Chapter 8

Data Organizational Methods

Without the ability to view, manipulate, and update data stored on the computer, the most sophisticated storage infrastructure is meaningless. Our last storage fundamental covers the methods and types of organizational models used to facilitate the productive use of stored data. Data organizational models and methods are part of the entire storage infrastructure. Their proximity to other elements within the infrastructure indicates that structures, such as file systems and databases, are the critical connectivity between application and end user. Data is stored for many different reasons and accessed in as many ways to support a rapidly changing business environment. The knowledge, awareness, and insight to the diversity of data organizational models is as important and critical as that of storage systems and connectivity options.

Organizing Data

Data in computer systems is organized by creating databases. Databases are defined as a collection of interrelated material stored in an environment that is both convenient and efficient to use in retrieving information. This is done through a combination of file systems, database management systems (DBMS), and applications. Combined with an effective storage system, the database can offer an environment where data can not only be stored and updated on a reliable basis but also be accessed in multiple ways.

Underlying the DBMS is the file system that works as a component of the operating system. The file system functions as the operating system's ability to organize and retrieve the physical data that exists on attached storage media and within memory devices. File systems deal with the physical aspects of storage media, the data locations, as they are stored within the media segments, and the status of the data within. They also provide the first level of transparency to the user or owner of the data.

Database management systems (DBMS) provide methods for defining the data, enhanced access to data generally through some type of enhanced indexing system, and functions for maintaining the data structures as well as the data itself. Defining and accessing the data through a DBMS, users can additionally perform calculations and logical functions on the stored data, and present the data in enhanced views (usually tabular reports that include graphics and charts).

Business applications take advantage of the file system and DBMS functions by using their transparent data retrieval and dealing directly with the data without having to understand or know the underlying operations of the DBMS or a native file system. As we indicated in Chapter 1, most application designers and programmers can use these high-level functions without regard to the underlying hardware and software infrastructure. Others can be locked into specific DBMSs and file systems that have become standardized within their data centers.

Interrelationships of Performance

File systems, database systems, and storage systems are interrelated and dictate levels of performance for the storage infrastructure. File systems, for their part, form the basis for efficient and effective access to an application's data. Problems occur, however, when the file system is doing its job with unoptimized disks, insufficient disk space, or bandwidth constraints in the bus. As application performance becomes problematic, the file system attributes should be considered for better alignment with the available storage infrastructure.

Many relational databases don't use an operating system's file system at all. Given that's the case in many database instances, then what substitutes the functionality of the file system within the database? Actually, many database systems interact closely with installed storage systems through the use of their own file system and virtual I/O manager.

Finally, many applications rely on the interaction of files that are networked. This provides yet another dimension to the interrelationship of the file system to storage and the challenges to the integrity and reliability of the data stored within a network.

Virtualization Times Three

Data organizational methods begin the first slide into the confusion of virtualization. This is because File Systems provide a layer of abstraction to the physical location of data, the location of disks, and the storage media itself. Database systems further abstract the physical data, physical location, and the actual data. Finally, the application using both organizational methods will further hide (for example, virtualize) the complexities of finding and managing the data. In many cases, we are left with the question of where the actual data is.

The operating system component that encompasses the allocation of space, data structures, and which keeps track of these organizational storage criteria is the File System. File Systems operate within the I/O manager component of the OS. In most operating systems, such as UNIX and Windows, the file system operates as a driver or an additional service that will interact with the OS's I/O management component. This allows multiple file systems to simultaneously operate on a single server.

Requirements for Enterprise-Class File Systems

Enterprise-class file systems need far more than convenience and efficiency to be considered adequate support for enterprise OLTP, data warehouse, Internet, and complex batch operations. The following are some of the global requirements that should be addressed by any enterprise-class file system and the issues they raise:

- **Security** Limiting user access to files and exercising control over sensitive physical resources—such as volumes and system files—continue to be the two major requirements for security within file systems. This has become cumbersome as multiple and often duplicate security functions are offered by both file and database systems. From the storage perspective, there exists a lack of effective and integrated resource security and control functions that are available within the storage infrastructure components.

- **Data Recovery** Disasters, both large and small, are going to occur. The ability to recover in a timely fashion has long been a requirement, as databases become systems of record for both large and small enterprises. This, however, becomes more sophisticated and complex as recovery functions for databases differ from recovery functions for file systems which, in turn, differ from the recovery system of an entire application.
- **Fault Resiliency/Tolerance** This is the "timely" part of the recovery requirement. Most notably, the requirement that says you have to be up for 24/7. Although a noble thought, the reality of keeping an enterprise system available in this manner does not exist yet. Having said that, however, the functionality to repair and recover from most hardware failures without disrupting the application can be accomplished at various levels. The problem continues to be that enterprise application systems are likened to living entities that consume resources, and many of these are just temporary and need to be refreshed from time to time. Secondly, the ability to design and place card-holders (for lack of a better term) in the processing time frames has failed to mature with the sophisticated hardware/recovery systems available today.

Note *24/7 systems have existed for some time in mainframe configurations, with full redundancy at various levels. However, as data centers migrate their operations from the mainframe to open systems using commodity-based platforms, additional levels of maturity must evolve before open system hardware and software provide the same levels of desired fault tolerance.*

- **Support for Advanced Storage Systems and Application Requirements** Last, but certainly not least, is the support for future enhancements to disk drives, disk formats, tape, and so on. There's no question that this is an important factor given that databases and file systems are closely tied to their storage relatives and must keep up. The typical IT oversight in this requirement concerns the effects that support of back level software (for instance, file systems and databases) has on hardware manufacturers. Although it looks altruistic on the outside, inefficiencies develop when this continues for any length of time.

File Systems

From the most academic definition, the file system is a logical sequence of blocks on a storage media. Files are a basic construct of the operating system. With the exception of certain data (refer to relational databases further in this chapter), all data is kept in some type of file format. As such, it requires a system to maintain and manage the relationship with storage on behalf of the operating system. These file system functions are shown in Figure 8-1 and can be categorized in the following manner: allocation, management, and operations. These functions are described as follows:

- **Allocation** File systems provide the capability to organize I/O devices into functioning units of storage.
- **Management** File systems provide the activities necessary to track, protect, and manipulate data stored within the I/O devices.
- **Operations** File systems locate logical sequences of data (for instance, a file) through various search methods depending on data recoverability and the sophistication of the system.

Allocation

The file system allocates the two most important elements of the storage media, the volume, which defines the physical device, and its related attributes. Secondly, the files, which are collections of data, are accessible through some type of naming convention to the operating system and, subsequently, the applications are stored physically on the storage media. In today's environments, multiple products can

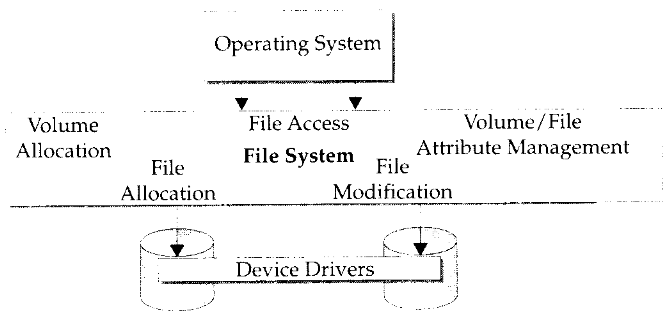

Figure 8-1. *The components of a file system*

make up these components. Accessible as add-ons to the basic components of the operating system, these products support two distinct but interrelated functions: file management and volume management. These products are available as recoverable, journal, or high-performance file systems and as enhanced or extended volume managers.

Volume Definition

Allocation of space within storage, especially in disks, begins with the initialization or formatting of the physical device itself. This is done by creating a volume that corresponds to a logical partition of space on a disk. Given the esoteric nature of volume initialization, volumes can be assigned to part of a disk or can span physical disks.

The activity of formatting defines the smallest unit of addressable space within the device and the creation of special files that will act as directories or catalogues for file allocation and access. In Microsoft operating systems, these are referred to as clusters and are mapped to the physical sectors of the physical disk. In UNIX, this is generally defined, given there are multiple UNIX variants, as a *super block*. Consequently, there exists a relationship between the file system allocation and the actual physical attributes of the disk, such as capacity and density. Both file system units—clusters and blocks—have an integral number of physical sectors. Therefore, large physical disks storing small files can become very inefficient when the lowest unit of storage is a cluster that utilizes multiple large sectors.

In addition, the overhead necessary to put the disk into activity (for instance, its formatting) explains the difference between formatted capacities of a drive versus unformatted capacity.

File Allocation

Once a volume is initialized, it is ready to store a file or logical sets of blocks. Files are allocated according to the attributes set with volume formatting. Given that, a file is allocated space through its smallest unit of measurement, which is the cluster or block. Files are tracked through special files that have been created with volume allocation. These master file tables (MFT), as they exist in Microsoft OSs (or inodes, as they're referred to in UNIX systems), contain data about the volume, space attributes, and files stored within the volume.

Access to files starts here with the master file table or inode. Within these special files are indexing information, security access lists, file attribute lists, and any extended attributes that the file may have. Once the file has been created, these attributes are managed from these locations stored on the volume.

Management

The MFT, file allocation table, or super block-inode structures are increasingly being referred to as metadata. Metadata simply means information about data. As we discussed previously, the operating system requires a function to store and access data. One of these is the fundamental boot structure needed by the OS to initialize itself during a power up or restart sequence. In addition to the metadata files allocation during formatting, the process identifies and creates a boot file that becomes instrumental in locating the special system information needed during the OS boot processes.

Another important allocation during this process is the creation of a log file, thereby providing a recoverable file system, one that is required for enterprise-level operations. A recoverable file system ensures volume consistency by using logging functions similar to transaction processing models. If the system crashes, the file system restores the volume by executing a recovery procedure that utilized activity information stored within the log file.

In file systems, all data stored on a volume is contained in a file. This includes the file allocation tables and volume table of contents structures used to located and retrieve files, boot data, and the allocation state of the entire volume. Storing everything in files allows the data to be easily located and maintained by the file system, and a security attribute or descriptor can protect each separated file. If a particular part of the physical disk becomes corrupt, the file system can relocate the metadata files to prevent the disk from becoming inaccessible.

Note *There is an important exception to the preceding statement that "all data stored on a volume is contained in a file." The exception is the allocation of disks using raw partitions where an application will bypass the file system and manage the raw storage segments. This is common for many relational database products.*

Types of File Systems

Windows utilizes several types of file systems that support both client and server operations. Most client OSs only need to support fundamental volume and file allocation functions. This is handled through its ability to initialize and partition volumes with formatting and file allocation through a simple file allocation table. However, within server products, a more robust file system is required, as shown in Figure 8-2. Windows server products provide a NT File System (NTFS), which provides a robust file system that supports enhanced volume and file allocation functions. These products include volume fault resiliency features for partitioning volumes with redundancy features, thus providing a subset of RAID features through software. Also supported is recoverability through logging functions. Enhanced security attributes are provided to support multiuser environments with levels of data protection for both user access and file and volume protection.

UNIX has several variants; however, most offer a POSIX-compliant file system that is a "bare-bones" component of the operating system. Enhanced and recoverable file systems are available through third-party products. These add-ons provide both enhanced performance and journaling. In UNIX, recoverable file systems are referred

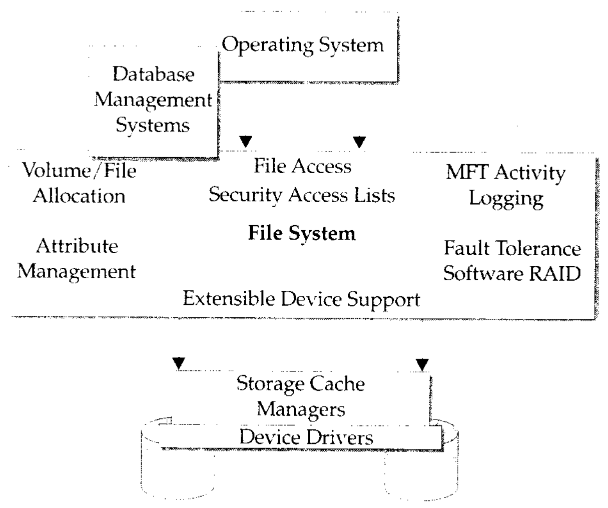

Figure 8-2. *An enterprise-class file system*

to as "journal" file systems, e.g., logging being synonymous with journaling. More importantly, the UNIX markets have the availability and legacy of network file systems, where files are shared across a network through a common file system and access protocol.

Note *The POSIX Standard, also known as the IEEE 1003.x POSIX standard, is the result of working groups under the auspices of IEEE. It describes a set of standard operating system interfaces.*

Access to information via the Internet and through World Wide Web interfaces has produced yet another variant of the file system, the Hyper Text Transport Protocol, or HTTP. Although more of a common file access protocol, it does define a file structure within the web-based environments depicted in Figure 8-3. Because it is supported across both UNIX and Windows, it provides a common protocol for accessing files through a network.

Another advancement in common file protocols is the Common Internet File System (CIFS), as shown in Figure 8-4. Starting as a Microsoft initiative, it has now become an open standard. Based on the need to provide file sharing among a diverse set of operating system environments within a network setting, it is a model derivative of NFS that allows Microsoft Operating Systems to access shared files on Microsoft servers, UNIX servers, the Web, and most recently, Apple servers. CIFSs are based on

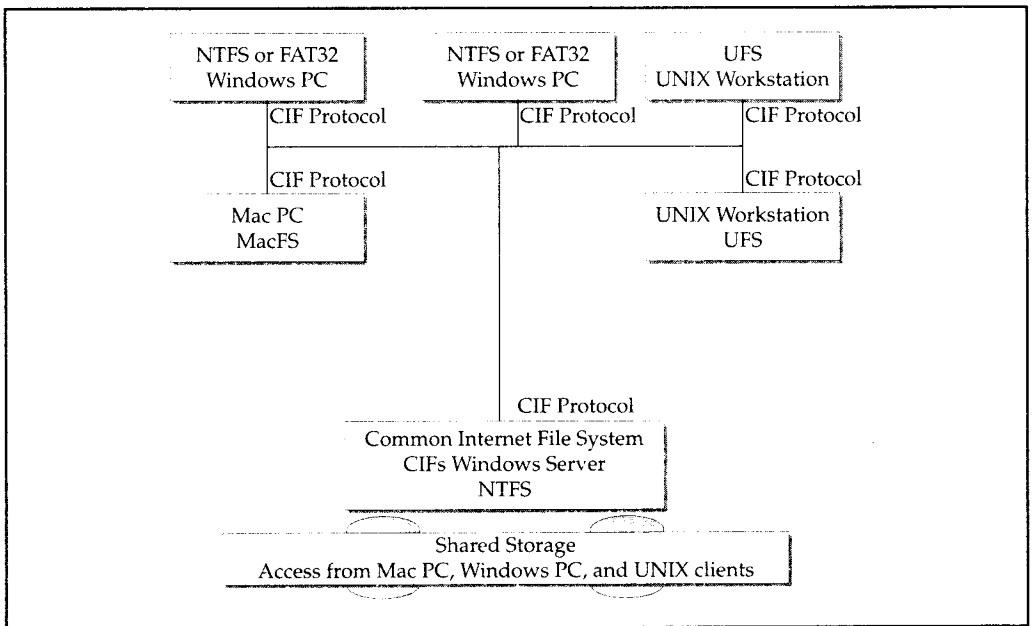

Figure 8-3. *Hypertext Transport Protocol (HTTP)*

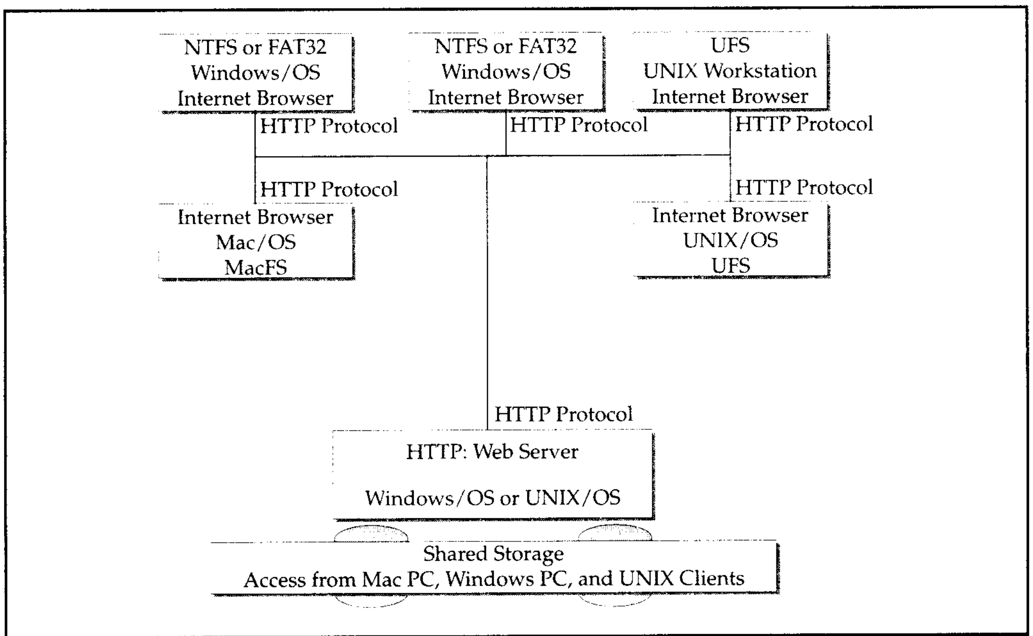

Figure 8-4. *The Common Internet File System (CIFS)*

a Microsoft networking protocol of Server Message Block (SMB), and have both advantages and limitations compared to NFS.

Other file systems are generally derivatives of UNIX variants and are used for specialized applications, such as scientific and academic/research applications. However, one alternative system (as shown in Figure 8-5) has produced a new type of access, especially within networked environments. This is the Direct Access File System, or DAFS. DAFS can be especially important given its use of Direct Memory Access (DMA) functions for transferring data. Future DAFS products will have extremely fast access times given their capability to transfer data between memory locations.

The importance of file systems as they relate to storage systems and infrastructures is straightforward and should not be overlooked. As you encounter more detailed discussions in Parts III and IV regarding NAS and SAN, we will discover another dimension to the importance of file systems and their capability to operate within storage networked environments. As you may have already discovered, NFS, CIFs, and HTTP all play a big role in these environments.

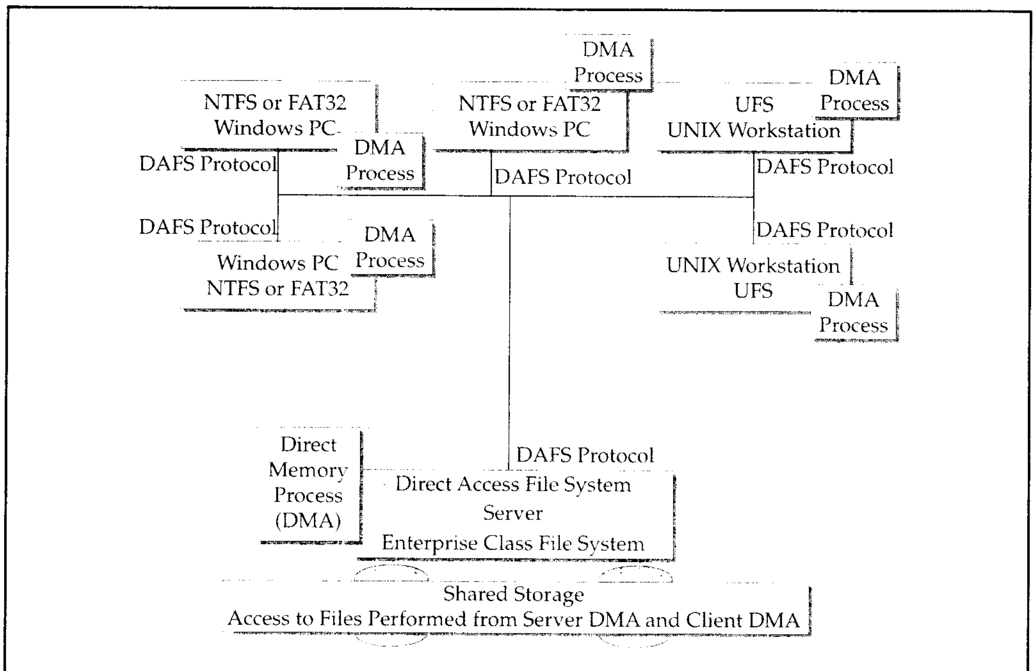

Figure 8-5. *Direct Access File System (DAFS)*

Databases

A database system is made up of individual software components that interact with each other and with other system components, enabling applications access to various abstractions of physical data. Some of the external components include the functionality of file systems, storage cache, and resource manager functions. Other database system components work as an interrelated system of functions to process data requests on behalf of applications. The database's sole purpose is to provide consistent, efficient, and reliable data to end users.

I suppose it's politically correct to substitute "information" for "data" here. However, end users today manipulate more data than the average application programmer did ten years ago. The point is, I don't think end users are intimidated any more regarding the handling, manipulation, and "programmed" access to computer data. The sophistication of computer usage among "smart" users, the available tools to extract, manipulate, and process data, have placed many users into the category of "Super-User," "Query Monster," or "Departmental Programmer." The innovation and continued evolution of the database system has almost singularly provided the impetus for these changes. Fueled by the standard query language, SQL, and its derivatives, where programming has been taken to more accessible levels for all computer users to understand and manipulate on their own, the world has seen an explosion in database implementation, population, and usage.

Looking for Data

A major purpose of a database system is to provide many users with an abstract view of the data. Depending on who you are, you need different levels of abstraction when accessing existing data. IT professionals obviously require different types of access than end users who assume enterprise OLTP and batch applications will provide them with the right information (sorry, I mean data) at the right time. Underlying both IT professional and end-user activities, designing applications, data models, and required maintenance activities is a diversity of data access procedures. The various levels of access and abstraction are described here:

- **Physical Abstraction** Used primarily by system database administrators (DBAs) and system administrators (storage administrators are slowly coming on board with these concepts), physical access is the lowest level of data abstraction. This allows complex low-level data types and structures to be described. It also provides a means to describe how the actual data is stored.
- **Conceptual Abstraction** Used by programmers, application designers, and application DBAs, this is a level of abstraction up from the physical level that describes what user data is actually stored in the database. This also describes relationships that exist within and among the data elements. The conceptual

level allows IT professionals to describe the entire database in terms of simple structures or data models. The physical level uses data models, structures, and descriptions to invoke complex physical structures to develop the database.

- **View Abstraction** Driven by those who derive value from its usage (usually the end user), views are the primary level of abstraction for development of graphical user interfaces (GUIs). End users are not concerned about the entire set of information within the database, or the complexities of its internal structures. As such, view levels become the highest level of abstraction of the data and describe only one part of a database, simplifying the interaction of end users with the database. Depending on the levels of abstraction required, most users require multiple views of the data.

From a storage point of view, it is important to understand not only the physical abstractions of the data, but also the database components that interact with native operating system functions and storage drivers. This becomes increasingly important as database systems push the limits of storage infrastructures unlike any other system or application. However, it is also important to study the translations of abstractions to understand storage-related problems as System and Application DBAs, Administrators, and end users encounter them.

The following are the typical components of a database system:

- **Database Manager** Functions that provide the interface between the physical data and the application programs, integrated application functions such as search and mathematical algorithms, and physical access to the data.
- **File Manager** Functions that allocate space on storage systems and manage the data structures used to represent the conceptual abstraction of data. This component, in many cases, is integrated into the database manager, bypassing the operating systems file system for these functions.
- **Query Processor** A component that translates statements from a query language, such as standard SQL, into low-level instructions that the database manager components will execute on behalf of the end-user query or application program.
- **Preprocessor Compiler** A component that translates database language queries, such as SQL that are embedded in application programs. The compiler turns these into instructions for the query processor, which, in turn, processes it and passes it on to the database manager.
- **Data Language Compiler** Functions that convert data language definitions into data structures called metadata . The Metadata table entries are then stored in a metadata database, sometimes called a data dictionary or repository.

Additional data structures are necessary as part of the physical implementation of the database system. These are the data dictionary or data repositories, as we just discussed.

Finally, index structures provide enhanced search access to data when searching for particular values or data files.

Databases change as data is added and deleted. The collection of data in the database at any given time is called an instance, while the overall design of a database is called the database scheme or schema. The latter is expressed as a conceptual abstraction, and is specified through a data definition language. The results of a data definition schema are placed into a data dictionary or metadata table. A data manipulation language is a set of procedures to access and manipulate the data such as the standard query language or SQL.

Database Systems: Different Animals

There are only a few popular database models that have survived the mainframe-to-client/server-to-Internet evolution. Interestingly, some of the older models continue to process workloads today, albeit mostly on mainframe processing complexes. Databases can be developed from a simple collection of sequential files, sometimes called flat files because of their non-dimensional access restrictions, to the most complex mainframe hierarchical databases. Regardless of implementation, major database management systems have evolved from three conceptual models.

These are the networked databases based on a network data model. Characterized by its capability to provide links within its fields of database records. The hierarchical data model is used by many file systems in their functions to provide faster access to directories and file indices. These are recognizable through their use of tree structures, such as the B-Tree and B+Tree data structures. The third and most popular model is the relational model, which forms the structure for all popular relational databases systems, such as Oracle, Sybase, IBM's DB2, Microsoft's SQL/Server and others. The database model that provides the most challenges, problems, and headaches to storage infrastructures today is the same one that provides the most challenges, problems, and headaches to storage networking infrastructures: the relational database.

Relational database systems consist of a collection of tables, each of which has a unique name. Each field in the table is considered a distinct entity with attributes and has relationships to other entities within its own table as well as to others in the collection of tables that make up the database. User data is populated throughout the tables by individual records. This forms the basis for the tremendous power of a relational database, the ability to compare data elements to perform set mathematical functions, or "What if" questions. From an overall systems perspective, this provides two challenges:

- First is the compute-intensive nature of relational processing. Because relational processing utilizes set mathematics to find the answer to a database query, the data necessary to construct the sets and processing to compute the answer is one of the most resource-intensive activities within data processing.

- Second is the I/O-intensive nature of relational processing. I/O content and activity becomes intensive as the rows of data from tables are loaded into memory to build set constructs to compare data elements until all relational operations are complete. With a database of any size, the compute requirements coupled with the I/O requirements render relational database processing one of the most resource-intensive activities for commercial data processing.

Consequently, the relationship with storage for databases becomes very special. Although there are several considerations when maintaining a storage infrastructure for RDBMS, here are some particularly important points to consider as we move into the area of storage networking.

Relational databases have a legacy from UNIX and mainframe environments. This has resulted in a design whereby most relational database systems (RDBMS) contain their own I/O processing and file management functions. This means that when installed, they bypass the operating systems' file system and utilize their own. This is referred to as an RDBMS that uses a raw disk partition. In effect, they are performing their own I/O and substituting their own file system, essentially mapping the raw disk to their own to enable low-level access. Historically, they had to do this to provide acceptable levels of performance and reliability in the days when UNIX did not have an optimized file system, much less a journaling file system. Given the resource-intensive nature of these relational operations, early RDBMS vendors developed their own I/O systems.

Another important factor to keep in mind is the nature of the relational data structure. It is a table, not a file, a table. The physical data is stored in native block formats for those RDBMSs that use their own I/O and file system, or for those whose physical data is masked from the system when using a standard file system. This means that without special utilities and knowledge of the database, its state, and its metadata, that the ability to reconstruct the data structure from a disruption is impossible. Therefore, databases are difficult, at best, to maintain from a data maintenance aspect (for instance, backup/recovery operations, archival, and volume management).

Databases, especially the relational model, are a challenge to maintain and manage. They have a close relationship with storage systems given their own processing characteristics and development legacy in the area of I/O technologies. Storage networking adds new dimensions to the existing challenges in data organization models and methods. We will explore the role of databases in the world of NAS and SAN in the upcoming sections.