Foundations and Trends® in
Databases
Vol. 1, No. 2 (2007) 141–259

DOI: 10.1561/1900000002

now
the essence of knowledge

Architecture of a Database System

Joseph M. Hellerstein[1], Michael Stonebraker[2] and James Hamilton[3]

[1] *University of California, Berkeley, USA, hellerstein@cs.berkeley.edu*
[2] *Massachusetts Institute of Technology, USA*
[3] *Microsoft Research, USA*

Abstract

Database Management Systems (DBMSs) are a ubiquitous and critical component of modern computing, and the result of decades of research and development in both academia and industry. Historically, DBMSs were among the earliest multi-user server systems to be developed, and thus pioneered many systems design techniques for scalability and reliability now in use in many other contexts. While many of the algorithms and abstractions used by a DBMS are textbook material, there has been relatively sparse coverage in the literature of the systems design issues that make a DBMS work. This paper presents an architectural discussion of DBMS design principles, including process models, parallel architecture, storage system design, transaction system implementation, query processor and optimizer architectures, and typical shared components and utilities. Successful commercial and open-source systems are used as points of reference, particularly when multiple alternative designs have been adopted by different groups.

1

Introduction

Database Management Systems (DBMSs) are complex, mission-critical software systems. Today's DBMSs embody decades of academic and industrial research and intense corporate software development. Database systems were among the earliest widely deployed online server systems and, as such, have pioneered design solutions spanning not only data management, but also applications, operating systems, and networked services. The early DBMSs are among the most influential software systems in computer science, and the ideas and implementation issues pioneered for DBMSs are widely copied and reinvented.

For a number of reasons, the lessons of database systems architecture are not as broadly known as they should be. First, the applied database systems community is fairly small. Since market forces only support a few competitors at the high end, only a handful of successful DBMS implementations exist. The community of people involved in designing and implementing database systems is tight: many attended the same schools, worked on the same influential research projects, and collaborated on the same commercial products. Second, academic treatment of database systems often ignores architectural issues. Textbook presentations of database systems traditionally focus on algorithmic

and theoretical issues — which are natural to teach, study, and test — without a holistic discussion of system architecture in full implementations. In sum, much conventional wisdom about how to build database systems is available, but little of it has been written down or communicated broadly.

In this paper, we attempt to capture the main architectural aspects of modern database systems, with a discussion of advanced topics. Some of these appear in the literature, and we provide references where appropriate. Other issues are buried in product manuals, and some are simply part of the oral tradition of the community. Where applicable, we use commercial and open-source systems as examples of the various architectural forms discussed. Space prevents, however, the enumeration of the exceptions and finer nuances that have found their way into these multi-million line code bases, most of which are well over a decade old. Our goal here is to focus on overall system design and stress issues not typically discussed in textbooks, providing useful context for more widely known algorithms and concepts. We assume that the reader is familiar with textbook database systems material (e.g., [72] or [83]) and with the basic facilities of modern operating systems such as UNIX, Linux, or Windows. After introducing the high-level architecture of a DBMS in the next section, we provide a number of references to background reading on each of the components in Section 1.2.

1.1 Relational Systems: The Life of a Query

The most mature and widely used database systems in production today are relational database management systems (RDBMSs). These systems can be found at the core of much of the world's application infrastructure including e-commerce, medical records, billing, human resources, payroll, customer relationship management and supply chain management, to name a few. The advent of web-based commerce and community-oriented sites has only increased the volume and breadth of their use. Relational systems serve as the repositories of record behind nearly all online transactions and most online content management systems (blogs, wikis, social networks, and the like). In addition to being important software infrastructure, relational database systems serve as

is established between the client and the database server directly, e.g., via the ODBC or JDBC connectivity protocol. This arrangement is termed a "two-tier" or "client-server" system. In other cases, the client may communicate with a "middle-tier server" (a web server, transaction processing monitor, or the like), which in turn uses a protocol to proxy the communication between the client and the DBMS. This is usually called a "three-tier" system. In many web-based scenarios there is yet another "application server" tier between the web server and the DBMS, resulting in four tiers. Given these various options, a typical DBMS needs to be compatible with many different connectivity protocols used by various client drivers and middleware systems. At base, however, the responsibility of the DBMS' client communications manager in all these protocols is roughly the same: to establish and remember the connection state for the caller (be it a client or a middleware server), to respond to SQL commands from the caller, and to return both data and control messages (result codes, errors, etc.) as appropriate. In our simple example, the communications manager would establish the security credentials of the client, set up state to remember the details of the new connection and the current SQL command across calls, and forward the client's first request deeper into the DBMS to be processed.

2. Upon receiving the client's first SQL command, the DBMS must assign a "thread of computation" to the command. It must also make sure that the thread's data and control outputs are connected via the communications manager to the client. These tasks are the job of the DBMS *Process Manager* (left side of Figure 1.1). The most important decision that the DBMS needs to make at this stage in the query regards *admission control*: whether the system should begin processing the query immediately, or defer execution until a time when enough system resources are available to devote to this query. We discuss Process Management in detail in Section 2.

3. Once admitted and allocated as a thread of control, the gate agent's query can begin to execute. It does so by invoking the code in the *Relational Query Processor* (center, Figure 1.1). This set of modules checks that the user is authorized to run the query, and compiles the user's SQL query text into an internal *query plan*. Once compiled, the resulting query plan is handled via the plan executor. The plan executor consists of a suite of "operators" (relational algorithm implementations) for executing any query. Typical operators implement relational query processing tasks including joins, selection, projection, aggregation, sorting and so on, as well as calls to request data records from lower layers of the system. In our example query, a small subset of these operators — as assembled by the query optimization process — is invoked to satisfy the gate agent's query. We discuss the query processor in Section 4.
4. At the base of the gate agent's query plan, one or more operators exist to request data from the database. These operators make calls to fetch data from the DBMS' *Transactional Storage Manager* (Figure 1.1, bottom), which manages all data access (read) and manipulation (create, update, delete) calls. The storage system includes algorithms and data structures for organizing and accessing data on disk ("access methods"), including basic structures like tables and indexes. It also includes a buffer management module that decides when and what data to transfer between disk and memory buffers. Returning to our example, in the course of accessing data in the access methods, the gate agent's query must invoke the transaction management code to ensure the well-known "ACID" properties of transactions [30] (discussed in more detail in Section 5.1). Before accessing data, locks are acquired from a lock manager to ensure correct execution in the face of other concurrent queries. If the gate agent's query involved updates to the database, it would interact with the log manager to ensure that the transaction was durable if committed, and fully undone if aborted.