

Towards Service-Based Data Management Systems

Ionut Subasu, Patrick Ziegler, Klaus R. Dittrich
{subasu, pziegler, dittrich}@ifi.unizh.ch
Database Technology Research Group
Department of Informatics, University of Zurich

Abstract: Today's database management systems are mainly large and heavy-weight monoliths that, even worse, undergo steady extensions in functionality. This makes them highly complex and creates problems when it comes to build tailored extensions. Our goal is to introduce more flexibility and extensibility by founding our database architecture on the principles of Service Oriented Architecture (SOA). In this paper, we present a Service-Based Data Management System (SBDMS) architecture that supports extensions according to user requirements. The architecture we propose consists of four major service layers (Storage Services, Access Services, Data Services, Extension Services). Each of these layers can flexibly be extended with new services that invoke other services from within or outside the system. In addition, the architecture's extension mechanism allows application functionality to be easily integrated into the Service-Based Data Management System to support reuse and optimization of commonly used application functionality.

1 Introduction

Today's Database Management Systems (DBMS) are mainly large and heavy-weight monoliths that come with a growing set of fixed extensions. This architecture makes them highly complex and does often not allow to build tailored extensions. In addition, DBMS users have to pay performance and cost penalties for hard-wired functionality they do not need. Research has been done to produce lighter [Sch01] [MAG⁺97] and more easily extensible DBMS architectures [RCF05]. However, these architectures are unsuitable for general DBMS applications since they generally do not provide complete DBMS functionality [Sch92].

Extensions that were needed over time are added mainly to the DBMS kernel, thickening it and causing loss of performance [SZ97, Rys05]. Due to their monolithic structure, existing DBMS have to be taken in their entirety. Moreover, it can be difficult to disable features that are not needed for particular applications. A monolithic structure cannot be easily extended or extensions require programmers with a high level of experience and knowledge concerning installation and maintenance [DG01, MAG⁺97]. Apart from this, monoliths cause considerable memory use and, because often a large number of processes run simultaneously, also a large processing overhead [DG01, MAG⁺97, SZ97]. In general, big and complex systems have to cope with two major problems: first, they are unable to rapidly adapt to new requirements and second, they struggle to meet software evolution

issues — for example, adding extensions for new functionality, flexible re-configuration according to individual user needs, and ease of maintenance [OB00].

During the 1980's, there was a research trend attempting to find solutions to support different database applications in response to the fact that existing relational database management systems (RDBMS) were not suitable for many non-business applications [Sch92]. Some of these approaches tried to find new ways to enhance existing commercial RDBMS, while others built new architectures and designed new systems (Exodus, Starburst, Probe, and others). In addition to flexible application support, existing DBMS have to provide an extensible set of data types and are facing an increasing need to simultaneously manage data described by different data models (e.g., relational and XML data). However, such extensions and services are typically closed as all propriety systems (as in Oracle or IBM's DB2) that do not easily allow to create further improvements [DG01]. In addition, these extension mechanisms are often deeply wired into the kernel of their monolithic systems, not allowing to be used or extended by common applications or DBMS customizers [SZ97].

In this paper, we present an architecture for Service-Based Data Management Systems (SBDMS) that can be easily extended by adding new functions and facilities. Our goal is to introduce more flexibility and extensibility by founding our database architecture on the principles of Service Oriented Architecture (SOA). We propose a system capable of handling different data types, being able to provide methods for adding new database features. Thereby, we give a solution to some of the still unsolved problems in today's DBMS systems, such as flexibly allowing permanent and tailored extensions to the system over time.

This paper is structured as follows: In the next Section, we present and discuss our approach to extensible DBMS. Section 3 covers related work and, finally, we draw conclusions in Section 4.

We must mention that this paper presents just the architecture idea. The main feature of the architecture is represented by the extensibility property and we do not want to obtain a very high processing performance.

2 A Service-Based Approach

The database architecture of the Data Independent Access Model (DIAM) developed by Mike Senko in 1983 [SAAF73] is composed of four hierarchically organized layers: the bottommost is the Physical Device Level Model that is followed by the Encoding Model, the String Model and the Entity Set Model. Later, Härder & Reuter proposed a model that has five layers. Here, the first layer in the database architecture is represented by the data stored as bytes on non-volatile drives. From one level to the other, as we go up to the top of the model, the elements are getting more complex. At the topmost layer, a declarative data

model is employed [Här05]. In order to support new data types, functional extensions, as well as XML data and data streams, Härder proposed in 2005 a new architectural model [Här05] based on the previous five layer architecture. An overview of the evolution of DBMS architectures is given in Figure 1.

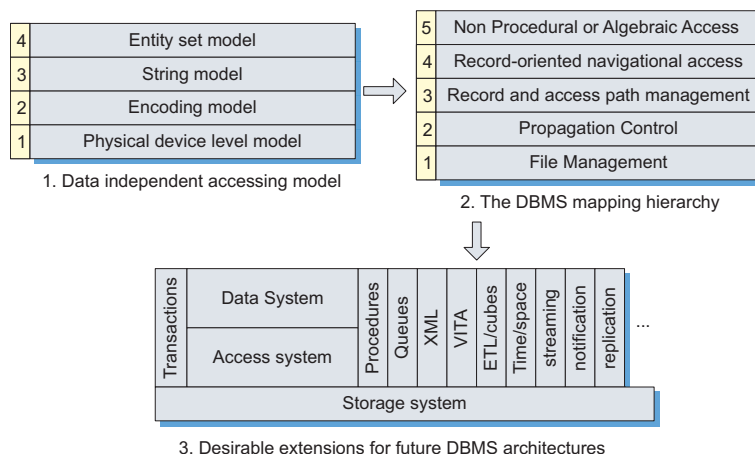


Figure 1: Evolution of DBMS: 1) DIAM [SAAF73], 2) The five level architecture presented by Härder & Reuter [HR85, Här05], 3) Desirable architecture proposed by Härder [Här05]

Today, applications can extend the functionality of DBMS through specific tasks that have to be provided by the data management systems, these tasks are called *services*, and allow interoperability between DBMS and other applications [GSD97]. This is a common approach for web-based applications. Implementing existing DBMS architectures from a service approach can introduce more flexibility and extensibility. Services that are accessible through a well defined and described interface enable any application to extend and reuse existing services without affecting other services. In general, Service Oriented Architecture (SOA) refers to a software architecture that is built from loosely coupled services to support business processes and software users. Typically, each resource is made available through independent services that can be distributed over computers that are connected through a network or on a single local machine. Services in the SOA approach are accessed only by means of a well defined interface, without requiring any knowledge on their implementation. SOAs can be implemented through a wide range of technologies like RPC, RMI, CORBA, COM, and web services, not making any restrictions on the implementation protocols. In general, services can communicate using an arbitrary protocol — for example, they can use a file system to send data between their interfaces. Due to loose coupling, services are not aware of which services they are called from; furthermore, calling services does not require any knowledge on how the invoked services complete their tasks.

Our SBDMS architecture borrowed the architectural levels from Härder [Här05], and includes new features and advantages introduced by SOA into the field of database architecture. It is organized into functional layers (see Figure 2) that each contain specialized services for specific tasks:

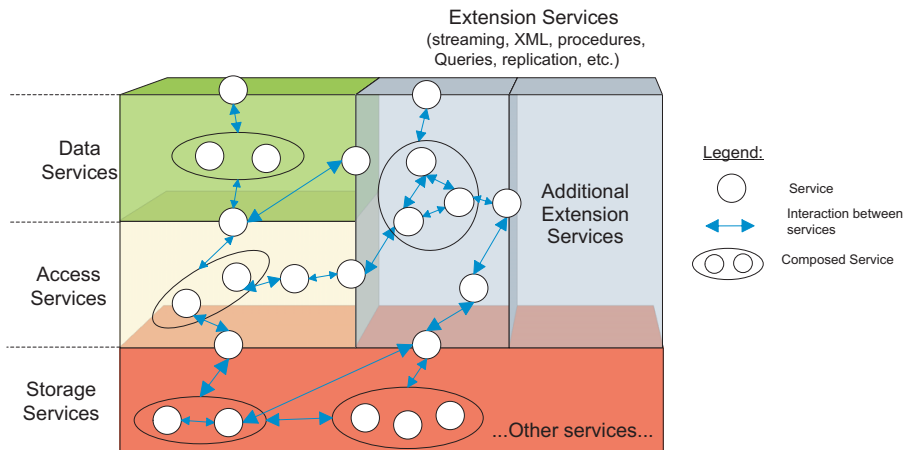


Figure 2: The Service-Based Database Architecture

- The *Storage Services Layer* provides services that work on the byte level, in very close collaboration with file management functions of the operating system. These services have to handle the physical specifications of each non-volatile device. In addition, they provide services for updating existing data and finding stored data, propagating information from the Access Services Layer to the physical level. Since different data types require different storage optimizations, special services are created to supply their particular functional needs. This layer is equivalent to the first and second layer of the five layer architecture presented by Härder & Reuter [HR85, Här05].
- In the *Access Services Layer*, more complex access paths, mappings, particular extensions for special data models, that are represented in the Storage Services Layer are located. This layer is in charge of the physical data representations of data records and provides access path structures like B-trees. Moreover, this layer is responsible for sorting record sets, navigating through logical record structures, making joins, and similar higher-level operations. By the way it is built, this layer represents a key factor to database performance. The Access Services Layer has functions that are comparable to those in the third and fourth layer as presented by Härder & Reuter [HR85, Här05].
- At the *Data Services Layer*, data is represented in logical structures like tables or views. These are data structures without any procedural interface to the underlying

ing database. The Data Service Layer can be mapped to the Non-Procedural and Algebraic Access level in the architecture by Härder & Reuter [HR85, Här05].

- Finally, at the *Extension Services Layer*, users can design tailored extensions — for example, creating new services or reusing existing services that are based on any available service from the other layers. These extensions help to manage different data types like XML files or streaming data. In this layer, users can integrate application specific services in order to provide specific data types or specific functionality needed by their applications (e.g., for optimization purposes).

By breaking down the DBMS architecture into services we obtain an flexible architecture, where components loosely coupled. Such a service-based architecture can be complemented with services that are used by applications and other services allowing easy integration and development of additional extensions. Instead of hard-wiring static components, we propose a loosely coupled architecture that can easily be distributed. Services allow dynamic binding in order to accomplish complex tasks for a particular client. Moreover, services are reusable and have high autonomy due to the fact that they are accessed only through well defined interfaces. This also helps to reduce complexity since new components can be built based on existing services. Organizing services on layers is a solution to composing a large numbers of services and will help in making decisions about their granularity.

Our suggested system architecture enables users to integrate specialized functionality from applications into the database. That is, different services from the SBDBMS can be combined with (potentially optimized) specialized services needed by particular applications. With this approach, we adapt the DBMS to applications instead of adapting applications to the DBMS. In addition, this allows to reuse optimized functionality that is shared by several applications instead of having to invest efforts for the implementation of the same functionality again. Another way of extending the system is by invoking internal services through calls from external web services or web servers. Users can thereby have their own tailored services on their personal computers to replace or extend existing SBDBM services according to their needs. For example, with this facility a developer can test a new database application without having to shut down and restart the whole DBMS. Figure 3 illustrates how our architecture can be extended. In general, services can run on different computers and can invoke each other in a variety of ways. An invoked service can reside on a different location in a network. Hence, users just call a remote service instead of locally executing the particular service on their computer, thereby reducing the necessary local processing resources. By their structure, services can provide great flexibility and a high level of reusability. More specifically, the SOA design style offers solutions to wider issues regarding usage and interpretation of software packages by providing ways to clearly describe the services [BBT04].

Developers of new applications can particularly benefit from service reuse by taking one or more services that run on the SBDBMS, from any available layer, and integrate these services into their application to provide optimized access to their application-specific data. For example, assume that an application needs access to the storage level of the DBMS

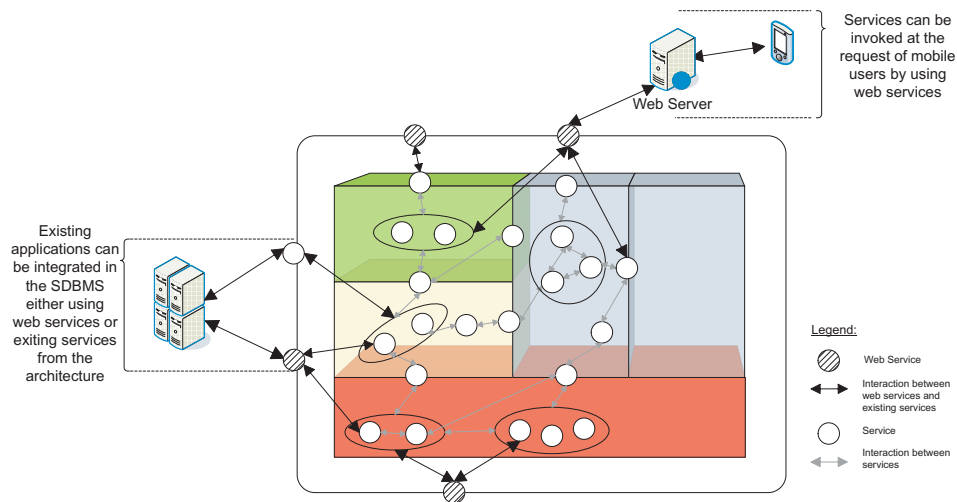


Figure 3: Examples illustrating extensions of the SBDBMS

in order to obtain statistical information, such as available storage space or data fragmentation. In this case, the developer of this application can add the necessary information services to the storage level. This way, he provides the information source required for his application. Then, the application just has to invoke these services to retrieve the data. Furthermore, other services from other layers can be used together with this kind of extension services if required.

To ensure interoperability between various external applications and internal services, interfacing services can be employed as bridging points. In this way, the system can be composed of third party services and applications that access existing database services through their well defined interfaces. Note that not every service has to be constantly running in a SBDBMS — services can be enabled and disabled as needed. Moreover, services can be distributed and be made redundant by using several computers connected through a network. Therefore, a SBDBMS can be customized to use services from other specific locations to optimize particular tasks. This approach introduces a high degree of adaptability into the database system. Priorities can be assigned to services that demand a considerable amount of resources, thereby enabling Quality of Service agreements for special data types like multimedia and streaming data.

2.1 Discussion

An important advantage of our service oriented approach to database architecture is that tailored services can be invoked when they are needed to precisely meet particular needs. Thus, a SBDBMS can run without default services so users can choose their own services

based on their needs and available resources. If needed, users are enabled to flexibly have additional services working on remote locations and use them in their SBDMS without having to install a complete new DBMS. Developers can deploy new services or update existing ones by redeploying or restarting just the affected components instead of the whole DBMS (as in a monolithic DBMS). In case of failure of services, alternative services can supply answers to requests, making the DBMS more robust and increasing reliability. If, in case of failure, no alternative services are available to answer a particular open request, then just this request fails — other services that do not rely on the failed service are still available. This way, services can bring higher degrees of reliability to the database system. Moreover, by adding services at run time to the SBDBMS, its functionality can dynamically be extended.

Our architecture provides the basis for user-friendly service interfaces and functions that are easy to use; therefore, users can generate their own tailored services. For example, services can be provided to store text documents or spreadsheets in the SBDBMS as provided by front-end applications. This way, enhanced and reusable storage services as offered by DBMS can be provided for common applications without requiring users to write SQL statements to store their data into a database.

The extensibility gained through the service oriented basis of our architecture allows to easily distribute the services, allowing to arbitrarily deploy new services on new locations in a network without affecting the existing SBDBMS. This facilitates the deployment of new services and new extensions. Existing services that run on behalf of applications can communicate with other services from the SBDBMS. Thus, interoperation of various services can be accomplished, even if they do not belong to core SBDBMS functionality. Developing new services and extensions takes less resources due to the high degree of reusability the proposed architecture provides, hence bringing in additional advantages like low maintenance and reduced development costs. As a special case, existing DBMS can also be integrated into to the SBDBMS through the development of services to provide database interoperability (e.g., interfacing services that forward data of a particular type to a specialized DBMS for storage).

From the security and data access point of view, our proposed database architecture enables more fine-grained access control — for example, access restrictions can be formulated for data of a particular type or based on service types instead of users and roles only. Additionally, access rights can be set to services depending on whether the services access other services or data. Possible access and interaction patterns between users, services, and data are (see also Figure 4):

1. Services can have defined rights to access other services or groups of services.
2. Special data areas can have defined access rights for different services.
3. Users can have defined access for groups of services or special (single) services, as well as for special DBMS extensions like multimedia data or data streams. For example, a digital multimedia store in which some users have access to movie files

and others to music files can be realized by restricting user access to services that provide access to the respective data type.

4. User rights for data can be also defined based on users and roles, as in traditional database systems.
5. In the complex case of combined of interactions, a strategy is needed to establish an access policy describing the propagation of access rights through the SBDBMS.

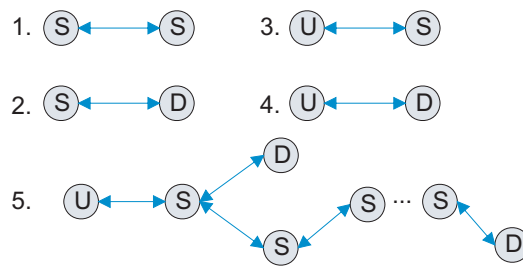


Figure 4: Interaction categories: 1. service with service, 2. service with data, 3. user with service, 4. user with data, 5. example of a mixed interaction

One of the problems in the suggested approach is to ensure “good” service design from a particular user perspective — i.e., what are the qualities of desired and useful services according to the needs of a given user or application. Further problems are the definition of the necessary service interfaces (which interfaces should be made available for particular user groups) and the fact that the reliability and stability of the DBMS must be ensured while the system is open for new services. To accomplish this, each service must have a meaningful description that can be read by both, machines and humans. Last, but not least, issues to be addressed are how to effectively change access rights in a SBDBMS and how to ensure secure communication between distributed DBMS services.

3 Related Work

The problems concerning monolithic DBMS stated above have dragged research in extensible database system architectures long ago. This issue was approached in different projects, such as Probe [DMB⁺90], Starburst [HCL⁺90], DASDBS [PSS87, SPSW90], Exodus [CDRS86], and Postgres [SR86].

DASDBS has been built as a family of database systems that are specially designed for certain application classes. The main idea was to create an application front-end that is going to run on a common kernel system [PSS87]. The DASDBS kernel functions like the storage manager provide only basic data management functions. On top of the kernel, the application-specific front-ends are situated that allow users to build tailored interfaces for

their applications. In contrast to this, our service approach allows users to integrate existing DBMS by building services that act as an interface to our SBDBMS. Furthermore, we permit increased flexibility by allowing users to employ and create services on top of the integrated DBMS. In addition, users can build new services and even define their own custom DBMS inside our SBDBMS architecture.

Starburst provides a skeleton architecture that allows extensions in several but predetermined ways. On the other hand, kernel systems like Exodus and DASDBS allow extension by enabling users to build new components on the top of their existing architecture. Furthermore, these systems allow to access some of the kernel's functionality. By the way these architectures are built, they do not provide a fully functional and complete DBMS; instead, it can be said that in fact they are powerful file systems [Sch92].

Even though our approach shares its goal with our predecessors (i.e., to create an extensible DBMS), our concept offers a different approach to the problem. Based on services that collaborate to perform a task, our system is more modular and applications can more easily be integrated into the DBMS. Unlike Starburst that is a full DBMS and can only be extended by programmers who have the necessary skills and knowledge [HCL⁺90], our approach is based on services that are well defined and easy to use by common users and programmers. Additionally, we provide an extended number of low level services for developers to fine-tune the system.

POSTGRES [SR86] is created as an extensible relational system that supports different data types for polygons, bitmaps, and texts. Extensions can be added by building new data types or by extending stored procedures using C functions. Although it is an open source database system that allows users to modify the system code, POSTGRES does not offer explicitly defined interfaces to build new extensions without impacts on the entire system. In contrast to this, services are in our approach based on well described, distinct interfaces. Adding new services in our architecture does not have an impact to the entire database system since the services we use are loosely coupled.

Extensible Database Management Systems offer a powerful and highly efficient kernel that allows adding new extensions (typically at the topmost level of their architecture). However, by taking a service approach we are able to extend the database system at every point in the architecture, simply by creating new services that use existing services. For example, we can create a new data type and define a special service responsible for the way this type of data is physically stored. Then, this service can be transparently called by existing and newly developed services, thereby providing a more versatile DBMS extension mechanism.

Component database management systems [DG01] are software frameworks that can be extended by adding new programming logic modules. Added extensions can implement database objects, such as data types, analytic algorithms. Component based architectures and SOA differ in various respects. First of all, using components is similar to reusing code, while integrating services means using functionality under a very well defined con-

tract. Second, composing services to create a higher-level processes is different than linking components together to create an application. Also the service contract is not the same as a component interface. Developing an architecture with services requires neither system compilation nor linking bringing in a more versatile approach to extend the architecture. On the other hand we must pay attention to the service granularity. A very fine granularity increases the message overhead. While coarse-grained services limit the extension capability of the Data Management System.

4 Conclusions

In this paper, we have described and discussed a novel approach for a Service-Based Database Management System (SBDBMS). Our goal is to introduce more flexibility and extensibility by founding our database architecture on the principles of Service Oriented Architecture (SOA), and, by that, providing an architecture that flexibly supports permanent and tailored extensions. In future work, we are going to design the proposed architecture in more detail and compose the necessary services and interfaces as a foundation for concrete implementations.

References

- [BBT04] David Budgen, Pearl Brereton, and Mark Turner. Codifying a Service Architectural Style. In *COMPSAC '04*, pages 16–22, 2004.
- [CDRS86] Michael J. Carey, David J. DeWitt, Joel E. Richardson, and Eugene J. Shekita. Object and File Management in the EXODUS Extensible Database System. In *VLDB '86*, pages 91–100, 1986.
- [DG01] Klaus R. Dittrich and Andreas Geppert. *Component Database Systems: Introduction, Foundations, and Overview*. Morgan Kaufmann, 2001.
- [DMB⁺90] U. Dayal, F. Manola, A. Buchmann, U. Chakravarthy, D. Goldhirsch, S. Heiler, J. Orenstein, and A. Rosenthal. Simplifying complex objects: the PROBE approach to modelling and querying them. pages 390–399, 1990.
- [GSD97] A. Geppert, S. Scherrer, and K. R. Dittrich. KIDS: Construction of Database Management Systems based on Reuse. Technical Report ifi-97.01, 1997.
- [Här05] Theo Härder. DBMS Architecture - Still an Open Problem. In *BTW*, pages 2–28, 2005.
- [HCL⁺90] L.M. Haas, W. Chang, G.M. Lohman, J. McPherson, P.F. Wilms, G. Lapis, B. Lindsay, H. Pirahesh, M.J. Carey, and E. Shekita. Starburst Mid-Flight: As the Dust Clears. *IEEE Transactions on Knowledge and Data Engineering*, 02(1):143–160, 1990.
- [HR85] T. Härder and A. Reuter. Architektur von Datenbanksystemen für Non-Standard-Anwendungen. In *BTW '85*, pages 235–286, 1985.
- [MAG⁺97] Jason McHugh, Serge Abiteboul, Roy Goldman, Dallon Quass, and Jennifer Widom. Lore: A Database Management System for Semistructured Data. *SIGMOD Record*, 26(3):54–66, 1997.

- [OB00] Mathee Olarnsakul and Dentcho N. Batanov. A Component Coordination Model for Customization and Composition of Component-Based System Design. *ecbs*, 00:228, 2000.
- [PSS87] H. B. Paul, H. J. Schek, and M. H. Scholl. Architecture and implementation of the Darmstadt database kernel system. In *SIGMOD '87*, pages 196–207, 1987.
- [RCF05] Michael Rys, Don Chamberlin, and Daniela Florescu. XML and relational database management systems: the inside story. In *SIGMOD '05*, pages 945–947, 2005.
- [Rys05] Michael Rys. XML and relational database management systems: inside Microsoft SQL Server 2005. In *SIGMOD '05*, pages 958–962, 2005.
- [SAAF73] M. E. Senko, E. B. Altman, M. M. Astrahan, and P. L. Fehder. Data Structures and Accessing in Data Base Systems. *IBM Systems Journal*, 12(1):30–93, 1973.
- [Sch92] M. Scholl. Extensions to the relational data model. In *In P. Loucopoulos and R. Zicari, editors, Conceptual Modelling, Databases and CASE: An Integrated View of Information Systems Development*. Jon Wiley & Sons, New York, 1992.
- [Sch01] Harald Schöning. Tamino - A DBMS designed for XML. In *Proceedings of the 17th International Conference on Data Engineering*, pages 149–154, 2001.
- [SPSW90] H. J. Schek, H. B. Paul, M. H. Scholl, and G. Weikum. The DASDBS Project: Objectives, Experiences, and Future Prospects. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):25–43, 1990.
- [SR86] M. Stonebreaker and L. A. Rowe. The design of POSTGRES. In *SIGMOD '86*, pages 340–355, 1986.
- [SZ97] Avi Silberschatz and Stan Zdonik. Database systems breaking out of the box. *SIGMOD Rec.*, 26(3):36–50, 1997.