Independent, Open Enterprise Data Integration

Joseph M. Hellerstein, Michael Stonebraker, and Rick Caccia
{jmh,mike,rick}@cohera.com

Abstract

Database researchers and practitioners have long espoused the virtues of data independence. When logical and physical representations of data are separated, and when multiple users can see different views of data, then the flexibility of usage, evolution, and performance of a database system is maximized. This tenet has been lost in the marketing crush of Data Warehousing, which prescribes a tight coupling of physical representation and high-level usability.

In this paper we describe the Cohera Federated DBMS, which reintroduces data independence to the heterogeneous databases present in today's enterprises. Cohera's physical independence features provide a scalable spectrum of solutions for physical design of enterprise-wide data. Its logical independence features remove the distinction between data transformation and querying, by using industry-standard SQL99 as a unified open conversion framework.

1 Introduction

One of the major challenges and opportunities in corporate computing today is to facilitate the integration of data from multiple sources. As is well known, large enterprises have valuable information stored in a number of systems and formats: multiple relational databases, files, web pages, packaged applications, and so on. The reasons for the mix of technologies are also well known, including mergers and acquisitions, disjoint organizational structures, and the difficulty and expense of migrating from legacy systems.

The integration challenge has not escaped the attention of database vendors and researchers, and over the last decade there has been a flurry of effort to develop and market techniques for *data warehousing*. The idea of data warehousing is simple: in order to integrate data from multiple sources, the data is extracted from these sources, transformed into a common schema, and loaded into a single, unified database for the enterprise.

1.1 Warehousing: Dousing the Flame of Data Independence

From the dawn of relational databases in 1970, database wisdom has argued and repeatedly demonstrated that *data independence* is key to the success of large, long-lived information stores. Codd originally justified the relational model by arguing that it

> provides a means of describing data with its natural structure only - that is, without superimposing any additional structure for machine representation purposes. [This can] yield maximal indepen-

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

dence between programs on the one hand and machine representations and organization of data on the other. [Codd70]

Today, data independence is typically decomposed into two aspects. Physical data independence divorces the storage of data from its logical representation, maximizing the ability to tune and evolve the physical storage of data without affecting existing applications. Logical data independence divorces the underlying logical representation from the many different views presented to applications or users, maximizing the ability to customize the presentation of information for purposes of convenience or security.

The lessons of data independence apply directly to an enterprise with its many data sources. The task of integrating multiple data sources should be divorced from the specifics of the physical integration - data should be able to move and replicate in the enterprise without affecting applications. Similarly, the logical representation of the data should be malleable, to allow different users to see different views of all the data in the enterprise. This is particularly important in global enterprises, where different users will speak different languages, use different currencies, support different business models, and care about different aspects of the enterprise.

Somehow, the message of data independence was lost in the crush of data warehouse marketing. Advocates of warehousing heard customers' desire for a unified *access* to enterprise-wide data, and addressed it with uniform *storage*: an inflexible centralized layout of all enterprise-wide data in a single system. The result is a chain of expensive, brittle software that is hard to configure, manage, scale and evolve. Warehouses and their associated tools are unable to provide the physical data independence that allows flexible storage. Moreover, their, proprietary data transformation frameworks are not reusable in query processing, which minimizes the opportunity for logical data independence to provide different views to different users.

1.2 Federated Database Systems: Open Systems for Integration

By contrast, a good *Federated Database System* (FDBS) provides full data independence. It ensures flexibility in how the data is stored, replicated, and represented across the enterprise, allowing graceful and cost-effective evolution of enterprise-wide databases. It also allows for multiple logical views of an enterprise's data, including user-customizable conversions of types, policies, and schemas. An FDBS extends Codd's vision of data independence to the federation. Note that a warehouse is a point in the FDBS design space, where all data is physically unified, and converted into a single logical representation of types and tables. (Figure 1).

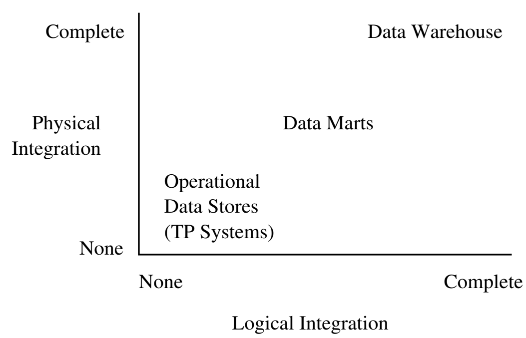

Figure1: The logical and physical integration spectra. Without data independence, current solutions only offer points in the space. Cohera provides full logical and physical independence, offering solutions spanning this space.

In this paper we contrast the integration technology of the Cohera FDBS with the available solutions in the warehouse space. We highlight the flexibility of the FDBS approach in enterprise-wide physical database design, and show how the high-performance techniques of data marts and warehouses can be leveraged as component technologies in the federated space. We also consider the effects of proprietary warehouse transformation packages on logical data independence, and contrast them with the open standard of SQL99.

2 Background: Buzzword Bingo

The research community has assimilated much of the warehousing lingo crafted by the business world. Before discussing the relevant issues, it is worthwhile to step back and have a fresh look at the terminology.

Data Warehouse: Consultant and industry pundit Bill Inmon is widely credited with coining this term. He meant by it a single, large system in which all an enterprise's data over time would be stored. Most database researchers, looking at a data warehouse engine, would instantly identify the software as nothing more or less than a relational database management system, albeit one tuned for a particular workload. *It is the workload that distinguishes warehouses from standard DBMSs*: warehouses are used in an append-only, query-mostly manner. As a result, they include support for high-volume, large-scale decision-support queries, with a focus on techniques like bitmap indices [OQ97], pre-computed results (e.g. materialized views [GM95]), aggressive query rewriting [LPS+98], and parallel query processing [DG92]. Warehouse databases are often laid out in denormalized ("star" or "snowflake") schemas, which favor high performance on queries, and low performance on updates.

Data Mart: Many organizations have found it too expensive and difficult to construct and maintain a complete data warehouse. In the absence of a single enterprise-wide data warehouse, the industrial consultants and vendors recommend constructing smaller *data marts*. These are relational database systems that consolidate the information in a particular area of an enterprise. Like warehouses, marts are targeted to append-only, read-mostly workloads. Data marts are essentially small data warehouses, and use the same technologies for enhancing query processing. Since marts are smaller than warehouses, they can also exploit some less-scalable but useful solutions, like the pre-computed multidimensional disk layouts used by some OLAP tools [DNR+97].

Who cares? Data warehouses and marts are nothing more or less than SQL database systems. So why the hype? Why do the database vendors discuss their "warehouse solutions" separately from their relational engines, when the software being used is identical? And what do warehouses and marts have to do with integrating enterprise-wide data? These questions are best answered by the marketing departments and consulting companies, of course. But the technical issues are really twofold.

First, the sheer scale of data warehousing has pushed the envelope in the design of relational database engines. In order to process decision-support queries over terabytes, vendors and researchers have been forced to deploy a whole host of the query processing technologies developed in research and industry over the years. The resulting systems represent impressive technical achievements, and have also produced well-known payoffs for some customers, particular large retailers.

The second technical issue that arises in data warehousing is the challenge of data integration. In the next section we say more about this issue, and comment on technology options for addressing it.

3 Approaches to Data Integration

Any enterprise of even moderate size has multiple sources of data: accounting systems, personnel systems, customer databases, and so on. Typically these data sources are managed by separate pieces of software. In large enterprises, there are often multiple data sources for each task, distributed geographically across the globe.

There are a number of products available for integrating the data in an enterprise. In this section we provide an overview of the extant approaches to data integration, and argue the merits of replication for physical independence, and SQL99 for logical independence.

3.1 ETL: Extract-Transform-Load

A number of vendors sell so-called Extract-Transform-Load (ETL) tools, including Informatica, Sagent, Ardent and Platinum. ETL tools extract data from underlying data sources via both native DBMS gateways (e.g. from relational vendors, ISAM, etc.) and via standard interfaces like ODBC; they then load the data into a warehouse Typically, an ETL tool also provides a facility to specify data transformations, which can be applied as the data is being extracted from the data sources and loaded into the warehouse. Most tools come with a suite of standard transformations (e.g., substring search/replace, units conversion, zip-code +4 expansion), and some extensibility interface for users to write their own transformations in a procedural language like C, C++ or Basic. Transformations can typically be composed into a pipeline via a scripting tool or graphical interface.

A main problem with ETL tools is that they were designed to solve a niche problem - warehouse loading - and are not useful for general data transformation. In particular, they have proprietary transformation extensibility interfaces: customers must program in to a specialized API or in a specialized language. Beyond imposing a learning curve on IT departments, the tight-focus philosophy prevents reuse. Transformations developed for the ETL tool cannot be reused in other data-conversion settings, particularly in ad hoc queries.

In essence, *ETL tools display a lack of logical data independence*: warehouse data can be transformed only during a physical load, not during querying. Among other problems, this prevents users from having different logical views of warehouse data. For example, once worldwide sales data is converted into dollars in the warehouse, French users cannot retrieve their local sales in Euros.

3.2 Replication Servers

Data replication servers evolved before the advent of warehousing, and typically have different features than ETL tools. In essence, replication servers perform only the extract (E) and load (L) facilities of ETL tools, but in a transactionally secure and typically more efficient manner.

The goal of a replication server is to ensure that transactions committed in one DBMS are reflected in a second DBMS. This can be used in a warehouse environment, to ensure that updates to the operational stores are reflected in the warehouse. It can also be used to support more flexible replication than the "single warehouse" model. For example, replication can be used in a master-slave mode to provide warm standby copies of important sites or tables to ensure high availability. Master-slave mode can also be used to improve performance, by allowing queries in a distributed enterprise to run on "nearby" copies of data. Replication can also be used in peer-to-peer mode, allowing updates to happen in multiple sites, with consistency ensured by conflict resolution schemes. Unlike ETL tools, replication servers do not work under an assumption that all data is loaded into a single warehouse. As a result, replication servers can be used to implement a flexible spectrum of physical replication schemes.

Replication servers are useful tools for flexible data layout, and hence can coexist gracefully with federated systems that support physical data independence. Since replication servers do no data conversion, they provide no solutions for logical data indepence.

3.3 FDBMS and SQL99: Scalar and Aggregate Functions

A federated DBMS extracts data from underlying stores on demand, in response to a query request. Data transformation is also done on demand by the FDBS. Transformations can be specified in the standard relational language, SQL99, extended with user-defined scalar and aggregate functions. Scalar functions operate a record at a time, taking a single record as input, and executing code to produce a modified record as output. In essence, scalar functions provide the utility to define simple conversions and combinations of columns. As an example, consider the following simple view that maps a table of customers from one data source, doing simple conversions: changing names into a single field of the form 'Lastname, Firstname', names of states into a standard postal code, and zip codes into the zip+4 format (user-defined functions in **boldface**):

SELECT	**concat**(**capitalize**(lastname), ", ", **capitalize**(firstname)) AS name,
	address, city, **twoletter**(state) AS state, **zipfour**(zip) AS zip
FROM	customer;

This simple example demonstrates data conversion using user-defined scalar functions, producing one record of output for each record of input.

By contrast, aggregate functions are used to "roll up" a number of rows into a single result row. With user-defined aggregation and grouping, SQL99 allows for queries that collect records into affinity groups, and produce an arbitrary summary per group. For example, consider the following view, which uses a heuristic to remove duplicate entries from a mailing list (user-defined scalar functions in **boldface**, user-defined aggregates in *italics*):

SELECT *std_last*(lastname) AS last, *std_first*(firstname) AS first, *typical_order*(order) AS std_order
FROM customer
GROUP BY address, **Name_LCD**(firstname, lastname);

In this example, the Name_LCD scalar function generates a "least common denominator" for a name (e.g. "Brown, Mr. Clifford and "Brown, Cliff are both reduced to "Brown, C"). All people at the same address with the same "LCD" name are grouped together. Then for each apparently distinct person in the cleansed list, a canonical output name is generated and a "typical" representative order is output based on all the person's orders (e.g., based on a classification algorithm, or a domain-specific rollup hierarchy).

Used aggressively, user-defined grouping and aggregation provide powerful features for data transformation: they can be used not only to convert units or replace substrings, but to *accumulate evidence* based on a number of observations, and generate a *conclusion* based on the evidence. In essence, SQL's grouping facility provides a means to gather evidence together, and SQL's aggregation facility provides a means to specify conclusions that can be drawn.

The extensibility features of SQL99 present a natural, open interface for data transformation. Unlike the proprietary APIs and scripting languages of ETL tools, SQL is the standard interface for combining and transforming sets of data. With the extensibility of SQL99, these features can be used aggressively to do custom domain-specific transformation. More importantly, user-defined scalar and aggregation functions can be reused in other SQL-based applications, which lets users leverage their investment in transformation logic for subsequent ad hoc querying. *Logical data independence requires that the transformation language and the query language be unified*; hence SQL99 is the only natural solution.

4 Cohera: Flexible, Scalable Enterprise Data Federation

Cohera is a next-generation FDBS, based on the Mariposa research done at UC Berkeley [SAL+96]. Cohera presents the best features of an FDBS: it provides the expressive transformation power of SQL99 along with a full spectrum of data independence achieved by coordinating with multiple existing replication tools. Cohera goes well beyond the state of the art in federated databases, however; the economic model pioneered in Mariposa provides the only available solution for serious *performance scalability* and *administrative scalability*.

We proceed to discuss the advantages of FDBSs for enterprise integration, and of Cohera in particular.

4.1 FDBS + Replication >> ETL + Warehousing

Cohera recognizes that the key to physical data independence in a federation is to allow replication to be carried out aggressively and incrementally. This means that the FDBS should operate on any replica of a table, or (de)normalized replica of a schema. In order to facilitate this, Cohera interoperates seamlessly with a variety of replication tools, including market leaders from Sybase and Oracle. Cohera also includes a built-in replicator that can provide extra efficiency for certain replication tasks Regardless of the replication tool used, Cohera can take advantage of whatever replicas are available in the federation, and choose the most efficient replica dynamically for each query.

By contrast, ETL and Warehousing provide a single point on the physical and logical spectra, breaking the notion of data independence. As a result, warehouse shops need to invest in a centralized computing environment large enough to hold all of an enterprise's data. This requires an enormous investment in equipment and software, and in the skilled administration required to keep such installations running. As one example from the real world, a large telecommunications firm plans employs warehouses and finds good return on investment from implementing them. However, before taking on the time, expense, and headcount to implement and support another warehouse, the firm wants to understand whether the data integration will be useful. A federated database will be used to deliver a logical view of multiple sources without requiring physical co-location. In effect, the FDBS delivers prototyping and migration facilities even for warehousing advocates.

An additional benefit of an FDBS over warehousing is the timeliness of data: an FDBS can provide access to live data from operational stores, rather than outdated data fed into a warehouse once a day, week or month. The tradeoff of timeliness and the impact on operational stores is one that can be made flexibly, depending on query vs. transaction load; this is another benefit of the data independence afforded by FDBSs.

Finally, note that since FDBSs and replication use standard SQL99 as their transformation scheme, existing tools and techniques for materialized views can be integrated with the replication process. This allows FDBSs to span the the logical/physical spectra of Figure 1, by allowing not only tables but also views (with SQL99 transformations!) to be either logical or physical.

4.2 Cohera $>>$ 1st-Generation FDBS

Like Mariposa, Cohera integrates underlying data sources into a *computational economy*, where data sources cooperate in distributed query processing through a metaphor of "buying" and "selling" their services to each other. Unlike Mariposa, Cohera explicitly focuses on federating heterogeneous systems.

When an SQL query is submitted to a Cohera site, a module at that site called the *contractor* translates the query into a single-site execution plan. The contractor must consider each operation in the execution plan - e.g. scanning a table, or performing a join - and choose the most efficient data source to execute that operation. To do this, it solicits *bids* for each constituent part of the plan: for example it might solicit bids for the operation of scanning a table called "sales". This bid is sent to other Cohera sites (*bidders*) that manage their own underlying stores. If a given bidder site manages a database with a copy of the "sales" table, it might choose to bid on the operation. In this case it uses its own local rules to place a cost on the operation, and it ships its bid to the contractor. The contractor collects bids for all the individual operations, and constructs a distributed query plan that minimizes the cost of the whole query.

This economic model provides enormous benefits over traditional FDBS designs. First, the economic model ensures *local autonomy* for managers of databases in the federation, who can control bidding policy based on local constraints (e.g., "during 9-5 only bid on federation queries from the office of the CEO".) Second, the economic model gives local managers the facility and incentive to participate in enterprise-wide *load balancing*: a standard bidding policy is to have the bid price be proportional to the product of work and local load average, which naturally spreads work to under-utilized servers. Third, the economic model provides *scalable performance*, by distributed the query optimization and scheduling process. First-generation FDBSs had centralized optimizers that needed to know the state of all machines in the federation in order to operate; this prevented scaling beyond a small size. Finally, federating the query optimization process provides *administrative scalability*, by allowing local administrators to set costs and policies based on dynamic properties (load balance, time of day, installation of new hardware, etc.); the centralized optimizers of first-generation FDBSs limited the autonomy of local administrators, since the centralized optimizer had to have control to work correctly. A lack of autonomy requires consensus among database administrators, which is politically (and often geographically) infeasible in an organization of any size.

As a final contrast between Cohera and earlier systems, note that the "open marketplace" in Cohera allows sites to be added and deleted over time. Coupled with replication, this facility gives an enterprise an incremental path to migrate along the spectrum of Figure 1. Note that a warehouse is not ready for production until *all the data is loaded*. By contrast, a federation can start out as a loose affiliation of a few operational stores, expand to include a denormalized query site (e.g. a data mart) to enhance some queries, and eventually grow to span the enterprise. This minimizes the management and cash flow needed at any point in time. It also allows for incremental upgrade of components, rather than a complete swap of old iron for new iron. Note that incremental upgrade means that you are always buying a little of the latest hardware at the lowest price, rather than buying a lot of today's hardware at today's prices to plan for tomorrow's capacity.

5 Conclusion

Enterprise data integration is one of the most pressing problems in IT today, and a subject of much research interest. Cohera's solution to this problem is to ensure full data independence: Cohera provides a flexible, scalable infrastructure for enterprise-wide data storage, and a unified transform/query interface based on the open standard of SQL99. Cohera's unique scalability is the result of its economic computing model, which scales both in performance and administrative complexity.

Bibliography

[Codd70] Codd, E.F. "A Relational Model of Data for Large Shared Data Banks." Communications of the ACM 13(6):377-387, 1970.

[DG92] DeWitt, D. J. and J. Gray. "Parallel Database Systems: The Future of High Performance Database Systems." Communications of the ACM 35(6): 85-98, 1992.

[DNR+97] Deshpande, P., J. F. Naughton, K. Ramasamy, A. Shukla, K. Tufte, amd Y. Zhao. "Cubing Algorithms, Storage Estimation, and Storage and Processing Alternatives for OLAP". Data Engineering Bulletin 20(1):3-11, 1997.

[LPS+98] Leung, T.Y.C., H. Pirahesh, P. Seshadri and J. M. Hellerstein. "Query Rewrite Optimization Rules in IBM DB/2 Universal Database", in Stonebraker and Hellerstein (eds.), Readings in Database Systems, Third Edition, Morgan-Kauffman, San Francisco, 1998.

[GM95] Gupta, A. and I. .S. Mumick. "Maintenance of Materialized Views: Problems, Techniques, and Applications."' Data Engineering Bulletin, 18(2), June 1995.

[OQ97] O'Neil, P. E. and D. Quass. "Improved Query Performance with Variant Indexes". Proc. ACM SIGMOD Conference, Tucson, June, 1997, pp. 38-49

[SAL+96] Stonebraker, M., P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, A. Yu. "Mariposa: A Wide-Area Distributed Database System." VLDB Journal 5(1): 48-63 (1996)