Evolution

Hi! Welcome to the fascinating world of DBMS. The whole Course will deal with the most widely used software system in the modern world, **The Database Management System.** Here in this lecture we are going to discuss about the evolution of DBMS and how it nurtured in to the most wanted software. Let's begin to devise.

The boundaries that have traditionally existed between DBMS and other data sources are increasingly blurring, and there is a great need for an information integration solution that provides a unified view of all of these services. This article proposes a platform that extends a federated database architecture to support both relational and XML as first class data models, and tightly integrates content management services, workflow, messaging, analytics, and other enterprise application services.

A Brief Introduction

- First, DBMSs have proven to be hugely successful in managing the information explosion that occurred in traditional business applications over the past 30 years. DBMSs deal quite naturally with the storage, retrieval, transformation, scalability, reliability, and availability challenges associated with robust data management.
- Secondly, the database industry has shown that it can adapt quickly to accommodate the diversity of data and access patterns introduced by e-business applications over the past 6 years. For example, most enterprise-strength DBMSs have built-in object-relational support, XML capabilities, and support for federated access to external data sources.
- Thirdly, there is a huge worldwide investment in DBMS technology today, including databases, supporting tools, application development environments, and skilled administrators and developers. A platform that exploits and enhances the DBMS architecture at all levels is in the best position to provide robust end-to-end information integration.

This paper is organized as follows:

1. We briefly review the evolution of the DBMS architecture.
2. We provide a real-world scenario that illustrates the scope of the information integration problem and sketches out the requirements for a technology platform.
3. We formally call out the requirements for a technology platform.
4. We present a model for an information integration platform that satisfies these requirements and provides an end-to-end solution to the integration problem as the next evolutionary step of the DBMS architecture.

How did database software evolve?

In the early days of computing, computers were mainly used for solving numerical problems. Even in those days, it was observed that some of the tasks were common in many problems that were being solved. It therefore was considered desirable to build special subroutines which perform frequently occurring computing tasks such as computing Sin(x). This lead to the development of mathematical subroutine libraries that are now considered integral part of any computer system.

By the late fifties storage, maintenance and retrieval of non-numeric data had become very important. Again, it was observed that some of the data processing tasks occurred quite frequently e.g. sorting. This lead to the development of *generalized* routines for some of the most common and frequently used tasks.

A general routine by its nature tends to be somewhat less efficient than a routine designed for a specific problem. In the late fifties and early sixties, when the hardware costs were high, use of general routines was not very popular since it became a matter of trade-off between hardware and software costs. In the last two decades, hardware costs have gone down dramatically while the cost of building software has gone up. It is no more a trade-off between hardware and software costs since hardware is relatively cheap and building reliable software is expensive.

Database management systems evolved from generalized routines for file processing as the users demanded more extensive and more flexible facilities for managing data. Database technology has undergone major changes during the 1970's. An interesting history of database systems is presented by Fry and Sibley (1976).

Figure 1 captures the evolution of relational database technology. Relational databases were born out of a need to store, manipulate and manage the integrity of large volumes of data. In the 1960s, network and hierarchical systems such as [CODASYL] and IMS™

were the state-of-the -art technology for automated banking, accounting, and order processing systems enabled by the introduction of commercial mainframe computers. While these systems provided a good basis for the early systems, their basic architecture mixed the physical manipulation of data with its logical manipulation. When the physical location of data changed, such as from one area of a disk to another, applications had to be updated to reference the new location.

A revolutionary paper by Codd in 1970 [CODD] and its commercial implementations changed all that. Codd's relational model introduced the notion of *data independence*, which separated the physical representation of data from the logical representation presented to applications. Data could be moved from one part of the disk to another or stored in a different format without causing applications to be rewritten. Application developers were freed from the tedious physical details of data manipulation, and could focus instead on the logical manipulation of data in the context of their specific application.

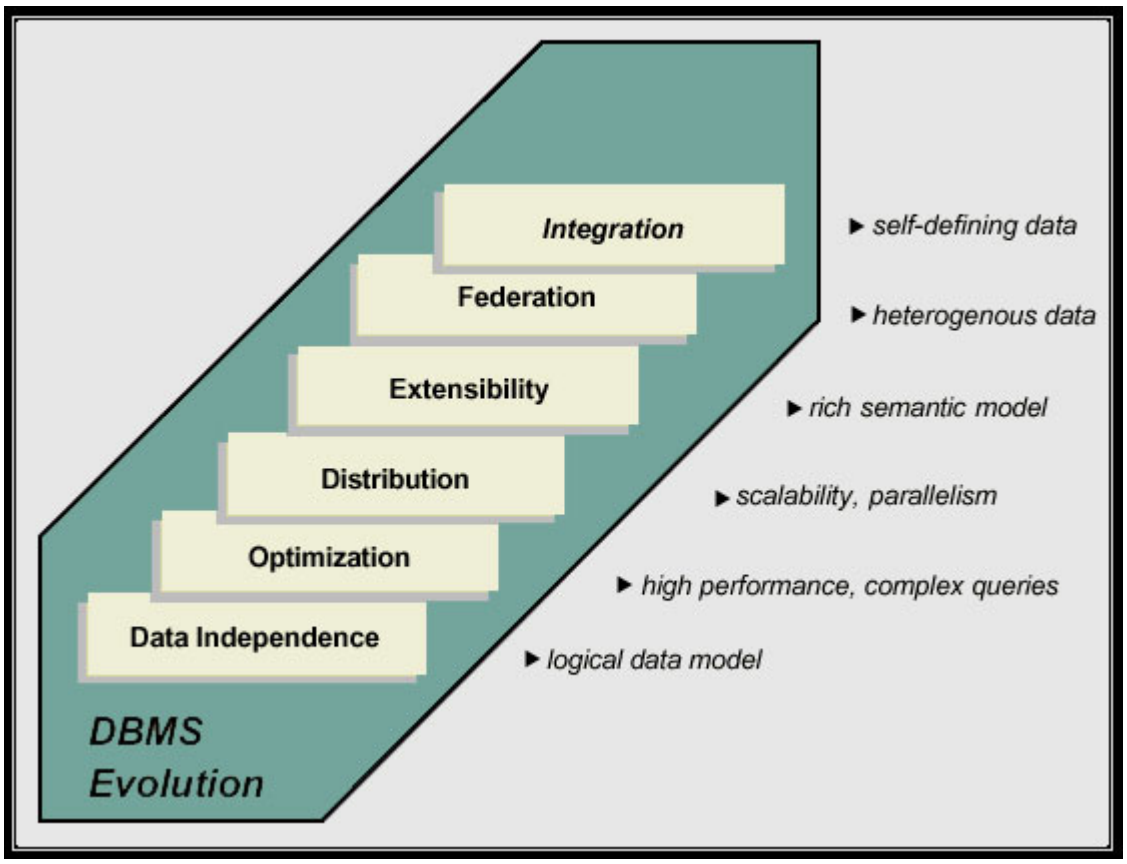
Not only did the relational model ease the burden of application developers, but it also caused a paradigm shift in the data management industry. The separation between *what* and *how* data is retrieved provided an architecture by which the new database vendors could improve and innovate their products. [SQL] became the standard language for describing what data should be retrieved. New storage schemes, access strategies, and indexing algorithms were developed to speed up how data was stored and retrieved from disk, and advances in concurrency control, logging, and recovery mechanisms further improved data integrity guarantees [GRAY][LIND][ARIES]. Cost-based *optimization*

Techniques [OPT] completed the transition from databases acting as an abstract data management layer to being high-performance, high-volume query processing engines.

As companies globalized and as their data quickly became distributed among their national and international offices, the boundaries of DBMS technology were tested again. Distributed systems such as [R*] and [TANDEM] showed that the basic DBMS architecture could easily be exploited to manage large volumes of *distributed* data. Distributed data led to the introduction of new parallel query processing techniques

[PARA], demonstrating the scalability of the DBMS as a high-performance, high-volume query processing engine.

Figure 1. Evolution of DBMS architecture



The lessons learned in extending the DBMS with distributed and parallel algorithms also led to advances in *extensibility*, whereby the monolithic DBMS architecture was deplumed with plug-and-play components [STARBURST]. Such an architecture enabled new abstract data types, access strategies and indexing schemes to be easily introduced as new business needs arose. Database vendors later made these hooks publicly available to customers as Oracle data cartridges, Informix® DataBlades®, and DB2® Extenders™.

Throughout the 1980s, the database market matured and companies attempted to standardize on a single database vendor. However, the reality of doing business generally made such a strategy unrealistic. From independent departmental buying decision to mergers and acquisitions, the scenario of multiple database products and other management systems in a single IT shop became the norm rather than the exception.

Businesses sought a way to streamline the administrative and development costs associated with such a heterogeneous environment, and the database industry responded with *federation*. Federated databases [FED] provided a powerful and flexible means for transparent access to heterogeneous, distributed data sources.

We are now in a new revolutionary period enabled by the Internet and fueled by the e-business explosion. Over the past six years, Java™ and XML have become the vehicles for portable code and portable data. To adapt, database vendors have been able to draw on earlier advances in database extensibility and abstract data types to quickly provide object-relational data models [OR], mechanisms to store and retrieve relational data as XML documents [XTABLES] , and XML extensions to SQL [SQLX].

The ease with which complex Internet-based applications can be developed and deployed has dramatically accelerated the pace of automating business processes. The premise of our paper is that the challenge facing businesses today is *information integration*. Enterprise applications require interaction not only with databases, but also content management systems, data warehouses, workflow systems, and other enterprise applications that have developed on a parallel course with relational databases. In the next section, we illustrate the information integration challenge using a scenario drawn from a real-world problem.

Scenario

To meet the needs of its high-end customers and manage high-profile accounts, a financial services company would like to develop a system to automate the process of managing, augmenting and distributing research information as quickly as possible. The company subscribes to several commercial research publications that send data in the Research Information Markup Language (RIXML), an XML vocabulary that combines investment research with a standard format to describe the report's meta data [RIXML]. Reports may be delivered via a variety of mechanisms, such as real-time message feeds, e-mail distribution lists, web downloads and CD ROMs.

Figure 2. Financial services scenario

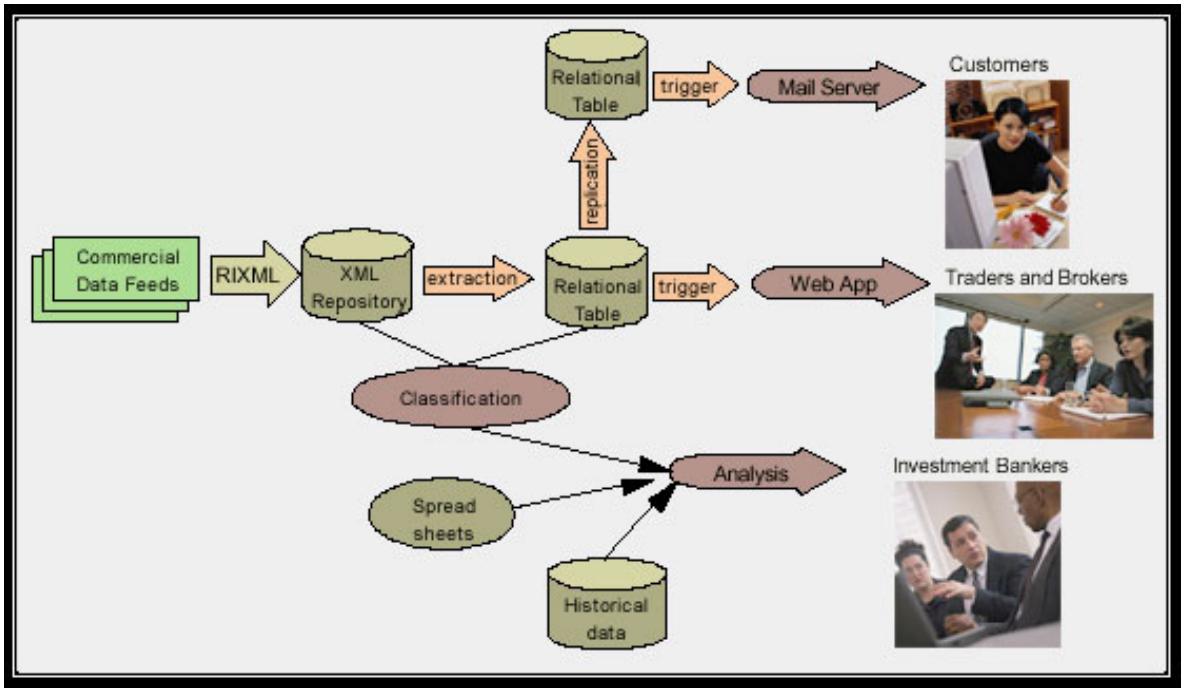


Figure 2 shows how such research information flows through the company.

1. When a research report is received, it is archived in its native XML format.
2. Next, important Meta data such as company name, stock price, earnings estimates, etc., is extracted from the document and stored in relational tables to make it available for real-time and deep analysis.
3. As an example of real-time analysis, the relational table updates may result in database triggers being fired to detect and recommend changes in buy/sell/hold positions, which are quickly sent off to equity and bond traders and brokers. Timeliness is of the essence to this audience and so the information is immediately replicated across multiple sites. The triggers also initiate e-mail notifications to key customers.
4. As an example of deep-analysis, the original document and its extracted Meta data are more thoroughly analyzed, looking for such keywords as "merger", "acquisition" or "bankruptcy" to categorize and summarize the content. The summarized information is combined with historical information made available to the company's market research and investment banking departments.

5. These departments combine the summarized information with financial information stored in spread sheetand other documents to perform trend forecasting, and to identify merger and acquisition opportunities.

Requirements

To build the financial services integration system on today's technology, a company must cobble together a host of management systems and applications that do not naturally coexist with each other. DBMSs, content management systems, data mining packages and workflow systems are commercially available, but the company must develop integration software in-house to integrate them. A database management system can handle the structured data, but XML repositories are just now becoming available on the market. Each time a new data source is added or the information must flow to a new target, the customer's home grown solution must be extended.

The financial services example above and others like it show that the boundaries that have traditionally existed between DBMSs, content management systems, mid-tier caches, and data warehouses are increasingly blurring, and there is a great need for a platform that provides a unified view of all of these services. We believe that a robust information integration platform must meet the following requirements:

- *Seamless integration of structured, semi-structured, and unstructured data from multiple heterogeneous sources*. Data sources include data storage systems such as databases, file systems, real time data feeds, and image and document repositories, as well as data that is tightly integrated with vertical applications such as SAP or Calypso. There must be strong support for standard meta-data interchange, schema mapping, schema-less processing, and support for standard data interchange formats. The integration platform must support both consolidation, in which data is collected from multiple sources and stored in a central repository, and federation, in which data from multiple autonomous sources is accessed as part of a search, but is not moved into the platform itself. As shown in the financial services example, the platform must also provide transparent transformation support to enable data reuse by multiple applications.

- *Robust support for storing, exchanging, and transforming XML data*. For many enterprise information integration problems, a relational data model is too restrictive to be effectively used to represent semi-structured and unstructured data. It is clear that XML is capable of representing more diverse data formats than relational, and as a result it has become the lingua franca of enterprise integration. Horizontal standards such as [EBXML][SOAP], etc., provide a language for independent processes to exchange data, and vertical standards such as [RIXML] are designed to handle data exchange for a specific industry. As a result, the technology platform must be XML-aware and optimized for XML at all levels. A native XML store is absolutely necessary, along with efficient algorithms for XML data retrieval. Efficient search requires XML query language support such as [SQLX] and [XQuery].
- *Built-in support for advanced search capabilities and analysis over integrated data*. The integration platform must be bilingual. Legacy OLTP and data warehouses speak SQL, yet integration applications have adopted XML. Content management systems employ specialized APIs to manage and query a diverse set of artifacts such as documents, music, images, and videos. An inverse relationship naturally exists between overall system performance and the path length between data transformation operations and the source of the data. As a result, the technology platform must provide efficient access to data regardless of whether it is locally managed or generated by external sources, and whether it is structured or unstructured. Data to be consolidated may require cleansing, transformation and extraction before it can be stored. To support applications that require deep analysis such as the investment banking department in the example above, the platform must provide integrated support for full text search, classification, clustering and summarization algorithms traditionally associated with text search and data mining.
- *Transparently embed information access in business processes*. Enterprises rely heavily on workflow systems to choreograph business processes. The financial services example above is an example of a *macroflow*, a multi-transaction

sequence of steps that capture a business process. Each of these steps may in turn be a *microflow*, a sequence of steps executed within a single transaction, such as the insert of extracted data from the research report and the database trigger that fires as a result. A solid integration platform must provide a workflow framework that transparently enables interaction with multiple data sources and applications. Additionally, many business processes are inherently asynchronous. Data sources and applications come up and go down on a regular basis. Data feeds may be interrupted by a hardware or a network failures. Furthermore, end users such as busy stock traders may not want to poll for information, but instead prefer to be notified when events of interest occur. An integration platform must embed messaging, web services and queuing technology to tolerate sporadic availability, latencies and failures in data sources and to enable application asynchrony.

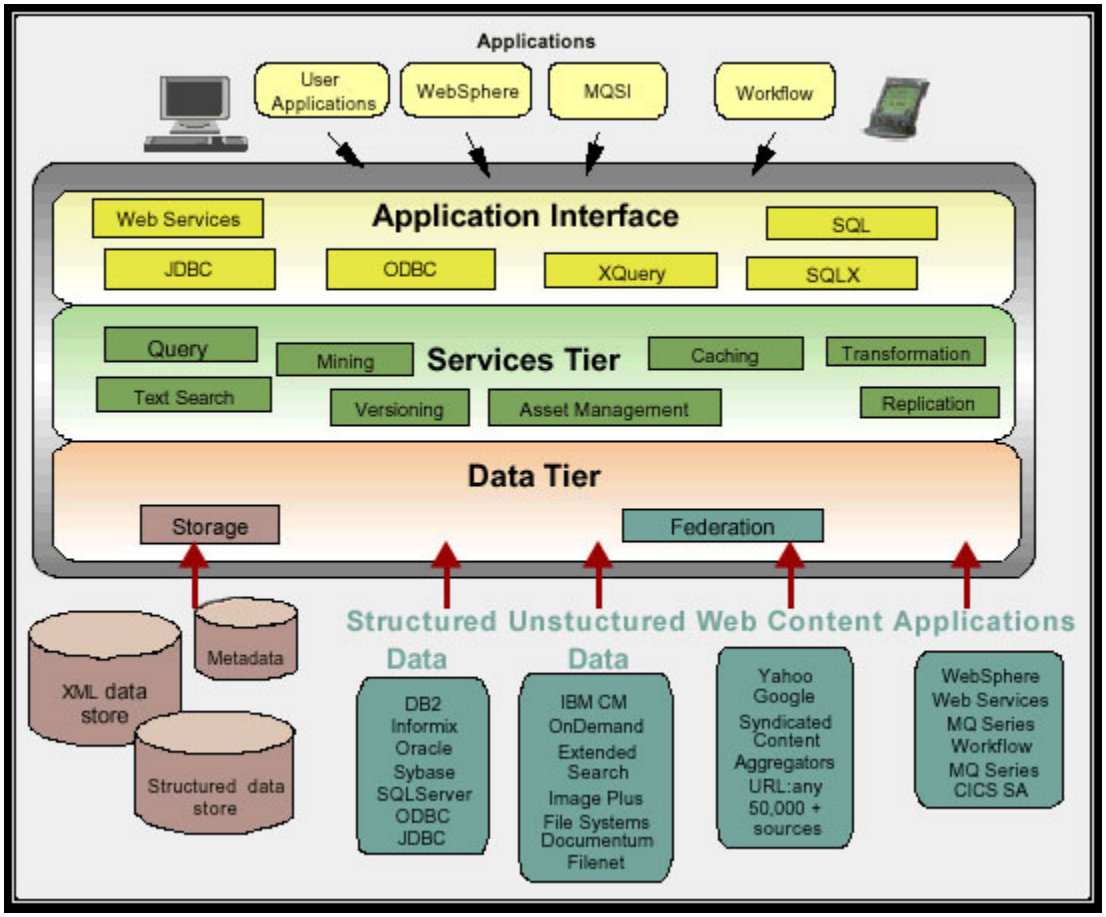
- *Support for standards and multiple platforms*. It goes without saying that an integration platform must run on multiple platforms and support all relevant open standards. The set of data sources and applications generating data will not decrease, and a robust integration platform must be flexible enough to transparently incorporate new sources and applications as they appear. Integration with OLTP systems and data warehouses require strong support for traditional SQL. To be an effective platform for business integration, emerging cross-industry standards such as [SQLX] and [XQuery] as well as standards supporting vertical applications [RIXML].
- *Easy to use and maintain*. Customers today already require integration services and have pieced together in-house solutions to integrate data and applications, and these solutions are costly to develop and maintain. To be effective, a technology platform to replace these in-house solutions must reduce development and administration costs. From both an administrative and development point of view, the technology platform should be as invisible as possible. The platform should include a common data model for all data sources and a consistent programming model. Metadata management and application development tools must be provided to assist administrators, developers, and users in both constructing and exploiting information integration systems.

Architecture

Figure 3 illustrates our proposal for a robust information integration platform.

- The foundation of the platform is the data tier, which provides storage, retrieval and transformation of data from base sources in different formats. We believe that it is crucial to base this foundation layer upon an enhanced full-featured federated DBMS architecture.
- A services tier built on top of the foundation draws from content management systems and enterprise integration applications to provide the infrastructure to transparently embed data access services into enterprise applications and business processes.
- The top tier provides a standards-based programming model and query language to the rich set of services and data provided by the data and services tiers.

Figure 3. An information integration platform



Programming Interface

A foundation based on a DBMS enables full support of traditional programming interfaces such as ODBC and JDBC, easing migration of legacy applications. Such traditional APIs are synchronous and not well-suited to enterprise integration, which is inherently asynchronous. Data sources come and go, multiple applications publish the same services, and complex data retrieval operations may take extended periods of time. To simplify the inherent complexities introduced by such a diverse and data-rich environment, the platform also provides an interface based on Web services ([WSDL] and [SOAP]). In addition, the platform includes asynchronous data retrieval APIs based on message queues and workflow technology [MQ][WORKFLOW]to transparently schedule and manage long running data searches.

Query Language

As with the programming interface, the integration platform enhances standard query languages available for legacy applications with support for XML-enabled applications. [XQuery] is supported as the query language for applications that prefer an XML data model. [SQLX] is supported as the query language for applications that require a mixed data model as well as legacy OLTP-type applications. Regardless of the query language, all applications have access to the federated content enabled by the data tier. An application may issue an XQuery request to transparently join data from the native XML store, a local relational table, and retrieved from an external server. A similar query could be issued in SQLX by another (or the same) application.

Summary

The explosion of information made available to enterprise applications by the broad-based adoption of Internet standards and technologies has introduced a clear need for an information integration platform to help harness that information and make it available to enterprise applications. The challenges for a robust information integration platform are steep. However, the foundation to build such a platform is already on the market. DBMSs have demonstrated over the years a remarkable ability to manage and harness structured data, to scale with business growth, and to quickly adapt to new requirements. We believe that a federated DBMS enhanced with native XML capabilities and tightly coupled enterprise application services, content management services and analytics is the right technology to provide a robust end-to-end solution.

Questions

1. Explain the evolution of the DBMS architecture.

2. Illustrates the scope of the information integration problem and sketches out the requirements for a technology platform.
3. Present a model for an information integration platform that satisfies these requirements and provides an end-to-end solution to the integration problem as the next evolutionary step of the DBMS architecture.

Selected Bibliography

- [ARIES] C. Mohan, et al.: ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging., TODS 17(1): 94-162 (1992).
- [CACHE] C. Mohan: Caching Technologies for Web Applications, A Tutorial at the Conference on Very Large Databases (VLDB), Rome, Italy, 2001.
- [CODASYL] ACM: CODASYL Data Base Task Group April 71 Report, New York, 1971.
- [CODD] E. Codd: A Relational Model of Data for Large Shared Data Banks. ACM 13(6):377-387 (1970).
- [EBXML] http://www.ebxml.org.
- [FED] J. Melton, J. Michels, V. Josifovski, K. Kulkarni, P. Schwarz, K. Zeidenstein: SQL and Management of External Data', SIGMOD Record 30(1):70-77, 2001.
- [GRAY] Gray, et al.: Granularity of Locks and Degrees of Consistency in a Shared Database., IFIP Working Conference on Modelling of Database Management Systems, 1-29, AFIPS Press.
- [INFO] P. Lyman, H. Varian, A. Dunn, A. Strygin, K. Swearingen: How Much Information? at http://www.sims.berkeley.edu/research/projects/how-much-info/.

- [LIND] B. Lindsay, et. al: Notes on Distributed Database Systems. IBM Research Report RJ2571, (1979).