Options in Physical Database Design

Goetz Graefe
Portland State University, Computer Science Department
P.O. Box 751, Portland, Oregon 97207–0751

Abstract

A cornerstone of modern database systems is physical data independence, i.e., the separation of a type and its associated operations from its physical representation in memory and on storage media. Users manipulate and query data at the logical level; the DBMS translates these logical operations to operations on files, indices, records, and disks. The efficiency of these physical operations depends very much on the choice of data representations.

Choosing a physical representation for a logical database is called physical database design. The number of possible choices in physical database design is very large; moreover, they very often interact with each other. We attempt to list and classify these choices and to explore their interactions. The purpose of this paper is to provide an overview of possible options to the DBMS developer and some guidance to the DBMS administrator and user.

While much of our discussion will draw on the relational data model, physical database design is of even more importance for object-oriented and extensible systems. The reasons are simple: First, the number of logical data types and their operations is larger, requiring and permitting more choices for their representation. Second, the state of the art in query optimization for these systems is much less developed than for relational systems, making careful physical database design even more imperative for object-oriented database systems.

1. Introduction

The most significant contributions of database research to computer science have probably been the transaction concept, non-procedural query languages and their effective optimization into efficient and parallel execution plans, and physical data independence. In the last two decades, formal specifications and efficient implementations have been developed for each of these three concepts. In the present survey, we focus on physical data independence.

The foundation of physical data independence is a separation of conceptual data and their physical representation. On the logical level, data are encapsulated in (abstract) data types with a well-defined set of operations. All instances of the same type support the same operations and exhibit, to the outside, exactly the same behavior. On the inside, however, i.e., on the level of physical representations, it is entirely possible that two instances of the same type be represented differently, as long as the different representations support the same behavior to the outside world. Reasons for different internal implementations are typically time and space efficiencies combined with an instance's size and its frequency and modes of retrieval and update. Of course, this separation and physical data independence are very closely related to the separation of interface specification and implementation in abstract data types.

Let us consider some very concrete examples. In the relational data model, a relation is a set of tuples. Notice that it is a set, not a sequence or list: thus, there is no inherent order. In the storage representation and in processing, however, order can make a big difference in query processing performance and therefore is an important physical database design issue. Partitioning or striping a relation over multiple disks as well as indexing are relevant and important performance techniques that also have no place in the relational data model, since it is a logical or conceptual data model. Moreover, tuple representations are not specified in the model: a tuple can be a record, it can be divided into multiple records (e.g., fixed-length and variable-length fields), it can be represented by pointers into files representing the underlying domains, etc. In the object-oriented database world, objects are individuals with identity and state — none of the object-oriented data models specifies physical issues such as object clustering or caching in workstation-server architectures, although these issues clearly can have a significant performance effect. These issues are captured in the physical database design, which is a separate step following logical database design in any effort of introducing a database management system.

A database management system must support three essential capabilities in order to achieve physical data independence. First, the DBMS software must provide for alternative representation of one logical data set. Conversely, different logical data sets may employ the same physical data structure, although not necessarily the same instance of one structure. Second, updates that are expressed on the logical level must be mapped correctly to all relevant physical structures. For example, if secondary (redundant) indices or data replication are employed, a single update operation on the logical level must be reflected correctly in all files and index structures. Third, if the physical database design includes redundancy (e.g., indices or replication), a user's retrieval operation on the logical level must be mapped to an access plan that specifies which physical structures should be accessed in which order and using which algorithms. Thus, need and opportunity for query optimization in database systems comes from two directions: on the one hand, non-procedural query language statements must be mapped to procedural instructions for the query evaluator; on the other hand, alternative access paths require automatic choices among them. An alternative view of the interaction of physical database design and query optimization is this: physical database design considers the entire usage of a database and attempts to provide for the best overall system performance, whereas query optimization (including access path selection) considers the semantics of a single query[1] and tries to find the best query evaluation plan given the current state of the logical and physical database design. Thus, the design of a query optimizer should take the physical database design options of the underlying database management system into account (e.g., the concept of "interesting orderings" in System R [SAC79] and many other systems), and a physical database design tool must consider the capabilities of both the DBMS's file and index level as well as its query optimizer.

This survey is meant for both the DBMS implementor and user. For the implementor, our goal is to provide a list of possible facilities to consider in a database implementation. When expanding an existing system's set of physical database design options, the larger picture must be kept in mind. The purpose of this survey is to provide that larger picture, and to provide references to relevant literature. For the user, we hope to provide some insight into the tradeoffs that must be considered when translating a logical database design into a physical database design. When purchasing database management software, a user might want to consider which options might actually be useful for the intended application and should therefore become part of the purchasing criteria. When administering an installed database, there are often more possible changes to an unsatisfactory physical database design than a user might think of immediately.

In the following section, we present a very brief overview over the wide variety of issues in physical database design and provide a number of relevant references for each issue. A much more detailed version of this survey that elaborates on these issues and explores their interactions is in preparation.

2. A Brief Overview of Physical Database Design

In order to minimize the I/O costs in a database system, it is important that (a) the data structures on disk permit efficient retrieval of only relevant data through effective access paths, and (b) data be arranged and placed on disk such that the I/O cost for relevant data is minimized. Both of these concerns are addressed in physical database design. For a first impression of physical database design, we start with a comprehensive list of issues to be considered in physical database design; the individual topics will be discussed in more detail in the following section. There is no existing database system at this point that supports all of these choices, although all of them could have a significant performance impact. These choices are not all orthogonal; the goal at this point is to outline the scope of physical database design and its concerns.

Item Representation

Physical representation types for **abstract data types** is only slowly gaining research attention for object-oriented database systems but will likely become a very important tuning option. Examples include sets represented as bit maps, arrays, or lists and matrices

[1] There also have been some efforts at "global" query optimization, which attempts to optimize multiple queries together, e.g. [Cha91, Sel88] and their references.

represented densely or sparsely, by row or by column or as tiles, e.g. [MaV93]. The goal is to bring physical data independence to object-oriented and scientific databases and their applications.

Physical **pointers**, references, or object identifiers to represent relationships support "navigation" through a database, which is very good for single-instance retrievals and often improves set matching, but also creates a new type of updates, structural updates, which may increase the complexity of concurrency control and recovery [CSL90, ChK84, RoR85, ShC90].

Vertical partitioning of large records into multiple files, each with some of the attributes, allows faster retrieval of the most commonly used attributes; the problem is that joins or join-like operations are required when attributes from multiple files are needed [CoY87, CoY90, HaN79, MCV93, NCW84, NaR89].

Data **compression** is undervalued in most database implementations but will increase in popularity with the introduction of database systems that process compressed data to improve performance and of multi-level storage hierarchies ranging from on-chip caches over main memory and disks to automated tape libraries [Cor85, GrS91, Jag90, LR87, LyB81, OlR89].

File Formats

File formats concern the most basic level of physical database design: page allocation on disk (e.g., pre-allocation in contiguous extents), grouping of pages into units of I/O (called "clusters" here), division of clusters in records, fixed vs. variable-length records, space management within clusters, and records spanning clusters.

Free space in physical containers such as pages and files is useful for all data organized by some logical rule. For example, to facilitate easy insertion into a sorted file, some free space is typically left when a file is loaded. Indices are also often created with some initial fraction of free space. The important tradeoff is the ability to insert new data vs. the desire for compact representations and fast scans of large data collections.

Associative Search

Index selection must determine which index structures to create for which attributes and attribute combinations by comparing their benefits for searching against their update costs [GuB91, HaC76, KeW87]. Moreover, many authors have suggested various forms of multi-file indices and path indices [BeK89, Ber90, Ber91, Hae78, KeM90, Val87] and "partial indices," i.e., indices that include only a subset of an underlying relation or data set.

Other Redundant Data

Replication for reliability and availability ensures access to correct data at any time and often increases retrieval performance during normal operation but requires complex and expensive consistency management on updates and after separation of storage or processing units [BiG88, CAB88, HsD91, PGK88].

Derived information, e.g., materialized relational views, permits fast access to precomputed query results but requires that the derived information is recomputed, invalided, removed, or marked out-of-date after updates to the based data [BlM90, Han87, HuK89, Jhi91, KiH92, QiW91, Rou91, ToB88, YaL87]. Another very important form of derived data is replicating individual attribute values for faster access, in particular in database systems that do not support clustering across multiple data types [Bab82, ScS81]. For example, two relations *invoice* and *invoice-item* will require very frequent joins and many installations "denormalize" such master-detail relations by redundantly keeping descriptive attributes from the master items in the detail items, e.g., customer identifier and invoice date. Since these are redundant, the database system should propagate updates and avoid joins automatically.

File Access Performance

Clustering means assignment of data items such as records to disk locations in order to maximize the relevant information fetched with each I/O operation and therefore to reduce the number of I/O operations required to satisfy a database request — three important problems are prediction of future access patterns, compromising among different access patterns such as navigational access and set-oriented processing, and the dynamic change of access patterns [ChK89, ChH91, GhD90b, HNK90, OmS90, SnS90].

Declustering (striping) of pages over multiple disks or multiple nodes in a parallel or distributed database system permits more disk I/Os per unit time and higher total I/O bandwidth for a data collection at the expense of control overhead [CAB88, GHW90, PGK88, SaG86, StS90, WZS91].

Horizontal partitioning of items in large data collections using either round-robin partitioning (which is similar to striping) or systematic schemes such as hash- or range-partitioning is generally useful; its problems are selecting the partitioning attribute(s) and that various partitioning schemes interact differently with sort- and hash-based query evaluation algorithms for selection, join, etc. [DGG86, Ger86, GhD90a, LS92, WDI88].

If data sets are partitioned, there are two principal methods of **partitioning the indices** and other redundant data sets such as materialized views: either they are local, i.e., a separate file or index structure is created for each partition of the underlying data set, or they are global, i.e., a single file or index structure for the entire logical data set is partitioned over multiple disks or nodes in parallel or distributed systems. Both schemes have advantages and disadvantages; the choice depends on usage patterns, communication costs, and availability and maintenance effects.

Update Management

Differential files and **log-structured file systems** enable append-only updates, particularly useful in environments with compression and RAID-style storage devices, but destroys the contiguity of each file's space allocation on the storage media [RoO91, RoO92, SeL76].

Versions are particularly useful in design environments where falling back to an earlier alternative is often useful after some exploratory updates and where different configurations require different versions of the same data item or object, e.g. [KaL84, KCB86, Sto87].

Versions have also been used in **optimistic concurrency control** [HaP87, KuR81, MWY92, PaK84], which requires special storage structures for versions e.g., a separate storage area for temporary, transient record versions (version pool) or small version "caches" on each database page [BoC92].

Archival Storage

Archival storage has recently been pushed to increased research interest by the very largest database installations and by the emerging interest in database systems for scientific applications; the problems are storage formats appropriate for archival storage devices, selection of data to be moved ("retired") to archives, and the fact that data volumes, bandwidths, and access latencies vary widely from level to level in the storage hierarchy which may not permit direct adaptations of existing buffer replacement strategies [Doz92, SSU91, Sto91].

Automatic and timely **staging** of data to higher storage levels is a generalization of prefetching used in all high-performance database systems; the new problems are again the new data volumes, bandwidths, and access latencies [GRA91, PaZ91].

Further Considerations

It is important to recognize that most of these choices exist independently of the data model. For example, replication has beneficial availability and performance effects in network, relational, semantic, and object-oriented databases alike. On the other hand, clustering might have more effect in systems with logical or physical references, i.e., network and object-oriented databases, but master-detail clustering is used in relational system as well and is very effective in conjunction with index and pointer joins. Thus, extensible database systems designed to build high-performance database systems must allow for a wide array of physical database design options.

Rather than grouping physical database design options by their usability with one or more particular data models, it might be more useful to distinguish them by the amount of semantic knowledge they require. For example, replication (through page-by-page copying) and striping (on the basis of file pages) do not require any knowledge about the semantics of the data. Sorting requires some knowledge pertaining to locations of records in pages and of sort field in records as well as the correct sort order. Derived data require even more. Some other techniques, e.g., clustering and compression, can work with very little semantic information but are more effective as more semantic information is provided.

While the number of choices for physical database design is confusing, the most significant source of complexity in physical database design is that many decisions are interdependent. For example, the optimal clustering of data items depends on whether or not replication of data items is supported. When the clustering module cannot decide among two advantageous locations for a data item, it might choose to place a copy in each location. Replication can increase system performance by giving the optimizer or retrieval algorithm the choice of which copy to use; however, it must not be used too freely since it can also decrease overall performance if the cost of updating and maintaining multiple copies dominates their benefits [BLT86, DKH90, HuK89]. Moreover, the performance effects of replication are different depending on whether or not replicas of collections are declustered over multiple storage media as wholes or by means of partitioning, and whether or not the replicas are clustered in the same way. There is only limited research into making physical database design easier, e.g., [FST88, Kao86]. Considering the complexity of physical database design, automating physical database design in a comprehensive and extensible way seems to be an extremely fruitful area for database research, in particular in light of the added choices and complexity faced by the database implementor and administrator in extensible and object-oriented database management systems.

3. Summary

After the conceptual design of a database has been completed, physical database design is the next important step in utilizing a database management system. Choices and decision made in physical database design have a significant impact on the run-time performance of database applications; thus, physical database design can also be seen as a first step in query optimization. The task of the actual query optimizer is to find the best access path given the semantics of a query and the current state of the physical database design.

The foundation of physical data independence and of physical database design is the separation of logical or conceptual data content from physical data representation, closely related to the separation of interface specification and implementation in abstract data types. In order to achieve physical data independence, a database management system must provide (i) alternative representations for logical data sets, (ii) automatic mapping from update operations on the logical level to changes in the physical level's file and index structures, and (iii) automatic choices among alternative access paths (access plan or query optimization).

We have outlined several dimensions that must be considered in physical database design: file and index structures, vertical partitioning, clustering, declustering (striping, horizontal partitioning), replication, materialization of derived information, compression, differential files versus update-in-place, versioning, and archival storage. We hope that this list of physical database design choices will assist implementors of database software in choosing which options to provide in their systems and users of database software in choosing which options to demand and to utilize for their applications.

Acknowledgements

This paper is based on research partially supported by the National Science Foundation with grants IRI-9006348, IRI-9116547, IRI-9119446, and ASC-9217394, ARPA with contract DAAB 07-91-C-Q518, Digital Equipment Corp., and Texas Instruments.

References

[Bab82] E. Babb, Joined Normal Form: A Storage Encoding for Relational Databases, *ACM Trans. on Database Sys. 7*, 4 (December 1982), 588.

[BaM72] R. Bayer and E. McCreighton, Organisation and Maintenance of Large Ordered Indices, *Acta Informatica 1*, 3 (1972), 173.

[Ben75] J. L. Bentley, Multidimensional Binary Search Trees Used for Associative Searching, *Comm. of the ACM 18*, 9 (September 1975), 509.

[BeK89] E. Bertino and W. Kim, Indexing Techniques for Queries on Nested Objects, *IEEE Trans. on Knowledge and Data Eng. 1*, 2 (June 1989), 196.

[Ber90] E. Bertino, Optimization of Queries Using Nested Indices, *Lecture Notes in Comp. Sci. 416*(March 1990), 44, Springer Verlag.

[Ber91] E. Bertino, An Indexing Technique for Object-Oriented Databases, *Proc. IEEE Conf. on Data Eng.*, Kobe, Japan, April 1991, 160.

[BiG88] D. Bitton and J. Gray, Disk Shadowing, *Proc. Int'l. Conf. on Very Large Data Bases*, Los Angeles, CA, August 1988, 331.

[BLT86] J. A. Blakeley, P. A. Larson, and F. W. Tompa, Efficiently Updating Materialized Views, *Proc. ACM SIGMOD Conf.*, Washington, DC, May 1986, 61.

[BlM90] J. A. Blakeley and N. L. Martin, Join Index, Materialized View, and Hybrid Hash-Join: A Performance Analysis, *Proc. IEEE Conf. on Data Eng.*, Los Angeles, CA, February 1990, 256.

[BoC92] P. M. Bober and M. J. Carey, On Mixing Queries and Transactions via Multiversion Locking, *Proc. IEEE Conf. on Data Eng.*, Tempe, AR, February 1992, 535.

[CSL90] M. J. Carey, E. Shekita, G. Lapis, B. Lindsay, and J. McPherson, An Incremental Join Attachment for Starburst, *Proc. Int'l. Conf. on Very Large Data Bases*, Brisbane, Australia, August 1990, 662.

[Cha91] S. Chakravarthy, Divide and Conquer: A Basis for Augmenting a Conventional Query Optimizer with Multiple Query Processing Capabilities, *Proc. IEEE Conf. on Data Eng.*, Kobe, Japan, April 1991, 482.

[ChK89] E. Chang and R. Katz, Exploiting Inheritance and Structure Semantics for Effective Clustering and Buffering in an Object-Oriented DBMS, *Proc. ACM SIGMOD Conf.*, Portland, OR, May-June 1989, 348.

[ChK84] H. Chen and S. M. Kuck, Combining Relational and Network Retrieval Methods, *Proc. ACM SIGMOD Conf.*, Boston, MA, June 1984, 131.

[ChH91] J. R. Cheng and A. R. Hurson, Effective Clustering of Complex Objects in Object-Oriented Databases, *Proc. ACM SIGMOD Conf.*, Denver, CO, May 1991, 22.

[Com79] D. Comer, The Ubiquitous B-Tree, *ACM Computing Surveys 11*, 2 (June 1979), 121.

[CAB88] G. Copeland, W. Alexander, E. Boughter, and T. Keller, Data Placement in Bubba, *Proc. ACM SIGMOD Conf.*, Chicago, IL, June 1988, 99.

[Cor85] G. V. Cormack, Data Compression In a Database System, *Comm. of the ACM 28*, 12 (December 1985), 1336.

[CoY87] D. W. Cornell and P. S. Yu, A Vertical Partitioning Algorithm for Relational Databases, *Proc. IEEE Conf. on Data Eng.*, Los Angeles, CA, February 1987, 30.

[CoY90] D. Cornell and P. Yu, An Effective Approach to Vertical Partitioning for Physical Design of Relational Databases, *IEEE Trans. on Softw. Eng. 16*, 2 (February 1990), 248.

[DGG86] D. J. DeWitt, R. H. Gerber, G. Graefe, M. L. Heytens, K. B. Kumar, and M. Muralikrishna, GAMMA - A High Performance Dataflow Database Machine, *Proc. Int'l. Conf. on Very Large Data Bases*, Kyoto, Japan, August 1986, 228. Reprinted in M. Stonebraker, Readings in Database Sys., Morgan-Kaufman, San Mateo, CA, 1988.

[Doz92] J. Dozier, Keynote Address: Access to Data in NASA's Earth Observing Systems, *Proc. ACM SIGMOD Conf.*, San Diego, CA, June 1992, 1.

[DKH90] P. Drew, R. King, and S. Hudson, The Performance and Utility of the Cactis Implementation Algorithms, *Proc. Int'l. Conf. on Very Large Data Bases*, Brisbane, Australia, August 1990, 135.

[FNP79] R. Fagin, J. Nievergelt, N. Pippenger, and H. R. Strong, Extendible Hashing: A Fast Access Method for Dynamic Files, *ACM Trans. on Database Sys. 4*, 3 (September 1979), 315.

[FiB74] R. A. Finkel and J. L. Bentley, Quad Trees: A Data Structure for Retrieval on Composite Keys, *Acta Informatica 4*, 1 (1974), 1.

[FST88] S. Finkelstein, M. Schkolnick, and P. Tiberio, Physical Database Design for Relational Databases, *ACM Trans. on Database Sys. 13*, 1 (March 1988), 91.

[Ger86] R. H. Gerber, Dataflow Query Processing using Multiprocessor Hash-Partitioned Algorithms, *Ph.D. Thesis, Univ. of Wisconsin – Madison*, October 1986.

[GhD90a] S. Ghandeharizadeh and D. J. DeWitt, Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machines, *Proc. Int'l. Conf. on Very Large Data Bases*, Brisbane, Australia, August 1990, 481.

[GhD90b] S. Ghandeharizadeh and D. J. DeWitt, A Multiuser Performance Analysis of Alternative Clustering Strategies, *Proc. IEEE Conf. on Data Eng.*, Los Angeles, CA, February 1990, 466.

[GRA91] S. Ghandeharizadeh, L. Ramos, Z. Asad, and W. Qureshi, Object Placement in Parallel Hypermedia Systems, *Proc. Int'l. Conf. on Very Large Data Bases*, Barcelona, Spain, September 1991, 243.

[GrS91] G. Graefe and L. D. Shapiro, Data Compression and Database Performance, *Proc. ACM/IEEE-Comp. Sci. Symp. on Applied Computing*, Kansas City, MO, April 1991.

[Gra93] G. Graefe, Query Evaluation Techniques for Large Databases, *ACM Computing Surveys 25*, 2 (June 1993), .

[GHW90] J. Gray, B. Horst, and M. Walker, Parity Striping of Disc Arrays: Low-Cost Reliable Storage with Acceptable Throughput, *Proc. Int'l. Conf. on Very Large Data Bases*, Brisbane, Australia, August 1990, 148.

[GuB91] O. Guenther and J. Bilmes, Tree-Based Access Methods for Spatial Databases: Implementation and Performance Evaluation, *IEEE Trans. on Knowledge and Data Eng. 3*, 3 (September 1991), 342.

[Gut84] 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.

[Hae78] T. Haerder, Implementing a Generalized Access Path Structure for a Relational Database System, *ACM Trans. on Database Sys. 3*, 3 (September 1978), 285.

[HaP87] T. Haerder and E. Petry, Evaluation of a multiple version scheme for concurrency control, *Inf. Sys. 12*, 1 (1987), 83.

[HaC76] M. Hammer and A. Chan, Index Selection in a Self-Adaptive Data Base Management System, *Proc. ACM SIGMOD Conf.*, 1976, 1.

[HaN79] M. Hammer and B. Niamir, A Heuristic Approach to Attribute Partitioning, *Proc. ACM SIGMOD Conf.*, Boston, MA, May-June 1979, 93.

[Han87] E. N. Hanson, A Performance Analysis of View Materialization Strategies, *Proc. ACM SIGMOD Conf.*, San Francisco, CA, May 1987, 440.

[HNK90] L. Harada, M. Nakano, M. Kitsuregawa, and M. Takagi, Query Processing Method for Multi-Attribute Clustered Relations, *Proc. Int'l. Conf. on Very Large Data Bases*, Brisbane, Australia, August 1990, 59.

[HsD91] H. Hsiao and D. DeWitt, A Performance Study of Three High Availability Data Replication Strategies, *Proc. Int'l. Conf. on Parallel and Distr. Inf. Sys.*, Miami Beach, FL, December 1991.

[HuK89] S. E. Hudson and R. King, Cactis: A Self-Adaptive, Concurrent Implementation of an Object-Oriented Database Management System, *ACM Trans. on Database Sys. 14*, 3 (September 1989), 291.

[Jag90] H. V. Jagadish, A Compression Technique to Materialize Transitive Closure, *ACM Trans. on Database Sys. 15*, 4 (December 1990), 558.

[Jhi91] A. Jhingran, Precomputation in a Complex Object Environment, *Proc. IEEE Conf. on Data Eng.*, Kobe, Japan, April 1991, 652.

[Kao86] S. Kao, DECIDES: An Expert System Tool for Physical Database Design, *Proc. IEEE Conf. on Data Eng.*, Los Angeles, CA, February 1986, 671.

[KaL84] R. H. Katz and T. J. Lehman, Database Support for Versions and Alternatives of Large Design Files, *IEEE Trans. on Softw. Eng. 10*, 2 (March 1984), 191.

[KCB86] R. H. Katz, E. Chang, and R. Bhateja, Version Modeling Concepts for Computer-Aided Design Databases, *Proc. ACM SIGMOD Conf.*, Washington, DC, May 1986, 379.

[KeW87] A. Kemper and M. Wallrath, An Analysis of Geometric Modeling in Database Systems, *ACM Computing Surveys 19*, 1 (March 1987), 47.

[KeM90] A. Kemper and G. Moerkotte, Access Support in Object Bases, *Proc. ACM SIGMOD Conf.*, Atlantic City, NJ, May 1990, 364.

[KiH92] K. C. Kinsley and C. E. Hughes, Analysis of a Virtual Memory Model For Maintaining Database Views, *IEEE Trans. on Softw. Eng. 18*, 5 (May 1992), 402.

[KuR81] H. T. Kung and J. T. Robinson, On Optimistic Methods for Concurrency Control, *ACM Trans. on Database Sys. 6*, 2 (June 1981), 213. Reprinted in M. Stonebraker, Readings in Database Sys., Morgan-Kaufman, San Mateo, CA, 1988.

[Lar81] P. A. Larson, Analysis of Index-Sequential Files with Overflow Chaining, *ACM Trans. on Database Sys. 6*, 4 (December 1981), 671.

[LR87] J. Li, D. Rotem, and H. Wong, A New Compression Method with Fast Searching on Large Data Bases, *Proc. Int'l. Conf. on Very Large Data Bases*, Brighton, England, August 1987, 311.

[LS92] J. Li, J. Srivastava, and D. Rotem, CMD: A Multidimensional Declustering Method for Parallel Data Systems, *Proc. Int'l. Conf. on Very Large Data Bases*, Vancouver, BC, Canada, August 1992, 3.

[Lit80] W. Litwin, Linear Hashing: A New Tool For File and Table Addressing, *Proc. Int'l. Conf. on Very Large Data Bases*, Montreal, Canada, October 1980, 212. Reprinted in M. Stonebraker, Readings in Database Sys., Morgan-Kaufman, San Mateo, CA, 1988.

[LoS90] D. B. Lomet and B. Salzberg, The hB-Tree: A Multiattribute Indexing Method with Good Guaranteed Performance, *ACM Trans. on Database Sys. 15*, 4 (December 1990), 625.

[LyB81] C. A. Lynch and E. B. Brownrigg, Application of Data Compression to a Large Bibliographic Data Base, *Proc. Int'l. Conf. on Very Large Data Bases*, Cannes, France, September 1981, 435.

[MaV93] D. Maier and B. Vance, A Call to Order, *Proc. ACM SIGACT News-SIGMOD-SIGART Symp. on Principles of Database Sys.*, Washington, DC, May 1993, 1.

[MWY92] A. Merchant, K. L. Wu, P. S. Yu, and M. S. Chen, Performance Analysis of Dynamic Finite Versioning for Concurrent Transaction and Query Processing, *Proc. 1992 ACM SIGMETRICS and PERFORMANCE '92 Int'l. Conf. on Measurement and Modeling of Computer Systems 20*, 1 (June 1-5, 1992), 103.

[MCV93] J. Muthuraj, S. Chakravarthy, R. Varadarajan, and S. Navathe, A Formal Approach to the Vertical Partitioning Problem in Distributed Database Design, *Proc. Parallel and Distr. Inf. Sys.*, San Diego, CA, January 1993.

[NCW84] S. Navathe, S. Ceri, G. Wiederhold, and J. Dou, Vertical Partitioning Algorithms for Database Design, *ACM Trans. on Database Sys. 9*, 4 (December 1984), 680.

[NaR89] S. Navathe and M. Ra, Vertical Partitioning for Database Design – A Graphical Algorithm, *Proc. ACM SIGMOD Conf.*, Portland, OR, May-June 1989, 440.

[NHS84] J. Nievergelt, H. Hinterberger, and K. C. Sevcik, The Grid File: An Adaptable, Symmetric Multikey File Structure, *ACM Trans. on Database Sys. 9*, 1 (March 1984), 38.

[OlR89] F. Olken and D. Rotem, Rearranging Data to Maximize the Efficiency of Compression, *J. of Computer and System Sciences 38*, 2 (1989), 405.

[OmS90] E. Omiecinski and P. Scheuermann, A Parallel Algorithm for Record Clustering, *ACM Trans. on Database Sys. 15*, 4 (December 1990), 599.

[PaZ91] M. Palmer and S. B. Zdonik, FIDO: A Cache that Learns to Fetch, *Proc. Int'l. Conf. on Very Large Data Bases*, Barcelona, Spain, September 1991, 255.

[PaK84] C. H. Papadimitriou and P. C. Kanellakis, On Concurrency Control by Multiple Versions, *ACM Trans. on Database Sys. 9*, 1 (March 1984), 89.

[PGK88] D. A. Patterson, G. Gibson, and R. H. Katz, A

Case for Redundant Arrays of Inexpensive Disks (RAID), *Proc. ACM SIGMOD Conf.*, Chicago, IL, June 1988, 109.

[QiW91] X. Qian and G. Wiederhold, Incremental Recomputation of Active Relational Expressions, *IEEE Trans. on Knowledge and Data Eng. 3*, 3 (September 1991), 337.

[Rob81] J. T. Robinson, The K-D-B-Tree: A Search Structure For Large Multidimensional Dynamic Indices, *Proc. ACM SIGMOD Conf.*, Ann Arbor, MI, April-May 1981, 10.

[RoO91] M. Rosenblum and J. K. Ousterhout, The Design and Implementation of a Log-Structured File System, *Proc. Thirteenth ACM Symp. on Operating System Principles 25*, 5 (October 1991), 1.

[RoO92] M. Rosenblum and J. K. Ousterhout, The Design and Implementation of a Log-Structured File System, *ACM Trans. on Computer Sys. 10*, 1 (February 1992), 26.

[RoR85] A. Rosenthal and D. S. Reiner, Querying Relational Views of Networks, in *Query Processing in Database Sys.*, W. Kim, D. S. Reiner, and D. S. Batory (ed.), Springer, Berlin, 1985, 109.

[Rou91] N. Roussopoulos, An Incremental Access Method for ViewCache: Concept, Algorithms, and Cost Analysis, *ACM Trans. on Database Sys. 16*, 3 (September 1991), 535.

[SaG86] K. Salem and H. Garcia-Molina, Disk Striping, *Proc. IEEE Conf. on Data Eng.*, Los Angeles, CA, February 1986, 336.

[Sam84] H. Samet, The Quadtree and Related Hierarchical Data Structures, *ACM Computing Surveys 16*, 2 (June 1984), 187.

[ScS81] M. Schkolnick and P. Sorensen, The Effects of Denormalization on Database Performance, *IBM Res. Report RJ3082*, 1981.

[SAC79] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, Access Path Selection in a Relational Database Management System, *Proc. ACM SIGMOD Conf.*, Boston, MA, May-June 1979, 23. Reprinted in M. Stonebraker, Readings in Database Sys., Morgan-Kaufman, San Mateo, CA, 1988.

[Sel88] T. K. Sellis, Multiple-Query Optimization, *ACM Trans. on Database Sys. 13*, 1 (March 1988), 23.

[SeL76] D. Severance and G. Lohman, Differential Files: Their Application to the Maintenance of Large Databases, *ACM Trans. on Database Sys. 1*, 3 (September 1976), .

[ShC90] E. J. Shekita and M. J. Carey, A Performance Evaluation of Pointer-Based Joins, *Proc. ACM SIGMOD Conf.*, Atlantic City, NJ, May 1990, 300.

[SSU91] A. Silberschatz, M. Stonebraker, and J. Ullman, Database Systems: Achievements and Opportunities, *Comm. of the ACM, Special Section on Next-Generation Database Systems 34*, 10 (October 1991), 110.

[SnS90] R. Snodgrass and K. Shannon, Semantic Clustering, *Fourth Int'l Workshop on Persistent Object Sys.*, Martha's Vineyard, MA, September 1990, 361.

[Sto87] M. Stonebraker, The Design of the POSTGRES Storage System, *Proc. Int'l. Conf. on Very Large Data Bases*, Brighton, England, August 1987, 289. Reprinted in M. Stonebraker, Readings in Database Sys., Morgan-Kaufman, San Mateo, CA, 1988.

[StS90] M. Stonebraker and G. A. Schloss, Distributed RAID – A New Multiple Copy Algorithm, *Proc. IEEE Conf. on Data Eng.*, Los Angeles, CA, February 1990, 430.

[Sto91] M. Stonebraker, Managing Persistent Objects in a Multi-Level Store, *Proc. ACM SIGMOD Conf.*, Denver, CO, May 1991, 2.

[ToB88] F. W. Tompa and J. A. Blakeley, Maintaining materialized views without accessing base data, *Inf. Sys. 13*, 4 (1988), 393.

[Val87] P. Valduriez, Join Indices, *ACM Trans. on Database Sys. 12*, 2 (June 1987), 218.

[WZS91] G. Weikum, P. Zabback, and P. Scheuermann, Dynamic File Allocation in Disk Arrays, *Proc. ACM SIGMOD Conf.*, Denver, CO, May 1991, 406.

[WDI88] J. L. Wolf, D. M. Dias, B. R. Iyer, and P. S. Yu, A Hybrid Data Sharing - Data Partitioning Architecture for Transaction Processing, *Proc. IEEE Conf. on Data Eng.*, Los Angeles, CA, February 1988, 520.

[YaL87] H. Yang and P. A. Larson, Query Transformation for PSJ-queries, *Proc. Int'l. Conf. on Very Large Data Bases*, Brighton, England, August 1987, 245.