Databases:
The Integrative Force in Cyberspace

Andreas Reuter

EML Research gGmbH, Heidelberg
Andreas.Reuter@eml-d.villa-bosch.de

Abstract. Database technology has come a long way. Starting from systems that were just a little more flexible than low-level file systems, they have evolved into powerful programming and execution environments by embracing the ideas of data independence, non-procedural query languages, extensible type systems, automatic query optimization (including parallel execution and load balancing), automatic control of parallelism, automatic recovery and storage management, transparent distributed execution—to just name a few. Even though database systems are (today) the only systems that allow normal application programmers to write programs that will be executed correctly and safely in a massively parallel environment on shared data, database technology is still viewed by many people as something specialized to large commercial online applications, with a rather static design, something substantially different from the "other" IT components. More to the point: Even though database technology is to the management of persistent data what communication systems are to message-based systems, one can still find many application developers who pride themselves in not using databases, but something else. This is astounding, given the fact that, because of the dramatic decrease in storage prices, the amount of data that needs to be stored reliably (and retrieved, eventually) is growing exponentially—it's Moore's law, after all. And what is more: Things that were thought to be genuinely volatile until recently, such as processes, turn into persistent objects when it comes to workflow management, for example.

The paper argues that the technological evolution of database technology makes database systems the ideal candidate for integrating all types of objects that need persistence one way or the other, supporting all the different types of execution that are characteristic of the various application classes. If database systems are to fulfill this integrative role, they will have to adapt to new roles vis-a`-vis the other system components, such as the operating system, the communication system, the language runtime environment, etc. but those developments are under way as well.

T. Härder and W. Lehner (Eds.): Data Management (Wedekind Festschrift), LNCS 3551, pp. 3-16, 2005.

1 Introduction

Databases have a tenacious habit of just not going away. This is true for the real databases, disks, tapes, software, etc. that are run by big applications; those databases, through the sheer mass of data accumulated have an inertia that reliably protects them from being "moved", technologically, platform-wise or in any other sense. And the same observation holds (albeit for different reasons) for the scientific discipline of database technology. Starting in the mid-80s, database researchers have been organizing workshops and/or panel discussions once every three to five years, fully devoted to the question of whether there is anything interesting left in the database arena. Mike Stonebraker in particular loved to contemplate database-related questions such as "Are we polishing a round ball?" [12]. The motivation for this is quite clear: After having solved all the issues that once were considered interesting and hard (or so it seemed), is the field going to lie fallow—which should trigger everybody to move on to something more fertile, more promising.

The answers suggested in those workshops were mixed—as you would expect. But even those people who still found interesting problems to work on had to admit, that most of those problems were highly specialized compared to the early problems in access paths, synchronization, and all the rest. So based on the discussions in those workshops one would have expected database technology to reach a saturation level quite soon with only marginal improvements in some niches.

Therefore, it is exciting to see that database technology has undergone a transformation during the last couple of years, which few people have predicted to happen this way—even though it is embarrassingly obvious in hindsight. Not only have databases been extended by a plethora of new data types and functions (this was what everybody expected), they rather have mutated from a big but separate system platform for passively storing data into a compute engine supporting a wide range of applications and programming styles. The once-hot debate about object-oriented databases has been settled completely by relational databases absorbing objects, complete with a powerful and efficient extensibility platform, and by database objects becoming first-class citizens in object-oriented languages through a common runtime system [4]. This has largely solved another "old" database problem, the so-called impedance mismatch between relational databases with their SQL-style, declarative programming model and the inherently procedural nature of many programming languages.

But not only have database systems adopted features from other system components; at the same time they have offered their own specific features such as set-oriented programming, parallel programming, content-addressability, etc. as components to a generic programming environment of modern systems.

All this still sounds very technical; one might say that the most annoying, long-standing difficulties have been resolved, that the database people finally got it right. But so what? Why would this justify the claim expressed in the title? The article will argue that databases, as a result of the evolution sketched above, will—in future systems—play a role that is very different from their traditional low-level, obscure role somewhere deep down in the system.