HITACHI
Inspire the Next

Pentaho Analyzer and Impala Data Source

HITACHI
Inspire the Next

Change log (if you want to use it):

Date	Version	Author	Changes

Contents

This page intentionally left blank.

Overview

This document covers some best practices on using Pentaho Analyzer against Impala data sources. In it, you will learn how to prepare data at the Hadoop Distributed File System (HDFS) level, partition your data, set the configuration for Impala and Mondrian. You will also learn about schema recommendations and settings for Analyzer.

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

Software	Version(s)
Pentaho Analyzer	6.x, 7.x, 8.0
Cloudera Impala	CDH 5.3 through CDH 5.7

The Components Reference in Pentaho Documentation has a complete list of supported software and hardware.

Cloudera Impala Recommendations

This section contains our recommendations for using Impala as a data source for Pentaho Analyzer. You can find details on these topics in the following sections:

Use Distributed Computing Architecture Concepts

For performance and efficiency reasons, Cloudera's current Impala planner (execution optimizer) requires the largest fact table to be listed first in the `FROM` clause. Tables listed after the first `FROM` clause table, which are typically the smaller dimension tables, are broadcast to all nodes that participate in the hash join.

Cloudera's later versions of the Impala optimizer introduced improvements in the query planner to prevent this from happening. Pentaho's online analytical processing (OLAP) engine generates structured query language (SQL) at runtime, so you cannot depend on table order to follow this Impala-imposed performance requirement.

Here are a couple of ways to list the largest table first:

- Use a view.
- De-normalize the tables to a single table to avoid joins at runtime.

We recommend using a de-normalized table, making all fields available in one table, and creating all dimensions as degenerated dimensions. The de-normalized table, along with the use of parquet (columnar storage) as the file format, will perform well and use storage efficiently.

Partition and Distribute Your Data Properly

Partition your Impala tables on commonly used fields. It is also helpful to set the data file sizes to be approximately the same as the recommended HDFS block size.

These general recommendations work well for Hive tables. You should also consider using Impala with parquet files, for Impala performance.

Impala Over Parquet Files

Using Impala over parquet files requires you to read from HDFS, and to build the tuple data. Typically, the HDFS reading is done quickly, but building the tuples can be a high-cost process. For example, if most of the data queried is stored in one or two nodes, those nodes will peg the Central Processing Unit (CPU) to build the tuples while other nodes are idle.

Because of this high cost, the parquet file size for the table needs to be reduced so that any partition query can distribute the activity to multiple nodes. This allows more data nodes to participate in the scans, aggregations, etc.

Make sure that you do not over-reduce the file size.

Although reducing file size helps to distribute work to multiple nodes, generating too many small files can cause other issues. Evaluate each use case and dataset for the best selection of partition fields and file sizes.

This document's section on Mondrian Schema Design recommendations has more information about partition-related topics. Here is an example to help illustrate:

> *If you have a dataset that is partitioned by `[YEAR, MONTH]`, after materialization of the parquet files, your files are about 256MB in size. However, a single month of data fills up a single file/node. If you perform queries for your current month, the pruning process pushes that work to the data node where your data file is stored. This can create a bottleneck on that single node, and in many cases, the node can go "hot."*
>
> *However, if you reduce the file size to about 32MB, your files will get distributed to multiple nodes, your queries will perform much faster, and the load on any individual node will be reduced accordingly.*

Refresh Statistics and Metadata

You should consider performing the following commands after changing data, ingesting new data, or data rebalancing or partitioning: `COMPUTE STATS` and `INVALIDATE METADATA`.

COMPUTE STATS

This command gathers information about the volume and distribution of data in a table and all associated columns and partitions. The information is stored in the metastore database, and is used by Impala to help optimize queries.

After you load new data into the partition, use `COMPUTE STATS` on an entire table or on the partition. The Cloudera documentation on Performance Considerations for Join Queries has more details on using different statistics.

*A known issue prior to Impala version 1.3.1 could cause excessive memory usage during a `COMPUTE STATS` operation on a parquet table. **Workaround**: Issue the command `SET NUM_SCANNER_THREADS=2` in the Impala-shell before issuing the `COMPUTE STATS` statement. Then issue `UNSET NUM_SCANNER_THREADS`, before continuing with queries.*

`INVALIDATE METADATA`

Use `INVALIDATE METADATA` if data was altered in a more extensive way, such as being reorganized by the HDFS balancer, to avoid performance issues like defeated short-circuit local reads.

If you use Impala version 1.0, the INVALIDATE METADATA statement works just like the Impala 1.0 REFRESH statement did. The Impala 1.1 REFRESH is optimized for the common use case of adding new data files to an existing table, and now requires the table name argument.

Enable Your Hadoop Cluster for Short-Circuit Reads

Reads will typically go through the `DataNode` in `HDFS`. When a client asks the `DataNode` to read a file, the file is a read-off of the disk and gets sent to the client over a Transmission Control Protocol (TCP) socket. Short-circuit reads bypass the `DataNode` and allow the client to directly read the file.

Set up your Hadoop cluster for short-circuit reads, on the client and on the `HDFS DataNode`, by enabling `libhadoop.so`. Apache's documentation on Native Libraries has more information about enabling this library.

JDBC and Parameter Recommendations

We recommend using Cloudera's Java Database Connectivity (JDBC) driver and disabling JDBC database pooling.

If you have not set a specific memory limit, Impala will create its own default limit of `1024`.

Implement an Impala Balancer

Using a load-balancing proxy server for Impala has the following advantages:

- Applications connect to a single well-known host and port, rather than keeping track of the hosts where the Impala daemon is running.
- If any host running the Impala daemon becomes unavailable, application connection requests still succeed because you always connect to the proxy server rather than a specific host running the Impala daemon.
- The coordinator node for each Impala query potentially requires more memory and CPU cycles than the other nodes that process the query. The proxy server can issue queries using round-robin.

More information on load-balancing Impala can be found in Impala Concepts and Architecture.

Snappy Compression

Use Snappy compression codecs in conjunction with container file formats that support splitting. Snappy alone does not support splitting. Large input files should not be compressed with just Snappy. Use it with container file formats that support both compression and splitting, such as Parquet or AVRO. Snappy compression is enabled by default in Cloudera.

Pentaho Settings and Recommendations

This section describes all settings and recommendations for your Pentaho Server(s) to improve user experience, reduce potential errors, and reduce the number of queries sent to your underlying data source (Impala).

Analyzer Settings

Analyzer tasks run concurrently in independent threads. It uses a maximum of three concurrent tasks by default, and when more tasks are required, they are queued. The maximum queue size by default is `100`.

There are some recommended settings for Analyzer to reduce the number of queries issued for Impala, and to increase the number of concurrent implementations of Analyzer views. These are the properties that handle queues in your `analyzer.properties` file:

- **`report.request.service.core.pool.size=3`**
  - Core number of threads in the pool
  - *Recommendation:* Keep the default settings.
- **`report.request.service.max.pool.size`**
  - Max number of threads in the pool
  - *Recommendation:* Make sure the value is 80% of the thread cores on the system. For a 16-core server, we recommend setting this parameter to 14.
- **`report.request.service.queue.size`**
  - Request queue size is the MAX number of items that can be in queue. Queue is processed in as many parallel threads as we have set up in the `report.request.service.max.pool.size`, and the remaining requests are queued.
  - Users will get a message saying cannot process due to limitation reached in cases of queue overloads.
  - *Recommendation:* Increase the queue size to 1000.
- **`filter.dialog.apply.report.context`**
  - When showing the list of available members in the filter dialog, limit the members by report measures and attribute filters.
  - *Recommendation:* Keep this value set to false.

Use APIs to Tune the Values

The following API provides information of the running values, in real-time. Use it while performing stress tests to see the queue and the remaining queues.

```
http://${BASE_PENTAHO_URL}/api/repos/xanalyzer/service/admin/pool
```

Mondrian Schema Design Recommendations

As described in the Use Distributed Computing Architecture Concepts section, having a wide unique table is the recommendation for keeping Impala work focused on per-node individual activities, without needing to distribute smaller data tables to other nodes.

Use Partitions

When it comes to partition usage, the Impala planner requires the partition fields to be included in the SQL statement `WHERE` clause[1]. Pay special attention while building a Mondrian schema that will force the SQL `WHERE` clause to be included in the queries, whenever possible. The Mondrian engine builds the SQL dynamically, and you will not be able to declare the `WHERE` clause.

The following technique describes how to force the Mondrian engine to create the SQL that you are looking for.

Forcing Hierarchies to Use Partitions

Mondrian hierarchies define the dimension's level and dependency. Each level describes the behavior of the levels in the underlying database: the OLAP engine and the display client.

Typically, the best practice is to create Unique Keys in levels to reduce the number of `WHERE` clause fields to be included in the SQL statement. You should also create an index on the unique field. However, for Impala or partitioned data sources, it is recommended to keep the `uniqueMembers` to `FALSE`. This will make any query with lower levels include the parent level element in the SQL statement.

Make your partitions and subpartitions based on hierarchy levels, whenever possible. If you intend to use the partitions that you already have, you must make sure that the highest level is fully-qualified.

The highest level in the example below is `YEAR`. Follow along to see how this works:

> *If we have a time dimension based on `YEAR`, `MONTH`, or `DAY`, by defining each level as `uniqueMembers=false`, any query done at `DAY` level will automatically include the `YEAR`, `MONTH`, and `DAY` in the `WHERE` clause.*

[1] Mondrian dynamic SQL generation does not support `PARTITION` statements.

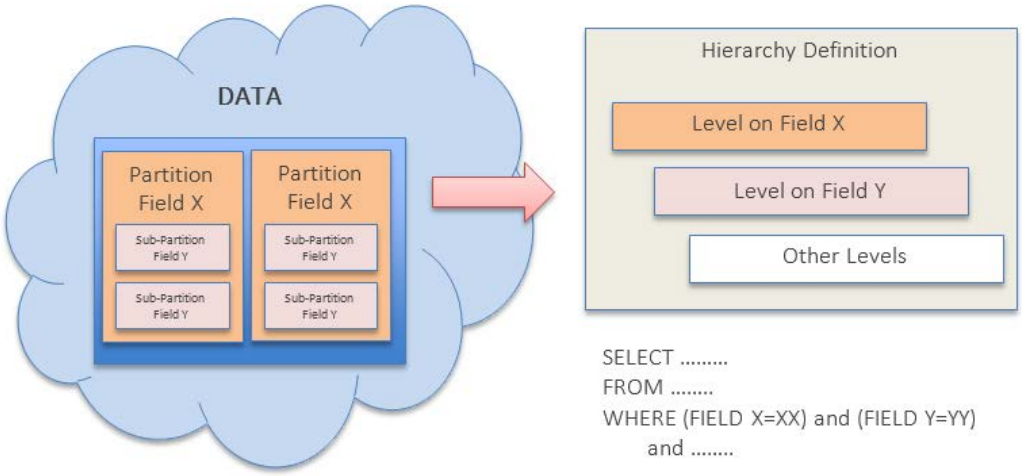

Figure 1: Example Partitioning

Model Your Hierarchy Based on Partition Design

Partitions designed at database levels are based on common field usage plus data grouping, size, and segmentation. In most cases, the selection of the fields does not follow a business logical grouping of fields. The OLAP engine will only include the partition fields if the end-user filters by the dimension elements, when the fields included in the partition belong to a different dimension's natural design. We recommend creating a hierarchy that simulates the partition design to give you additional control.

By adding a dimension that includes the same partition fields in the correct order, you can force the engine to include them in any query that is done (see the Use Partitions section). Dimension design is not restricted to partition fields only; it can include other fields, as well.

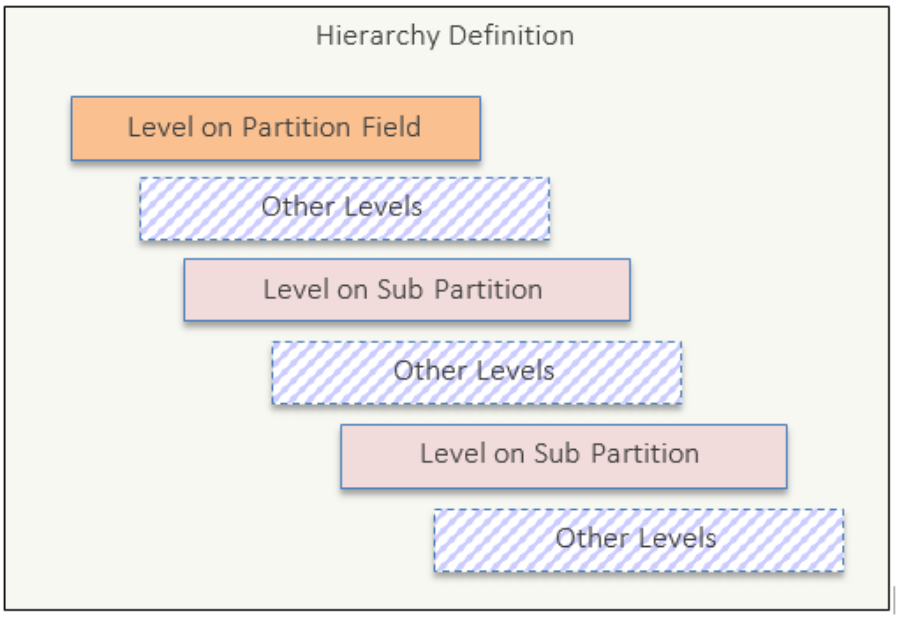

Figure 2: Hierarchy

This technique cannot be applied to every use case. It is only recommended to be used when business logic can be aligned with the partition design, forcing any selection of a subpartition to include all parent partition elements, as well.

Advanced Modeling Techniques (Impala)

This section explains advanced implementation methods based on the Streamlined Data Refinery (SDR), Session Level filtering with Dynamic Signal Processor (DSP), roles, or both. These methods make it possible to place Mondrian cubes on top of a smaller dataset. The concepts explained in this section are abstract and provide an overview of some alternatives to increase query performance.

The techniques described will constrain the cube to use a subset of the data. Therefore, it is important for the end-user to understand that when the cube/view displays it is showing all values, it refers to all values in the constrained subset.

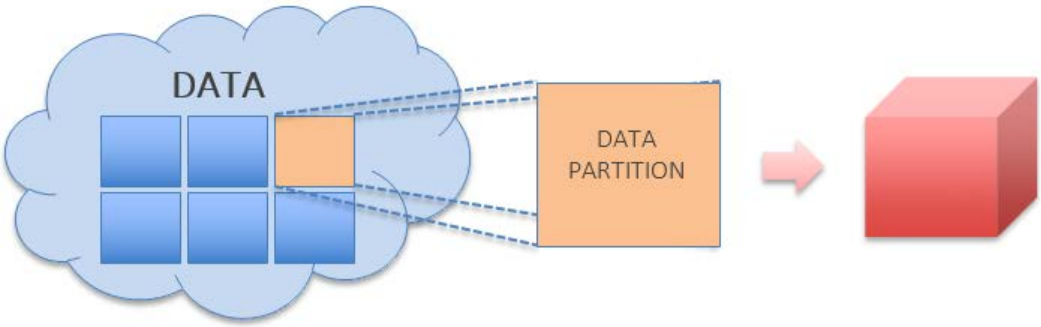

Figure 3: Cubes

As described in the section Mondrian Schema Design recommendations, make sure that Impala planner uses partitions, as this will improve performance.

Using the SQL Tag

Mondrian schema can inject `WHERE` clauses on all SQL queries. This is much easier to implement when there is one wide fact table.

Before you use SQL tags, consider how this might restrict the data that your users can view.

SQL statements are included as <SQL> tags in the <TABLE> tag as follows:

```
<TABLE name="table">
      <SQL dialect="generic">
            Statements
      </SQL>
</TABLE>
```

The inclusion of this statement can force the use of partitions. This can also be done with SDR or DSP and session variables.

Any change to the schema with DSP will generate cache segmentation. This means different cache sections will be created for every different schema that exists, without the possibility of reusing the cache.

Using Roles

Use roles in Mondrian to restrict the data that the end-user can view, and enforce restrictions to specific members on the dimensions. These roles are implemented as `WHERE` clauses when querying the `FACT` table. One difference with this approach, and previous approaches, is that roles are based on a logical model level and the permissions level can only be **With Access** or **Without Access**.

When a role is defined based on a dimension member(s), it is translated to the engine. This includes those members internally, such as when they are included in a filter statement.

Roles can help enforce filters on dimensions. Furthermore, they can act on hidden or nonvisible dimensions, as well as regular dimensions.

The maintenance of roles in a Mondrian schema can be expensive and could require the need to create a Custom Role Delegate, like this:

```
protected  Access getAccess(Member member, Access access) {
  boolean grantAccess = false;

  if (member.getHierarchy().getName().contains("Market")){
    List <Member> members = member.getAncestorMembers();
    for (Member mem : members) {
      if (territory.trim().equalsIgnoreCase(mem.getName().trim())){
        return Access.ALL;
      }
    }

    if (territory.trim().equalsIgnoreCase(member.getName().trim())){
      return Access.ALL;
    }
  }
  return Access.NONE;
}
```

When you use roles, it does not generate cache segmentation.

The Mondrian Schema Element Reference has a list of elements along with definitions.

Known Issues and Solutions

There are a couple of known issues for using Impala as a data source with Pentaho Analyzer.

Performance – Mondrian Not Hitting Cache for Non-Empty Measure Values

Due to the way that Analyzer generates non-empty filters in MDX, a lot of Analyzer queries are pushed down to the cluster.

The MDX snippet that is being generated is `Filter( SET, Not IsEmpty( [Measure] ) )`. This snippet is then translated by Mondrian into SQL as a `HAVING` clause of the type `HAVING not( sum(measure_column) is null )`. This results in many combinations of measures for different conditions, all of which must be individually cached.

Solution:

Setting the `mondrian.native.filter.enable=false` in the `mondrian.properties` file causes the non-empty clause to be performed by Mondrian, which limits the amount of different SQL queries that are sent to the cluster. While this reduces the need for excess querying the underlying data source, it pushes more work to the OLAP engine. This may not be ideal, based on the dimension sizes.

Limitation – Impala Multiple Distinct Counts

Impala does not support multiple distinct counts on the same query. Your Mondrian cube may generate such a request, in which case, the Impala query will fail.

Approximations may not be suitable for your use case.

Solution:

The solution is based on a system-wide setting that enables Impala to do `APPROX` count. Read more from the `APPX_COUNT_DISTINCT` article in the Cloudera documentation.

Related Information

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

- Apache
  - APPX_COUNT_DISTINCT Query Option (CDH 5.2 or higher only)
  - Hadoop Native Libraries Guide
  - Impala-110: Add support for multiple distinct operators in the same query block
  - Impala-488: IO Mgr should take instance memory limit into account when creating io buffers
- Cloudera
  - Impala Concepts and Architecture
  - Performance Considerations for Join Queries
  - Snappy Compression
- Pentaho
  - Components Reference
  - Mondrian Schema Element Reference
  - Segment Cache Architecture

Finalization Checklist

This checklist is designed to be added to any implemented project that uses this collection of best practices, to verify that all items have been considered and reviews have been performed.

Name of the Project:_____

Date of the Review:_____

Name of the Reviewer:_____

Considerations	Response	Comments
Are all fields used by the cube on the Mondrian schema on the same table?	YES_____ NO_____	
Did you review File Size and Distribution?	YES_____ NO_____	
Is your load process refreshing Metadata and Statistics?	YES_____ NO_____	
Have you disabled JDBC Pooling?	YES_____ NO_____	
Implemented Impala Balancer?	YES_____ NO_____	
Implemented parquet files with Snappy compression?	YES_____ NO_____	
Was your data partitioned?	YES_____ NO_____	
Is data partition by Time Dimensions?	YES_____ NO_____	
Applied Mondrian Schema design, keeping partition usage tips in consideration?	YES_____ NO_____	
Was any of the advanced partitioning methods explained here used in this project?	YES_____ NO_____	