PROOF Overview

A Reference Design for BI Applications

Today's customers have a wide variety of choices for business intelligence (BI) software platforms that vary in features and functions. As software platforms, they provide a set of software reference architectures for the topology of servers and services. These reference architectures are useful standards for configuring servers and services, but are not aimed at application design. The task of enforcing BI best practices for application design is largely left up to each individual developer.

Building a BI application for performance management (PM) is a complex undertaking. PM applications vary by function and industry and can take the form of dashboards, scorecards, analytical investigations, and reporting. The reference architectures provided by software platforms do little to address the need for a reference design for building a performance reporting application. Developers are left with deciding how best to implement PM applications. Typically, much of the logic in a PM system is scattered throughout the system in reports, scorecards, and dashboards. This approach leads to a duplication of effort, since identical metrics must be reproduced across the reporting spectrum.

To meet the need for a prescriptive approach for creating solutions that are common across performance reporting applications, Symmetry created a reference design methodology called the Performance Reporting Framework (PROOF™). PROOF provides a set of standard design patterns for the Microsoft BI platform based on BI best practices. The PROOF design patterns specify generalized solutions to common situations that arise in building PM applications. PROOF takes a structural approach to defining metrics rather than an ad hoc approach. By following PROOF's structured design principles, PM applications can be developed using a consistent, repeatable, and centrally maintainable approach.

PROOF Design Principles

PROOF is driven by three design principles: intelligence, integration, and integrity.

- ***Intelligence*** – While every PM application has unique aspects, there are common dimensions and metrics across PM applications. These standard dimensions and metrics can be codified to create an intelligent application. The primary examples of standard dimensions are Time and Scenario. Standard metrics are those metrics that can be leveraged across all data in any PM application, e.g., year-to-date, variance, or percent-of-total. Standard metrics are created using smart measures. Smart measures are measures with properties that control the calculation of metrics. For example, associating the aggregation type with a measure is useful in calculating year-to-date. A sales measure is always summed, but headcount is never summed and would be expressed as an average or ending balance.

By leveraging standard metrics across the data, there is no need for duplicate logic when new data is introduced or a new PM application is added. This minimizes development time and ensures that metrics are defined consistently across not only PM applications, but also the organization. Furthermore, because standard metrics operate within the intelligent context enforced by standard dimensions and smart measures, even when new data is introduced, numbers and indicators are displayed correctly.

- ***Integration*** – The key to creating an integrated PM application is adhering to the concept of "form follows function" (which specifies that design should reflect function). In order to facilitate a unified view of standard metrics, PROOF specifies that all of the business logic for standard metrics should be in the server and not in the client presentation layer. Since metrics are calculation-intensive, storing metrics in a server environment optimized for calculations is a logical choice whenever practical. This approach allows for the greatest leverage and flexibility in constructing and maintaining a PM application.

  Even more importantly, with a centralized, server-based approach, a single version of the truth becomes enforceable for metrics as well as data. Each standard metric has a single definition centrally accessible by every client, whether the client is a spreadsheet, report, scorecard, or dashboard. The client layer becomes simply a matter of presentation, focusing on the functions of layout, format, and navigation. Definitions for standard metrics are created only once and the same definitions are used by everyone. Consequently, each display, whether a report or dashboard, is not a separate development effort with its own embedded business logic. This allows for the creation of new displays with a minimum of time and effort.

- **Integrity** – A structured approach is required to ensure that corporate metrics are systematically developed and maintained. The PROOF taxonomy classifies metrics that operate similarly into categories and further classifies how calculations and business models interact with those metrics. By using a set of consistent approaches, the calculation engine will better know how to manage the interaction of data correctly. By codifying a generalized, prescriptive solution to common recurring situations, regardless of the dimensions or data in a given application, the same approach is taken and the correct answer is returned. For example, only one approach is taken to specify year-to-date, variance, or percent-of-total. By standardizing on a single approach, the structure of the data becomes inconsequential. Furthermore, following a single approach also means that the percent-of-total of the year-to-date variance will also be correct.

PROOF Design Patterns

The PROOF methodology ensures that the majority of BI standards and best practices are built into the application, providing a universal platform that is available across all reporting environments. In order to create a PM application that abides by the design principles of integration, intelligence, and integrity, the PROOF methodology encompasses the following set of design patterns that enforce BI best practices.

Standard Schema

In order to enforce naming conventions and a standard structure for dimension and fact tables, PROOF prescribes a standard template schema. Naming conventions benefit developers by enabling them to easily differentiate a dimension table from a fact table. Standard structures ensure that the structure of tables is consistent within and across all databases.

Standard Dimensions

Standard Dimensions are dimensions about which much information is known even before an application has a detailed specification. This is because much of their structure and usage is prevalent across applications. An interesting characteristic of standard dimensions is that they drive the processing and the calculation of metrics. Because these dimensions are so common, PROOF specifies dimension design templates for both the schemas and metrics. The standard dimensions that PROOF specifies fall into two classes:

- Basic dimensions –Time and Scenario
- Conversion dimensions - Currency and Unit of Measure

Time and Scenario are well understood and are common components across almost every PM application. Because they are so ubiquitous, it is critical that they be included with a standard structure and set of functionality built-in. The Scenario dimension is a relatively simple dimension and drives the set of target metrics around variance.

The Time dimension is a more complex dimension, but many important aspects of it are known. It is a sequential dimension (the position of members is pre-defined) and cumulative sums (period-to-date) are important, unlike most of the other dimensions. While some BI software addresses standard Time dimension functions, there are many alternative hierarchies for time associated with industry and accounting practices. The PROOF Time dimension offers the ability to create any number of alternative or fiscal hierarchies.

Data by time often has repeating patterns (seasonality). To analyze seasonality, a different (non-hierarchical) view is needed to create a period-of-period view (day of week or month of year). The metrics driven from the Time dimension are the trend metrics and include metrics such as change and growth. The PROOF design template for time includes these kinds of aspects and ensures that they work together correctly.

The two conversion dimensions, Currency and Unit of Measure, are not as pervasive as Time and Scenario but they are relatively common. Both dimensions perform basically the same function, conversion, but on measures of different value types: Currency conversion on financial measures and unit of measure conversion on volume measures. Currency conversion rates vary by time and each measure might have a different conversion basis (i.e., ending or average). Unit of measure conversions have different classes (i.e., weight, volume, or duration). For those applications that need this functionality, PROOF includes design templates.

Item – An Enhanced Measure Dimension

A commonly accepted best practice in data warehousing, data marts, and reporting systems in general is that queries, by users or reporting tools, should not query base physical tables but views built on the physical tables. This creates an abstraction layer whereby the physical structure of the database can be changed with minimal or no impact on reporting and usage. This is not a common practice in using multidimensional databases. One reason for this is that the multidimensional databases do not have a native functionality equivalent to views in a relational database. Another reason is that historically multidimensional databases were not considered part of an enterprise strategy and an integral part of the database platform, but more as an afterthought to allow slicing and dicing of data. This has changed in the last 10 years. Multidimensional databases are full development environments for production applications and are critical to the delivery of information and analytics to the enterprise.

To separate the physical structure of measures in a multidimensional database from the logical structure that users and reports query, PROOF specifies the creation of an Item dimension, an enhanced Measure dimension. Cubes are physically built of measures, but the measures are mapped to items and the individual measures are then hidden from the users.

Another reason for creating an Item dimension is that the Measure dimension is not the same as other dimensions. The members of the Measure dimension are schema-based, i.e., mapped to specific columns in a table, whereas the members of all other dimensions are value-based, i.e., each member maps to a row in a table. Because of this, the Measure dimension is more limited and less flexible than other dimensions. A hierarchy of measures cannot be built. Nor can measures have custom properties that control processing and selection.

It is not uncommon to have tens, if not hundreds of measures in a PM application. Without a hierarchy to organize measures, it can be cumbersome for users to find the measure(s) they need for a query or analysis. Also, hierarchies allow for native drill down in client tools, further facilitating analysis. Drilling down on Net Sales to see its components of Gross Sales, Discounts, and Adjustments is helpful. Having an Item dimension that is value-based allows for hierarchies to be maintained with minimal effort.

Specifying properties about measures creates a flexible application. For example, currency conversion is only applicable for financial data. Therefore, it is important to know which data should be converted and which data should not be converted. Rather than hard coding a list of financial measures in the current conversion calculation, a better practice would be to use a value type property to control which data is converted.

An Item dimension can have properties. The PROOF methodology specifies an initial schema for the Item dimension that contains a set of properties that are useful across applications. This schema can be extended to include other properties that might be useful in a given application.

Dimensionalized Metrics

A major aspect of a PM application is building metrics from the measures that are collected from business operations or external sources. Metrics are comparative in nature and provide a context for

judging the performance of measures. A specific value for sales has no context. People in an organization naturally add context because they have knowledge about the business. Is the value better or worse than expected? Has it grown since last year? What percent of the total is it? PM is about evaluating performance based on these comparisons. Comparative performance can be classified into what Symmetry calls the 3T's of business performance:

- Trends – measure performance vs. history, such as growth from last year or last period
- Targets – measure performance vs. goals, such as variance between actual and budget or forecast and industry benchmarks
- Typical Rankings – measure performance vs. peers in a group (generally a hierarchy), such as the percent-of-total or rank

The common method of building metrics from measures is to create new calculated measures that embody the metric. From sales, metrics would be created such as last year sales, change from last year, variance from plan, product percent-of-total, customer rank, and so on. This would be done for each measure in the application. A robust PM application can have many source measures, with each source measure having 10 or 20 derived metrics. This can lead to a situation called "measure bloat" where users must navigate through hundreds of measures, making it very hard to find the specific metrics desired – particularly if the measures do not adhere to standard naming conventions. Another problem with this approach is that there is no leverage. When a new measure is added, all of the associated metrics must be developed. Ultimately, this approach creates a situation where each metric for each measure is a separate development effort. This is not a scalable design pattern.

Using a dimensional approach to metrics provides consistency and facilitates application extensibility. Metrics are generally defined by a combination of attributes. These attributes can be used to simplify the navigation of the metrics. A "dimensionalized" metric can be created by decomposing the metric into:

- a dimension that controls a base value
- a dimension that controls the comparison value
- the metric to be calculated from the comparison

To support dimensionalized metrics, the PROOF design patterns use a set of analytical dimensions.

Analytical Dimensions

Analytical dimensions differ from application dimensions in that they define metrics, not data, and apply across all measures. Application dimensions define the structure of data, e.g., a sales measure can be dimensioned by time, product, and customer. Measures are not natively dimensioned by analytical dimensions. (From a technical point of view, an analytical dimension is not part of the composite key of a measure.)

A simple illustration of the use of analytical dimensions is building the target metrics around the Scenario dimension. Assume there are three scenarios: actual, budget, and forecast. A common report might show actual, budget, actual/budget variance, and actual/budget variance %. Another report that

shows actual, forecast, actual/forecast variance, and actual/forecast variance % would have a different definition. In the common approach, a series of metrics would be derived: actual budget variance, actual forecast variance, and budget forecast variance. Also, the series of metrics would be extended to include variance percent.

The PROOF design pattern for this series of metrics is much more flexible and maintainable. The target metrics around the scenario dimensions are decomposed into the following three dimensions:

Scenario Dimension
Actual
Budget
Forecast

Scenario Compare Dimension
Actual
Budget
Forecast

Scenario Variance Dimension
Base
Compare
Variance
Variance %

The Scenario dimension defines the base member. The Scenario Compare dimension defines the Scenario member being compared and the Scenario Variance dimension defines the metric to calculate. Using Actual as the Scenario and Budget as the Scenario Compare, the Scenario Variance dimension would display the following members for a Sales Item:

- Base – the Actual value for Sales
- Compare – the Budget value for Sales
- Variance – the numerical variance between the Actual and Budget Sales value
- Variance % – the percent variance between the Actual and Budget Sales value

In this example, the Scenario Variance dimension is using a relative reference to the members chosen from the Scenario and Scenario Compare dimensions, much the same way a relative reference is written in Excel. This relative reference approach does not require that more calculations be added to the cube with the addition of new scenarios. By dimensionalizing the metrics, the dimension and its members can be used as relative references in the calculation script. With a generalized calculation script, the addition of new data does not require changes to the calculation script, reducing the need to verify that the calculation script is correct. Additionally, with this approach, reporting is more flexible as reports can be parameterized thus eliminating the need to develop additional reports when additional dimension members are added.

Embedded Metadata

The PROOF methodology stipulates that application metadata should be embedded with the data, though hidden from end users. While database engines, both relational and multidimensional, have vehicles for querying metadata, the metadata accessible through the engines is structural in nature, not application oriented. Having a set of metadata that is identical across the relational and multidimensional platforms is application-oriented and extensible and gives the application developer more power in building a flexible and maintainable application. The embedded metadata is particularly useful in building process control modules and in reporting. Another benefit of this approach to

metadata is that it is extensible, while the native metadata of both the relational and multidimensional databases is not.

Relational metadata is very complete in its knowledge about the relational schema but does not know about the application schema. As a result, the relational metadata is ignorant about which tables are dimensions and which tables are facts. The multidimensional metadata understands dimensions and measures, but the method to query metadata is separate and distinct from the method to query data and is not accessible through the client tools, only through an API.

To remedy these deficiencies, the PROOF methodology specifies a set of metadata schemas for both the relational and multidimensional databases. For example, PROOF specifies the creation of a cubes dimension table in the relational database and a cubes dimension in the multidimensional database. Having a set of metadata that is consistent and synchronized between the relational and multidimensional databases provides easier and more useful access to the application metadata.

Conclusion

The design patterns in PROOF have been created as the result of lessons learned over the last 25 years of building management reporting, decision support systems, executive information systems, business intelligence systems, and performance management systems in a variety of technologies and products for a broad spectrum of clients and industries.

In summary, the benefits of the PROOF approach are:

Correctness = Credibility

Accurate metrics and validated formulas are key components of a successful BI application. Incorrect data is one of the quickest ways to lose credibility within an organization. Typically, metrics are scattered throughout an organization in spreadsheets and in reports. Few standards exist for managing metrics. BI software provides a data platform and a presentation platform, but no BI software currently on the market provides a metrics platform. Developers handcraft metrics. This means the processes are not easily repeatable, scalable, or auditable. Adhering to the PROOF methodology ensures that metrics are centrally created and maintained to ensure a single version of the truth.

Consistency = Dependability

Since most organizations have multiple BI applications, it is important that applications follow BI best practices that provide both a consistent user experience and consistency in how metrics are calculated and accessed. Practices such as standard naming conventions and the use of uniform solutions for similar problems are very important. By adhering to PROOF's standard approaches, application usability is maximized because of consistency across applications. Furthermore, the experience of both developers and business users can be leveraged more efficiently across the organization since less training is necessary when moving from one application to the next.

Context = Better Interpretation of Data

Analysis must have context. Metadata provides context to determine how results should be calculated and displayed. Being clear about the context for data affects the interpretation of that data. For example, specifying the polarity for variance can determine whether an increasing number is good or bad for business. Clearly for revenue numbers, increases are favorable, whereas for expenses, increases are unfavorable. Context can also vary based on the type of data (e.g., financial, volume, or price). For example, when scaling numbers, financial numbers may be scaled in thousands, but price should not be scaled. The PROOF methodology specifies a full set of application metadata that provides a more robust context for the creation and interpretation of metrics.