Paper 044-2009

Implementing a BI reporting environment – making dashboards more OLAP friendly

Gerard Quinn, Credit Corp Group, Sydney, Australia

ABSTRACT

The Credit Corp reporting environment is used on a daily basis by the management team to track performance at all levels of the business. Taking advantage of SAS® Enterprise BI Server (in particular SAS BI dashboards, SAS OLAP Server®, and SAS® Web Report Studio), users are able to get a high-level view of performance through a series of dashboards and then can further analyze data at a lower level with OLAP reports. This presentation is aimed at SAS® users implementing or thinking of implementing a BI portal solution. It discusses in detail the main features of the reporting environment including applying data security to OLAP cubes in an automatic way and building BI dashboards that are dynamically generated to show the appropriate information for each user.

INTRODUCTION

As a SAS BI consultant previously I wasn't new to the prospect of implementing a SAS® Business Intelligence solution, but in my first role since leaving SAS it was great to work with the new SAS® BI dashboard framework for the first time.

Credit Corp was very much a blank canvas with limited reporting capability. This meant there was the opportunity to approach the design from a fresh perspective. The implemented solution is an online reporting environment that makes use of SAS® Enterprise BI Server components like SAS® Information Delivery Portal, SAS® OLAP Server for slice and dice analysis, the SAS® BI Dashboard to show performance indicators at a glance and report delivery components like SAS® Web Report Studio and stored processes to provide a more detailed level of reporting.

KPI performance monitoring and dashboards tailored to the individual levels of management have been delivered in the first phase of development to monitor the health of the business and be able to react to the ever changing landscape of the credit industry. Credit Corp is becoming more and more focused on driving higher returns from an aging debt portfolio rather than collections from regular periodic purchases. This means working smarter not harder.

The project to implement the BI portal at Credit Corp, named Creditport, began in July 2007. The time to delivery from initial requirements gathering to the first phase going live was about 6 months including both data and report development. Since that time we have continued to add new reports and increase the user base. As with any project of this nature there were a number of pressing business 'pain points' creating the need for a robust reporting solution:

- Limited visibility of company wide performance through existing reports
- Limited effective capability to recreate historical figures accurately
- The need to create a single version of information that is not open to interpretation
- There was little or no security on the existing reports so we needed a way to introduce data security for different levels of management.

The BI environment at Credit Corp is predominately powered by OLAP cubes. Data visualization tools like SAS® Web Report Studio provide some great functionality for the end users but this in turn introduces some challenges when trying to reuse that data to display key performance indicators using the BI dashboard. The purpose of this paper is to share knowledge with the audience about some of the fine details involved in an implementation of SAS® BI Server. Particular attention is paid to the variety of ways data is disseminated to the user base and also showing some techniques to better integrate OLAP cubes and dashboard functionality.

There are a lot of examples and screen shots taken from the completed reporting environment to give you a real picture of how the implementation was done and how challenges were overcome. We know the information contained within the reporting environment is directly contributing to increased efficiency and accounts being worked more thoroughly by collections staff. Some figures regarding improvements to productivity and ROI will be presented later. To protect the intellectual property contained within the reports used at Credit Corp I have changed measure names to unrelated topics.

I have split the paper into two main sections, the first relating to data and security related issues and the second focused on the reporting side of the implementation.

1. DATA AND SECURITY

During the initial requirements gathering phase of the project an important aspect that emerged was that reports had to be flexible enough that power users had the ability to navigate around a report and even change the view to find points of interest. At the same time, it had to be easy enough to use so that people with little or no training could open and understand key pieces of information that were being presented. This core requirement made OLAP cubes and multidimensional reports the natural choice as the primary report delivery mechanism. This also fit in very well with our data security requirements as OLAP cube security is defined in the SAS® Metadata Server and is applied automatically in nearly every case as opposed to trying to protect datasets that require data security rules to be applied whenever the data is used.

To make reports easy to use and understand, wherever possible they have been built with common hierarchies for navigation. The benefit being that all reports have a familiar element even though each one deals with a different subject area. The organizational structure of Credit Corp's collections group is the basis for one of the common hierarchies used in all reports and is the key element in the underlying security model.

There are 4 levels of the current structure

- Company wide (top) level
- Site level – based on geographical location (Sydney, Brisbane, …)
- Team level – each site has many collection teams (Team Moneybags, All Stars, Bandits, Pirates, Gold Diggers)
- Account manager level – Collectors are responsible for negotiating a manageable ongoing payment arrangement with our customers (debtors)

A partial organizational structure can be seen in Figure 1 below. The figure shows a sample of reporting lines for account managers or customer relationship managers (CRMs). This same structure is used for summarization when required in reports.

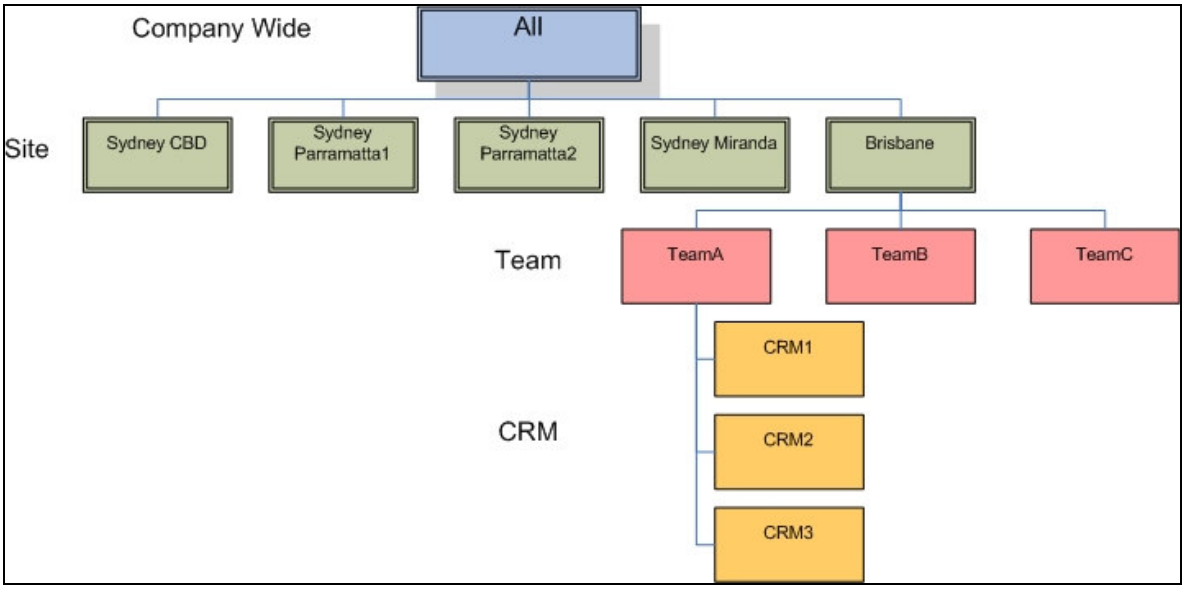


Figure 1
Partial organizational structure

The other common hierarchy is Time which allows the user to compare historical performance figures at different levels of the hierarchy such as day, week and month. Depending on the report, different components of the time hierarchy will be used. Some reports may go down to the day level whereas others only go as far as month. The deciding factor is usually the amount of storage for both the cube and the base table. For example, one report that shows balances of accounts could not go down to day level as that would require too much data to be stored so instead it stops at the month level. The following is a screen shot from SAS management console showing the levels that make up the two core dimensions that are used in every report.

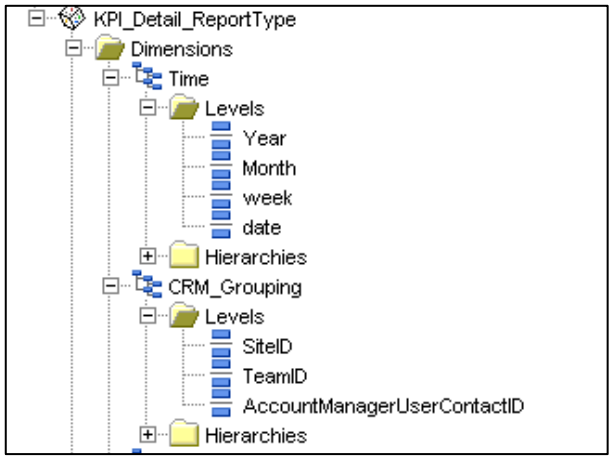
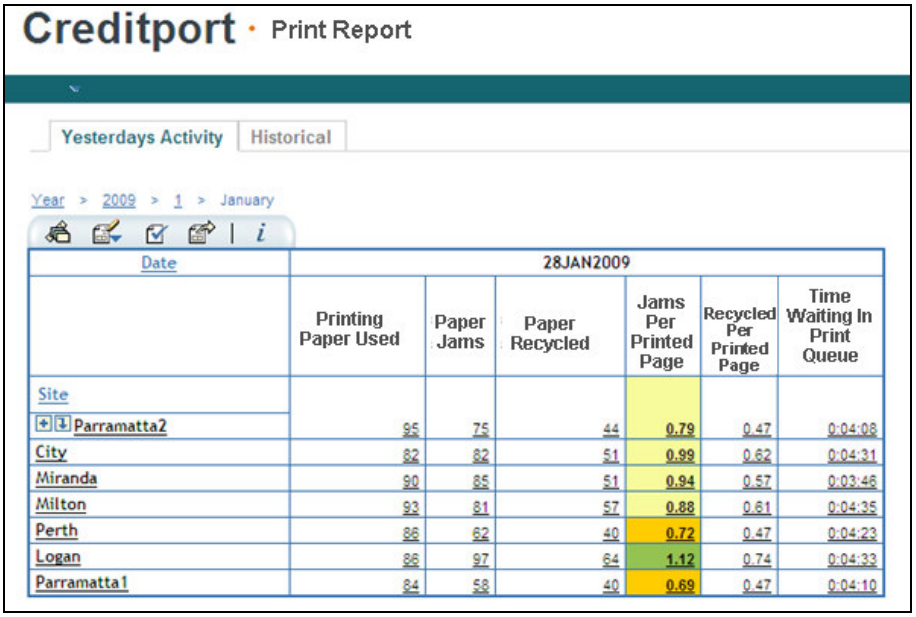


Figure 2
Common OLAP dimensions

Besides being able to navigate through a report, it was also a major requirement for data level security to be applied at every level so that a user could only see what was relevant to them, depending on their position in the management hierarchy and geographical location. For most reporting needs a user should be able to see detailed data for their group, plus everything in the hierarchy below them, plus summary information of the levels above them to better put their own team's performance into perspective. For example, using the structure in Figure 1 the team leader for TeamA can see all information relating to TeamA as well as details regarding CRM1, CRM2 and CRM3. The team leader can also see summary level information for Brisbane and summary level information on the company wide performance. They cannot see how TeamB or TeamC are performing.



Date	28JAN2009					
	Printing Paper Used	Paper Jams	Paper Recycled	Jams Per Printed Page	Recycled Per Printed Page	Time Waiting In Print Queue
Site						
Parramatta2	95	75	44	0.79	0.47	0:04:08
City	82	82	51	0.99	0.62	0:04:31
Miranda	90	85	51	0.94	0.57	0:03:46
Milton	93	81	57	0.88	0.61	0:04:35
Perth	86	62	40	0.72	0.47	0:04:23
Logan	86	97	64	1.12	0.74	0:04:33
Parramatta1	84	58	40	0.69	0.47	0:04:10

Figure 3
Web report with security applied

Figure 3 above shows an example from a web report where data security has been applied to a cube so the user has the ability to see the summary for each site and then the detailed data of the site Parramatta2, which is the one they belong to. The ability to view a lower level of detail is indicated on the report by the plus symbol and down arrow next to the site Parramatta2. Clicking on the down arrow will change the report to show all of the teams in the site and there will then be the option to drill down further to see all of the CRMs within a team. This is a variation on the scenario described previously where the user was restricted from seeing other teams at the same level as their own. This is the type of view that might be used by a site manager that allows them to compare the performance of their site to the others across the company.

This type of data level security was achieved using MDX member level permission conditions. MDX is the language that is used to query OLAP cubes. MDX stands for Multi Dimensional Expression and has a similar structure to SQL queries. For details on how to use MDX see the MDX Guide section of the SAS OLAP Server chapter in the SAS online documentation at http://support.sas.com/onlinedoc/913/ .

MDX permission conditions are applied in the metadata to a dimension for each user or group, and define what data members a user is able to access. In the Credit Corp environment as much as possible data security has been applied at the group level to utilize common MDX expressions for users with common data access requirements. The following code in Figure 4 is an example of an MDX expression which applies security to a single cube for the team leader of TeamA in Brisbane.

```
1. Ascendants([CRM_Grouping].[All CRM_Grouping].[Brisbane].[TeamA]),
2. [CRM_Grouping].[All CRM_Grouping].[Brisbane].[TeamA],
3. Descendants([CRM_Grouping].[All CRM_Grouping].[Brisbane].[TeamA])
```

Figure 4
Sample MDX code for a member level permission condition

To explain the expression I will address each of the 3 lines separately

1. Uses the MDX function 'Ascendants' to define access to the summary levels above Team A which in this case is the Brisbane site and the All level
2. Defines access to TeamA at the team level summary
3. Uses the MDX function 'Descendants' to define access to all of the detailed Account manager level results for Team A

These permission conditions need to be applied to the metadata in one of two ways. Either manually using the SAS® Management Console or in batch using the Open Metadata Interface. Using the SAS® Management Console is the easiest method to get started as it doesn't require any coding or understanding of the underlying metadata structures. But there are a number of steps that make it not viable for multiple users. To give you an idea of the effort required the following example shows how to apply member level permission conditions using SAS® Management Console.

One cube at a time, open the properties window for the dimension that you are applying security to. In this case data level security is being applied to the CRM_Grouping dimension.

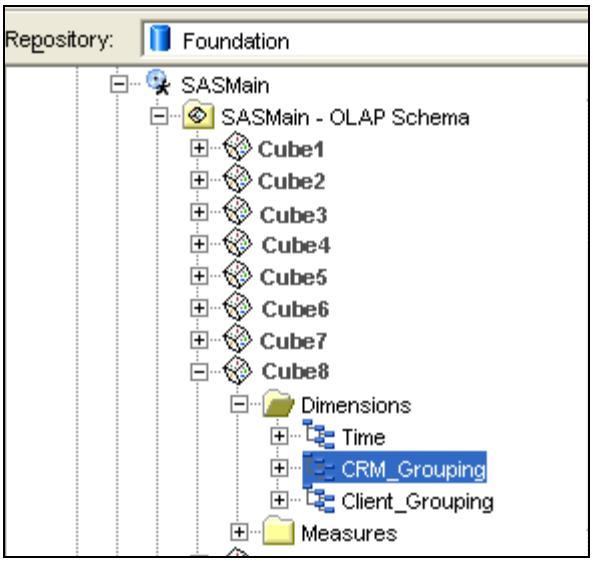


Figure 5
Applying security to an OLAP dimension in SAS® Management Console

On the Authorization tab, add the user or group and click the Grant check box for the Read permission so explicit read access is granted. You can tell explicit read access has been granted when the background for that cell is white instead of the grey which indicates implicit access. You must have explicit read permission granted for the Add/Edit

condition button to be enabled. If the metadata identity does not already have a permission condition then the button will say Add, otherwise it will say Edit.

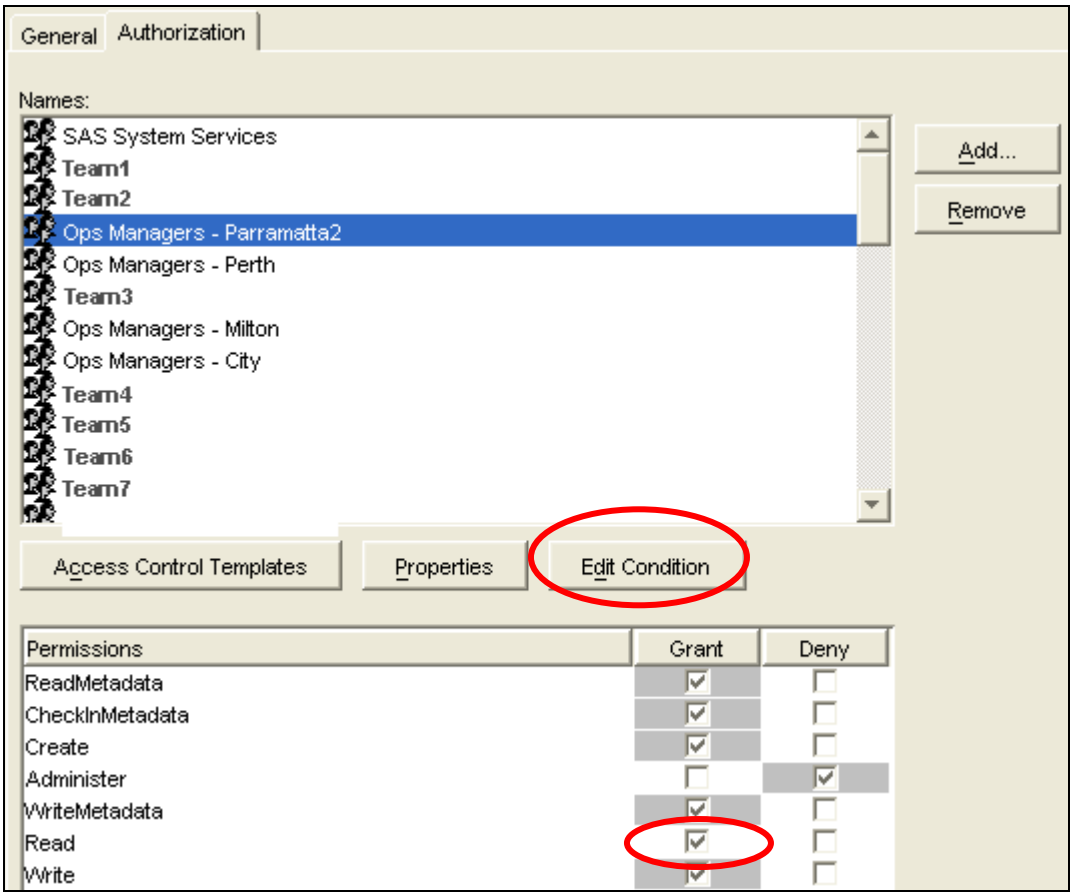


Figure 6
Setting metadata permissions to apply MDX security

Clicking on the Add\Edit condition button will open a window that will allow you to type in text. In this window you enter the MDX code to be applied to the user/group. The expression needs to result in a MDX set that is the subset of the particular dimension that you want the user to be able to see. In the case of the example in Figure 7 the returned set is the CRM_Grouping All level, the site Parramatta2 and all teams and account managers that belong to the Parramatta2 site.

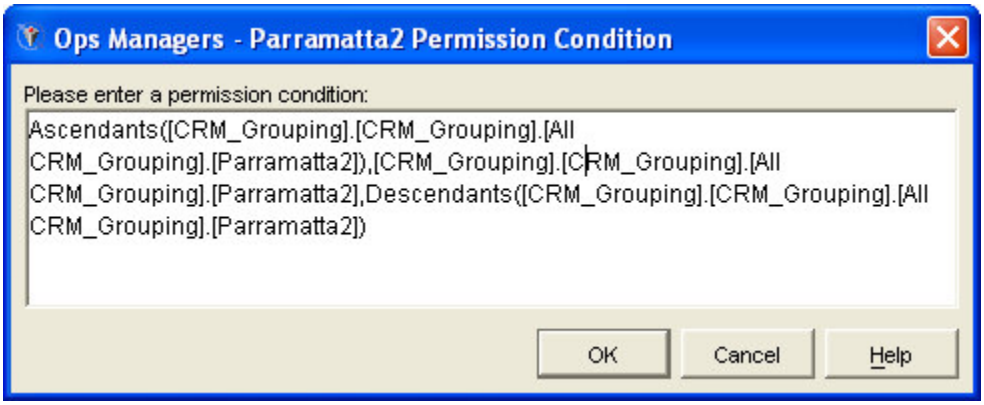


Figure 7
MDX to apply member level security for a metadata identity

The problem with this manual approach is that each group requiring security would need to have it applied to each cube. In the Credit Corp implementation there are currently 46 groups and ten cubes. That would mean a lot of point and clicking. Instead, permission conditions get applied in a batch as part of the automated metadata synchronization process. This process creates and deletes metadata users and groups to match changes in the business. The metadata update process makes use of bulk load macros like %mduextr, %mduimpc, %mducmp, %mduchgv and %mduchgl to extract the current metadata, compare it to the organizational structure in the transaction system and apply updates. Documentation about how to apply this type of metadata update can be found in the appendix "Bulk-Load Processes for Identity Management" of the SAS 9.1.3 Intelligence Platform Security Administration Guide.

Even though most of our reports are driven by OLAP cubes there are a few that use SAS datasets or direct queries to the live transaction database. To maintain the same security policy for this type of data we used a combination of information maps with a security association table and stored processes to apply a where clause to a dataset before results are displayed to the user. Stored processes use the _metauser macro variable to dynamically subset each summary table accessed by a user in Creditport.

```
PROC SQL noprint;
 SELECT DISTINCT
       input(HR_Table.userID,4.)) into : userID
 FROM HR_Table AS HR_Table
WHERE HR_Table.userID NOT IS MISSING and HR_Table.WindowsID =
trim(left("%scan(&_metauser,1,@)"));
QUIT;
```

Figure 8
Example of using _metauser in a stored process

The code in Figure 8 above gets the username component from the _metauser macro variable which corresponds to the Windows user ID and then subsets the table to get the transactional system user ID.

2. REPORTING ENVIRONMENT

The SAS Information Delivery Portal is the main user interface for Credit Corp's BI solution, Creditport. Currently there are approximately 90 web users from all levels of management across the business with plans to add access for all account managers (350) this year. Each user sees only what is applicable to their role by implementing role based security to control which pages each user can see with the three main roles being senior management (company wide view), site manager (geographic site view) and team leader (single team view). These roles are maintained by the same metadata synchronization program that updates OLAP cube MDX permission conditions. The main components used to convey information in the portal are dashboards, web report studio reports, and stored processes.

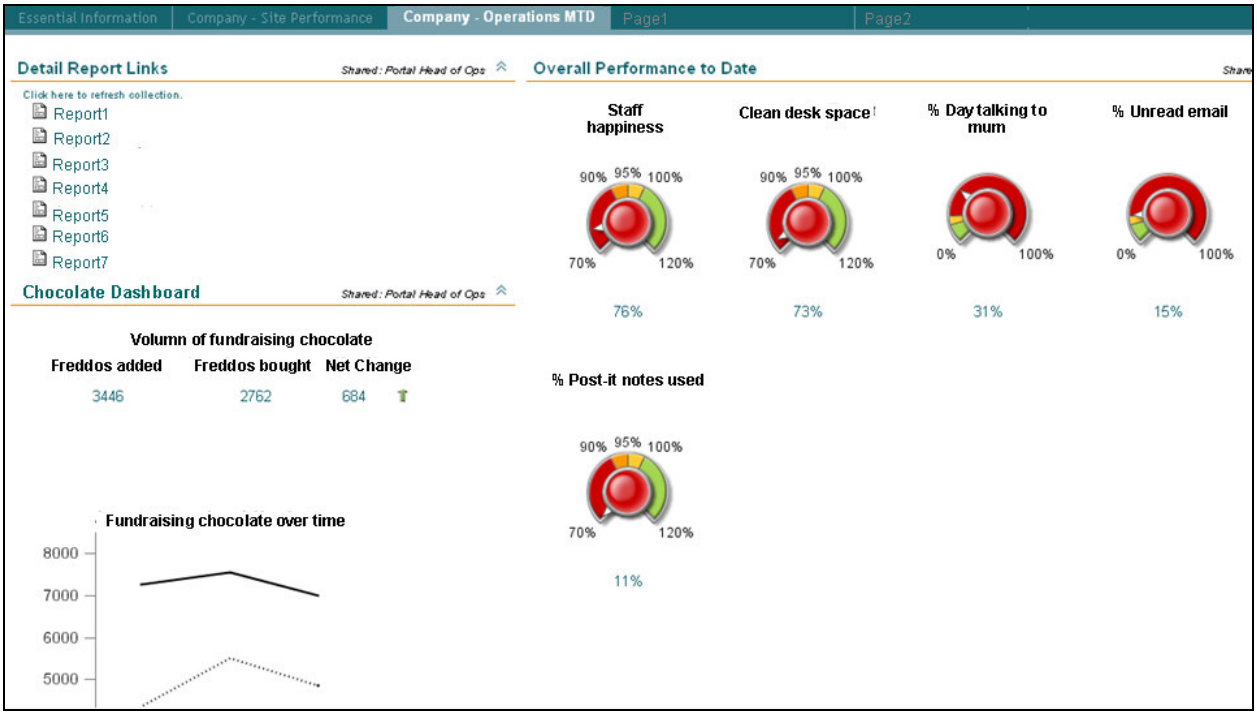


Figure 9
Senior manager view of Creditport

Figure 9 above shows the portal view of a member of senior management. Users in this group get the welcome page which is visible to all users plus a 'summary by site' page and the 'overall performance month to date page' which contains the month to date dashboards. Then, depending on what other roles the user has assigned, they may be able to see one or more additional pages.

Being a member of a role gives access to a common set of reports in the Detail Report Links section but it is possible to add more reports into that list that can only be seen by a smaller group of users. Using the BI Manager, authorization can be changed for individual reports by denying ReadMetadata (RM) access to the wider group and granting RM access for smaller groups or single users. Figure 10 shows where to find a web report definition in SAS® Management Console. In the default setup web report can be found in Report Studio -> Shared -> Reports folder of the BI Manager.

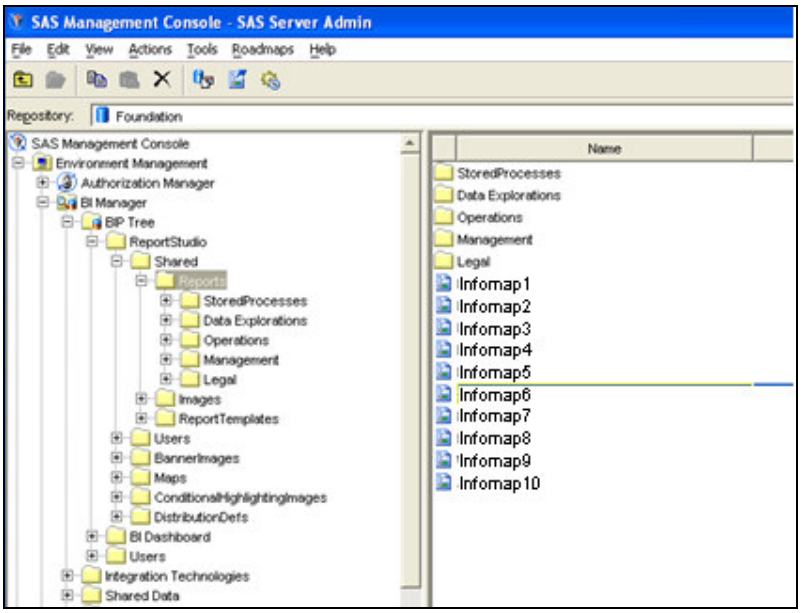


Figure 10
Selecting a shared SAS® Web Report Studio report in the BI Manager

After opening the properties window for a selected report, checking the deny ReadMetadata property will hide the report from the wider audience.

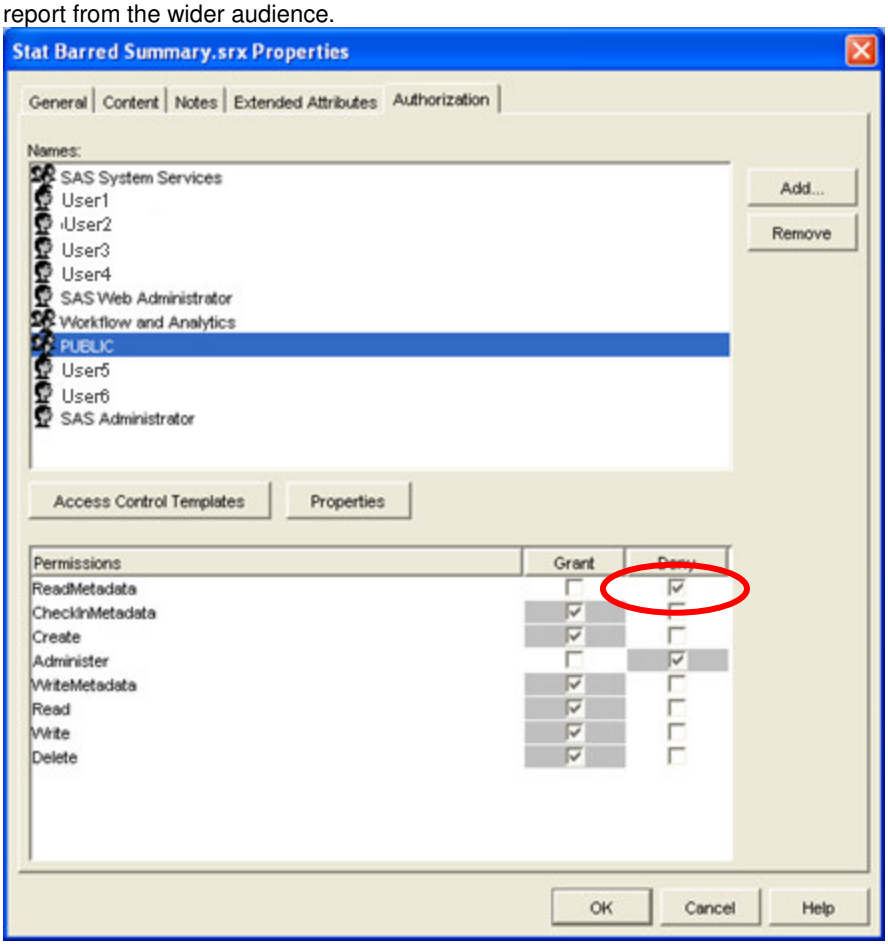


Figure 11
Changing permissions for a report

2.1 USING THE BI DASHBOARD FRAMEWORK

The performance dashboard plays a crucial role in giving users a high level view of how performance is going against the main indicators. The screen shots in Figure 12 and Figure 13 show examples of some of the dashboard types that have been created using the BI Dashboard framework. The first is a series of bar charts that are used by upper management to get a comparison of the same measure across multiple sites.

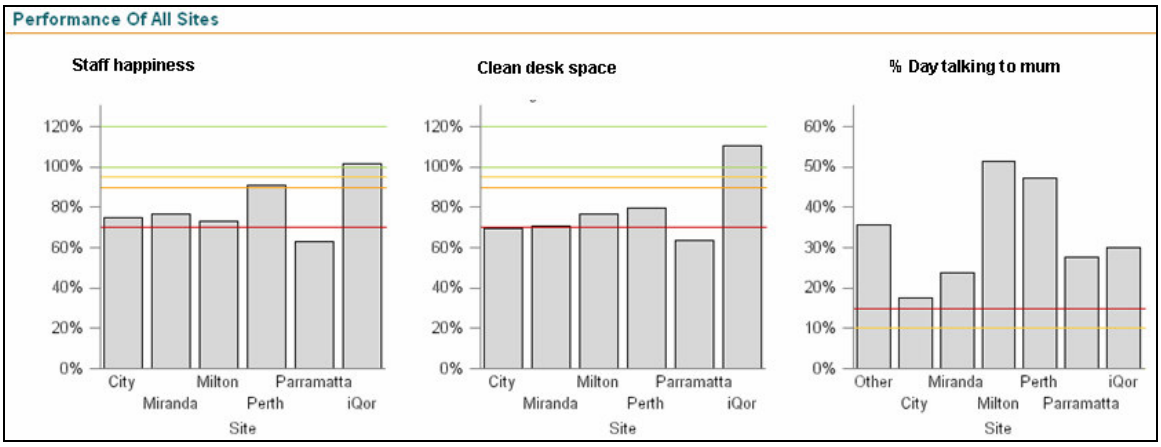


Figure 12
Site comparision dashboard

The second is showing the same measures but for just one team. This type of view would be used by a team leader.

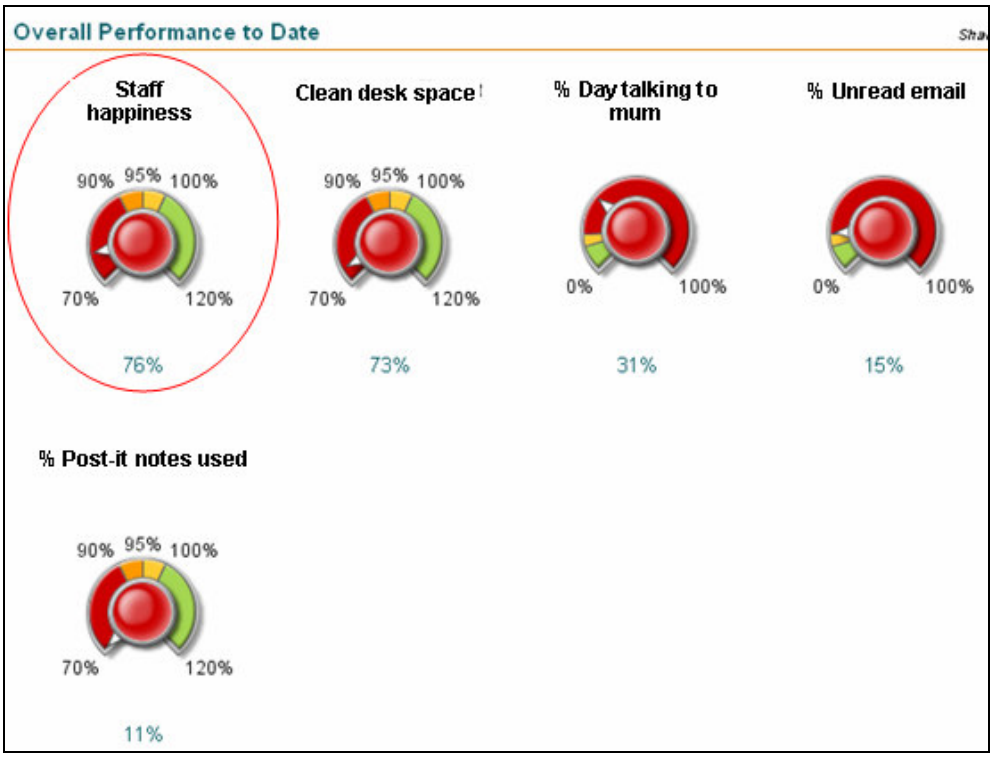


Figure 13
Individual team dashboard

The original design was for all reporting levels to utilize the dashboard framework to display relevant figures from OLAP cubes but this was changed part way through so only the company level is now built using the dashboard framework. The main reasons for this are

- Scalability and maintenance
- Difficulty in displaying OLAP hierarchies

Early in the implementation phase it became clear that scalability and maintenance would be an issue as there was simply too much overhead involved in using the BI Dashboard user interface to create dashboards at all levels. To illustrate the issues, the following series of screen shots show the setup process for a single dashboard indicator.

First a data model must be defined to connect to a data source. A few of the choices for the data source type are JDBC query and SAS Information Map. All of the data models in the Credit Corp implementation use OLAP Information Maps as the data source. As part of the data model definition you must select what classification variables and measures will be available in the data model.

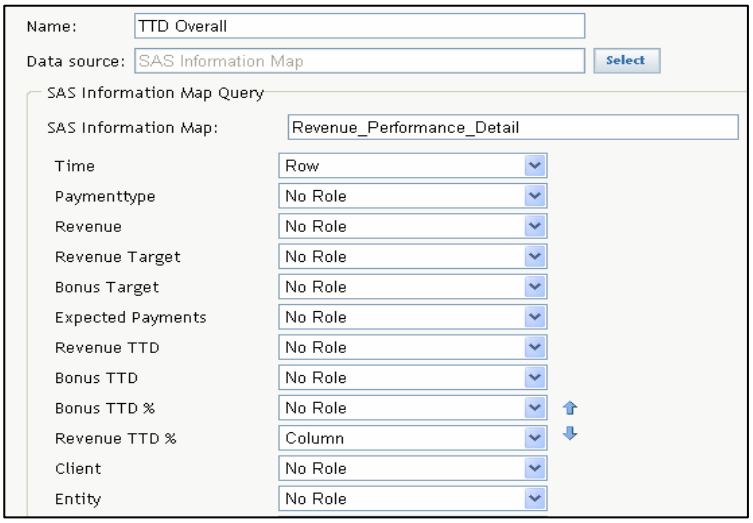


Figure 14
Defining a data model

Then the indicator needs to be defined to display data from a data model. It is possible to use one data model for multiple indicators, but in the Credit Corp case the purpose of the dashboard was to show performance of different parts of the operation - which meant, in most cases, different data sources. When defining each indicator as well as setting up the type and metrics to be displayed, you have the opportunity to define other properties like indicator size and HTML links to view more detailed reports.

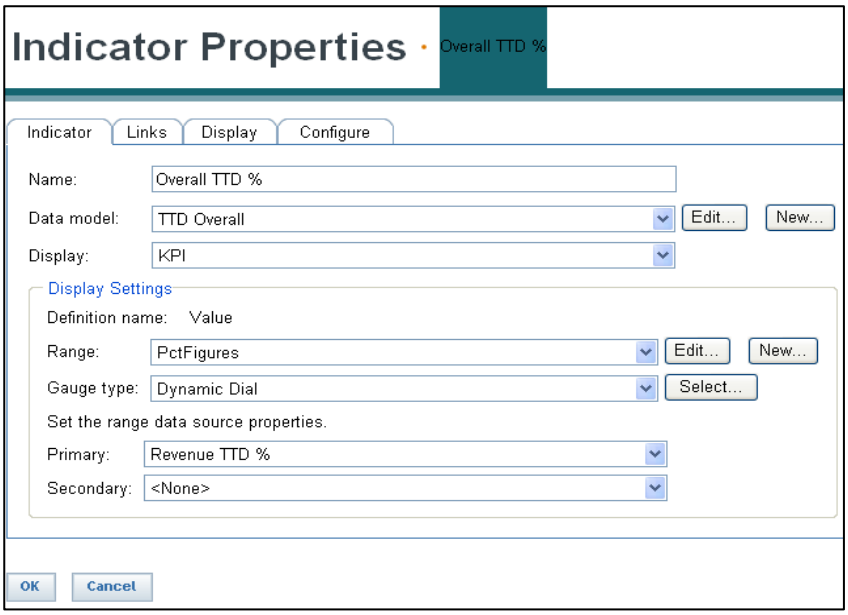


Figure 15
Setting up properties of an indicator

The last step is to define a range - if a suitable one has not been defined already. Ranges are a form of traffic lighting used to highlight poor, fair and good performance. Once the indicator is finished it can be added to one or more dashboards.

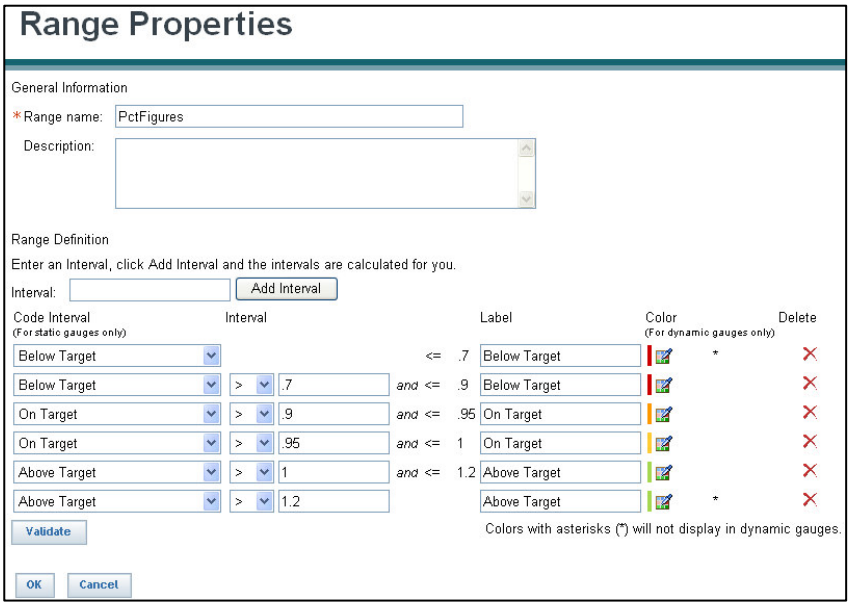


Figure 16
Creating a range for indicators

During the implementation we ran into trouble when trying to set up data models using OLAP cubes as the data source. OLAP hierarchies contain one or more levels (variables) that allow users to see the data summarized at different levels of granularity. But I found that the data model for the dashboard does not let you choose which level of the hierarchy to display. Looking at Figure 17 below, the two highlighted areas are the hierarchies of the OLAP cube. By selecting these as part of the data model, the highest level values will be available to the indicator. For example, in the CRM hierarchy only Site would be displayed. The value of Team or Account Manager could not be shown. The way to work around this would be to create custom data items in the information map that would select all Teams or all Account Managers but this is a little cumbersome.

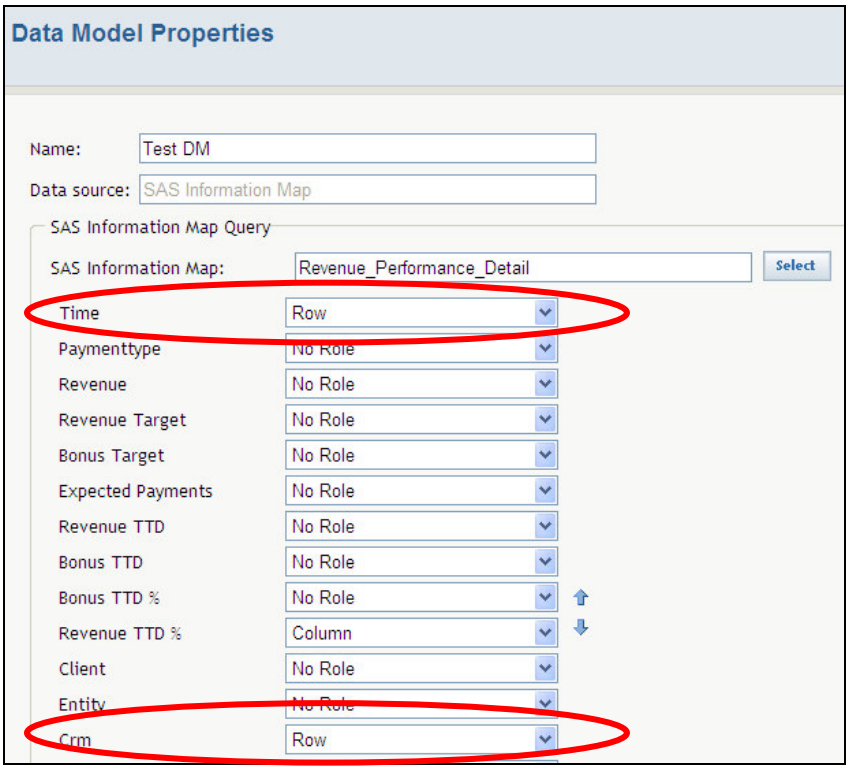


Figure 17
Selecting a hierarchy on the data model

Currently there are 11 dashboard indicators being used at all 3 levels of reporting. Even though the same metrics are being shown, in the case of this implementation 8 data models were required for the 11 indicators at every level of reporting. This means 33 dashboard indicators and 24 data models would need to be created and maintained.

The other issue with using OLAP as a data source was applying filters. Filters defined as part of an information map will be applied when retrieving data through a BI Dashboard data model, but the catch is that it always applies all of the filters that are defined in the information map. This becomes a problem if you want to use an information map to meet more than one need. For example in the Credit Corp environment I created a filter that produces a rolling three month subset but I also wanted to put in a filter to have the option to be able to see open accounts only. The end result was that the data returned was always for a three month rolling window showing open accounts only. The way around this issue is to create multiple information maps, one for each purpose.

2.2 AN ALTERNATIVE APPROACH

To overcome the issues relating to the BI Dashboard we decided to develop an alternative approach. We continued to use the BI Dashboard when reporting at the company wide level as everything was already being summarized to the highest level. Plus it is relatively easy to get new indicators setup and running at this level and most new reporting requirements would come from senior management. This meant a fast turn around time on new requests. But for people further down the organizational structure we needed some way to dynamically build a dashboard based on the permissions of the user to avoid having to setup and maintain all of the components required for the BI Dashboard.

As an alternative we developed a stored process that determines the role of the user by once again using the _metauser automatic macro variable to determine the identity of the user calling the stored process and then produce a dashboard with the indicators summarized to the appropriate level. If the user is a site manager then the indicators needed to be summarized to a site level and if they are a team leader then it should be summarized to the team level.

As all of the data had already been built into OLAP cubes we decided to leverage that instead of creating more datasets as part of the work around. The SQL pass through facility could be used to query an OLAP cube as long as appropriate MDX can be generated based on the users' security permissions. The key to making this process successful is identifying that the user is a site manager or team leader and then generating MDX for the SQL query to pick up the right level of information from the cube. Essentially, dynamically determining which level of a hierarchy in the cube needs to be displayed based on the user's role. The task of producing a dashboard via a stored process follows this general set of steps.

1. Get the username of the person using the portal via _metauser
   This can be achieved by using code similar to that in Figure 8.

2. Using security tables, determine the role and location of the user.
   As part of our batch process that synchronizes metadata users with the Credit Corp transaction database, we produce a dataset that groups users into their level of access and location. This security table makes it easy to quickly lookup what a user's access should be. A basic table similar to the one below is sufficient to take the user's portal login (WindowsUID) and use that to determine their team and site and whether they are a site manager or not.

UserID	TeamID	SiteID	SiteMgrID	WindowsUID

3. Generate the MDX to pull out the appropriate level of information.
   Based on the information from step 1 and 2 you can determine what the MDX should look like to access data for a particular time period and team/site. For example the MDX to summarize to a single month is

```
[Time].[All Time].[2009].[1].[January]
```

   The MDX to identify a single team is

```
[CRM].[All CRM].[Brisbane].[TeamA]
```

   These pieces of MDX will be used in the SQL in step 4.

4. Using PROC SQL pass-through, query OLAP cubes for each indicator required.
   The following example code in Figure 18 shows the type of query used to extract data out for one dashboard indicator.

```
PROC SQL;
CONNECT TO olap (host=SERVER port=5451 user="USERNAME" pass="PASSWORD");
       CREATE TABLE revenue_indicator AS
       SELECT *
       FROM connection to olap
       (
       SELECT { CrossJoin ( {[Time].[All Time].[2009].[1].[January]},
       {[Measures].[Revenue], [Measures].[Revenue_TTD]})
       } ON COLUMNS ,
       non empty {[CRM].[All CRM].[Brisbane].[TeamA]}
       ON ROWS
       FROM [Revenue_Performance]
       );
DISCONNECT FROM olap;
QUIT;
```

Figure 18
Querying an OLAP cube through PROC SQL

This is a very fast way to pull data from an OLAP cube providing it is optimized for this type of task. As the result of the query above will be a single row, as long as you have the appropriate aggregation setup up in the cube, the SAS® OLAP server will not have to perform any summarization. If, on the other hand, your cube has a large number of cells and no aggregations to speed things up then summarizing to a high level could take some time. Querying an OLAP cube in this manner however raises another problem regarding security. As with querying a regular database, the PROC SQL pass-through requires a connection be made to the OLAP server using a username and password. So a system account is used to connect in this situation. The issue is any member level security that is in place is dependant on the user so as soon as a system account is used that security is not applied.

The result of the above query is a single row that contains the two measures Revenue and Revenue Target to Date for the month of January 2009 and for TeamA which is part of the Brisbane site.

5. Create a dashboard layout including each indicator.
   The dashboard itself is produced using a data NULL step to create a custom HTML table layout that is streamed back to the portal page. Figure 19 contains an example of how the layout of a dashboard on a page is controlled. In this case two measure dials with names and values will be displayed.

```
put '<table>';
put "<tr><td align='center'><b>Revenue % TTD</b><p> &teamname.</td><td
align='center'><b>Predicted % Target</b><p> &teamname.</td></tr>";
put "<tr><td align='center'>";

%generate_TTD_dynamic_dial(value=&ttd_pct., alttext=Revenue % TTD &teamname.:
&ttd_pct_display.,
reportlink=Report%2Bomi%3A%2F%2FFoundation%2Freposname%3DFoundation%2FTransformation%3
Bid%3DA5CSLH82.AY000A38);

put '</td><td>';
%generate_TTD_dynamic_dial(value=&predict_pct., alttext=Predicted % Target &teamname.:
&predict_pct_display.,
reportlink=Report%2Bomi%3A%2F%2FFoundation%2Freposname%3DFoundation%2FTransformation%3
Bid%3DA5CSLH82.AY000A3R);

put '</td></tr>';
put "<tr><td align='center'>&ttd_pct_display.</td><td align='center'>
&predict_pct_display.</td></tr></table>";
```

Figure 19
Creating a dashboard layout

The code above contains calls to the user written macro generate_TTD_dynamic_dial, that generates the appropriate HTML to display a dashboard indicator on the screen. By passing the value to be displayed as well as other information like the URL of a detailed report we are able to create an indicator with traffic lighting to highlight performance as well as drill down to a SAS® Web Report Studio report to get more detailed information.

Up to this point all that has been done is preparing data for the indicators and deciding how they will be organized on the page. Actually producing a visual representation of the indicator has not been covered yet. The BI Dashboard framework uses a model/viewer approach, meaning that the preparation of the data is performed by one component and rendering that into an indicator is done by another. While working with the BI Dashboard we found that we were able to utilize the application that displays the indicators by calling a URL and passing the appropriate parameters.

The output of the generate_TTD_dynamic_dial macro is simply a HTML anchor tag (<a>) with an associated image. The anchor links to the detailed report and the image is actually a URL which calls the BI Dashboard application that produces the indicator as shown below in Figure 26. All of the indicators are the dynamic dial type which behaves differently to all of the static versions of indicators. Static indicators use a fixed image so the arrow on the dial points to the general range of values but not the exact number. Dynamic dials, on the other hand, point to the exact number which made them a better choice for our purposes.

Figure 25 below contains part of the URL created by the macro to produce the indicator, split into two parts. The first part is the link to the detail report as the HREF, which is stored in the macro variable &reportlink. It also contains the start of the image definition which is the indicator graphic by calling the BIDashboard web application. The final name/value pair in the first section is the actual value to be shown by the indicator. The second half of the example is a partial definition of the indicator range. Each segment requires a lower and upper boundary as well as the color of the section.

```
<a href= "http://BIServer/Portal/syndication.do?com.sas.portal.ItemId=&reportlink
target=_top> <img border="0" width="150" height="110" src=
"BIDashboard/pdv?_pdv_handler=kpi_visual_handler&_pdv_program=full_color_tach&&_pdv_wi
dth=150&_pdv_height=110&_pdv_display_scale=true&_pdv_ds1_scale_format=PERCENTN&_pdv_or
ientation=h&_pdv_enhanced=true&_pdv_ghost_inactive_intervals=false&_pdv_ds1_value=.91
```

```
&&_pdv_rs1_id=belowTarget&_pdv_rs1_data_color=CC0000&_pdv_rs1_lower_bound=0.701&_pdv_r
s1_upper_bound=0.901&_pdv_rs1_lower_bound_operator=LT&_pdv_rs1_upper_bound_operator=LE
.
.
.
&&_pdv_rs1_id=aboveTarget&_pdv_rs1_data_color=A5D753&_pdv_rs1_lower_bound=1.002&_pdv_r
s1_upper_bound=1.202&_pdv_rs1_lower_bound_operator=LT&_pdv_rs1_upper_bound_operator=LE
.
&_pdv_rs1_lower_bound_operator=LT&_pdv_rs1_upper_bound_operator=LE&"
alt="&alttext"/></a>
```

Figure 20
URL to create a dashboard indicator

When run in a stored process, the result is what you see in Figure 21. Building dashboards in this way utilizes the look and feel of the BI Dashboard while at the same time makes better use of OLAP cubes by making dashboard creation more dynamic.

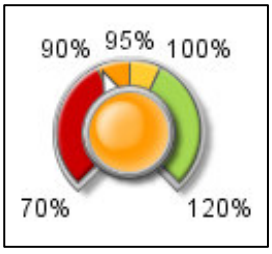


Figure 21
Dashoboard indicator produced by a stored process

The finished product looks the same as the original. As an example, Figure 22 shows a site manager's performance page using two stored process dashboards.

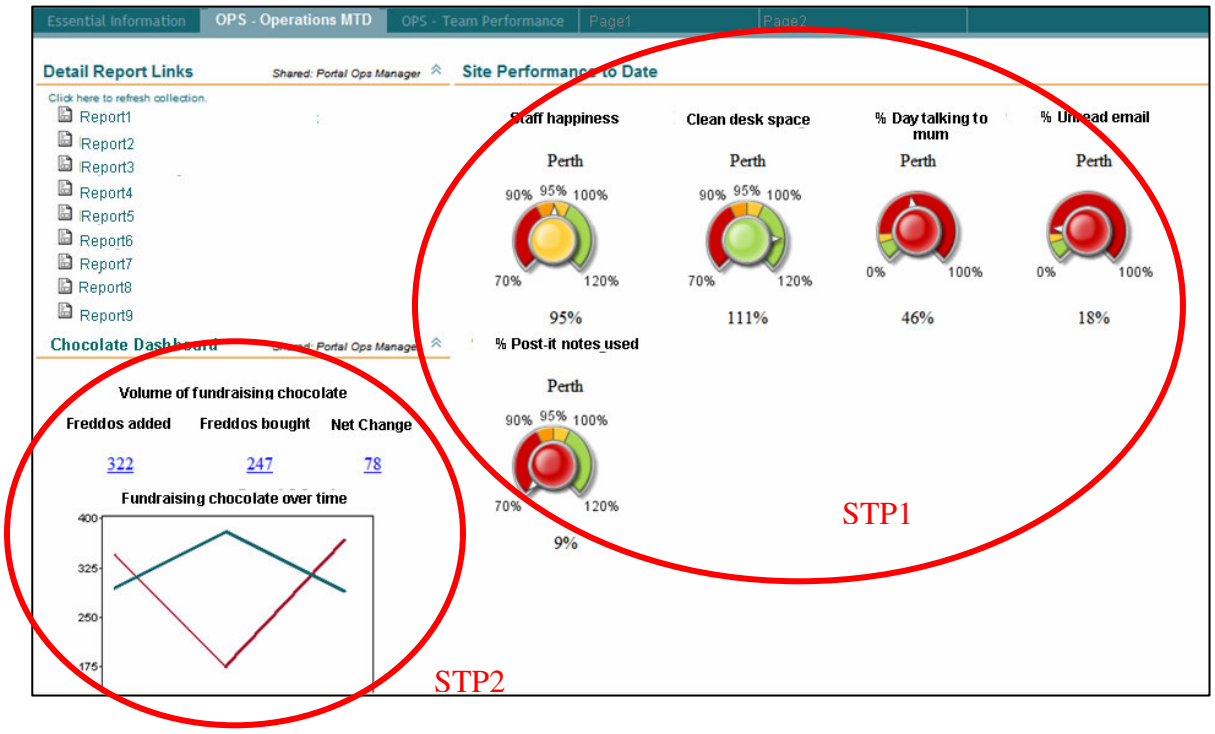


Figure 22
Completed dashboard created with a stored process

When a user wants to investigate a particular measure in more detail, there are a series of detailed reports to help them do so. Every dashboard indicator has a report attached to it, so to investigate a particular aspect of the business to see why it is performing the way it is, a user can click on an indicator to view a report Alternatively they can select one from a list that is built with a collection portlet.

2.3 DETAILED REPORTS

Web reports in the Credit Corp environment come in three different types. The majority are OLAP reports produced in Web Report Studio that allow the user to navigate through the data to see different aspects in more detail or view historical performance. There are also some WRS reports based on data sets and finally when our reporting requirements cannot be met using the BI tools we build a report as a stored process and surface it using the Stored Process Web Application.

Most reports are time sensitive in some way and Creditport is no different. Creating Web Report Studio OLAP reports can be a little tricky when it comes to setting a time period. In a normal SAS program it is easy to determine today's date using functions but when building calculated members with MDX you only have access to a limited number of Base SAS functions. It is possible to build relative date expressions but it is usually a complex task and is dependant on how your Time hierarchy is defined. So instead the conventional method is to define a filter as part of the web report using the Web Report Studio interface. The options for doing this are to select a category value from the time dimension, or to create a relative time filter. Each method has its pros and cons.

Selecting a category value will show each value in the dimension starting at the highest level and allowing the user to expand out to the next level, eventually selecting one or more time periods to filter the report by. The main problem with this method is that the filter will not automatically change as time rolls on so someone will have to update this filter regularly. The advantage is that it is simple for users to make changes to this filter when using Web Report Viewer and they can select time periods that are not continuous. For example a user may want to compare last month to the same month for last year. This would involve simply selecting both of those months and they would be included in the report.

A relative time filter allows you to select the next N periods from the first date or a selected point in time in the cube; or previous N periods from the last date or a selected point in time in the cube. The obvious advantage of this method is that by using the start date or end date of the cube, the filter will always move with time so the user does not have

to manually update it. In the normal case you might create a relative time filter to show the last month from the ending period. However if your cube has any future data in it like forecasted revenue then the end period cannot be used as it will be well into the future.

A third option which is used sometimes in Creditport is to include the filter date into a level of the cube and then use that value to subset the data. The date is determined as part of a SAS program that builds the base table for the cube, and because we have access to the full set of SAS functions it is easy to programmatically set the date. Figure 23 shows an example of what the values in the date columns should look like to be used to subset a cube. There are two different date columns because the time filters differ depending on the report.

AccountID	ReceiptingDate	LastWorkingDate
999999	Time].[Time].[All Time].[2009].[January].[14JAN09	Time].[Time].[All Time].[2009].[January].[15JAN09

Figure 23
Variables containing dates to be used to subset a cube

The way that it is used is to create a named set based on the date and then use that named set when creating calculated measures. Figure 24 below shows a named set being defined and then used in a measure as part of the PROC OLAP code. The expression to create the member Today uses the value stored in lastWorkingDay, turns it into a string, and then creates a set from the result.

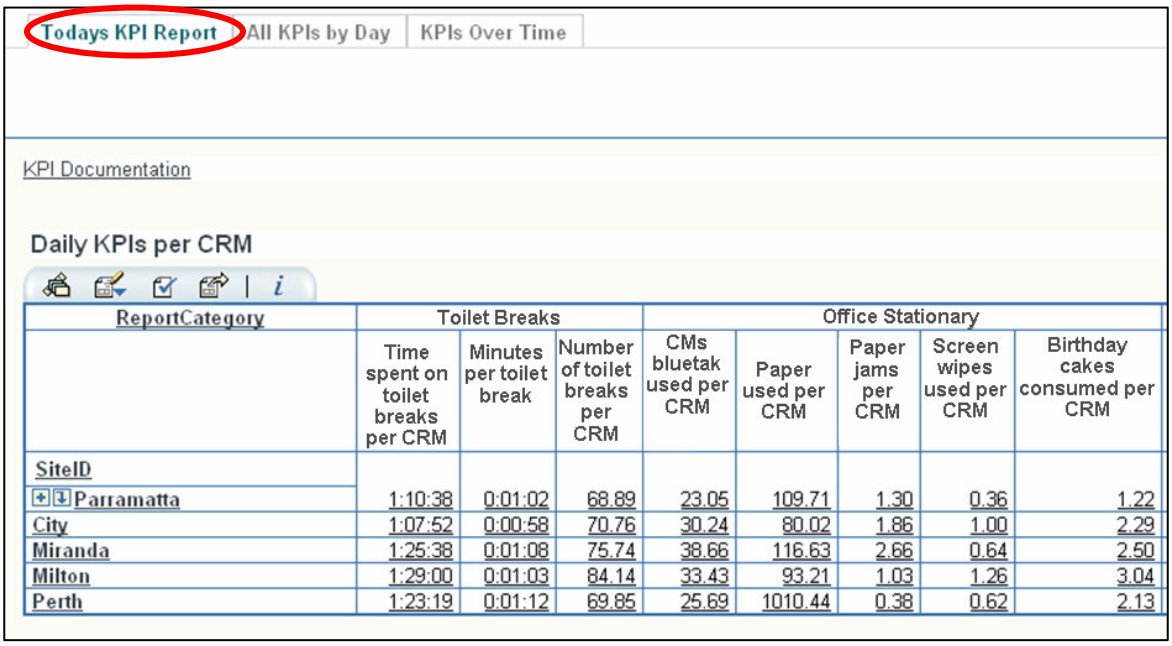
```
DEFINE
       SET '[KPI].[Today]' AS
       'strtoset(SAS!substr(membertostr([LastWorkingDay].[All
LastWorkingDay].lastchild),39))',
       MEMBER '[KPI].[Measures].[Daily Chocolate Consumption Total]' AS
       'SUM([Today],[Measures].[ChocolateConsumption]), FORMAT_STRING="12."'
;.
```

Figure 24
Creating a named set

A good example of how to present time in different ways is the Creditport KPI report. It shows activity for all account managers across the business summarized up to the appropriate level looking at different time periods.

- Time relative to the last business day
- A single day view of the users choice
- A comparison of multiple time periods

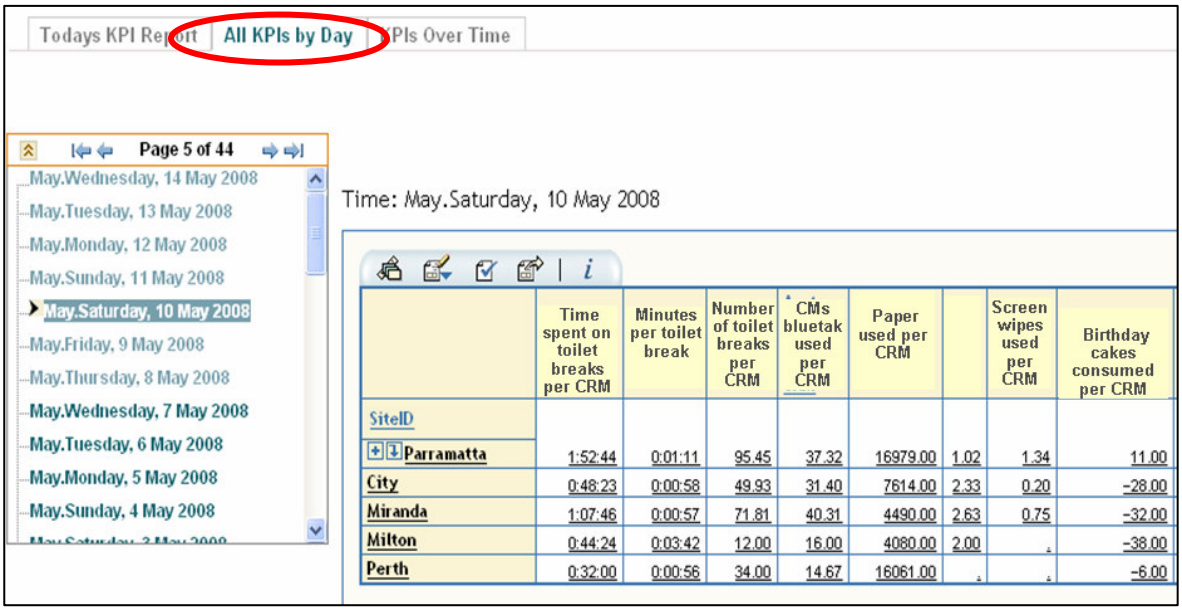
The first page doesn't show a time period on the page at all. The measures on this page are calculated members like the example in Figure 24 using the named set to determine the date to display. In this case the date is calculated as the last week day not including public holidays.



ReportCategory	Toilet Breaks			Office Stationary				
	Time spent on toilet breaks per CRM	Minutes per toilet break	Number of toilet breaks per CRM	CMs bluetak used per CRM	Paper used per CRM	Paper jams per CRM	Screen wipes used per CRM	Birthday cakes consumed per CRM
SiteID								
Parramatta	1:10:38	0:01:02	68.89	23.05	109.71	1.30	0.36	1.22
City	1:07:52	0:00:58	70.76	30.24	80.02	1.86	1.00	2.29
Miranda	1:25:38	0:01:08	75.74	38.66	116.63	2.66	0.64	2.50
Milton	1:29:00	0:01:03	84.14	33.43	93.21	1.03	1.26	3.04
Perth	1:23:19	0:01:12	69.85	25.69	1010.44	0.38	0.62	2.13

Figure 25
Report showing data for the last working day

The second page shows one day at a time and allows you to choose a day from the group break list box. This allows team leaders to look at a day in the past as well as look at work that occurred on the weekend, which is not shown in the default view.



	Time spent on toilet breaks per CRM	Minutes per toilet break	Number of toilet breaks per CRM	CMs bluetak used per CRM	Paper used per CRM		Screen wipes used per CRM	Birthday cakes consumed per CRM
SiteID								
Parramatta	1:52:44	0:01:11	95.45	37.32	16979.00	1.02	1.34	11.00
City	0:48:23	0:00:58	49.93	31.40	7614.00	2.33	0.20	-28.00
Miranda	1:07:46	0:00:57	71.81	40.31	4490.00	2.63	0.75	-32.00
Milton	0:44:24	0:03:42	12.00	16.00	4080.00	2.00	.	-38.00
Perth	0:32:00	0:00:56	34.00	14.67	16061.00	.	.	-6.00

Figure 26
Report with option to select a previous day

The purpose of the final page is to perform analysis of a particular measure over time. The user can select one or more measures by adding them into the columns on the table view and then navigate through the report to the desired time series. In this example you can see the improvement of the amount of cake consumed per CRM over the five weeks from 30th March to 27th April.

Year > 2008 > April

Week	Week beginning 30MAR08	Week beginning 06APR08	Week beginning 13APR08	Week beginning 20APR08	Week beginning 27APR08
	Cake per CRM	Cake per CRM	Cake per CRM	Cake per CRM	Cake per CRM
SiteID					
Parramatta	0.42	1.64	1.75	1.90	2.06
Other	.	.	.	.	3.00
City	0.06	1.31	1.43	1.33	1.23
Miranda	2.59	3.03	3.43	3.91	2.97
Milton	0.05	1.28	1.57	1.35	1.05
Perth	1.48	1.88	1.72	1.22	1.06

Figure 27
Report with historical comparision

One of the benefits of using an OLAP table in a WRS report is the ability to reach through to detail data. The term 'reach through' refers to being able to click on a summary number and see the records in the underlying base table (or other nominated table) that make up that summary number. We take advantage of this feature by using a WRS report to identify some sort of behavior across the portfolio, and then the user can extract the detail table via reach through and use that information to produce a task list or run a campaign. As long as the base table contains the right variables to match rows to the OLAP cell selected, other pieces of information can be added to the table as well. For example we could add basic contact information for an account and possibly a payment history to enhance what is already in the cube. It is a basic form of self service without the users having to have in-depth knowledge of table structures or be highly trained in using WRS. The screen shot in Figure 28 shows detail records that contribute to summary numbers in an OLAP report. From here the user can view the data as is or export it into Microsoft Excel format. Values shown here have been changed for privacy reasons.

View Detail - Microsoft Internet Explorer

Export Close Window Help

	reportGroup	date	AccountManagerUserContactID	AccountID	actionsRun
1	Group1	08MAY08	John Smith	111111	ABCD
2	Group1	08MAY08	John Smith	222222	ABCD
3	Group1	08MAY08	John Smith	333333	ABCD
4	Group1	08MAY08	John Smith	444444	ABCD
5	Group1	08MAY08	John Smith	555555	ABCD
6	Group1	08MAY08	John Smith	666666	ABCD
7	Group1	08MAY08	John Smith	777777	ABCD
8	Group1	08MAY08	John Smith	888888	ABCD
9	Group1	08MAY08	John Smith	99999	ABCD
10	Group1	08MAY08	John Smith	000000	ABCD

Figure 28
Reach through to detail

CONCLUSION

Over a 6 month period the first implementation phase of the Credit Corp SAS® Enterprise BI Server laid the ground work for the reporting environment needed. Now new reports are continually added as new requirements arise. This has lead to a big push in productivity reporting as well as gaining a better understanding of the make-up of the Credit Corp portfolio. Creditport has been and continues to be an effective tool for all levels of management, enabling managers to focus their attention on the areas of the business that relate directly to them.

Through using dashboards and KPI reporting, Credit Corp is able to focus on setting goals to improve those activities that drive revenue growth. Creditport is also used to coach Customer Relationship Managers, to focus on areas requiring improvement and to reward improvements in performance.

Within the first three months of Creditport KPI Reporting, the following productivity improvements have occurred:

- Talk Time per Customer Relationship Manager increased by 12%
- Debtor locate rate has increased by 30%
- Daily Revenue per Customer Relationship Manager increased by 17%

The productivity increases that were seen in the first three months were very significant to the business. The ongoing use of Creditport as a management tool means that these levels of activity have been maintained for the last 12 months.

As mentioned previously the fast changing landscape of the credit industry means that Credit Corp has had to become more focused on driving higher returns from an aging debt portfolio rather than collections from regular periodic purchases. Recent reporting additions in the last 6 months allow management to not only monitor CRM activity but focus that activity on particular subsets of our portfolio and monitor the revenue impact. This means we can now pinpoint where we want to put our time and effort, working smarter not harder.

As with everything in the SAS world there is always more than one way to approach a solution and I hope the Credit Corp experience has given you some ideas for your own implementation.

REFERENCES

SAS OLAP Server: MDX Guide in the SAS OnlineDoc® documentation
http://support.sas.com/onlinedoc/913/docMainpage.jsp

SAS® BI Dashboard 3.1 User's Guide Second Edition
http://support.sas.com/documentation/onlinedoc/91pdf/sasdoc_913/bidash_ug_10285.pdf

SAS® 9.1.3 Intelligence Platform Security Administration Guide Second Edition
http://support.sas.com/documentation/configuration/bisecag.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Gerard Quinn
Enterprise: Credit Corp Group
Address: 11/10 Barrack St
City, State ZIP: Sydney, NSW 2000
Country: Australia
Work Phone: +61 2 9347 3664
E-mail: gquinn@creditcorp.com.au