C-STORE: A COLUMN-ORIENTED DBMS

Author: Mike Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O’Neil, Pat O’Neil, Alex Rasin, Nga Tran, Stan Zdonik

Presenter: Songtao Wei
What is a column-oriented DBMS?

Row-oriented systems
001: Bob, 25, Math, 10K;
002: Bill, 27, EECS, 50K;
003: Jill, 24, Biology, 80K;

Column-oriented systems
Bob: 001, Bill: 002, Jill: 003;
25: 001, 27: 002, 24: 003;
Math: 001, EECS: 002, Biology: 003;
10K: 001, 50K: 002, 80K: 003;
Why using column-oriented DBMS?

- Minimize the number of hard disk seeks
- Compress data
  
  Bob:001, Bill:002, Jill:003; 
  25:001, 27:002, 24:003; 
  Math:001, EECS:002, Math:003; 
  10K:001, 50K:002, 80K:003; 

  -> Math:001,003, EECS:002; 

- Read only the data necessary to answer the query.
Commercial products

- SAP IQ – owned by SAP

- Sensage

- Kdb+ - Owned by Kx Systems

- Vertica(Vertica Analytic Database)*

C-Store

- C-Store stores a collection of columns
- Projections: Groups of columns sorted on the same attribute.
- Three components architecture:
  - WS component optimized for frequent insert and update
  - RS component optimized for read-only query performance.
  - Tuple Mover move blocks of tuples in a WS to the corresponding RS, and updating any join indexes in the process.
- Allows redundant storage of elements of a table in several overlapping projections in different orders
- Heavily compressed columns using one of several coding schemes.
Data model

- Standard SQL semantics
- No physical tables stored using logical data model, only implement projections.
- Able to contain other table’s attributes, as long as its’ N:1 relationship (foreign key)
- the term projection is slightly different than common practice, since there is no base table stored.

EMP(name, age, salary, dept)
DEPT(dname, floor)
Data model

- the sort order of a projection by appending the sort key to the projection separated by a vertical bar.
Data model

- Covering set of projections: every column in the table is stored in at least one projection.
- Reconstructions of table using Join index and Storage key.
- Values from different columns in the same segment with matching storage keys belong to the same logical row.
- Every projection is horizontally partitioned into 1 or more segments, which are given a segment identifier, Sid, where Sid > 0.
Read-optimized Store (RS)

- Storage keys are not stored in RS, but calculated from tuple's physical position in the column.
- 4 Encoding Schemes.
- Encoding chosen for a column depends on its ordering
  - self-order: the column ordered by values in that column
  - foreign-order: the column ordered by corresponding values of some other column in the same projection
Read-optimized Store (RS)

- 4 encoding schemes
  - Self-order, few distinct values: \((v,f,n)\)
    \[
    0,0,0,1,2,2,2,3,3,3 \rightarrow (0,1,3),(1,4,1),(2,5,3),(3,8,3)
    \]
  - Foreign-order, few distinct values: bitmap \((v,b)\)
    \[
    0,0,1,1,2,1,0,2,1 \rightarrow (0, 110000100), (1, 001101001), (2, 000010010)
    \]
  - Self-order, many distinct values: represent as delta from previous value
    \[
    1,4,7,7,8,12 \rightarrow 1,3,3,0,1,4
    \]
  - Foreign-order, many distinct values: unencoded

- All use B-tree for indexing in order to minimize disk reads.
Writeable Store (WS)

- Less data compare to RS
- No need to compress for better insert and delete performance
- Identical DBMS design as RS
- Unique storage key is stored in WS segments and given to each insert of a logical tuple in a table.
- Every column in a WS projection is represented as a collection of pairs, (v, sk), such that v is a value in the column and sk is its corresponding storage key
  - Structure is represented in a conventional B-tree on sk
- The sort key(s) of each projection is additionally represented by pairs (s, sk) such that s is a sort key value and sk is the storage key describing where s first appears.
  - Structure represented as a conventional B-tree on s
Join Index and Tuple Mover

- Every projection is represented as a collection of pairs of segments, one in WS and one in RS.
- For each record in WS, need to store the sid and storage key of a corresponding record in RS.
- This data movement process is done by Tuple Mover.
Storage Management

- Allocate segments to different nodes in a grid system using a storage allocator.
- Still in plan (implemented in Vertica, section 3.6)
Read-only Transactions

- Provides Snapshot isolation
- No need locking by using HWM
- timestamp authority (TA) broadcasting timestamps to other sites.
- The time unit is epoch.
- TA has received epoch complete messages from all sites for epoch $e$, it sets the high watermark (HWM) to be $e$. 

![Diagram showing the progress of transactions across sites and the high watermark (HWM) for one epoch.](image)
Read-write transactions

- Read-write transactions use strict two-phase locking for concurrency control among each site
- Update is turned into a insert and a delete
- Using an insertion vector (IV) for each projection segment in WS that record the time (epoch) the record is inserted
- Using a deleted record vector (DRV) for each projection, which has one entry per projection record, containing a 0 if the tuple has not been deleted
- Resolve deadlock via timeouts
- Tuple Mover will choose WS segment insert time $\leq$ LWM, then separate into two groups
  - Deleted $\leq$LWM, discarded
  - Not deleted or delete after LWM, sent to RS
Query Operators and Optimization

- Operators
  - Similar to SQL operator, include Decompress, Select, Mask, Project, Sort, Aggregation Operators, Concat, Permute, Join, Bitstring Operators

- Optimization
  - Use a Selinger-style optimizer that uses cost-based estimation for plan construction
  - The major optimizer decision is which set of projections to use for a given query.
Performance

- Only test for RS
- No Segments, update, WS, and tuple mover.
- Limited to read-only queries.

- benchmarking system:
  - 3.0 Ghz Pentium
  - RedHat Linux
  - 2 Gbytes of memory
  - 750 Gbytes of disk
Storage Performance

- simplified version of TCP-H, one site
- TPC-H scale_10 totals 60,000,000 line items (1.8GB)

<table>
<thead>
<tr>
<th>C-Store</th>
<th>Row Store</th>
<th>Column Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.987 GB</td>
<td>4.480 GB</td>
<td>2.650 GB</td>
</tr>
</tbody>
</table>

- C-Store uses 40% of the space of the row store and 70% of Column Store
  - Because of the compression and no padding of word.

CREATE TABLE LINEITEM (  
  L_ORDERKEY INTEGER NOT NULL,  
  L_PARTKEY INTEGER NOT NULL,  
  L_SUPPKEY INTEGER NOT NULL,  
  L_LINENUMBER INTEGER NOT NULL,  
  L_QUANTITY INTEGER NOT NULL,  
  L_EXTENDEDPRICE INTEGER NOT NULL,  
  L_RETURNFLAG CHAR(1) NOT NULL,  
  L_SHIPDATE INTEGER NOT NULL);

CREATE TABLE ORDERS (  
  O_ORDERKEY INTEGER NOT NULL,  
  O_CUSTKEY INTEGER NOT NULL,  
  O_ORDERDATE INTEGER NOT NULL);

CREATE TABLE CUSTOMER (  
  C_CUSTKEY INTEGER NOT NULL,  
  C_NATIONKEY INTEGER NOT NULL);

C-store schema

D1: (l_orderkey, l_partkey, l_suppkey,  
     l_linenumber, l_quantity,  
     l_extendedprice, l_returnflag, l_shipdate  
     | l_shipdate, l_suppkey)

D2: (o_orderdate, l_shipdate, l_suppkey |  
     o_orderdate, l_suppkey)

D3: (o_orderdate, o_custkey, o_orderkey |  
     o_orderdate)

D4: (l_returnflag, l_extendedprice,  
     c_nationkey | l_returnflag)

D5: (c_custkey, c_nationkey | c_custkey)
Query performance

- D2 and D4 are materialized (join) views
- D3 and D5 are added for completeness since we don’t use them in any of the seven queries.

Q1. Determine the total number of lineitems shipped for each day after day D.
   
   ```sql
   SELECT l_shipdate, COUNT(*)
   FROM lineitem
   WHERE l_shipdate > D
   GROUP BY l_shipdate
   ```

Q2. Determine the total number of lineitems shipped for each supplier on day D.
   
   ```sql
   SELECT l_suppkey, COUNT(*)
   FROM lineitem
   WHERE l_shipdate = D
   GROUP BY l_suppkey
   ```

Q3. Determine the total number of lineitems shipped for each supplier after day D.
   
   ```sql
   SELECT l_suppkey, COUNT(*)
   FROM lineitem
   WHERE l_shipdate > D
   GROUP BY l_suppkey
   ```

Q4. For every day after D, determine the latest shipdate of all items ordered on that day.
   
   ```sql
   SELECT o_orderdate, MAX(l_shipdate)
   FROM lineitem, orders
   WHERE l_orderkey = o_orderkey AND
   o_orderdate > D
   GROUP BY o_orderdate
   ```

Q5. For each supplier, determine the latest shipdate of an item from an order that was made on some date, D.
   
   ```sql
   SELECT l_suppkey, MAX(l_shipdate)
   FROM lineitem, orders
   WHERE l_orderkey = o_orderkey AND
   o_orderdate = D
   GROUP BY l_suppkey
   ```

Q6. For each supplier, determine the latest shipdate of an item from an order made after some date, D.
   
   ```sql
   SELECT l_suppkey, MAX(l_shipdate)
   FROM lineitem, orders
   WHERE l_orderkey = o_orderkey AND
   o_orderdate > D
   GROUP BY l_suppkey
   ```

Q7. Return a list of identifiers for all nations represented by customers along with their total lost revenue for the parts they have returned. This is a simplified version of query 10 (Q10) of TPC-H.
   
   ```sql
   SELECT c_nationkey, sum(l_extendedprice)
   FROM lineitem, orders, customers
   WHERE l_orderkey=o_orderkey AND
   o_custkey=c_custkey AND
   l_returnflag='R'
   GROUP BY c_nationkey
   ```
Query performance

- Column representation
- Storing overlapping projections, not the whole table
- Better compression of data
- Query operators operate on compressed representation

<table>
<thead>
<tr>
<th>Query</th>
<th>C-Store</th>
<th>Row Store</th>
<th>Column Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>0.03</td>
<td>6.80</td>
<td>2.24</td>
</tr>
<tr>
<td>Q2</td>
<td>0.36</td>
<td>1.09</td>
<td>0.83</td>
</tr>
<tr>
<td>Q3</td>
<td>4.90</td>
<td>93.26</td>
<td>29.54</td>
</tr>
<tr>
<td>Q4</td>
<td>2.09</td>
<td>722.90</td>
<td>22.23</td>
</tr>
<tr>
<td>Q5</td>
<td>0.31</td>
<td>116.56</td>
<td>0.93</td>
</tr>
<tr>
<td>Q6</td>
<td>8.50</td>
<td>652.90</td>
<td>32.83</td>
</tr>
<tr>
<td>Q7</td>
<td>2.54</td>
<td>265.80</td>
<td>33.24</td>
</tr>
</tbody>
</table>
Query performance

Add materialized views that correspond to the projections used with C-store.

Performance catch up but with a cost of consuming too much storage.

<table>
<thead>
<tr>
<th>C-Store</th>
<th>Row Store</th>
<th>Column Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.987 GB</td>
<td>4.480 GB</td>
<td>2.650 GB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query</th>
<th>C-Store</th>
<th>Row Store</th>
<th>Column Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>0.03</td>
<td>6.80</td>
<td>2.24</td>
</tr>
<tr>
<td>Q2</td>
<td>0.36</td>
<td>1.09</td>
<td>0.83</td>
</tr>
<tr>
<td>Q3</td>
<td>4.90</td>
<td>93.26</td>
<td>29.54</td>
</tr>
<tr>
<td>Q4</td>
<td>2.09</td>
<td>722.90</td>
<td>22.23</td>
</tr>
<tr>
<td>Q5</td>
<td>0.31</td>
<td>116.56</td>
<td>0.93</td>
</tr>
<tr>
<td>Q6</td>
<td>8.50</td>
<td>652.90</td>
<td>32.83</td>
</tr>
<tr>
<td>Q7</td>
<td>2.54</td>
<td>265.80</td>
<td>33.24</td>
</tr>
</tbody>
</table>
Query performance

Space-constrained case:
- 164 times faster than the commercial row-store
- 21 times faster than the commercial columnstore

Without space limitation:
- 6.4 times faster than the commercial row-store,
  - row-store takes 6 times the space.
- 16.5 times faster than the commercial column-store,
  - column-store requires 1.83 times the space.
Advantages

- Efficient on space usage
- Fast speed compare
- Vertica further reduced the size of data.
- Vertica using software engineering methods such as vectorized execution and more complex compression algorithms in order to achieve twice speed as C-store on a single core machine

Disadvantages

- Single threaded (massively parallel processing hardware supported in Vertica)
- Need to design schema for the query in order to get best speed result.
- Join index is hard to design and maintenance is very expensive. (Replace by one or more “super projection” containing every column of the anchoring table in Vertica)
- Only support integer data type. (more datatype support added in Vertica such as FLOAT and VARCHAR)
- Not able to process SQL NULLs. (support in Vertica)
Conclusion

- C-store is a prototype, but still have good performance on typical situation.
- The Vertica take the main idea of C-store and implement it further more on data type support, speed and compression size.
Reference
