Database Trends and Directions: Current Challenges and Opportunities

George Feuerlicht[1,2]

[1] Department of Information Technology, University of Economics, Prague
W. Churchill Sq. 4, Prague, Czech Republic

[2] Faculty of Engineering and Information Technology University of Technology, Sydney
P.O. Box 123 Broadway, Sydney, NSW 2007, Australia
jiri@it.uts.edu.au

Abstract. Database management has undergone more than four decades of evolution producing vast range of research and extensive array of technology solutions. The database research community and software industry has responded to numerous challenges resulting from changes in user requirements and opportunities presented by hardware advances. The relational database approach as represented by SQL databases has been particularly successful and one of the most durable paradigms in computing. Most recent database challenges include internet-scale databases – databases that manage hundreds of millions of users and cloud databases that use novel techniques for managing massive amounts of data. In this paper we review the evolution of database management systems over the last four decades and then focus on the most recent database developments discussing research and implementation challenges presented by modern database applications.

Keywords: Relational Databases, Object-Relational Databases, NoSQL Databases

1 Introduction

Databases, in particular relational databases, are a ubiquitous part of today's computing environment. Database management systems support a wide variety of applications, from business to scientific and more recently various types of internet and electronic commerce applications. Database management systems (DBMS) are a core technology in most organizations today and run mission-critical applications that banks, hospitals, airlines, and most other types of organizations rely on for their day to day operation. Over the last three decades relational DBMS technology has proven to be highly adaptable and has evolved to accommodate new application requirements and the ever-increasing size and complexity of data. But, there are indications that some of the recently emerging data-intensive applications (e.g. internet searches) cannot be satisfactorily addressed using existing DBMS technology, and some experts argue that significant innovation is needed (a new database paradigm) to overcome the limitations of the current generation of database technology.

The combination of inexpensive and high capacity storage and the prevalence of digital devices (digital cameras, sound recorders, video recorders, mobile phones,

J. Pokorný, V. Snášel, K. Richta (Eds.): Dateso 2010, pp. 163–174, ISBN 978-80-7378-116-3.

RFID readers, and various types of sensors) is creating a deluge of digital information. According to a recent article in the Economist [1] the amount of data collected by various sensors, computers, and devices is growing at a compound annual rate of 60%. A 2008 study by International Data Corporation (IDC) predicted that over a thousand exabytes of digital data will be generated in 2010 [2]. Scientific applications in astronomy, earth sciences, etc. (e-science) tend to produce massive amounts of data; well-documented examples include the Large Hadron Collider at CERN [3] that generates 40 terabytes of data every second. Storing and analyzing such volumes of data represents an insurmountable challenge for the current generation of database technology. Another relatively recent development that may require a revision of current database paradigms are internet-scale applications (e.g. search engines, social networking applications, cloud computing services, etc.) that typically process petabytes of data, use thousands of servers, and serve millions of users that demand sub-second access to information. Companies like Google, Facebook, Amazon, and eBay manipulate petabytes of data every day. For example, Facebook handles 20 petabytes of data, managing 20 billion photographs in 4 different resolutions, growing by 2 billion photographs per month. The Facebook database is serving 600,000 photographs per second for a user base of 300 million active users [4]. Google manages vast amounts of semi-structured data: billions of URLs with associated internet content, crawl metadata, geographic objects (roads, satellite images, etc.), and hundreds of terabytes of satellite image data, with hundreds of millions of users and thousands of queries per second [5]. The scale and level of functionality required for such "big data" applications has not been anticipated by commercially available DBMSs, and almost invariably internet companies were forced to develop their own database solutions. But, even more traditional database applications manage increasingly large volumes of data; for example the retail chain WalMart handles more than one million transactions per hour, and manages databases with more than 2.5 petabytes of data.

It is estimated that structured data constitutes only about 5% of the total volume of generated data, with the rest of this "digital universe" in semi-structured or unstructured form, making it more difficult to manage and to extract meaningful information from it. This massive increase in the volume and complexity of data is challenging available database management techniques and technologies, forcing a re-evaluation of the direction of database research. Some fundamental questions arise, including what constitutes a database application. Can applications that search petabytes of unstructured data (e.g. Web pages) using thousands of servers working in parallel be classified as database applications?

In this paper we firstly review the past achievements of database research and technology solutions (section 2), and then discuss the research challenges and opportunities created by new types of database applications (section 3). The final sections (section 4) are our conclusions.

2 Evolution of Database Technology

While the origin of commercial database management systems can be traced to hierarchical and CODASYL (Conference on Data Systems Languages) databases of 1960s and 1970s it was the emergence of relational DBMS during the 1980s that started a revolution in data management. The simplicity and elegance of the relational model proposed by E.F. Codd in 1970 [6] resulted in unprecedented volume of research activity and the emergence of highly successful relational DBMS (RDBMS) implementations. Relational databases are a rare example of a theoretical model preceding and guiding the implementation of technologies. Codd is often credited with turning the previously *black art* of data management into an engineering discipline providing a blueprint for the design and implementation of databases and the foundation of modern database technology. The basic idea of the relational model is to represent data as two-dimensional tables with well-defined properties and to use of a high-level query language for data access. This remarkably simple set of ideas based on the underlying relational theory had a major impact on the development of database technology over the following two decades. Relational databases solved two major interrelated problems of the earlier database approaches. The first achievement was to de-couple the database from application programs by providing effective support for data independence. Second, and equally important achievement of the relational approach was to free database application developers from the burden of programming navigational access to database records by introducing a non-procedural query language.

A number of different relational languages were proposed following Codd's original description of the relational model, notably a language called QUEL (Ingres DBMS) developed at University of California at Berkeley, and IBM's Structured Query Language (SQL) developed at the IBM San Jose Research Laboratory. The next major milestone in the evolution of relational databases was the acceptance by ANSI (American National Standards Institute) of a subset of IBM's SQL as the first version of the standard relational database language - SQL86. Although SQL86 lacked many important features of the relational model as originally proposed by Codd, including key aspects of the model such as referential integrity and domains, it quickly became universally accepted as the database language for relational DBMS systems. The shortcomings in SQL86 were largely rectified in the subsequent releases of the SQL standard (SQL89, SQL92) and SQL has evolved from a relatively simple language into a comprehensive database language implemented in all significant RDBMS products today. Many of the enhancements incorporated into SQL over the last two decades were integral features of the relational model omitted from the earlier standard specifications, other features, such as triggers, role-based security, and stored procedures were retrofitted into the standard as a result of their widespread use in commercial products.

Given the computing environment of the 70s and early 80s, relational databases were initially used for relatively simple business applications running on large mainframe computers; data used in such traditional business applications (e.g. financial and banking) can be structured into tables and stored in a relational database with relative ease. The main concern of early RDBMS implementations was to ensure adequate performance, in particular for online transaction processing (OLTP)

applications. Initially, relational DBMSs had inferior performance when compared with earlier DBMS approaches as SQL uses *expensive* join operations and relies on a query optimizer to determine how to access data records instead of using faster pointer-based navigational access implemented in hierarchical and CODASYL databases. For that reason the main use of relational DBMSs was initially confined to decision support applications that did not involve users waiting for query results online. However, as computer hardware became more powerful and optimization techniques improved, relational systems became the technology of choice in most application environments, including those with stringent response time requirements.

Relational DBMSs proved to be extraordinarily successful in taking advantage of new computing platforms, architectures and environments. The first significant demonstration of the adaptability of relational databases was the extension of the relational model to cover distributed database environments. The origin of distributed relational database was IBM's research project System R* (continuation of the Project R) which addressed distributed database issues including distributed query optimization, distributed transactions, and catalog management. Following on from the System R* database researchers solved most of the problems that concern running applications transparently across multiple databases. Most commercial RDBMSs incorporate a whole range of distributed database features, including reliable (two-phase commit) distributed transactions, optimized distributed queries, and advanced replication facilities. Similarly, relational DBMSs were among the first technologies to support applications with large number of users in distributed client/server environments. This was largely due to the non-procedural nature of SQL, which made it possible for database queries to be packaged and send over a computer network as messages from a client application to a database server. This type of client/server interaction later supplemented with remotely executed database stored procedures using RPC calls (Remote Procedure Calls) enabled the implementation of scalable client/server database applications.

Relational DBMSs were quick to take advantage of the new multiprocessor architectures and provide support for parallel execution of SQL queries. Query decomposition, necessary for parallel execution is made possible by the declarative nature of the SQL language enabling queries to be decomposed into well-defined sub-queries that run in parallel across multiple processors. Parallel SQL was implemented for shared memory, shared-disk, and shared-nothing parallel architectures with excellent performance and scalability. Both distributed and parallel databases benefit from the theoretical underpinning provided by the relational model. As a result of such developments relational databases became the fastest and most scalable commercially available DMBS systems.

2.1 Objects and Databases

RDBMs have shown remarkable ability to take advantage of new computing platforms and continuously improve functionality, performance and scalability to a point where relational databases became the dominant database technology in 1990s, supporting mission-critical environments with tens of thousands of users. However, by mid 90s it became quite clear that the simple data structures and a limited set of

data types that characterize SQL92 relational DBMSs constitute a significant drawback when implementing new types of applications that use complex data. Modern database applications are characterized by four categories of requirements:

(1) need to store and manage large multimedia data objects – images, sound clips, videos, maps, etc.
(2) requirement for database data types to mirror application-level data types, including the ability for users to define their own data types as needed by specific applications
(3) representation of complex relationships, including composition and aggregation, e.g. multi-level component assemblies used in CAD (Computer-Aided Design) and similar applications
(4) need for seamless integration with object-oriented programming languages; with Java in particular

Such requirements are particularly evident in applications that use multimedia data, GIS (Geographical Information Systems), e-science and web applications. Web applications typically contain a whole range of multimedia data types such as textual information, images, video and audio clips, and fragments of program code. Many modern applications require specialized data types, for example GIS applications involve spatial data types (e.g. points, lines, polygons, etc.) and spatial operations (e.g. distance, area, etc.). The initial solution adopted in relational databases to accommodate non-traditional data (e.g. multimedia, GIS, etc.) was to allow the storage of large objects (LOBs) as columns in database tables. However, using this approach multimedia data is treated as unstructured large granularity objects – the data type of the object is not explicitly recognized by the database type system and only very limited processing of the object data is supported.

In addition to the need to store large and complex objects in the database, there is another important requirement that motivated the introduction of object support at the database level. Most modern applications are developed using object-oriented programming languages (i.e. Java, C++, C#) and close integration of the database language SQL with object-oriented programming languages reduces *impedance mismatch* (i.e. differences between the type systems, error handling, etc.) with corresponding improvements in programmer productivity. This requirement, while not new gained urgency with the emergence of Java as a *de facto* standard programming language for internet applications, making it imperative to ensure that Java objects can be easily mapped into database objects.

While there was a wide agreement within the database research community about the need to support objects at the database level, there was a considerable divergence of opinion about how this should be achieved. Two competing approaches emerged: the *revolutionary* approach, seeking to develop a completely new fully object-oriented database solution [7], and the *evolutionary* approach which took the path of adding object features to SQL. In early 90s a number of database management systems were developed ground-up as pure object DBMS (ODBMS) systems with the goal to address the limitations of relational databases by adopting a completely new database model with support for objects with unique identifiers, methods, inheritance, encapsulation, polymorphism and other features commonly associated with object

systems. The basic idea was to build on top of object-oriented programming languages and provide persistence for application objects achieving homogeneous programming environment with close correspondence between application objects and objects stored in the database. This (revolutionary) approach popularized by the Object Database Management Group (ODMG) resulted in the proposal for a new database model and Object Query Language (OQL). As the commercial ODBMS products appeared on the market and attempted to capture market share from the established relational DBMSs, many regarded object-oriented databases as the next generation of database technology destined to supersede relational databases in much the same way as relational technology superseded earlier databases approaches. However, this radical attempt to break with the past has been largely unsuccessful as ODBMSs have not been able to match RDBMS technology in a number of important aspects, including reliability, scalability and level of standardization. Even more importantly, while popular in some niche application areas (e.g. CAD/CAM), object databases have not been able to address the wider requirements of mainstream corporate applications.

As a response to ODBMS enthusiasts a number of influential database researches formed a Committee for Advanced DBMS Function with the objective to define the requirements for the next generation database systems, and published the Third-Generation Database Systems Manifesto [8] as a blueprint for future database development. While recognizing the limitations of relational databases, this important effort argued that the next generation database systems should subsume the existing (second generation) DBMSs and preserve the benefits of relational databases, in particular non-procedural access and data independence. The essential point of difference from the advocates of object-oriented databases was the insistence on natural evolution from the existing relational DBMSs technology, and the implementation of object identity, abstract data types, inheritance, and other object features as relational database extensions.

The evolutionary approach resulted in a new breed of hybrid Object-Relational database technology. In retrospect, Object-Relational databases to a very large extent achieved the original objective of the Third-Generation Database Systems Manifesto, to preserve the benefits of relational database and at the same time to take advantage of object features. However, bringing object features into SQL did not turn out to be an easy task, and the evolutionary approach has struck numerous challenges and produced a number of changes in direction. At a superficial level there seems to be a good match between relations and objects, more specifically the concepts of relational rows and object instances. But, at closer inspection there are deep conflicts between the two models. For example, encapsulation, a key feature of object systems is difficult to reconcile with a database query language, as encapsulated data cannot be queried directly and requires access via methods, imposing unacceptable performance overheads. Various attempts at the unification of relations and objects using concepts such as ADTs (Abstract Data Types) have been proposed and discussed extensively by the ISO WG3 (Working Group 3), the working group responsible for database language standardization, but failed to gain the necessary wide support. After more than five years of intensive work by the WG3 Working Group on Database Languages some of the early ambitious attempts to incorporate object-orientation into the SQL standard were significantly scaled down. The resulting SQL:1999 standard is

a very pragmatic solution, which addresses the main limitation of relational databases by enabling data type extensibility, providing the basis for a rich database type system [9]. The mainstream database vendors (Oracle, IBM, and others) have strongly endorsed the object-relational approach and actively participated in the development of SQL:1999, and most leading DBMS products support object-relational features of SQL:1999.

2.2 XML and Databases

Another challenge to the dominance of relational databases that echoed the efforts to incorporate object support into databases in the 90s arose approximately a decade later with the emergence of XML. XML became the *de facto* standard formatting language for semi-structured data and has been widely adopted in e-business applications as a standard data interchange language and a core standard for Web Services and related technologies. The availability of a standard XML query language XQuery [10], and a standard schema definition language XML Schema [11], and numerous other XML tools and languages (XPath, SAX, DOM, XQL, etc.) provided a basis for the implementation of XML DBMSs. This lead to the development of a number of research prototypes, e.g. Lore [12], XTABLES [13], SilkRoute [14], and some commercial products, e.g. Tamino [15].

Native XML Databases (NXD) that store XML documents in their native format and use XML query languages for retrieval were regarded by some as a new generation of DBMS technology destined to supersede relational DBMS. However, similar to ODBMS, NXD did not replace relational DBMS and remain a solution in niche application domains, mainly in document and content management applications. A detailed analysis of the benefits and limitations of NXD databases is available in [16].

As an alternative to the Native XML Database approach, ORDBMSs (e.g. Oracle) provide a repository functionality called XML native type to store XML documents in the database without any conversion, and support updates, queries, indexing, and views on this data type. Another option available in ORDBMS is to convert XML data into a relational or object-relational form (so called *shredding*) and store the resulting rows of data in corresponding typed tables. The SQL/XML specification [17] that is a part of the SQL:2003 standard defines the XML data type and provides mapping rules between XML schemas and SQL structures, as well as functions that support manipulation of XML data within SQL queries.

3 Database Research Directions

As per our discussion in the previous section (section 2), database research and associated standardization activities have successfully guided the development of database technology over the last four decades and SQL relational databases remain the dominant database technology today. This effort to innovate relational databases to address the needs of new applications is continuing today. Recent examples of database innovation include the development of streaming SQL technology that is

used to process rapidly flowing data ("data in flight") minimizing latency in Web 2.0 applications [18], and database appliances that simplify DBMS deployment on cloud computing platforms [19]. It is also evident from the above discussion that the relational database approach has proven to be extraordinarily durable, and has adapted to new hardware architectures as well as new application requirements, successfully subsuming both object-oriented and XML paradigms. Database research has played an important role in solving key research problems and facilitating rapid technology transfer making DBMS technology one of the most successful efforts in computer science [20]. However, it is equally evident that database research is facing major new challenges due to *explosion of data*, novel usage scenarios, and a major shift in computing architectures. A recent meeting of leading database experts characterized the present situation as a "turning point in database research" and identified a number of trends that necessitate re-evaluation of research directions, and at the same time present new research opportunities. "The Claremont Report on Database Research" [21] identified the following trends:

(1) Big Data (applications that process very large volumes of data, e.g. Web search, e-science, etc)
(2) Data analysis as a profit center (increasing number of companies where the main business is data)
(3) Ubiquity of structured and unstructured data (mainly originating from various Web sources)
(4) Developer demands (as adoption of open source relational DBMS accelerates, developers demand more intuitive programming models)
(5) Architectural shifts in computing (emergence of cloud computing services brings about a fundamental change in software architecture towards parallel clusters of computers; shift away from increasing CPU clock speed to increasing the number of processor cores)

The report goes on to identify the following research opportunities:

(1) Re-design of architecture of database engines, to overcome the limitations of current relational databases (RDBMS provide poor price/performance for many popular applications, including text indexing, serving web pages, and media delivery)
(2) Declarative Programming for Emerging Platforms (support for data independence, declarative programming and cost-based optimization for new programming models, e.g. MapReduce)
(3) The Interplay of Structured and Unstructured Data (managing a rich collection of structured, semistructured and unstructured data, spread over many repositories in the enterprise and on the Web)
(4) Cloud Data Services (improving manageability of cloud databases, Federated cloud architectures, etc.)
(5) Mobile Applications and Virtual Worlds (manage massive amounts of diverse user created data, and provide real-time services)

The report notes that a number of additional research areas were not included as these are the subject of ongoing investigation, and includes a summary of past reports in the appendix. Setting agenda for database research is a challenging task, and forty years of database research and development illustrates both the successes and failures of such efforts. Database management is a very pragmatic field and many promising research ideas were discarded as they did not provide any practical benefits, or turned out not to be a database problem (e.g. deductive databases [22], expert databases [23], etc.).

4 Conclusions

The Claremont Report on Database Research made an interesting observation noting that the database research community has doubled in size over the last decade (as measured by the number of publications and number of database related conference sessions), but at the same time there was a perception that the quality of reviews (and consequently, the quality of publications) has been decreasing over time. Notwithstanding this massive research effort most significant recent innovations came out of research labs of various companies (e.g. Google, Facebook, etc.), who are facing urgent challenges of unprecedented size and complexity of data, and millions of users running many thousands of transactions per second. These developments have not been fully anticipated by the database research community and interestingly not even by the traditional database vendors whose products offerings were dwarfed by the scale and complexity of new application domains.

Recent rise of the NoSQL movement whose proponents regard the existing relational DBMSs as inefficient, complex and expensive, and favor open source non-relational solutions, demonstrates this point. For example, the MapReduce programming model developed by Google [24], and its open source clone Hadoop [25] initially used to simplify the construction of inverted indexes, has been applied to text processing and numerous other tasks that require parallel computation over a very large set of data. MapReduce is designed to automatically parallelize and execute a program on a large cluster of commodity machines (typically, tens of thousands of machines), managing data partitioning, task scheduling, inter-machine communication, and recovery from machine failures. The combination of Hadoop with Hypertable [26] (an open source version of Google BigTable [26]), enables the concurrent execution of programs on tens of thousands of machines processing petabytes of data on a daily basis [27].

These types of applications were traditionally the domain of parallel databases, and a number of commercial database machines (e.g. Teradata, Oracle Exadata, etc.) have been available for some time with proven performance characteristics for processing very large data volumes. The comparison of MapReduce and parallel databases has been the subject of a recent publication [28], concluding that "using MapReduce to perform tasks that are best suited for DBMSs yields less than satisfactory results", and that MapReduce resembles more Extract-Transform-Load (ETL) system rather than a DBMS, and therefore is a complimentary technology rather than competing with DBMS. But, these conclusions are being hotly disputed by MapReduce proponents

who claim that the scalability benefits of this approach will eventually "relegate relational DBMS to the status of legacy technology". Database vendors are taking different approach to adopting the open source version of MapReduce (Hadoop), some implementing this technology in their products (e.g. Teradata, and IBM), while others (e.g. Microsoft) adopting a more cautious attitude [29].

Other vendors, notably Netezza have developed massively parallel database machines (Data Warehouse Appliances) that perform data filtering directly on the disk so that only the relevant portions of the data are propagated to the SQL database, gaining significant performance improvements over more traditional parallel database architectures. Additional functionality such as data analytics functions can also be implemented directly in hardware, achieving further performance gains [30]. Such approaches are using standard SQL database technology and are betting on further advances in computer hardware (larger and less expensive computer memory and more powerful CPUs as predicted by Moore's Law), and innovative, massively parallel database architectures to overcome the challenges of *big data*.

A key to understanding present and likely future database developments is a firm view of what constitutes the database paradigm, i.e. defining the scope of database research problems. Many of the recent challenges (in particular those faced by internet companies such as Google, Facebook, etc.), concern situations where three key elements that normally constitute a database environment are not present. While these applications are clearly data-intensive, there is no database (data is not loaded into a database), there is no database schema (data is mostly semi-structured and sparse), and there is no support for database queries (application involve mainly text search over semi-structured data). It is therefore difficult to regard such applications as database applications. Learning from history we can observe that a similar situation arose in the 1990s with Object-Oriented databases and later with XML databases, and conclude that there is little benefit in applying database solutions to problems that do not fit the database paradigm.

ACKNOWLEDGMENT

This research has been supported by GAČR (Grant Agency, Czech Republic) grant No. P403/10/0092 - Advanced Principles and Models for Enterprise ICT Management, University of Economics, Prague IGA (Internal Grant Agency) grant No. IG406040 – Cloud Computing Adoption and Governance, and the Research Centre for Human Centered Technology Design at the University of Technology, Sydney.

References

1. *Data, data everywhere - A special report on managing information*, in *The Economist*. 2010.
2. Gantz, J.F., *The Diverse and Exploding Digital Universe*. 2008, IDC. IDC. http://www.emc.com/collateral/analyst-reports/diverse-exploding-digital-universe.pdf
3. CERN. *The Large Hadron Collider*. 2010 14 MArch 2010]; CERN]. Available from: http://public.web.cern.ch/public/en/LHC/LHC-en.html.

4. Rothschild, J. *High Performance at Massive Scale – Lessons learned at Facebook*. CNS 2009 Lecture Series Archives 2009 [cited 2010 15 March 2010]; Centre for Networked Systems Lecture]. Available from: http://cns.ucsd.edu/lecturearchive09.shtml.
5. Dean, J. *Designs, Lessons and Advice from Building Large Distributed Systems*. 2009 [cited 2010 18 March 2010]; Available from: http://www.odbms.org/download/dean-keynote-ladis2009.pdf.
6. Codd, E.F., *A relational model of data for large shared data banks.* Commun. ACM, 1970. **13**(6): p. 377-387.
7. Atkinson, M., et al. *The object-oriented database system manifesto*. 1989: Citeseer.
8. Stonebraker, M., et al., *Third-generation database system manifesto.* SIGMOD Rec., 1990. **19**(3): p. 31-44.
9. Eisenberg, A. and J. Melton, *SQL: 1999, formerly known as SQL3.* ACM SIGMOD Record, 1999. **28**(1): p. 138.
10. Chamberlin, D. *XQuery: A query language for XML*. 2003: ACM New York, NY, USA.
11. W3C. *XML Schema*. 2010 [cited 2010 18 March 2010]; XML Schema Standard Sepcification]. Available from: http://www.w3.org/XML/Schema.
12. Widom, J., *Data management for XML: Research directions.* Bulletin of the Technical Committee on: p. 44.
13. Funderburk, J., et al., *XTABLES: Bridging relational technology and XML.* IBM Systems Journal, 2002. **41**(4): p. 616-641.
14. Fernández, M., W. Tan, and D. Suciu, *SilkRoute: trading between relations and XML.* Computer Networks, 2000. **33**(1-6): p. 723-745.
15. Schöning, H. *Tamino-a DBMS designed for XML*. 2001: IEEE Computer Society.
16. Bourret, R. *XML and Databases*. 2005 [cited 2010 18 March 2010]; Available from: http://ece.ut.ac.ir/DBRG/seminars/AdvancedDB/2006/Sanamrad-Hoseininasab/Refrences/6.pdf.
17. Eisenberg, A., et al., *SQL: 2003 has been published.* ACM SIGMOD Record, 2004. **33**(1): p. 119-126.
18. Hyde, J., *Data in flight.* Commun. ACM. **53**(1): p. 48-52.
19. Aboulnaga, A., et al., *Deploying Database Appliances in the Cloud.* Data Engineering, 2009. **32**(1): p. 13.
20. Silberschatz, A., M. Stonebraker, and J. Ullman, *Database research: Achievements and opportunities into the 21st century.* SIGMOD RECORD, 1996. **25**(1): p. 52–63.
21. Agrawal, R., et al., *The Claremont report on database research.* SIGMOD Rec., 2008. **37**(3): p. 9-19.
22. Ramakrishnan, R. and J. Ullman, *A survey of deductive database systems.* The journal of logic programming, 1995. **23**(2): p. 125-149.
23. Kerschberg, L., *Expert database systems: Knowledge/data management environments for intelligent information systems.* Information Systems, 1990. **15**(1): p. 151-160.

24. Dean, J. and S. Ghemawat, *MapReduce: Simplified data processing on large clusters.*
25. Borthakur, D., *The hadoop distributed file system: Architecture and design.* Hadoop Project Website, 2007.
26. Chang, F., et al. *Bigtable: A distributed storage system for structured data*. in *OSDI '06*. 2006.
27. Lai, E., *No to SQL? Anti-database movement gains steam*, in *Computerworld*. 2009. http://www.computerworld.com/s/article/9135086/No_to_SQL_Anti_database_movement_gains_steam_
28. Stonebraker, M., et al., *MapReduce and parallel DBMSs: friends or foes?* Commun. ACM. **53**(1): p. 64-71.
29. Lai, E., *Big three database vendors diverge on Hadoop*, in *Computerworld*. 2009, IDG. http://www.computerworld.com/s/article/9142406/Big_three_database_vendors_diverge_on_Hadoop
30. Morgan, T.P. *Netezza to bake analytics into appliances*. 2010 [cited 2010 18 March 2010]; Available from: http://www.theregister.co.uk/2010/02/24/netezza_data_analytics/.