Fundamentals of Columnstore Index

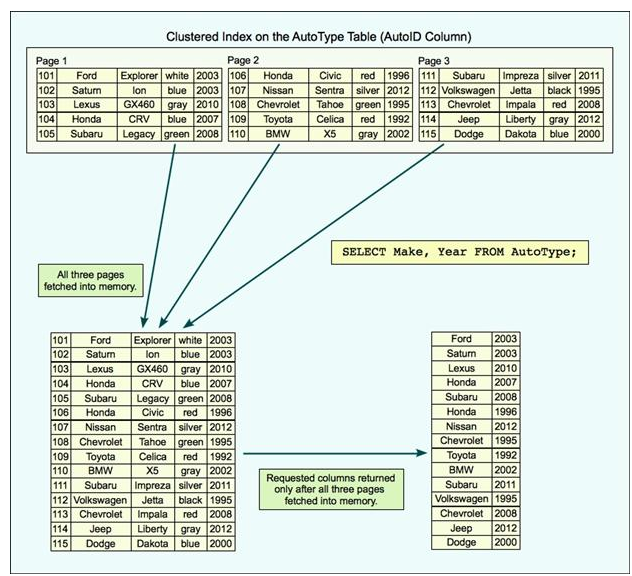
There are two kind of storage in database. Row Store and Column Store. Row store does exactly as the name suggests – stores rows of data on a page – and column store stores all the data in a column on the same page. These columns are much easier to search – instead of a query searching all the data in an entire row whether the data is relevant or not, column store queries need only to search much lesser number of the columns. This means major increases in search speed and hard drive use. Additionally, the column store indexes are heavily compressed, which translates to even greater memory and faster searches. I am sure this looks very exciting and it does not mean that you convert every single index from row store to columnstore index. One has to understand the proper places where to use row store or column store indexes. Let us understand in this article what is the difference in Columnstore type of index.

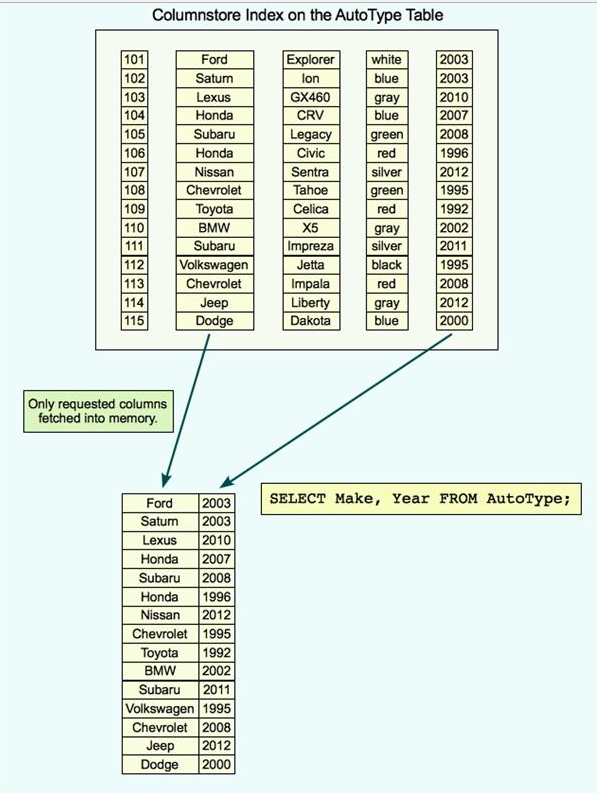
Column store indexes are run by Microsoft’s VertiPaq technology. However, all you really need to know is that this method of storing data is columns on a single page is much faster and more efficient. Creating a column store index is very easy, and you don’t have to learn a new syntax to create them. You just need to specify the keyword “COLUMNSTORE” and enter the data as you normally would.

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. The difference between column store and row store approaches is illustrated below:

In case of the row store indexes multiple pages will contain multiple rows of the columns spanning across multiple pages. In case of column store indexes multiple pages will contain multiple single columns. This will lead only the columns needed to solve a query will be fetched from disk. Additionally there is good chance that there will be redundant data in a single column which will further help to compress the data, this will have positive effect on buffer hit rate as most of the data will be in memory and due to same it will not need to be retrieved.

Let us see small example of how columnstore index improves the performance of the query on a large table.



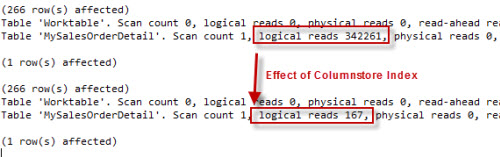


As a first step let us create databaseset which is large enough to show performance impact of columnstore index. The time taken to create sample database may vary on different computer based on the resources.

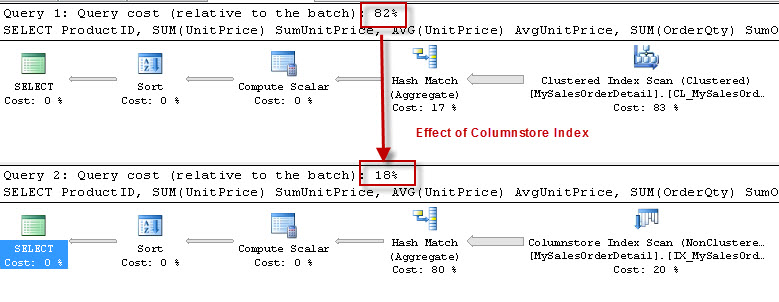
|  |  |
| --- | --- |
|  | USE AdventureWorks  GO  -- Create New Table  CREATE TABLE[dbo].[MySalesOrderDetail](  [SalesOrderID] [int] NOT NULL,  [SalesOrderDetailID] [int] NOT NULL,  [CarrierTrackingNumber] [nvarchar](25) NULL,  [OrderQty] [smallint] NOT NULL,  [ProductID] [int] NOT NULL,  [SpecialOfferID] [int] NOT NULL,  [UnitPrice] [money] NOT NULL,  [UnitPriceDiscount] [money] NOT NULL,  [LineTotal] [numeric](38, 6) NOT NULL,  [rowguid] [uniqueidentifier] NOT NULL,  [ModifiedDate] [datetime] NOT NULL  ) ON[PRIMARY]  GO  -- Create clustered index  CREATE CLUSTERED INDEX[CL\_MySalesOrderDetail] ON[dbo].[MySalesOrderDetail]  ( [SalesOrderDetailID])  GO  -- Create Sample Data Table  -- WARNING: This Query may run upto 2-10 minutes based on your systems resources  INSERT INTO[dbo].[MySalesOrderDetail]  SELECT S1.\*  FROM Sales.SalesOrderDetail S1  GO 100 |

Now let us do a quick performance test. I have kept STATISTICS IO ON for measuring how much IO following queries take. In my test first I will run query which will use regular index. We will note the IO usage of the query. After that we will create columnstore index and will measure the IO of the same.

|  |  |
| --- | --- |
| 6 | -- Performance Test  -- Comparing Regular Index with ColumnStore Index  SET STATISTICS IO ON  GO  -- Select Table with regular Index  SELECT ProductID, SUM(UnitPrice) SumUnitPrice, AVG(UnitPrice) AvgUnitPrice,  SUM(OrderQty) SumOrderQty, AVG(OrderQty) AvgOrderQty  FROM[dbo].[MySalesOrderDetail]  GROUP BY ProductID  ORDER BY ProductID  GO  -- Table 'MySalesOrderDetail'. Scan count 1, logical reads 342261, physical reads 0, read-ahead reads 0.  -- Create ColumnStore Index  CREATE NONCLUSTERED COLUMNSTORE INDEX[IX\_MySalesOrderDetail\_ColumnStore]  ON[MySalesOrderDetail]  (UnitPrice, OrderQty, ProductID)  GO  -- Select Table with Columnstore Index  SELECT ProductID, SUM(UnitPrice) SumUnitPrice, AVG(UnitPrice) AvgUnitPrice,  SUM(OrderQty) SumOrderQty, AVG(OrderQty) AvgOrderQty  FROM[dbo].[MySalesOrderDetail]  GROUP BY ProductID  ORDER BY ProductID  GO |



It is very clear from the results that query is performance extremely fast after creating ColumnStore Index. The amount of the pages it has to read to run query is drastically reduced as the column which are needed in the query are stored in the same page and query does not have to go through every single page to read those columns.



If we enable execution plan and compare we can see that column store index performance way better than regular index in **this** case.

Let us clean up the database.

|  |  |
| --- | --- |
|  | -- Cleanup  DROPINDEX[IX\_MySalesOrderDetail\_ColumnStore] ON[dbo].[MySalesOrderDetail]  GO  TRUNCATETABLEdbo.MySalesOrderDetail  GO  DROPTABLEdbo.MySalesOrderDetail  GO |