

MooD 15

Aggregation Performance

This document covers aspects of aggregation performance in Solutions. This includes an overview of the features MooD has for creating, implementing and testing aggregations and some best practice guidelines. It includes reference material on execution methods and caching. **Implementing caching wherever possible is the best thing you can do to improve the performance of Aggregation matrices.**

Document Number: MooD15PBP62

© 2013 MooD Enterprises Ltd, all rights reserved**.**

#### Notice of Copyright and Trademarks

MooD 15 Aggregation Performance

® MooD, MooD Smarter Decisions, Performance Activation, Synchronization Activation Technology and Knowledge Map are registered trademarks of MooD Enterprises Ltd. in the United Kingdom and / or other countries.

Microsoft and Windows are trademarks of Microsoft Corporation in the USA and other countries.

Rights to all other referred trademarks or registered trademarks reside with their respective owners.

Aspects of the Enterprise Business Model, Model-Driven Data Aggregation and Business Solutions to Support Smarter Decisions are protected by International Patent and Patent Pending. These include the Meta-Architecture Framework, Panels Technologies, Auto- Explorer, Business Orchestration, the Activator mechanism, Process Driven System, Performance Activation, Model-Driven Enterprise Management, Dynamic Aggregation, Smart Columns, the Variant Mechanism, and other technologies and mechanisms implemented within MooD Business Architect and MooD Active Enterprise.

© MooD Enterprises Ltd., all rights reserved. No part of this document may be reproduced by any means, or transmitted, or translated into machine language without the written permission of the company.

2

**Contents**

[About this guide 4](#_bookmark0)

[About aggregation performance 5](#_bookmark1)

[Testing aggregations 5](#_bookmark2)

[The Timings command and the Timings Report 6](#_bookmark3)

[The Query Reports 8](#_bookmark4)

[About Builds and upgrading 9](#_bookmark5)

[Best practice 10](#_bookmark6)

[Best practice and workflow overview 10](#_bookmark7)

[Best practice for creating aggregations 11](#_bookmark8)

[Query performance guidelines 11](#_bookmark9)

[Best practice for implementing aggregations 12](#_bookmark10)

[Refactoring existing aggregations 13](#_bookmark11)

[Aggregation matrix caching 14](#_bookmark12)

[What is the aggregation cache? 14](#_bookmark13)

[Benefits and possible pitfalls of aggregation caching 15](#_bookmark14)

[How to decide if caching is suitable for a matrix 16](#_bookmark15)

[How to monitor and maintain caching 16](#_bookmark16)

[Enabling aggregation caching 17](#_bookmark17)

[The **Caching** frequency/lifespan setting 18](#_bookmark18)

[Matrix Cache reports 19](#_bookmark19)

[The Manage Aggregation Cache synchronizer 20](#_bookmark20)

[Execution methods for Aggregation matrices 22](#_bookmark21)

[The Use Fact Values Cache setting 24](#_bookmark22)

[The Performance Report 25](#_bookmark23)

[Best practice for execution methods 26](#_bookmark24)

[Aggregation SQL 27](#_bookmark25)

3

# About this guide

These initial sections will help you understand what you can do to maximise and test aggregation performance.

* [About aggregation performance](#_bookmark1) (page [5](#_bookmark1))

A brief introduction to how you can improve the performance of aggregations in MooD. It also outlines the test facilities available.

* [About Builds and upgrading](#_bookmark5) (page [9](#_bookmark5))

It is important to use the latest possible build, and if you are upgrading repositories, to check whether you need to run an upgrade script.

* [Best practice](#_bookmark6) (page [10](#_bookmark6))

Guidelines and design principles to help you during development and implementation. Divided into:

* + [Best practice and workflow overview](#_bookmark7) (page [10](#_bookmark7))
	+ [Creating aggregations](#_bookmark8) (page [11](#_bookmark8))
	+ [Implementing aggregations](#_bookmark10) (page [12](#_bookmark10))
	+ [Refactoring existing aggregations](#_bookmark11) (page [13](#_bookmark11))

The remaining sections look at very specific aspects of aggregation performance and how to implement them. This gives you the detail for features mentioned in the preceding sections.

* [Aggregation matrix caching](#_bookmark12) (page [14](#_bookmark12))

What Aggregation matrix caching does and how to configure it. This includes a section on the **Manage Aggregation Cache** synchronizer.

* [Execution methods for Aggregation matrices](#_bookmark21) (page [22](#_bookmark21))

Build 59+. By default, MooD automatically chooses what it thinks is the best way to generate and execute the SQL for an Aggregation matrix. However, you can override this and choose specific methods. This section covers this and the **Use Fact Values Cache** setting which is also on the ribbon.

* [Aggregation SQL](#_bookmark25) (page [27](#_bookmark25))

How to see the actual SQL used in an aggregation.

4

# About aggregation performance

Aggregations (matrices and Smart Columns) are used throughout MooD to display calculated results in tables and Info panels, and to drive charts. Their performance has a significant impact on page load times and general operating performance. You can affect performance at two key stages:

* The aggregation design and creation phase.
* The implementation phase.

Maximum benefits can be achieved if you follow the guidance in this document from the outset. However, if you have existing aggregations, you may be able to improve their performance by fine tuning their implementation through features such as execution methods and caching.

Best practice is covered later, but it is useful to understand from the outset what features MooD has to help you maximise performance. The first two are the most important:

* ***Caching*** – result sets can be generated, stored and reused. Although you can set the lifespan of the cache and manage it directly using a synchronizer, caching is only suitable when you know that the cache will remain valid for its lifespan. However, caching is the best way to improve performance and should always be considered.
* ***Execution methods*** – the way MooD creates the code that is used to generate the aggregation. By default, MooD selects what it considers the best method, but you can override this. The Performance Report helps you choose.
* The **Use Fact Values Cache** setting – whether to use a temporary table to store values used within facts before those values are then used to generate the actual aggregations. The Performance Report can help you determine whether this will be beneficial.
* ***Aggregation SQL*** – for advanced use only under guidance from MooD International. This lets you replace the MooD generated SQL with your own SQL. Generally, you will not use this for performance purposes, but it is a possibility.

There are several ways to test aggregations – some specifically for aggregations and some more general. These are outlined in the next section.

All of these features and settings are on the ribbon. Smart Columns have a reduced set, for example, you cannot cache Smart Columns.

## Testing aggregations

MooD includes the following test facilities specifically for aggregations:

* The Matrix Performance Report (the **Performance Report** command on the ribbon). This is probably the most useful report for testing Aggregation matrix performance as it lets you compare execution methods, and find out what Aggregation matrices are not cached. It was introduced in Build 60 and is documented on page [25](#_bookmark23) within the [*Execution methods*](#_bookmark21) section.
* For Aggregation matrices with caching enabled, the Cache Reports (the **Matrix Cache Report** and **Global Cache Report** commands on the ribbon). These are documented on page [19](#_bookmark19) within the [*Aggregation matrix caching*](#_bookmark12) section.

5

These are complemented by these general test features:

* The Timings Report (the **Timings** command on the ribbon) shows you the timings within an aggregation allowing you to identify bottlenecks. See [The Timings command and the Timings Report](#_bookmark3) next for details.
* The Query Reports (generated from the cog icon on the Queries pane) show you general query performance across your repository (aggregations are run as queries and are therefore included). See [*The Query Reports*](#_bookmark4) section (page [8](#_bookmark4)) for details.

Together with the general user experience of performance, these testing features give you ample means of evaluating and understanding any performance issues and areas that should be investigated. In all cases, you should test with representative data and variables.

Across all reports, look for oddities such as stages that take too long in comparison to other stages or aggregations. Also look out for spurious variables as these can be pointers to inefficiencies in your underlying queries.

### The Timings command and the Timings Report

The ribbon for Aggregation matrices and Smart Columns includes a **Timings** command. When this is selected, the load time is added to the **Results** area.

Whilst the load time gives a good general indication of performance, you can generate a report that breaks this timing down further. This gives you a much clearer indication of what is happening within your aggregation and where the inefficiencies might be. For example, you can see precisely how long the aggregation’s source query takes to execute.

To generate this report, click the lower part of the **Timings** command, and then select **Open Timings Report**. This executes the aggregation and displays the report in your browser.

6

Here is a section of one report:

Timings are milliseconds, and you can filter out faster actions.

7

### The Query Reports

The Query reports that you generate from the cog icon on the Queries pane give general information on performance. As aggregations run as queries, these can be useful. They are also good reports for getting an overview of the queries and aggregations being used in a repository.

8

# About Builds and upgrading

Continual performance improvements are being made to MooD 15. Make sure you are using the latest build available to you that is suitable for your Solution. In particular, use Build 60 or later.

As part of this, look in the release notes (or ask MooD International Support) to see if upgrade scripts are required. You may need to run a script against the database (repository) to get existing aggregations to work or perform better in a newer build. On the installation media, the release notes are always located in the **MooD** folder.

At the time of writing, you need to run a script if you are upgrading repositories created in:

* SQL Server Builds 42 to 56
* Oracle Builds 42 to 51

On the installation media, the upgrade scripts are always located in the **MooD\Extras** folder. Their names always refer to the database type and builds affected. Use the database administration software for your SQL Server or Oracle instance to execute the script against each repository affected. If you need assistance with this, contact MooD International Support.

**Note:** If you are upgrading existing aggregations, see the [*Refactoring existing*](#_bookmark11)[*aggregations*](#_bookmark11) section on page [13](#_bookmark11) for some specific advice about possible changes you can make. This is in addition to the general best practice advice regarding design and implementation.

9

# Best practice

The first section below is an overview of best practice and a suggested workflow for the whole process of creating, testing and implementing aggregations. This is followed by sections giving more detailed best practice advice and suggestions covering:

* [Creating aggregations](#_bookmark8) (page [11](#_bookmark8)). Primarily this covers Aggregation matrices, but much of the advice is relevant to Smart Columns.
* [Implementing aggregations](#_bookmark10) (page [12](#_bookmark10)). Once you have aggregations you can take steps to improve their performance by controlling how they are implemented.
* [Refactoring existing aggregations](#_bookmark11) (page [13](#_bookmark11)). If you have some aggregations that have been developed in earlier builds, it might be possible to improve their performance by refactoring certain aspects to make use of subsequently introduced features.

## Best practice and workflow overview

What follows is our basic best practice guidelines and suggested workflow. The remainder of this document expands upon this and some additional features not introduced here. However, if you follow the points below, you’ll be well on your way to developing fast aggregations.

* Make sure you are using the latest build available to you that is suitable for your Solution, and that if you have upgraded, you have run any upgrade script required. See [*About Builds and upgrading*](#_bookmark5) on page [9](#_bookmark5) for details.
* Create your Aggregation matrix or Smart Column and incorporate it into your Solution. In general, the principles of good query design apply such as restricting the scope of the source query as much as possible. See the [*Best practice for creating aggregations*](#_bookmark8)section on page [11](#_bookmark8) for some specific advice.
* Decide whether aggregation caching is appropriate. If caching is appropriate, it is always a good idea to implement it.
* Test its performance. If possible, use representative data and variables.
* If you feel you need to improve performance, generate a Performance Report and see whether it recommends a particular execution method. Note that the recommendation will only have merit if you use representative data and variables. You also need to test execution methods with caching disabled. Once you have found the best execution method, you can enable caching again.

See the [*Best practice for implementing aggregations*](#_bookmark10) section on page [12](#_bookmark10) for more details on caching and execution methods.

In most cases, aggregations will perform well and you will not have to go beyond the decision on whether to use caching.

Finally in this overview, it should be noted that complex aggregations on large data sets will take time and performance assessments can be subjective. However, the guidance in this document should help you achieve the best possible performance across your aggregations.

10

## Best practice for creating aggregations

Key advice during the creation phase is:

* Use the latest MooD build possible.
* Restrict the source query as far as possible (much of the general advice for efficient query development apply to aggregations as they share the same query engine). The standard query performance guidelines are in the next section.
* Try and restrict the number of variables that might be involved. Variables add to the permutations being processed.

o If you are going to implement caching, take note of any initial selections that you use and ensure that you at least cache the results for those variables. This will significantly improve your initial page load times. See the [*Best practice for implementing aggregations*](#_bookmark10) section on page [12](#_bookmark10) for more on this.

* If you are going to have variables that will be set from drop-down controls in Active Enterprise, try and include them in the source query (as opposed to subsequent levels or dimensions).
* Where possible turn off totals in your dimensions and levels.
* Make sure the query is optimized (showing  at each stage).

Once you have your working aggregation, see the [*Best practice for implementing aggregations*](#_bookmark10)section on page [12](#_bookmark10) for details on steps you can take to ensure that it operates as efficiently as possible.

### Query performance guidelines

The following guidance for faster queries is taken from the ***MooD 15 Getting Started Guide***:

* The starting block should involve as few elements as possible. The most efficient starting point is **the parameter**, followed by **elements**, **the results of query**, and then **type**. The **all elements except** and **everything** settings are the least efficient.

o When using **the parameter**, use a **where type** condition to stop users executing queries over more elements than necessary. You should always try and restrict the breadth of a query.

* Conditions and their order can significantly affect performance. If a chain of conditions does not show the optimized icon  at each point, experiment with their order and try to get the optimized parts of the query nearer the top of the chain. You can also attempt to restrict your query to conditions that lend themselves to optimized performance. These include:
* **where** conditions based on name (excluding measure instances), type, parent, children, reference, and the contents of simple type fields.
* Find blocks that find children, parents, descendants, and reference/referenced by.

When ordering conditions, put the following as early as possible in the query:

* **AND** conditions that are most likely to fail.
* **OR** conditions that are most likely to succeed.

11

* If using sub-queries, consider the performance of each, and pay particular attention to condition ordering when sub-queries are **AND**’d or **OR**’d together.

o **And Finally** blocks and the **Take Top *N*** condition can stop sub-queries being optimized.

* If possible, locate activated data constructs at the end of a query. Activated data is held externally and is slower to access. The less you need to access the better. Note that because of the introduction of aggregations in MooD 15, you are unlikely to use field activations.

## Best practice for implementing aggregations

Key advice during the implementation phase is:

* Deploy on the latest MooD build possible.
* Consider whether caching is appropriate. The biggest performance gains come from caching. So, wherever possible, cache.
* Consider implementing a cache refresh regime using the Manage Aggregation Cache synchronizer. For example, if the data that the aggregation uses is uploaded monthly and does not change between uploads, consider deleting and recreating the cache immediately after the upload.
* The Manage Aggregation Cache synchronizer gives you a fine level of control over what variable permutations are included in the cache. If your models include aggregations or charts driven from aggregations that gather variables from the user, ensure that these possibilities are included in the cache. This will significantly improve your initial page load times.
	+ It is very important to make sure that all variables are accounted for and included in any caching regime.

**Note:** To quickly see which Aggregation matrices aren’t cached, generate a Performance Report for all Aggregation matrices and look at its **Time** column. Cached matrices all have **(cached)** in this column.

* Consider whether setting a specific execution method is suitable. To determine this use the **Performance Report** command. However, note that for this to be of any use, you must use representative data and variables.

o You need to test execution methods with caching disabled. If you decide to test execution methods on a live site, make sure you switch caching back on when you are finished.

**Note:** Smart Columns do not use caching or execution methods.

The [*Aggregation matrix caching*](#_bookmark12) and [*Execution methods for Aggregation matrices*](#_bookmark21) sections on pages [14](#_bookmark12) and [22](#_bookmark21) respectively explain how to implement these features.

12

## Refactoring existing aggregations

As well as considering the general best practice given in the preceding sections (such as making sure you have run any upgrade script), consider the following when refactoring existing aggregations:

* Prior to Build 59, in Aggregation matrices, to get grand totals only, you had to add a totalling level and have that as the only visible level. Whilst this still works, in Build 59 onwards if you leave all levels hidden (unpinned), you get the grand totals. We recommend you remove totalling levels in existing Aggregation matrices.
* Prior to Build 59, MooD recommended replacing chains of hidden levels with single queries spanning those levels. In Build 59 onwards we recommend using levels as intended and replacing such queries in upgraded aggregations.
* Also consider whether you have written complex workarounds before these features were introduced:
* Build 58 introduced current date times in facts (as literal values or variables).
* Build 61 introduced the ability to find aggregated pick states in facts by means of a query.
* Build 52 introduced the ability to add levels that don’t exist in the source query, and better handling of nulls.

You are unlikely to be on a build earlier than 52.

13

# Aggregation matrix caching

Aggregation matrix caching improves performance significantly. Use caching whenever possible. To explain caching, this section begins with:

* [What is the aggregation cache?](#_bookmark13)
* [Benefits and possible pitfalls of aggregation caching](#_bookmark14).
* [How to decide if caching is suitable for a matrix.](#_bookmark15)
* [How to monitor and maintain caching](#_bookmark16).

These initial sections give background details and essential guidance to help you determine when caching is suitable for your Solution. The remaining sections cover actually using MooD’s aggregation caching features.

* [Enabling aggregation caching](#_bookmark17) (page [17](#_bookmark17)).
* [The **Caching** frequency/lifespan setting](#_bookmark18) (page [18](#_bookmark18)).
* [Matrix Cache reports](#_bookmark19) (including how to use the information they contain) (page [19](#_bookmark19)).
* [The **Manage Aggregation Cache** synchronizer](#_bookmark20) (page [20](#_bookmark20)).

## What is the aggregation cache?

The aggregation cache stores the calculated results of specified Aggregation matrices. Values already calculated can be retrieved from the cache quickly without the need to recalculate each time the matrix is accessed (hit). Because of this, aggregation caching improves performance.

Cached results have a lifespan. This ranges from hourly to indefinitely. You need to determine which lifespan is best for each matrix. Base this decision on the likelihood of the results changing, and the possibility and danger of using outdated and hence incorrect values retrieved from the cache.

Once a set of cached results has expired, the next time the matrix is hit, its result set is recalculated and recached, thereby repeating the caching cycle.

If caching is enabled, values are added to the cache as they are calculated for the first time. For example, if the matrix includes variables, some variable permutations will be calculated and added to the cache before others. It depends on use. However, there is a Manage Aggregation Cache synchronizer that you can use to populate the cache.

Some matrices will not be suitable for caching. Of those that are, some will be suitable for indefinite caching, and some will require regular refreshes to counter the possibility of inaccuracy. MooD supports all of this, but you have to decide what is best for each matrix.

Key points:

* MooD maintains a single aggregation cache. This cache stores individual result sets for cached Aggregation matrices.
* Caching is set within each Aggregation matrix. There is a setting called **Caching** on the ribbon (**Home** tab, **Caching** group). This setting does two things: it enables caching, and it sets the lifespan of the cached result set and hence the potential frequency of recaching.

o By default, aggregation caching is disabled for each Aggregation matrix.

14

* Once an Aggregation matrix allows caching, its results are cached the next time the matrix is hit by a model or included in a Manage Aggregation Cache synchronizer that creates the cache for that matrix.
* Executing the matrix within its Designer tab does not populate the cache.
* Results for variables permutations within aggregations are cached as those permutations are hit. As the cache won’t include everything immediately, the first user to request an uncached variable permutation will take the performance hit as those results are calculated and cached. You can mitigate this using the Manage Aggregation Cache synchronizer.
* Cached result sets are valid until they expire, or until the matrix is next saved in Business Architect.
* Cached results expire when their lifespan is exceeded or they are actively deleted by a Manage Aggregation Cache synchronizer.
* Matrices are recached when they are next hit and those results are then used until they too expire. Again, you can force a recache using the Manage Aggregation Cache synchronizer.
* You can generate individual and global aggregation cache reports.
* As mentioned, there is a Manage Aggregation Cache synchronizer. This can do four things:
* Populate the cache for a matrix. You can do this for all variable permutations or for selected variables (such as the initial page load ones). This allows you to bypass the default caching mechanism. For example, for a matrix whose cache expires at midnight, you could use this synchronizer to schedule a recache at 1am, rather than wait for the first true hit of the day which would be slower due to the necessity to recache. Equally importantly, you can populate the cache with all variable permutations at once, rather than piecemeal as those permutations are requested (the ability to select the variables cached gives you a fine level of control over this).
* Delete cached values for specific matrices.
* Delete cached values not used within a timeframe, for example, the last month.
* Delete all cached values in the repository. This is Build 60+.
* Smart Columns cannot be cached. Hence the **Caching** setting is disabled for Smart Columns.

### Benefits and possible pitfalls of aggregation caching

Significantly improved performance is the obvious benefit of caching.

The pitfall is potentially incorrect results. The aggregation matrix cache will not recognise changes that render its cached values inaccurate. You, as the Solution developer, must know and judge when it is appropriate to use caching.

15

### How to decide if caching is suitable for a matrix

To decide if caching is suitable we suggest:

* If the source values in the aggregation constantly change, you can rule out caching immediately.
* Make a judgement on the suitable lifespan for the cache. Base this on the frequency with which the source data used by the aggregation will change. The **Caching** setting sets the lifespan.
* Decide if your Solution will benefit from a refresh regime using the Manage Aggregation Cache synchronizer. For example, to schedule a cache refresh immediately after a regular data upload.

### How to monitor and maintain caching

To monitor and maintain caching once enabled:

* Use the Matrix Cache reports to see how the cache is being used.
* Occasionally generate a Performance Report for all Aggregation matrices. The report’s **Time** column tells you which matrices are cached and which aren’t. Check that some have not had caching inadvertently disabled (for example, to test execution methods), or whether uncached aggregations can be cached.
* Speak to users regarding their confidence in the validity of the results they are seeing and the speeds they are experiencing. As part of this, make Solution administrators aware of what is being cached and what isn’t. This will help them identify problems and caching is best managed collaboratively. Watch out for:
* Changes to data update frequencies. For example, if data starts being uploaded fortnightly instead of monthly.
* New variables. For example, if users can select one of three areas and a fourth is subsequently added, make sure that any Manage Aggregation Cache synchronizer is updated to include that additional variable permutation.
* New synchronizers or working practices that change values that you thought were static.

You may have to:

* Change the lifespan of the cache.
* Create a Manage Aggregation Cache synchronizer regime to update the cache at specific intervals, or alter an existing regime.
* Stop caching.

16

## Enabling aggregation caching

By default, Aggregation matrices are not cached. Caching is enabled per matrix, and you set the lifespan of each set of cached results.

**Task 1** To enable caching for an Aggregation matrix:

1. Open the Aggregation matrix in Aggregation Designer.
2. On the ribbon, on the **Home** tab, in the **Caching** group, click **Caching** and select a frequency/lifespan from the drop-down list.

The cached results have this lifespan and are discarded at the selected frequency. For example, **Hourly** means that the cached results will be used until the current hour is up, after which they will be discarded and a new set of cached results generated when the matrix is next executed on a model or by the Manage Aggregation Cache synchronizer. The exception to this is **Infinite**. This means that after results have been cached for the first time, those results are used indefinitely and not updated unless specifically included in a scheduled Manage Aggregation Cache synchronizer. See [*The Caching*](#_bookmark18)[*frequency/lifespan setting*](#_bookmark18) on page [18](#_bookmark18) for details.

Provided an Aggregation matrix has caching enabled, results are cached when the matrix is first ***hit***. Hits occur when an Aggregation matrix is:

* Executed on a model. This includes opening a model that includes the matrix.
* Included in a Manage Aggregation Cache synchronizer.

The cached results will then be used for subsequent hits during the cache’s lifespan. Once the cached results have expired, the cache is recreated on the next hit and the cycle resumes.

If the Aggregation matrix uses variables, the cache gradually builds as variable permutations are hit during the lifespan i.e. in normal use such a cache isn’t created all at once. You can however create a Manage Aggregation Cache synchronizer to populate the cache for all variables at once.

When you directly open an Aggregation matrix in Business Architect and execute it (using the **Execute** command on the ribbon), this **does not** create or refresh the cache. Also note that when you save an Aggregation matrix, any cached results held for it are removed. This includes doing a save when no changes have been, or a minor change such as the label used for a fact.

17

The **Caching** frequency/lifespan setting

The **Caching** setting on the ribbon for an Aggregation matrix can be set to several lifespans/frequencies. Setting one of these enables caching. The following table details the individual settings.

#### Table 1. Aggregation Matrix Caching Settings

|  |  |
| --- | --- |
| **Off** | Caching is not enabled for this Aggregation matrix. This is the default setting. Including a matrix set to **Off** in a Manage Aggregation Cache synchronizer will not override this. The synchronizer will fail for that matrix and report that the matrix is not cached. |
| **Infinite** | Cache and use the same set of results indefinitely. Indefinitely actually means until the matrix is changed (saving a matrix deletes its cached results) or until the cache is refreshed by a Manage Aggregation Cache synchronizer. |
| **Hourly** | Cache and use the results until the start of the next hour. For example, if the results are cached at 12:37pm, they will expire at 12:59:59. The key point is that the cache is not used for 60 minutes. It is only used for the remaining part of the hour within which it is created. Once the cache has expired, the next hit recreates it.The remaining settings work in the same way. |
| **Daily** | Cache the results until 23:59:59 that day. |
| **Weekly** | Cache the results until 23:59:59 on the Sunday of that week. |
| **Monthly** | Cache the results until 23:59:59 on the last day of the current month. |
| **Yearly** | Cache the results until 23:59:59 on the last day of the current year. |

18

## Matrix Cache reports

When an Aggregation matrix is open in Business Architect, you can generate two caching reports from the ribbon:

* + A **Matrix Cache Report** for the matrix you are looking at.
	+ A **Global Cache Report** for all cached Aggregation matrices.

These reports are displayed in your browser and include the details outlined in the following table.

#### Table 2. Aggregation Cache Reports - Content

|  |  |
| --- | --- |
| **Matrix** | The name of the Aggregation matrix. |
| **Policy** | The lifespan/frequency setting. |
| **Count** | The number of result permutations that have been created for this matrix. |
| **Variables** | Any variables and their settings used when the cache was generated. |
| **Hit Count** | The number of times the matrix cache has been used (hit) since the cache was generated. |
| **Last Hit** | The last time the cache was accessed. |
| **Generation Time** | When the cache was generated. |
| **Expiry Time** | When the cache will be discarded. **Infinite** caches will show **n/a**. |
| **Record Count** | The number of records accessed when the cache was generated. This gives you an idea of the size of the cache. The size of the cache itself should never be an issue. |
| **Total** row | Some statistics showing the performance of the overall cache. This is more relevant to the Global Cache Report. |

How to use this information:

* + Use the Global Cache Report to quickly find out what matrices are being cached.

o To find out what matrices are not being cached, generate a Performance Report for all Aggregation matrices. This report’s **Time** column will tell you what matrices aren’t being cached. See [*The Performance Report*](#_bookmark23) on page [25](#_bookmark23) for details.

19

## The Manage Aggregation Cache synchronizer

Use this synchronizer to manage the cache of Aggregation matrix results.

The **Manage Aggregation Cache** synchronizer in the **MooD Repository** group of synchronizers gives you this dialog box:



You can create a Manage Aggregation Cache synchronizer to do any of the following cache management tasks:

* Delete the cached results that are held for an Aggregation matrix. Do this when you know that the cached values are incorrect or no longer required. For example, if you have run an Excel Import synchronizer that affects the values used in a cached Aggregation matrix, you might want to delete the cache and then repopulate it.
* Delete values in the cache that have not been used within a specified period.

20

* Delete all cached values.
* Create cached values for an Aggregation matrix and add them to the cache. You select the Aggregation matrix and set any variables it requires. This can include permutations of variables. When you execute the synchronizer, the result sets are generated for the different permutations and added to the cache.

21

# Execution methods for Aggregation matrices

#### Build 59+ (execution methods)

#### Build 60+ (Performance Report and Use Fact Values Cache setting)

Aggregation matrices can be executed in a number of ways called ***execution methods***. Each method uses different SQL constructs and each is suited to specific matrix design characteristics. Execution methods apply to SQL Server only.

For each Aggregation matrix, the (execution) **Method** setting is on the ribbon (**Home** tab,

**Execution** group).

The default **Auto** setting lets MooD examine your matrix and choose what it thinks will give best performance. However, **Auto** will not always select the best method. Hence, you can manually select a specific method based on the characteristics of your matrix and the results of your own testing and the Performance Report.

Click in the **Execution** box to display the **Execution Optimization** dialog box shown in the next image. This lets you see the different execution methods and choose the one to use.

22



The following table gives guidance on when to use each setting.

#### Table 3. Aggregation matrix Execution Methods

|  |
| --- |
| **Auto** |
| MooD decides and chooses the best method. This is the default and will generally be accurate. However, you can manually select another method if testing shows that another is better.You cannot see what method **Auto** chooses. |
| **Temporary workings table** |
| Usually provides the best performance.Try this when you have large numbers of source records ***and*** a small number of levels. |
| **Cache source values** |
| Try this when you have one of the following:* A complex source query that returns a small number of elements. If the source
 |

23

|  |
| --- |
| query is the slow part of your aggregation, this method is likely to be your best choice.* Many levels – particularly visible (pinned) levels.
* Many facts.
 |
| **Temporary workings table (indexed)** |
| Try this when you have several levels visible. |
| **Temporary workings table (heavily indexed)** |
| Try this when you have several complex levels visible. |
| **Permanent workings table (for compatibility only)** |
| Use this when advised to by MooD International Support or if the other methods prove ineffectual. It is unlikely to be any better. |
| **Single statement (caution)** |
| You are highly unlikely to need this method. If your source query is the only level and doesn’t show totals, this can be fast. However, such aggregations are rare and using this method with typical aggregations could make your machine unresponsive (it timesout after approximately 5 minutes). |

The Performance Report helps you select an execution method. You must test execution methods using data and variables that are truly representative of the deployed Solution. Testing with small data sets will not give you credible results on which to base a decision. You could manually select a method that in practice performs worse than **Auto**. See [The Performance](#_bookmark23) [Report](#_bookmark23) section on page [25](#_bookmark23) for details on generating and using this report.

The **Use Fact Values Cache** setting

#### UseFactValuesCache.pngBuild 60+

By default, **Use Fact Values Cache** is enabled for all Aggregation matrices as it will generally be beneficial to temporarily cache the values used within an aggregation’s facts. However, you can use the Performance Report to test this assumption. See the next section for details.

We cannot give firm guidance as to when deselecting this check box will be beneficial.

24

## The Performance Report

#### Build 60+

The **Performance Report** command on the ribbon works for all matrix types, but its primary purpose is to generate a report showing the performance of Aggregation matrices and comparing execution methods. Provided you test with representative data and variables, this can help you select the best performing execution method for each Aggregation matrix. It also shows you the effect of the **Use Fact Values Cache** setting.

When you click **Performance Report**, you get this dialog box:

Using this you can generate a Performance report for:

* The current matrix only. This executes the matrix ten times and shows you those ten results and their average.
* All matrices in the repository. Each matrix is listed once.
* All Aggregation matrices. Each matrix is listed once.

Each report lists the variables used. By default, the report uses the variables stored in each matrix. However, you can specifically set the values used. Note that if you generate a report for multiple matrices that include duplicate variable names, you will get one instance (and hence one setting) of each duplicate variable. Similarly, you will have one current epoch setting that will be used across all applicable matrices.

The **Compare various execution methods** check box is for Aggregation matrices (it is ignored for other matrix types). It tests the different execution methods and gives you a report like this:

25



The figures in bold (**Auto** in the example) indicate the currently selected execution method. Cells in green or yellow indicate a better execution method (green is best). Provided you are testing with representative data and variables, you should change the execution method to the suggested method.

The permutations of different execution methods include some showing the effect of clearing the **Use Fact Values Cache** check box.

If the **Compare various execution methods** check box is not selected, the **Timings** column shows the performance of the currently selected execution method.

**Note:** Test execution methods with matrix caching disabled. If the matrix is cached, your report will include **(cached)** in the **Time** column and the different execution method columns will be empty. Once you have selected the best execution method, you can then enable caching if required.

Looking at the **Time** column is a good way to see which Aggregation matrices are not cached. The Global Cache Report will only show you which matrices are cached.

The general advice for Aggregation matrices is to generate a report across all your matrices and look for yellow or green cells. These indicate potential gains through changing the execution method for that matrix. Generate an individual report for the matrix and check the findings. If the individual report confirms the potential benefits, change the execution method accordingly. **We reiterate the importance of testing against representative data and variables.**

## Best practice for execution methods

The current best advice is:

* Make sure you have representative data/variables to test against.
* See how **Auto** performs first.
* Generate a Performance Report and compare **Auto**, **Temporary workings table** and

**Cache source values**. One of these three is likely to be the best method.

If you still don’t get acceptable performance, you should revisit the original design of your aggregation matrix and consider whether caching is suitable (if not already enabled).

26

# Aggregation SQL

To see the SQL MooD generates for an aggregation, on the ribbon, click **Show SQL**.



MooD International developers will be able to interpret this SQL and its role in the aggregation’s performance.

It is also possible to customise the SQL used in the aggregation. If you click **Custom SQL**, you can edit the existing SQL and use it instead. Ordinarily this is done by MooD International developers and should only be attempted under their guidance.

When Custom SQL is in use, the Aggregation Designer confirms this as shown here:

27