Proven Practice

Writing Efficient Queries on OLAP Data Sources

Product(s): IBM Cognos 8

Area of Interest: Report Design

Copyright

This document is maintained by the Best Practices, Product and Technology team. You can send comments, suggestions, and additions to cscogpp@ca.ibm.com .

Contents

1 Introduction

1.1 Purpose

Just as we have developed good habits for writing relational queries through understanding SQL and databases and databases optimizers, we also have to understand the nature of OLAP queries and OLAP providers and ensure we develop good habits for the creation of OLAP queries. This document discusses several OLAP query subjects dealing with OLAP query performance and optimization. The application of the habits ingrained in this document should help you build better, faster, more efficient OLAP queries/

This document is broken down by the following major sections:

1. Optimizing queries for OLAP Cross join behaviour
2. Impact of scalar functions on OLAP query performance
3. Long Name Filters
4. Explicit Aggregate Usage

1.2 Applicability

This discussion applies primarily to IBM Cognos PowerCubes and may apply to other OLAP sources[1].

1.3 Exclusions and Exceptions

This discussion does not apply to RDBMS or DMR[2].

This document does not cover summary filters at this time.

Please see the IBM Cognos 8 Read Me documentation regarding limitations to filters on report items that also leverage smart prompts in the filter expressions.

[1] Capabilities and features vary between OLAP providers. Some constructs may be available in IBM Cognos 8 but will be processed locally rather than by the OLAP data source due to limitations within the OLAP source. Such local processing will increase the overhead of query processing.

[2] Dimensionally Modelled Relational – RDBMS with a dimensional Framework Manager model

2 Optimizing Queries for OLAP Cross Join Behaviour

OLAP[3] and relational queries[4] behave very differently when items are nested in a crosstab:

- Relational queries will populate the edge items only where there is supporting data
    - A report with Retailer name and Order method nested on the crosstab rows will show only Retailer name /Order method combinations for which there is data that matches the relationship criteria between the tables involved[5].
- OLAP queries which nest member sets from the same hierarchy on the edge of a crosstab will not cross join of the members from each set.
    - A report with Product line and Product type nested on the crosstab rows will show the Product line/Product type combinations as defined in the hierarchy.
- OLAP queries which nest member sets from different hierarchies on the edge of a crosstab will cross join of the members from each set.
    - A report with Retailer name and Order method nested on the crosstab rows will show all Retailer name /Order method combinations regardless of whether there is data.

To ensure we fully understand these points, let's compare and contrast the results of a relational and OLAP query examples to see the behaviour difference.

We will compare two very similar reports, a relational query created on GO Sales and Retailers relational database, and an OLAP query created on Great Outdoors Company cube to see this difference and then see how we can optimize the OLAP query.

[3] Not including SAP
[4] Assuming a non-dimensional Framework Manager model
[5] Modeling relational data with outer joins will give you the unmatched records similar to true OLAP data sources. However, there are performance implications due to the outer joins.

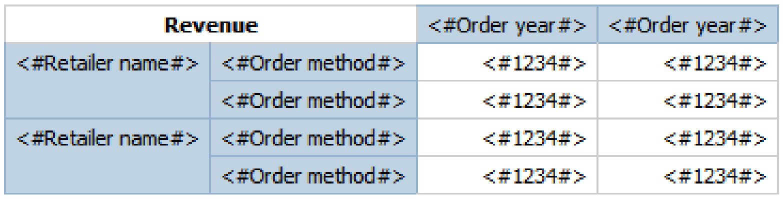
2.1 Comparing Relational and OLAP Query Behaviour

2.1.1 Relational Queries Do Not Cross Join

Let's first look at a relational query.

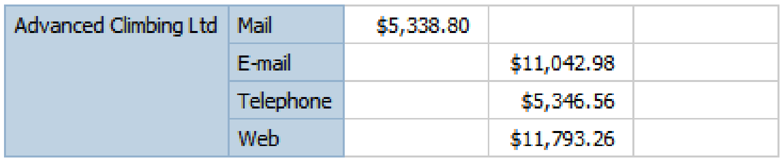
Open Report Studio with the GO Sales and Retailers package.

Create this crosstab:



Revenue		<#Order year#>	<#Order year#>
<#Retailer name#>	<#Order method#>	<#1234#>	<#1234#>
	<#Order method#>	<#1234#>	<#1234#>
<#Retailer name#>	<#Order method#>	<#1234#>	<#1234#>
	<#Order method#>	<#1234#>	<#1234#>

Run the report and page down to find Advanced Climbing Ltd:



Advanced Climbing Ltd	Mail	$5,338.80		
	E-mail		$11,042.98	
	Telephone		$5,346.56	
	Web		$11,793.26	

We see rows in the crosstab only *for which there is data*. There are only 4 Order methods indicating that there were no sales for Advanced Climbing Ltd by the other Order methods.

Close the report. There is no need to save it.

2.1.2 OLAP Queries Do Not Cross Join Member Sets from the Same Hierarchy

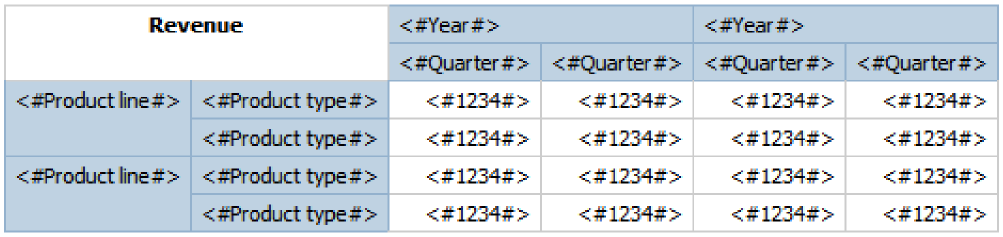
Before we look at how to optimize OLAP queries, we must also understand when the nested member sets are not cross joined.

When member sets from the same hierarchy are nested, the hierarchical structure determines the relationships between the members and no cross join is produced.

Let's take a look using Product line and Product type which are both in the Products hierarchy.

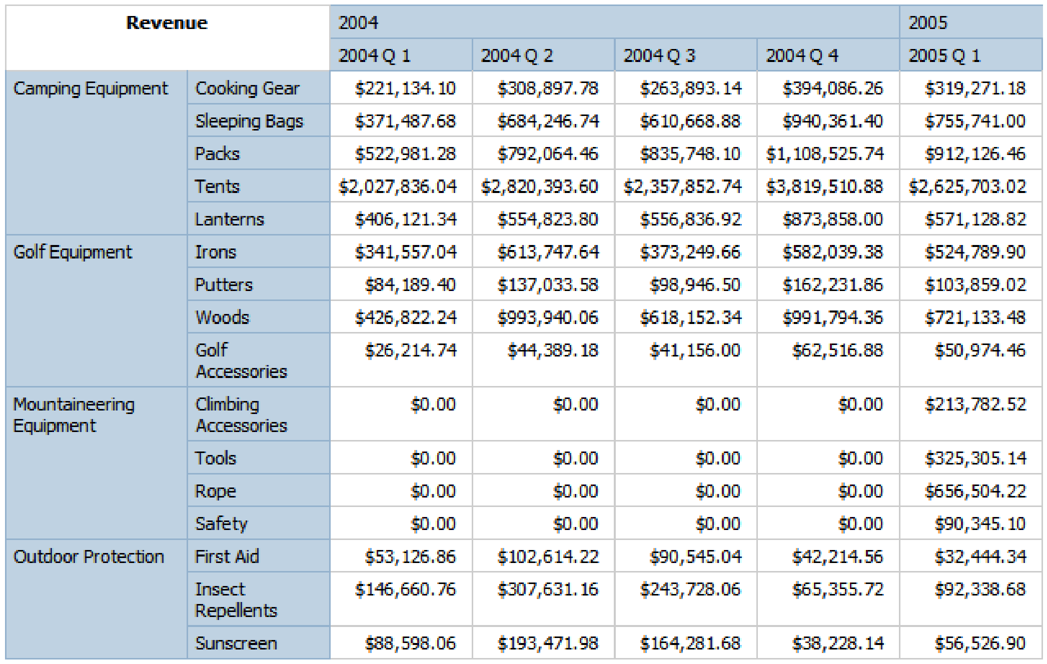
Open Report Studio with the Great Outdoors Company package.

Create a new crosstab report as so:



Revenue		<#Year#>		<#Year#>	
		<#Quarter#>	<#Quarter#>	<#Quarter#>	<#Quarter#>
<#Product line#>	<#Product type#>	<#1234#>	<#1234#>	<#1234#>	<#1234#>
	<#Product type#>	<#1234#>	<#1234#>	<#1234#>	<#1234#>
<#Product line#>	<#Product type#>	<#1234#>	<#1234#>	<#1234#>	<#1234#>
	<#Product type#>	<#1234#>	<#1234#>	<#1234#>	<#1234#>

Run the report:



Revenue		2004				2005
		2004 Q 1	2004 Q 2	2004 Q 3	2004 Q 4	2005 Q 1
Camping Equipment	Cooking Gear	$221,134.10	$308,897.78	$263,893.14	$394,086.26	$319,271.18
	Sleeping Bags	$371,487.68	$684,246.74	$610,668.88	$940,361.40	$755,741.00
	Packs	$522,981.28	$792,064.46	$835,748.10	$1,108,525.74	$912,126.46
	Tents	$2,027,836.04	$2,820,393.60	$2,357,852.74	$3,819,510.88	$2,625,703.02
	Lanterns	$406,121.34	$554,823.80	$556,836.92	$873,858.00	$571,128.82
Golf Equipment	Irons	$341,557.04	$613,747.64	$373,249.66	$582,039.38	$524,789.90
	Putters	$84,189.40	$137,033.58	$98,946.50	$162,231.86	$103,859.02
	Woods	$426,822.24	$993,940.06	$618,152.34	$991,794.36	$721,133.48
	Golf Accessories	$26,214.74	$44,389.18	$41,156.00	$62,516.88	$50,974.46
Mountaineering Equipment	Climbing Accessories	$0.00	$0.00	$0.00	$0.00	$213,782.52
	Tools	$0.00	$0.00	$0.00	$0.00	$325,305.14
	Rope	$0.00	$0.00	$0.00	$0.00	$656,504.22
	Safety	$0.00	$0.00	$0.00	$0.00	$90,345.10
Outdoor Protection	First Aid	$53,126.86	$102,614.22	$90,545.04	$42,214.56	$32,444.34
	Insect Repellents	$146,660.76	$307,631.16	$243,728.06	$65,355.72	$92,338.68
	Sunscreen	$88,598.06	$193,471.98	$164,281.68	$38,228.14	$56,526.90

(Etc…)

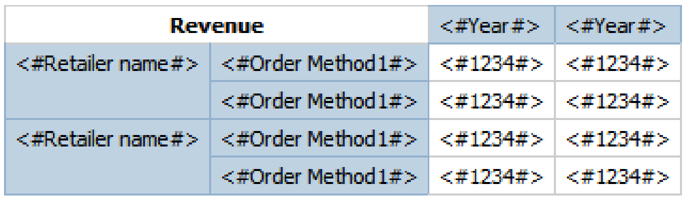
Note that the (Product line & Product Type) and (Year & Quarter) are NOT cross joined, rather, the natural structure is used. A cross join is only used when the members come from different hierarchies.

2.1.3 OLAP Queries Cross Join Member Sets from Different Hierarchies

Let’s look at an OLAP query to see how the member set cross joins produce very different results.

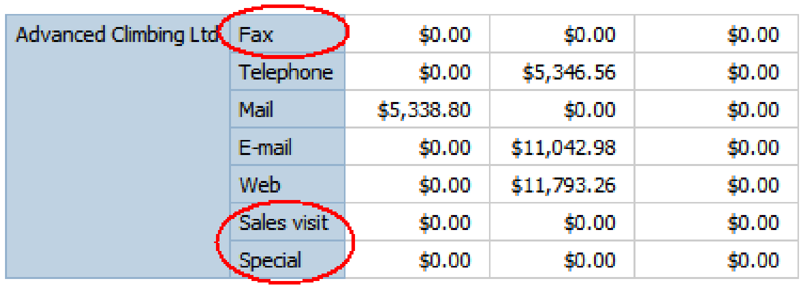
Open Report Studio with the Great Outdoors Company package.

Create this crosstab.



Revenue		<#Year#>	<#Year#>
<#Retailer name#>	<#Order Method1#>	<#1234#>	<#1234#>
	<#Order Method1#>	<#1234#>	<#1234#>
<#Retailer name#>	<#Order Method1#>	<#1234#>	<#1234#>
	<#Order Method1#>	<#1234#>	<#1234#>

Run the report and page down to find Advanced Climbing Ltd:



Advanced Climbing Ltd	Fax	$0.00	$0.00	$0.00
	Telephone	$0.00	$5,346.56	$0.00
	Mail	$5,338.80	$0.00	$0.00
	E-mail	$0.00	$11,042.98	$0.00
	Web	$0.00	$11,793.26	$0.00
	Sales visit	$0.00	$0.00	$0.00
	Special	$0.00	$0.00	$0.00

Note that we see <u>all</u> Order methods *even those with no data*.

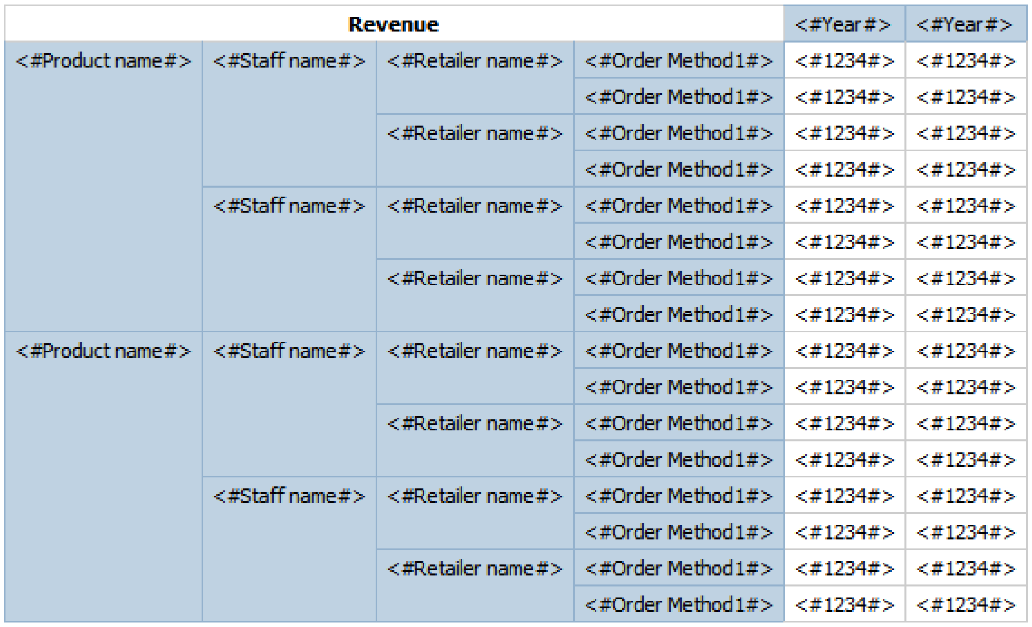
This is the correct result according to the report you created. The results differ due to the differences between relational and OLAP queries.

Without proper filtering, this can lead to large reports. The Go Sales and Retailers RDBMS report is as long as the data (675 rows). As you filter the report, you’ll see fewer rows of data.

The Great Outdoors Company OLAP report is always the same length - 798 rows. 798 is the cross join of the number of Retailers (114) times the number of Order methods (7).

Unless you filter <u>the member sets</u> you’ll always see the same number of rows of data regardless of what other filters you apply.

While the difference in report sizes does not seem excessive, consider this report:



Revenue				<#Year#>	<#Year#>
<#Product name#>	<#Staff name#>	<#Retailer name#>	<#Order Method1#>	<#1234#>	<#1234#>
			<#Order Method1#>	<#1234#>	<#1234#>
		<#Retailer name#>	<#Order Method1#>	<#1234#>	<#1234#>
			<#Order Method1#>	<#1234#>	<#1234#>
	<#Staff name#>	<#Retailer name#>	<#Order Method1#>	<#1234#>	<#1234#>
			<#Order Method1#>	<#1234#>	<#1234#>
		<#Retailer name#>	<#Order Method1#>	<#1234#>	<#1234#>
			<#Order Method1#>	<#1234#>	<#1234#>
<#Product name#>	<#Staff name#>	<#Retailer name#>	<#Order Method1#>	<#1234#>	<#1234#>
			<#Order Method1#>	<#1234#>	<#1234#>
		<#Retailer name#>	<#Order Method1#>	<#1234#>	<#1234#>
			<#Order Method1#>	<#1234#>	<#1234#>
	<#Staff name#>	<#Retailer name#>	<#Order Method1#>	<#1234#>	<#1234#>
			<#Order Method1#>	<#1234#>	<#1234#>
		<#Retailer name#>	<#Order Method1#>	<#1234#>	<#1234#>
			<#Order Method1#>	<#1234#>	<#1234#>

The relational equivalent query can never exceed the number of rows in the database (11,000 or so aggregated to the Order method level).

This OLAP query will cross join the members resulting in a very large crosstab: 115 Product names * 104 Staff name * 114 Retailer names * 7 Order methods = 12,847,800 rows.

> Note: If you intend to try the above report please do not do so on a Production system.

Note that most of the rows returned by the report have no data and will have zeros in the cells. The solution to this problem is to suppress the rows that have no data – zero suppression.

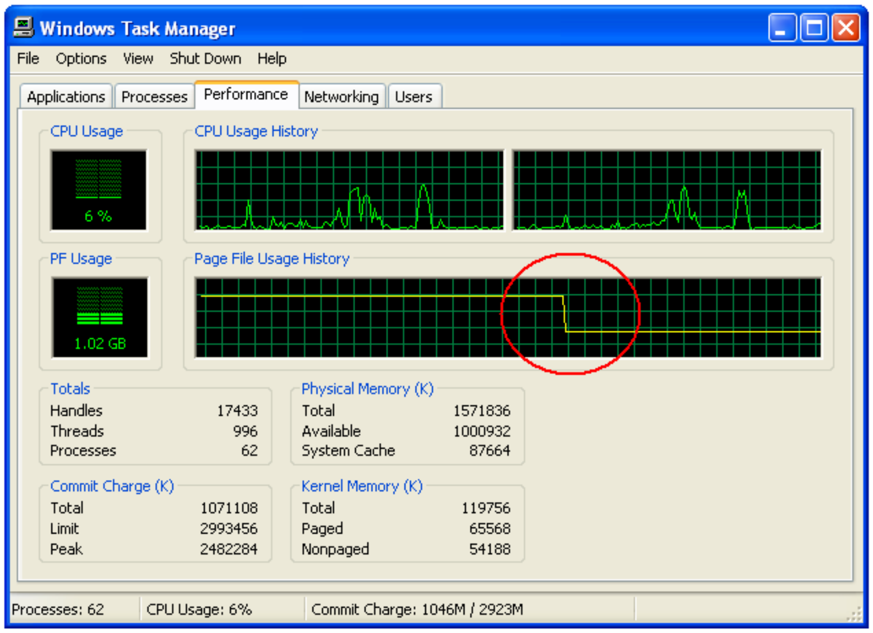
Close Report Studio. There is no need to save the report.

2.1.3.1 OLAP Crosstab Reports are Slow or Fail

As cross joins can easily result in very large reports, those reports typically show symptoms such as:

1. Report runs a very, very long time for no apparent reason.

2. Report fails with an RSV-BBP-0022 The absolute affinity request 'asynchWait_Request' failed, the requested session does not exist often accompanied by the BIBusTKServerMain (report server) consuming a precipitous amount of memory followed by a sudden drop in memory usage as the process fails:



Totals	
Handles	17433
Threads	996
Processes	62

Physical Memory (K)	
Total	1571836
Available	1000932
System Cache	87664

Commit Charge (K)	
Total	1071108
Limit	2993456
Peak	2482284

Kernel Memory (K)	
Total	119756
Paged	65568
Nonpaged	54188

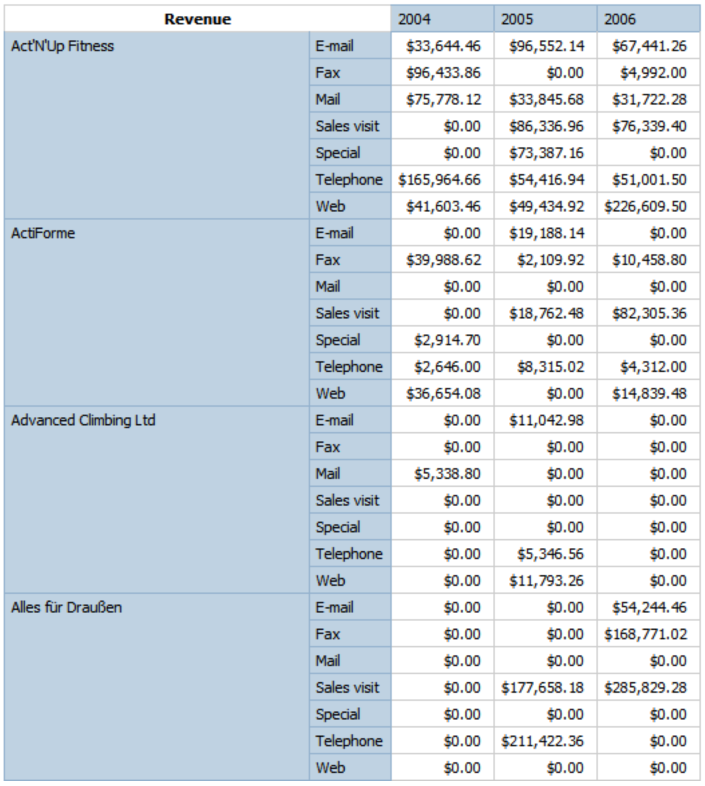
Processes: 62 CPU Usage: 6% Commit Charge: 1046M / 2923M

2.2 Creating Efficient OLAP Queries

The key to creating efficient queries is to limit the number of items involved in the cross joins. When constructing member sets, your goal should always be to include only the minimum number of required members.

For example, we know there are 114 Retailer names and 7 Order methods in the data source which may yield a report of 798 rows. If we include only those Order methods through which the Retailers actually sold, we can reduce the report size substantially.

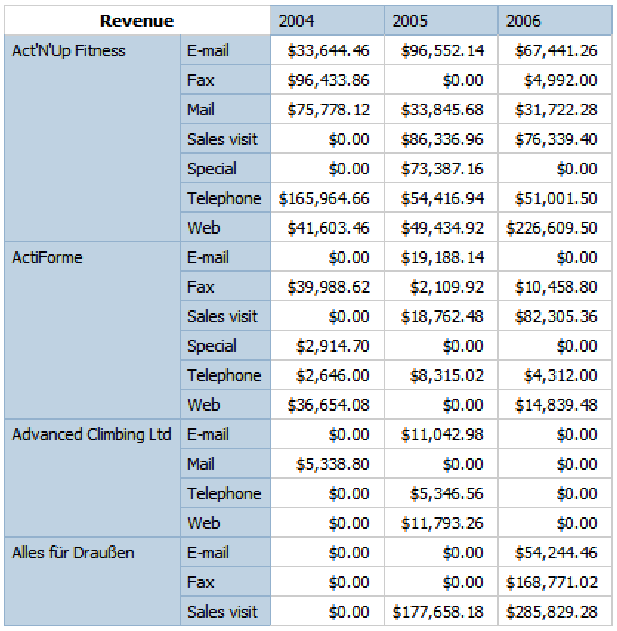
Without filtering the member sets prior to the cross joins, your queries will always operate like this: {104 Retailer names} * {7 Order Methods) = 798 rows as so:



Revenue		2004	2005	2006
Act'N'Up Fitness	E-mail	$33,644.46	$96,552.14	$67,441.26
	Fax	$96,433.86	$0.00	$4,992.00
	Mail	$75,778.12	$33,845.68	$31,722.28
	Sales visit	$0.00	$86,336.96	$76,339.40
	Special	$0.00	$73,387.16	$0.00
	Telephone	$165,964.66	$54,416.94	$51,001.50
	Web	$41,603.46	$49,434.92	$226,609.50
ActiForme	E-mail	$0.00	$19,188.14	$0.00
	Fax	$39,988.62	$2,109.92	$10,458.80
	Mail	$0.00	$0.00	$0.00
	Sales visit	$0.00	$18,762.48	$82,305.36
	Special	$2,914.70	$0.00	$0.00
	Telephone	$2,646.00	$8,315.02	$4,312.00
	Web	$36,654.08	$0.00	$14,839.48
Advanced Climbing Ltd	E-mail	$0.00	$11,042.98	$0.00
	Fax	$0.00	$0.00	$0.00
	Mail	$5,338.80	$0.00	$0.00
	Sales visit	$0.00	$0.00	$0.00
	Special	$0.00	$0.00	$0.00
	Telephone	$0.00	$5,346.56	$0.00
	Web	$0.00	$11,793.26	$0.00
Alles für Draußen	E-mail	$0.00	$0.00	$54,244.46
	Fax	$0.00	$0.00	$168,771.02
	Mail	$0.00	$0.00	$0.00
	Sales visit	$0.00	$177,658.18	$285,829.28
	Special	$0.00	$0.00	$0.00
	Telephone	$0.00	$211,422.36	$0.00
	Web	$0.00	$0.00	$0.00

While we could use a detail filter to remove the zero value rows (Order methods) from the final result the intermediate result will still create the entire cross join result.

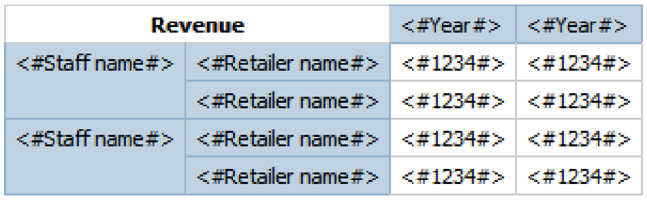
Were we able to filter out Order methods prior to executing the cross join, we'd only create the intermediate result with the smallest number of members:



Revenue		2004	2005	2006
Act'N'Up Fitness	E-mail	$33,644.46	$96,552.14	$67,441.26
	Fax	$96,433.86	$0.00	$4,992.00
	Mail	$75,778.12	$33,845.68	$31,722.28
	Sales visit	$0.00	$86,336.96	$76,339.40
	Special	$0.00	$73,387.16	$0.00
	Telephone	$165,964.66	$54,416.94	$51,001.50
	Web	$41,603.46	$49,434.92	$226,609.50
ActiForme	E-mail	$0.00	$19,188.14	$0.00
	Fax	$39,988.62	$2,109.92	$10,458.80
	Sales visit	$0.00	$18,762.48	$82,305.36
	Special	$2,914.70	$0.00	$0.00
	Telephone	$2,646.00	$8,315.02	$4,312.00
	Web	$36,654.08	$0.00	$14,839.48
Advanced Climbing Ltd	E-mail	$0.00	$11,042.98	$0.00
	Mail	$5,338.80	$0.00	$0.00
	Telephone	$0.00	$5,346.56	$0.00
	Web	$0.00	$11,793.26	$0.00
Alles für Draußen	E-mail	$0.00	$0.00	$54,244.46
	Fax	$0.00	$0.00	$168,771.02
	Sales visit	$0.00	$177,658.18	$285,829.28

Note that the intermediate result set has dropped in size from 28 rows to 20 rows just on these 4 Retailers. Applying this across the entire report will result in 763 rows, a savings of 35 rows.

Let's consider a more complex example with nested items on the rows, Staff name and Retailer name:



Revenue		<#Year#>	<#Year#>
<#Staff name#>	<#Retailer name#>	<#1234#>	<#1234#>
	<#Retailer name#>	<#1234#>	<#1234#>
<#Staff name#>	<#Retailer name#>	<#1234#>	<#1234#>
	<#Retailer name#>	<#1234#>	<#1234#>

The final report we want is just 478 rows and would include only the Retailer names associated with a given Staff name. The intermediate cross join result could be as 11,856 rows.

When dealing with more spare data, this can result in savings of thousands or even millions of rows that need to be retrieved or evaluated for some detail filters (as we'll see in the customer use case later in this document). Clearly we need to ensure that the intermediate result set contains only the minimum set of members that are required to produce the final result set.

2.2.1 Filtering Overview

There are many filtering options for OLAP queries, far more than you might have considered. We will take a look at a number of these options and discuss them from the perspective of performance implications and then we will discuss how to apply these filter options to avoid the large intermediate result sets.

We are all used to ensuring filters on relational data are efficient, filtering on indexed items for example, and there similar rules that apply to OLAP

The goals of any filter operation should be both to:

- Ensure that the user gets the correct data and
- Ensure that the filtering operation is carried out in an efficient manner.

To that end, we will look at some options available to you based on what you are actually trying to accomplish.

2.2.2 Filtering Members/Member Sets

2.2.2.1 Filtering Members That Appear in the Report

When determining which members display on an edge, there is wide variety of options that provide *good performance*.

1. Hierarchy reference – returns the set of all members in the hierarchy

   Ex. [great_outdoors_company].[Products].[Products]

   This is only a good option if you want a single data item with members from multiple levels of the hierarchy.
2. Level reference - returns the set of all members in the level

   Ex. [great_outdoors_company].[Products].[Products].[Product line]

   As we will see later, this set can also be filtered using detail filters.
3. Individual members – drop individual members into report

   Ex. [Golf Equipment]

   This is a very fast option as no other operation is necessary to determine which member is required.
4. Dimension functions - returns the specified set of members
   a. Set( ) function

      Ex. set([Golf Equipment], [Camping Equipment])
   b. Children( ) function

      Ex. children([Camping Equipment])
   c. Descendants( ) function

      Ex. Descendants(([Golf Equipment], 2)
   d. Etc.

      Many other dimensional functions exist

   These options are all well supported by the OLAP providers.
5. Detail filter using members

   Ex. [Product line] = [Camping Equipment]

   Note: [Product line] is a level and [Camping Equipment] is a member

   The combination of a level reference in an edge with a detail filter is both popular and efficient.

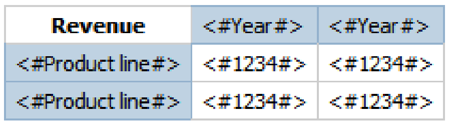
   You can also combine detail filters such as this with the other edge definitions we saw above.

These filtering approaches can be used as required to get the result you desire without concern of introducing performance issues. You can freely choose from these approaches based on your preference and report requirement.

2.2.2.2 Filtering Members That Do Not Appear in the Report

The effect of filters on member sets that do not appear in the report is to reduce the cell values without changing the edge definitions (which members appear on the edges). By definition, these are slicers.

Consider this report:



Revenue	<#Year#>	<#Year#>
<#Product line#>	<#1234#>	<#1234#>
<#Product line#>	<#1234#>	<#1234#>

We want to filter the report (cell values) to only show Revenue sold by the Fax Order method (which does *not* appear on the report).

We could create this filter in several ways with no difference in performance:

1. A slicer using the [Fax] member
2. A detail filter using members

    Ex. [Order Method1] = [Fax]

    Note: [Order Method1] is a level and [Fax] is a member

Remember that slicers and detail filters are not the same. Detail filters will remove items from the edge where slicers will not. If you are unsure of the difference, there are training materials available from your local training office which will explain the concepts behind each area.

Again, you can choose the mechanism you are most comfortable with without concern of introducing a performance issue.

2.2.2.3 Filtering on Member Attributes

Filters of all kinds using member roleValue( ) attributes[6], intrinsic member attributes[7] or user defined attributes[8] will generally not provide fast performance. While the performance degradation will differ by data source, as a general rule these types of filters are to be avoided.

Ex. roleValue('_memberCaption', [Product line]) = 'Camping Equipment'

While these filters are still rolled into the generated MDX, the underlying OLAP providers are not generally well optimized to support them. Thus, the effect is often similar to a table scan in SQL where all of the data is read and compared to the filter value rather than an indexed retrieval where the required data is retrieved directly.

Detail filters against attributes such as Caption, Long Name and other attributes are also applied after the cross join[9] have been executed and the intermediate result created. While we will not work through an example of this, it is important to bear this in mind especially when using attributes such as Caption for purposes such as drill through filters.

Note that there are times when a caption( ) or similar filter is your only option. For example, if you are drilling through from an RDBMS based report to an OLAP report you will not be able to pass a member. In that case, passing a key or description value is your only option. That's OK, just be aware of the performance implications when designing your application.

There are also other cases where member info (MUNs) are not available. An SDK application for example has no access to MUN definitions. When trying to share the same parameter value across filters on different cubes you will have to use caption or other filters as the MUNs will differ across different OLAP cubes and providers.

Note that when filtering on captions you must be careful about uniqueness. A filter like City = 'London' may not guarantee uniqueness. Generally, you want to pass relational key values to the unique business key using a filter expression something like this:

Ex. roleValue('_businessKey', [Great Outdoors].[Years].[Years].[Year]) = ?p1?

[6] Such as roleValue('_memberCaption', [Year]) or caption([great_outdoors_company].[Years].[Years].[Year])

[7] Such as the PowerCube member attributes Caption, Long Name, Short Name, Member Description and Category Code.

[8] Such as Square Footage on a store member

[9] A OLAP filter() function will have similar performance concerns but will be performed prior to the crossjoin. With more complex filter criteria the filter function may require just as much processing as generating the entire result set via a crossjoin.

2.2.2.3.1 Generating and Caching MUNs

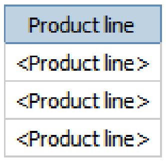
The are two solutions for dealing with MUNs outside of the Report Studio OLAP environment.

First, if the cube is built with appropriate category codes from a known business key then the MUN can be generated rather than fetched from the cube. In such a case it would be useful to understand the MUN structure so that the MUN could be generated programmatically during an RDBMS ETL process. This is beyond the scope of this paper but additional information is available on the MUN structure and how this might be accomplished. There are training materials available from your local training office which will explain the concepts behind each area.

Secondly, it is possible to store the OLAP MUNs in the RDBMS and pass the MUN from the relational report to the OLAP report on drill through. This is as simple as creating an OLAP report with the RDBMS key information and the MUN and loading this info back into the RDBMS. Let's see how we can capture the MUN info easily. We will capture the MUNs for the Product line, Product type and Product name members.

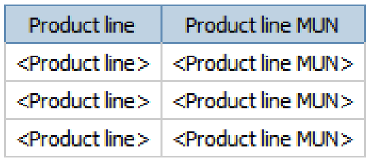
Open Report Studio with the Great Outdoors Company sample package.

Create a new list report with the Product line level:



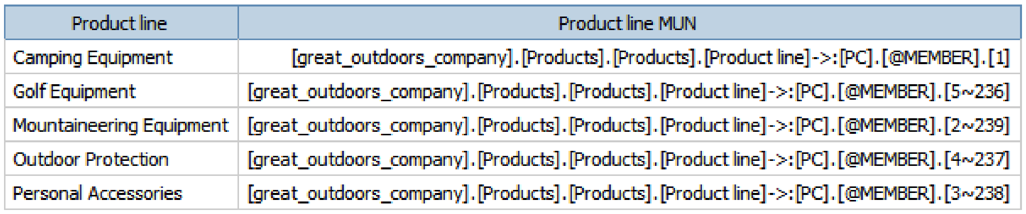
Product line
<Product line>
<Product line>
<Product line>

Drop in a query calculation called *Product line MUN* with the following expression: roleValue('_memberUniqueName', [Product line]):



Product line	Product line MUN
<Product line>	<Product line MUN>
<Product line>	<Product line MUN>
<Product line>	<Product line MUN>

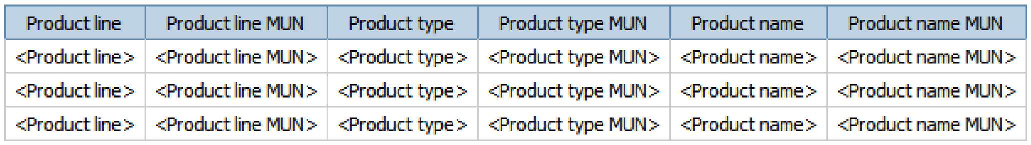
Run the report:



Product line	Product line MUN
Camping Equipment	[great_outdoors_company].[Products].[Products].[Product line]->:[PC].[@MEMBER].[1]
Golf Equipment	[great_outdoors_company].[Products].[Products].[Product line]->:[PC].[@MEMBER].[5~236]
Mountaineering Equipment	[great_outdoors_company].[Products].[Products].[Product line]->:[PC].[@MEMBER].[2~239]
Outdoor Protection	[great_outdoors_company].[Products].[Products].[Product line]->:[PC].[@MEMBER].[4~237]
Personal Accessories	[great_outdoors_company].[Products].[Products].[Product line]->:[PC].[@MEMBER].[3~238]

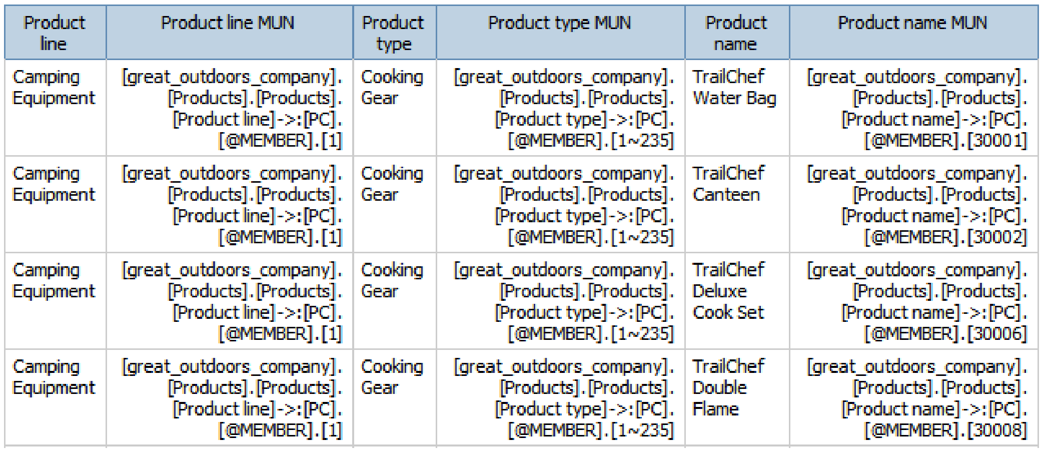
There is no need to understand the structure or meaning of the MUN at this time as we are simply caching them and not manipulating them. Suffice to say it's very meaningful to IBM Cognos 8.

Return to Report Studio and repeat these steps for the Product type and Product name levels:



Product line	Product line MUN	Product type	Product type MUN	Product name	Product name MUN
<Product line>	<Product line MUN>	<Product type>	<Product type MUN>	<Product name>	<Product name MUN>
<Product line>	<Product line MUN>	<Product type>	<Product type MUN>	<Product name>	<Product name MUN>
<Product line>	<Product line MUN>	<Product type>	<Product type MUN>	<Product name>	<Product name MUN>

Run the report:



Product line	Product line MUN	Product type	Product type MUN	Product name	Product name MUN
Camping Equipment	[great_outdoors_company].[Products].[Products].[Product line]->:[PC].[@MEMBER].[1]	Cooking Gear	[great_outdoors_company].[Products].[Products].[Product type]->:[PC].[@MEMBER].[1~235]	TrailChef Water Bag	[great_outdoors_company].[Products].[Products].[Product name]->:[PC].[@MEMBER].[30001]
Camping Equipment	[great_outdoors_company].[Products].[Products].[Product line]->:[PC].[@MEMBER].[1]	Cooking Gear	[great_outdoors_company].[Products].[Products].[Product type]->:[PC].[@MEMBER].[1~235]	TrailChef Canteen	[great_outdoors_company].[Products].[Products].[Product name]->:[PC].[@MEMBER].[30002]
Camping Equipment	[great_outdoors_company].[Products].[Products].[Product line]->:[PC].[@MEMBER].[1]	Cooking Gear	[great_outdoors_company].[Products].[Products].[Product type]->:[PC].[@MEMBER].[1~235]	TrailChef Deluxe Cook Set	[great_outdoors_company].[Products].[Products].[Product name]->:[PC].[@MEMBER].[30006]
Camping Equipment	[great_outdoors_company].[Products].[Products].[Product line]->:[PC].[@MEMBER].[1]	Cooking Gear	[great_outdoors_company].[Products].[Products].[Product type]->:[PC].[@MEMBER].[1~235]	TrailChef Double Flame	[great_outdoors_company].[Products].[Products].[Product name]->:[PC].[@MEMBER].[30008]

You can now run this report is CSV or XML format and load the info back into your RDBMS. You would first create a new column in the RDBMS to hold the MUN value for each Product line and then load the Product line MUNs matching by name. Of course, if name is not unique you need a system to ensure uniqueness. Repeat this for each data item Product type, Product name and so on that you need to have as drill through items. If an item will not be used on drill through, there is no need to cache the MUNs.

The MUN is the value that IBM Cognos 8 uses to identify a member. In other words, you need a Member Unique Name type parameter to pass the MUN value to on drill. Therefore, when creating your target OLAP drill through report, you need a filter such as [Product line] in ?p1? where [Product line] is a level. The parameter p1 will be of type Member Unique Name.

In the RDBMS parent drill report, you need to include the data item, such as Product line, and the associated MUN in the query. The MUN does not have to be visible in the report, and therefore you will have to set it as a Property of the data frame. When defining the drill through, you would pass the MUN to the target OLAP report rather than the relational data item.

It is important to understand that the MUN is unique to a given OLAP data source. If you have two PowerCubes with the 'same' structure, the members may have different MUNs[10]. If you have a PowerCube and an MSAS cube with the 'same' Product line, they will have different MUNs. Therefore, if you wish to drill to different cubes you will have to store MUNs for each cube. Of course, this does limit the practicality of this solution.

As the MUNs are generally stable across time[11], you do not need to update the existing MUNs. You do, however, need to account for new members or members that move within the structure when the OLAP cube is updated. Therefore, each time the cube is re-built, the build process must include the subsequent export and import of the MUNs to the RDBMS.

Admittedly, this is not a perfect solution and must be used judiciously. If you can pass the Caption or Business Key with acceptable performance that approach is obviously much less work to setup and maintain. You'll probably keep this technique in your back pocket for more advanced drill through scenarios.

If the cube is built with appropriate category codes from a known business key then the MUN can be generated rather than fetched from the cube. In such a case it would be useful to understand the MUN structure so that the MUN could be generated programmatically during an RDBMS ETL process. There are training materials available from your local training office which will explain the concepts behind each area.

[10] If the category codes are the same then the portable portion of the MUN should be identical.

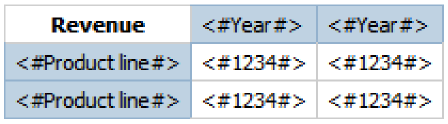
[11] The MUNs generated by Transformer with the tilde should not be considered stable as these are manufactured to provide uniqueness. If the categories within the Transformer cube are regenerated then the category codes could potentially be different depending on the order and content of the source data.

2.2.3 Filtering Measures

Measure filters serve to determine which members are used in the query by testing the value of a measure. The most common measure filters are detail filters and filter( ) function expressions.

Detail filters are applied to the cell values of the crosstab. In other words, they filter values within the context of the lowest level item on each edge.

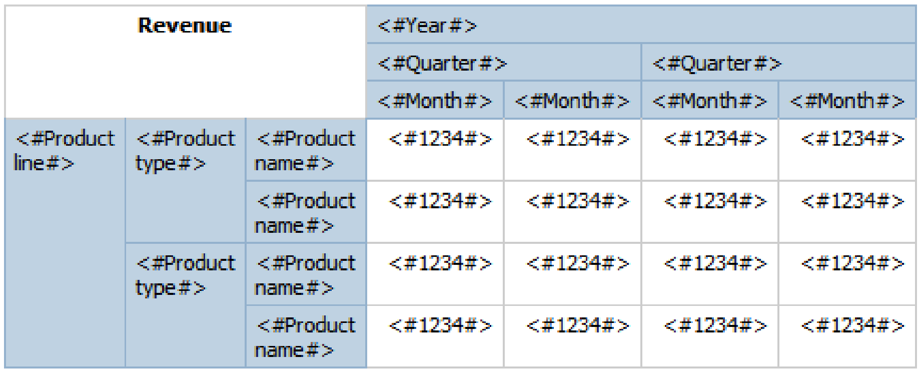
For example, this crosstab:



Revenue	<#Year#>	<#Year#>
<#Product line#>	<#1234#>	<#1234#>
<#Product line#>	<#1234#>	<#1234#>

The detail filter [Revenue] > 124 is applied to the intersection of the Product line and Year (and Revenue) items.

Consider this crosstab:



Revenue			<#Year#>			
			<#Quarter#>		<#Quarter#>	
			<#Month#>	<#Month#>	<#Month#>	<#Month#>
<#Product line#>	<#Product type#>	<#Product name#>	<#1234#>	<#1234#>	<#1234#>	<#1234#>
		<#Product name#>	<#1234#>	<#1234#>	<#1234#>	<#1234#>
	<#Product type#>	<#Product name#>	<#1234#>	<#1234#>	<#1234#>	<#1234#>
		<#Product name#>	<#1234#>	<#1234#>	<#1234#>	<#1234#>

The detail filter [Revenue] > 124 is (still) applied (only) to the intersection of Product name and Month (and Revenue) items only and not intersections involving the Product line, Product type, Year or Quarter items.

Because the detail filter is applied to the lowest level item intersections, the intermediate result set must be created in order to calculate those intersection values.

Or out another way, detail filters are applied *after the intermediate result set has been created*.

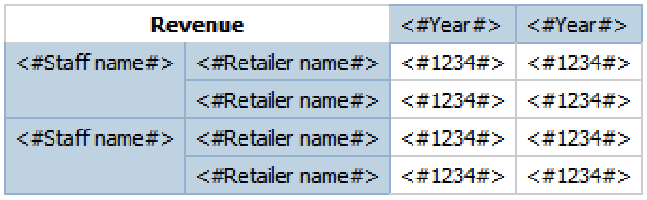
Let's consider a more complex example with nested items on the rows, Staff name and Retailer name.

In the business, Staff name represents a sales representative who sells to retailers (who then in turn sell to their customers). Each sales representative (Staff name) is responsible for a small number of retailers (Retailer name). Therefore, there is no data (for example, no Revenue) for most combinations of Staff name and Retailer name.

Let's see this in Report Studio.

Open Report Studio with the Great Outdoors Company package.

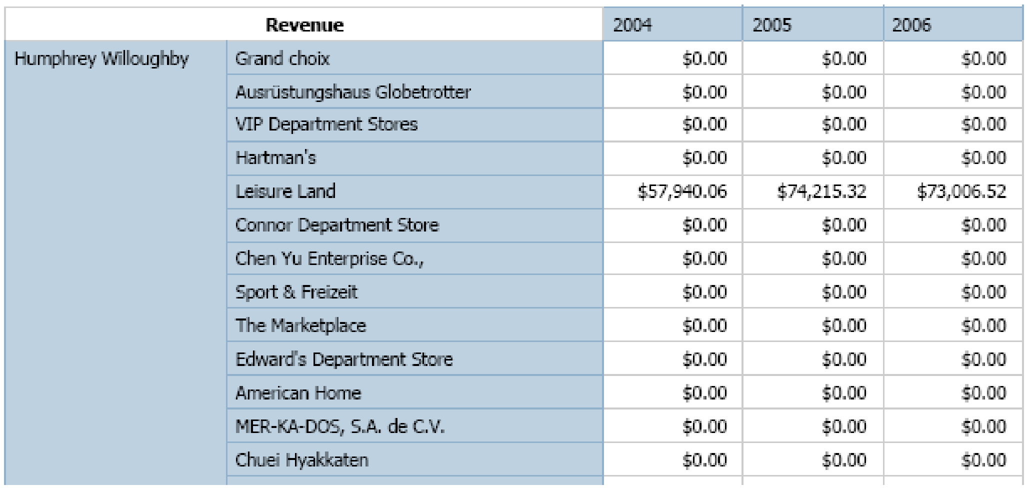
Create a new crosstab report:



Revenue		<#Year#>	<#Year#>
<#Staff name#>	<#Retailer name#>	<#1234#>	<#1234#>
	<#Retailer name#>	<#1234#>	<#1234#>
<#Staff name#>	<#Retailer name#>	<#1234#>	<#1234#>
	<#Retailer name#>	<#1234#>	<#1234#>

Run the report in PDF.

The report is 11,856 rows (258 pages) long. Just a glance at the first page shows all the zero Revenue values:



Revenue		2004	2005	2006
Humphrey Willoughby	Grand choix	$0.00	$0.00	$0.00
	Ausrüstungshaus Globetrotter	$0.00	$0.00	$0.00
	VIP Department Stores	$0.00	$0.00	$0.00
	Hartman's	$0.00	$0.00	$0.00
	Leisure Land	$57,940.06	$74,215.32	$73,006.52
	Connor Department Store	$0.00	$0.00	$0.00
	Chen Yu Enterprise Co.,	$0.00	$0.00	$0.00
	Sport & Freizeit	$0.00	$0.00	$0.00
	The Marketplace	$0.00	$0.00	$0.00
	Edward's Department Store	$0.00	$0.00	$0.00
	American Home	$0.00	$0.00	$0.00
	MER-KA-DOS, S.A. de C.V.	$0.00	$0.00	$0.00
	Chuei Hyakkaten	$0.00	$0.00	$0.00

We want our report to suppress all those zero rows leaving us only 478 rows. There are several ways we can define a filter to do this. We need to ensure we choose the most efficient approach so as to avoid creating the entire 11,856 row intermediate result set only to discard 95% of those rows.

Return to Report Studio.

We want to filter the Retailer names to only show those with Revenue <> 0.

We can take three routes to this:

1. 'Complete' filter( ) expression that specifies the full context of the filter:
   ```
   filter([great_outdoors_company].[Retailer].[Retailer].[Retailer
   name],
   tuple(currentMember([great_outdoors_company].[Retailer].[Retail
   er]), currentMember([great_outdoors_company].[Staff].[Staff]),
   [Revenue]) <> 0)
   ```
   [12]
2. 'Simple' filter( ) expression that takes advantage of the fact that we set the context within the current edge (grand/parent items).
   ```
   Filter([great_outdoors_company].[Retailer].[Retailer].[Retailer
   name], [Revenue] <> 0)
   ```
3. Detail filter –
   ```
   [Revenue] <> 0
   ```

Note, that the detail filter will also affect both the crosstab rows and column items (Retailer name and Year) where the filter( ) expression affects only the row item (Retailer name).

Let's try these filters in Report Studio.

Edit the existing Retailer name expression to match the 'complete' filter in bullet 1 above:

```
filter([great_outdoors_company].[Retailer].[Retailer].[Retailer
name],
tuple(currentMember([great_outdoors_company].[Retailer].[Retail
er]), currentMember([great_outdoors_company].[Staff].[Staff]),
[Revenue]) <> 0)
```

Run the report in PDF noting the execution time and results. On my laptop, this took about 6 seconds.

Return to Report Studio.

[12] This expression says:
Filter the Revenue name level
    filter([great_outdoors_company].[Retailer].[Retailer].[Retailer name],
Where the value of the intersection
    tuple(
Of the current Retailer name in the Retailer hierarchy:
    currentMember([great_outdoors_company].[Retailer].[Retailer]),
and the current Staff name in the Staff hierarchy:
    currentMember([great_outdoors_company].[Staff].[Staff]),
and the Revenue measure
    [great_outdoors_company].[Measures].[Revenue])
Is not equal to zero
    <> 0)

Edit the existing Retailer name expression to match the 'simple filter in bullet 2 above:

```
Filter([great_outdoors_company].[Retailer].[Retailer].[Retailer
name], [Revenue] <> 0)
```

Run the report in PDF noting the execution time and results. On my laptop, this took about 6 seconds.

Return to Report Studio.

Undo to return the Retailer name expression to its original state:

```
[great_outdoors_company].[Retailer].[Retailer].[Retailer name]
```

Create the detail filter in bullet 3 above:

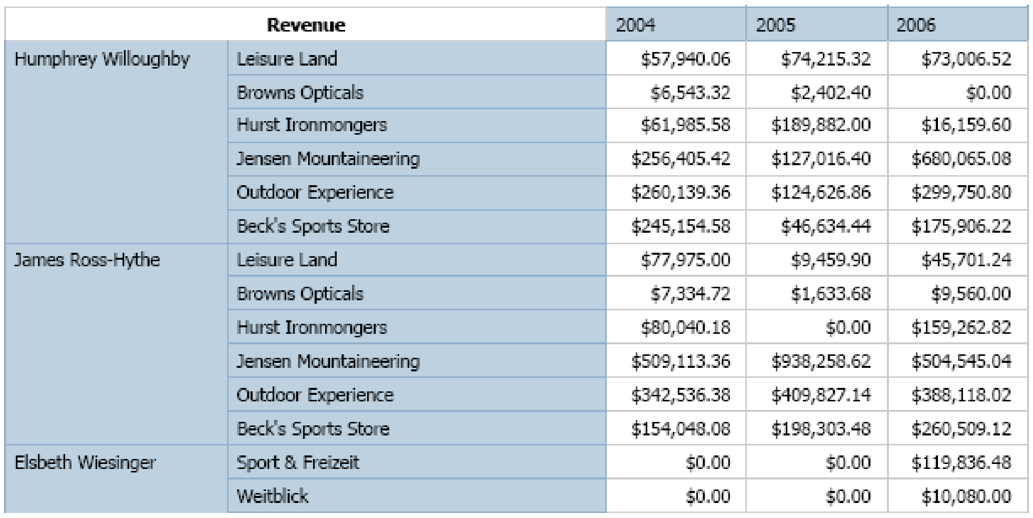
```
[Revenue] <> 0
```

Run the report in PDF noting the execution time and results. On my laptop, this took about 6 seconds.

In other words, in this use case the filters offer equivalent performance.

Of course, were there any Year members with zero values the detail filter would remove them from the report whereas the filter( ) expressions on Retailer name would not. Therefore, these filters are only equivalent for Retailer name.

In all cases, if you run the report in PDF you should see something like this:



Revenue		2004	2005	2006
Humphrey Willoughby	Leisure Land	$57,940.06	$74,215.32	$73,006.52
	Browns Opticals	$6,543.32	$2,402.40	$0.00
	Hurst Ironmongers	$61,985.58	$189,882.00	$16,159.60
	Jensen Mountaineering	$256,405.42	$127,016.40	$680,065.08
	Outdoor Experience	$260,139.36	$124,626.86	$299,750.80
	Beck's Sports Store	$245,154.58	$46,634.44	$175,906.22
James Ross-Hythe	Leisure Land	$77,975.00	$9,459.90	$45,701.24
	Browns Opticals	$7,334.72	$1,633.68	$9,560.00
	Hurst Ironmongers	$80,040.18	$0.00	$159,262.82
	Jensen Mountaineering	$509,113.36	$938,258.62	$504,545.04
	Outdoor Experience	$342,536.38	$409,827.14	$388,118.02
	Beck's Sports Store	$154,048.08	$198,303.48	$260,509.12
Elsbeth Wiesinger	Sport & Freizeit	$0.00	$0.00	$119,836.48
	Weitblick	$0.00	$0.00	$10,080.00

Our report is now only 447 rows/11 pages long, a reduction of 11,409rows/247 pages.

More importantly, our filters are very efficient resulting in excellent performance.

What we have now established is that the detail filter is applied at the cell level i.e. to the leaf level items on each edge. While not as obvious, in the event that a given (grand)parent member(s) – in this case Staff name – has no child members – Retailer names – the (grand)parent member(s) will also be removed from the report.

Thus far, the detail filter and filter(s) expressions offer the same results and performance. We have demonstrated this by filtering Retailer name to suppress zeros.

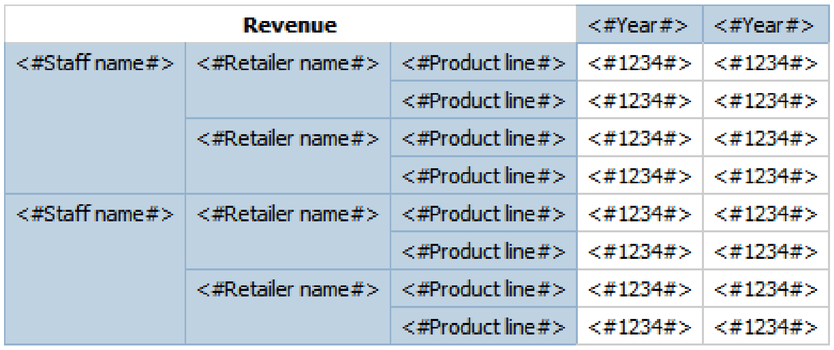
What if we change this report a bit?

For example, you may want to see Revenue by Staff Name by Retailer name *and* Product line.

Return to Report Studio.

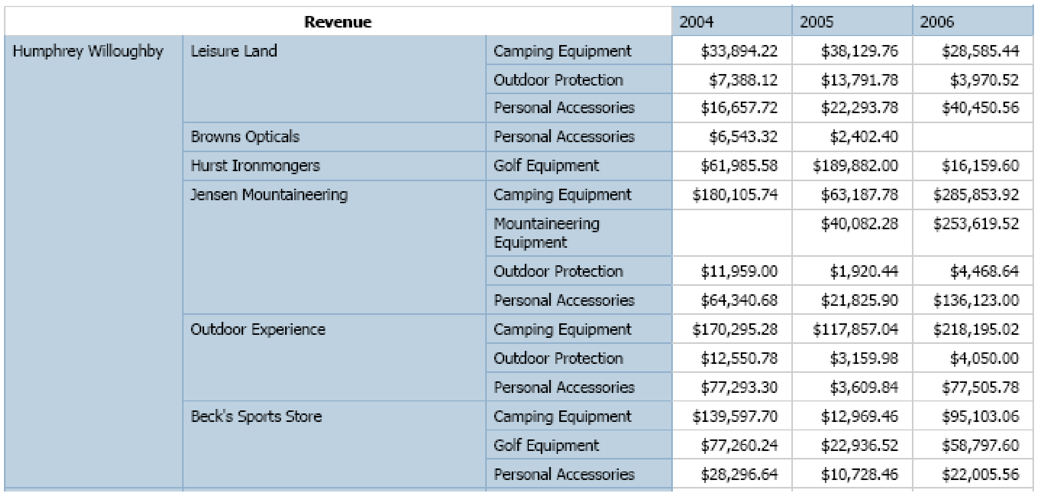
Remember that our current filter is a detail filter [Revenue] <> 0.

Add Product line nested under Retailer name as so:



Revenue			<#Year#>	<#Year#>
<#Staff name#>	<#Retailer name#>	<#Product line#>	<#1234#>	<#1234#>
		<#Product line#>	<#1234#>	<#1234#>
	<#Retailer name#>	<#Product line#>	<#1234#>	<#1234#>
		<#Product line#>	<#1234#>	<#1234#>
<#Staff name#>	<#Retailer name#>	<#Product line#>	<#1234#>	<#1234#>
		<#Product line#>	<#1234#>	<#1234#>
	<#Retailer name#>	<#Product line#>	<#1234#>	<#1234#>
		<#Product line#>	<#1234#>	<#1234#>

Run the report in PDF noting the execution time and results:



Revenue			2004	2005	2006
Humphrey Willoughby	Leisure Land	Camping Equipment	$33,894.22	$38,129.76	$28,585.44
		Outdoor Protection	$7,388.12	$13,791.78	$3,970.52
		Personal Accessories	$16,657.72	$22,293.78	$40,450.56
	Browns Opticals	Personal Accessories	$6,543.32	$2,402.40	
	Hurst Ironmongers	Golf Equipment	$61,985.58	$189,882.00	$16,159.60
	Jensen Mountaineering	Camping Equipment	$180,105.74	$63,187.78	$285,853.92
		Mountaineering Equipment		$40,082.28	$253,619.52
		Outdoor Protection	$11,959.00	$1,920.44	$4,468.64
		Personal Accessories	$64,340.68	$21,825.90	$136,123.00
	Outdoor Experience	Camping Equipment	$170,295.28	$117,857.04	$218,195.02
		Outdoor Protection	$12,550.78	$3,159.98	$4,050.00
		Personal Accessories	$77,293.30	$3,609.84	$77,505.78
	Beck's Sports Store	Camping Equipment	$139,597.70	$12,969.46	$95,103.06
		Golf Equipment	$77,260.24	$22,936.52	$58,797.60
		Personal Accessories	$28,296.64	$10,728.46	$22,005.56

Which resulted in an execution time of 45 seconds or so on my laptop.

Note that Product lines with no non-zero values have been removed from the report. But, the performance was slow because the detail filter is applied only after the intermediate result set with ALL Staff names and ALL Retailer names has been created.

Return to Report Studio.

Disable the detail filter (please do not delete it; you'll need it again later in the workshop).

Edit the existing Retailer name expression to match either the 'complete' filter in bullet 1 above:

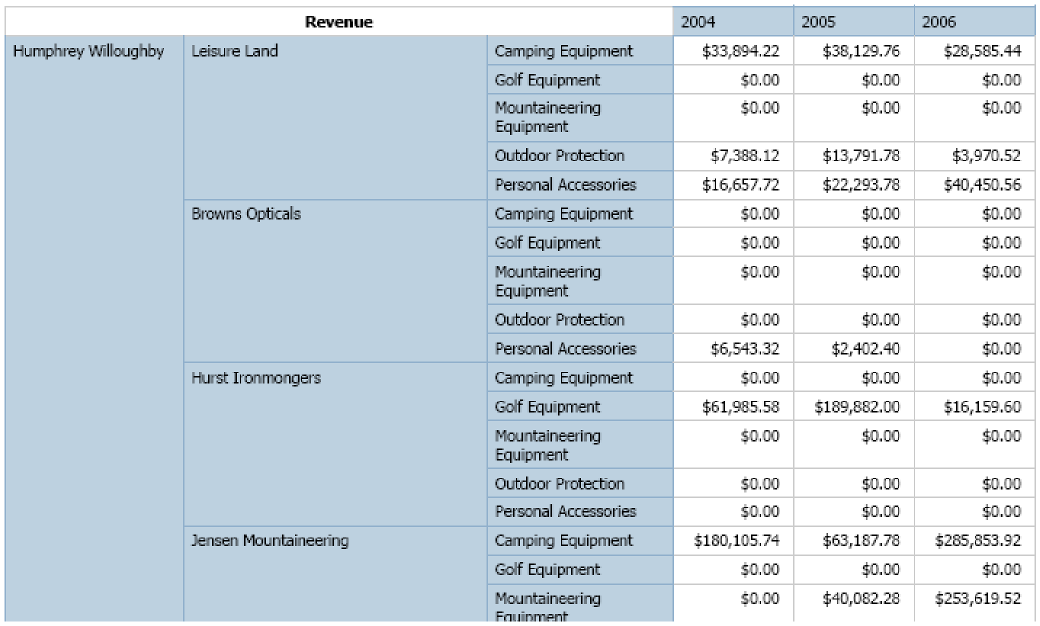
```
filter([great_outdoors_company].[Retailer].[Retailer].[Retailer
name],
tuple(currentMember([great_outdoors_company].[Retailer].[Retail
er]), currentMember([great_outdoors_company].[Staff].[Staff]),
[Revenue]) <> 0)
```

Or the the 'simple filter in bullet 2 above:

```
Filter([great_outdoors_company].[Retailer].[Retailer].[Retailer
name], [Revenue] <> 0)
```

As we automatically apply the (grand)parent content of the report layout, these 2 filters are equivalent.

Run the report in PDF noting the execution time and results:



Revenue			2004	2005	2006
Humphrey Willoughby	Leisure Land	Camping Equipment	$33,894.22	$38,129.76	$28,585.44
		Golf Equipment	$0.00	$0.00	$0.00
		Mountaineering Equipment	$0.00	$0.00	$0.00
		Outdoor Protection	$7,388.12	$13,791.78	$3,970.52
		Personal Accessories	$16,657.72	$22,293.78	$40,450.56
	Browns Opticals	Camping Equipment	$0.00	$0.00	$0.00
		Golf Equipment	$0.00	$0.00	$0.00
		Mountaineering Equipment	$0.00	$0.00	$0.00
		Outdoor Protection	$0.00	$0.00	$0.00
		Personal Accessories	$6,543.32	$2,402.40	$0.00
	Hurst Ironmongers	Camping Equipment	$0.00	$0.00	$0.00
		Golf Equipment	$61,985.58	$189,882.00	$16,159.60
		Mountaineering Equipment	$0.00	$0.00	$0.00
		Outdoor Protection	$0.00	$0.00	$0.00
		Personal Accessories	$0.00	$0.00	$0.00
	Jensen Mountaineering	Camping Equipment	$180,105.74	$63,187.78	$285,853.92
		Golf Equipment	$0.00	$0.00	$0.00
		Mountaineering Equipment	$0.00	$40,082.28	$253,619.52

On my laptop this report completed in approximately 10 seconds,

But, we now have Product lines with zero values in the report. Why? Because we filtered Retailer name not Product line.

So, we are left in a quandary. The detail filter solution gets us the result we want but may result in poor performance because the non-leaf level items are cross joined before the filter is applied. On the other hand, the filter( ) solution gives us the performance but not the desired result.

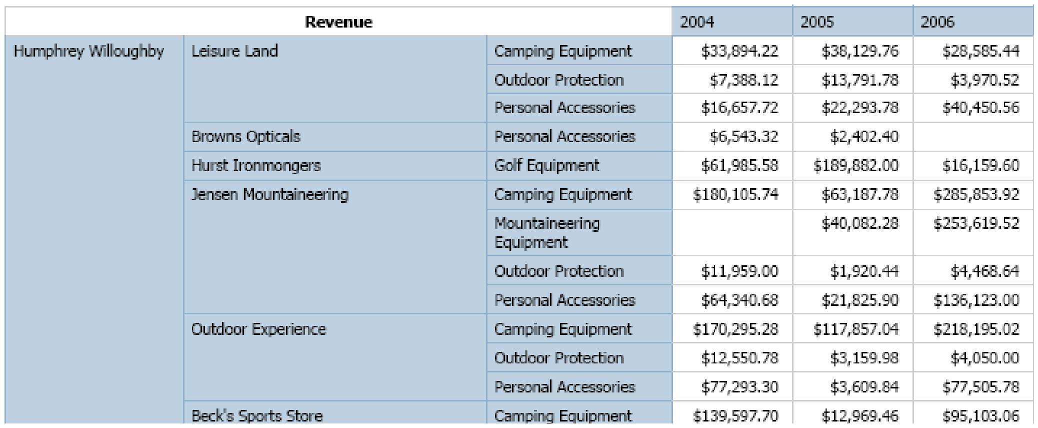
Solution: Use both the detail filter and filter( ) expressions.

Return to Report Studio.

Remember we already have a filter( ) expression on Retailer name.

Enable the detail filter [Revenue] <> 0.

Run the report in PDF noting the execution time and results:



Revenue			2004	2005	2006
Humphrey Willoughby	Leisure Land	Camping Equipment	$33,894.22	$38,129.76	$28,585.44
		Outdoor Protection	$7,388.12	$13,791.78	$3,970.52
		Personal Accessories	$16,657.72	$22,293.78	$40,450.56
	Browns Opticals	Personal Accessories	$6,543.32	$2,402.40	
	Hurst Ironmongers	Golf Equipment	$61,985.58	$189,882.00	$16,159.60
	Jensen Mountaineering	Camping Equipment	$180,105.74	$63,187.78	$285,853.92
		Mountaineering Equipment		$40,082.28	$253,619.52
		Outdoor Protection	$11,959.00	$1,920.44	$4,468.64
		Personal Accessories	$64,340.68	$21,825.90	$136,123.00
	Outdoor Experience	Camping Equipment	$170,295.28	$117,857.04	$218,195.02
		Outdoor Protection	$12,550.78	$3,159.98	$4,050.00
		Personal Accessories	$77,293.30	$3,609.84	$77,505.78
	Beck's Sports Store	Camping Equipment	$139,597.70	$12,969.46	$95,103.06

With an execution time of about 10 seconds on my laptop.

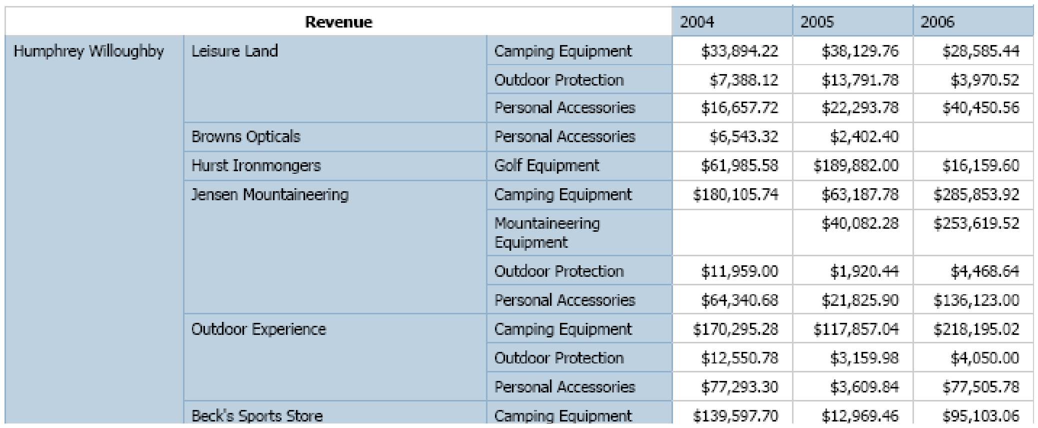
Return to Report Studio.

Disable the detail filter.

Edit the Product line expression as so:

```
filter([great_outdoors_company].[Products].[Products].[Product line],
[Revenue] <> 0)
```

Run the report in PDF noting the execution time and results:



Revenue			2004	2005	2006
Humphrey Willoughby	Leisure Land	Camping Equipment	$33,894.22	$38,129.76	$28,585.44
		Outdoor Protection	$7,388.12	$13,791.78	$3,970.52
		Personal Accessories	$16,657.72	$22,293.78	$40,450.56
	Browns Opticals	Personal Accessories	$6,543.32	$2,402.40	
	Hurst Ironmongers	Golf Equipment	$61,985.58	$189,882.00	$16,159.60
	Jensen Mountaineering	Camping Equipment	$180,105.74	$63,187.78	$285,853.92
		Mountaineering Equipment		$40,082.28	$253,619.52
		Outdoor Protection	$11,959.00	$1,920.44	$4,468.64
		Personal Accessories	$64,340.68	$21,825.90	$136,123.00
	Outdoor Experience	Camping Equipment	$170,295.28	$117,857.04	$218,195.02
		Outdoor Protection	$12,550.78	$3,159.98	$4,050.00
		Personal Accessories	$77,293.30	$3,609.84	$77,505.78
	Beck's Sports Store	Camping Equipment	$139,597.70	$12,969.46	$95,103.06

With an execution time of about 10 seconds on my laptop.

We have not yet filtered Staff name. We need to think about whether we can filter out members in this report or not. Does the Staff name include both sales staff and non sales staff? What does the report consumer expect to see? All staff? All sales staff? All sales staff with non-zero sales?

In this sample data Staff name is only sales people so we might expect the report consumer to want to see all Staff all the time and therefore no filter is required.

2.3 Customer Use Case

Let's look at a real life customer example. This example came to IBM's attention because the report, as initially written, exhibited all of the symptoms discussed earlier in this document, poor performance, massive memory consumption, report server failure, etc.

The details have been changed to remove customer proprietary information but the test case is real.

As the customer PowerCube proprietary customer information, it cannot be shared to allow you to try these examples, nor can we show the report output.

2.3.1 PowerCube Description

The PowerCube has 25 dimensions and approximately 40 thousand members. The dimensions pertinent to this example are:

Dimension	**Levels**	**Leaf Members**
Customers	9	1,500
Products	7	15,000
Item	7	19,000
Supplier	9	1,500
Time	4	2,500
Customer Organization	1	20
Currency	1	200
Event Type	1	5
Event Start Date	1	1,900
Event End Date	1	1,900
Delay Reason	1	10
Reorder Status	1	10
Delay Cause	1	35
Item Deleted Flag	1	2
Etc for 25 dimensions		
Measures	1	Event Count

2.3.2 Report Definition

The report we will examine showed information about failed customer order delivery. As only a tiny fraction of the thousands of customer orders have delivery problems, the report is looking for the few exceptions and a final report of a few hundred rows would be normal. Therefore, it was quite puzzling to the customer to understand why they were encountering difficulties.

The report is a crosstab defined as so:

Crosstab Rows

- Customer Location: The Customer Location - Long Name attribute
- Event Type: The Event Type - Long Name attribute
- Product: The Product - Long Name attribute
- Item: The Item - Long Name attribute
- Supplier Site: The Supplier Site - Long Name attribute
- Delay Reason: The Delay Reason - Long Name attribute
- Delay Cause: The Delay Root Cause - Long Name attribute

Crosstab Columns

- Last 7 Days member

Crosstab Cells

- Event Count measure

Detail Filters

- Customer Location in ('Central Europe', 'Northern Europe', 'Southern Europe ')
- Event Type in ('Forecast Out of Stock', 'Out of Stock')
- [Item Deleted Flag] = 'No'
- [Event End Date] = 'Open'
- [Event Count] > 0
    - This acts as a zero suppression filter

2.3.3 Diagnosis

Let’s examine the report in detail and see if we can determine why this report was problematic as written and then prescribe and test some solutions.

<u>Crosstab Rows</u>

The row edges items all refer to the long name attributes. It is not known why the long name attribute used especially when the long name and member caption contained the same information. This complication does not affect the report (yet) but could cause confusion to subsequent authors trying to determine why the long name attribute is used when the level reference would suffice. It’s odd for no good reason.

<u>Crosstab Columns</u>

A single member poses no problem.

<u>Crosstab Cells</u>

A single measure poses no problem.

<u>Detail Filter</u>

The detail filters are problematic because:

1. The member filters are all on the long name attributes resulting in very slow filtering operations
2. The Event Count filter is applied only after the intermediate row set has been created. As most rows will have zero data, this means we create an enormous intermediate row set and then filter out virtually all of the rows it rather than filtering the member sets prior to cross joining.

How big will the intermediate result set be? Remember that OLAP queries will fetch all members in each nested level. The Customers level has 1,500 members in it as so:

{Wael Abdelazim,
Vincent Adabe,
…
George Zywotco,
Al Zyanski}

The Event Type level has 5 members in it as so:

{Out of Stock,
Forecast Out of Stock,
Re-Stock,
Forecast Re-Stock,
Cancelled}

When we nest these items on the crosstab edge, we create a query which displays all 5 Event Types for every customer. In other words, we create a cross join of Customers and Event Types. That's 1500 customer members cross joined to 5 members or 1500 * 5 = 7,500 rows.

This is already a large crosstab. What happens when we factor in the other data items? Nesting Product under Event Type gives a cross join of 7,500 rows * 15,000 Products = 112,500,000 rows.

I'd argue that a report with 112 million rows or about ~2.5 million pages is not what the author intended. And, there are still several more items to account for. If we expand the cross joins for all of the items on the crosstab rows we get this:

> Customers * Event Types * Products * Reason * Item * Supplier * Root Cause
>
> = 1,500 * 5 * 15,000 * 10 * 19,000 * 1,500 * 35
>
> = 1,122,187,500,000,000,000 rows
>
> = 1.1 Quintillion rows

2.3.4 Solution

We will focus on filtering the member sets prior to the cross joins to reduce the size of the intermediate result set.

Crosstab Rows

Replace long name attribute references with level references to improve report maintainability.

- Customer Location: The Customer Location level reference
- Event Type: The Event Type level reference
- Product: The Product level reference
- Item: The lowest level Item reference
- Supplier Site: The Supplier Site level reference
- Delay Reason: The Delay Reason level reference
- Delay Cause: The Delay Root Cause level reference

Crosstab Columns

No change

Crosstab Cells

No change

Detail filters

Customer Location

Our first detail filter tests the long name attribute (the Customer Location data item was originally the long name attribute) using string comparisons rather then member references resulting in poor performance:

```
Customer Location in ('Central Europe', 'Northern Europe',
'Southern Europe ')
```

The locations are members in this level so we'll delete the existing detail filter and replace the Customer Location level reference with a set( ) expression referring to just the required locations:

```
set([Central Europe], [Northern Europe], [Southern Europe])
```

Or a member based detail filter

```
[Customer Location] in ([Central Europe], [Northern Europe],
[Southern Europe])
```

Event Type

Out second filter detail filter also tests the long name attribute using string comparisons rather then member references resulting in poor performance:

```
Event Type in (‘Forecast Out of Stock', 'Out of Stock')
```

We will delete the existing detail filter move this to the row edge replacing the Event Type level reference with a set( ) expression:

```
set([Forecast Out of Stock], [Out of Stock])
```

Or a detail filter:

```
[Event Type] in ([Forecast Out of Stock], [Out of Stock])
```

Item Deleted Flag and Event End Date

The last two filters are very interesting.

- `[Item Deleted Flag] = ‘No’`
- `[Event End Date] = ‘Open’`

They filter members which do not appear and are therefore equivalent to slicers. We have three ways to treat these filters:

1. Convert them to member based detail filters.
   - `[Item Deleted Flag] = [No]`
   - `[Event End Date] = [Open]`
2. Convert them to slicers.
3. Move them into the row edge filter expressions something like:
   ```
   filter([Customer Locations], tuple(currentMember([Customers]),
   [N]) <> 0)
   ```
   where [N] is a member in the Item Deleted Flag level.

Event Count

The original filter was:

```
[Event Count] > 0
```

This filter acts as a zero suppression filter and is the key filter in determining which members are required on this report. We know that as written the filter is applied to the lowest level edge items only and is therefore applied after the cross joins. We want filter the edge items prior to the cross joins. We will do this using filter( ) expression on each row edge item as required. Let’s examine each row edge item in turn and discuss whether this filter needs to be applied.

Customer Locations

There are only 4 Customer Locations included in the report. This does not seem to be the highest priority.

Event Type

There are only 2 Event Types included in the report. This does not seem to be the highest priority.

Product

There are over 15,000 products. Removing as many as possible early in the processing would be very beneficial. We will change the expression to:

```
filter([Product], [Event Count]) <> 0)
```

Item

There are over 19,000 items. Removing as many as possible early in the processing would be very beneficial. We will change the expression to:

```
filter([Item], [Event Count]) <> 0)
```

Supplier Site

There are over 1,500 sites. Removing as many as possible early in the processing would be very beneficial.

```
filter([Supplier Site], [Event Count]) <> 0)
```

Delay Reason

While there are only 12 reasons, we will err on the side of caution.

```
filter([Delay Reason], [Event Count]) <> 0)
```

Delay Cause

As this is the lowest level edge item, we must filter on it to provide the same functionality as the original detail filter:

```
filter([Delay Cause], [Event Count]) <> 0)
```

These changes resulted in this report executing in ~2 minutes 10 seconds and returning ~145 rows of data. This is a significant improvement over the hours required and failures when the query engine was forced to process the 1.1 Quintillion rows of the intermediate result set of the original report.

This serves as a good test of our theories and was in fact the test case used in the development of this document.

3 Step-By-Step Query Creation

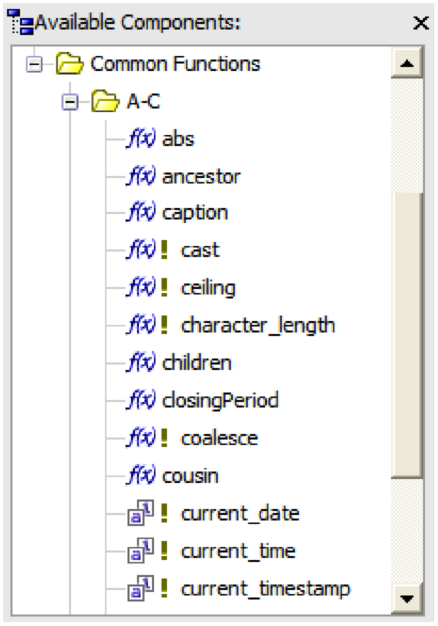
Now that we have some ideas on how to create efficient queries, we need to consider the authoring process. As we have seen, putting all the data items onto the report without filtering and then trying to apply filters after the fact is not a good approach as it results in intermediate data sets which either consume excessive resources or fail.

A better approach is to create the report step-by-step. That is, adding one data item at a time and filtering that item down to the smallest number of members and testing before proceeding to the next data item. By using this building block approach, you ensure that at no stage is your report unnecessarily large thereby reducing intermediate testing times, resource consumption and the possibility of failures. Your motto should be "Filter Early, Filter Often".

Also, develop your reports on data sources with realistic data volumes and data distribution. More than one deployment has been derailed because reports which ran quickly in the development environment exhibit poor performance when run against production data volumes. Often a snapshot of production data (or sterilized data) can be used to ensure an equitable development environment. If it is not possible or practical to make production data volumes available in the development environment, then you should perform performance and final tests in the production environment prior to declaring the report ready for production.

4 Scalar Functions in OLAP Expressions

Report Studio uses Quality of Service indicators on functions to indicate if a given function is considered an expensive function to use. For example, in this list of functions we can see that several functions are marked with an exclamation mark, !, to indicate the relative cost or quality of service:



But, what does this mean in practical terms?

The ! indicator means that the function is not supported by the underlying data source and will be executed within IBM Cognos software after the data has been retrieved.

In the case of OLAP data, this means:

- Data will be fetched from the underlying OLAP data provider
- The data will be streamed through the data engine where the scalar functions will be applied.
- The data will be transformed into an in-memory OLAP cube
- The in-memory cube will be queried to create the final report

Clearly this adds considerable overhead and incurs a performance penalty. Therefore, the use of such functions is recommended only with the utmost care and a clear understanding of the impact on performance and resource consumption.

5 Explicit Aggregate Usage

The underlying PowerCube data provider does not provide support for non-default aggregation. This means that if a measure in the cube has a default aggregation of total specified and your report uses a different aggregation such as minimum( ), the PowerCube data provider is unable to process the query.

Therefore, the IBM Cognos 8 query engine will process the aggregate within the IBM Cognos software using processing steps similar to those for scalar functions. When the in-memory PowerCube is created, the measure will be re-aggregated using the required aggregate function.

Clearly this adds considerable overhead and performance penalty. Therefore, the use of non-default aggregate functions is recommended only with the utmost care and a clear understanding of the impact on performance and resource consumption.

To be sure, always use the aggregate( ) function rather than explicit total( ), minimum( ) or other functions. The aggregate( ) function will leverage the default aggregate rules defined within the OLAP data source for each measure.

6 Conclusions

If you bear in mind the rules in this document you will be well on your way to creating efficient OLAP queries. This document will remain a work in progress and will be updated as the product enhancements are made.

Of special importance is section 3 Step-by-Step Query Creation. Applying both the technical and authoring process lessons can result in a much better development experience.