Introduction to Mondrian Performance Tuning
## Contents

Overview .............................................................................................................................................................. 1

Introduction to Mondrian Performance Tuning ............................................................................................ 2
  Maximizing Performance ............................................................................................................................... 2

Strategies for Increasing Mondrian Performance ......................................................................................... 4
  Performance Improvement Process ........................................................................................................... 4
  Monitoring Performance ............................................................................................................................ 4
  Optimizing the Database ............................................................................................................................. 4
    Aggregate Tables .................................................................................................................................... 5
  Tuning the Mondrian Schema ...................................................................................................................... 7
    Approximate Cardinality ......................................................................................................................... 7
  Optimizing the Mondrian Configuration .................................................................................................... 8
    Memory ................................................................................................................................................. 8
    Caching ............................................................................................................................................... 8
  Mondrian Configuration Properties ......................................................................................................... 9
  Native Evaluation Properties .................................................................................................................. 10

Related Information ......................................................................................................................................... 11
This page intentionally left blank.
Overview

This document covers some best practices on the Business Analytics (BA) tool called Mondrian.

This document is not intended to dictate what the best options are, but rather to present some best practices for customers working with analysis of large amounts of data. Some of the topics covered here include optimization of resources and databases.

The intention of this document is to speak about topics generally; however, these are the specific versions covered here:

<table>
<thead>
<tr>
<th>Software</th>
<th>Version(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pentaho</td>
<td>5.4, 6.x, 7.x</td>
</tr>
<tr>
<td>Mondrian</td>
<td>3.x</td>
</tr>
</tbody>
</table>

The [Components Reference](#) in Pentaho Documentation has a complete list of supported software and hardware.
Introduction to Mondrian Performance Tuning

By providing insight into key performance metrics, BA tools such as Mondrian provide competitive advantage in the analysis of historical trends and data. While traditional BA tools have often been expensive and difficult to maintain and use, Mondrian is a web-based, open source BA tool that enables organizations to quickly and easily analyze critical business data.

Once Mondrian is set up, you can interact securely with large volumes of business data to gain valuable understanding of your company's vast data without waiting for the assistance of IT or database administrators.

Mondrian responds to queries fast enough to allow users to interactively explore business data by drilling and cross-tabulating information.

**Maximizing Performance**

When you make a change to a report such as adding or removing a dimension or measure, you need the report to update within seconds. Mondrian optimizes performance to make sure you get your results quickly, and it can be further customized for even better results.

![Figure 1: Mondrian Function](image)

As shown in Figure 1, an OLAP MDX query is sent by an application such as Pentaho Analyzer. The Mondrian engine receives the query and checks its internal cache for any results already held in memory in the OLAP cube. Data not found in the cube will be retrieved from a star schema within a database using SQL. Data returned from the database star schema is returned to the Mondrian engine and cached internally to fulfill future data requests.

With proper caching and a well-designed star schema, Mondrian performance is good for a wide variety of datasets and queries. By default, Mondrian will perform some caching to improve overall throughput, but enhancing performance further requires some additional configuration.
A Mondrian schema is a map to your data. It contains a logical model and a mapping of this logical model onto a physical model in the database. While the logical model comprises the constructs used to write queries: cubes, dimensions, hierarchies, levels, measures, and members, the physical model is the source of the data that is presented through the logical model. The physical model is represented by a star schema as shown in Figure 2:
Strategies for Increasing Mondrian Performance

Given a solid foundation of hardware resources and system fundamentals such as low latency networks, the following strategies are available to increase Mondrian performance:

- Optimizing the Database
- Tuning the Mondrian Schema
- Optimizing the Mondrian Configuration

Performance Improvement Process

There are five high-level steps for optimizing Mondrian performance:

1. **Set up a test environment and prepare the data.** It is helpful to have a dedicated environment for testing that is separate from your development environment, to ensure the captured metrics measure Mondrian performance and do not include extraneous factors that may also possibly impact performance.

2. **Run queries in your test environment and capture the performance metrics.** It is important to have a record of your baselines so you have real numbers to measure optimizations against.

3. **Tune the database.** If baseline performance is not satisfactory, begin by ensuring indexes are created and your database optimizer is aware of your star schema configuration.

4. **Tune the Mondrian schema.** If tuning the database does not solve the problem, the next step is to speed up Mondrian queries using things like aggregate tables and caching.

5. **Find alternative ways of presenting the information.** If the prior steps do not solve the performance issues, look for alternative ways of presenting your information, like breaking up the data into multiple cubes or doing analysis with preset filters.

Monitoring Performance

Mondrian can log both the MDX statements sent to the Mondrian engine and the SQL generated and sent to the database. Like many Java applications, Mondrian uses the common logging framework log4j.

Although Mondrian logging is turned off by default, it can be enabled by configuring the log4j files, as detailed in Analysis SQL Output Logging.

To turn on Mondrian logging, stop the Pentaho server, edit the log4j.xml file, and at the bottom of the file, uncomment the loggers that you want to have logged. Restart the server after changing the log settings. After this, query performance will be logged and you will be ready to begin performance tuning.

Optimizing the Database

Because Mondrian eventually retrieves data from a relational database, that database can be a performance bottleneck, so that is the first place to start looking for performance enhancement possibilities.
Mondrian works with many database types and each vendor’s database has specific methods of performance tuning. Your database administrator will be a valuable resource in tuning your specific database, but we will also discuss common issues you can look for when dealing with database performance while using Mondrian.

*Several database vendors provide appliances and features designed for analytic queries, automating the indexing, clustering, and optimization necessary for high-speed interaction and scalability. Using these products can benefit large-scale, intensive Mondrian applications.*

To optimize all types of queries, look at the timings in the log files. Since the Mondrian MDX and SQL logs can be configured to display how long each query is taking, try executing some slow Mondrian queries and then review the execution times for both MDX and SQL. If the SQL query is a significant portion of the time, this is an indication that the database needs optimization and/or performance tuning.

If the SQL queries are a problem:

**Troubleshoot:** Begin the optimization process by running the same query in the native database tools.

This will determine if some database-related problem exists that is not the database itself. For example, there may be a problem with the database driver configuration that is separate from the database itself.

**Troubleshoot:** Since Mondrian retrieves data from a star schema, another step to take is to make sure your indexes are properly created. A query that takes several minutes without indexing may return within seconds with properly configured indexes.

Surrogate keys in dimension tables should always be indexed, as these are typically the primary keys for the dimension.

If there are natural keys that will be used for joins in the dimension table, index these natural key columns as well.

**Aggregate Tables**

Once the database has been tuned and is running as fast as possible, the next step to tune Mondrian is to use aggregate tables. Because it is typical to store the data at the lowest grain possible, analytic databases often contain millions of records. This implies that a detailed record of transactions is stored so that the measurements can be rolled up into useful business metrics.

For example, if a data warehouse stores details of an individual sales transaction, it would be useful for a business executive to be able to view the data in aggregate based on daily, weekly, or monthly trends. That same executive may also need to view details of a specific product on a particular day. Even though we have detailed transactions, there is still a need to view the data at a higher level. We can still roll up the data at higher levels to display longer-term trends, even if the data is stored at the detailed (lowest grain) level.

The ability to roll up detailed transactions into higher levels allows Mondrian to precompute values and thus improve overall performance. You can specify aggregate tables that precalculate low-level
detailed transactions into a higher level for analysis, allowing Mondrian to get higher-level details from an aggregate table and finer details from a detailed table, as shown in Figures 3 and 4:

Figure 3: Detail Tables

Figure 4: Aggregate Table

An aggregate table is physically created in the database and is then normally populated as part of the ETL process. There is nothing special about the data in an aggregate table; Mondrian simply uses the data in such a table when it has been configured to do so and the query can be answered by the table.

To enable the use of aggregate tables in Mondrian, edit the `mondrian.properties` file and set the following to true:
mondrian.rolap.aggregates.use = true
mondrian.rolap.aggregates.read = true

A couple of different approaches can be used to populate the aggregate table, but the best method is to first populate the detailed fact table from the operational data store and then populate the aggregate table from the detailed fact table. This approach ensures internal consistency between the fact table and the aggregate tables.

Although it may be tempting to create many aggregate tables, we recommend you create only those aggregate tables where you want your intersections precalculated.

Overall performance will be significantly improved if you first do your performance testing, identifying which queries need some performance help, and then create only those aggregate tables necessary to speed up slow queries, monitoring the performance impact as you go.

Further information on Mondrian aggregate tables is available in the Pentaho documentation: What are aggregate tables?

Tuning the Mondrian Schema

A Mondrian schema defines the objects that your users will be able to query. The schema is typically a set of columns and tables for which your users will be expecting to get fast aggregate results. It is important that your data source be ready to answer queries to those columns quickly, at the levels defined. This is normally accommodated by the star schema design pattern in the database; however, care should be taken at the model level to make data available at manageable segments.

For example, the hierarchy has Year, Quarter, and Month levels. This allows users to access monthly aggregates rapidly rather than having to set up their own filters:

```xml
<Dimension name="Time" foreignKey="time_id">
  <Hierarchy hasAll="false" primaryKey="time_id">
    <Table name="time_by_day"/>
    <Level name="Year" column="the_year" type="Numeric" uniqueMembers="true"/>
    <Level name="Quarter" column="quarter" uniqueMembers="false"/>
    <Level name="Month" column="month_of_year" type="Numeric" uniqueMembers="false"/>
  </Hierarchy>
</Dimension>
```

Parent-child hierarchies are a known challenge for analytics, but Pentaho documentation has further info on how to tune these hierarchies.

Approximate Cardinality

Mondrian needs to know the relationship between groups of data to answer certain queries. While Mondrian typically queries the data source to learn this relationship, the approxRowCount attribute can make a significant improvement by setting appropriate expectations for Mondrian.
When evaluating an MDX query, Mondrian often needs to know how many distinct values of an attribute are present in the star. It will use this information to determine whether to load all members of a level rather than just those explicitly requested, for example. In many cases, it is good enough to have a reasonably close guess. Setting the approximate cardinality in the schema can help you avoid the SQL queries needed to retrieve the exact value.

**Optimizing the Mondrian Configuration**

The following sections cover different strategies for optimizing your Mondrian configuration:

- Memory
- Caching
- Mondrian Configuration Properties
- Native Evaluation Properties

**Memory**

Memory (RAM) is very important for Mondrian performance. Mondrian uses a variety of in-memory caching techniques to internally optimize throughput. However, physical memory management is up to the operating system. If there is too little memory, the operating system will swap page files to the hard drive and performance will drop dramatically. Increasing memory is one of the easiest, lowest-cost approaches to improving performance.

*The Java Virtual Machine (JVM) that contains Mondrian should have at least 6GB accessible for typical enterprise use. JVMs larger than 24GB are not recommended without special monitoring and settings to manage Java's garbage collection.*

Instructions for increasing the memory allocation are available at the [Pentaho Tuning Guide](#).

**Caching**

Whereas aggregate tables reduce the amount of data read from the database, caching can eliminate the need for reading entirely by storing the data in memory. Because memory access is thousands of times faster than disk access, this can lead to an order of magnitude increase in performance. Mondrian has three different types of cache.

*Table 1: Mondrian Cache Types*

<table>
<thead>
<tr>
<th>Cache Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema cache</td>
<td>Stores the schema in memory after the schema is read the first time. The schema will be kept in memory until the cache is cleared, so it is important to clear this cache when you update a schema.</td>
</tr>
<tr>
<td>Member cache</td>
<td>Stores members of dimensions in memory, and is populated when members of dimensions are first read.</td>
</tr>
<tr>
<td>Segment cache</td>
<td>Holds data from the fact table, and has the biggest impact on improving performance. There is also support for extended segment caches that allow memory to be accessed on distributed local and networked JVMs.</td>
</tr>
</tbody>
</table>
Mondrian will automatically update the caches as schemas and dimensions are read and aggregates are calculated. This means that the first user to access the data is populating the cache, rather than getting the benefits of using it. Therefore, before business users start performing analysis, you will want to prepopulate the caches.

By reviewing the log files, you can monitor Mondrian for slow-performing queries. Call all the slow-performing Mondrian queries from a script, populating the cache before business users access the data, to keep the users from experiencing the slowdowns associated with populating the cache themselves.

Large and complex data models with many defined dimensions, levels, and measures can quickly outgrow memory that is available for cache. Since Mondrian allows users to query the data in combinations that are not predetermined, there are countless potential results. Although results are reused where possible, the database must still respond quickly in the event results are not found cached, so use analytic databases, optimized for this purpose, where you can.

**Mondrian Configuration Properties**

Mondrian has a set of internal parameters which guides its general operation. While we do not recommend you change the defaults for general use, you can access the parameters for specific advanced cases where you need certain insights about Mondrian's behavior or to perform certain tests.

Some of the settings you can change, if necessary, include:

```properties
# Maximum number of simultaneous queries the system will allow.
mondrian.query.limit=100

# mondrian.properties
mondrian.result.limit=15000000
mondrian.olap.FetchChildrenCardinality=false
```

In addition, you can configure these properties to best fit your needs:

*Table 2: Mondrian Configuration Properties*

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mondrian.rolap.maxConstraints</td>
<td>The maximum number of items allowed in an IN list in SQL, defaulting to 1000. For many relational database management systems (RDBMS) such as Postgres or MySQL, this number can be set much higher. A larger IN list may allow Mondrian to execute better-constrained queries.</td>
</tr>
<tr>
<td>mondrian.rolap.EnableInMemoryRollup</td>
<td>With this property enabled, Mondrian will attempt to fulfill new segment requests using already-cached segments, even if there is not an exact match. For example, if two segments have been cached, one with unit_sales for 2017 Q1 and Q2, and another with unit_sales for Q3 and Q4, then a query for all of 2017 can be fulfilled by rolling up the two segments.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mondrian.rolap.groupingsets.enable</td>
<td>If this is set to <code>true</code>, certain databases such as Teradata or Oracle will attempt to load multiple levels of aggregation in a single SQL query using the <code>GROUPING SETS</code> syntax.</td>
</tr>
<tr>
<td>mondrian.rolap.cellBatchSize</td>
<td>This property, which defaults to 100000, controls the number of cells Mondrian will batch together during evaluation before firing a fact table query. In most cases, 100000 is more than adequate, but for some workloads with large dimensions and/or deeply nested reports, you may exceed this limit. Long-running MDX that has an unexpectedly large number of queries against the fact table may indicate that the batch limit is being hit frequently. Raising this value should be done with care, since it raises the amount of resources (memory/CPU) that Mondrian requires during evaluation.</td>
</tr>
</tbody>
</table>

**Native Evaluation Properties**

Native evaluation is one of the more important features of Mondrian performance. Several relevant properties control this capability, governing whether Mondrian will attempt to push down constraints while loading dimension members, reducing the amount of evaluation that must be performed.

For example, suppose a report includes customers and products on the rows, filtered by January 2017. The non-native method Mondrian would use to evaluate such a query would be to retrieve all customers and products separately, construct a set with all possible combinations, and then iterate through each. Even for reasonably small dimensions, this could easily be millions of combinations. With native evaluation, Mondrian will execute a SQL query for just those customers and products that have the fact data for January 2017. This is typically a much smaller set of items to evaluate, creating a great performance benefit.
The following properties control the distinct types of native evaluation:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mondrian.native.crossjoin.enable</td>
<td>Determines whether push down optimizations will be used with a crossjoin in a NON EMPTY context.</td>
</tr>
<tr>
<td>mondrian.native.topcount.enable</td>
<td>Used to enable push down and limiting of results for queries using the TopCount MDX function.</td>
</tr>
<tr>
<td>mondrian.native.filter.enable</td>
<td>If this is set to true, Mondrian will attempt to push filter constraints to the database for set evaluation. Only certain forms of filter constraints are supported, such as simple base measure filtering like [Measures].[Unit Sales] &gt; 100.</td>
</tr>
<tr>
<td>mondrian.native.nonempty.enable</td>
<td>Determines whether to attempt push down of the MDX WHERE clause constraint when evaluating a set in a NON EMPTY context.</td>
</tr>
<tr>
<td>mondrian.native.ExpandNonNative</td>
<td>Many MDX expressions cannot be directly evaluated natively. ExpandNonNative allows sub-expressions to be evaluated non-natively, with their results used in a parent expression for native evaluation. For example, if a crossjoin involves two HEAD() sub-expressions, Mondrian will non-natively evaluate the two HEAD functions, and then plug the results as a simple set into the crossjoin. The resulting expression can then be natively evaluated.</td>
</tr>
</tbody>
</table>

Related Information

Here are some links to information that you may find helpful while using this best practice document:

- Analysis SQL Output Logging
- Mondrian Documentation
- Pentaho Tuning Guide