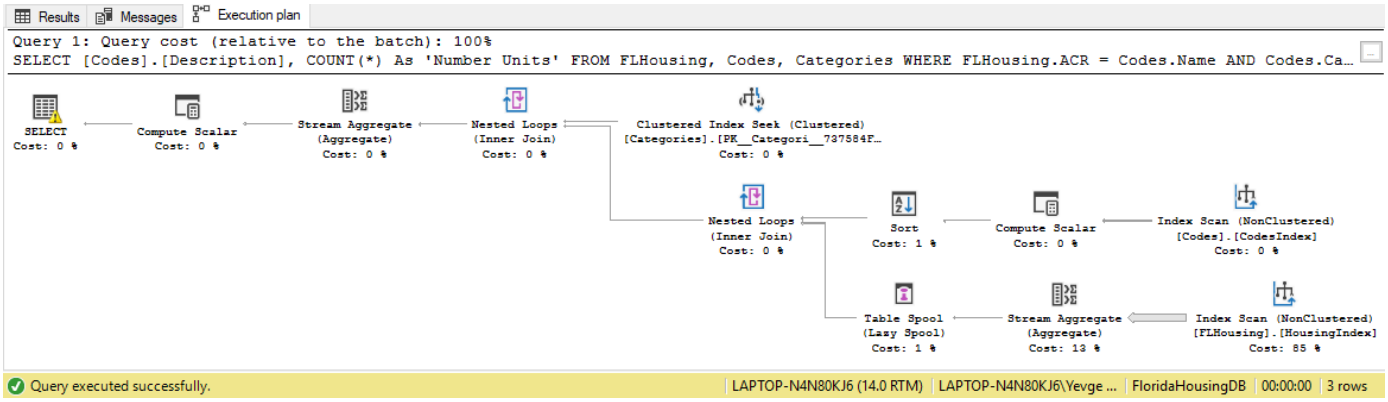
Part III: Post-Optimization Performance

Shown below is the total time taken to execute the primary query after creating the indexes.

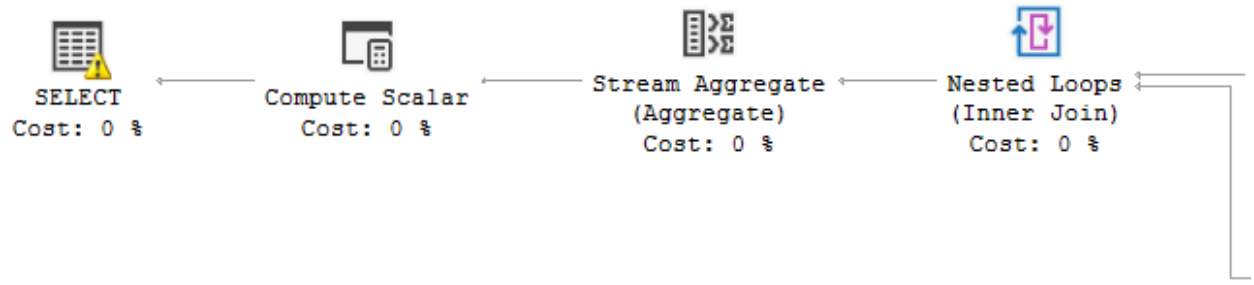
LAPTOP-N4N80KJ6 (14.0 RTM) | LAPTOP-N4N80KJ6\Yevge ... | FloridaHousingDB | 00:00:00 | 3 rows

With the indexes in place, the test query now runs in 0 seconds as opposed to the 7 seconds it took to run prior to creating the indexes.

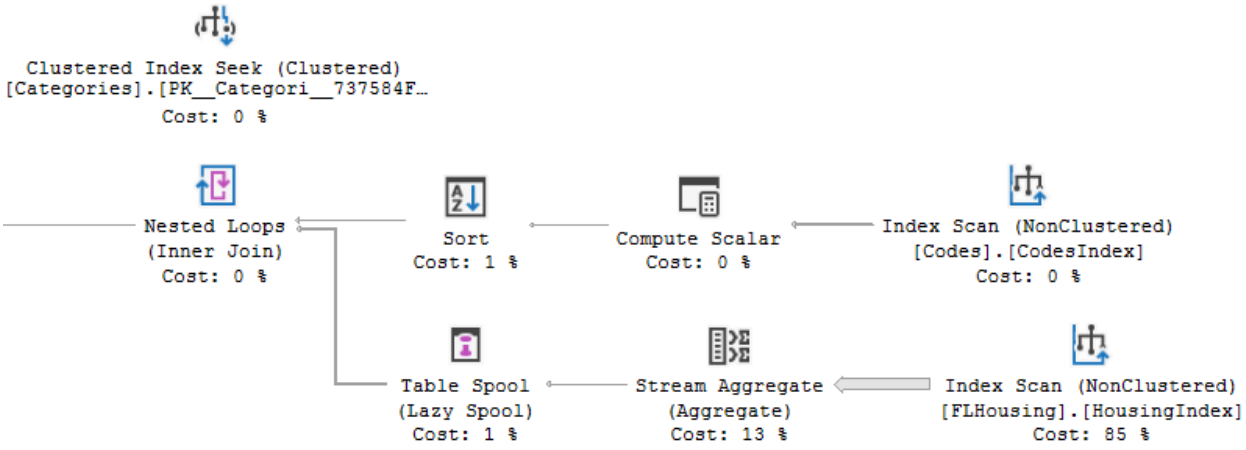
Execution Plan With Indexes - Full



Execution Plan – Left Side



Execution Plan – Right Side



Index Scan

Index Scan (NonClustered)

Scan a nonclustered index, entirely or only a range.

Physical Operation	Index Scan
Logical Operation	Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	273060
Actual Number of Rows	273060
Actual Number of Batches	0
Estimated I/O Cost	0.754977
Estimated Operator Cost	1.0555 (85%)
Estimated CPU Cost	0.300523
Estimated Subtree Cost	1.0555
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	273060
Estimated Number of Rows to be Read	273060
Estimated Row Size	11 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	11

Object

[FloridaHousingDB].[dbo].[FLHousing].[HousingIndex]

Output List

[FloridaHousingDB].[dbo].[FLHousing].ACR

SQL Server Profiler Trace

EventClass	TextData	DatabaseID	DatabaseName	ObjectID	ObjectName	ServerName	BinaryData	SPID	StartTime
Trace Start									2019-04-13 18:19:31...
ExistingConnection	-- network protocol: LPC set quote...	16	FloridaHo...			LAPTOP-...	0X60280...	53	2019-04-13 15:21:40...
SQL:BatchStarting	SELECT @@SPID;	16	FloridaHo...			LAPTOP-...		53	2019-04-13 18:19:38...
SQL:BatchStarting	SET STATISTICS XML ON	16	FloridaHo...			LAPTOP-...		53	2019-04-13 18:19:38...
SQL:BatchStarting	SELECT [Codes].[Description], CO...	16	FloridaHo...			LAPTOP-...		53	2019-04-13 18:19:38...
SQL:BatchStarting	SET STATISTICS XML OFF	16	FloridaHo...			LAPTOP-...		53	2019-04-13 18:19:39...

Conclusion

The results gathered from this test demonstrate that the use of indexes in a database offers a significant reduction in the time it takes to run queries for that database. The test query took seven full seconds to run prior to the implementation of indexes, while it took less than one second to run once the indexes were created. The indexes also caused a dramatic shift in the execution plan for the query.

The execution plan for the test query with indexes is comprised of 12 elements, whereas the execution plan for the test query without the indexes is comprised of 17 elements. There is also a change in the cost associated with these computations, as the cost of the index scan is 85% compared to the 94% cost of the table scan.

These demonstrable differences between the table scan and index scan are indicative of a drastic increase in performance when using indexes to optimize database queries.