Data Legibility In Decision Support Systems

Marc Demarest
demarest@hevanet.com

Revision 7.4
November 2001

ABSTRACT

Data legibility -- knowledge workers' ability to understand fully and immediately the data contained in a decision-support system (DSS) -- is the single most difficult problem in client/server DSS design. To be effective, DSS environments must provide readable, clear, and navigable data to the knowledge worker or end user. Also, the data must present itself in a fashion consistent with knowledge workers' business models, and must promote swift and simple formulation of business questions consistent with knowledge workers' goals.

A version of this paper appeared in *DBMS Magazine* in May of 1994 (v7 n5 p55). This appearance was the first systematic treatment of the concept of multi-dimensional schema (MDS), also known as data cubes or star schema, and the relationship between MDS design and large-scale decision support.

Although the concept of multi-dimensional schema design, and the architectural model of combining a single warehouse and multiple marts into an enterprise-wide decision support environment, are now taken for granted by most DSS architects and system builders, at the time the article was published, the concept of denormalization was extremely controversial.

Introduction: Understanding Data As A Computer User

Data legibility -- knowledge workers' ability to understand fully and immediately the data contained in a decision-support system (DSS) -- is the single most difficult problem in client/server DSS design. To be effective, DSS environments must provide readable, clear, and navigable data to the knowledge worker or end user. Also, the data must present itself in a fashion consistent with knowledge workers' business models, and must promote swift and simple formulation of business questions consistent with knowledge workers' goals.

Organizations interested in DSSs, but not familiar with the problem of data legibility, typically employ replication or consolidation strategies to build their initial systems. Because these strategies do not take data legibility into account, the DSS environments are usually significantly less useful to knowledge workers than they should be. Conversely, organizations interested in DSSs, and familiar with the problem of data legibility, typically employ reschematization or aliasing strategies to rebuild production data into DSS data, which produces significant, strategic DSS environments.

The choice of a data legibility strategy -- reschematization or aliasing -- has a number of DSS performance implications and depends on a combination of factors, including the skill level, scope of operations, and business requirements of the knowledge worker community; the corporate strategy with respect to software development; and the cost model and constraints associated with the DSS project itself.

But first, let's take a look at why data legibility is important in the workplace.

demarest@hevanet.com

Data Legibility

To build a DSS environment that connects knowledge workers directly to the corporate data they need in order to make effective, informed, and comprehensive business decisions, the DSS data on which knowledge workers operate must be legible; that is, it must be easily understood, assimilated, and manipulated. Because data retrieval from the DSS data store is just the beginning of a complex decision-making process, misunderstandings or misinterpretations of the DSS data caused by legibility problems will always result in flawed or misdirected decisions, which immediately destroys the customer or shareholder value at stake in the decision-making process.

Knowledge workers can immediately and intuitively recognize legible data as either a business object (invoice number, employee ID, customer, and so forth), a component of a business model employed by the knowledge worker in the decision-making process (markets or products, for example), or a dimension of time (day, week, month, quarter, or year). You can intuitively recognize the words on this page without consulting a dictionary or asking a coworker for help; legible data should be equally as obvious. It is there for knowledge workers to seize and use.

Unfortunately, the data from OLTP and other production systems drives most initial DSS implementations. Computer processes serving data-entry personnel enter this production data, which is then stored in systems that are designed and maintained by database administrators (DBAs) and system programmers. While the data in these production systems is no doubt legible from the DBA's perspective, it is usually illegible from a non-technical knowledge worker's perspective. This illegibility has implications for data delivery, the DSS data store, and the client applications accessing the data store.

demarest@hevanet.com

Replication and Consolidation

Organizations embarking on their first DSS implementations typically provide data to an end-user organization by copying data from one or more production systems to another system. They then open this data warehouse or store to knowledge workers who, in theory, use off-the-shelf, client-based applications to access and retrieve relevant data. This approach attempts to alleviate the problem of query drain -- a common operation problem in which the CPU and I/O drain from a single complex query temporarily impedes a production system's OLTP performance. Ironically, IS professionals often report the same phenomenon when these warehouses and stores are first opened to knowledge workers. Subsequently, the usage pattern changes, as follows:

1. An initial access surge occurs immediately after the warehouse or store is opened for use.

2. Almost immediately thereafter, use drops of dramatically. After a few weeks, only a handful of users implement the DSS.

3. Subsequent interviews reveal that a select group of technically inclined end users in the original target end-user community perform all the query and reporting work against the warehouse or store, for both themselves and other users.

This trend is easy to understand. Most end users cannot understand the data they have been given because it is illegible. There may be hundreds of tables, each with cryptic names, containing thousands of equally cryptic columns. Data, such as dates, which should be commonly understood, appear as strings of apparently random numbers (for example, dates may be stored in an M4DATE format appropriate for OLTP transaction-level data). And finally, assuming the end users understand the concepts of joins and foreign keys, multiple join path seem to exist between any set of tables, suggesting multiple ways of asking the same question.

Despite these deficiencies, IS professionals persist in delivering data to end users in the form of normalized schemas with cryptic data sets and multiple complex join paths. In many cases, this situation exists because data normalization is a bedrock IS process -- the foundation on which data integrity in the enterprise is based.

Normalization, as a complex technical discipline, has several advantages for data integrity in an OLTP environment, but works strongly against data legibility in a DSS environment, as follows:

- A large number of tables. The goal of schema normalization is to separate independent bits of information into logical units (columns and tables) that roughly map to entities such as customer or shipper, or to attributes (or characteristics) of those entities. It is common, in complex activities such as manufacturing, to find databases with more than 1000 tables.

- Multiple join paths between any set of tables. Because of the sheer number of tables, answering simple business questions such as, "Which customers in the European

demarest@hevanet.com

community purchased goods in excess of $10,000 or 50 units during the first quarters of 1990, 1991, and 1992?" presents a significant obstacle to the business user. Even if users can determine which of the 1000 tables may contain parts of the answer, they must determine how to join those tables. Unfortunately, the tables share multiple keys and can be joined in several different ways. Worst of all, users discover that the answer to the business questions differs substantially depending on which keys they use and the order in which the tables are joined. Also, the user has no tools to determine which possible answer is correct.

- Cryptic naming conventions and data formats. As trivial as it may seem, one of the primary reasons why business users cannot read normalized schema is that normalization encourages cryptic, shorthand naming conventions for columns and tables. These naming conventions often require a glossary for translation into end-user terms. While such naming conventions are not useful for OLTP applications, they are not useful for DSS applications. When OLTP applications privy to naming conventions access the database, end users to do not see table or column names, and problems do not arise.

- In addition, normalized schemas store data in illegible forms. Dates, for example, are commonly stored as a long number representing "the number of days since April 1, 1971" or some mathematical construct. While a computer program can calculate the date very nicely from this number, a human user can make little sense of it.

All these characteristics support data integrity, which is perfectly appropriate for OLTP environments. However, none supports data legibility, which works against the DSS environment's business goals. Business analysts are resolutely non-technical; most do not understand the concepts of joins, foreign keys, or primary keys, nor do they understand the design principles (that is, normalization) behind the data capture process. Finally, business analysts pursue nonlinear, heuristic, and sometimes wholly intuitive avenues of analysis and discovery, using tools designed to support these interrogations. Unfortunately, these discovery processes can be misinterpreted and their value lost when the data does not cooperate.

Normalization supports the DBA's agenda and the company's goal of data integrity within the data-capture process and the OLTP environment. But it ultimately results in an expensive DSS that does not support corporate decision making, and is not used by the majority of business analysts because they cannot understand the system's information, and therefore cannot construct basic business questions.

demarest@hevanet.com

Two Data Legibility Strategies

Once data legibility is thoroughly considered in a database design process focusing on decision support, DSS architects typically opt for one of two strategies to achieve legibility:

1. Reschematization. Decision-support data is extracted from one or more normalized schemas into another denormalized schema designed to end-user specifications, and supporting the business model of the people asking the business questions.

2. Aliasing. Using one or more of the technologies described later, a data "model" approximating the business analysts' business models is interposed between the business analysts and a data set. This model typically remains highly normalized and illegible.

Both strategies offer the potential for legible data. The choice of approach is or should be, based on the business goals of the DSS system itself, the stated and derived needs of the analyst communities, the organization's strategic IT directions, and a simple cost/benefit analysis of the two strategies applied to the given situation.

Reschematization

DSS architects committed to reschematization as a data-legibility strategy typically have four common characteristics. First, they understand their end users' business models and decision-making processes, and are committed to implementing DSS environments supporting those models and processes. Second, they are committed to enabling the personal productivity tools already on knowledge workers' desktops, and buying commercial off-the-shelf technology rather than hand-coding DSS environments. Third, they are interested in client/server computing as a next-generation business-automation strategy. And fourth, they are interested in harnessing the relative power of the server to bear most of the data-legibility burden.

Furthermore, DSS architects committed to data reschematization understand that a database schema can be used as a powerful tool to alleviate the already excessive processing burden on client workstations. The database schema can also eliminate the need to write and maintain custom client/server software, and is able to maintain an optimal degree of flexibility with respect to knowledge worker's changing information requirements. Focusing on the design of the DSS database can result in all of these benefits.

You can align database schemas somewhere along the spectrum shown below.

demarest@hevanet.com

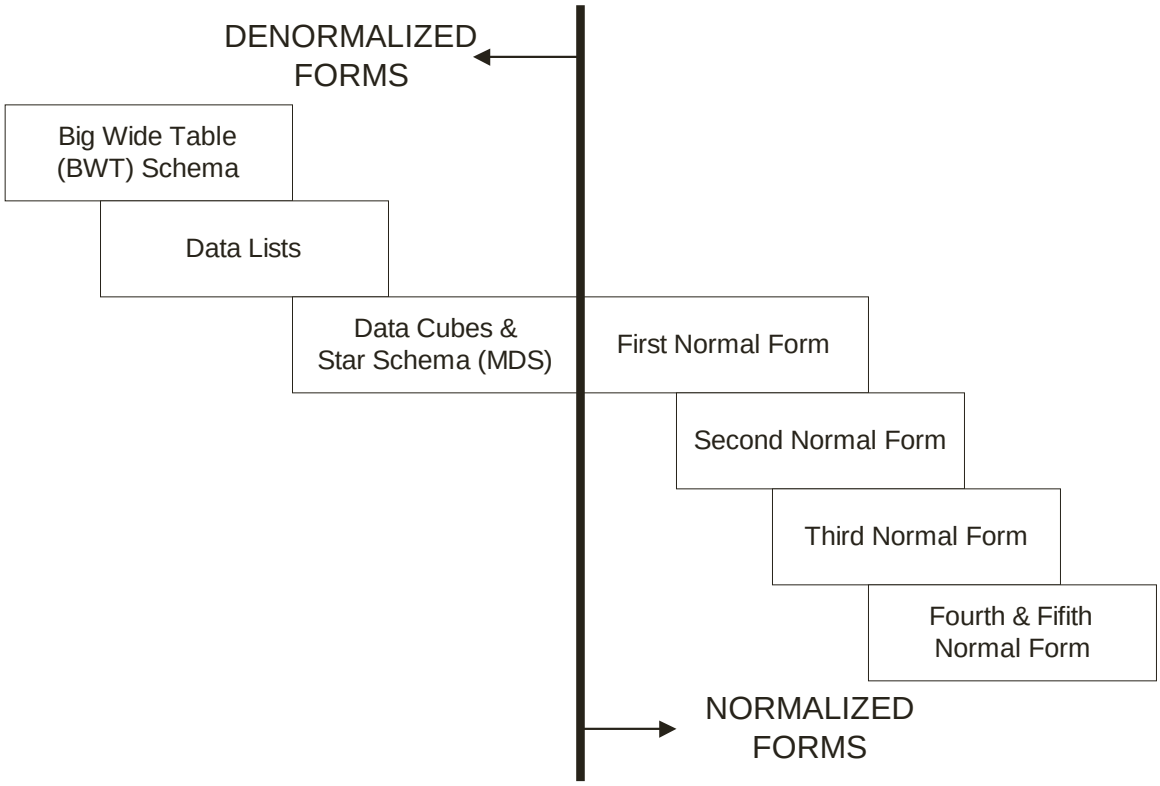

Figure 1 -- Schema Spectrum

Typically, most OLTP schemas are normalized to third normal form; the fourth and fifth normal forms are difficult to design and typically unnecessary for meeting commercial data-integrity requirements. The vast majority of decision-support schemas, which are implemented in spreadsheets on individual knowledge workers' desktops, are single-table schemas (that is, big wide tables [BWTs] custom-built by the knowledge worker, usually by reentering data into the spreadsheet application or into a PC database like Microsoft Access from one or more IS reports).

After working with a generation of spreadsheet and PC LAN flat-file database managers, business analysts understand and use BWT schemas daily. Although you can theoretically obtain a BWT schema from any normalized schema in one operation (called a full outer join), BWT schemas are impractical in large-scale decision-support applications because they are so big: The resultant table is too wide (sometimes containing hundreds so columns) and too long (often in the hundreds of millions of rows) to be directly navigable.

In addition, BWT schemas are inefficient in two dimensions: They do not use disk space efficiently and cannot be processed efficiently at query time when the data set is more than a few megabytes in size. As such, DSSs have, over the years, developed decision-support schema models that maintain the required legibility levels, but are closer to normal form schemas than BWT schemas. Two of these prototypical schemas -- the

demarest@hevanet.com

data cube and the data list -- are worth exploring to understand how decision-support schemas can support the question-building process underlying a DSS.

The Data Cube

The data cube, championed by Red Brick Systems and other DSS vendors, is based on three fundamental insights about the business models that analysts use during the question-building process:

1. Analysts are always after facts and statistics, such as units sold, dollars in revenue generated, and so forth.

2. Analysts always approach facts through business dimensions; for example, the business's products or services, the business's target markets, and the channels through which products or services reach these markets.

3. Time is always important to business analysts.

Thus, a data cube supporting a drug store chain's marketing department would look similarly to the diagram below.

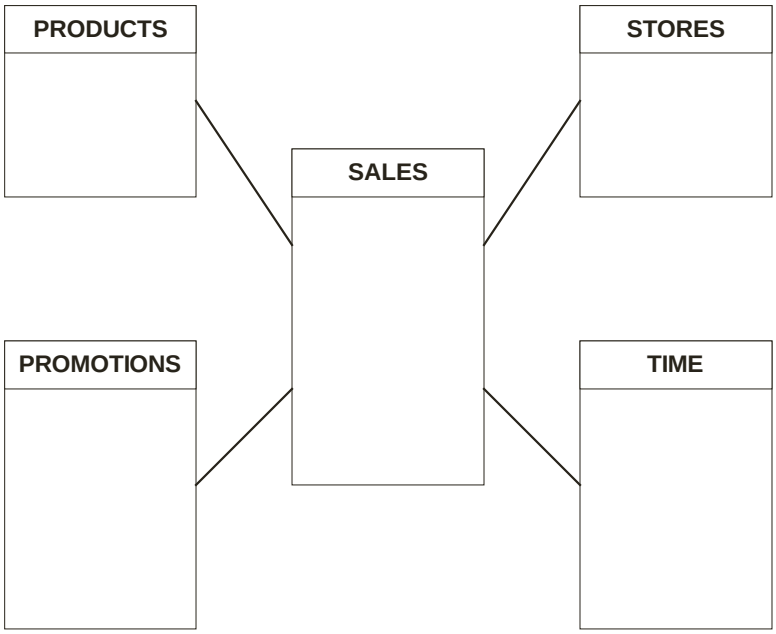

Figure 2 -- Sample Data Cube Schema

The Sales table in this example contains all the business facts: units sold, dollars sold, payment method, and so forth. All the remaining tables are key business dimensions that are typically organized to reflect the natural organization of those dimensions. For example, suppliers provide products to the company for sale in the retail stores, while stores are typically organized into regions, each of which is made up of districts with more than one store. Therefore, the dimensions reflect the knowledge worker's model of

demarest@hevanet.com

that dimension -- the way the people asking the business questions perceive the business.

The data cube is admirably suited to several common question-building models, including the progressive refinement model, in which a question begins simply. "Tell me which stores sold suntan lotion," and ultimately reaches the desired level of specificity and complexity: "Tell me which stores in Southern California sold more than 1000 units of the Delta Products Number 2 suntan lotion supplied by Acme Distributors during the months of November and December"; and the cross-temporal model in which a question involves multiple points in time? "How is suntan lotion selling an Montana now versus this time last year and the year before that?"

In addition, the data cube promotes high levels of legibility within the data itself. For example, dates within the Time table are always organized into a clear and immediately legible hierarchy of time, in which years are divided into quarters, which are subdivided into months, weeks, and days. The column names of the Time table are year, quarter, month, week, and day. Moreover, there is one -- and only one -- obvious join path to retrieve data from the Fact table; no mistakes are possible. Some DSS technologies designed for the data-cube schema do not require that joins be specified to process queries.

Intellectually, the data cube, or "star schema," has several important antecedents. In fact, the first examples of "normalized" schemas defined by Codd and Date in their groundbreaking normalization work were data-cube schemas. Only later did the normalization discipline begin to produce complex normalized schemas with hundreds of tables and multiple join paths.
Database design experts at IBM, Metaphor, and other DSS technology leaders have worked for years with variations of the data-cube schemas; although they often call these structures multidimensional databases, multikey files, or grid files. In addition, PC software vendors have been working for years on multidimensional spreadsheet technology embodying the concepts underlying the data-cube schema. Today, the data-cube schema is one of the best practical methods for implementing legible DSSs.

The Data List

Like the data cube list is based on a set of fundamental ideas about how end users process information. But unlike data cubes, which assume a set of facts and a set of dimensions to drill into those facts, the data list recognizes that 1) some of the work that analysts do is within one or two dimensions, and 2) some DSSs support a business model in which only two important objects exist: customers and transactions.

A data-list schema supporting the marketing department at a mail-order catalog house would look similar to the figure below.

demarest@hevanet.com

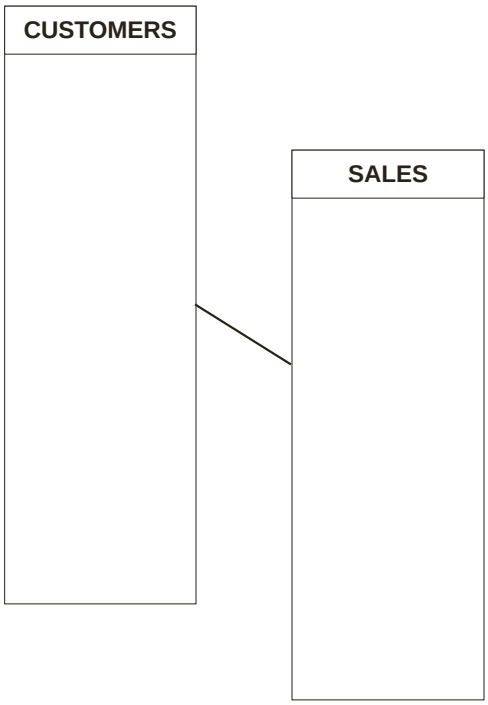

Figure 3 -- Sample Data List Schema

All relevant customer information -- name, address, zip code, credit limits, and so forth -- is stored in a relatively static Customer table, while all the information about sales -- product sold, quantity, cost, shipment method, and so forth -- is stored in a relatively dynamic Sales table. There is one join path, which is probably facilitated by a common key such as customer ID.

Because data lists reflect a basic understanding of the knowledge worker's business models, they are particularly well-suited for some proprietary DSS environments like Teradata. For this reason, retail sales DSS designers and DSS architects in telecommunications companies frequently employ data-list schema when using these kinds of proprietary systems, but tend to move to data cubes when using conventional RDBMS technology or proprietary MDDBMS technology (sometimes referred to as OLAP).

Benefits of Reschematization

Reschematization avoids the problems associated with normalized schemas, and embeds data legibility into the DSS data store itself. Instead of designing the schema to support a technical goal such as data integrity, legible schemas embody (and therefore enforce) the business models already in use within end-user communities. This makes the data immediately legible to knowledge workers. Therefore, knowledge workers can use the data schema to frame and clarify their questions to ensure that the answers they retrieve are the only possible answers. In turn, this immediate legibility lets knowledge workers use commercial off-the-shelf client tools to access and analyze the data.

demarest@hevanet.com

Costs of Reschematization

Reschematization involves more design work than does replicating normalized schemas, and requires that data destined for DSSs be reengineered as it moves from the production systems capturing the data to the decision-support store. As such, the data-extraction process becomes more complex as columns of data are reorganized, transformed, and eliminated from production source data.

Reschematization in many IS organizations also requires that data architects and DBAs put aside their training in normalization techniques and return to more business model- and user-oriented database design techniques. In addition, off-setting the relatively higher costs of data extraction, reschematized data forces the cost of client/server connectivity to its lowest possible point by enabling organizations to purchase, rather than build, their client/server connectivity options.

demarest@hevanet.com

Aliasing

Reschematization sets out to make data immediately legible to end users. Aliasing strategies interpose, between and data and end users, one or more "models" that translate or map between end-user business models and data may be completely illegible to decision makers, as shown below.

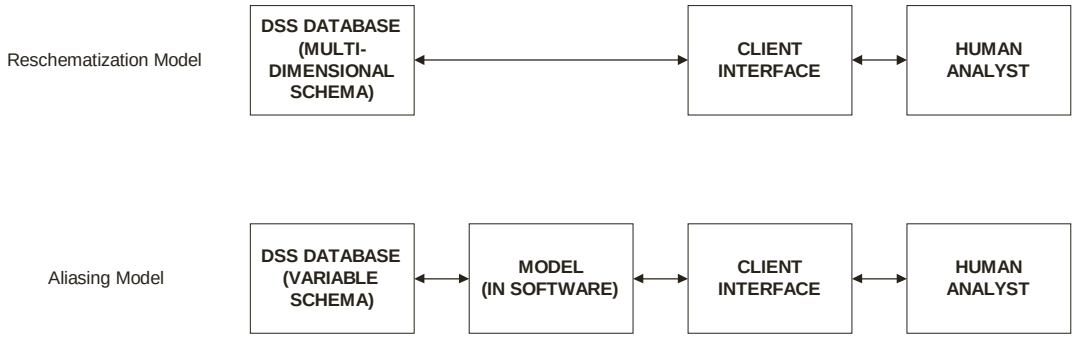

Figure 4 -- Aliasing Versus Reschematization

For this reason, aliasing is sometimes called model-based DSS, while reschematization is referred to as data-based DSS.

- DSS architects committed to aliasing as a data-legibility strategy have several of the following characteristics:
- They may or may not understand their end users' business models.
- They are committed to implementing DSS environments that support IS guidelines regarding normalization.
- They have access to substantial programming resources or sophisticated design tools in order to build the decision-support model software.
- They work in IS organizations committed to building custom client applications for DSSs.
- They are interested in client/server computing as a next-generation business automation strategy but will continue to support host-based, terminal-oriented decision support.
- They support business models that can be coded into a DSS application without fear of frequent rewrites (because the business models are long-lived).
- Furthermore, these DSS architects believe that database schemas are not the proper place to focus on database legibility, rather, they believe that all OLTP design model is appropriate for DSSs. They also feel that the cost of data reengineering implicit in reschematization is too high.

demarest@hevanet.com

Aliasing strategies typically result in one of the application architectures illustrated below.

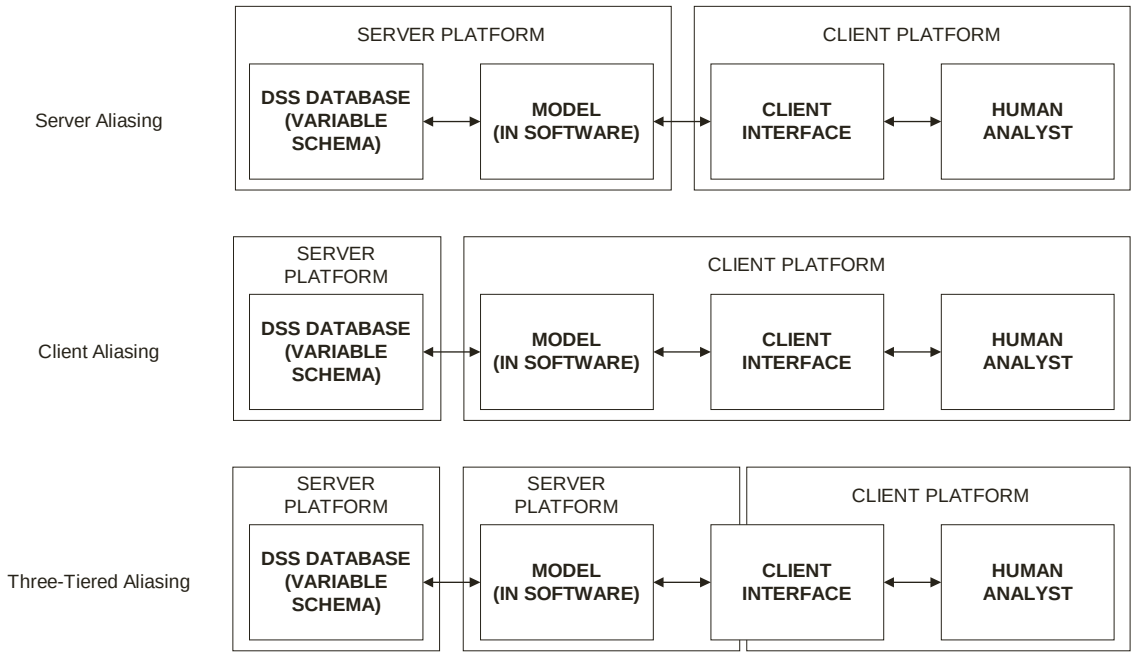

Figure 5 -- Aliasing Architectures

A client aliasing application provides a client-resident mapping between an end user's view of the data and that data itself. A server aliasing application divides the application architecture along identical lines, but places the mapping software on the server platform, with the data set rather than the end-user tools. Repository models adopt a three-tiered client/server architecture, placing the modeling application -- typically in the form of a metadata repository -- in a location distinct from either the end-user tools or the data sets on which the repository keeps metadata.

Client Application Aliasing

Client application aliasing strategies, supported by the explosion of PC-based rapid application development (RAD) tools, are by far the most common aliasing strategies. In a nutshell, these strategies call for an "intelligent" application that presents a visual (iconic or pictorial) representation of data that end users can manipulate. It can then generate SQL and retrieve data, based on the end user's path through the application. This kind of aliasing strategy is used typically in environments where the sophistication of the end user is a major concern, and where the available commercial off-the-shelf tools are considered sufficiently inadequate to warrant substantial in-house investment in application development. In the commercial software marketplace, products such as Business Objects from Business Objects Inc. have adopted this client application strategy by letting DBAs encapsulate SQL fragments and entire queries as commonly used business objects for end-user manipulation.

demarest@hevanet.com

Server Application Aliasing

Server application aliasing follows the same general application development model as client application aliasing, but makes a different choice with respect to the model's location. Typically, this kind of strategy places the modeling application on the server rather than on the client for several reasons:

- The DSS environment's goal is to provide end-user access to a "report catalog" in which reports are maintained on the server.

- The problems associated with version control of client applications in large client/server environments (with more than 100 client workstations) is difficult and expensive.

- The client workstation is not seen as reliable in terms of mean time between failure or general stability.

- The modeling application uses next-generation artificial intelligence (AI) or expert-system technology, which requires central management and more processing power than a client workstation can provide. This technology also feeds on all the data in the DSS data store to answer business questions.

- The model embedded in the modeling layer is seen as a shared resource and is therefore located on the server platform.

Holos from Holistic Systems Inc. and IDIS from Intelligence Ware Inc. are examples of top-flight commercial DSS applications that implement a server modeling application. In fact, so-called OLAP applications generally ought to be seen as proprietary aliasing applications employing server-based modeling. Unfortunately, the limitations of most OLAP technologies' MDDBMS components – proprietary databases built into the aliasing application to solve a data access problem that disappeared almost a decade ago – limit the usefulness of these applications in many cases.

Repositories and Directories

Organizations that have embarked on enterprise data-modeling activities, or are bent on giving end users direct access to production system data, typically turn to repositories to provide the aliasing and mapping functions for client applications. In this version of the aliasing architecture, client applications interact with a directory that, at the very least, contains instructions for locating each DSS data source on the network. The applications also interact with this directory when interpreting each DSS data source's system catalog. The client application then uses this information to represent the combination of several physical data sources as one data set to the end user, who can then "join" data sets from different physical databases.

In extremely sophisticated cases, the repository actually provides a nonstandard query language to allow client applications, which are still custom-coded, to interact with the repository. The repository then translates the client application request into a set of SQL queries to launch against multiple data sources.

demarest@hevanet.com

While there are no full commercial examples of this technology, some companies have cobbled together their own working versions. Repository strategies generally require a well-established enterprise data architecture, an extreme amount of available network bandwidth, and sophisticated development tools that are not yet readily available in the commercial marketplace. They also often reinstate the original DSS problem: query drain on production systems and applications.

Benefits of Aliasing

Ideally, aliasing strategies allow for a finely tuned match between end users' business models and data sets that have not been reschematized for DSS. In practice, aliasing strategies have only been able to achieve the same level of legibility provided by reschematized data stores -- making the choice between reschematization and aliasing largely a matter of cost comparison over the average life of the DSS application.

Costs of Aliasing

The costs of aliasing obviously involve the development and maintenance of the software providing the aliasing function. These costs are well-understood and substantial. As such, the key question when evaluating aliasing strategies is how much value the application provides beyond simply mapping between the end user and data.

In some technical applications, such as AI or expert system-based decision support, server-based aliasing architectures have been instrumental in putting the resource-hungry AI and expert system engines at the disposal of business analysts. In business environments such as healthcare, where data is complex and associated with behavior, the growth of model-based DSS is natural and appropriate. Even healthcare-practice managers have difficulty comprehending the complexity of the relationships between healthcare providers, regulatory agencies, and clients, and require models to "remember" and present intuitively the nature of those relationships. In other environments, knowledge workers thoroughly understand their business models and are sufficiently skilled with their desktop tools to dispense with intermediate aliasing if the schema itself is legible.

demarest@hevanet.com

Conclusions

Data legibility requires an architecture designed to produce easily assimilated data. Both reschematization and aliasing strategies offer approaches to achieve data legibility, and have historically produced similar results, leaving the choice of strategy to be determined by comparative cost, as well as IS and corporate strategies.

Aside from repository-based aliasing strategies that are not yet commercially viable, the question of which data-legibility strategy to choose cannot be answered definitively on technical grounds. Simple replication and consolidation strategies do not provide data legibility, but the strategies providing legibility, including data reschematization, client application aliasing, and server application aliasing, are all technically feasible. As such, the strategic decision must be made based on business automation, cost, and performance criteria.

Reschematization strategies are more in line with organizations that are committed to the "buy side" of the buy-versus-build decision, as well as with those companies decreasing in-house application development and maintenance, and data-enabling their knowledge workers' personal and team productivity environments.

Aliasing models are more in tune with organizations that favor building rather than buying. These companies have substantial in-house client/server software development and maintenance expertise, and believe that custom client applications will deliver substantial end-user value over commercially available substitutes.

In the long term, reschematization is preferable in terms of deployment and maintenance costs. If the connections between the production systems and the data warehouse are well-designed, and data reengineering is performed on the data store instead of the production system, reschematization delivers legible data while avoiding costs associated with application software development and maintenance.

Performance is not a characteristic of architectures; it is the result of an architecture's implementation in a system configuration. Because reschematization has a minimal number of software layers, it is likely to perform better than aliasing. However, the ultimate performance of the DSS environment depends on the system's server and DBMS components.

demarest@hevanet.com