SQL Server Technical Article

Columnstore Indexes for Fast Data Warehouse Query Processing in SQL Server 11.0

Writer: Eric N. Hanson

Technical Reviewer: Susan Price

Published: November 2010

Applies to: SQL Server 11.0, code named “Denali”

Summary: The SQL Server 11.0 release (code named “Denali”) introduces a new data warehouse query acceleration feature based on a new type of index called the *columnstore*. This new index, combined with enhanced query processing 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 decision support queries. This can allow end users to get more business value from their data through fast, interactive exploration. IT workers can reduce development costs and ETL times since columnstore indexes limit or eliminate the need to rely on pre-built aggregates, including user-defined summary tables, and indexed (materialized) views. Furthermore, columnstore indexes can greatly improve ROLAP performance, making ROLAP more attractive.

Copyright

Contents

Introduction

The SQL Server 11.0 release (code named "Denali") introduces a new data warehouse query acceleration feature based on a new type of index called the *columnstore*. 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. It does all this within the familiar T-SQL query language, and the programming and system management environment of SQL Server. It's thus fully compatible with all reporting solutions that run as clients of SQL Server, including SQL Server Reporting Services.

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

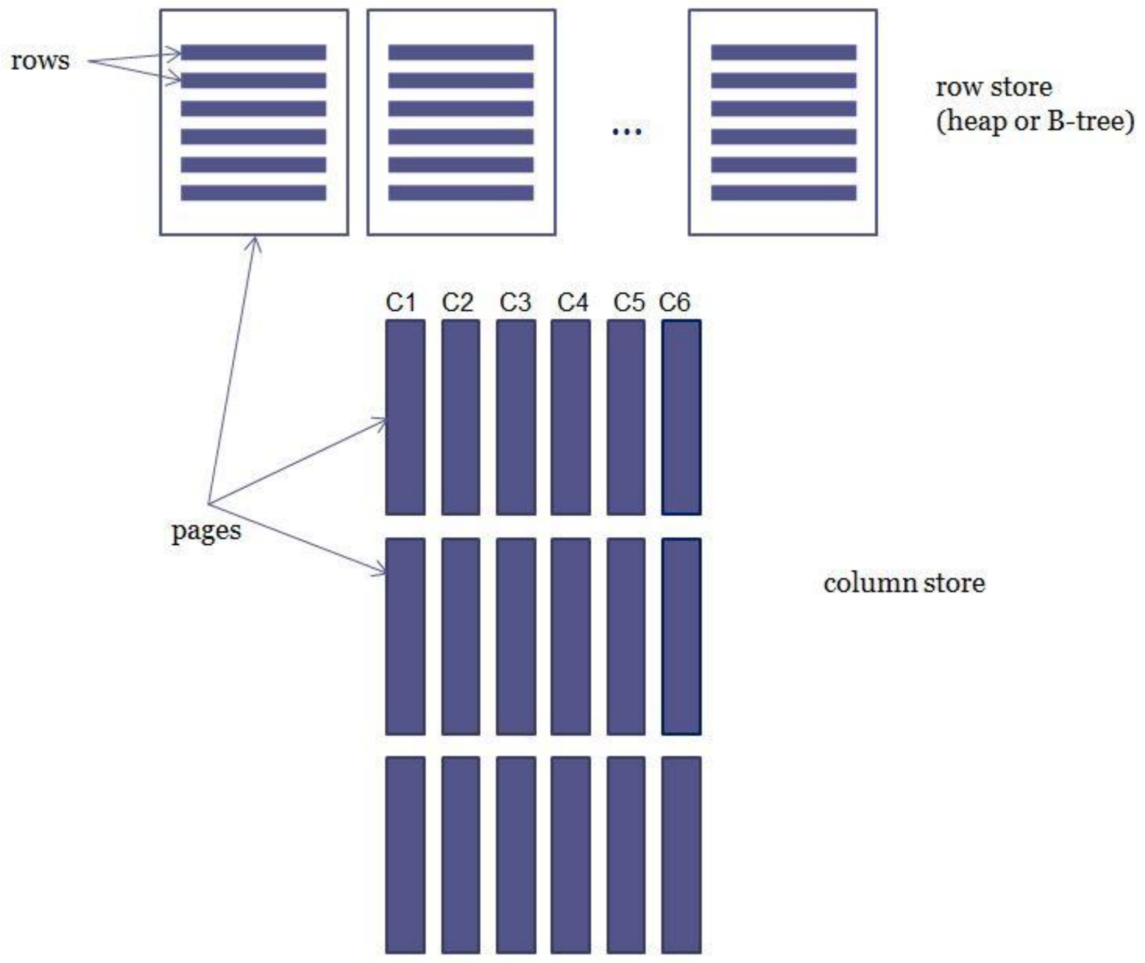

Figure 1: Comparison between row store and column store data layout

The columns C1…C6 are stored in different groups of pages in the columnstore index. Benefits of this are:

- only the columns needed to solve a query are fetched from disk (this is often fewer than 15% of the columns in a typical fact table),
- it's easier to compress the data due to the redundancy of data within a column, and
- buffer hit rates are improved because data is highly compressed, and frequently accessed parts of commonly used columns remain in memory, while infrequently used parts are paged out.

The columnstore index in SQL Server employs Microsoft's patented Vertipaq™ technology, which it shares with SQL Server Analysis Services and PowerPivot. SQL Server columnstore indexes don't have to fit in main memory, but they can effectively use as much memory as is available on the server. Portions of columns are moved in and out of memory on demand.

SQL Server Denali columnstore indexes are "pure" column stores, not a hybrid, because they store all data for separate columns on separate pages. This improves I/O scan performance and buffer hit rates. SQL Server is the first major database product to support a pure columnstore index. Others have claimed that it was impossible to leverage pure columnstore technology in an established database product with a broad market. We're happy to prove them wrong and we think you'll be glad we did!

Using Columnstore Indexes

To improve query performance, all you need to do is build a columnstore index on the fact tables in a data warehouse. If you have extremely large dimensions (say more than 10 million rows) then you may wish to build a columnstore index on those dimensions as well. After that, you simply submit queries to SQL Server, and they can run much, much faster.

For example, we created a 1 TB internal test data warehouse. The catalog_sales fact table in this database contains 1.44 billion rows. The following statement was used to create a columnstore index that includes all the columns of the table:

```
CREATE COLUMNSTORE INDEX cstore on [dbo].[catalog_sales]
           ([cs_sold_date_sk]
           ,[cs_sold_time_sk]
           ,[cs_ship_date_sk]
           ,[cs_bill_customer_sk]
           ,[cs_bill_cdemo_sk]
           ,[cs_bill_hdemo_sk]
           ,[cs_bill_addr_sk]
           ,[cs_ship_customer_sk]
           ,[cs_ship_cdemo_sk]
           ,[cs_ship_hdemo_sk]
           ,[cs_ship_addr_sk]
           ,[cs_call_center_sk]
           ,[cs_catalog_page_sk]
           ,[cs_ship_mode_sk]
           ,[cs_warehouse_sk]
           ,[cs_item_sk]
```

```
            ,[cs_promo_sk]
            ,[cs_order_number]
            ,[cs_quantity]
            ,[cs_wholesale_cost]
            ,[cs_list_price]
            ,[cs_sales_price]
            ,[cs_ext_discount_amt]
            ,[cs_ext_sales_price]
            ,[cs_ext_wholesale_cost]
            ,[cs_ext_list_price]
            ,[cs_ext_tax]
            ,[cs_coupon_amt]
            ,[cs_ext_ship_cost]
            ,[cs_net_paid]
            ,[cs_net_paid_inc_tax]
            ,[cs_net_paid_inc_ship]
            ,[cs_net_paid_inc_ship_tax]
            ,[cs_net_profit])
```

Performance Illustration

On a 32-logical processor machine with 256GB of RAM, we ran this query on a pre-release build of Denali, with and without the columnstore index on the fact table:

```
select w_city, w_state, d_year, SUM(cs_sales_price) as cs_sales_price
from warehouse, catalog_sales, date_dim
where w_warehouse_sk = cs_warehouse_sk
and cs_sold_date_sk = d_date_sk
and w_state in ('SD','OH')
and d_year in (2001,2002,2003)
group by w_city, w_state, d_year
order by d_year, w_state, w_city;
```

The query was run twice, and only the second run was measured, so that data was cached in memory to the extent that it would fit. These runtimes were observed:

	Total CPU time (seconds)	Elapsed time (seconds)
Columnstore	31.0	1.10
No columnstore	502	501
Speedup	16X	455X

This performance is truly stunning. SQL Server columnstore index technology gives near-instantaneous response time for a star join query against a 1.44 billion row table, on a relatively economical SMP system. You could go for coffee in the time it takes to run the query without the column store. This is all the more impressive because SQL Server 2008 R2 and earlier have a very efficient and competitive query processing capability for data warehousing, having introduced compression and star join query enhancements in SQL Server 2008.

In memory-constrained environments when the columnstore working set fits in RAM but the row store working set doesn't fit, it is easy to demonstrate thousand-fold speedups. When both the column store and the row store fit in RAM, the differences are smaller but are usually in the 6X to 100X range for star join queries with grouping and aggregation. Your results will of course depend on your data, workload, and hardware.

Performance Characteristics

Columnstore index query processing is most heavily optimized for star join queries, but many types of queries can benefit. Fact-to-fact table joins and multi-column join queries may benefit less from columnstore indexes, or not a tall. OLTP-style queries, including point lookups, and fetches of every column of a wide row, will usually not perform as well with a columnstore index as with a B-tree index. Columnstore indexes don't always improve data warehouse query performance. When they don't, normally, the query optimizer will choose to use a heap or B-tree to access the data. If the optimizer chooses the columnstore index when in fact using the underlying heap or B-tree performs better for a query, the developer can use hints to tune the query to use the heap or B-tree instead.

Compression Results

We've observed a factor of 4 to a factor of 15 compression with different fact tables containing real user data. The columnstore index is a secondary index; the row store is still present, though during query processing it is often not need, and ends up being paged out. A clustered columnstore index, which will be the master copy of the data, is planned for the future. This will give significant space savings in addition to the performance gains already provided.

Loading Data

In the Denali release, tables with columnstore indexes can't be updated directly using INSERT, UPDATE, DELETE, and MERGE statements, or bulk load operations. To move data into a columnstore table you can switch in a partition, or disable the columnstore index, update the table, and rebuild the index. Columnstore indexes on partitioned tables must be partition-aligned. Most data warehouse customers have a daily load cycle, and treat the data warehouse as read-only during the day, so they'll almost certainly be able to use columnstore indexes.

You can also create a view that uses UNION ALL to combine a table with a column store index and an updatable table without a columnstore index into one logical table. This view can then be referenced by queries. This allows dynamic insertion of new data into a single logical fact table while still retaining much of the performance benefit of columnstore capability.

All tables that don't have columnstore indexes remain fully updateable. This allows you to, for example, create a dimension table on the fly and then use it in successive queries by joining it to the column store-structured fact table. This can be useful, for example, when a retail analyst wants to put, say, about 1000 products into a study group, and then run repeated queries for that study group. The IDs of these products can be placed into a study group dimension table. This table can then be joined to the columnstore-structured fact table.

Index build times for a columnstore index have been observed to be 2 to 3 times longer than the time to build a clustered B-tree index on the same data, on a pre-release build. Customers will need to accommodate this time difference in their ETL processes. However, these customers typically will no longer need summary aggregates, which can take a lot of time to build, so in fact, ETL time may decrease.

Benefits of Columnstore Indexes

The primary benefit of columnstore indexes is that they can allow your users to get much more business value from their data by encouraging them to interactively explore it. The excellent performance that column stores provide makes this possible. You can get interactive response time for queries against billions of rows on an economical SMP server with enough RAM to hold your frequently accessed data.

Columnstore indexes also can reduce the burden on IT and shorten ETL time by decreasing reliance on pre-built summary aggregates, whether they are indexed views, user-defined summary tables, or OLAP cubes. Designing and maintaining aggregates is often a difficult, labor-intensive task. A single columnstore index can replace dozens of aggregates. Column stores are less brittle than aggregates because if a query is changed slightly, the columnstore can still support it, whereas a specific aggregate may no longer be useful to accelerate the query.

Users who were using OLAP systems only to get fast query performance, but who prefer to use the T-SQL language to write queries, may find they can have one less moving part in their environment, reducing cost and complexity. Users who like the sophisticated reporting tools, dimensional modeling capability, forecasting facilities, and decision-support specific query languages that OLAP tools offer can continue to benefit from them. Moreover, they may now be able to use ROLAP against a columnstore-indexed SQL Server data warehouse, and meet or exceed the performance they were used to in the past with OLAP, but save time by eliminating the cube building process.

The Source of Columnstore Performance Benefits

As discussed above, the columnstore index feature improves performance by decreasing the need to fetch data from disk to memory, due to both compression and the fact that most queries touch few columns of a table. But this is only one source of performance benefits. Multiple other proprietary techniques are employed. These methods will remain a mystery, but one thing is certain – the performance they provide is very real!

Conclusion

The columnstore index and associated query processing capabilities in SQL Server Denali are breakthrough technologies that give unheard-of performance benefits for data warehouse query processing. Your end users will be the ultimate beneficiaries; now they can get more business value from their data in much less time, using their favorite reporting tools.