White Paper

A Best Practice Guide to Designing TM1 Cubes

What You'll Learn in This White Paper:

- General principles for best practice cube design
- The importance of the Measures dimension
- Different approaches to the Time dimension
- When to use separate dimensions vs. hierarchies vs. attributes
- Considerations for budgeting and forecasting

TM1 is an extremely flexible application development platform and includes all of the tools required to develop a complete application from end to end. The flexibility of the environment allows applications to be developed very quickly; application prototypes can be developed in a matter of hours and full applications can be developed in just days or weeks.

The TM1 development environment is also very fast to learn with newcomers to TM1 capable of developing their own applications within just a couple of days of training. The system is very intuitive, especially for those who have developed complex spreadsheet models in the past.

The flexibility described above enables applications to be built quickly and easily, however this level of flexibility can also have some undesirable side effects if projects are not developed or managed well. It is very easy to create "bad" applications that may solve specific business problems, but leave little room for future development and improvement.

To ensure that the TM1 platform is fully leveraged and systems are developed to a high standard there are two key principles that are recommended:

- Implementation of systems that follow best practice development principles
- Management of TM1 projects using a methodology that complements the technology

This document is intended to address the first point; that is to describe an approach to developing TM1 systems using best practice principles.

Why Use Best practice?

Using a best practice approach to developing systems is a way of developing systems that are:

- Efficient
- Consistent
- Reliable
- Easy to understand
- Easy to maintain
- Easy to extend

The intention of best practice is to establish a series of development guidelines and principles that are the result of the collective experience and learning of many TM1 developers over many TM1 projects and systems. Often this is the result of trying many different approaches to solving problems and settling on the approach that is most suitable in the widest variety of circumstances.

Best practice is also a way of making the overall development approach consistent across the entire TM1 system (or systems). This enables the system to be easily understood, maintained and extended by multiple developers with potentially different backgrounds and experience.

It is not the intention of best practice to dictate how problems should be solved, that will always be the creative responsibility of the developer or developers involved in developing TM1 systems. Instead it is a framework of guidelines and principles that allows the creative process of developing systems to result in an efficient and consistent outcome that is easily understood by fellow developers and allows the systems to be easily maintained and extended in the future.

In addition, using best practice allows the ongoing maintenance and development of the systems to happen faster and be more efficient. This is because new developers will be working with a system that is easy to understand and therefore faster to learn. It will also result in the build up of a library of standardised code that acts as a blueprint for future development.

Introduction

Like most development platforms, there are many different ways of achieving similar outcomes when building TM1 systems, and this includes cube design. Two different developers presented with the same business problem are unlikely to come up with the exact same design; in some cases their respective designs may be vastly different.

Some multi-dimensional platforms have strict constraints around cube design, particularly when defining concepts like time and measures within a cube; however TM1 does not impose these constraints, or at least not as strictly. It is possible to create TM1 cubes with virtually any combination of dimensions, and it is not mandatory that they include time or measures dimensions.

This gives TM1 developers great freedom when designing and building TM1 cubes, however this flexibility can lead to cubes that are inconsistent and even incompatible with other cubes unless best practice design principles are followed.

Sample Business Requirement

To help illustrate the recommendations in this white paper, the following sample business requirement will be used:

> A TM1 application is required to report sales information for Company ABC. ABC uses sales representatives to sell many different products across varied product categories to customers. ABC sets a standard list price for each product on a monthly basis; however sales people are allowed to negotiate these prices with customers in order to make a sale.
>
> ABC would like to achieve as close to the list price as possible to maximise their revenue and would like the ability to monitor the level of discounting that has been used across their sales force in order to make sales.
>
> As well as reviewing total sales data across the company, they would also like to drill down through the data by sales person, product & product category, customer and time. They would like to see sales data summarised by year, quarter and month as well as by week. Monthly totals should include all days in the month; weekly totals should be based on a Monday to Sunday sales cycle.
>
> At some point in the future, ABC would like to extend their sales reporting system to include budgeting. Budgets will be held by month, by sales person, by customer and by product category. List prices are budgeted on a monthly basis and sales people are expected to achieve at least 90% of list price on average for the month.
>
> Data will be available to be loaded from the ABC sales system on a daily basis. Data on individual sales will be available in one extract; monthly list prices will be available in another extract.
>
> The following fields are required at a minimum for reporting from the TM1 application:
>
> - Units Sold
> - List Price
> - Sale Price
> - Revenue
> - Discount $
> - Discount %
>
> There will be multiple end users of the application. Each user may use any of the standard TM1 interfaces for reporting e.g. Perspectives (Excel & TM1Web), Executive Viewer etc. In the future, ABC would also like to use the TM1 cubes as a data source to their existing Cognos BI system.

Cube Architecture

The foundation of all TM1 systems is the collection of cubes that contain the application's data. Careful consideration should be made about the number of cubes to include in the application and the granularity of data stored in each cube.

Single Cube vs. Multiple Cubes

There can be a temptation to limit the number of cubes in an application and try and store as much data as possible in a single cube to make it easier for the end user to retrieve data. However, TM1 has many capabilities for retrieving data from multiple cubes so the real focus should be on what is the best cube architecture that supports the whole application, regardless of how that data may be presented and through which end user interface.

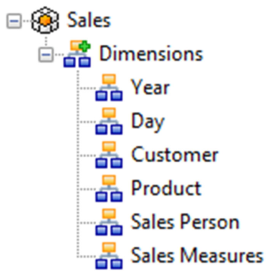
For example, consider the sample business requirement described earlier in this white paper. The requirement is to analyse and report on sales information, so the temptation may be to create a single sales cube and load all data into that cube including daily sales, list prices etc. However not all of the source data is at the same level of granularity, for example:

- **Sales data** including quantity sold and sales price is recorded at ***daily*** level. This data includes information about the sales person, product and customer.

- **List price data** is recorded at ***monthly*** level. This data includes information about the product only and does not include information about the sales person or the customer.

As you can see, the data for sales and the data for list prices are clearly at a different level of granularity for two reasons: (1) the level of detail of the required time dimension is different between the two data sets, one requires daily detail and one requires monthly detail; and (2) the fields that define the two data sets are not the same, one uses sales person, product and customer and the other only uses product.

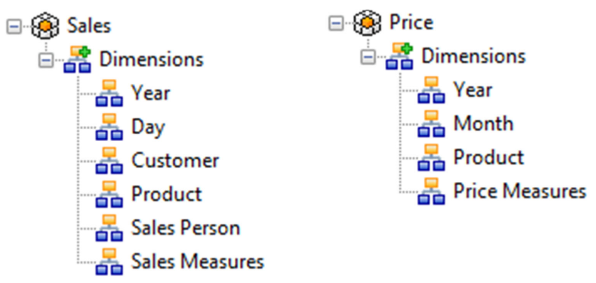
If a system was designed with only a single cube then the cube would need to be designed to cater for the lowest level of granularity, in this example it would need to have dimensions for sales person and customer to be able to store the sales data. However this would mean that when list prices were loaded into the cube they would have to be loaded against sales person and customer as well. In addition, list prices would now need to be stored at daily level. The cube would likely have the following dimensionality:



Given that the list price data we are receiving from the sales system does not specify sales person, customer or day, we would have to make a decision about how to store this data. Do we store the list prices against every customer, every sales person and every day in the month? This would potentially result in storing the list prices many thousands of times more than we need. This would in turn result in the cube becoming unnecessarily bigger than it needs to be. After all, we only need to see list prices where an actual sale has been made so that we can calculate discounts.

Alternately, do we create "dummy" elements in the sales person, customer and time dimensions to hold list price data? This would limit the amount of times we store the data but would likely result in confusion when users are retrieving data from the cube. In the case of the time dimension, the situation is even worse as dummy elements would be required for months, but month would already be a consolidated element containing all of the days in the month! As numeric data cannot be stored in consolidated elements there would need to be two elements for every month causing considerable confusion.

A better solution in this example would be to have separate cubes for sales data and list price data. The sales cube would have the appropriate dimensionality to store data at a daily level against sales person, customer and product; and the list price cube would have the appropriate dimensionality to store data at a monthly level against product only. For example:



Note that the sales cube is the same as it was in the earlier example, however the new price cube does not contain the customer or sales person dimensions, it uses a month dimension instead of a day dimension and it has a different measures dimension.

Of course, we still need to retrieve the list price for each sale in the sales cube so that we can calculate the discounts. This can be easily achieved using an inter-cube rule in the sales cube as follows:

```
['List Price'] = N: IF(['Units Sold'] <> 0, DB('Price', !Year, ATTRS('Day', !Day, 'Month'), !Product, 'List Price'), CONTINUE);
```

Note that the list price is only retrieved when there is sale i.e. when units sold is not equal to zero. Note also that the rule converts from a daily level in the sales cube to a monthly in the price cube, effectively applying a monthly list price to a daily sale.

The advantage of this approach over the single cube approach is that we will "look up" the list price only when it is needed, rather than a "brute force" approach to storing this data.

Dimension Order

Once you have decided how many cubes you need and what dimensions they will contain, the next consideration is the order in which the dimensions are included in the cube. The dimension order has a number of impacts across the TM1 system and requires careful consideration. Some of the key considerations to be made include:

- Performance
- Memory Consumption
- Usability
- Consistency
- Maintenance

An important observation needs to be made here:

Ordering dimensions for best performance and the least amount of memory consumption can often result in a dimension order that is the polar opposite of the order that would be used for best usability, greatest consistency with other cubes and lowest levels of maintenance.

That is why TM1 supports two different dimension orders: one order for presentation purposes and one order for physical storage.

When initially creating a cube, the order in which the dimensions are specified will be used for both the presentation order and the physical order. However, cubes can be "re-dimensioned" after they have been created to change their physical order without having any impact on the presentation order. Cubes can re-dimensioned over and over if needed, however the presentation order cannot be changed after the cube has been created.

Therefore, the primary considerations for dimension order when initially creating a cube should be based on usability, consistency and maintenance. Considerations for performance and memory consumption should be made afterwards.

When considering a dimension order for best usability, think about what questions the end user would ask when selecting slices of data from the cube or writing a formula in a spreadsheet to retrieve the data. Are they likely to select a customer from the customer dimension first or a product from the product dimension first? You should order the dimensions in the order that the user is most likely to choose them.

If your cube has multiple time dimensions then users are most likely going to select from the broadest category first and then the more detailed categories second, e.g. select year followed by month followed by day. Therefore it makes sense to add the time dimensions to the cube in that order.

When considering dimension order, it is also important to choose an order which is consistent with other cubes in the system. Dimensions such as Version, Year and Month are likely to appear in many cubes and should be selected as the first dimensions in a cube and in the same order. This will make writing and maintaining rules, turbo integrator processes and excel based reports much easier.

After the presentation order has been decided and the cube has been created, it is then time to consider the best physical order of the dimensions to achieve the most optimal performance and the lowest amount of memory consumption. There are a number of general rules of thumb for determining the best physical order of dimensions within a cube such as:

- Order dimensions from smallest to largest
- Order dimensions from sparsest to densest

The first option places the dimension with the least number of elements first, then the next smallest number of elements second and so on, placing the dimension with the largest number of elements last. The second option places the sparsest dimension first and then the next sparsest dimension second and so on until the densest dimension is placed last. The second option is a little more challenging to work out as you will have to have a clear understanding of how the data is "profiled" across your dimensions.

These rules are often combined further to create a single rule of thumb:

- Order dimensions from smallest sparse to largest sparse, followed by smallest dense to largest dense.

So, those are the theoretical rules of thumb when ordering dimensions for best performance and least amount of memory consumption. However, these rules may be a good starting point but they do not always result in the optimal dimension order. Other factors such as cube rules and their associated feeders also have an impact on cube performance and size, and these factors are not always well reflected in these general rules of thumb.

Based on our experience with finding optimal dimension orders, we have actually seen evidence where the most optimal order does not always follow these rules. In some cases we have even seen optimal dimension orders which contradict these rules completely!

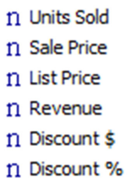
In any event, finding the optimal physical order can have significant impacts to performance and memory consumption, and time should be taken to find the order that is most optimal. This can be analysed using trial and error using the cube re-dimensioning tool within the TM1 Environment, however this can time consuming. The Cubewise Vizier application includes a cube optimisation feature that takes the guesswork out of this process, finding the optimal dimension order for you.

One final note to make on cube optimisation: the optimal dimension order can change over time and should therefore be reviewed regularly. As cubes grow with the addition of more data, new rules and feeders, and as the size and structure of dimensions change over time, it is likely that at least some of the cubes will require regular re-dimensioning to find the optimal order. Cubes should be reviewed on a quarterly basis to ensure their dimension order is as optimal as possible.

Measures Dimension

As stated earlier in this white paper, TM1 doesn't impose strict rules on cube design like some other platforms. It is therefore possible to create TM1 cubes that don't contain separate dimensions for time or separate dimensions for measures. However this doesn't mean that such dimensions shouldn't be used, on the contrary they should be used for all but a small handful of circumstances.

In some cases, it is plainly obvious that a separate measures dimension is required and you would naturally create one and use it in your cube. In our sample business requirement there are clearly a number of measures that are required including units sold, sale price, revenue etc. These measures would be created in a separate measures dimension in the sales cube as follows:


Units Sold
Sale Price
List Price
Revenue
Discount $
Discount %

However, there are some cases where a measures dimension is not so obvious and in these cases it may be tempting to omit a measures dimension altogether. A good example from our sample business requirement is the price cube. This cube will contain list prices only, not units or revenue or discounts. As a result, the measures dimension contains only a single element as follows:

List Price

For any dimension other than the measures dimension, creating a dimension with only a single element would generally be considered bad practice as the user will only be able to select a single element from the dimension, effectively make it redundant. There are some cases when you might include a dimension with a single element to enable expansion of the system in the future, but if there is no chance of additional elements ever being required to that dimension it will generally be omitted.

Another classic example is the general ledger cube. More often than not, a general ledger cube will hold information about balances and/or movements from the source general ledger system. There will likely be dimensions for company, cost centre, account, year, month, currency, version etc. However, it is often the case that general ledger cubes are built without a measures dimension. The main reason for omitting the measures dimension in the case of a general ledger cube is that it would contain only a single element which would be called Value or Movement or Balance.

However, even though the measures dimension is not strictly required by TM1, it does receive some special treatment. For this reason, cubes should always contain a measures dimension even if it only contains a single element. The main reasons for this are described below.

Firstly, TM1 can be used as a data source for other applications. Some applications such as Cognos BI will access TM1 cubes directly as they are part of the same product family. Other applications can access TM1 cubes via MDX queries as an ODBO data source. As many other OLAP platforms ***do*** have strict requirements for a measures dimension, TM1 cubes simply will not work in these other platforms without the presence of a measures dimension. Our sample business requirement stated that company ABC would like to use TM1 cubes in their Cognos BI system in the future, so a measures dimension is definitely required in this case.

Secondly, TM1 uses the data types from the elements of the ***last*** dimension in a cube to determine the data types of the cells within the cube. This is particularly important when a cube contains string measures. By creating a measures dimension and always including it as the last dimension in the cube, you will always have complete control over the data types of elements in the cube regardless of the data types that may have been used in the other dimensions.

While many of the dimensions within a cube may be shared with other cubes, there should always be a separate measures dimension that is unique to each cube.

Time Dimensions

One of the most important considerations when designing a cube are the time dimensions. The vast majority of TM1 cubes will have at least one time dimension; it is the fundamental concept that allows data to be trended over a period of time and allows the comparison of data between different periods of time.

Furthermore, TM1 is often used for both retrospective (looking back) and prospective (looking forward) applications using the same subject matter. It is the role of the time dimensions to allow the analysis of data in the past, present and future.

There are numerous ways to structure time dimensions within a TM1 cube depending on the data that it holds and the requirements of the business. This section describes the different approaches to creating time dimensions and when each approach is most appropriate.

There are a number of important considerations to be made when deciding on how many time dimensions are required for your application. These considerations include:

- What is the lowest level of time detail required in your cube?
- Will the data in your cube require regular and consistent time cycles?
- Will the data in your cube require any unusual, overlapping or inconsistent time cycles?
- Will your cube require any “rolling” time periods?
- Will your cube be sharing data with other cubes already in your system?

In addition to these considerations, it is important to think about the level of effort required for the end user when accessing data from the cube, and for the administrator when performing maintenance on the system. Your design should also consider:

- Flexibility when comparing data across multiple time periods
- Flexibility when creating new consolidated time periods in the future
- Maintenance effort required to update time dimension(s) as new cycles are added

The following sections provide guidance for some of the most common requirements:

Month as the lowest level of granularity

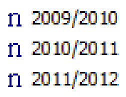
For cubes that will store data at the monthly level of detail the best approach is to have two time dimensions: one dimension for year and one dimension for month.

The advantages of using a separate year and month dimension are as follows:

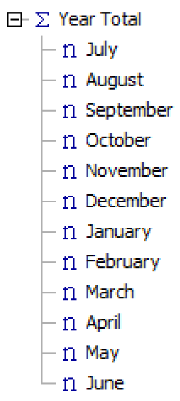
- Provides for easier year on year comparisons
  Enables easier use of relative data spreading from one year to another
  Requires less maintenance than a single time dimension as only one element needs to be added to the year dimension once a year ***versus*** adding twelve new periods and updating all of the relevant hierarchies for a single time dimension

For financial cubes such as general ledger, the year and month dimensions should usually conform to the financial year of the business. For example, for the financial year from July 2010 to June 2011, the corresponding element in the year dimension would be 2010/2011 and the corresponding elements in the month dimensions would be July, August, September, October, November, December, January, February, March, April, May and June. The month dimension would also include a consolidated element for the Year Total and potentially other consolidations for quarters, half years, year to date etc. For example:

Year Dimension


2009/2010
2010/2011
2011/2012

Month Dimension


Σ Year Total
July
August
September
October
November
December
January
February
March
April
May
June

For non financial cubes, the year and month dimensions may conform to the same structure as described above for financial cubes, or in some cases it may be more appropriate that the year and month dimensions use a calendar year and a set of months running from January to December.

The use of separate year and month dimension would be the best approach in most cases. However, there are some circumstances where using a single time dimension may be more valuable. If the application requires rolling totals for example, it is easier to model these using consolidations in a single time dimension rather than using rule calculations in a cube that has separate year and month dimensions. Single time dimensions can also make the calculation of certain depreciation, annuity and other calculations easier using TM1 rules.

Week as the lowest level of granularity

For cubes that will store data at the weekly level of detail the best approach is to have two time dimensions: one dimension for year and one dimension for week.

Per the previous section, the year dimension would usually follow the financial year of the business or perhaps a calendar year for non financial cubes. However, the week dimension generally requires a little more thought.

The use of a week dimension creates two challenges that need to be carefully considered when deciding on the best design:

- A 365 or 366 day year does not contain an exact number of weeks
- A 29, 30 or 31 day month does not contain an exact number of weeks

With regard to the first point, it is generally accepted that a year contains 52 weeks. This of course is not exactly true as 52 weeks = 364 days and a year contains 365 days (or 366 days in a leap year). Using a 52 week system will eventually result in the days that are assigned to each year becoming more and more out of sync with the calendar year as years pass.

For this reason, week dimensions should be set up with ***53*** weeks, not 52. In most years only the first 52 weeks will be used, however once every 5 or 6 years the 53rd week will be required to "catch-up" on the loss of 1 or 2 days each year. When comparing data year on year, it is important for users of the system to know when they are comparing 52 week or 53 weeks years as this will have some obvious impacts on variance analysis.

With regard to the second point, it is often a requirement to "roll up" weeks into months for monthly reporting and variance analysis. Once again, weeks just don't fit "neatly" into a month and so a decision needs to be made on which weeks roll into which months. A commonly used approach is to roll up weeks on a 4-5-4 or 4-4-5 basis.

Using the 4-5-4 approach would result in the first 4 weeks being assigned to the first month, the next 5 weeks assigned to the next month and the next 4 weeks assigned to the third month. This pattern repeats again for the next 3 months and so on. Regardless of the pattern, e.g. 4-5-4 or 4-4-5, there should always be 13 weeks assigned to each quarter. The only exception is when the 53rd week is used, in which case one quarter will contain 14 weeks.

Note that a month in this case will not contain an exact calendar month of data as each month will contain either 28 days (4 weeks) or 35 days (5 weeks) of data. The only occasional exception may be when February on a non-leap year coincidentally aligns.

One final point needs to be made here. In a system that uses a separate year and week dimension, the pattern of allocation of weeks to months, i.e. 4-5-4 or 4-4-5, needs to be consistent from year to year. This ensures that when comparing a month in the current year to the same month in the previous year you will always be comparing the same time period, i.e. 4 weeks vs. 4 weeks or 5 weeks vs. 5 weeks. If for any reason the pattern needs to change from year to year, then a single time dimension will be more appropriate.

Day as the lowest level of granularity

For cubes that will store data at the daily level of detail there are a number of different options.

Option 1: If the business concept being modelled follows primarily a weekly cycle then time can be modelled using three dimensions as follows:

- Year
- Week (rolling up to month)
- Weekday (Monday, Tuesday etc)

This construct is useful when the day of the week is likely to have significance in the review and trending of data. For example, a retail business will likely have greater sales on weekend days than working days. It supports the ability to analyse day on day, week on week, month on month and year on year comparisons.

Per the previous section, this approach requires that the number of weeks assigned to each month is consistent year to year e.g. weeks roll into months on 4-5-4 or 4-4-5 basis. Each month will contain exactly 4 or 5 weeks, it will not contain the 1st day to the last day of the calendar month in most cases.

The maintenance using this approach is low; an additional year needs to be added to the year dimension once each year.

Option 2: If the business concept being modelled follows primarily a monthly cycle then time can be modelled using two dimensions, with the option of a third as follows:

- Year
- Day (Jan 1, Jan 2 etc. rolling up to month)
- Week (Optional extra dimension if weekly reporting is also required)

This construct is useful when data needs to be primarily analysed on a monthly basis with the ability to drill down to individual days. The addition of the extra week dimension adds the ability to create weekly reporting; however the data load process will need to know which week to assign each day worth of data to during the load process.

Option 3: If the data being modelled needs to be more flexible than the two options described above, then a single time dimension may be more appropriate. For example, if there are overlapping or alternating time consolidations required. Consider the following requirements:

- Analysis of data is required at a fortnightly level, however some parts of the business use one set of fortnights and other parts of the business uses the alternate fortnight e.g. Jan 1 – Jan 14 versus Jan 8 – Jan 21.

- Analysis is required at a weekly level, however one region uses a Monday – Sunday cycle and another region uses a Sunday – Saturday cycle.

Both of these examples require consolidations that overlap. In such cases time can only be modelled effectively using a single time dimension with multiple hierarchies.

Using a single time dimension allows the creation of any possible grouping of individual dates; however it requires the most amount of maintenance and affords the least amount of flexibility when performing period on period comparisons.

Day Part as the lowest level of granularity

If data needs to be stored at a lower level of detail than day, then an additional dimension should be added for the day part. For example if data is to be captured by minute, then a minute dimension should be added that rolls up minutes into hours and the hours into full days. Minutes could also be rolled up into half hourly or 15 minute increments if relevant.

Dimension vs. Hierarchy vs. Attribute

Often, one of the most difficult decisions when designing a cube is whether to create a separate dimension, a dimension hierarchy or an attribute to model a particular requirement.

Consider our sample business problem once again. The requirements were to be able to analyse sales data by sales person, customer and product. This may sound straightforward in principle, however may be more complicated in practice. Consider the following scenario:

> Since the beginning of the year John has been responsible for selling a particular category of products. However, last month John left the company and was replaced by Tim who is now responsible for selling the same category of products that John had been selling.

There are a number of important questions that need to be asked in order to correctly model a system that supports this type of behaviour. For example:

- Should any sales that John made prior to leaving now be re-assigned to Tim as well as any new sales Tim makes?; or

- Should any sales that John has made previously stay with John and only new sales be assigned to Tim?

The answer to these questions will likely have an impact on the cube design. In this scenario, if the answer to the first question is yes, then it would be best to model the sales person concept as a consolidation in the product dimension. This way, when John left and Tim took over the only change to the system would be to change the consolidation point from John to Tim.

However, if the answer to the second question is yes, then simply changing a consolidation point won't work. In this case, the sales of the products in the category in question will now contain some sales that should be associated with John and some with Tim. It would be possible to create two hierarchies, one for John and one for Tim however this would not indicate which sales each salesperson made. In this case, it would be better to create a separate sales person dimension and assign each sale to the appropriate sales person at the time the data is loaded.

Consider another example. What if some sales people were assigned to product categories and some sales people were assigned to certain key customers? The sales people assigned to particular product categories are allowed to sell to any customer but only the products they are responsible for. The sales people assigned to key clients can sell only to their assigned clients but can sell any product to them.

In this example, it would not be possible to model the sales person relationship using a hierarchy as sometimes sales people are responsible for products and sometimes sales people are responsible for customers. In this case, it would be appropriate to include a separate dimension for sales person.

Naming Conventions

Naming conventions should be used throughout a TM1 system to ensure that object names are meaningful and consistent. This is particularly important in TM1 compared to other systems as the names of TM1 objects are exposed directly to the end user via the TM1 server explorer and through the reporting tools. Cubes, dimensions, elements, views, subsets, applications, processes, chores etc may all be seen by end users if their security access permits it.

In most cases, TM1 does not impose strict rules for naming objects leaving the developer with a great deal of flexibility when deciding on names. Of course, the downside of this is that anyone building an object can call it anything they like which can lead to systems becoming inconsistent and potentially confusing to the end user. This section describes a best practice approach to naming objects.

General Conventions for All Objects

Object names should be descriptive without being too verbose. If a cube contains sales information then it should likely be called **Sales**, not **Product Sales and Revenue**. However, names should also not be overly abbreviated, it is better to call a cube **General Ledger** than **GL** for example.

Object names should be user friendly and convey business terms and meanings rather than be too technical. They should also contain spaces, and use case effectively to make them more user-friendly. Calling an object **General Ledger** is much better than calling it **GeneralLedger** or **general_ledger**.

Object names should avoid using punctuation or other special characters such as #, &, * and %. This can cause problems when object names are used in rules, turbo integrator process or as parameters in SQL statements. Call a dimension **Item Number** rather than **Item #**.

Conventions for Naming Cubes

As stated earlier, objects should be named using business terms. Cubes should be named things like **Sales**, **General Ledger**, **Inventory** or **Payroll**.

Avoid using embellishments in the name, like **Payroll Data** or using specific system names like **PeopleSoft General Ledger** or **SAP Payroll**.

Conventions for Naming Dimensions

Dimension names should always be singular rather than plural. Use **Cost Centre** rather than **Cost Centres**. They should be user friendly and use conceptual names rather than “field names”. If dimensions are being built using a turbo integrator process that accesses data from another system, you don’t have to use the name from the source system. It will make more sense to a user to call dimensions **Business Unit**, **Operating Unit**, **Cost Centre** rather than **Segment 1**, **Segment 2** and **Segment 3**.

Dimension names should always describe the lowest level of detail in the dimension. For example if you have a dimension that has cost centres that roll up to regions, and then to countries then the dimension should be called **Cost Centre** and not **Region** or **Country**.

Conventions for Naming Elements

Element names should also be as meaningful as possible; however element names have a very powerful additional feature: the alias. Aliases allow the same element to be referred to by different names which can be extremely useful.

For element names that are loaded from an external system, the leaf level elements should use the same name as they do in the source system and an alias used to create a more meaningful descriptive name. It may be useful to create multiple aliases for different reporting requirements. Aliases that display descriptions, short names and long names, code and description combinations are all useful.

Make sure that the convention you use for naming elements is as consistent as possible between different dimensions. For example if you use square brackets for codes in one dimension you should use the same in another. If you have **[100] Marketing** in the cost centre dimension you should use **[01] Company ABC** in the company dimension.

When creating consolidation points in different dimensions be as consistent as possible. If you have a top node called **All Cost Centres** in your cost centre dimension it would be best to have an **All Business Units** node in your business unit dimension rather than **Consolidated Business Units**.

Where possible, avoid using the word **Total** in element names. It will be obvious to the user that the element is a subtotal if they are viewing in the cube viewer so you can call a node **Revenue** rather than **Total Revenue** or **Gross Profit** rather than **Total Gross Profit.**

Numeric vs. String Cubes

TM1 cubes can contain two different data types: numeric and string. It is possible to combine both numeric and string data in a single cube; however this has some impacts on system performance and can introduce some undesirable constraints.

Firstly, cubes that contain both numeric and string data will calculate slower than if the cube contained numeric data only.

Secondly, when re-dimensioning a cube for optimal performance and memory consumption, the presence of string elements will constrain the ability to find the optimal order. Per the earlier section on dimension ordering, it is the last dimension in a cube that determines the data types of the cells within the cube.

When a cube contains only numeric elements, the dimensions can be re-dimensioned in any order, including an order where the measures dimension is no longer the last dimension. However, when a cube contains string elements, the measures dimension must always be the last dimension in the cube, otherwise the data types will not work as intended. It is possible to re-order the other dimensions in the cube, but this may lead to an order that is far from optimal.

For these reasons, it is far better to create two separate cubes, one containing numeric data and one containing string data. The only exception to this is when working with very small cubes where performance is not really an issue. Re-dimensioning cubes such as assumption, parameter, system info, will have very little impact.

Comment Cubes

One of the most useful features of TM1 is the ability to store commentary or free text data in cubes. These comments are commonly used to describe the numeric data in plain English and provide reasons for trends, variances and other movements in data. They assist the reader in understanding the data they are looking at.

Per the previous section, it is best to store numeric and string data in separate cubes, so comment data should be kept in separate cubes from the cubes they are commenting on. In some cases, commentary data may be required at the same level of granularity as the numeric data; in other cases comments may only be required at a more summarised level of detail.

In either case, a comment cube should include:

- The same dimensions as the cube it is commenting on with the exception of the measures dimension
- The last dimension should be standard comment dimension containing a string element for the comment data

Even if the current requirement is to provide commentary at a more summarised level than the numeric data itself, it is best to keep the dimensionality of the two cubes the same with the exception of the final measures dimension. This allows commentary to be captured at a lower level of detail in the future if the requirements change. In the meantime, commentary can be stored in consolidated elements such as Gross Profit if that is where the comment is required.