Modeling Proven Practices for IBM Cognos Framework Manager

Event ID: 154472

Chris McPherson: My name is Chris McPherson. I'm the Product Manager for Framework Manager and Metadata in the Cognos platform. I am going to be speaking to you today about modeling proven practices for IBM Cognos Framework Manager. Some of the things we are going to talk about today are modeling metadata for business intelligence, modeling for consistent, predictable results. We are going to talk a little bit about maintainable models. We will talk about purpose built models to match reporting requirements. Then we are going to move into some modeling concepts. We will touch on topics of cardinality, determinants, granularity and SQL generation.

And finally we will talk a little bit about some useful features in Framework Manager to help you streamline your modeling. We are going to look at the bulk change capabilities, Context Explorer and the Model Adviser.

Modeling for consistent and predictable results – at the end of the day what you really want to get out of a good Framework management model is consistent and predictable results. You also want good performing queries, but the numbers really need to be correct. Some of the keys to driving those results you want are understanding your reporting requirements. You need to sit down with your report authors and make sure that their reporting requirements are clear. You really need to have a sense for the sort of business questions report authors will be trying to answer. If you are not clear on what it is that they want, it is unlikely that you will be able to deliver what they want.

Questions like what kind of reports will they be writing. Do they need relational packages or dimensionally remodel packages? Or do they need both? What additional requirements are there for the application? Do they need security implemented, and if so, what kind? Is there a requirement for multi-lingual data? These things are the easiest to manage when you know about them in advance and can implement them as you go rather than trying to retrofit them afterwards.

Know your data. It is important to communicate with your DBA and become familiar with the structure you will be modeling. You should have he or she identify the fact tables, dimensions and which tables connect as both.

Great purpose specific views. Give the report authors what they want by organizing it into a logical fashion tailored for their reporting requirements.

The last point to make in this slide is to leverage modeling resources. There is a wealth of information available in the FM documentation, the guidelines for modeling metadata and the proven practices website. There are a lot of documents out there that can explain and illustrate how to tackle specific challenges. Chances are you are not the first person to run into them and there might be an answer already out there.

Building maintainable models. Building a maintainable model is something that I think is really fundamental. You want to build a model that is well organized and it is easy to navigate. To that end import only what you need. Importing only the objects that you need at the start and then gradually adding as you go will make it easier to navigate and find things in your model. Also, change as little as possible in the import view because changes at this level are universal in the sense that they will affect all of the upper level objects that might use them as a source.

After you import, always verify the relationships and query item properties in the import view. You want to ensure that the necessary relationships are present and that the cardinality is correct for the reporting purposes. You should also ensure that the query items are correctly identified as this can affect query generation.

I recommend creating a business view or a logical view to apply your business rules. You can add your calculations, filters, et cetera at this level as well as give your objects names that report authors will understand. Something else that a business view does is give you a level of insulation from changes that you might need to make down the road. Objects at this business level can be retargeted to different import view sources without impacting the higher level objects.

Create a presentation view to organize your objects for the report authors. Presentation view is the layer that you want the report authors to see in the metadata tree in the studios. So organize your objects accordingly so that it is easy to navigate and understand.

The last item in this section that I would like to speak about is keeping models to a manageable size. The larger the model is, the more difficult that it can be to navigate. Also very large models can cause performance issues in Framework Manager and can make the UIs sluggish and unresponsive. Certain memory intensive operations are more frequently affected but it is a matter of good practice to keep your models below 50 megabytes in size.

Purpose built models. I would recommend avoiding the temptation to build a one-size fits all model. The reality is that it will be difficult to build and maintain a model that will satisfy all of the reporting requirements of all of your lines of business.

A better approach is to build several smaller, more focused models that are really targeted at a line of business or a set of requirements. Likewise, for ad hoc reporting models that are easy to navigate and almost guide the user are best. Most people do ad hoc reporting because they want a quick and simple way to get out their data. So we should model and publish packages accordingly.

For professional report authoring, more complex models and packages may be required. And finally, for analysis you are talking about a dimensionally modeled package to allow for the drill-up, drill-down experience.

Now we are going to talk a little bit about some modeling concepts in Framework Manager. First modeling concept I would like to talk about is cardinality and how Framework Manager uses it. Cardinality is used to identify query subjects that behave as dimensions and facts. One to end cardinality implies fact data on the inside and dimensional data on the one side. This is also used to help query engine avoid things like double counting and to support loop joints.

The role of cardinality in the context of a query is important because cardinality is used to determine when and where to split the query when generating multiple fact queries. If dimensions and facts are incorrectly identified, stitch queries can be created unnecessarily. This is costly to performance or can result in incorrectly formed queries which can give incorrect results. Query subject may behave as a fact or a dimension depending on the query subject included in the query.

So an important recommendation to keep in mind is that you should always review and verify the imported relationships for their accuracy and cardinality. Check for multiple relationships, optional cardinality, although sometimes this is necessary for your business requirements. Wherever possible use cardinality to clearly define facts in your data source view.

Determinants – what are they and when do I need them? Determinants reflect levels of granularity by representing subsets or groups of data in a query subject. They are used to provide the query engine with information to ensure correct aggregation of repeated data in the query subject. We recommend that you always review the determinants that are imported and if necessary modify them or create additional determinants. By modifying determinants you can override the index and key information in your data source, replacing it with information that is better aligned with your reporting and analysis needs.

Determinants are required to avoid double counting of fact data when dimensions connect to fact tables at levels of granularity that have repeating keys. A simple example that illustrates this scenario is that of a time dimension that has months and days. The month will repeat for each day in the month. For example, May 1, May 2, May 3, et cetera. You wouldn't want to aggregate the values of the month for each day in the month.

The second scenario is one where a query subject behaves as a dimension and has multiple levels of granularity and will be joined to fact data on different sets of keys. For example, the time dimension has multiple levels. It is joined to inventory on the month key because inventory is tracked by month. It is also joined to the sales fact on the day key because sales are tracked by day.

Another less common scenario includes the use of BLOB data types in a query subject. Query and BLOB data type values in a query subject require additional key and indexed information. If this information is not present in the data source, you can add it via determinants.

To try to give a simple example that will explain determinants and granularity, let's return to the time dimension scenario. In this case each unique instance of month key will repeat once for every day in a month. If you don't tell the query engine that month key requires grouping at the month level, sales target will be double counted for every day of the month, resulting in wildly incorrect sales target numbers.

To eliminate the potential that double counting will occur, you should set a determinant at each level of granularity. To continue the previous example, in this case you would set a unique determinant on the day key as it is the unique key in the table and is at the lowest level of granularity. As such, you would select the uniquely identified check box and do not check the group by box.

For month key you would not select the uniquely identified box because it repeats in the data, once for every day, and therefore is not unique. You would select the group by determinant for the month key.

The next topic we are going to touch on is SQL generation. You can specify how Framework Manager generates the SQL that retrieves data from relational data sources for data source query subjects or model query subjects. The SQL generation type of a query subject can be set to either As View or Minimized. By default it is set to Minimized. When the generation type is set to Minimized, the generated SQL contains only the minimum set of tables and joins needed to obtain values for the selected query items. This is generally desirable because the fewer tables included in your SQL statement, typically the faster your queries can be processed.

When generation type is set to As View, Framework Manager generates queries that contain the full SQL statement that define the query subject. The SQL is treated as a view. For example, you want the query to return the same number of rows each time that it is run.

So now let's have a look at a little demo where we can see where the SQL generation settings are configured.

So I select a query subject, I right click and select Edit Definition. From the resulting dialogue I will select Query Information. And then Options. And then finally the SQL Setting. You will see I have the option to set as Minimized or As View. As I mentioned, Minimized is the default setting.

Generally speaking, because of the performance benefits of having fewer tables referenced in the queries, it is recommended to use Minimized SQL wherever possible. However, there are certain settings that will trigger As View SQL and will override the setting, for example, determinants on a model query subject, relationships between model query subjects and filters, macros, calculations, et cetera in your data source query subjects.

It is important to remember that As View SQL is not always bad. In fact it can sometimes be necessary depending on your requirements. At the end of the day, our modeling guidelines focus on ensuring consistent, predictable results, so in some cases tradeoffs need to be made to ensure that.

Now we are going to spend a few minutes talking about features that can help streamline the modeling process. First off we will talk about Bulk Change capabilities. This feature allows users to search for query items by property and then make changes in Bulk. As we noted earlier, it is very important that query items be correctly identified in terms of usage as this can affect query generation. It is important to review the usage of the query items in your data source query subjects after import to ensure that they are properly identified. Because you have the ability to search for query items by property as well as by name, this feature would be useful in making modifications to collections of query items when performing your validation after import.

In this demo, after we have done an import we are going to search for all of the query items identified as fact to ensure that they are correctly identified.

I click the search button and I can now see all of the query items that have the usage of fact. I can see from the list that a number of key fields have been incorrectly identified as facts. I want to change the property to ensure that they are handled as identifier by the query engine. I can simply select all of these columns and I see them listed in the properties panel below.

To make these change I just need to select the usage dropdown and select Identifier. Once I have done that I can pull down on the little arrow to make the change across all of the columns. I can now go back and examine those columns and see that they have been changed to Identifier.

The next feature I would like to talk a little bit about is the Context Explorer. Now the Context Explorer is a mechanism that lets you select query subject in your model and then launch a diagram to display the other object around it. It will show the selected query subject and all of those query subjects that are related. This gives you a means to do some lower level exploration of a portion of your model.

In the Context Explorer itself you have the ability to perform a lot of the same actions you could do in the regular view. You can create new query subjects, you can edit existing query subjects, modify existing relationships between query subjects or add new ones.

There are also a couple of handy features like the ability to print the diagram you see or have organized in Context Explorer. There is also a screen capture capability so that you can easily paste an image of the diagram you are using into an email or other document for sharing and collaborating with team members.

To launch the Context Explorer, simply select the query subject from the metadata tree in Framework Manager. From the right click menu you will have the option to launch Context Explorer. You can see that the result is a diagram that has the selected object at the center and all of the related objects laid out on the screen around it. From here you can edit the relationships, add relationships, modify the query subjects as required. You also have the option to show all of the related objects with all of the relationships.

This might be a little too much information, so the default view might be fine for most operations. As I mentioned you also have the ability to print and from the right click menu you can take a screen capture that you can easily share with teammates.

The last feature I want to talk about is one that was first introduced in version 8.3 of Framework Manager. And that is the Model Advisor. Model Advisor is a tool that analyzed your model or a part of your model based on the documented modeling guidelines and identifies potential issues that should be examined. It will look for things like ambiguous join paths, potential issues with cardinality and determinants and so on. For novice modelers, the Model Advisor can be helpful as an assistive tool; problems that are encountered are documented and there are links to images to demonstrate the issue encountered as well as links to the documentation where you can find more detailed information about the problem at hand.

For more experienced users, the Model Advisor can be a good diagnostic tool. Periodically in the modeling process users can run the Model Advisor to identify any potential areas of concern that should be reexamined. Model Advisor will check for potential issues in the following area – cardinality; join path; determinants; setting and [governor] conflicts. You have the ability to configure the Model Advisor to check for all of these conditions or select areas depending on your requirements and where you are in your modeling process.

If issues are identified, there are links directly to help documentation, as well as an option to examine the problem more closely in the Context Explorer.

Something worth noting is the difference between Model Advisor and the Verify Model feature in Framework Manager. Both are very useful features and can help you along the modeling process but in different ways. The verify model checks the validity of the model or selected objects in the model, for example a package. It will check the syntax of expressions to ensure they are valid. It will check to ensure the determinants you have set are completely defined and will identify any invalid object references. Verify model does not look for potential modeling issues. It is simply checking to ensure that the model or the selected objects are valid. Verify model should be used in conjunction with Model Advisor as you go through the modeling process. You should run verify model periodically as you build your model to ensure that all objects are valid and correctly defined.

Now let's have a look at Model Advisor. We can simply select a name space or a query subject, and from the right click menu we can invoke Model Advisor. From here we have the option to configure the Model Advisor to check for the potential issues that we are interested in. We have the option of enabling or disabling all of these options and the Model Advisor will do its analysis based on these settings.

Once we have configured the Model Advisor we simply click the Analyze button and we end up with a useful list of potential issues based on the criteria that we set. There are links to the Context Explorer where you can have a more visual view of the problem that it is describing. From here you can also take steps to remedy the potential issue. And there are also helpful links to the documentation where you can drill down into the problem and get some more detailed information along with some potential paths to resolve.

You can also print or save this list and have it on hand as we work through our modeling issues and have it available to share with other team members.

So as we saw, the Model Advisor can be invoked to analyze your entire model or can be used on a per object basis. Like the verify model feature, it is something that should be used in stages as you build your model. One suggested workflow would be to import your metadata, run Model Advisor, and then address any issues before beginning to model your business rules. As you apply your business rules and your business view takes shape, run the Model Advisor again and address any issues and so on.

One important point I would like to make about using the Model Advisor is that not everything that the feature detects needs to be addressed. The Model Advisor uses modeling guidelines to detect potential issues in the areas that we talked about. Depending on your reporting requirements and your data, some of the areas that the Model Advisor identifies might be completely valid or even necessary to ensure that you get the consistent, predictable results you are looking for out of your model.

For additional resources there are many good documents and proven practices documents available at the IBM Cognos website. Resources of note are the IBM Cognos 8 Framework Manager Guidelines for Modeling Metadata, the IBM Cognos 8 Framework Manager User's Guide, both of which ship with the Framework Manager Product, as well as the IBM Cognos Proven Practices site at www.ibm.com/developerWorks.

Thank you very much for your time. I hope you enjoyed this presentation.

Hello everybody it is Chris McPherson here. I would be glad to answer whatever questions you have. If you want to submit them by chat what I will do is I will read out the question and I will give some comments. There are a lot of people on the line, so if we don't get to all of them please don't hesitate to send me an email and I can follow up with you offline.

Obviously if the question is something that we need some back and forth or need to talk about something specific then I would probably advise that you just send me an email and I will follow up with you directly.

So, one of the first questions we have got here is when should a query subject item be set as an identifier and not an attribute? So generally speaking the rules Framework Manager uses for this are if a query item is part of a key in a determinant it generally speaking should be an identifier. Likewise if it participates in a relationship it should be an identifier. And if it is a data type date or time it would be an identifier.

If it is none of the above and it is not a fact than it would be an attribute. Attributes are typically strings.

Next question. Why is it preferred to apply relationships to data sources, query subjects and model query subjects? Well, generally speaking we recommend keeping the relationships at the data source level because these are the objects that are going to be reused. Now, having said that, like anything else it really depends on the reporting requirements. Adding relationships to your model query subjects can trigger As View behavior and it also requires that all relationships to that model query subject would need to be recreated at that level. The encapsulated relationships, in other words the relationships between the data source query subjects that made up that model query subject would be respected but joins to that data source query subject would be ignored. So you would have to recreate those relationships at your business level to the other model query subject level. However, adding your relationships at the model query subject level, at the business level rather can be useful to override relationships if that happens to match the reporting requirements or if you want to generate some specific query behavior.

We have another question – can you address the situation when the data modeler and the report developer are the same person? Are all the levels in the model still necessary when the report developer understands the database design?

Different schools of thought on this. generally speaking we advocate having multiple levels for a number of reasons to give you a layer of abstraction so that you can keep your data source view in tact and keep it as close to the structure of your data source as possible and then give you, have a business view where you can create some model, query subject, do some denormalization if that is what you need to do, add filter and calculations and so on.

So it also gives you a level of insulations so that you can retarget these model query subjects if you happen to have a change in the data source. So it gives you that level of insulation. We also generally advocate having a presentation view which can be either model query subjects or shortcuts to model query subjects. That is, again, just a rule of thumb, in a scenario where the report developer isn't necessarily the modeler.

If in a case where one person is doing both roles, then obviously you may be able to dispense with the middle layer and just have a series of model query subjects that then point to the data source and you can publish those directly. There is no real technical reason or requirement to have that. It is just in terms of organization and giving you a layer of abstraction, that gives you some insulation against change – that is typically why we recommend doing that.

Another question – why should I not change anything in the database query subjects in my import layer. What if I need to do this?

We generally recommend not changing anything in the database query subjects because that, if it is simply a select star from table, which is the default when you do a straight import, it is called the simple query subject as far as a query engine is concerned. So it doesn't need any additional metadata. If you modify that the query engine can sometimes require additional metadata and it won't get it from the model. So you can encounter a scenario where it has to go to the data source to fetch that information. That is what we call a metadata callback. In some cases this is really not that big of a deal. Some database vendors handle this very well and it doesn't result in much of a problem. But in some cases it does. So there are a series of techniques you can do to avoid this if you absolutely need to make modifications at the data source level.

Another reason we generally recommend not doing that is that As View SQL can be triggered, we talked about that a little bit in the presentation. Putting filters or calculations in your data source query subjects can trigger As View behavior. And lastly these, as I mentioned earlier, these data source query subjects are really global. They are the foundation upon which all of the higher level objects in your model are created. So any change you make there is a global change. So that is why we generally recommend giving yourself a layer of abstraction and making changes like that in your business view.

We have another question – what is the best practice for joining tables from different databases? Well, joining tables from different databases is something we support. You can have multiple data sources in your Framework Manager model. The caveat there really is that you are going to be joining that on the report server in most cases. If you are talking about two different schemas in the same database, then there could be an opportunity to have the database take on some of that work. But if you are talking about let's say doing an import from DB2 and another database from Oracle or two different Oracle databases, you are talking about joining that locally on the report servers. So there is a performance consideration to be considered there because you are going to crunching all of that data on the server before you get your results set.

I have another question – the fairly large model with many nested name spaces and levels of organization, how can I get an inventory or a list of all the objects in my model? We do have a feature in Framework Manager called the Model Report. It is available from the tools menu, and what it will do is what you do is you select an object or a level of organization in your model, so a name space or you can select it at the highest level of the project. And what it will do is it will run a report and give it to you in either an XML or HTML, an inventory with all of the hierarchies, the organization of the model. So all of the query subjects, the query items with their data types and so on for every object in the project, if that is what you selected, or on the name space, if that is what you happened to select.

Now, it is kind of in a, it may not be tailored directly to the format you are looking for so we do have the option of specifying an XSLT file in the FM.INI file. So if you wanted a filtered view or you wanted to control how that report looks you have an option of writing an XSLT file and then referencing it in the FM.INI file and we will generate the report in the format that you are looking for.

I have another question – is it possible to copy a few objects like query subjects, source table joins from one model to another using Framework Manager? If not, is there a better way to do it?

You do have copy and paste functionality in Framework Manager so you should be able to simply copy and paste objects over. I would say the best way to do that would be to do an import from another model. Through copying and pasting you would have to recreate the data source in the new model, so you would have to make sure that that was there and all of the required objects were there. The supported way to do that or the recommended way to do that would be to do a metadata import and then select the original Framework Manager model as the source to use when you are doing the import. And that should give you the ability to select what it is you want to import from that model and it will look after taking the necessary objects for you.

Next question, how do governors work when a query is going to return more than the specified limits if the query on the DB terminated? Yes, the way this should work is you set the governors in Framework Manager and that is at the package level. So those governors are set at the package level. There are different governors you can set, limit the number of rows and so on. What it will do is it will start to prepare the query and execute it and it should just stop the query whenever it reaches the limit that you set. I don't think it has a way of knowing in advance how many records it is going to return. So it will start the query and then it should just stop it whenever it gets close to the limit or whenever it realizes that it is going to exceed the limit.

Another question – is there a way to maintain versions to the packages like for models? There is versioning for the packages that you publish, that is an option when you are doing your publish. You can select how many versions you want to keep on the server. That can be pretty useful if you are publishing out and you want to make sure that you don't break anything in the reports that are accessing the old version. You simply keep a version or two on the server and then it gives you a little bit of insurance that you are not going to break anything. Having said that, the next time you open one of those reports that access the old package it is going to try and synch up with the new package, so that is something to keep in mind.

So that is available. In terms of being able to sort of archive packages on the server? No, we don't have anything right now that would let you say keep three versions of the package on the server and then write a report and indicate which version of that package you wanted to write. When you author a new report or you open a report in the studio, it is always going to use the newest version of the package that is available.

Let me go through some more questions here. Question – What is the main use of minimized SQL? So generally speaking we advise using minimized SQL because it will reduce the number of columns that you need to return for any given query. So if you create a model query subject and it is using a series of columns from let's say a product line and product type, if you are only wanting to use one column from there, using minimized SQL will let you just query and return that one column. If you have As View behavior triggered it is going to treat that model query subjects As View. So in other words it is going to retrieve all of those columns that are included in the model query subject even though you only were interested in the one. So it is – it is all in the interest of the economy, accessing the fewest number of objects because that typically will perform more quickly.

Another question – is there an easy way to export portions of your model out and then import this work into another model? This would be helpful if you want several smaller models for your reporting instead of one huge model.

There is a feature that is pretty similar to what the listener is referring to here. We do have the option of both merging and branching. So you can create kind of a master model and then branch off pieces of it into sub models. Generally speaking that is used for trying to model in a multi-modeler environment so you are designating different people to work on different portions of the model. And it gives you a mechanism to merge those changes back in. But there really isn't any reason why you couldn't leverage that functionality just to sort of segment your model into smaller pieces. There also is a linking in segmenting which is similar to the merge and branch in the sense that you can segment out pieces of the model, but it doesn't have the kind of merge back into the master. In the case of segmenting it is actually sub-model, if you will. So changes that are made to the original source are automatically pushed out – so there is kind of a live link there, whereas in merge and branch it is kind of a manual process to merge changes back in.

Another question—if there are more than one layer in the Framework Manager model, what is the best practice to create a relationship at the top layer. If we have a database view, intermediate view and business view – which layer is it preferred t keep the relationship between the objects?

As I said, it is really, there is no one universal answer that will be right 100% of the time. But generally speaking we recommend keeping the relationships, if possible, at the database layer. For reasons I mentioned earlier, if you have them on model query subjects you end up triggering As View behavior and so on, so as I say it is not 100% all the time way to go, but generally speaking as a rule of thumb we recommend people keep those relationships at the database layer.

Can an individual create items be secured so that some users see them and some users don't?

Yes, you do have the ability to set object level security in Framework Manager so you can assign permissions based on group's enrolls, using the security model that we have in Cognos Connection. So it is, if you select your query subject, there is a right click option to specify object security and that is the interface that you would use to set that. You have access to the name space and you can say that members of this group or members of this role, I think even down to the user level, can either have access or not have access to those objects.

Another question – how to automatically detect changes made on the data source columns in FM?

So I think the question really is how do we detect if the underlying data source has changed. What you would do is you would have to, I believe if you clicked the name space or the database of your name space and did verify model, it should try and test each one, do a quick test on each one of those data source query subjects and make sure that the projection list still matches the underlying data source. So you would see, if there were changes to the data source you would see, either if it was an additive change you would see an additional column appear in our data source query subject and if it was a case where the name of the column in your table in the database changed, you would see a broken icon because it wouldn't be able to resolve that.

So, you could do it on a query subject by query subject level as well because you just verify and it would tell you whether it was still valid or not. Because the data source query subject is generally speaking a select [star] from cable, that is how it just returns all of the rows – or excuse me – all of the columns.

Similar functionality for model query subjects. If you change something in the query item in your data source query subject, you can verify model ship, pick that up as well, or you could do it on a case by case basis by verify query subject.

Question – does creating a query subject using your own SQL have any effect on the performance? When you are creating a data source query subject you hand type in the SQL – by default when you do your import it will be select star from table. You can modify that. You have a couple of options here. You can do native SQL or you can enforce what we call Pass through SQL. Pass through SQL is typically used in cases where there is a specific database vendor, specific functionality that you want to leverage and our query engine may not be able to resolve it. So you want to basically pass it through to the data source without us touching it. So there are ways to do that. I think it is double curly braces around the syntax in the query subject definition.

It is definitely outlined in the – I think it is in the Framework Manager User's Guide. I was just looking at it the other day. So I think it is the curly braces, double curly braces on either side of the SQL statement. But definitely it is covered in the Framework Manager User's Guide, so you can get the syntax from there.

Question – when should we consider using DMR versus a cube?

Good question. Really a cube is really a static set of data at a moment in time. So you build your cube and because it is already pre-built the records are already there, the aggregation is already done. For reporting purposes generally speaking are going to be very fast. Having said that, because it is static, if you have a requirement to get sort of updated results on a regular basis it may not be the best approach for you because it does require the maintenance of rebuilding the cube periodically. So if it was a case where you needed records, data that is up to date as of today, that might mean building your cube every day. So that could be an additional maintenance task that you didn't want to take on. In terms of DMR, because we are sort of spinning the cube in memory and then querying that cube, the performance is going to be slower than a cube but it is actually accessing the data warehouse or the data source sort of as you are running it. So the results that you are getting, your reports will be as up to date as the reporting data sources are. So generally speaking that is one of the main benefits of using DMR or a cube.

The other thing is, with a cube you are talking about a static file that is sitting on the server. So there are some, depending on the platform you are on there are some limitations around that; f you get into cubes that need to be over 2 Gigs let's say, that could be an issue. So DMR has the advantage there as well. There really isn't any – you can hit larger data sets than a cube; you are not limited to just having a cube.

Of course, with the performance issue I mentioned earlier with DMR, we spinning a cube in memory as you are running your reports. The larger the data set, generally speaking you are going to see the reports, the processing time get longer as well. So not to say that DMR has sort of an infinite limit here.

Okay, how to export the metadata such as screen tips for selected name space?

You do have the ability to do an export of the string properties for query items. So in other words the names, the screen tips, all of the string type properties that you would have for different query items. Really the way that functionality was implemented was to ensure that people who want to use multi-lingual models have the ability to basically export those strings so that they can work with them in an easier interface than having to type in the multi-lingual strings by scrolling across in that window. So you can export that to I believe it is CSV that you can open in Excel, so you would an Excel spreadsheet and then you could add in the other names that you want there. So you could leverage that ability to get an Excel spreadsheet with all of the names and screen tips and descriptions of your query items.

I think that that is definitely under one of the menu items so if you select name space and then went to – I bet it is under tools or actions – you would be able to export that to Excel.

What information does Model Advisor display beyond the current messages thrown out when a Framework Manager model is saved?

Well, Model Advisor is going to, depending on the configuration that you have, you can configure it to look for specific things or ignore specific things. So what Model Advisor will do is it will apply our best practices rules for things like relationships, cardinality, usage, things like that. So if you don't want to see all of that you can pick specific topics that you are interested in and then it will export those messages accordingly. It gives it in a really nice kind of organization because it gives you diagrams and you can launch out to help directly from there. So, when you are doing a save it is going to try and verify the model and just make sure that it is a valid Framework Manager model, you have a data source connection, things like that. When you are doing Model Advisor it is actually doing a much more analytical pass through the model in identifying potential problems.

Something I mentioned in the slides was not all of these issues have to be addressed. These are just areas that are worth a second look. So when it alerts you about something it is a good idea to go and have a look and just make sure that it is in deed what you intended to do there.

Next question – what is the best practice when an upgrade of the model is needed and customer reports need to be updated?

Well there are really two pieces to that. Generally speaking our upgrade work flow is to deploy from a pervious version so you deploy your packages and your reports from a previous version to your new serer, so let's say going from 8.3 to 8.4. You do deployment, import from 8.3, import the deployment archive into 8.4 and by virtue of doing that, that should update the metadata and the reports on the server.

Your Framework Manager model should also be upgraded with means you should open that model in version 8.4 Framework Manager. And then we generally recommend republishing that model once you have upgraded it in 8.4. Until you do that you are not going to be able to leverage any of the new functionality. So it will still work in the server in kind of a compatibility mode but our general recommendation is once you have upgraded your model in Framework Manager you should publish it out again. Like any upgrade, you should really do this in a development environment and then when you are satisfied that things have upgraded correctly and they are all at the most recent version then you move it over to your QA or your UAT environment and then from there to production.

Next question – is there versioning capability in Framework Manager?

There is in 8.4 and previous versions there is native support for source control. We have support for [Visual Source Safe] and [CVS] in 8.4 and all previous versions of Framework Manager. Having said that, we did issue a deprecation notice in 8.4 about our native support for those source control systems. So as of the next major release of Framework Manager we won't have native support for those source control vendors.

We do, however, have a proven practices document, it is on our proven practices site and it describes how you would leverage any existing source control system. So whatever you happen to be using, a Microsoft tool or whatever, they should all work equally well using this approach. It identifies what files you would need to check in and check out and how you would do that.

So as I say, as of the next major release we won't have the native source control support, but we do have a proven practices document. It is available on the website that you can use as kind of a guideline for integrating with whatever source control system you are using.

Can you give a brief discussion on settings and governors?

Governors really, as I mentioned earlier, are set on a package level so they are set there to sort of give hints to the query engine on how to react in certain situations. It is a good idea for a modeler if you are talking about a massive data set, it would be a good idea to set a query governor to limit the number of rows returned. You wouldn't want someone in query studio let's say to be able to drag query items onto the canvas and execute a query that might return 5 million rows or 5 billion rows even. So it is a good practice to set kind of realistic limits. Some common limits that I see are limit it to say 500,000 rows or 1 million rows at the maximum; query times is another one. You can set the query duration so that if a query is going to run longer than ten minutes, stop. Generally speaking those two settings go hand in hand. If you are going to be returning more than a million rows it is likely going to take more than 10 minutes. So you probably don't want someone doing that. So it is a good idea to set query governors like that in Framework Manager just to make your system administrator happy so that he doesn't see a real hit on the server.

How can we set drill ups and drill downs?

Well, in terms of modeling you are talking about analysis now. So what you are talking about there would be dimensionally modeled relational source. So what you would do is you would do your regular modeling because the foundation of any dimensionally modeled model is a good relational model. So the same best practices would apply. So do your import, create your model query subjects, organize your star schema and then in your dimension query subjects organize levels. Once you have your star schema created directly, we do have a nice automatic way of converting those to regular dimensions. It is from the right click menu; you can select I think, if you just organize them in the name space you can right click the name space and it will convert the dimension query subjects into regular dimensions and the fact query subject into a measured dimension.

From there you will have to do, you may have to do a little more work making sure that we got the levels correct. But with a little bit of tweaking you should be in a pretty good position to publish out and then your users would be able to do drill ups and drill downs in a tool like Query Studio or Analysis Studio.

So the drill up/drill down functionality comes for free in the studios once you have got a dimensionally modeled model.

There are a couple of questions about this presentation and whether or not it is going to be available. My understanding is there is going to be a thank you email sent out next week and it should have a link to this presentation so that you can refer to it. If anyone would like to have a set of the slides, I would be glad to send that as well. I would have to send it in PDF so I don't the movies would comes across, but as far as the content is concerned, if anyone is interested in getting a PDF copy of the presentation, I would be glad to send that as well. So you can just reach out to me directly.

One more question – I have a rather large model and I find it takes longer to publish. What is Framework Manager doing that causes this delay?

What we are doing when you do a publish is we are going to analyze the entire model and make sure that we include all of the objects that you need. So you can imagine when you are creating a package and let's say you have your database view and you have your business view and you have your presentation view, when you create your package you select your presentation, your name space – it has let's say 30 or 40 query subjects, that's all you want, or short cuts rather, and that is what you want to publish at the studios.

So what we are going to do is make sure that we also include the underlying model query subjects and the necessary database query subjects. So you are not going to want the report authors to see them and we are not going to show them to them, but of course the query engine needs them in order to generate the queries. So we need to scan through the model and figure out what the dependencies are. So this shortcut points to this model query subject, points to this data source query subject. And we will sort of scan through that and unwind the model and figure out what objects are needed and we will include them but keep them hidden.

So the larger the model the more time and resources it takes to scan through the model to figure that out. So you will see if you have a very large model in the 20 to 30 megabyte range you may see publish take several minutes whereas if you are using a small model, publish could happen. You could do a publish in a few seconds.

I think we are going to stop there. There are a few more questions but some of them are really of the type that I think kind of a back and forth dialogue would be in order. So I know there are some questions here that I didn't get to and I apologize for that, we are running out of time. So what I would like to do is just invite anyone who has a question that we didn't cover, please do send me an email and I would be glad to follow up with you offline and we can hopefully address your question.

So I would just like to thank everyone for joining. I enjoyed answering these questions and hopefully you all found the session worthwhile. And as I said, please do drop me a note by email if there are any questions that I did not cover today.