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.

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.

2 New data sources, new usage patterns

Modern database technology has its roots in business data processing. In his textbook on "Data Organization" [14], which was published in the early 70s and describes databases as an emerging technology, Hartmut Wedekind characterizes databases as the core of management information systems, designed to accommodate different structural and operational schemes of a company—and to provide a query interface. Application areas such as banking, inventory control, order processing, etc. were the driving forces behind database products in the 60s and 70s—and to a certain degree they still are. But all the time databases were considered some kind of passive, low-level infrastructure (something close to the operating system) the only purpose of which was to enable new types of applications, integrated management of data for both online and batch processing, and a whole range of functions for archiving, recovery, etc. Databases were systems for application developers, not for end-users. Even the query interfaces based on different types of "user friendly" metaphors required a fairly educated user—they had to understand the schema (their particular view of it), for example. This is the reason why Paul Larson, senior researcher at Microsoft is quoted saying "Database systems are as interesting as the household plumbing to most people. But if they don't work right, that's when you notice them." This is the view we have come to accept: Databases are definitely necessary, but not particularly attractive—let alone exciting. Jim Gray [6] likes to say that "databases are the bricks of cyberspace". This again emphasizes the importance and the substance of the subject, but bricks are not very attractive either. One could find many other such statements, but why worry about this in the first place?

Let us explain the reason by considering a very instructive analogy: In the early 1980s, one would have ascribed the same properties as quoted above to messaging and distributed systems—necessary yet unappealing. But then something happened that made Butler Lampson say that the greatest failure of the systems research community over the past ten years was that "we did not invent the Web" [10]. And if you think about it: The technology enabling the WWW is exactly what was developed by the "systems research community", but in contrast to what distributed systems were before, the Web is attractive, and it made distributed computing technology available to everybody, not just to technical people. So in order to understand / anticipate what database technology might—or rather: should evolve to, it is essential to understand what caused the transformation of distributed systems from something nerdy and boring into the hottest thing in the IT arena—and beyond.

Obviously, the key point was that the Web offered the possibility of reaching beyond the confines of just one system, of unlimited connectivity on a global scale. True, the underlying Internet had been around for quite a while when the Web came up, but the TCP/IP-stack is not something many people can or want to develop applications on. HTML was not a nice interface either, but it worked on a metaphor people are easily familiar with, i.e., documents, rather than something arcane like a communication protocol. And it offered a totally new quality, the possibility of sharing contents without any restrictions, and without the need to worry about the internal workings. So if a new solution significantly increases the users' power (be it in terms of functionality, or reach, or speed), the quality of the interface initially does not matter too much. Many

experts predicted that the Web would never succeed because of the awkwardness of HTML; the same predictions were made for SMS. The experts were wrong on both counts. On the other hand, nice interfaces do not translate into ready success if they do not sufficiently empower the user. The quality of simple, unrestricted access to HTML documents quickly created new usage patterns and new applications. Sharing of text was augmented by function sharing, simple access grew into more sophisticated processing patterns, organizational layers such as portals were introduced, overlay networks came into use, etc. Again, none of these things represented a genuinely new technology, but the way the existing technology was used and employed made all the difference.

Now the reader might ask: Where is the analogy to databases? Clearly, modern database systems have much more powerful and high-level programming interfaces than the TCP/IP protocol stack, there are databases embedded into end-user-oriented application generators, database systems can handle distributed processing in a transparent manner—so what can be learned from the comparison with the Web?

The answer can best be illustrated through an anecdote: At a workshop on future research issues in databases that was held in 1992, about 40 researchers presented their views on great new ways of improving database technology. Towards the end one of the organizers asked a question: "You all have to manage a lot of data, contact addresses, email, project-related data, references and the like. How many of you are using a database system for that purpose?" Two hands went up, while the others eagerly explained how database systems were too complicated to set up and to maintain, how the data types were not adequate, how the integration with other tools was insufficient, and many other such things. This was more than 10 years ago, but had the survey been conducted today, the outcome would certainly not have been much different, because the reasons have not changed. We store, manipulate and query data in many contexts, for a large variety of purposes. Some data is strictly private; other data is shared to some degree. Some data is transient, other data is (semi-) permanent. Each type of data, however, is used and supported by a specific tool: The email end everything pertaining to that lives in the mail system; the appointment data lives in the calender system; project-related data lives in the planning tool; data that requires some calculation may live in a spreadsheet; shared data lives in a company-wide database; and finally, one might have one's own database application (using, for example, Access) for keeping track of the CDs, records, and books. But all those systems are separate as far as data management is concerned. Clearly, for any given project, data related to that project will be found in the mail system, in the calender, in the planning tool, and in some spreadsheets. But there is no way of dynamically creating a "view" on the project that encompasses data from all these sources. Or put it another way: Even if the mail system were built on top of an SQL database (which most mail systems still aren't), there would still be no way of querying that database together with the company database, even if the data types were overlapping. That explains why we are still using a specific tool for every relevant purpose rather than managing all our data in one consolidated store—even if those tools force us into storing the same data redundantly, into representing the same kind of information in different formats, and into manually synchronizing the different versions of the same information—if we ever bother to do so.

Table 1: Overview of categories of personal data and the types of (technical) management support

Category	Tool/Platform	Properties of data store	Data model	Ref. to other categories
Mail	Email system	Closed file system or database	Folder hierarchy; weakly structured text strings	Many: structural; value-based; concept-based
Addresses	Mail system or directory	Closed or open file system (LDAP)	Quasi-relational; various „standards“	Many: structural; value-based
Appointments	Calender system	Closed file system	Hierarchy of time intervals; unstructured text strings	Many: structural; value-based; concept-based
Scheduling	Planning tool	Closed file system or database	Dependency graph; weakly structured text strings	Many: structural; value-based
Budgeting	Spreadsheet	Closed file system or database	Array; arithmetic expressions; unstructured strings	Various: value-based
Personal inventory	4GL tool	Open database	Relational	Various: value-based
Personal finance (account mgmt.)	Web frontend to bank application	Closed database	Forms-oriented	Many: value-based
Personal finance (invoices, receipts)	Shoebox, paper folder	n/a	n/a	Many

This situation, which characterizes the “state of the art” in managing a person’s private data, is summarized in Table 1.

The above table is far from complete; our personal data “ether” comprises many more categories: Messages from mobile phones and PDAs, insurance contracts and related claims, medical data, tickets, photos, and many more. Thanks to the advances in (mobile) communication technology, ever more powerful data sources are entering the personal domain. But the table suffices to clarify one important fact—a fact so trivial that it is mostly overlooked, or regarded as irrelevant: When it comes to our personal data, we have to deal with many different systems and technologies, ranging all the way from advanced Web services down to paper in boxes. The electronic tools are strictly categorized, using different platforms, different data models, and different engines. As a consequence, integrating related data from different categories is not supported by any of the participating tools and has to be done either by the owner of the data or—more likely—is not done at all. This means both a significant loss of information, as is suggested by the shaded column, and a high level of redundancy.

Table 1 also shows that most of the tools involved do employ database technology somewhere deep down in the system. But even if a tool uses a full-fledged SQL system, it restricts its functionality to its own needs, which means the database's capabilities of selecting, aggregating and joining data cannot be used for integrating the tool's data with those from other sources. It is ironic that many tool developers use this observation as some kind of reverse argument, saying that they build their tool on top of a normal file system instead of database system, because "we don't need all these features". But obviously, considering Table 1, the lack of those features, especially those helping with data integration, is causing a major problem for almost everybody. And the other argument that is very popular with developers, "our customers never asked for that", does not count; hardly anybody asked for the Web before it became available.

So the private domain is a large application (i.e., there are many users) where data of different types have to be managed, some of them in collaboration with other parties. If we view data integration as one of the key reasons for using a database, then here is a big task for database systems, a task they do not fulfil today—even though all the technology needed is there. But this observation holds for other areas (outside the household) as well; let us briefly review some of the more demanding new database applications and usage patterns, again without any claim of completeness.

2.1 Science

In many fields of science, such as astronomy, biology, particle physics, etc. measurement devices ranging from satellites to sequencers and particle colliders produce huge amounts of raw data, which have to be stored, curated, analyzed and aggregated in order to become useful for scientific purposes [7]. The raw data is only partially structured, with some parts that conform to the relational model, but with other parts as well, such as images of many different types, time series of measurements, event logs, and text fields that either contain natural language or some kind of application-specific vernacular [13]. The key properties of those data collections (irrespective of the many differences) are:

- The raw data is written once and never changed again. As a matter of fact, some scientific organizations require for all projects they support that any data that influence the published results of the project be kept available for an extended period of time, typically around 15 years.
- Raw data come in as streams with high throughput (hundreds of MB/s), depending on the sensor devices. They have to be recorded as they come in, because in most cases there is no way of repeating the measurement.
- For the majority of applications, the raw data is not interesting. What the users need are aggregates, derived values, or—in case of text fields—some kind of abstract of "what the text says".
- In many cases, the schema has hundreds or thousands of attribute types, whereas each instance only has tens of attribute values.

- The schema of the structured part of the database is not fixed in many cases. As the discipline progresses, new phenomena are discovered, new types of measurements are made, units and dimension are changed, and once in a while whole new concepts are introduced and/or older concepts are redefined. All those schema changes have to be accommodated dynamically.

Digital libraries belong into this category, too. Traditionally, libraries were treated as something different, both organizationally and technically, but in the meantime it no longer makes sense to separate them from the core business of storing and managing scientific data, because whatever ends up in a scientific library—article, book, or report—is to some degree based on scientific data, which thus should be directly linked with the publications they support [6].

2.2 Data streams

There is a growing number of applications where databases are used to create a near-real-time image of some critical section of the environment. For example,

- RFIDs support tracking physical parts from the supplier, through the production process, into the final product—until they need to be replaced for some reason;
- the activities of cell phones can be tracked both with respect to their physical location and the calls they place and receive;
- credit card readers allow tracking the use of credit cards and their physical locations;
- sensors allow monitoring processes of all kinds: in power plants, in chemical reactors, in traffic control systems, in intensive care units, etc.

The main purpose of such databases is to provide flexible query functionality, aggregation and extrapolation of the respective processes that can't properly be achieved on the physical objects. Based on those complex evaluations, one can support process optimization, fraud detection, early-warning functions, and much more.

For that purpose, the database must be able to absorb the data at the rates of their arrival. But the situation is different from gathering scientific data, where the streams typically run at a fairly constant speed. For monitoring applications, the system must be able to accommodate significant fluctuations in the data rate, including sharp bursts. And in addition, the data must not simply be recorded. Rather, the incoming data has to be related to the existing data in complex ways in order to compute the type of derived information that is needed for, say, early warning applications. This gives rise to the notion of continuous queries [1], the implementation of which requires mechanisms quite different from classical database algorithms—and different data structures as well.

An important application of this type of processing is the publish-subscribe scenario. Users can subscribe to certain patterns in the incoming data stream and/or to certain events related to them, which are expressed as complex (continuous) queries on the database. Depending on the application, there can be millions of subscribers using thousands of different queries. Subscribers need to be notified of relevant changes in the in-

coming data in real time, so in case of many subscribers there is a correspondingly huge stream of outgoing data, i.e., notification messages.

Another characteristic property of monitoring applications is the fact that they often track properties of certain objects in space and time. Space is not necessarily the normal 3D space in which we move about, but at any rate, both space and time need to be first-class citizens of the data model rather than just another set of attributes. References to value histories must be supported at the same level as references to the current value, which is what databases normally do.

And finally, the applications require the database to handle (and to trigger) events.

2.3 Workflow management

Automatic management of complex, long-lived workflows has been a goal for at least three decades [11]. The problem has been tackled from different angles, but so far only partial solutions for special cases have been implemented. There is consensus, though, that database technology has to be at the core of any general-purpose solution. Each workflow instance is a persistent, recoverable object, and from that perspective is similar to "traditional" database objects. On the other hand, workflows have many additional features that go beyond what databases normally support.

A workflow has a very complex internal structure that is either completely described in the workflow schema, or that can change/evolve over time. The latter is particularly true for workflows with a very long duration, because it is impossible to fully structure them in the beginning. Workflows are active objects, as opposed to the passive view that databases normally hold of their objects; workflows react to events, to interrupts, they wait for pre-conditions to become true, they trigger events, etc. Workflows have a huge amount of state (which is why databases are needed), partially ordered (as defined by the schema) by activation conditions, by the temporal dimension, and many other criteria. Workflow variables need to maintain their instantiation history, because references to an earlier execution state are required both for normal execution as well as for recovery purposes.

Another aspect of workflows that can be supported by database technology is synchronization of concurrent activities. Because workflows are long-lived, there will be a large number of them executing in parallel, accessing shared data, competing for resources, creating events that may be conflicting in various ways. Some of those conflicts may not be resolvable immediately, so the conflicting state together with the resources involved has to be stored in a recoverable manner such that automatic or application-dependent conflict resolution can be initiated at the proper time.

We could discuss more areas with novel requirements in terms of data management, but the ones mentioned above suffice in order to make the key point: We see an increasing need for consolidating the management of all kinds of data for all kinds of processing patterns on a single, homogeneous platform—whatever the name of the platform may be. People want to manage all their personal data in a consistent way, creating much more than just a "digital shoebox"—the realm of personal data may well extend

into the professional domain, depending on the way people organize their lives [2]. In the scientific domain, we see a convergence of storing scientific data (experimental measurements), the outcome of all types of analyses, and the final publications, including patents and the like. And in the business domain, there is a clear movement towards integrating the management of business data and the management of applications working on those data.

Traditionally, all these fields were treated separately, with different underlying concepts and theories, different techniques, and different technical and scientific communities. Databases were viewed as representatives of the world of structured data (and still are, to a certain degree), whereas collections of text were the subject of "information retrieval systems". The notion of "semi-structured" systems [5] tried to bridge this gap, but still convergence has not been achieved. In the same vein, temporal databases, active databases, real-time databases, etc. have been viewed as different communities, focused more on solving their particular problems rather than trying to come up with a framework for integration. This definitely made good sense because solving the integration problem definitely is a tall order.

Right now it is ironic to see that many people believe in a very simple recipe for integration: XML. As Gray observes in [8], the set of people believing in this approach and those not buying it is stratified by age—yet he continues to say "... but it is hard at this point to say how this movie will end."

3 Technological trends

When sketching technological trends that will be useful in solving the problems outlined above, we have to consider the database field as well as adjacent areas—whatever measure of "nearness" one may choose to apply. This could result in a fairly lengthy list of new ideas and techniques, which would be beyond the limitations of this paper. Therefore, we will only name some of the key technologies expected to be instrumental in extending the scope of databases such that they can support the novel applications and usage patterns that already have emerged—and that will keep emerging in the future. Since we cannot discuss any of the technological trends in detail, we will only use them to support our core argument that all the ingredients are there (or at least a sufficient number is) to unleash the integrative power of database technology.

3.1 Trends in database technology

It is hard to judge which technological change is more important than another one, but clearly one of the most consequential developments in database technology in recent history has been the integration of the relational model (one should rather say: the SQL-model) with object technology. The model-specific problems aside, this required an extension of the database systems' runtime engine in order to accommodate the dynamic nature of object-orientation—as opposed to the static characteristic of a relational sche-

ma. In the end, this led to an integration of the databases' runtime engine with the runtime systems of classical programming languages, which greatly enhanced the capabilities of both worlds: The traditional "impedance mismatch" between the database operators and the programming languages they are embedded in largely disappeared. One can run code in (almost) any language inside the database, and / or one can include database objects into class definitions of an object-oriented programming language. Given the powerful declarative programming model of SQL, one can use a database as an execution environment for procedural, object-oriented, declarative, and rule-based programming, whatever fits the problem structure best—all within one coherent framework. This in itself is an extremely useful basis for integrating different data models and execution patterns, as is illustrated by the current work on integrating text into classical databases—two domains that have traditionally been quite separate.

The other important development has to do with the proliferation of methods and techniques for indexing, combining and aggregating data in any conceivable manner. Databases in the meantime efficiently support cubes of very high dimensionality with a vast number of aggregation operators. They also include machine-learning techniques for detecting clusters, extracting rules, "guessing" missing data, and the like. Novel indexing techniques help in supporting a variety of spatial and temporal data models—or rather: embeddings of the underlying data models. All this is accompanied by advanced query optimizers that exploit the potential of those access paths and dynamically adapt to changing runtime conditions. Again, those changes are essential for the task of integrating everything in cyberspace.

Integrating the ever-growing volume of data requires, among many other things, a database system that is highly scalable and can be tuned to the specific performance needs of a wide range of applications. Modern database systems respond to these needs by supporting new internal storage structures such as transposed files (aka column stores), by exploiting the potential of very large main memories, by using materialized views and others types of replication, by applying a rich set of algorithms for computing complex queries, etc.

3.2 Trends in adjacent fields

The new developments in database technology have either been provoked by or complemented by new approaches in related fields. Whether it was the competitive or the symbiotic scenario does not matter, it is the result that counts.

A key development in the field of programming languages and systems is the notion of a common language runtime environment [2], which allows for a seamless integration of database functionality and programming languages. It also enables database systems to schedule and execute application programs autonomously, i.e., without the need for a separate component like a TP monitor. This also means that database systems can provide Web services without the necessity of a classical application execution environment.

The consequences of distributed computing, rapidly increasing storage capacities, demands for non-stop operation (to name just a few) have caused operating systems and

other low-level system components to adopt database techniques. For example, most operating systems now support ACID transactions in some form, and they offer recovery functionality for their file systems similar to what database systems provide.

Devices other than general-purpose computers increasingly employ database systems. There are two main reasons for that: The first one is to have the device expose a powerful standard interface (SQL) instead of an idiosyncratic, device-specific interface. The other reason is that the device has to keep large amounts of data, which are most easily managed by a standard database system. An example of the latter category is a home-entertainment device that can store hundreds of CDs and provide the user with sophisticated search functions.

The implementation of workflow engines requires an even closer collaboration between database systems and operating systems. The reason is obvious: A workflow is a long-lived recoverable execution of a single thread or of parallel/interleaved computations, so everything that is volatile information for normal OS processes now has to be turned into persistent objects, just like the static objects that are normally stored in databases. Examples for this are recoverable queues, sequences of variable instantiations for keeping track of execution histories, etc. Many database systems support queues as first-class objects, so in a sense the database is the real execution environment of workflows, with operating systems acting only on its behalf by providing expendable objects such as processes and address spaces.

A last important trend is the adoption of schema-based execution in many areas. Databases have had schemas (albeit rather static ones) all along, and so had operating systems. Markup languages made messages and documents schema-based, a development that led to Web services, among other things. Similar ideas can be found in many application systems, where the generic functionality is adapted to the needs of a specific user by populating various "schema" tables with the appropriate values. This process is often referred to as customization, and—just like that classical notion of a schema—it is an example of the old adage that "any problem can be solved by introducing just another level of indirection." And, of course, ontologies as a means of extracting concepts from plain text can be viewed as yet another incarnation of the same idea.

4 The shape of future database systems

This section will not give a description of what the title refers to—that would be way too ambitious. We will rather summarize the observations from the previous chapter and, based on this, identify a number of aspects that will determine the shape of future database systems.

But first let us re-state the assumption that this paper is built on: Due to the availability of virtually unlimited storage at low cost[1], data from a large variety of sources will be gathered, stored, and evaluated in unforeseen ways. In many application areas,

1. This development is expected to be complemented by the availability of a high-bandwidth mobile communication infrastructure.

those data collections will establish ever more precise images of the respective part of reality, and those images will be more and more up-to-date. So for many purposes, decisions will not be based on input from the "real world", but on query results from the digital images. Examples of this have been mentioned above.

Since this scenario talks about managing large amounts of data, we will consider it a database problem, even though there is no guarantee that the technology finally supporting such applications will not be given some other name—probably because "database" does not sound cool enough. Anyhow, database systems capable of integrating data of all types and supporting all kinds of processing patterns will have to be extremely adaptive, both in terms of representing the data and in terms of interacting with the environment—which can be anything from sensor devices to applications programs and users. They must not enforce structure upon the data, if there is no structure, or if any structure that can be identified is likely to change. In those cases, schema modifications (together with schema versioning) must be supported as well as (dynamic) schema transformation [3]. In other cases, ontology-based representations might be a better option—XML will certainly be the lowest level of self-descriptive data. Classical applications will still use their more or less static schema descriptions, and in an integrated system, all those techniques must be available at the same time, allowing queries to span different models. For some data it must be possible to have it encapsulated by an object, but make it accessible for certain types of queries as "raw data" as well. Many standard tools and applications organize their data in a hierarchical fashion (e.g. folders), so this kind of mapping must be supported—whether or not the data structure is inherently hierarchical.

Future database systems will, because of the integrative role they have to assume, have to deal with high levels of parallelism and with requests ranging from milliseconds to months or more. This requires new synchronization primitives and new notions of consistency—beyond those implied by the classical ACID transaction model. Most likely, such extensions will be developed in collaboration with researchers from the fields of programming languages, dependable systems, and maybe even hardware architects [9].

Another consequence of the integrative role is the necessity of keeping the system up at any time, under all circumstances. Any tuning, reorganization, repair, recovery or whatever has to be performed automatically, in parallel to normal execution. This demand for a self-organizing, self-healing, self-you-name-it database is a subject of ongoing research, and its feasibility will be determined by technical as well as economical constraints.

As was mentioned above, it is not clear if the resulting solution will be perceived as a database system (however "future" it may be), or if it will be dressed up in a different fashion. One possible solution is to move a database system with (ideally) all the extensions mentioned into the operating system and build a next-generation file system on top of that. Such a file system would offer the conventional file types as well as XML-stores, semi-structured repositories, stores for huge data streams, object repositories, queues, etc. But they would all be implemented on top of an underlying database system, which would still be able to talk SQL and provide all the mechanisms for consis-

tency, synchronization, recovery, optimization, and schema translation. But again: This is just a possibility, and by no means the only one.

5 Conclusions

The key message of this article is plain and simple: There are many different applications and usage modes out there, some rather old, some emerging, which hold the potential for integration at the level of the data they are dealing with. Everybody, home user as well as professional, would benefit immensely from a technology that enables them to transparently access and manipulate data in such an integrated view. Database technology, together with a host of "neighboring" technologies, has all the components required to do that. All it needs is an innovation comparable to the creation of the Web on top of the Internet. Referring back to Lampson's statement quoted in the beginning, we should ask ourselves (as members of the technical database community): Will we create this future, global, unified repository? If so, what will it look like? If not, why not?

References

[1] Babu, S., Widom, J.: Continuous Queries over Data Streams. in: SIGMOD Record 30:3, Sept. 2001, pp. 109-120.

[2] Bell, G.: MyLifeBits: A Lifetime Personal Store. in: Proc. of Accelerating Change 2004 Conference, Palo Alto, Ca., Nov. 2004.

[3] Bernstein, P.A., Generic Model Management—A Database Infrastructure for Schema Manipulation. in: Lecture Notes on Computer Science, No. 2172, Springer-Verlag.

[4] Common Language Runtime. in: Microsoft .NET Framework Developer's Guide, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconthecommonlanguageruntime.asp

[5] Goldman, R., McHugh, J. and Widom, J.: From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. in: Proc. of the 2nd Int. Workshop on the Web and Databases (WebDB '99), Philadelphia, Pennsylvania, June 1999, pp. 25-30.

[6] Gray, J.: The Laws of Cyberspace. Presentation at the International University in Germany, October 1998, http://research.microsoft.com/%7Egray/talks/1998_laws.ppt

[7] Gray, J., Szalay, A.S. et al.: Online Scientific Data Curation, Publication, and Archiving. in: Proc. of SPIE Astronomy, Telescopes and Instruments, Waikoloa, 2002, pp. 103-107.

[8] Gray, J.: The Revolution in Database Architecture. Technical Report, MSR-TR-2004-31, March 2004.

[9] Jones, C., et al.: The Atomic Manifesto: a Story in Four Quarks. in: Dagstuhl Seminar Proceedings 04181, http://drops.dagstuhl.de/opus/volltexte/2004/9.

[10] Lampson, B.: Computer systems research: Past and future. Invited talk, in: Proc. of SOSP'99.

[11] Leymann, F., Roller, D.: Production Workflow—Concepts and Techniques. Prentice Hall, 1999.

[12] Proceedings ICDE Conference on Data Engineering. Vienna, 1993

[13] Ratsch, E., et al.: Developing a Protein-Interactions Ontology. in: Comparative and Functional Genomics, Vol. 4, No. 1, 2003, pp. 85-89.

[14] Wedekind, H.: Datenorganisation. Walter de Gruyter, Berlin New York, 1975.