Lecture 10: Query Processing

15-445/645 Database Systems (Fall 2017)
Carnegie Mellon University
Prof. Andy Pavlo

Query Plan

1. Operators are arranged in a tree. Data flows from the leaves towards the root
2. The out put of the root node is the result of the query
3. Typically operators are binary (1-2 children)

Processing Model

1. A DBMS **processing model** defines how the system executes a query plan (different trade-offs for different workloads)

Iterator Model

1. Most common processing model, used by almost every DBMS
2. Top-Down Processing
3. **Every query plan operator implements a next function**
   (a) On each call to next, the operator returns either a single tuple or a null marker if there are no more tuples
   (b) The operator implements a loop that calls next on its children to retrieve their tuples ad then process them (calling next on a parent calls next on their children)
4. Allows for tuple pipelining
5. Some operators will block until children emit all of their tuples (joins, subqueries, order bys)
6. Output control works easily with this approach (LIMIT)

Materialization Model

1. Each operator processes its input all at once and then emits its output all at once
2. The operator ”materializes” its output as a single result

3. Bottom-Up processing
4. Better for OLTP workloads because queries typically only access a small number of tuples at a time
5. Lower execution/coordination overhead, but more difficult to parallelize
6. Not good for OLAP queries with large intermediate results

Vectorization Model

1. Like the iterator model, each operator implements a next function
2. Each operator emits a **batch** of data instead of a single tuple
3. Ideal for OLAP queries
4. Greatly reduces the number of invocations per operator

Access Methods

1. How the DBMS accesses the data stored in a table
2. Three basic approaches
   (a) Sequential Scan
   (b) Index Scan
   (c) Multi-index / "bitmap" scan

Sequential Scan

1. For each page in table, retrieve from buffer loop
2. For each page, iterate over all the tuples and evaluate the predicate to decide whether to include tuple or not
3. This is a fall-back method, it's slow, but safe
4. **Optimizations**
   (a) Prefetching: Fetches next few pages in advance
   (b) Parallelization: parallelize the scan
   (c) Zone map: Precomputed aggregate for the attribute values in a page
   (d) Buffer pool bypass: Scan operator will store pages fetched from disk in local memory instead of the buffer pool
   (e) Heap clustering: Tuples are stored in the heap pages using an order specified by a clustering index

Index Scan

1. The DBMS picks an index (or indexes) to find the tuples that the query needs
2. Commonly done on B+Tree because hash indexes aren't in sorted order
3. Given multiple indexes, the DBMS will pick the best index to use based on the data the indexes hold
4. Retrieving tuples in the order that they appear in an unclustered index is inefficient. The DBMS can first figure out all the tuples that it needs and then sort them based on their page id

Expression Evaluation

1. The DBMS represents a WHERE clause as an expression tree
2. The nodes in the tree represent different expression types
3. During evaluation, you store a context containing meta data for the execution, such as the current tuple, the parameters, and the table schema
4. Evaluating predicates in this manner is slow

Conclusion

1. The same query plan can be executed in multiple ways
2. (Most) DBMS will want to use an index scan as much as possible
3. Expression trees are flexible but slow