In this paper we examine different techniques which can be used to partition a database: horizontal, group horizontal, single vertical, physical vertical, group vertical, and mixed partitioning. We evaluate these techniques by examining eight properties which determine the behavior of database operations. For each property, we derive rankings of the partitioning techniques. These rankings can be used to help select the best partitioning technique for a database system. An approach to obtain the overall ranking is discussed. While no best partitioning approach exists for all environments, the information and discussions contained within this paper can be used by a database designer to aid in selecting the best technique for a target environment.

The performance of a database system is largely dependent on the database design. Database design is the process of developing a database structure based on the requirements gathered about the system (Teorey & Fry, 1982). An important issue in this process is how to physically organize the data so that an optimal system performance can be achieved (March, 1983). One of the issues raised in physical database design is the placement of data within physical pages, database partitioning. Database partitioning is the process of assigning a logical object (relation) from the logical schema of the database to physical pages. A good partitioning technique can reduce the number of pages required to store a database as well as reduce the I/O needed to bring data into main memory for processing.

Database partitioning can be applied in different architectural environments (Ceri, Negri, & Pelagatti, 1982):

- distributed database systems.
- data allocation on physical devices of the same or different types.
- data stored in different memory levels such as primary and secondary.

In distributed database systems, the partitioning goal is to maximize the local transaction processing and thus reduce the amount of data transmitted on the network. Allocating data to physical devices requires determining the best partitioning approach to reduce the overall transaction processing time (including I/O and CPU). The objective of data partitioning in multi-memory levels is to store the data that is accessed by most transactions in faster memories.

There are many different techniques one can use to partition a database into logical groups. One technique may require more pages to store a relation than another. Likewise, one technique may need more time to process a transaction than another. In this paper, we analyze possible partitioning techniques in terms of the number of I/O accesses and number of main memory references during transaction processing. The objective of this paper is to analyze different data partitioning approaches and to provide the database designer insight into determining the best partitioning technique for his/her environment. Since the best partitioning strategy depends on the system requirements, we can not provide one precise answer as to the “best” technique. However, we examine each technique based upon several performance parameters and provide a ranking of the techniques on each parameter. The database designer can use these results as an aid in the design process.

The remainder of this paper is organized as follows. Section 2 introduces possible partitioning techniques. Section 27
3 describes the overall approach we take to perform our analysis. Sections 4 and 5 contain the analysis and recommendations to the designer. Section 5 concludes the paper.

**Database Partitioning Techniques**

Many database partitioning techniques have been developed to physically organize data on the storage devices. Each technique divides the data into groups which are then assigned to physical pages. We classify these into the following six categories:

- horizontal partitioning.
- group horizontal partitioning.
- single vertical partitioning.
- physical vertical partitioning.
- group vertical partitioning.
- mixed partitioning.

Horizontal partitioning subdivides a relation to form groups of tuples without taking tuple affinity into account. Each group contains a number of complete tuples. Due to its simplicity this technique has been used as a conventional storage structure. The group horizontal partitioning technique decomposes a relation into groups of tuples based on their affinity. Tuples that are more frequently used together are placed in the same group. This technique has been implemented in several distributed database systems such as SDD-1 (Rothnie, 1980), distributed INGRES (Stonebraker & Neuhold, 1977), and R' (Williams, 1985). Ceri has proposed a group horizontal partitioning algorithm based on predicates deduced from transactions (Ceri, Negri, & Pelagatti, 1982). Shin has proposed a knowledge-based approach to group tuples (Shin & Irani, 1985). His purpose is to derive more precise user reference clusters from the queries.

Single vertical partitioning focuses on references to single attributes without considering attribute affinity. It vertically subdivides a relation into groups each of which contains only one type of attribute. Original tuples can be identified by replicating either primary keys or system controlled tuple identifiers with the attribute value or by storing all attributes in the same relative order. This technique has been used to implement inverted file models in several database management systems, such as ADABAS and DATACOM/DB, to physically store their secondary keys (Pratt & Adamski, 1987).

In the physical vertical partitioning technique, each tuple is divided into physical groups of fixed sizes that are independent of attribute length. Some attributes may spread over several groups. This technique has not been implemented in any database management systems that we are aware of. We include it here for completeness. In this technique, a group size is chosen independently of attribute sizes or tuple sizes. Therefore the same partitioning mechanism and size can be applied to all relations.

Group vertical partitioning has been extensively studied in a variety of computing environments. It subdivides attributes of a relation into groups based on affinity. “Attributes that are commonly required together are then physically stored and accessed together. Conversely, attributes that are not commonly required together are not stored together” (March, 1983). Babad formulates the group vertical partitioning problem for variable length attributes as a nonlinear, zero-one program (Babad, 1977). Hoffer presents an attribute clustering technique applicable to physical database design (Hoffer & Severance, 1975). A metric with which to measure the similarity of usage among data items is developed and used by the clustering algorithm which is constructed by using the Bond Energy Algorithm (BEA) (McCormick, Schweitzer & White, 1972). Navathe extends the results of Hoffer by giving algorithms for an automatic selection of vertical fragments (Navathe, Ceri, Wiederhold & Don, 1984). Cornell proposes a vertical partitioning algorithm that takes into account the type of scan on the relation by each transaction type (Cornell & Yu, 1987). This algorithm partitions attributes to minimize number of disk accesses, based on the type of scan: segment or index, and scan frequency which are provided by the query optimizer.

The purpose of the mixed partitioning technique is to take advantages of both horizontal and vertical partitioning (Ceri, 1984). Partitions of data are constructed by either applying group horizontal to physical/group vertical partitioning or vice versa. The SDD-1 distributed system is one of the database systems that uses mixed partitioning to partition its database (Rothnie, 1980). In SDD-1, the database first is divided into some subsets based on some simple restrictions on attributes using group horizontal partitioning. In the second phase, each of these horizontal subsets is partitioned into fragments defined by projections.

**Example:**

In this example (Figures 2-7) we illustrate each of the six possible partitioning techniques using the following employee relation (Figure 1):

- Relation size: 100 tuples.
- Each tuple has three fields: Name (15 bytes), Phone (10 bytes), and Address (50 bytes).
- Physical page size: 750 bytes.

With the horizontal partitioning approach 10 tuples fit into each page and so 10 physical pages are required to store the relation. Tuples emp1, emp2, ..., emp10 are stored in page 1, tuples emp11, emp12, ..., emp20 are stored in page 2, and so on. Figure 2 shows page 1 and page 10 from 10 pages needed to store 100 tuples using horizontal partitioning. In the group horizontal partitioning technique at least 10 pages would be required, however the order of tuples would differ from that of
Related Content

Node Partitioned Data Warehouses: Experimental Evidence and Improvements
[www.igi-global.com/article/node-partitioned-data-warehouses/3352?camid=4v1a](www.igi-global.com/article/node-partitioned-data-warehouses/3352?camid=4v1a)

The Impact of Network Layer on the Deadline Assignment Strategies in Distributed Real-Time Database Systems
[www.igi-global.com/article/impact-network-layer-deadline-assignment/51163?camid=4v1a](www.igi-global.com/article/impact-network-layer-deadline-assignment/51163?camid=4v1a)

Enhancing UML Models: A Domain Analysis Approach
[www.igi-global.com/chapter/enhancing-uml-models/7993?camid=4v1a](www.igi-global.com/chapter/enhancing-uml-models/7993?camid=4v1a)

Enhancing Database Access Control by Facilitating Non-Key Related Cover Stories
[www.igi-global.com/article/enhancing-database-access-control-facilitating/3334?camid=4v1a](www.igi-global.com/article/enhancing-database-access-control-facilitating/3334?camid=4v1a)