Query Processing

Query processing refers to the range of activities involved in extracting data from a database. The activities include translation of queries in high-level database languages into expressions that can be used at the physical level of the file system, a variety of query-optimizing transformations, and actual evaluation of queries.

Bibliographical Notes

[Graefe (1993)] presents an excellent survey of query-evaluation techniques. [Faerber et al. (2017)] describe main-memory database implementation techniques, including query processing techniques for main-memory databases, while [Kemper et al. (2012)] describes techniques for query processing with in-memory columnar data. [Samet (2006)] provides a textbook description of spatial data structures, while [Shekhar and Chawla (2003)] provides a textbook description of spatial databases, including indexing and query processing techniques. Textbook descriptions of techniques for indexing documents, and efficiently computing ranked answers to keyword queries may be found in [Manning et al. (2008)].

A query processor must parse statements in the query language, and must translate them into an internal form. Parsing of query languages differs little from parsing of traditional programming languages. Most compiler texts cover the main parsing techniques, and present optimization from a programming-language point of view.

[Knuth (1973)] presents an excellent description of external sorting algorithms, including an optimization called replacement selection, which can create initial runs that are (on the average) twice the size of memory. [Nyberg et al. (1995)] shows that due to poor processor-cache behavior, replacement selection performs worse than in-memory quicksort for run generation, negating the benefits of generating longer runs. [Nyberg et al. (1995)] presents an efficient external sorting algorithm that takes processor cache effects into account. Query evaluation algorithms that take cache effects into account have been extensively studied; see, for example, [Harizopoulos and Ailamaki (2004)].

According to performance studies conducted in the mid-1970s, database systems of that period used only nested-loop join and merge join. These studies, including [Blas-
gen and Eswaran (1976)], which was related to the development of System R, determined that either the nested-loop join or merge join nearly always provided the optimal join method. Hence, these two were the only join algorithms implemented in System R. However, [Blasgen and Eswaran (1976)] did not include an analysis of hash-join algorithms. Today, hash joins are considered to be highly efficient and widely used.

Hash-join algorithms were initially developed for parallel database systems. Hybrid hash join is described in [Shapiro (1986)]. [Zeller and Gray (1990)] and [Davison and Graefe (1994)] describe hash-join techniques that can adapt to the available memory, which is important in systems where multiple queries may be running at the same time. [Graefe et al. (1998)] describes the use of hash joins and hash teams, which allow pipelining of hash joins by using the same partitioning for all hash joins in a pipeline sequence, in the Microsoft SQL Server.


Bibliography


Further Reading


