**International Journal of Advances in Engineering Research** http://www.ijaer.com

(IJAER) 2012, Vol. No. 4, Issue No. VI, December **ISSN: 2231-5152**

# ANALYSIS OF QUERY EVALUATION TECHNIQUES FOR LARGE DATABASES

**JADHAO BHARATI RAMRAO, KADAM KIRAN PRABHAKAR**
*RESEARCH SCHOLARS, DEPT. OF COMPUTER SCIENCE*
*CMJ UNIVERSITY, SHILLONG, MEGHALAYA*

## ABSTRACT

*Database management systems will continue to manage large data volumes. Thus, efficient algorithms for accessing and manipulating large sets and sequences will be required to provide acceptable performance. The advent of object-oriented and extensible database systems will not solve this problem. On the contrary, modern data models exacerbate it: In order to manipulate large sets of complex objects as efficiently as today's database systems manipulate simple records, query processing algorithms and software will become more complex, and a solid understanding of algorithm and architectural issues is essential for the designer of database management software.*

*This survey provides a foundation for the design and implementation of query execution facilities in new database management systems. It describes a wide array of practical query evaluation techniques for both relational and post-relational database systems, including iterative execution of complex query evaluation plans, the duality of sort- and hash-based set matching algorithms, types of parallel query execution and their implementation, and special operators for emerging database application domains.*

## INTRODUCTION

Effective and efficient management of large data volumes is necessary in virtually all computer applications, from business data processing to library information retrieval systems, multimedia applications with images and sound, computer-aided design and manufacturing, real-time process control, and scientific computation. While database management systems are standard tools in business data processing, they are only slowly being introduced to all the other emerging database application areas.

In most of these new application domains, database management systems have traditionally not been used for two reasons. First, restrictive data definition and manipulation languages can make application development and maintenance unbearably cumbersome. Research into semantic and object-oriented data models and into persistent database programming languages has been addressing this problem and will eventually lead to acceptable solutions. Second, data volumes might be so large or complex that the real or perceived performance advantage of file systems is considered more important than all other criteria, e.g., the higher levels of abstraction and programmer productivity typically achieved with database management systems. Thus,

object-oriented database management systems that are designed for non-traditional database application domains and extensible database management systems toolkits that support a variety of data models must provide excellent performance to meet the challenges of very large data volumes, and techniques for manipulating large data sets will find renewed and increased interest in the database community.

The purpose of this paper is to survey efficient algorithms and software architectures of database query execution engines for executing complex queries over large databases. A "complex" query is one that requires a number of query processing algorithms to work together, and a "large" database uses files with sizes from several megabytes to many terabytes, which are typical for database applications at present and in the near future [Dozier 1992; Silberschatz, Stonebraker, and Ullman 1991]. This survey discusses a large variety of query execution techniques that must be considered when designing and implementing the query execution module of a new database management system: algorithms and their execution costs, sorting vs. hashing, parallelism, resource allocation and scheduling issues in complex queries, special

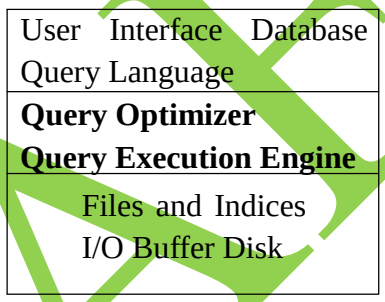


Figure 1. Query Processing in a Database System.

operations for emerging database application domains such as statistical and scientific databases, and general performance-enhancing techniques such as precomputation and compression. While many, although not all, techniques discussed in this paper have been developed in the context of relational database systems, most of them are applicable to and useful in the query processing facility for any database management system and any data model, provided the data model permits queries over "bulk" data types such as sets and lists.

## SORTING AND HASHING

The purpose of many query processing algorithms is to perform some kind of matching, i.e., bringing items that are "alike" together and performing some operation on them. There are two basic approaches for such "value-based" operations, one using pre-computed data structures such

as indices and the other using large amounts of memory. The memory-based algorithms can be divided further into those based on sorting and those based on hashing. This pair, sorting and hashing, permeates many aspects of query processing, from indexing and clustering over aggregation and join algorithms to methods for parallelizing database operations. Therefore, we first discuss these approaches first in general terms, without regard to specific algorithms. The subsequent sections survey specific algorithms for unary (grouping, aggregation, duplicate removal) and binary (join, semi-join, intersection, division, etc.) matching, followed by a review of the duality between sort- and hash-based query processing algorithms.

Both sort- and hash-based query processing algorithms are memory-intensive. Their memory allocation should exceed the size of their input data for best performance. Moreover, the allocated memory should be physical memory, not virtual memory that can be swapped out by the operating system. While virtual memory is heavily relied on in most computer systems because it offers an easy way to run algorithms on large inputs, it typically does not provide optimal database performance. All memory-intensive algorithms have variants that use temporary disk files to cope with limited amounts of memory, i.e., less memory than the size of the input data. Therefore, a choice must be made for large inputs whether to run an algorithm's in-memory version in virtual memory or to run an algorithm's version that uses explicit I/O. Since operating systems and virtual memory implementations are general-purpose software that must function reasonably well for all applications and memory access patterns, not only for database query processing, algorithm variants using explicit I/O operations on temporary files typically outperform their in-memory equivalents using virtual memory. One of the specific reasons is that explicit I/O can be planned, which permits moving larger amounts of useful data in a single operation between memory and disk and enables read-ahead for overlap of CPU processing and I/O. Virtual memory can write data to disk asynchronously, but it must rely on page faults to detect which pages to read back.

**Sorting**

Sorting is used very frequently in database systems, both for presentation to the user in sorted reports or listings and for query processing in sort-based algorithms such as merge-join. Therefore, the performance effects of the many algorithmic tricks and variants of external sorting deserve detailed discussion in this survey. All sorting algorithms actually used in database systems use merging, i.e., the input data are written into initial sorted runs and then merged into larger and larger runs until only one run is left, the sorted output. Only in the unusual case that a data set is smaller than the available memory can in-memory techniques such as quicksort be used. An excellent reference for many of the issues discussed here is Knuth [Knuth 1973], who analyzes algorithms much more accurately than we do in this introductory survey.

### Hashing

For many matching tasks, hashing is an alternative to sorting. In general, when equality matching is required, hashing should be considered because the expected complexity of set algorithms based on hashing is $O(N)$ rather than $O(N \log N)$ as for sorting. Of course, this makes intuitive sense if hashing is viewed as radix sorting on a virtual key [Knuth 1973].

Hash-based query processing algorithms use an in-memory hash table of database objects to perform their matching task. If the entire hash table (including all records or items) fits into memory, hash-based query processing algorithms are very easy to design, understand, and implement, and outperform sort-based alternatives. Note that for binary matching operations, such as join or intersection, only one of the two inputs must fit into memory. However, if the required hash table is larger than memory, *hash table overflow* occurs and must be dealt with.

## DISK ACCESS

All query evaluation systems have to access base data stored in the database. For databases in the megabyte to terabyte range, base data are typically stored on secondary storage in form of rotating random-access disks. However, deeper storage hierarchies including optical storage, (maybe robot-operated) tape archives, and remote storage servers will also have to be considered in future high-functionality high-volume database management systems, [Carey, Haas, and Livny 1993; Stonebraker 1991]. Research into database systems supporting and exploiting deep storage hierarchies is still in its infancy.

On the other hand, motivated both by the desire for faster transaction and query processing performance and by the decreasing cost of semi-conductor memory, some researchers have considered in-memory or main memory databases, e.g., [Analyti and Pramanik 1992; Bitton, Hanrahan, and Turbyfill 1987; Bucheral, Theverin, and Valduriez 1990; DeWitt et al. 1984; Gruenwald and Eich 1991; Kumar and Burger 1991; Lehman and Carey 1986; Li and Naughton 1988; Severance, Pramanik, and Wolberg 1990; Whang and Krishnamurthy 1990]. However, for most appications, an analysis by Gray and Putzolo demonstrated that main memory is cost-effective only for the most frequently accessed data [Gray and Putzolo 1987]. The time interval between accesses with equal disk and memory costs was five minutes for their values of memory and disk prices. Only data accessed at least every five minutes should be kept in memory, whereas other data should reside on disks. Although this threshold time interval is expected to grow as main memory prices decrease faster than disk prices, there will always be substantial data volumes that are accessed less frequently than the threshold but that must be queried efficiently. Therefore, for the purposes of this survey, we presume a disk-based storage architecture and consider disk I/O one of the major costs of query evaluation over large databases.

## AGGREGATION AND DUPLICATE REMOVAL

Aggregation is a very important statistical concept to summarize information about large amounts of data. The idea is to represent a set of items by a single value or to classify items into groups and determine one value per group. Most database systems support aggregate functions for minimum, maximum, sum, count, and average (arithmetic mean). Other aggregates, e.g., geometric mean or standard deviation, are typically not provided, but may be constructed in some systems with extensibility features. Aggregation has been added to both relational calculus and algebra and adds the same expressive power to each of them [Klug 1982].

## DUALITY OF SORT- AND HASH-BASED QUERY PROCESSING ALGORITHMS

In this section, we conclude the discussion of individual query processing by outlining the

| Aspect | Sorting | Hashing |
|---|---|---|
| In-memory algorithm | Quicksort | Classic Hash |
| Divide-and-conquer paradigm | Physical division, logical combination | Logical division, physical combination |
| Large inputs | Single-level merge | Partitioning |
| I/O Patterns | Sequential write, random read | Random write, sequential read |
| Temporary files accessed simultaneously | Fan-in $F \approx M / C$ | Fan-out $F \approx M / C$ |
| I/O Optimizations | Read-ahead, forecasting | Write-behind |
| | Double-buffering input or output | Double-buffering output or input |
| | Striping merge output | Striping partitioning input |
| Very large inputs | Multi-level merge | Recursive partitioning |
| | Merge levels | Recursion depth |
| | Non-optimal final fan-in | Non-optimal hash table size |
| Optimizations | Merge optimizations | Bucket tuning |
| Better use of memory | Reverse runs & LRU | Hybrid hashing |
| | Replacement selection | ? |
| | ? | Single input in memory |
| Aggregation and duplicate removal | Aggregation in replacement selection | Aggregation in hash table |

Table a. Duality of Sort- and Hash-Based Algorithms.

many existing similarities and dualities of sort- and hash-based query processing algorithms as well as the points where the two types of algorithms differ. The purpose is to contribute to a better understanding of the two approaches and their tradeoffs. We try to discuss the approaches in general terms, ignoring whether the algorithms are used for relational join, union, intersection, aggregation, duplicate removal, or other operations. Where appropriate, however, we indicate specific operations.

## REFERENCES

Adam and Wortmann 1989: N. R. Adam and J. C. Wortmann, Security-Control Methods for Statistical Databases: A Comparative Study, *ACM Computing Surveys 21*, 4 (December 1989), 515.

Buneman, Frankel, and Nikhil 1982: P. Buneman, R. E. Frankel, and R. Nikhil, An Implementation Technique for Database Query Languages, *ACM Trans. on Database Sys. 7*, 2 (June 1982), 164.

Cacace, Ceri, and Houtsma 1993: F. Cacace, S. Ceri, and M. A. W. Houtsma, A Survey of Parallel Execution Strategies for Transitive Closure and Logic Programs, *to appear in Distr. and Parallel Databases*, 1993.

Drew, King, and Hudson 1990: P. Drew, R. King, and S. Hudson, The Performance and Utility of the Cactis Imple- mentation Algorithms, *Proc. Int'l. Conf. on Very Large Data Bases*, Brisbane, Australia, August 1990, 135.

Effelsberg and Haerder 1984: W. Effelsberg and T. Haerder, Principles of Database Buffer Management, *ACM Trans. on Database Sys. 9*, 4 (December 1984), 560.

Fushimi, Kitsuregawa, and Tanaka 1986: S. Fushimi, M. Kitsuregawa, and H. Tanaka, An Overview of the System Software of a Parallel Relational Database Machine GRACE, *Proc. Int'l. Conf. on Very Large Data Bases*, Kyoto, Japan, August 1986, 209.

Gallaire, Minker, and Nicolas 1984: H. Gallaire, J. Minker, and J. M. Nicolas, Logic and Databases: A Deductive Approach, *ACM Computing Surveys 16*, 2 (June 1984), 153.

Guttman 1984: A. Guttman, R-Trees: A Dynamic Index Structure for Spatial Searching, *Proc. ACM SIGMOD Conf.*, Boston, MA, June 1984, 47. Reprinted in M.

Stonebraker, Readings in Database Sys., Morgan- Kaufman, San Mateo, CA, 1988.

Haas et al. 1982: L. M. Haas, P. G. Selinger, E. Bertino, D. Daniels, B. Lindsay, G. Lohman, Y. Masunaga, C. Mo- han, P. Ng, P. Wilms, and R. Yost, *R*: A Research Project on Distributed Relational DBMS*, IBM Res. Divi- sion, San Jose CA, October 1982.

Jhingran 1991: A. Jhingran, Precomputation in a Complex Object Environment, *Proc. IEEE Int'l. Conf. on Data Eng.*, Kobe, Japan, April 1991, 652.

Kao 1986: S. Kao, DECIDES: An Expert System Tool for Physical Database Design, *Proc. IEEE Int'l. Conf. on Data Eng.*, Los Angeles, CA, February 1986, 671.

Lyytinen 1987: K. Lyytinen, Different Perspectives on Information Systems: Problems and Solutions, *ACM Com- puting Surveys 19*, 1 (March 1987), 5.

Mackert and Lohman 1989: L. F. Mackert and G. M. Lohman, Index Scans Using a Finite LRU Buffer: A Validated I/O Model, *ACM Trans. on Database Sys. 14*, 3 (September 1989), 401.

Nyberg et al. 1993: C. Nyberg, T. Barclay, Z. Cvetanovic, J. Gray, and D. Lomet, AlphaSort: A RISC Machine Sort, *Digital Equipment Corp. San Francisco Systems Center Tech. Rep. 93.2*, April 1993.

Olken and Rotem 1989: F. Olken and D. Rotem, Rearranging Data to Maximize the Efficiency of Compression, *J. of Computer and System Sciences 38*, 2 (1989), 405.