Relational OLAP: An Enterprise-Wide Data Delivery Architecture

In this paper, we detail key end-user requirements for decision support applications, the components of a robust data delivery architecture which address these end-user requirements, and the functional characteristics of each architectural component. We describe an architecture that requires little or no application maintenance, is extensible as new data sources are added, provides excellent query performance, supports complex filtering criteria, and is robust enough to support mission-critical data delivery applications. This architecture leverages investments in existing SQL-based relational databases and does not require additional investment in proprietary multidimensional databases.

Data Delivery Architecture Component Overview

Decision Support System/Executive Information System (DSS/EIS) architectures have quickly evolved from the first generation of mainframe-based, proprietary application development environments to the second generation of open, distributed architectures, integrating client-resident query tools with server-based relational databases.

These second generation products, while providing more rapid development and improved graphical user interfaces (GUIs), are too often simply data access tools which provide rudimentary data access and report presentation. These tools lack key features needed by both the decision support analyst and the Information Systems (IS) developer.

DSS analysts require:
- A multidimensional, conceptual view of data.
- The ability to create complex criteria sets which allow pinpoint access to required information.
- Support for hierarchical consolidation of data, and the ability to drill down into detail.
- Rapid query response.

IS developers require:
- Data warehouse scalability, from several gigabytes to tens of terabytes.
- The ability to leverage corporate and industry investments in relational technology.
- Minimum application redevelopment as the data model is modified.
- “Plug and play” support for normalized as well as a variety of performance-optimized data models.
- Application extensibility so that additional application features can be added easily.

Before addressing how these features should be provided via an appropriately designed and developed data delivery architecture, it is useful to provide a high-level overview of the various data delivery components. The architecture and its components are graphically depicted in Figure 1.
Operational Data
Corporations have a variety of on-line transaction processing (OLTP) systems such as financial, order entry, work scheduling, and point-of-sale systems which create operational data. This data is part of the corporate infrastructure and is detailed, non-redundant, updateable and reflects current period values. In contrast, data required by decision support analysts is often summarized, has a lengthy time horizon, is redundant to the support of varying data views, and is non-updateable. In order to provide data to decision support analysts, relevant operational data is extracted from OLTP systems, cleansed, encoded, and summarized. After being transformed into a format suitable for decision support, the data is uploaded into the data warehouse.

Data Warehouse
The data warehouse is the foundation of any DSS application, and its proper design is critical for effective DSS data delivery and performance. Decision support queries, due to their broad scope and analytical intensity, typically require data models to be optimized to improve query performance. In addition to impacting query performance, the data model affects data storage requirements and data loading performance.

Metadata
Metadata is to the data warehouse what the card catalog is to the traditional library. It serves to identify the contents and location of data in the warehouse. In our architecture, we expand the traditional data cataloging function of metadata to include definitions of data from a decision support perspective. Since metadata provides decision support-oriented pointers to warehouse data, metadata is a bridge between the data warehouse and the decision support application. In addition to providing a logical linkage between data and application, metadata, when combined with a decision support engine, enables 1) pinpoint access to information across the entire data warehouse and 2) development of maintenance-free applications which "automatically" update themselves to reflect data warehouse content changes.

DSS Engine
The DSS engine is the heart of the DSS architecture. It transforms data requests into SQL queries to be sent to the data warehouse and formats query results for presentation to the DSS analyst. To support these functions, the DSS engine includes a dynamic SQL query generator, a multidimensional data analysis engine, a mathematical equation processor, and a cross-tabulation engine.

Application Development Environment
The Application Development Environment (ADE) consists of a suite of object-oriented tools for the construction of the application used by DSS analysts. The DSS application allows, at a minimum, the specification of data to be retrieved and the viewing of query results.

The ADE contains the following tools:

• Criteria Set Manager, which enables the analyst to specify, save, and reuse filters to retrieve data from the data warehouse.

• Presentation View Manager, which permits the analyst to specify the format (e.g., tabular, graphical, geographic) in which query results will be presented.

• Data Rotation Manager, which allows the analyst to specify how query results should be presented on the x and y axes.

• Intelligent Agent Manager, which enables the analyst to easily create custom data analyses (Intelligent Agents) which automatically sift through the data warehouse using predefined criteria sets and present the resulting data set to the user. Intelligent Agents can be used to create analyst-definable executive-level presentations.

Additionally, the Application Development Environment should permit:

• Development of DSS applications that automate white-collar work. Tasks such as writing letters or sending electronic mail messages, performing sophisticated mathematical analyses, or issuing requests for more detailed information often directly follow decision support investigations and should be automated activities.
Development of closed-loop Decision Support Systems. Traditional DSS systems are “open-loop” in that they present data to DSS analysts and then require some action to be taken. Closed-loop systems interpret query results and automatically execute transactions such as changing product prices or shifting product inventory. At a minimum, these systems should allow the analyst to create operational transactions while interactively viewing DSS query results, as depicted in Figure 2.

Desktop Applications
Since decision support activity is typically not complete after data retrieval, strong data delivery architectures have tight integration with desktop personal productivity applications such as word processors, spreadsheets, electronic mail, and personal database managers. These linkages allow data to flow directly into presentation, analytical, and communication tools already used by the DSS analyst. Query results are directed to these applications using industry-standard data interfaces such as DDE, OLE, MAPI, and ODBC.

Definitions
Before discussing the data delivery architecture in detail, there are several terms which require definition.

Metrics
Metrics are variables or measures, typically numeric in nature, which are the focus of the decision support investigation. Examples of metrics include Sales, Revenues, Budgets, Inventory, and Market Share.

Dimensions
When considering a metric such as Sales, it is important to consider what data is available. Does Sales information exist for each of my products? Does Sales data exist for each of the countries, regions, and states I am interested in? Does Sales data exist for the last five years? Metrics such as Sales do not exist in isolation, but rather in the context of dimensions such as Product, Geography, and Time which define what type of Sales data is available. It is natural to think of data multidimensionally as shown in Figure 3, where Sales is the metric and is qualified by the dimensions of Geography, Product, and Time.
Attributes
Attributes are specific metric qualifiers and are defined by columns in the data warehouse. Attributes belong to dimensions. Dimensions are conceptual metric qualifiers and are not physically represented in the data warehouse. An attribute is always associated with a dimension. If we consider the above Sales example in more detail, we realize that the dimensions of Product, Geography, and Time only generally describe the available Sales data. To perform an analysis, I may need Sales data by Region, State, and City, each of which is an attribute of the Geography dimension. Additionally, I may require Sales data by Month, Week, and Day, each of which is an attribute of the Time dimension.

One or multiple attributes may be associated with a single dimension. For example, in Figure 4, the Geography dimension contains three attributes: Region, State, and City. While the Geography dimension would not physically be represented in the data model, each of the Geography attributes would be represented by columns in the database.

Attribute Elements
Attribute elements are the actual values of an attribute. For example, the Region attribute could have Midwest, Mid-Atlantic, and Southeast as elements. Correspondingly, the State attribute could include Virginia, Maryland, North Carolina, and Georgia as attribute elements.

Attribute Hierarchy
Attributes of a dimension are often classified along a well-defined hierarchy. For example, within the dimension Geography, a Region can be broken down into States, which in turn can be broken down into their respective Cities. Region, State, and City define a hierarchy within the dimension Geography.

Optimizing the Data Warehouse for Decision Support
The data warehouse is the foundation of any data delivery system, and the warehouse data model must address the conflicting goals of system flexibility and data delivery performance. Flexibility is required so that additional attributes, dimensions, and metrics can be added to the warehouse, complementing and leveraging existing data. Quick query response is required in all effective interactive Decision Support Systems, since it not only impacts end-user satisfaction but directly determines the number of analyses the DSS analyst can perform given limited time.
In considering the data warehouse data model, it is useful to first review the types of tables found in a data warehouse. The three types of tables are:

- Primary Data Tables
- Descriptor Tables
- Characteristics Tables

Primary Data Tables contain both metrics and attributes, and contain the data that end-users are seeking. In Figure 5, the Primary Data Table contains the attributes Product_ID, Store_ID, and the metric Sales. In large data warehouses, the full-text attribute description is not stored in the Primary Data Table, but rather in a Descriptor Table, while numeric element ID codes are stored in the Primary Data Table. Numeric ID codes can be indexed faster, yield smaller indices, and provide faster WHERE clause matching than their text counterparts.

Descriptor Tables often contain only two columns, the attribute ID code and the common-English description of the attribute. There is a one-to-one relationship between the ID and the description. These tables are used to replace the ID codes used in queries with common business terms familiar to the user. In Figure 5, there are two Descriptor Tables which map Product_ID and Store_ID codes to their respective user-understandable business terms. In smaller warehouses, where load performance and storage concerns are lessened, text descriptors may appear in the Primary Data Tables to increase data comprehensibility.

Characteristics Tables contain additional information about an attribute and can be used to segment data in an ad-hoc manner. Each column in a Characteristics Table represents an additional attribute that can be used as a filtering criterion in queries. In Figure 5, there is a single Characteristics Table which contains additional attributes related to the Store attribute. Using this Characteristics Table, Sales could be segmented by store square footage, or any other attribute in the Characteristics Table.

The ability to define and join with Characteristics Tables containing many attributes is a key advantage of RDBMS technology over multidimensional data storage architectures, since multidimensional architectures do not support multi-table joins.

Performance Optimization

DSS query performance, while a function of the performance of every component in the data delivery architecture, is strongly correlated to the physical data model. Intelligent data modeling, through the use of techniques such as denormalization, consolidation, and partitioning, can provide orders of magnitude performance gains compared to the use of normalized data structures.

Denormalization

Denormalization improves performance by either:

1) Reducing the number of joins required during query execution, or
2) Reducing the number of rows to be retrieved from the Primary Data Table.

There are three principal denormalization techniques:

1) Include Descriptors in Primary Data Tables:

Rather than placing ID descriptors in a separate Descriptor Table, descriptors may be placed in the Primary Data Table in order to eliminate a join. In Figure 6A on page 6, the Store Descriptor is contained in the Primary Data Table along with the Store_ID.

2) Include Child Records in Parent Record:

By adding columns to the Primary Data Table, child records can be added to the parent record, thereby allowing all child records to be retrieved with every parent record retrieval. This technique improves performance by eliminating a join and
removing child record criteria from the WHERE clause. However, query flexibility is impaired since child record attributes are added to columns of the parent record and can no longer be used as filtering criteria. This technique works best when there are a fixed and relatively small number of child records. In Figure 6B, Q1, Q2, Q3, and Q4 Sales are represented as columns in the Primary Data Table.

3) Include Most Recent Child Record in Parent Record: In some systems, the most recent child record is used much more frequently than other child records. In these cases, adding the most recent child record to the parent record can improve performance. In Figure 6C, the most recent child data, last quarter’s sales, is added to the parent record.

### Summarization

One of the most powerful performance optimization techniques is the use of multiple, summarized Primary Data Tables. Data is typically loaded into the data warehouse from operational systems at an atomic level of detail. Most decision support queries however, require aggregate-level, consolidated information and not atomic detail. If processed against the atomic level, these queries require the retrieval of many records to yield aggregate information, and require that the data be mathematically summarized at query runtime, further impeding query performance.

To improve query performance, data can be pre-summarized and stored along frequently accessed consolidation hierarchies. It is impossible to pre-summarize atomic detail if the consolidation hierarchy is not known. Fortunately, these consolidation hierarchies are often identical to the attribute hierarchies within a dimension. In fact, for best query performance, Primary Data Tables should exist at each attribute intersection across all attribute hierarchies.

At first glance, the addition of Summary Tables may appear to dramatically increase data storage requirements. However, since Summary Tables get progressively smaller as one approaches the most aggregate level of the attribute hierarchy, only a 20-100% increase in data storage is typically required to physically store all attribute hierarchy-defined consolidation paths. This modest increase in disk storage typically yields a two-to-ten-fold increase in query performance.

---

**Figure 6A: Denormalization Techniques**

<table>
<thead>
<tr>
<th>Store_ID</th>
<th>Store_Name</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Fairfax</td>
<td>1,000</td>
</tr>
<tr>
<td>102</td>
<td>Vienna</td>
<td>1,500</td>
</tr>
<tr>
<td>103</td>
<td>Richmond</td>
<td>2,000</td>
</tr>
<tr>
<td>104</td>
<td>Landover</td>
<td>2,500</td>
</tr>
</tbody>
</table>

Primary Data Table Containing both Store_ID and Store_Descriptor

**Figure 6B: Denormalization Techniques**

<table>
<thead>
<tr>
<th>Store_ID</th>
<th>Q1 Sales</th>
<th>Q2 Sales</th>
<th>Q3 Sales</th>
<th>Q4 Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>1,000</td>
<td>3,000</td>
<td>5,000</td>
<td>0</td>
</tr>
<tr>
<td>102</td>
<td>2,000</td>
<td>4,000</td>
<td>10,000</td>
<td>0</td>
</tr>
<tr>
<td>103</td>
<td>3,000</td>
<td>5,000</td>
<td>15,000</td>
<td>0</td>
</tr>
<tr>
<td>104</td>
<td>4,000</td>
<td>6,000</td>
<td>20,000</td>
<td>0</td>
</tr>
</tbody>
</table>

Quarterly Sales Child Records Added to the Store_ID Parent Record

**Figure 6C: Denormalization Techniques**

<table>
<thead>
<tr>
<th>Store_ID</th>
<th>Q1 Sales</th>
<th>Q2 Sales</th>
<th>Q3 Sales</th>
<th>Last Qtr Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>1,000</td>
<td>3,000</td>
<td>5,000</td>
<td>5,000</td>
</tr>
<tr>
<td>102</td>
<td>2,000</td>
<td>4,000</td>
<td>10,000</td>
<td>10,000</td>
</tr>
<tr>
<td>103</td>
<td>3,000</td>
<td>5,000</td>
<td>15,000</td>
<td>15,000</td>
</tr>
<tr>
<td>104</td>
<td>4,000</td>
<td>6,000</td>
<td>20,000</td>
<td>20,000</td>
</tr>
</tbody>
</table>

Most Recent Child Record in Parent Record
increase in query performance over non-consolidated data warehouse designs.

The example Figure 7 shows two dimensions, Product and Geography, each of which has multiple hierarchical attributes. Data is extracted from the source systems at the lowest level of both hierarchies, which are the item and zip code levels. A Summary Table is then added for each attribute combination of both attribute hierarchies.

Partitioning
A final technique used to improve query performance is data partitioning. In a partitioned data model, multiple tables are created to store atomic level data. Figure 8 contrasts an atomic non-partitioned table with an atomic set of tables partitioned by month.

Partitioning of large atomic data tables into multiple smaller tables provides the following advantages:

1) Query response time is improved.
2) Incremental data backup and recovery is accelerated.
3) Time required to load into indexed tables is decreased.

Disadvantages of partitioning include:

1) Joins and unions are required to retrieve related data from multiple tables.
2) More intelligent query generation is required to determine which tables contain user-requested data.
3) Additional metadata is required to describe the partitioned warehouse.

Naming Conventions
All column names referring to the same attribute or metric should be consistent throughout the warehouse. While perhaps obvious, name consistency is extremely important for understanding data. Consistency is absolutely required when using query generators which dynamically determine join columns based on column name.

Data Warehousing Summary
In enterprise-wide data warehouses, decision support query performance often needs to be improved to satisfy the requirements of interactive users. Through the use of techniques such as denormalization, summarization, and partitioning, the data warehouse data model can be optimized to deliver dramatically improved performance over highly normalized models. While these techniques are not new, it traditionally has been difficult to create a data delivery architecture which seamlessly supports performance-optimized models since metadata requirements have been ill-defined and DSS engines have had limited query generation capabilities.
**Metadata**

Before warehouse data can be accessed efficiently, it is necessary to understand what data is available in the warehouse, and where that data is located. Metadata, or data about data, provides DSS analysts with a catalog of data in the data warehouse and the pointers to this data. In addition to helping DSS analysts locate the data they desire, the metadata may contain:

- Data extraction/ transformation history.
- Column aliases.
- Data warehouse table sizes.
- Data summarization/modeling algorithms.
- Data usage statistics.

While much has been written about metadata, there has been little discussion of the impact of metadata on data delivery architectures. In a well-designed architecture, metadata maps the entities with which DSS analysts are familiar (dimensions, attributes, and metrics) to tables and columns within the data warehouse. Specifically, this metadata includes:

- Dimension, attribute, metric definitions.
- Attribute hierarchies – this is a mapping of parent/child relationships amongst attributes.
- Attribute to dimension mapping – indicates the dimension to which each attribute belongs.
- Metric to attribute mapping – indicates which attributes logically qualify each metric.
- Physical mapping of attributes, metrics – this is a mapping of attributes and metrics to the tables and columns in the data warehouse.
- Performance metrics – these metrics detail the performance characteristics of tables in the warehouse. Performance metadata includes the number of rows in each table and the indexes existing on each table. While most RDBMSes have query optimizers, this information is required for optimized query construction, rather than the tactics of how to optimize the performance of a given SQL query.
- Distinct elements per attribute – often, DSS Analysts need knowledge of available elements within an attribute in order to formulate their queries. By presenting analysts with available elements, they do not need to guess whether elements exist in the warehouse, the spelling of those elements, etc. Metadata provides a list of distinct elements for each attribute, or a pointer to such a list.

In well-designed data delivery architectures, the application is coupled to the warehouse via the metadata, allowing changes to the data warehouse to be immediately reflected in the end-user data access application. For example, if a corporation restructures to eliminate a layer of management, as soon as the data corresponding to the new organizational hierarchy is added to the warehouse, the DSS application should “reconfigure” itself using the metadata to reflect the new hierarchy. As shown in Figure 9, almost all data delivery applications built using GUI forms painters, report writers, or EIS tools have little or no coupling between the metadata and the DSS application. As a result, applications built using these tools become obsolete and must be modified by the designer when new attributes or metrics are added to the warehouse. By using metadata to describe warehouse content and structure, and by integrating this metadata with a powerful DSS engine, application construction and maintenance can be simplified dramatically.

---

**Figure 9: Outmoded DSS Configuration**

In existing data access architectures, there is little or no coupling between metadata and the DSS application.
Application Development Environment

DSS analysts interact with the DSS application, which is constructed using the ADE. The ADE provides tools which allow:

- Creation, modification, and deletion of:
  - criteria sets
  - data presentations
  - data rotations
  - intelligent agents
- Combination of criteria sets, data presentations, and data rotations to create custom analyses.
- Seamless integration of the DSS application with other desktop productivity applications.
- Automatic update of the DSS application when the data warehouse is modified.

Applications constructed using the ADE shield the DSS analyst from the other components of the DSS architecture. This allows the analyst to use the application without learning the intricacies of the data warehouse model, SQL, or the DSS engine.

Criteria Set Manager

Criteria sets are user-defined filters which specify the subset of data to be retrieved from the data warehouse. Criteria sets can consist of attribute element values (e.g., City = 'Boston', Product = 'trucks') and/or metric qualifiers (e.g., Sales > 100, Earnings < $1000). Since criteria sets determine the data to be returned from the warehouse, they are used to construct the WHERE clause in SQL SELECT statements. One of the most powerful features of the ADE is the Criteria Set Manager, which allows the DSS analyst to create, modify, and delete criteria sets.

In addition to allowing the creation of sets, the Criteria Set Manager supports:

Criteria Set Libraries – Saved criteria sets are invaluable for performing the same analyses repeatedly, over time. Once a set is saved, it can be recalled at any future point and used to subset the data. Saved sets may be either private or public.

Private sets are often stored locally and are typically only used by the DSS analyst who created the set.

Public sets are stored on a server and are accessible by multiple analysts. These shared, public sets can help enforce a common analysis framework across an enterprise, minimizing inconsistent interpretation of data due to differences in analytical scope. A common shared set repository also helps eliminate the proliferation of redundant, identical sets with differing names. Also, public sets can be pre-compiled and saved on the server in validated SQL syntax, minimizing both the network traffic and the delay associated with query construction.

Criteria Set Operators – Criteria sets consist of a number of attribute elements or metric qualifiers that, when combined, constitute a criteria set. There are many operators available for combining multiple attribute elements or metric qualifiers, including AND, OR, NOT AND, NOT OR, UNION, INTERSECTION, etc. In fact, many operators used to combine or modify “mathematical” sets are useful in the development of decision support criteria. An example of the use of set mathematics to specify decision support criteria is given in Figure 10. In this figure, the criteria “profitable stores in malls, excluding new stores” is represented via a Venn diagram, as well as the equation \( \{\text{Profitable Stores}\cap\text{Mall Stores}\}\setminus\{\text{New Stores}\}\). There are additional rules that must be applied to set operators in the decision support environment. Operators which have a single meaning to an analyst must be interpreted in the context of dimensions and attributes. For example, assume an analyst desires to analyze sales of cars and trucks in Boston, San Francisco and Detroit. The following elements comprise the set:

City = 'Boston', City = 'San Francisco', City = 'Detroit', 
Product = 'Cars', Product = 'Trucks'

Figure 10: Using Set Mathematics to Specify Data of Interest

\[ \text{Profitable Stores} \cap \text{Mall Stores} \setminus \{\text{New Stores}\} \]

"Show me profitable stores in mall, excluding new stores."
While this set appears simple enough, neither of the following WHERE clause predicates will return the information requested:

**Predicate A**
- `City = 'Boston' OR City = 'San Francisco' OR City = 'Detroit'` AND
- `Product = 'Cars' OR Product = 'Trucks'`

**Predicate B**
- `City = 'Boston' AND City = 'San Francisco' AND City = 'Detroit'` AND
- `Product = 'Cars'` OR `Product = 'Trucks'`

Predicate A will return car and truck sales in all cities and sales of all products in Boston, San Francisco and Detroit. Predicate B will return no data.

To operate properly, intradimensional elements such as Boston, San Francisco and Detroit must be ORed and interdimensional elements must be ANDed. The working predicate would appear as:
```
(City = 'Boston' OR City = 'San Francisco' OR City = 'Detroit') AND (Product = 'Cars' OR Product = 'Trucks').
```

The ADE should have an easy-to-use, logical, graphical user interface that permits the DSS analyst to create complex criteria sets that leverage the principles of set mathematics, but do not require detailed understanding of mathematical operators or SQL semantics.

**Presentation View Manager**
The ADE contains a Presentation View Manager which allows creation, modification, and deletion of data presentations. Data presentations may be of the following types:
- Graphical (Bar, Line, Pie, Bubble, Scatter, etc.).
- Grid (commonly used for cross-tabulated results).
- Columnar Report.
- Custom (Stoplight, other).

The Presentation View Manager supports the use of predefined data presentations, as well as the development and use of user-defined data presentations.

**Rotation Manager**
The Rotation Manager allows the user to specify the x and y position of metrics and/or attributes within a presentation. A sample rotation is given in Figure 11. Note that multiple attributes and metrics can be specified on each axis; in Figure 11, the x axis displays both the attribute Product, and the metrics Sales and Volume. As with the other managers in the ADE, the Rotation Manager enables analysts to access a predefined set of data rotations, as well as create customized rotations.

**Performing Analyses**
Criteria Sets, Data Presentations, and Data Rotations are the building blocks for performing analyses in the decision support environment. An analysis is created when a set, a presentation, and a rotation are combined. Multiple analyses can be developed from a single set simply by varying the rotation and the presentation. Since sets, presentations, and rotations are independent, they can be combined in any manner to yield an analysis.
The object-oriented nature of sets, presentations, and rotations enable “data surfing.” Data surfing can be defined as the ability to interactively:

- Apply multiple sets to a single presentation and rotation combination.
- Apply multiple presentations to a single set and rotation combination.
- Apply multiple rotations to a single set and presentation combination.

Data surfing allows the analyst to navigate through the entire data warehouse via the interactive creation of analyses. As with other objects, analyses can be saved and recalled for future investigation.

**Intelligent Agents**

Once users have created the Criteria Sets, Presentations, and Data Rotations which make up an analysis, they need the capability to schedule and run analyses in the background in addition to interactively running the analyses. Analyses which run in the background are known as “Intelligent Agents.” The ADE enables the user to create two types of Intelligent Agents: Exception Reporting Agents and Periodic Agents.

Exception Reporting Agents run in the background, and notify the analyst only if some predefined condition is encountered. For example, the user can create an Exception Agent which runs a sales analysis for every city in a particular region every time the user enters the DSS application. If the sales analysis uncovers a city which is an order of magnitude (or some reasonable amount) different from the median value for the other cities, the user will be notified.

The Periodic Agent runs at defined time intervals and presents its findings to the user automatically. For example, a Periodic Agent can be defined to run a sales report at the end of each month, and present the results to the analyst automatically. Analysts have the ability to create, modify, and share agents, as well as the ability to enable and disable them.

**Desktop Application Integration**

With the advent of standards such as Microsoft’s Object Linking and Embedding (OLE), technology for applications which can share data and functionality with other desktop applications is becoming more common and available. Users increasingly expect the ability to include the results or conclusions of their decision support analyses into word-processing or spreadsheet reports, transfer their DSS data into other databases for additional processing, and share these analyses with other users through electronic mail applications. The ADE should support Microsoft Windows standards such as ODBC, OLE, DDE, DLLs, and MAPI.

**Application Maintainability**

Dimensions, attributes, attribute hierarchies, and metrics which an organization collects in the data warehouse continually change over time. In traditional DSS/EIS architectures, the user interface is static and does not update itself to reflect these inevitable changes. While many existing DSS/EIS applications are data driven, few are dimension and metric driven. These applications quickly become obsolete unless they are physically updated when the warehouse is modified. The ADE solves this common maintenance bottleneck by providing dynamic links between the Criteria Set Manager and Data Rotation Manager to the dimension, attributes, and metrics in the metadata. Whenever the metadata and data warehouse are updated, all DSS applications tied to the data warehouse are also automatically updated. Analysts benefit from this coupling since information becomes available as soon as it enters the data warehouse. Information Technology (IT) shops benefit through reduced application maintenance.

**DSS Engine**

The DSS engine is the nexus between the warehouse, the metadata, and the client-based DSS application. The engine synthesizes data requests, metadata, and its own query-building algorithms to create SQL queries to send to the data warehouse. Upon query completion, the DSS engine performs necessary data manipulations, such as cross-tabulation, before sending the data to the DSS application.

**Automatic Query Generator**

Of all of the tasks which the DSS engine performs, translating the analyst request into a SQL query is the most challenging. In order to create SQL queries, the engine requires inputs from the end-user application and the metadata, and then builds the SQL query with algorithms within the DSS engine.

**End-user Query Specifications**

The DSS engine requires a Criteria Set from the user, which specifies the attributes and metric
qualifiers which the DSS analyst wishes to apply to the data in the data warehouse. The DSS engine also uses a Data Rotation, to specify how columnar data (attributes, metrics, or both) from the data warehouse should be oriented on the x and y axes for transmittal to the DSS application.

**SQL Generation Algorithm**

Once the DSS engine has received the end-user information request, it applies the Data Rotation and Criteria Sets to the metadata to build the SELECT and WHERE clauses of the SQL query. The DSS engine first determines how the Criteria Sets should be resolved to form the foundation of the SQL WHERE clause. The DSS engine then determines the SQL SELECT clause using the Data Rotation.

To build the FROM clause, the engine determines which tables must be accessed to retrieve the requested data by referencing the metadata. The engine must also determine which tables in the warehouse provide optimum query performance and create inter-table joins, as required.

The above SQL generation must occur dynamically at runtime, whenever the analyst creates an analysis by combining a criteria set, a data rotation, and a presentation. To summarize, the DSS engine must synthesize metadata and specifications from the DSS application to produce SQL queries, as shown in the upper half of Figure 12.

**Mathematical Manipulation**

The DSS engine provides the capability to perform complex mathematical manipulations on the data returned from the SQL query. These include the ability to apply various arithmetic expressions to the data, and the ability to perform standard statistical analyses on the data before presenting the data to the end-user.

**Cross-Tabulation**

Once the query results are returned and manipulated by the DSS engine, the data may be cross-tabulated before being returned to the user in the rotation which the DSS analyst has specified. This module of the DSS engine, the cross-tabulation engine, converts the data from columnar format to cross-tabular format. The cross-tabulation engine can perform various manipulations on the data, including summarizing across like elements in the columnar data, performing averages on the data, and providing maximum and minimum values of the data. Because the Application Development Environment has the ability to display multiple parameters on any given axis, the cross-tabulation engine must be able to manipulate hierarchies of data elements, rather than single elements.
Multidimensional Databases

Recently, there has been much discussion of multidimensional database engine-based decision support architectures. Multidimensional technology can be attractive since it provides the DSS analyst with a multidimensional conceptual view of data. However, it is important to separate the requirement to view data multidimensionally from the requirement to physically store data multidimensionally. Data can be stored relationally (or in any other format) and still be viewed multidimensionally.

In short, multidimensional database technology is not necessary for decision support and, in fact, faces significant hurdles before its adoption as a viable alternative to relational database technology. The Gartner Group has identified eight concerns regarding multidimensional databases that must be considered when evaluating this technology in lieu of relational technology:

1) Database Management Tools – As with other database engines, multidimensional databases need to provide adequate management tools. These include the ability to:
   • Ensure database integrity through backup and restore capabilities.
   • Tune databases for optimum performance.
   • Provide user security at multiple levels.
   • Limit resource usage based on user privileges.

2) Support for Drill-Down to Atomic Level Detail – Multidimensional databases do not allow retrieval of row-level detail data transparently when the user wishes to drill down to atomic detail. Since transaction detail is never stored in multidimensional databases, it is awkward to access row-level detail when using multidimensional technology since an entirely different API is required. Atomic level transactions are often stored in relational formats, enabling relational-based Decision Support Systems to provide a seamless row-level drill-down capability.

3) Incremental Database Refresh – A multidimensional database should allow updates to any subset of the database without restricting access to unaffected parts, or rebuilding the database from scratch. With multidimensional databases, often the entire “data cube” must be reloaded when data is modified and all data is inaccessible during the data loading process. Relational databases provide incremental data update and insert capabilities and do not restrict use of existing data as new data is loaded.

4) Multiple Arrays Support – Multidimensional databases do not support the creation of multiple, related multidimensional arrays within a single database structure. Relational architectures typically permit defining up to 256 tables in a single database.

5) Database Joins – Multidimensional databases do not support the logical joining of multiple multidimensional arrays. The inability to join multidimensional databases with each other, or with relational databases, limits query flexibility by eliminating the possibility of using Characteristics Tables to dynamically segment data. This limitation is depicted in Figure 13 on page 16.

6) Subset Selection – As with relational databases, a multidimensional database should provide for data subsetting, limiting the amount of data for analysis.

7) Local Data Support – In addition to client/server support, a multidimensional database should seamlessly support local storage and data manipulation, enabling users to select subsets of data for local, unattached processing. Since there are numerous local RDBMSes which support the SQL standard for data access, data subsets can easily be downloaded into a variety of engines for local, unattached processing.

8) SQL Interface – Multidimensional databases do not support industry-standard SQL interfaces for data access. Rather, they each have their own own, proprietary data access APIs. As long as the SQL standard is not supported, proprietary, engine-specific data access and analysis tools must be utilized.

The previously mentioned multidimensional technology limitations and the requirement to incorporate a proprietary database technology into the corporate data processing environment should be weighed by those considering multidimensional databases.
Historically, buyers of multidimensional technology have based their purchase decisions on the need for improved DSS query performance and on the requirement for a multidimensional conceptual data view. With recent, dramatic improvements in indexing techniques, superior support for handling of sparse data, and the advent of symmetric multiprocessor hardware support, relational technologies have effectively closed the performance gap that once existed. Coupled with a data delivery architecture that provides data consolidations and a multidimensional conceptual view, industry standard relational architectures provide a full complement of multidimensional database features without resorting to use of proprietary database engines. Performance scalability, the ability to support a multitude of data access paths, the ability to leverage standard data access tools, and support for multidimensional data views make relational databases the preferred choice for enterprise-wide DSS implementations.

**Conclusion**

Many corporations have completed and operationalized their initial executive information and decision support applications with moderate success. Users have recognized the value of Decision Support Systems and are driving the demand for applications that span additional business functions and are deeper in scope than initial pilot applications. Information system developers have used a variety of tools to create these Decision Support Systems, including 3rd generation languages, 4GL application development tools, and Executive Information System tools. These tools provide valuable application development functionality, but used outside the context of an integrated, decision support focused architectural framework, often lead to the development of data structure-specific, brittle applications that cut corners on desired functionality.
Lacking in many of these initial pilots is an architecture that integrates the relational data warehouse, metadata, and GUI development tools to provide both advanced DSS features and ease of application construction and maintenance requirements. Without an architectural blueprint, it is nearly impossible to define and develop fundamental DSS objects which, when combined, provide a full complement of DSS capabilities such as multidimensional data views, drill everywhere, and “data surfing” for all data in the data warehouse.

IS managers looking to provide DSS applications should strive to revolutionize the delivery of decision support solutions in the same way spreadsheets revolutionized business analysis. Until the release of PC-based spreadsheets such as Visicalc and Lotus 1-2-3, which contained fundamental analytical objects, IS developers were responsible for hand-programming each spreadsheet required by the business user. Today, it would unthinkable for IS developers to program custom business analyses since these analyses are better developed by business analysts armed with spreadsheet tools. Similarly, DSS applications are better developed by business analysts armed with the right data access and analysis tools. However, IS developers too often must construct these business-oriented applications since corporations have not defined the architecture and the fundamental objects required for decision support.

The ideal architecture consists of the components described in this paper, each of which has a distinct purpose in the delivery of data. In order to effectively utilize existing data models and allow reengineering of inefficient models, this architecture should provide seamless access to existing, normalized data models and a variety of performance-optimized data models including consolidated, denormalized, and partitioned structures.

The architecture should leverage metadata to facilitate application construction and minimize application redevelopment as additional data structures are added to the warehouse. It should also contain an engine which interprets analyst data requests and generates SQL queries to retrieve data from the warehouse. The engine should be capable of mathematically processing and cross-tabulating this data to provide the analyst with the exact data view required.

The application is the most visible part of the architecture, and should be constructed using an environment that supports the definition and use of complex criteria sets, the specification of data presentations and rotations, and the seamless integration of desktop personal productivity applications. The application should also permit creation of Intelligent Agents that automate white collar work by sifting through and analyzing data and taking actions such as sending electronic mail, writing letters, or updating transaction processing systems.

Taken as a whole, the data delivery architecture should leverage the substantial, growing investments in SQL-compliant relational databases and off-the-shelf GUI development tools and incorporate a DSS-focused engine to provide analysts with a full complement of decision support features. DSS Agent, a next-generation, Windows-based decision support development environment available from MicroStrategy, Incorporated, complies with the architectural framework described in this white paper and provides, through fundamental DSS objects, the features required by Decision Support System users.