Case Study: Comparison Between Traditional Indexing and Columnstore Indexes in SQL Server 2012

Anas A. Amro, Mohammed Abutaha
Department of Computer and Information Technology
Palestine Polytechnic University
Palestine, Hebron
{anasamro, m_abutaha}@ppu.edu

ABSTRACT: *The SQL Server 2012 has a new feature to improve performance of data warehouse queries called Columnstore index, stores data by columns instead of by rows. This paper discusses the new type of index in some details, what is a column store, how and why improves the speed up and performance of the same queries that processing in SQL Server when it used this index, and talk a limitation and advantages of this new indexes, then the summarization and resulting is illustrated by example queries. And it will clarify many things on this subject.*

Keywords: Columnar Index, Column Store, Columnstore, Data Warehousing

Received: 12 November 2012, Revised 18 December 2012, Accepted 29 December 2012

1. Introduction

Columnstore index is one of the technology that is specialized in typical data warehousing. In this type of indexing we can transform the data warehousing by enable faster performance and speed up for some queries such as aggregating, grouping and star-join queries, and filtering [1] [9]. *"This new index, combined with enhanced query optimization and execution features, improves data warehouse query performance by hundreds to thousands of times in some cases, and can routinely give a tenfold speedup for a broad range of queries fitting the scenario for which it was designed "*. [12]

2. Methodology

The authors describe a new model for indexing which supports all the same index Operations (scans, lookups, updates, and so on) that heaps and B-tree indexes support. All index types can be made functionally equivalent but they do differ in how efciently various operations can be performed.

Using Column-wise Index Storage which stores all rows as column segments. The rows are First divided into row groups of about one million rows each. Then each row group is encoded and compressed independently, producing one compressed column segment for each column included the index [2]. A Columnstore index stores each column in a separate set of disk pages, rather than storing multiple rows per page as data traditionally has been stored. We use the term "*row store*" to describe either a heap or a B-tree that contains multiple rows per page.

3. Technical Review

In this paper the authors talks about the SQL server 2012 indexing which called column store [9].

3.1 Column-wise Index Storage

Which store all rows as column segments. The rows are First divided into row groups of about one million rows each. Then each row groups is encoded and compressed independently, producing one compressed column segment for each column included in the index [3] [11], shown in Figure [1].

3.2 I/O and Caching

Column segments and dictionaries are brought into memory as needed. They are stored in a new cache designed for handling large objects. Each object in the cache is contiguously stored. This simplies and speeds up scanning of a column because there are no page breaks [3].

3.3 Batch Mode Processing

• Standard query processing in SQL Server is based on a row-at-a-time iterator model, that is, a query operator.

• Processes are almost near one thousand rows at a time. To reduce CPU time, a new set of query operators were introduced instead of process a batch of rows at a time. They greatly reduce CPU time and cache misses on modern processors when processing a large number of rows. [3]

4. Originality and Achievement

The paper made some contributions like describes a new principle indexing that increase performance and speed up.

5. Interpretation

The authors uses an example to describe his model in a perfect way.he always makes comparison between traditional indexing and his new model.

6. Evaluations and Comments

6.1 Strengths

• Using analysis and example.

• The conclusion is clearly defined, and the ideas are easy to understand.

6.2 Content

• The paper contains rich scientific article gives information well SQL server indexing.

• The content is effectively introduced and the paragraphs are presented in reasonable order and give a good impression to the reader.

• The origin of information and ideas are clearly cited and documented properly.

•Sentences are correctly constructed.

• The mechanics are correct: the format of paper is good, using capital, and abbreviations.

• The title is appropriate , informative and consistent with paper content.

7. Future enhancement

"*For reasons of scope and schedule, the implementation of column store indexes had to be scoped down in the initial release, leaving important features unsupported. These limitations will be addressed in future releases though we cannot at this stage disclose on what schedul*" [12].

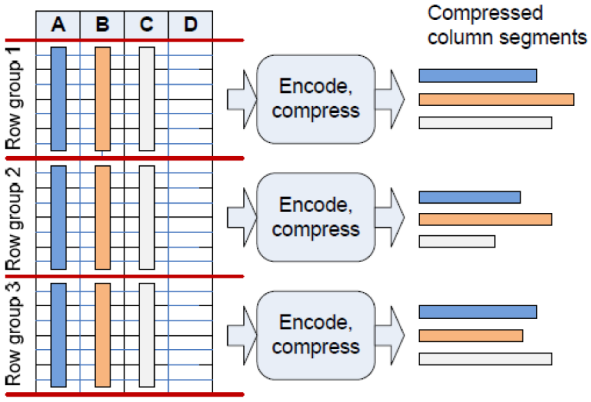

Figure 1. Converting rows to column segments

Benefits of using SQL server Columnstore index There are several benefits of using Columnstore indexes over Row Store Indexes as mentioned [4] [5].

• Faster performance for query

• Reduce storage space using compression technique

• Frequently used pages remains in memory

• Enhanced query processing/optimization and execution feature (new Batch Operator or batch mode processing) improves common data warehouse queries' performance.

7.1 Limitations of SQL Server Columnstore Indexes

There are several limitations of using SQL Server Columnstore indexes over Row Store indexes including [6] [7] [8] [11]:

• A table with a Columnstore Index cannot be updated

• Columnstore index creation takes more time (1.5 times almost) than creating a B-tree index (on same set of columns) because the data is compressed

• A table can have only one Columnstore Index and hence you should consider including all columns or at least all those frequently used columns of the table in the index

• A Columnstore Index can only be non cluster and non unique index; you cannot specify ASC/DESC or INCLUDE clauses

• Not all data types (binary, varbinary, image, text, ntext, varchar (max), nvarchar (max), etc.) are supported

• The definition of a Columnstore Index cannot be changed with the ALTER INDEX command, you need to drop and create the index or disable it then rebuild it

• You can create a Columnstore index on a table which has compression enabled, but you cannot specify the compression setting for the column store index

• A Columnstore Index cannot be created on view

• A Columnstore Index cannot be created on table which uses features like Replication, Change Tracking, Change Data Capture and Filestream

The mentioned limitation is discussed in future enhancement to be solved in the future work. A proposed model to solve update problem use table partitioning to use the ability to do fast partition switch-in operations, which allow for partitioned Columnstore indexes to be updated.

8. Experimental Results

This section presents early experimental results for database and performance improvement for two example queries. All results

were obtained on a prerelease build of SQL Server 2012. A Adventure Work Database contains a lot of tables and data, I have been using a computer within certain specifications, (*HP Intel Core i*5, *CPU* 2.30*GHz*, *RAM* 8*GB*, *HD* 500*G*, *Windows* 7), and use VMware within certain specifications (*HP Intel Core i*5, *CPU* 2.30*GHz*, *RAM* 1*G*, *HD* 100*G*, *Windows* 8) to conduct the experiment and practical application.

8.1 Example About Cost

This section shows two queries, and show the excution plan for two queries.

8.1.1 Query One

```
set statistics io on;
set statistics time on;
SELECT F.OrderDateKey,
SUM(F.SalesAmount) AS TotalSales
FROM dbo.FactInternetSales AS F
INNER JOIN dbo.DimProduct AS D1
ON F.ProductKey = D1.ProductKey
INNER JOIN dbo.DimCustomer AS D2
ON F.CustomerKey = D2.CustomerKey
GROUP BY F.OrderDateKey
with (index([AnasColumnStore])
```

In this query forced the use of Columnstore with the SQL statement *with*, and *use set statistics io, time* to show some detalis about time, input, ouput,...etc.

8.1.2 Query Two

```
SELECT F.OrderDateKey,
SUM(F.SalesAmount) AS TotalSales
FROM dbo.FactInternetSales AS F
INNER JOIN dbo.DimProduct AS D1
ON F.ProductKey = D1.ProductKey
INNER JOIN dbo.DimCustomer AS D2
ON F.CustomerKey = D2.CustomerKey
GROUP BY F.OrderDateKey
OPTION (IGNORE_NONCLUSTERED
_COLUMNSTORE_INDEX)
GO
```

In this query was to prevent the use of M using SQL statement *OPTION*), and Figure [2] shows the Excuction Plan of the two queries, and also shows the cost of two queries.

Then Excution Plan, in Figure[3] shows some detalis about costs for Columnstore index, and Figure[4] show the same detalis but for Non-Columnstore index.

8.2 Example About Time

8.2.1 Query One

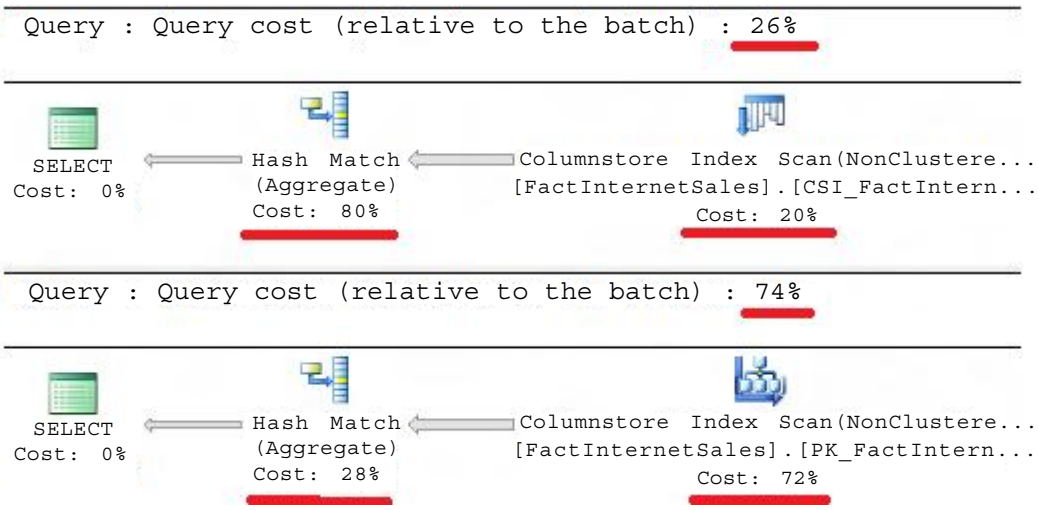

Figure 2. Excution Plans For above queries

Columnstore Index Scan (NonClustered)

Scan a columnstore index, entirely or only a range.

Physical Operation	Columnstore Index Scan
Logical Operation	Index Scan
Actual Execution Mode	Row
Storage	ColumnStore
Actual Number of Rows	60398
Actual Number of Batches	0
Estimated Operator Cost	0.0808309 (20%)
Estimated I/O Cost	0.0142361
Estimated CPU Cost	0.0665948
Estimated Subtree Cost	0.0808309
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	60398
Estimated Row Size	19 B
Actual Rebinds	0
Acctual Rewinds	0
Ordered	False
Node ID	1

Figure 3. Many Costs in Columnstore Index

```
SELECT *
FROM [dbo].[FactInternetSales] AS F
INNER JOIN dbo.DimProduct AS D1
ON F.ProductKey = D1.ProductKey
INNER JOIN dbo.DimCustomer AS D2
ON F.CustomerKey = D2.CustomerKey
```

• Elapsed Time = 00:00:03:250

• CPU Time = 16 ms

8.2.2 Query Two

```
SELECT *
FROM [FactInternetSales]AS F
INNER JOIN dbo.DimProduct AS D1
ON F.ProductKey = D1.ProductKey
INNER JOIN dbo.DimCustomer AS D2
ON F.CustomerKey = D2.CustomerKey
OPTION (IGNORE_NONCLUSTERED
_COLUMNSTORE_INDEX)
GO
```

• Elapsed Time = 00:00:42:094

• CPU Time = 343 ms

Columnstore Index Scan (Clustered)

Scanning a Clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows	60398
Actual Number of Batches	0
Estimated I/O Cost	0.761644
Estimated Operator Cost	0.828238 (72%)
Estimated CPU Cost	0.0665948
Estimated Subtree Cost	0.828238
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	60398
Estimated Row Size	19 B
Actual Rebinds	0
Acctual Rewinds	0
Ordered	False
Node ID	1

Figure 4. Many Costs in Non-Columnstore Index

The query was run twice, and only the second run was measured, these runtimes were observed above, and below points shown improvment:

• Elapsed Time = 14*X*

• CPU Time = 21*X*

9. Acknowledgements

We would like to thank our parents, who always have our support to get to the best.We also thank all the people who have helped us and our support.we must thank most especially to **Microsoft Team** for their advice and continuing support for this paper.

10. Conclusions

Column Store Index is a new feature in SQL Server 2012 that improves performance of data warehouse .Column store index cannot has a problem in update and view.

References

[1] IQ Columnar database, http://www.sybase.com/products/datawarehousing/sybaseiq

[2] Abadi, D. J., Madden, S. R., Ferreira, M. Integrating compression and execution in columnoriented database

[3] Systems. SIGMOD, (2006), 671-682. P.-A . Larson, C. Clinciu, E. N. Hanson, A. Oks, S. L. Price, S. Rangarajan, A. Surna, and Q. Zhou. SQL server column store indexes. *In*: SIGMOD Conference, p. 11771184.

[4] Columnstore Architecture. (n.d.). Retrieved 12 15, 2012, from TechBubble: http://www.techbubbles.com/sql-server/columnstoreindex- in-sql-server-2012/

[5] Index feature in SQL Server (2012). (n.d.). Retrieved 12 15, 2012, from MSSQLTips: http://www.mssqltips.com/sqlservertip/2666/columnstoreindex- feature-in-sql-server-2012/

[6] Columnstore Indexes. (n.d.). Retrieved 12 15, 2012, from msdn: http://msdn.microsoft.com/enus/ library/gg492088.aspx

[7] Columnstore Indexes: A New Feature in SQL Server known as Project Apollo. (n.d.). Retrieved 12 15, 2012, from Technet: http://blogs.technet.com/b/dataplatforminsider/archive /2011/08/04/columnstore-indexes-a-new-feature-insql- server-known-as-project-apollo.aspx

[8] INSIDE THE SQL S ERVER 2012 COLUMNSTORE INDEXES. (n.d.). Retrieved 12 15, (2012), from RUSANU CONSULTING LLC: http://rusanu.com/2012/05/29/inside-the-sql-server- 2012-columnstore-indexes/

[9] SQL SERVER Fundamentals of Columnstore Index. (n.d.). Retrieved 12 15, (2012), from SQL Authority: http://blog.sqlauthority.com/2011/10/29/sqlserver- fundamentals-of-columnstore-index/

[10] Understanding New Column Store Index of SQL Server 2012. (n.d.). Retrieved 12 15, 2012, from Database Journal: http://www.databasejournal.com/features/mssql/understandingnew- column-store-index-of-sql-server-2012.html

[11] Retrieved 12 15, (2012), from Microsoft SQL Server: http://download.microsoft.com/download/8/C/1/8C1CE06BDE2F- 40D1-9C5C-3EE521C25CE9/Columnstore Indexes for Fast DW QP SQL Server 11.pdf

[12] Larson, Per-Ake, Others. (2012). Columnar Storage in SQL Server 2012, sites.computer.org/debull/A12mar/apollo.pdf