

# A Thought Paper on the Economics of Data Warehousing

Kevin Lacobie  
President  
Agoric Source, LLC  
February, 2004

The "enterprise data warehouse" always evokes an image, to me, of the final scene from *Raiders of the Lost Ark*, where the crated and locked Ark is being loaded into a huge, massive warehouse packed yards high with other similar-looking crates. One of the bureaucrats remarked that the Ark was now "safe" in government hands, but the implication was that it was safe only because it would become completely inaccessible, lost in the deep, dark warehouse chambers.

Enterprise data warehouses seem to be places where "data checks in, but never checks out". Large volumes of data are stored on enterprise-scale mainframes or mini-computers, but it's difficult to navigate through the numerous tables or write any basic reports without large-team programming support. These large, often multi-million dollar warehouse systems do indeed store tremendous amounts of corporate data, but don't seem to serve their purposes very well. Perhaps it's because their purposes are not well defined, and it's a case of technology moving first, before identifying user needs.

Enterprise data warehouses may serve a purpose in the corporate infrastructure, but my analysis of the economics of data warehousing suggest that data warehouses best serve *departmental*, not *enterprise-wide* interests. This is especially important for departmental managers and directors to understand, since it is they whom must decide whether to rely on an enterprise warehouse or deploy their own, and to carefully control how a data warehouse may serve their department's needs.

Data warehouses come with very identifiable costs: additional hardware, software licensing, additional maintenance chores, etc. Perhaps the most obvious cost of all is that they seem very *redundant*. The data already exists in some original, official, location -- any attempt to duplicate it also duplicates the costs for storing it in the first place. Duplication also violates one of the central tenants of pure data design, a computer science principle known as *normalization*, which, in its simplest form, means no data shall be duplicated. Unnormalized data, the theory warns, adds to the cost of storage, and is difficult to modify without producing inconsistency between tables or databases.

However, with these additional costs come the potential for some significant benefit. Various economic pressures, some real, some not, have caused an explosion in data warehousing initiatives in the past several years, so there's a least some implicit benefit driving this growth. However, what are the real benefits?

From a purely engineering point of view, data warehousing involves one or more of these three computational strategies:

- *Collection* - storing data in a central source
- *Consolidation* - joining together data from disparate sources into a uniform, "relationally compatible" format
- *Conforming* - applying computational or business rules that promote a uniform meaning to data

These strategies have some inherent engineering benefits to them (as well as costs), but we're interested in the business benefit that each strategy brings. These strategies and benefits will be examined in detail below, but in

Departments face a tradeoff between utilizing the enterprise data warehouse and employing their own data warehouse infrastructure.

The cost side of the equation: *Redundancy* ~ doubling of costs. But, this is an exaggeration, especially in the case of departmental data warehouses.

summary, a shorthand benefits equation for these strategies is:

- *Collection = Faster Performance* (satisfying user needs for reports and analysis more quickly)
- *Consolidation = More Features* (satisfying report/analysis needs that cut across multiple data sources)
- *Conforming = Increased Data Quality* and an automation of departmental business processes

## Warehousing for Performance

In its simplest form, a data warehouse is nothing more than a reporting server. Data is duplicated -- that is, collected -- to a second server by one of many techniques (all of which have various technical tradeoffs, but include database mirroring, backup/restore processes, import/export processes, etc.), and the data sits on the second server solely for report writers and other analysis programs to read from. Data on the second server is always static, or "dead"; it is never updated, modified, or otherwise manipulated (this is a trait repeated with all warehousing architectures -- warehouses are for storing, not managing, data).

The data need not be an exact copy of the original source; often, the data that enters a reporting server warehouse is summarized from its detailed form on the original server. This summarization serves one of the fundamental needs of a reporting server: the need for speed. Trying to use the original server -- whose primary purpose is to serve a corporate application needs, including processing, data updating, user interfaces, etc. -- for reporting is often an unsatisfying experience; users want a report to pop up within seconds of requesting it, but the typical experience of a moderately complicated report is more in the order of minutes from request to display. Another typical experience is that more complicated reports can even "crash" the application server by taking up so many CPU cycles in generating the report that other application uses are starved for cycles. Attempts at prioritizing use usually leads to further disappointment, especially for the report user. An application user or process may be working with only a single, or small number of records, while the reporter often needs to report on all the records on a database, thus consuming much more processing resources, and if his CPU priority is low, reports can take intolerably long to complete.

Poor performance in these cases is due to two reasons: a) scarcity of computing resources, and b) internal conflicts of the relational database model. If all users and processes within a corporation must use the same computer resource, eventually that resource will become scarce, and all users' performance will suffer. Since reporting is a well-defined, separate task that doesn't depend on dynamic data (and in fact, most often requires that all the data be "set in stone" before releasing a report), it became natural to separate these tasks onto a separate server. In a typical financial reporting environment, for example, data that is duplicated to a reporting server is one day, or one month old; not a good solution for "real-time" reporting, but is perfect for historical reports, especially financial reports that can only be released after a period closing. So, by separating uses onto separate servers, resource scarcity is eliminated, and all users will see better performance.

However, this separation is also dictated by the nature of the relational database model. Relational databases (RDB for short), also known by its scripting language, Structured Query Language (SQL), are the most common database model in corporations today. One fundamental facet of the RDB model is the use of indices to index the data in its tables. An index, sorting all employee names by last name, for example, allows for faster access to a

Performance gain #1: Separate users, to reduce competition for scarce computational resources.

particular record in a database, or for reporting data in a specific order, or for joining information from one table to another. Good indices mean faster performance for reports. However, indices must be maintained: every time a record is created or updated, all the indices that reference that record must be updated as well. More indices slow down users or processes that must create or update records.

Designing optimal indices for use in an environment that includes both reporting and updating requires a lot of analysis of how the data is used, and of course, some tradeoffs. Even in an "optimally" indexed environment, some of the database users will experience slower performance than expected, or what could be optimally provided in ideal environments. The more sophisticated, commercial RDB systems today do an excellent job at index optimization, and their performance is visibly superior to their cheaper brethren. But, in large-scale commercial use, even these systems can perform poorly.

Performance gain #2: Develop indices optimized for reports.

So, a successful reporting warehouse strategy is to divide users among a) those who maintain data and deal with dynamic information, and b) those who simply report on static information. Each server can have indices developed that are optimal for their use (hence, why the "mirrored" servers technique is not a great solution for data warehousing) -- the reporting server leaning heavily towards several more indices, that index several different columns and combinations of columns, while the dynamic server utilizes just a few indices for joining tables or searching for individual records. In this setup, both users' needs for performance can be satisfied, and the reporting server off loads use on the scarce main server as well.

Additional performance can be gained on the reporting server by summarizing data and "flattening" the relational database structure. Instead of storing hundreds of thousands of detailed journal entries, for example, the warehouse could just store the sum total of all transactions during the month. And, instead of storing it in the structure of the original application, perhaps with dozens of tables as dictated by the relational model, one storing account descriptions, another storing company name descriptions, another storing state abbreviations and state names, etc., the warehouse may choose to denormalize, or consolidate all these tables into just a few. Since the data on a warehouse server is static, this denormalized form is not such a violation of Computer Science rules that it would be in a dynamic environment.

Performance gain #3: "Denormalize" and summarize data.

Summarization and denormalization, however, must be chosen carefully to make sure the data still meets users' analysis needs. Population statistics, for example, summarized per nation is useless to the user who needs regional population statistics. Such data techniques must be chosen carefully to meet the needs dictated by warehouse goals. Hence, this is why departmental warehousing systems make more sense than enterprise-wide systems: departmental uses are often well-defined, dictated by their processes or final output, whereas, with the corporation as a whole, some many disparate uses prohibit any useful data summarization. For example, with financial data, a tax department typically needs account balances summarized per month (or year), broken down by business unit and location, but a cost accounting group is only interested in analyzing finances by project, or perhaps by product line. If a single warehouse is chosen to serve all departments, opportunities for summarization and performance optimization must be sacrificed, or traded off among multiple department uses. The major goal for having a warehouse in the first place, that of performance, is defeated in this situation.

## Data Consolidation

The argument for data consolidation is simple: much of business analysis requires an examination of data from several sources. While some operational departments are happy with the data simply within their "reach" (a maintenance shop with its parts catalog and maintenance records, or a factory floor with its assembly instructions), many enterprise-scale departments, such as Marketing and Tax departments, are involved in broader analysis that requires data from a variety of sources. Data warehouses address this need by depositing data from all sources into one common RDB repository, available (from a technical point of view) for SQL joins and consolidated reports.

Without a consolidated source, this analysis is usually prohibitively expensive, as data is laboriously assembled (typically, a spreadsheet program like Excel is the tool employed) by hand. With a consolidated data warehouse, the same assembly is often accomplished, in the hands of a skillful developer or report writer, by a single SQL "query".

This benefit, however, is often not exploited properly within the corporation, because of a lack of expertise or experience with SQL, or other declarative query languages. In my observation, this tendency is exasperated by corporate IT departments staffed largely by developers who are more comfortable with procedural, instead of declarative, languages. That is, they'd more likely develop a program in a traditional language like VBScript or Java than write a report in SQL. Not that programming is inherently bad; the problem is that such development is typically burdened with significant maintenance costs in the future.

Best practice: use declarative report tools (SQL, XSL) instead of programming languages for data reporting

On the other hand, business users are largely unfamiliar with SQL, considering it more of a programmer's language than a business tool. Business users are much more comfortable with desktop applications like word processors and spreadsheets, both of which have weak, if any, data consolidation capabilities. Luckily, report writing tools like *Crystal Reports* and *Business Objects* have made the chore of learning reporting languages much easier.

One special case of data consolidation involves historical and trend analysis. In essence, this involves consolidating data from the past with data in the present. A warehousing server is often required from these uses because many application databases do not keep old data, either by design or to keep storage needs down and performance high. Without regular data purges, an application server can be bogged down by all the old records laying in its database.

A special case: historical data

Departments that must maintain records for regulatory retention purposes, such as tax departments with its record of past tax returns, require some sort of record warehouse, but the real need for a data warehouse is demanded by trend analysis. Such analysis involves comparing the past with the present. This can be used to see trends and strategize for the future, or to assess the impact of past strategies. Such analysis is often of strategic importance to the enterprise, as well as strategic functions such as Tax Planning. A typical consumer of such analysis is the high-level executive, and the whole field of Decision Support Systems (DSS) has evolved to meet their needs.

Historical analysis, trend reports, and DSS reports are usually easy to write with report queries, provided all the data is online and consolidated into a central data repository. To reduce space needs, data is often summarized before it is stored in the warehouse ( see above, under "Performance"), and a successful technical approach is to have the warehouse take periodic "snapshots" of data from the source application.

Best practice: take periodic snapshots of business data

## Conforming Data

Consolidating data into a single, central source does bring significant new reporting and analysis benefits to the corporation, and is usually the compelling reason for executing a data warehousing strategy. The image of the vast warehouse again comes to mind -- one large, easy-to-find place for all data. Data warehouse projects that proceed with the one, simple, goal in mind, however, often discover that they face significant costs even after they've succeeded in collecting all the data in one spot. This happens because they discover, after the fact, that much of the data doesn't "fit" together.

Joining together data in a single query or report requires there to be a matching "key" between the two data tables. For example, an employees' name, or id number, could be the key between a payroll record and a requisition approval application. The problem of fit is finding exact matches between the keys. Both sources must be in complete agreement about how common data is entered.

Problem: lack of fit between data

Names are notoriously hard to find matches -- slight misspellings, additional spaces between first and last names, the inclusion or exclusion of middle initial (or middle name). These problems are well known, and has been solved in most situations by the use of ids, or unique numbers, to identify unique identities. Then, the agreement problem involves which ids are used -- the payroll system, for example, may employ social security numbers, while the requisition application uses arbitrarily-assigned id numbers; no match between the two.

This lack of fit introduces significant costs into any warehousing project, but this cost is implicit within the enterprise anyway. Any business analyst who must consolidate data faces this issue, and often spends several weeks per year resolving it. A warehousing strategy, however, can reduce this cost by establishing a conscious "data conforming" strategy.

Data conforming involves the application of business rules and data mapping that provides consistency and interconnectedness between data records. Often, conforming rules are needed even if the data is from a single source: missing attributes in a record, for example, must be implied from other attribute values in the same record, or some other context.

Solution: develop "data conforming" business rules

This implied values and context are often known by the more experienced business users by heart ("oh, department 077 has always been located in Colorado -- everybody knows that"), but hard to capture in print, or more importantly, in computer algorithms, without a conscious effort. Not planned in advance, the conforming problems often bog down a warehousing project. Planned for, capturing these rules becomes the primary focus of the designers and analysts on the project, and can reduce project implementation time consistently.

One challenge of conforming data is to utilize a consistent design strategy to embed these business rules. One successful and widely understood strategy is to simply employ a mapping table in the database. The mapping is simply a record that maps keys between 2 databases or systems; for example, a table of employee ids and their matching social security numbers. The map table is stored and maintained in the data warehouse, regularly updated as new keys are added or changed

Best practice: mapping tables.

Another approach, less widely understood, is to incorporate all the conforming

Best practice: a business rules

business rules in an "application" layer, that sits between the data model and the reports. For example, a business rule that says all companies with a name ending in "Ltd." are automatically a "foreign" business type, can be encoded in an application layer report that categorized business by type. With SQL systems, this business layer can be implemented simply with SQL Views or stored procedures. Other environments may require more sophisticated techniques, but the challenge will be to maintain these rules in a cost-effective (i.e., easy-to-change) fashion. The first challenge is to consciously address data conforming problems; the next challenge is to capture conforming rules in an easy-to-maintain manner.

Best practice: a business rules "application layer"

## Some Notes on Project Costs

Most available discussion of data warehouse project costs have focused on enterprise-wide warehouses. The costs of such projects are usually huge, especially given the daunting challenge of a) trying to warehouse all potentially relevant enterprise data, and b) trying to meet the reporting needs of multiple corporate departments. For project at such a scale, the Meta Group has reported an average project cost of \$3 million. And, despite such costs, most of the projects end up either failing, or not living up to enterprise expectations -- the Gartner Group reports a failure rate of at least 60%. However, while statistics on departmental data warehouses are not available, we estimate that the typical cost of such a project is at least a magnitude less, and anecdotal experience suggests an average project cost more in the \$100k range.

Average enterprise-scale project cost: \$3 million.  
Source: Meta Group [find URL]

Enterprise data warehouse project failures: 60%.  
Source: Gartner Group [find URL]

Data warehouse "frameworks" are available to lower project costs and get a working system to completion quicker, but unfortunately there are no "shrink-wrapped" warehousing systems on the market today. Departmental needs are just too varied, potential data sources too numerous, and data conforming needs too loosely defined for a off-the-shelf package to work; raw development and customization are still required in order to succeed with any kind of data warehouse.