



Gold
Microsoft Partner



Solutions provider of enterprise-wide business analysis and planning systems...

TM1Connect White Paper

Understanding TM1Connect Performance

Table of Contents

Table of Contents	2
Overview	3
TM1 Query Handling	3
Performing the query using a relational database	3
Performing the query using an OLAP database.....	4
Comparing the work in a complex query between OLAP and relational databases	5
So why use TM1 then?	6
What does this all mean and how does it relate to TM1Connect?	6
TM1Connect Query Handling.....	7
How TM1Connect resolves the SELECT clause	9
How TM1Connect resolves WHERE clauses	9
How TM1Connect resolves ORDER BY and GROUP BY clauses	10
TM1Connect does not support JOIN clauses.....	10
Dimension Caching.....	10
View Caching	12
Other Performance Considerations.....	13
Use TM1 subsets or MDX for large dimensions.....	13
Create subject specific views rather than an all-encompassing detail view	13

Overview

This document details the performance considerations when connecting TM1 to other applications using TM1Connect. The intent of this document is to provide a solid background on how TM1 and TM1Connect interoperate and how to properly structure your environment to achieve the best overall performance.

TM1 Query Handling

Before we get into the specifics of TM1Connect, it is important to understand some basic information on how TM1 operates when a view request is performed. TM1, at its core, is a high performance, in-memory, OLAP database. An OLAP database differs dramatically from a relational database in how data is stored and accessed internally.

A relational database can be considered as a two dimensional array (rows and columns) and the data (fundamentally) is stored in this manner, so that when you access data on a specific row, the relational database knows how to access the remaining columns based on the current row.

In an OLAP database, however, there is no such relationship between rows and columns, even though the data being loaded into the OLAP database may come from a format which has rows and columns (such as a CSV file or an SQL Query). Data in an OLAP database is (fundamentally) stored as an array, and to access data in this array, you need the coordinates (or address) of an item in the array in order to access it.

So what does this mean? Well, let's take a look at a simple two dimensional table (relational) vs a two dimensional cube (OLAP) and see how the same query operates in each system.

We will start with a simple set of employees (shown here as a cube view in TM1). Represented in a relational database, it would be a table (Employees) with 6 columns. In TM1, it is a two dimensional cube (Employees) with Employee and Employee_m dimensions.

Employee	Employee Start Date	Employee End Date	Employee Title	Employee Billable Hours Per Week	Department
Robert Stevenson	1998-04-13		Northwest Regional Accounts Manager	40.00	US Accounting
Catherine Batestein	2011-01-12	2012-06-25	Customer Support Representative	20.00	Support
Paolo Martinez	2012-03-13		Customer Support Representative	20.00	Support
Paul McCowns	2002-07-20		Senior Mechanical Engineer	40.00	Product Engineering
Susan Charlestine	2009-04-02		Corporate Branding Associate	40.00	Quality Assurance
Michelle Alberts	2007-06-19		Hardware Maintenance Supervisor	40.00	Computers and Hardware
Elen Schminsky	2012-08-04	2013-09-22	Administrative Assistant	13.00	Financial Analysis

For this example, let's assume that we want to return all of the columns of data for the employee Paul McCowns.

Performing the query using a relational database

In the relational world the above request would be represented in SQL as:

```
SELECT * FROM Employees WHERE Employee = 'Paul McCowns'
```

When this query executes, the relational database will scan (or use an index) to locate the row that matches 'Paul McCowns'. In this case, it finds that Paul McCowns is on row 4. Since a relational database stores all of the related columns to a particular row in the same location, it is a simple matter of parsing out the columns of data and returning the record (as illustrated below):

Employee	Employee Start Date	Employee End Date	Employee Title	Employee Billable Hours Per Week	Department
Robert Stevenson	1998-04-13		Northwest Regional Accounts Manager	40.00	US Accounting
Catherine Batestein	2011-01-12	2012-06-25	Customer Support Representative	20.00	Support
Paolo Martinez	2012-03-13		Customer Support Representative	20.00	Support
Paul McCowns	2009-07-20		Senior Mechanical Engineer	40.00	Product Engineering
Susan Charlestine	2009-04-02		Corporate Branding Associate	40.00	Quality Assurance
Michelle Alberts	2007-06-19		Hardware Maintenance Supervisor	40.00	Computers and Hardware
Elen Schminsky	2012-08-04	2013-09-22	Administrative Assistant	13.00	Financial Analysis

If a query is encountered that asks for multiple employees, all a relational database has to do is gather the list of row numbers that are to be included, and then simply loop through those rows gathering the same set of columns (positionally the same data types and offsets between records) to return the results.

The result of this is that a relational database can iterate extremely fast over a list of detail records to return the information stored in a table. We will discuss a little bit later where a database starts to encounter problems.

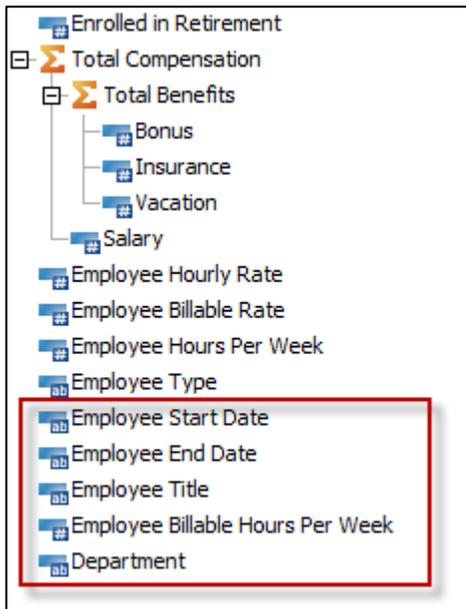
Performing the same query using an OLAP database

Now, let's perform the same query using TM1. The first thing to understand is that there is no concept of a 'record' in TM1; so SELECT * has no meaning. What is needed for TM1 to return a set of data is a list of coordinates (addresses) for what data is needed. This set of coordinates can originate from view or an MDX query. In either case, the elements within a dimension first need to be interrogated to determine if they are part of the query and then once that has been determined, they can be assembled into a list of coordinates to retrieve from the cube (array).

For this example, the Employee and Employee_m dimensions will be interrogated. First the Employee dimension will be scanned (via an index) to find the element 'Paul McCowns' (Element Index 15) as shown in the dimension element list below:

- Employee 055
- test
- Jordan
- Llamar Newson
- Employee 050
- Nancy Halverson
- Employee 046
- Jordan McAffe
- Alex Beemer
- Leroy McCallister
- Zach Wambost
- Robert Stevenson
- Catherine Batestein
- Paolo Martinez
- Paul McCowns**
- Susan Charlestine
- Michelle Alberts
- Elen Schminsky
- Bradford Keller
- Karl Melloy

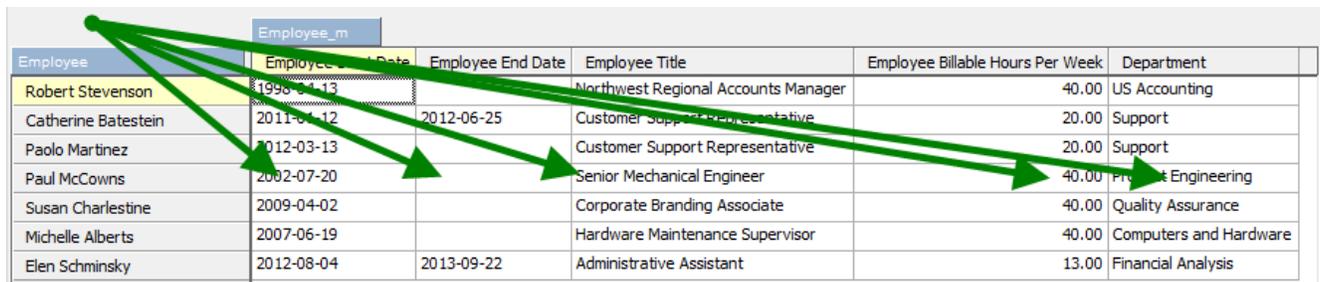
Next, it does the same for the dimension Employee_M (Elements 12-16):



Now, with these indexes, it can now assemble the list of addresses needed to return the proper information. The list of addresses are:

(15,12), (15,13), (15,14), (15,15) and (15,16)

Now, TM1 is ready to retrieve the data by looking up those locations in memory one at a time:



Employee	Employee Start Date	Employee End Date	Employee Title	Employee Billable Hours Per Week	Department
Robert Stevenson	2009-04-13		Northwest Regional Accounts Manager	40.00	US Accounting
Catherine Batestein	2011-04-12	2012-06-25	Customer Support Representative	20.00	Support
Paolo Martinez	2012-03-13		Customer Support Representative	20.00	Support
Paul McCowns	2002-07-20		Senior Mechanical Engineer	40.00	Product Engineering
Susan Charlestine	2009-04-02		Corporate Branding Associate	40.00	Quality Assurance
Michelle Alberts	2007-06-19		Hardware Maintenance Supervisor	40.00	Computers and Hardware
Elen Schminsky	2012-08-04	2013-09-22	Administrative Assistant	13.00	Financial Analysis

With this simple query, TM1 had to perform 6 lookups, whereas a relational database had to perform 1 lookup.

Comparing the work in a complex query between OLAP and relational databases

With the basic understanding of how relational and OLAP perform queries, now consider something closer to a real world situation:

It is not uncommon for a relational table to have many columns. Oftentimes many of these columns have to be represented in TM1 as a separate dimension (either because of analytical reasons, or because of the data relationship between columns). Let's assume, that a table with 10,000 rows has 30 columns and 10 columns are dimensionally separate. In TM1, this would (loosely) become a 10 dimensional cube with 20 measures.

To request a dump of all 10,000 rows from a relational database, would be a trivial matter of iterating through all the rows and spitting out the results in the requested manner. At worst case, it would require 10,000 lookups, best case (if contiguously laid out) would just do one lookup and then loop through all the records.

In the TM1 world (assuming a 10 dimensional cube with 100 unique values per dimension), the number of lookups would become $100^{10} = 100,000,000,000,000,000$ (or 100 quintillion) lookups! Ok, well TM1 is smarter than that, and

knows how to deal with combinations that are not actually populated. Assuming a 10 dimensional cube and 20 measures, TM1 would have to do at least 200,000 lookups to return the same data that the relational system can do in 10,000 lookups. This does not take into account all of the manipulation required to identify which of the combinations of the 100 element dimensions are valid (which takes more processing time).

So why use TM1 then?

Because TM1 is array based and not record based, this means it can access any information just by knowing its address, including totals and subtotals (consolidations) of detail data. This is crucial when trying to perform complex calculations that would ordinarily be extremely time consuming in a relational database. Consider, for example, what it would take in a relational database to calculate the percent of hours worked in each department by employee.

Employee	Employee Start Date	Employee End Date	Employee Title	Employee Billable Hours Per Week	Department
Robert Stevenson	1998-04-13		Northwest Regional Accounts Manager	40.00	US Accounting
Catherine Batestein	2011-01-12	2012-06-25	Customer Support Representative	20.00	Support
Paolo Martinez	2012-03-13		Customer Support Representative	20.00	Support
Paul McCowns	2002-07-20		Senior Mechanical Engineer	40.00	Product Engineering
Susan Charlestine	2009-04-02		Corporate Branding Associate	40.00	Quality Assurance
Michelle Alberts	2007-06-19		Hardware Maintenance Supervisor	40.00	Computers and Hardware
Elen Schminsky	2012-08-04	2013-09-22	Administrative Assistant	13.00	Financial Analysis

With the above table, Catherine Batestein would be working 50% of the effort in the Support department and Paolo Martinez would work the other 50%. To get that answer in a relational database requires either some advanced processing or a severe performance hit on the query because it has to first know how much, at a total level, is in Support before it can determine what the utilization percent is of that person. Sometimes calculations like this are not even possible with a single query and you are forced to perform pre-processing.

With TM1, however, you can directly request this information without any extra work. And, once this value is calculated and cached in TM1, it can be referenced by subsequent queries instantaneously.

What does this all mean and how does it relate to TM1Connect?

A little understanding of how TM1 handles queries will help in designing the views in TM1Connect to service your users (or applications).

TM1Connect offers two methods for obtaining data from TM1. Batch Mode and On-Demand mode. Batch mode is designed for the data dumps from TM1 because TM1 will need time to re-construct a record by record dump of the contents in the cube. On-Demand mode, should be used to pull out only the information needed to satisfy the user’s request of information, and the views should be defined in such a way as to facilitate this.

While it may seem natural to define a query the encompasses the whole cube to allow the user to pick and choose what they want out of the cube, it causes TM1 to first render that NxNxNxN view only to be largely thrown out because the user only wanted certain pieces.

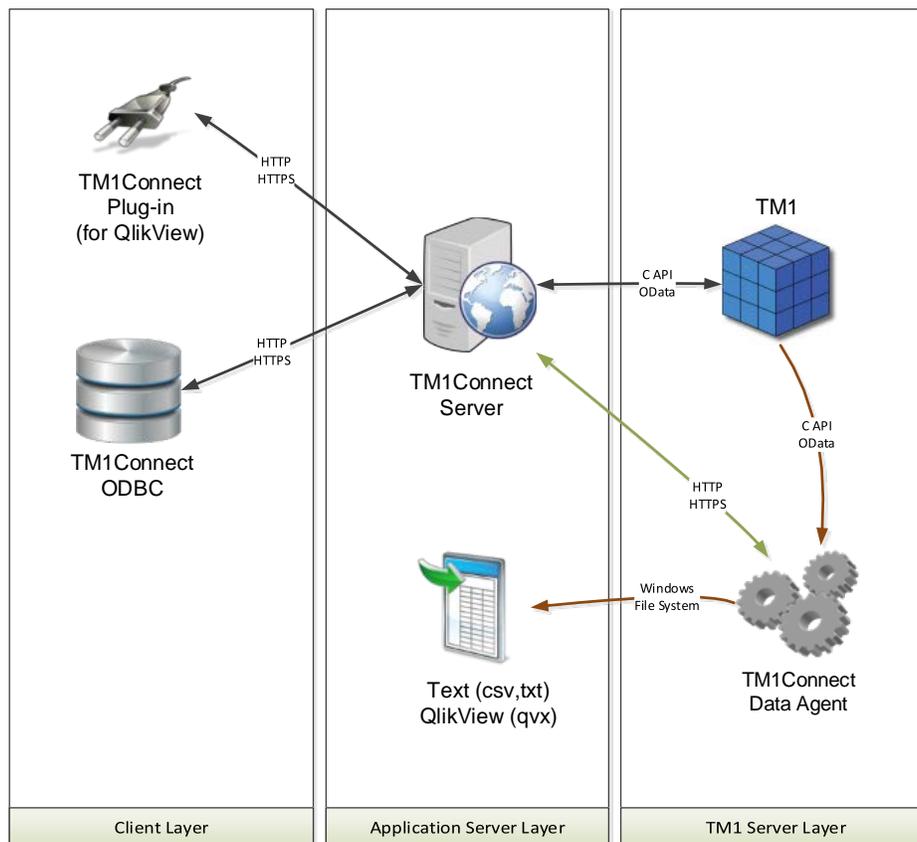
It is better to define multiple narrowly focused views (such as 2015 Forecast and 2015 Budget) than a single all-encompassing view (all versions, all years, all departments, etc). It can be done, but you must understand that there is a performance penalty for doing so in the OLAP world.

For example: If your report or dashboard shows quarterly sales by customer, don’t use a view which was defined at the monthly level (so that the values can be summed in the report), instead, create/use a view which already returns quarterly values and then use that in the report/dashboard. This will pull the pre-aggregated value directly from TM1 with no additional processing required.

TM1Connect Query Handling

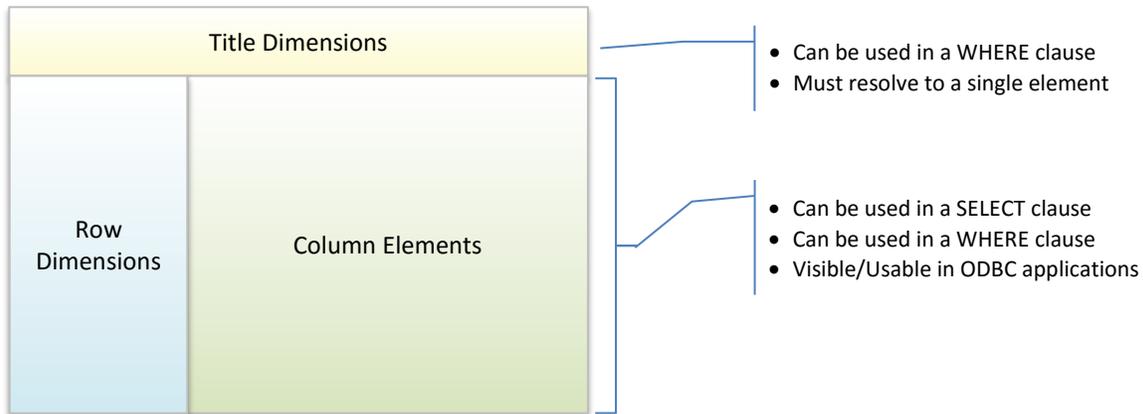
With a better understanding of how TM1 handles queries, it is now time to look closer at how TM1Connect works in conjunction with TM1. TM1Connect has two different methods for requesting data from TM1; via the C API and via OData (REST) API. TM1Connect operates mostly the same between these two API interfaces with some nuances relating to the nature of each API. This document will lightly touch on some of these differences, but not go into specific details about one interface vs another; that is another document entirely! ;-)

TM1Connect, at its core, is a web service provider that receives requests for information from TM1 via several different interfaces, namely ODBC Driver, QlikView Plug-In, and a plain ol' web service query. The TM1Connect Data Agent, however, uses our internal TM1Connect API to talk directly with TM1 and does not go through an HTTP web server.



Because the QlikView Plug-In and the ODBC driver operate over HTTP, TM1Connect can actually be hosted (along with TM1) on a remote server via a WAN connection. Both the QlikView Plug-In and the ODBC driver communicate via standard relational SQL queries that are then translated by TM1Connect server into an appropriate TM1 view. This is made possible because TM1Connect pre-defines the layout of a view in the TM1Connect manager. This layout is called a View Definition.

A view definition allows you to position which dimensions in TM1 are placed on the titles, rows or columns of a TM1 view much like TM1 Perspectives/Architect. The elements within the column dimension(s) become columns in the data returned from a query to TM1Connect. In contrast, row dimensions become the columns in the returned data from TM1Connect as opposed to the elements within the dimensions. At the same time, TM1Connect allows the elements within the title and row dimensions to be altered at runtime. This enables an SQL style query to be written against a TM1 cube. Without a view definition, a cube, by itself, has no concept of rows and columns and an SQL Query is meaningless. The diagram below illustrates, conceptually, the structure of a view:



Now, consider the SQL Query:

```
SELECT Employee, Name, StartDate, EndDate
FROM Employees
WHERE Employee = 'Paul McCowns'
```

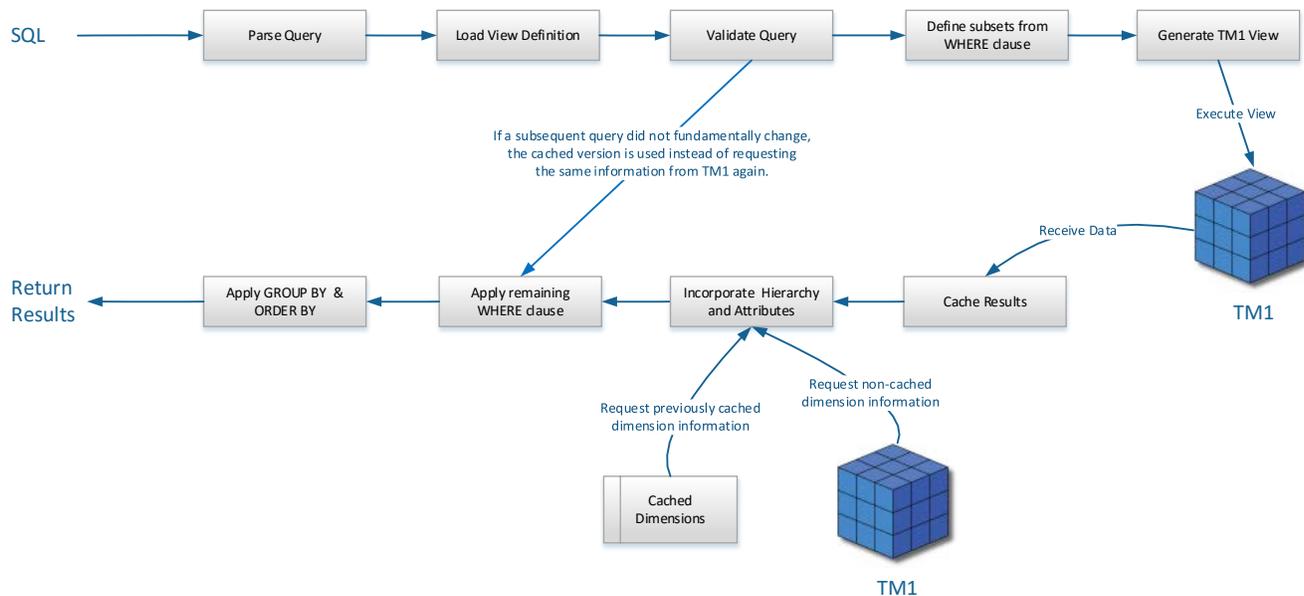
The `SELECT` clause will pick out from the available columns (row dimensions and column dimension elements) from the returned data out of the view definition called 'Employees'. Note that the `FROM` clause references a TM1Connect view, not the name of a cube. This allows you to have multiple views against the same cube, each targeted with relevant information. The `WHERE` clause changes what elements are selected (filtered) in the rows and title dimensions (subsets) of the view.

NOTE:

Dimensions placed in the title area of a view do not appear in the rows or columns of the returned data. This means that applications using the ODBC driver will not natively be aware that the title dimensions can be used as part of a `WHERE` clause (because they rely on the results dataset). You still can use a where clause against title dimensions, but you may need to edit the query manually.

So now that we have an established view definition, when a request using SQL Query syntax comes in from the QlikView Plug-In or the ODBC driver, TM1Connect uses that definition to formulate the appropriate view in TM1. Some conditions in the `WHERE` clause will dynamically limit what elements are used in the title and row dimensions of the resulting TM1 view, but limiting the columns via a `SELECT` clause, however, will not reduce elements in the column dimensions of the TM1 view. So, once a view definition is created, the columns will always be requested from TM1 first, but then removed at the point where the results are returned back to the user. This is important to understand, because *if you define a view that has all the columns your user will ever need in a single view definition, TM1Connect will pull all of the columns from TM1 each time that view is used, regardless if only one or two columns are requested in the `SELECT` clause.*

The following diagram illustrates the process TM1Connect follows when an SQL statement is received:



Notice, as this diagram illustrates, that some `WHERE` clause operations and all `GROUP BY` and `ORDER BY` operations are performed after the information is received from TM1. The ordering and the grouping of data is not performed in TM1, and under certain cases, neither is the `WHERE` clause. Let's first dive a little deeper into how TM1Connect handles an SQL query internally.

How TM1Connect resolves the `SELECT` clause

When an SQL statement is encountered, any columns specified in the `SELECT` clause must correspond to a column as defined by the view definition (must be a row dimension, or a column dimension element). As the view is received from TM1, the appropriate columns (including formula based columns) are determined and the result record is created. Once all of the result records have been processed, the data is ready for sorting and grouping.

Keep in mind that selection of data from the columns are done after the view has been requested and returned from TM1. The primary reason for this is because TM1Connect allows for calculated columns which may depend on columns not part of the `SELECT` statement. And, as previously mentioned, creating an all-encompassing single view with few columns used will have a negative impact on performance.

How TM1Connect resolves `WHERE` clauses

Since TM1Connect only requests the explicit dimension element and the associated data from the cube, careful attention must be made by TM1Connect when attempting to optimize queries that contain `WHERE` clauses. If the `WHERE` clause explicitly requests elements from a row dimension, TM1Connect will structure the query such that the view requested from TM1 will be restricted to the specified elements. If, however, the `WHERE` clause uses an expression that cannot be determined in advance by TM1, all rows in the view will be requested from TM1 and then TM1Connect will perform the `WHERE` clause on the result set. If TM1Connect is forced into this mode of operation and the defined view is sufficiently large but the filtered result set is small, it may be "perceived" as a performance penalty over similar views in which the `WHERE` clause could be optimized in TM1.

Some examples of `WHERE` clauses that will leverage TM1 to perform the filtering:

```
WHERE Employee = '458'
WHERE Employee in ('123', '573', '225')
WHERE Employee like '*2*'
```

Examples of `WHERE` clauses that will force TM1Connect to filter rows after the view has been returned from TM1:

```
WHERE Left(Employee,1) = '4'
WHERE Employee = Column2
```

How TM1Connect resolves ORDER BY and GROUP BY clauses

All ORDER BY and GROUP BY clauses are performed after the data has been returned from TM1 and before it is returned to the client. Although TM1 does understand how to sort elements within a dimension, it does not understand how to perform sorting across dimensions or across attributes and data values within the context of a view. TM1Connect, however, can perform sorting and grouping over any columns defined in the view definition, including calculated columns that are not part of the data available in TM1.

TM1Connect does not support JOIN clauses

At the time of writing this document, TM1Connect does not yet support JOIN clauses. Joining of data must be performed by the ODBC consumer.

Dimension Caching

TM1Connect understands how to obtain dimension attributes and hierarchy information and present it in a relationally friendly manner for ODBC or QlikView plug-in consumers. It achieves this by flattening the dimension levels into separate columns; one for each level within the hierarchy. And, in a similar way, element attributes are placed in separate columns. This allows relationally oriented clients to sort, aggregate and group data from TM1 by each attribute and/or level without needing to filter out/manage subtotals intermixed with detail records.

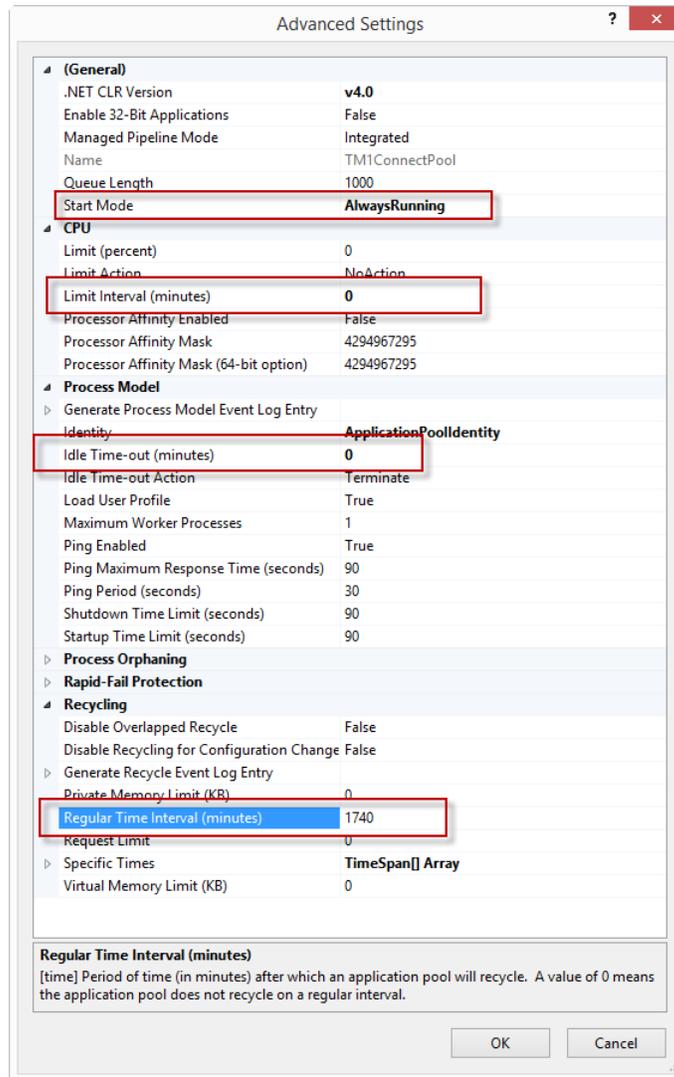
In addition to flattening a dimension with a level based hierarchical structure, TM1Connect can also provide some basic ragged hierarchy handling as well; i.e. filling in the gaps where the levels are not uniform, filling in gaps from left to right or right to left etc. TM1Connect can also bring in attributes for the specified elements from the view definition as well as the attributes of the elements within the hierarchy.

These operations are data intensive and can impact the performance of capturing this information from TM1 each time a request is made. To minimize this impact, TM1Connect, provides the ability to cache dimension information at server startup so that only cube data needs to be requested from TM1 for each request. The only caveat, however, is that this should only be used where the dimension structure and attributes are not changed frequently or invalid results may occur.

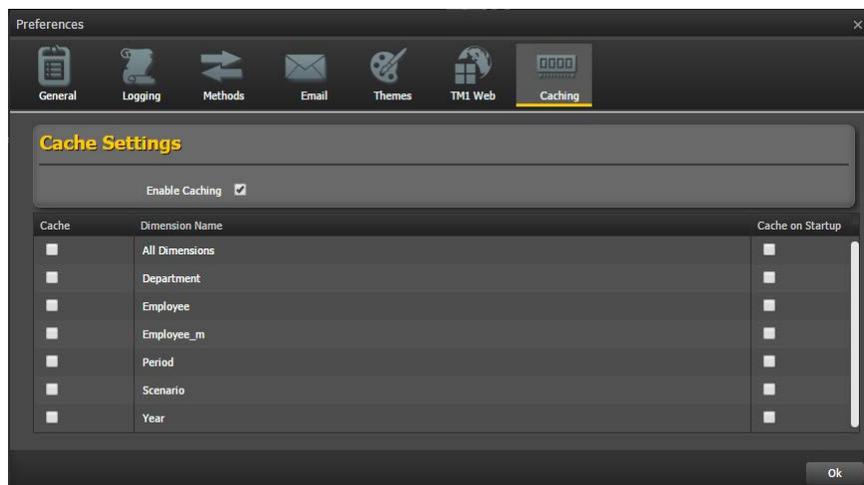
IMPORTANT:

When enabling cache on startup, especially for large dimensions, ensure the proper timeout settings for IIS are configured for the TM1Connect application pool (usually called TM1ConnectPool). If this is improperly configured, TM1Connect may not be able to load the dimension information or may not operate at all.

To set the dimension caching on startup, IIS must first be configured to allow TM1Connect to start upon IIS startup, and the web site refresh settings should be adjusted appropriately for your environment. A sample set of values are shown below.



With IIS properly configured, TM1Connect can then be instructed to cache one or all dimensions. To enable caching, click on the settings button  in the TM1Connect manager and select the Caching options. Click on the checkbox 'Cache on Startup' for all of the desired dimensions. The next time the application pool is restarted, TM1Connect will load the selected dimensions in memory.



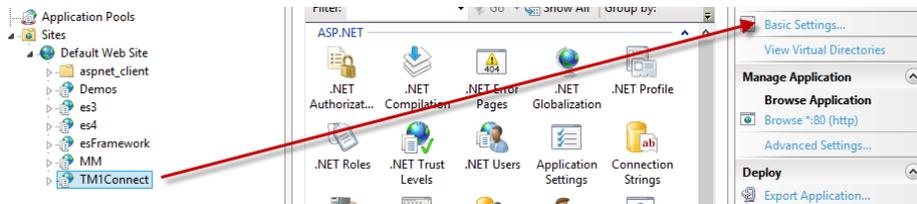
NOTE: At the time of writing this document, TM1Connect (currently and in the future using the C API) supports dimension caching on startup. However, with the upcoming releases of TM1Connect using the TM1 10.2.2 REST API, additional caching options, such as full, partial dimension caching as well as startup, timeout and automatic refreshing may be offered.

Session-Based View Caching

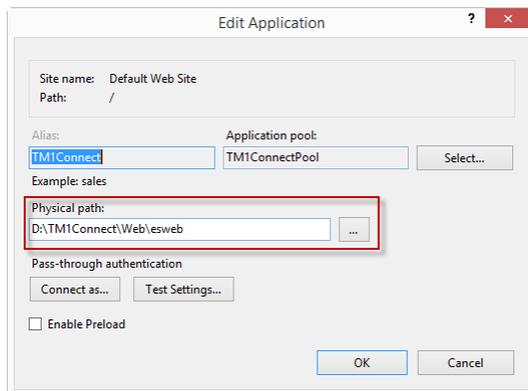
TM1Connect is designed to provide real-time information from TM1 through the ODBC driver or QlikView Plug-in. Some ODBC compliant applications (like Tableau) perform multiple queries over the same result set in order to discover information about the content of the data, such as number of rows, unique elements in a column, etc. For this reason, TM1Connect provides session-based caching of view data. Session based view caching is meant to be a short term buffer for multiple requests over the same data.

The default configuration of the view caching in TM1Connect is set at 60 seconds between requests, which is sufficient for most applications to capture all of the metadata information about the result set. This value, however, can be increased or decreased based on your environment, but understand that it applies to all views.

To change this setting, edit the web.config file located where the TM1Connect web service references (by default, this value is C:\TM1Connect\Web\ESWeb). If you are unsure, locate and select the virtual directory for TM1Connect in IIS Manager and click Basic Settings...



The path will be located in the text box labeled Physical path:



Global View Caching

Session caching in TM1Connect provides a short term cache of view data to service rapid-fire discovery queries issued from SQL based client applications. The data in the session-based cache is stored on a per session (user) basis. Meaning, if two users pull the same data each will have a version of the data stored in TM1Connect. Session-based caching is important when security must be applied to the data being returned, but if the data does not have security restrictions, global caching will query the data from TM1 only once, and provide that data to all users requesting the same information.

For further information on view caching please consult the TM1Connect Help guide.

Other Performance Considerations

Use TM1 subsets or MDX for large dimensions

If a lot of members are needed from a dimension, use an MDX expression or a public TM1 subset in the view definition rather than an explicit list of members.

When creating a view in TM1Connect, a definition is saved on the TM1Connect server (separately from TM1) which contains all the information required to re-construct the view. If a TM1Connect view is created from an existing TM1 view which uses private un-named subset(s), all of the members defined in that subset become explicitly part of the view definition, and the size of the view definition file can become large.

Consequently, each request of data for that view, requires TM1Connect to read this information, translate it to a TM1 view, and request the view to be generated. Using views with large amounts of members will incur additional transmission and processing time; impacting the performance.

Create subject specific views rather than an all-encompassing detail view

TM1Connect always pulls the full amount of data from TM1 as it is defined in the view definition; even if the ODBC client only requests a small portion of that information. Create subject specific views (and use these subject-specific views at the appropriate time), rather than one single all-encompassing view that forces the user to always pull low level data for every query.

Consider some cases such as these, when creating your views:

- **Separate views at different levels within a hierarchy.**
Create separate yearly, quarterly, and monthly views of the data so that when quarterly totals are needed, the aggregation is done in TM1, not on the client after all the detail data has been transmitted.
- **Use title dimensions to restrict data.**
Segment data by placing dimensions in the title area so the user can select which set of data they need, when they need it. Consider, for example, a scenario/version dimension. Allow the users to select which version they need (and ultimately query TM1 for that specific scenario/version), rather than have them filter the results for the specific version after all versions have been queried.
- **Segment the views even if the data is stored in the same cube.**
Split related data into multiple views where it makes business/logic sense. If you have a single financials cube, for example, consider splitting out Balance Sheet accounts into a separate view from Income Statement accounts and allow users to use these views with a more narrowly focused query.