Architecture of a Database System
Architecture of a Database System

Joseph M. Hellerstein
University of California
USA
hellerstein@cs.berkeley.edu

Michael Stonebraker
Massachusetts Institute of Technology
USA

James Hamilton
Microsoft Research
USA
Foundations and Trends® in Databases
Volume 1 Issue 2, 2007
Editorial Board

Editor-in-Chief:
Joseph M. Hellerstein
Computer Science Division
University of California, Berkeley
Berkeley, CA
USA
hellerstein@cs.berkeley.edu

Editors
Surajit Chaudhuri (Microsoft Research)
Ronald Fagin (IBM Research)
Minos Garofalakis (Intel Research)
Johannes Gehrke (Cornell University)
Alon Halevy (Google)
Jeffrey Naughton (University of Wisconsin)
Jignesh Patel (University of Michigan)
Raghu Ramakrishnan (Yahoo! Research)
Editorial Scope

Foundations and Trends® in Databases covers a breadth of topics relating to the management of large volumes of data. The journal targets the full scope of issues in data management, from theoretical foundations, to languages and modeling, to algorithms, system architecture, and applications. The list of topics below illustrates some of the intended coverage, though it is by no means exhaustive:

- Data Models and Query Languages
- Query Processing and Optimization
- Storage, Access Methods, and Indexing
- Transaction Management, Concurrency Control and Recovery
- Deductive Databases
- Parallel and Distributed Database Systems
- Database Design and Tuning
- Metadata Management
- Object Management
- Trigger Processing and Active Databases
- Data Mining and OLAP
- Approximate and Interactive Query Processing
- Data Warehousing
- Adaptive Query Processing
- Data Stream Management
- Search and Query Integration
- XML and Semi-Structured Data
- Web Services and Middleware
- Data Integration and Exchange
- Private and Secure Data Management
- Peer-to-Peer, Sensornet and Mobile Data Management
- Scientific and Spatial Data Management
- Data Brokering and Publish/Subscribe
- Data Cleaning and Information Extraction
- Probabilistic Data Management

Information for Librarians
Foundations and Trends® in Databases, 2007, Volume 1, 4 issues. ISSN paper version 1931-7883. ISSN online version 1931-7891. Also available as a combined paper and online subscription.
Architecture of a Database System

Joseph M. Hellerstein\textsuperscript{1}, Michael Stonebraker\textsuperscript{2} and James Hamilton\textsuperscript{3}

\textsuperscript{1}University of California, Berkeley, USA, hellerstein@cs.berkeley.edu
\textsuperscript{2}Massachusetts Institute of Technology, USA
\textsuperscript{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.
Contents

1 Introduction 1
  1.1 Relational Systems: The Life of a Query 2
  1.2 Scope and Overview 7

2 Process Models 9
  2.1 Uniprocessors and Lightweight Threads 12
  2.2 DBMS Threads 19
  2.3 Standard Practice 20
  2.4 Admission Control 22
  2.5 Discussion and Additional Material 24

3 Parallel Architecture: Processes and Memory Coordination 25
  3.1 Shared Memory 25
  3.2 Shared-Nothing 27
  3.3 Shared-Disk 30
  3.4 NUMA 31
  3.5 DBMS Threads and Multi-processors 32
  3.6 Standard Practice 33
  3.7 Discussion and Additional Material 34

4 Relational Query Processor 37
  4.1 Query Parsing and Authorization 38
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2 Query Rewrite</td>
<td>40</td>
</tr>
<tr>
<td>4.3 Query Optimizer</td>
<td>43</td>
</tr>
<tr>
<td>4.4 Query Executor</td>
<td>49</td>
</tr>
<tr>
<td>4.5 Access Methods</td>
<td>54</td>
</tr>
<tr>
<td>4.6 Data Warehouses</td>
<td>57</td>
</tr>
<tr>
<td>4.7 Database Extensibility</td>
<td>63</td>
</tr>
<tr>
<td>4.8 Standard Practice</td>
<td>68</td>
</tr>
<tr>
<td>4.9 Discussion and Additional Material</td>
<td>69</td>
</tr>
</tbody>
</table>

| 5  Storage Management                        | 71   |
| 5.1 Spatial Control                          | 71   |
| 5.2 Temporal Control: Buffering              | 73   |
| 5.3 Buffer Management                        | 75   |
| 5.4 Standard Practice                        | 77   |
| 5.5 Discussion and Additional Material       | 77   |

| 6  Transactions: Concurrency Control and Recovery | 79   |
| 6.1 A Note on ACID                            | 80   |
| 6.2 A Brief Review of Serializability         | 81   |
| 6.3 Locking and Latching                      | 83   |
| 6.4 Log Manager                               | 89   |
| 6.5 Locking and Logging in Indexes            | 92   |
| 6.6 Interdependencies of Transactional Storage| 96   |
| 6.7 Standard Practice                         | 98   |
| 6.8 Discussion and Additional Material        | 99   |

| 7  Shared Components                          | 101  |
| 7.1 Catalog Manager                           | 101  |
| 7.2 Memory Allocator                          | 102  |
| 7.3 Disk Management Subsystems                | 105  |
| 7.4 Replication Services                      | 107  |
| 7.5 Administration, Monitoring, and Utilities | 109  |
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
1.1 Relational Systems: The Life of a Query

A well-understood point of reference for new extensions and revolutions in database systems that may arise in the future. As a result, we focus on relational database systems throughout this paper.

At heart, a typical RDBMS has five main components, as illustrated in Figure 1.1. As an introduction to each of these components and the way they fit together, we step through the life of a query in a database system. This also serves as an overview of the remaining sections of the paper.

Consider a simple but typical database interaction at an airport, in which a gate agent clicks on a form to request the passenger list for a flight. This button click results in a single-query transaction that works roughly as follows:

1. The personal computer at the airport gate (the “client”) calls an API that in turn communicates over a network to establish a connection with the Client Communications Manager of a DBMS (top of Figure 1.1). In some cases, this connection

Fig. 1.1 Main components of a DBMS.
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 process-
ing 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 com-
munications 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 appro-
priate. 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 out-
puts are connected via the communications manager to the
client. These tasks are the job of the DBMS Process Man-
ager (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.
6  Introduction

In Section 5, we discuss storage and buffer management in more detail; Section 6 covers the transactional consistency architecture.

5. At this point in the example query’s life, it has begun to access data records, and is ready to use them to compute results for the client. This is done by “unwinding the stack” of activities we described up to this point. The access methods return control to the query executor’s operators, which orchestrate the computation of result tuples from database data; as result tuples are generated, they are placed in a buffer for the client communications manager, which ships the results back to the caller. For large result sets, the client typically will make additional calls to fetch more data incrementally from the query, resulting in multiple iterations through the communications manager, query executor, and storage manager. In our simple example, at the end of the query the transaction is completed and the connection closed; this results in the transaction manager cleaning up state for the transaction, the process manager freeing any control structures for the query, and the communications manager cleaning up communication state for the connection.

Our discussion of this example query touches on many of the key components in an RDBMS, but not all of them. The right-hand side of Figure 1.1 depicts a number of shared components and utilities that are vital to the operation of a full-function DBMS. The catalog and memory managers are invoked as utilities during any transaction, including our example query. The catalog is used by the query processor during authentication, parsing, and query optimization. The memory manager is used throughout the DBMS whenever memory needs to be dynamically allocated or deallocated. The remaining modules listed in the rightmost box of Figure 1.1 are utilities that run independently of any particular query, keeping the database as a whole well-tuned and reliable. We discuss these shared components and utilities in Section 7.
1.2 Scope and Overview

In most of this paper, our focus is on architectural fundamentals supporting core database functionality. We do not attempt to provide a comprehensive review of database algorithmics that have been extensively documented in the literature. We also provide only minimal discussion of many extensions present in modern DBMSs, most of which provide features beyond core data management but do not significantly alter the system architecture. However, within the various sections of this paper we note topics of interest that are beyond the scope of the paper, and where possible we provide pointers to additional reading.

We begin our discussion with an investigation of the overall architecture of database systems. The first topic in any server system architecture is its overall process structure, and we explore a variety of viable alternatives on this front, first for uniprocessor machines and then for the variety of parallel architectures available today. This discussion of core server system architecture is applicable to a variety of systems, but was to a large degree pioneered in DBMS design. Following this, we begin on the more domain-specific components of a DBMS. We start with a single query’s view of the system, focusing on the relational query processor. Following that, we move into the storage architecture and transactional storage management design. Finally, we present some of the shared components and utilities that exist in most DBMSs, but are rarely discussed in textbooks.
References

References


119

References

References


References


References


Full text available at: http://dx.doi.org/10.1561/1900000002


Full text available at: http://dx.doi.org/10.1561/1900000002