6

TRADITIONAL DATA MANAGEMENT

DATA MANAGEMENT

Electronic data management has a long and rich history dating back to the 1950s. Many data management systems have tried to make their way into the mainstream of information technology, some more successfully than others; hierarchical, network, object, and in-memory are only a few examples. The most successful data management system has been the relational data management technology. For our purposes, I will start at this point to look at its development and what has made it succeed. This is the "yardstick" used to measure how far forward it has moved in comparison to any other data management system.

History

Dr. E. F. Codd, a researcher at IBM, defined the relational model in a 1970 paper entitled "A Relational Model of Data for Large Shared Data Banks,"[20] which initiated a chain reaction of research into the concepts, both internally at IBM and everywhere else. The research timeline included

- 1974—the System/R project gave birth to the Structured English Query Language (SEQUEL).
- 1976–1977—SEQUEL is extended to support multiple users, tables, and so on and later eventually renamed Structured Query Language (SQL).

Distributed Data Management for Grid Computing, by Michael Di Stefano

- 1979—Oracle is launched.
- 1983—IBM introduces DB2.
- Other products are introduced to the marketplace during this same period, including Sybase.

Dr. Codd was also responsible for the mathematical foundation on which SQL is built, namely, relational algebra. SQL is the interface that allows most users to simply access and modify relational data. Through a series of publications in the 1970s and 1980s, Dr. Codd introduced such concepts as the separation of the physical and logical aspects of data management, a model that is understood by a wide user community. The ability to process multiple records simultaneously via an access method that enables ad hoc queries into the data management system has fostered the widespread adoption of the relational model more widely than perhaps any other data management system today.

Features

I will highlight some of the features of a data management system that are essential from the user perspective. I will not go into great detail as to how relational data management systems implement these features as these are complex topics in their own right. Rather, I will discuss these features at a high level including what they are, what they are designed to address, and in some cases, by way of example, highlight how a relational data management system addresses specific features. These topics include

- The mechanics or engine of the data management system
- How data are structured within the data management system
- Data integrity
- Transactional support
- Support for external events and event handling within the data management system
- Backup, recovery, and availability of the data management system
- Security

Each of these features is important, and in some instances one feature plays a supporting role for another feature, as is the case of data integrity. For a more in-depth analysis of these and other topics, the reader can refer to Richard T. Watson, *Data Management, Databases and Organizations*, 4th edition, Wiley, 2004.

Mechanics. The data engine is the part of the data management system that ties the logical representation of data to the physical media that stores the data. It is this engine that binds the data management system to the physical topology of the compute environment that supports it. A generic diagram of an engine consists of the media manager (i.e., the disk manager) and the data container manager (i.e., the file manager) as illustrated in Figure 6.1.

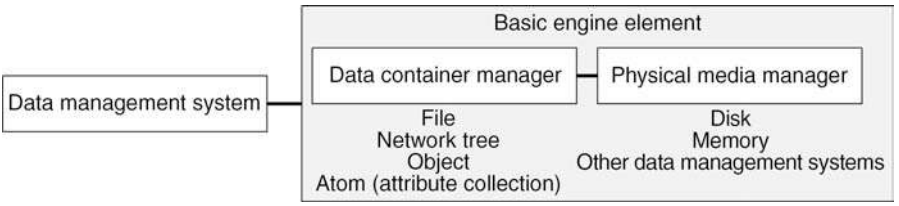

Figure 6.1. Data management system and its engine.

The characteristics of the engine determine the data management system's ability to support the user community for issues such as speed of access, storage of data, scalability, and data replication speed as well as efficiency. The separation of the physical and the logical representations of the data have allowed relational database technology to evolve in terms of increased speed performance and the ability to query and access data without adversely affecting the user community or forcing it to reprogram its data access logic.

I will illustrate that in a distributed compute topology the architecture and the implementation of the engine will have direct consequences for the data management system's ability to support the features and quality-of-service (QoS) levels expected by the users of the system.

Data Structure. In the relational model, the basic data structure is a table that is a two-dimensional structure of rows and columns. Physically, a table is mapped to a file on disk via the engine, which is an integral part of the relational data management system. The relational data management system provides the tools to organize tables so as to describe more complex data relationships. A breakdown of the data structure in the relational model, its extensibility to model complex relationships, and its ability to maintain ease of use is presented in the following text. This combination has lead it to its widespread acceptance as the de facto standard in data management and as a long-term data persistence store.

The relational model starts with a collection of one-dimensional structures made up of rows and columns that form the table. Columns are the fields (attributes) that define the structure of an entity (record) of the table. The fields of the table support basic data types such as numeric, string, date/time, graphic, and Boolean. Records are the rows of the table. Entity uniqueness can be established by creating and maintaining a primary key in the table. The primary key establishes uniqueness of entity identity within the table. Primary keys must not be an empty or have a null value. The primary keys must be unique for each record of the table, and cannot change during the life of the record.

Relationships are established between multiple tables on the basis of common data types or columns of the tables to be joined. The columns establishing the join are called "secondary" or "foreign keys." Matching foreign keys in two tables establishes the relationship and enforces "referential integrity" between the tables. Both primary and foreign keys are discussed in maintaining data integrity

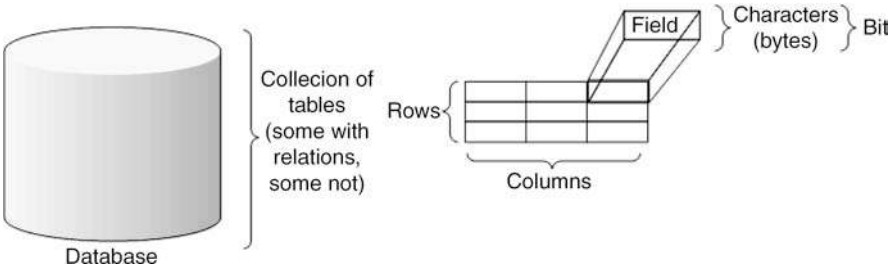

Figure 6.2. Data structure.

later in this chapter. Figure 6.2 illustrates the database made up of collections of tables and the composition of a table.

The logical structure of the relational model is quite simple, starting with the two-dimensional structure of the table (rows and columns) combined with the ability to be joined with other tables so as to extend the complexity of the data management system and enable it to model the more complex data structures as they actually exist in the real world. The basis of the relational model is simple to comprehend and is capable of modeling complex systems, thus enabling a wide range of users to build real-world systems very quickly.

Access. Access to the data in the data management system is primarily through the Structured Query Language (SQL). SQL is a powerful query language that is simple to understand. The learning curve for SQL is very reasonable for a programming-proficient individual to quickly access to the relational data model of tables within the data management system. SQL allows the users to process multiple rows of a table and related tables without requiring an intimate knowledge of the low-level mechanics required to do so.

SQL is a perfect example of the separation between the logical data model and the physical representation that enables developers, programmers, and architects to model data closer to the business without requiring an intimate knowledge of both the physical disk representation and access of the physical disk layer. Through the use of SQL it is simple for the user to

- Create a data scheme is through commands such as “create table”
- Query the data via the simple syntax of a selected statement
- Create relations between tables
- Query tables with multiple relations with simple extensions to the select statement such as the “where” clause
- Modify existing tables via the update and delete statements

Through the simple interface of SQL, the relational model can be accessed and manipulated by a vast number of users. The foundation of SQL is based on an in

sound mathematical principle that allows for the extensibility of the system to represent complex structures.

Integrity. The data going (updated or inserted) into the database has to ensure that the data are correct and accurate; this cannot be done through the data management system—it must be done through the business applications. The applications need to ensure that the data they are inserting into the database are accurate. Once in the data management system, it is the responsibility of that to maintain the integrity of the data. Data integrity is essential for a data management system, since the users must be confident in the quality of the data on which their business is depending in making key business decisions. Otherwise, the data management system would be of little value. In the relational model, data integrity has two levels: entry-level integrity rules and referential integrity rules. The entry-level integrity rule is for primary keys that must be assigned to a valid value. The referential integrity rule, on the other hand, guarantees that the relationships between the tables are valid. So, integrity within the table as well as integrity between tables must be maintained by the data management system.

The end result of data integrity is data availability, data quality, and confidentiality. Each of these outcomes is required and achieved through other key features of the data management system, for example, events, transactions, and backup/recovery/availability of the system.

Additional levels of data integrity begin to enter into the realm of the architects and developers alike. The data management system must support event-driven mechanisms so that when data are inserted, changed, or deleted, these events will enforce the data integrity rules for entity and reference. External to the database, data integrity has to be maintained and manage by the user community, here the adage of "garbage in, garbage out" has special meaning.

Transaction. The data management system must be able to support data transactions internally within the database, with external systems to the database, and with "user programs." Support of transactions integrity is also important to many business applications, to ensure that any of the changes, updates, and deletes are guaranteed and delivered from start to finish (end to end). Typically, transactions are implemented via a technique called a "two-phase commit."

Transactions are used to support and ensure data integrity, which in turn guarantee the user that the data represented in the application are secured all the way down to the long-term physical storage, which in this case would be the database or disk. The transaction is processed so that there exists a recoverable state typically through the use and maintenance of transaction logs, managed by the data management system. A transaction log is a detailed record of all changes that have been applied to the data within a database including information on who made the change, when the change was made, and what the exact change was for audit trail purposes. With this information also being logged, the recovery of the database is both possible and simple. The recovery is merely the reenactment of the steps recorded in the transaction log starting from the "last known good state" of the system.

Events. The data management system must be able to support and manage events, both internal and external to the system. Events are real-time mechanisms, typically user-based, starting as an external event to the data management system that can also trigger subsequent events within the data management system. Events are a tool used to support data integrity as well. For example, referential integrity can be supported through event mechanisms; should a user (with the proper entitlements) delete a record in a table, this deletion will need to be reflected in other tables to maintain referential integrity between two joined tables.

Backup/Recovery/Availability. The data management system must be a reliable resource of information technology. It contains the knowledge required by the business units in order to perform their functions. The quality of the data and availability of the data must be ensured. If this were not the case, then the community would have little confidence in the data management system and would be little value to the business. Therefore, the goal is how to ensure that the data are protected from loss and are available when needed.

First, ability to back up the images of the database makes this possible. The data management system must support methods of backing up the data and the state of the data management system at regular intervals. The database must be recoverable; some methods used to achieve this are "backward recovery." Backward recovery unwinds the state of the database to a previous well-known, high-confidence-state level of the complete data management system. Another method is "forward recovery," which starts from the last well-known state and reapplies the changes and updates to bring in the more current states that existed after the last of the full backup of the system. Both methods require the ability to reprocess transactions.

The availability that the database must meet is driven by the demands of the business for the information contained within to be available and in a well-known reliable state. The system must support the ability for disaster recovery in a quick and efficient manner. Some of the more traditional implementations for data availability are the mirroring of data across multiple instances of the database. One instance is the primary database; the second instance is the fallback/recovery system. The primary and failover/recovery systems are maintained in transactional lock with each other through a mechanism that is implemented by the data management system provider. This technique is called "mirroring" databases. In conjunction with the database mirroring is the ability to determine when the primary system is no longer available to the user community and to take action to reestablish the "service" of the database to the users. Techniques such as high availability (HA) detect full or partial system failures and switching to the redundant systems automatically in a relatively short time so that the user community does not notice the system failure and recovery taking place.

Security. System security is an important aspect for assuring the user community that the system is protecting the data from unwanted eyes. Securing data starts with user entitlements, defining each user with the authority to read, change,

insert, or delete specific data sets within the system. Security models must include the following capabilities at a minimum:

- Identify users and check their authorization levels through the use of user profiles.
- Ensure the security of the data stored on the physical medium. Implementations may include some level of encryption of the data to prevent unwanted interception or "listening" to the data. It may even carry the encryption all the way to the storage medium or disk in an effort to secure the data from programs other than the data management system itself and bypassing the security and access channels of the system by directly accessing the data that reside on the disk.

KEY FOR USABILITY

The usability of any system or product is driven by two factors; simplicity and consistency. The saying "keep it simple" are words to live by and is never more evident than when designing a system intended for a broad consumer audience and to last beyond the moment. Ergonomics needs to be applied to system design so that a broad range of people can understand and effectively use the product. Simple examples of this are in the automotive industry; examples can be found in companies such as General Motors, Ford, Toyota, Mercedes, Ferrari, BMW, and Kia. Yet, just about anyone in the world can get behind the wheel of these products; of complex, intertwined mechanical, hydraulic, and electronic systems; and be able to drive them with very little or no car-manufacturer-specific training. The interface to the automobile is simple and standardized; each automobile must provide the same basic features and functions for it to be usable. The steering wheel, brake and gas pedals, the braking system, the headlights and turn signals, the engine and transmission (automatic or manual, which will then include a clutch pedal and stick shift), seatbelts and airbags, and the position of lease instruments are all in the usual place and function in the usual way.

Other industries are starting to follow the example that has been set by the automotive industry. One such example is in air travel, namely, the small-aircraft industry.[21] In an effort to make the small-aircraft industry safer, cheaper, ubiquitous, and routine, aircraft vendors (and their supporting industry groups) are starting to standardize on instrumentation and controls of the aircraft. So, when moving from one make of aircraft to another, a pilot will not require special training for each of these individual machines. Aircraft companies, taking the lead of the auto industry, can standardize on interfaces. The end result is a consistent operation of the plane from one vendor to another. It also instills a sense of comfort and security for passengers.

Just as with these industries, distributed data management systems must follow the same usability and levels of interface previously established as the de facto standard by the relational data management systems. The majority of data management

systems in use today are relational, and generally have the same basic interface and support the same feature sets. Yet, each vendor may deviate slightly from the other as a way to "differentiate themselves" from the pack. Oracle differs from Sybase, which may differ from DB2; however, they are pretty much the same. The learning curve to go from one to the other is not steep, especially with regard to the basic data management and access features.

Any new data management system such as in the area of distributed computing that is specifically designed to fit a distributed compute topology such as the grid must maintain the same level of ease of user interface and must support, at a minimum, the same feature sets described in this chapter and throughout this book. The basic data structures that the majority of the developers, application architects, and managers understand revolve around the ability to support transactions, data integrity, and security, and most of the system as a whole must be equally usable by the development team and other types of staff currently in place within corporations. Failure to meet this de facto standard that has been set will hinder the adoption of not only the data management system itself but the spread of distributed computing, and grid computing, as a whole.