Introduction to Mondrian Performance Tuning

This page intentionally left blank.

Contents

This page intentionally left blank.

Overview

Business Analytic Tools such as Mondrian provide a competitive advantage in the analysis of historical trends and data by gaining insight into key performance metrics. The traditional tools used for business analytics have often been expensive and difficult to maintain and use. Mondrian, however, is a web-based open-source business analytic tool that enables organizations to have quick and easy interactive analysis of critical business data. Users can securely interact with large volumes of business data to gain valuable insight into their company's vast data without waiting for IT or database administrators, once it is set up.

Analyzing large quantities of data is much easier with Mondrian. The system responds to queries fast enough to allow for an interactive exploration of the data, even with millions of rows, occupying several gigabytes.

Software	Version
Pentaho	5.4, 6.x, 7.x
Mondrian	3.7

Maximize Performance with Mondrian

Mondrian supports faster analytics so that users see their report updates quicker. Performance is extremely important, since analysis is done over millions of records.

- Strategies for Increasing Mondrian performance
- Performance Improvement Process
- Monitor Performance
- Tune the Mondrian Schema
- Optimize the Database
- Aggregate Tables

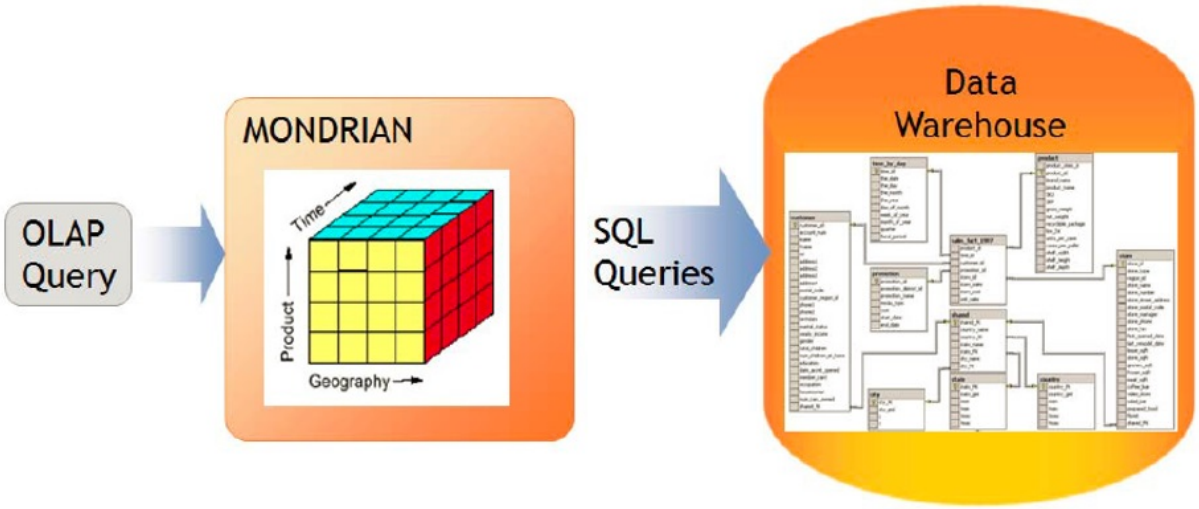

Figure 1: Queries for Mondrian

This figure shows how an OLAP MDX query is sent by an application like Pentaho Analyzer. The Mondrian engine receives the query and checks its internal cache for 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. Mondrian performance is very good for a wide variety of datasets and queries, with proper Mondrian caching and a well-designed star schema. By default, Mondrian will perform some caching to improve overall throughput, but enhancing performance even further requires some additional configuration.

A Mondrian schema is a map to your data. That schema contains a logical model, consisting of cubes, hierarchies, levels, measures and members, and a mapping of this logical model onto a physical model in the database. The logical model consists of the constructs used to write queries, such as cubes, dimensions, hierarchies, levels, 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 the diagram below:

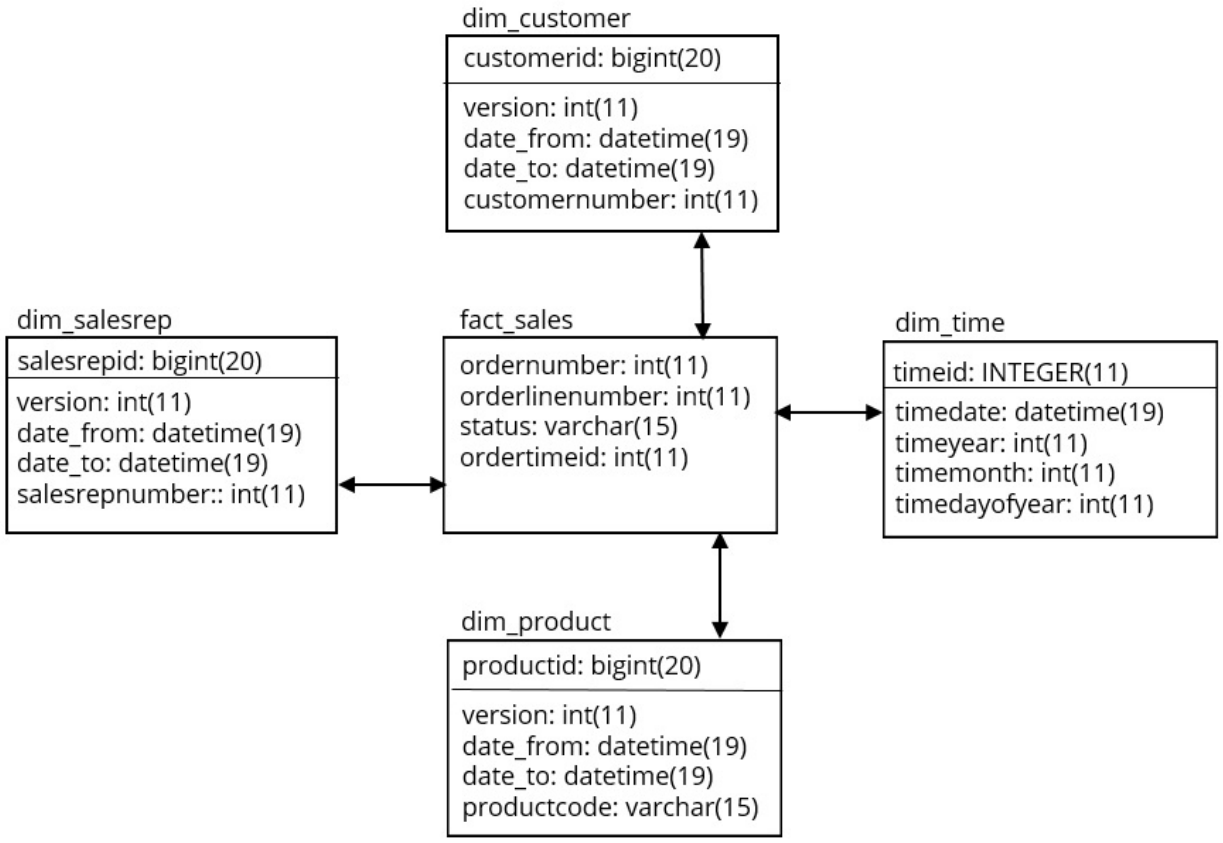

Figure 2: OLAP MDX Query – Star Schema

Strategies for Increasing Mondrian Performance

The following are main strategies for increasing Mondrian performance, given a strong foundation of hardware resources and system fundamentals, such as low-latency networks:

- Optimizing the Database
- Optimizing the Mondrian Schema
- Optimizing Mondrian configuration
    - Mondrian Settings
    - Using in-memory caching to improve throughput
    - Configuring clusters for scalability

This document will discuss all three approaches and define techniques you can use to optimize Mondrian performance.

Performance Improvement Process

There are five high-level steps to perform when evaluating Mondrian performance:

1. Prepare for performance testing by setting up a test environment and preparing the data. It is helpful to have a dedicated environment for testing that is separate from your development environment that does not include external factors possibly impacting performance.

2. Evaluate the current performance by running queries in your test environment and capture the performance metrics, once you are set up. It is important to have a record of these baselines in order to have real numbers to measure optimizations against.

3. Start by tuning the database if performance is not satisfactory. Ensure that the indexes are created and your database optimizer is aware of your star schema configuration.

4. The next step is to tune the Mondrian schema if tuning the database does not solve the problem. Later sections in this document on aggregate tables and caching will discuss how to speed up Mondrian queries.

5. Finally, look for alternative ways of presenting the information, such as breaking up the data into multiple cubes or doing analysis with preset filters if the above steps do not solve the performance issues.

Monitor Performance

Mondrian can log the MDX statements sent to the Mondrian engine and log the SQL generated and sent to the database. Mondrian uses the common logging framework `log4J`, like many Java applications. Mondrian logging is turned off by default, but can be enabled by configuring the `log4J` files. Analysis SQL Output Logging in Pentaho Documentation has more information on logging.

1. To turn on logging, edit the `log4j.xml` file.
2. Go to the bottom and uncomment the loggers you want to have logged.
3. Be sure to restart the server after changing the log settings. Query performance will be logged and you're ready to begin performance tuning, with the logging set up.

Tune the Mondrian Schema

A Mondrian schema defines the objects that your users will be able to query. This typically translates to a set of columns and tables that your users will be expecting to quickly get aggregate results for. It is important that your datasource be ready to answer queries quickly to those columns, 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 below has `Year`, `Quarter` and `Month` levels. This allows users to quickly access monthly aggregates rather than setting up their own filters:

```
<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. The Mondrian article on Closure tables has more information on these hierarchies.

Approximate Cardinality

Mondrian must know the relationship between groups of data to answer certain queries. Mondrian typically queries the data source to learn this; however, the `approxRowCount` attribute can make a significant improvement by setting expectations for Mondrian.

Mondrian often needs to know how many distinct values of an attribute are present in the star schema when evaluating an MDX query. It will use this information to determine whether to load all members of a level rather than just those explicitly requested. In many cases, it's good enough to have a reasonably close guess. Setting the approximate cardinality in the schema can avoid certain SQL queries needed to retrieve the exact value. The Mondrian article has more information on setting the approximate cardinality.

Optimize the Database

A relational database can be a performance bottleneck, since Mondrian eventually retrieves data from it. Therefore, it is the first place to start looking for performance enhancements. Mondrian works with many databases, and each vendor's database has specific methods of performance tuning. Your database administrator will be a valuable resource in tuning your specific database. We will discuss the common issues to look for when dealing with database performance while using Mondrian.

Several database vendors provide database appliances and features that are designed for analytic queries. These products help automate the indexing, clustering, and optimization necessary for high-speed interaction and scalability. They are a best practice data source for large-scale intensive Mondrian applications, when available and cost-justified.

A good starting point for optimizing all types of queries would be the timings in the log files. The Mondrian MDX and SQL logs can be configured to display how long each of those queries is taking. Therefore, execute some slow Mondrian queries and review the execution times of both MDX and SQL. The database needs optimization or performance tuning if the SQL query consumes a significant amount of time. Begin the optimization process by running the same query in the native database tools if the SQL queries are a problem. 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.

Another best practice is to make sure your indexes are properly created, since Mondrian eventually retrieves data from a star schema. 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. A query that takes several minutes without indexing may return within seconds with properly configured indexes.

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. Analytic databases often contain millions of records, since it is typical to store the data at the lowest grain possible. 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 want to view the data in aggregate, based on daily, weekly or monthly trends. That same business executive may also need to view details of a specific product on a particular day. There is a need to view the data at a higher level, even though we have detailed transactions. We can still roll-up the data at higher levels to display longer term trends by storing data at the detailed level.

The ability to roll-up detail transactions into higher levels allows Mondrian to precompute values and thus improve overall performance. Mondrian allows you to specify aggregate tables that pre-calculate low-level detailed transactions into a higher level for analysis. Thus, Mondrian can get higher-level details from an aggregate table and get the finer details from a detailed table, as shown in the figure below:

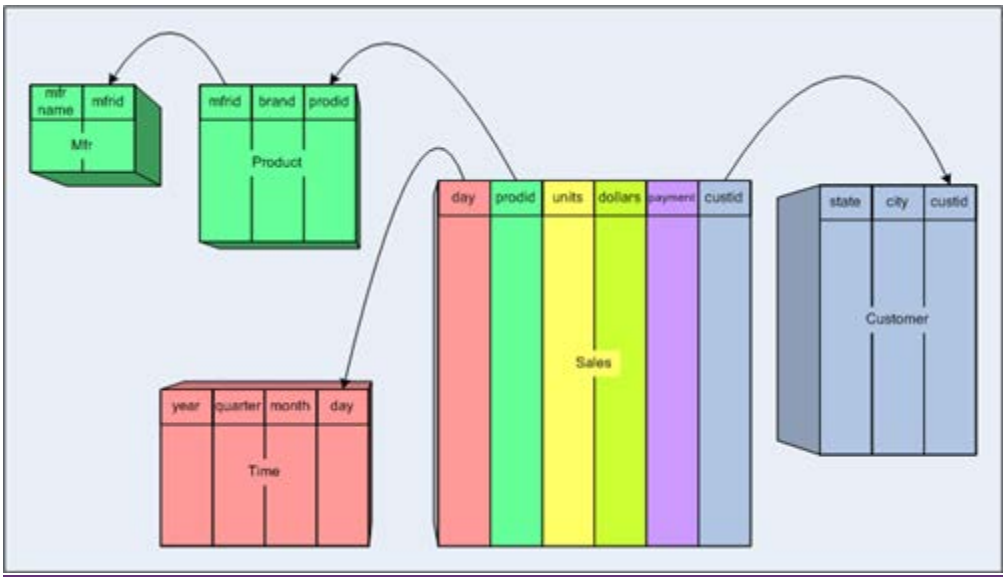

Figure 3: Example Detail Tables

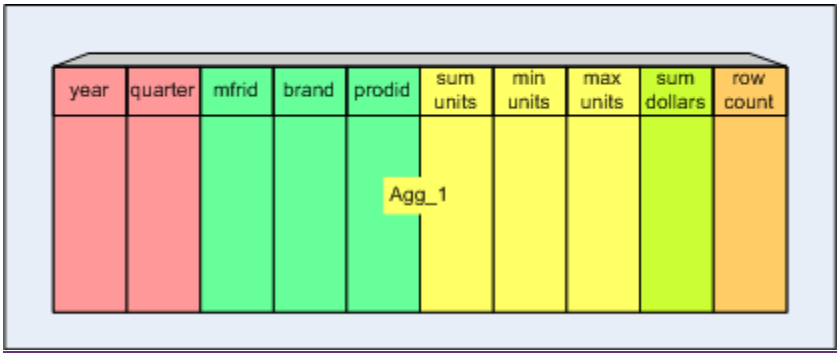

Figure 4: Example Aggregate Table

The physical creation of an aggregate table is done in the database and is 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 the aggregate table when it has been configured to do so and the query can be answered by the aggregate 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
```

There are a couple of approaches that can be used to populate the aggregate table. Best practices are 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. The Mondrian article will provide more information aggregate tables.

It is not recommended to create many aggregate tables. The best practice would be to determine your particular use case. Define only the aggregate tables where you want intersections to be pre-calculated and thus significantly improving overall performance. Your performance testing should identify which queries could use some performance help. Aggregate tables can be added after the fact to speed up slow queries. Start with the queries, create an aggregate table, and then see the impact on overall performance.

Optimize Mondrian Configuration

This section provides information about optimizing throughput performance, and eliminating entire reads, using in-memory caching and basic caching. There is also information on the importance of 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. The OS will swap page files to the hard drive and performance will drop dramatically, if there is insufficient memory. Increasing memory is one of the easiest low-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 GC.

Cache

Caching can eliminate reads entirely by storing the data in memory, whereas aggregate tables reduce the amount of data read from the database. Because accessing memory is thousands of times faster than accessing disk, this can lead to an order of magnitude gain in performance.

Mondrian has three different types of cache:

- **Schema Cache:** The schema cache stores the schema in memory after it has been initially read, and it will be kept in memory until the cache is cleared. This implies that whenever you update a schema, you need to clear the schema cache.
- **Member Cache:** The member cache stores members of dimensions in memory and is populated when members of dimensions are initially read.
- **Segment Cache:** The segment cache 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.

Mondrian will automatically update the caches as schemas and dimensions are read and aggregates are calculated. This means 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 should pre-populate the caches. You can monitor Mondrian performance for slow-performing queries by reviewing the log files. All reports in Pentaho are URL-addressable including Pentaho

Analyzer which is based on Mondrian. A best practice would be to call all the slow-performing Mondrian queries from a script populating the cache before business users access the data. In this way, the Mondrian caches will be populated and ready for action before your business users use the data.

Large and complex data models with many defined dimensions, levels, and measures can quickly outgrow memory that is available for cache. Mondrian allows users to query the data in combinations that are not pre-determined. This results in an exponential number of possible results. Mondrian reuses results efficiently where possible, however, when results are not found in cache, the database is expected to quickly provide the desired combination. Analytic databases are optimized for this purpose and are a best practice when available.

Mondrian Configuration Properties

Mondrian has a set of internal parameters that guides its general operation. These parameters are accessible for advanced use cases where there are insights about Mondrian's specific behaviors and where testing can be done. It is not recommended to change the defaults for general use. Below is an example of the maximum number of simultaneous queries the system will allow:

- `mondrian.query.limit=100`
- `mondrian.properties`
- `mondrian.result.limit=15000000`
- `mondrian.olap.FetchChildrenCardinality=false`

Other Important Properties

Here is a group of other properties that are vital to Mondrian's performance.

- `mondrian.rolap.maxConstraints`: This is the maximum number of items allowed in an IN list in SQL, defaulting to 1000. For many RDBMs, such as Postgres, MySql, this number can be set much higher. Larger IN lists allow Mondrian to execute better constrained queries in some cases.

- `mondrian.rolap.EnableInMemoryRollup`: Mondrian will attempt to fulfill new segment requests using already cached segments, with this property enabled, even if there is no exact match. For example, if two segments have been cached, one with `unit_sales` for 2015 Q1 and Q2, and another with `unit_sales` for Q3 and Q4, then a query for all of 2015 can be fulfilled by rolling up the two segments.

- `mondrian.rolap.groupingsets.enable`: Certain databases, such as Teradata, Oracle, and DB2, will attempt to load multiple levels of aggregation in a single SQL query using the GROUPING SETS syntax, if this property is set to true.

- `mondrian.rolap.cellBatchSize`: This property defaults to 100000. It 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 or deeply nested reports, it is possible to 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 the value should be done with care, since it raises the amount of resources (Memory/CPU) that Mondrian requires during evaluation.

Native Evaluation Properties

Native evaluation is one of the more important features of Mondrian performance. There are many relevant properties controlling this capability. These properties govern whether Mondrian will attempt to push down constraints while loading dimension members, reducing the amount of evaluation Mondrian needs to perform. For example, suppose a report includes Customers and Products on the rows, filtered by January 2015. 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. This could easily be millions of combinations, even for reasonably small dimensions. Mondrian will execute an SQL query for just those customers and products that have fact data for just January 2015, with native evaluation. This is typically a much smaller set of items to evaluate, and can have great performance benefits.

The properties below control the various types of native evaluation:

- `mondrian.native.crossjoin.enable`: This property determines whether push-down optimizations will be used with a `crossjoin` in a NON EMPTY context.
- `mondrian.native.topcount.enable`: This property is used to enable push-down and limiting of results for queries using the `TopCount` MDX function.
- `mondrian.native.filter.enable`: Mondrian will attempt to push filter constraints to the database for set evaluation, if this property is true. Only certain forms of filter constraints are supported, such as simple base measure filtering like `[Measures].[Unit Sales] > 100`.
- `mondrian.native.nonempty.enable`: This property determines whether to attempt push-down of the MDX `WHERE` clause constraint when evaluating a set in a NON EMPTY context.
- `mondrian.native.ExpandNonNative`: Many MDX expressions cannot be natively evaluated directly. `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.

Related Information

- Analysis SQL Output Logging
- Mondrian article