Designing In-memory Cubes- Best practices for effective in-memory reporting and dashboard design

Mohamed Diakite Pineda
Principal Product Owner, Technology
Safe Harbor Notice

This presentation describes features that are under development by MicroStrategy. The objective of this presentation is to provide insight into MicroStrategy’s technology direction. The functionalities described herein may or may not be released as shown.

This presentation contains statements that may constitute “forward-looking statements” for purposes of the safe harbor provisions under the Private Securities Litigation Reform Act of 1995, including estimates of future technology releases. Forward-looking statements inherently involve risks and uncertainties that could cause actual results of MicroStrategy Incorporated and its subsidiaries (collectively, the “Company”) to differ materially from the forward-looking statements.

Factors that could contribute to such differences include: the Company’s ability to develop, market and deliver on a timely and cost-effective basis new or enhanced offerings that respond to technological change or new customer requirements; delays in the Company’s ability to develop or ship new products; the extent and timing of market acceptance of MicroStrategy’s new offerings; continued acceptance of the Company’s other products in the marketplace; competitive factors; general economic conditions; and other risks detailed in the Company’s registration statements and periodic reports filed with the Securities and Exchange Commission. By making these forward-looking statements, the Company undertakes no obligation to update these statements for revisions or changes after the date of this presentation.

Copyright © 2017 MicroStrategy Incorporated. All Rights Reserved.
Agenda

- Overview: In-Memory Reporting
- Modeling: Attribute Relationships
- Aggregation Behavior
- Performance Considerations: Joins, Partitioning, Dossiers
- Q&A
## Flexible Deployment Options

Cloud, On-premise and Hybrid

<table>
<thead>
<tr>
<th>MicroStrategy 10 Capabilities</th>
<th>Cloud</th>
<th>On-premise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete enterprise BI solution covering self-service, enterprise reporting, governance and scalability</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Easy for small teams and departments to set up and manage independently in minutes</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Self-service dashboards (dossiers) and data discovery.</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Sharing, collaboration, and alerting capabilities to keep everyone informed</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Access to over 300 data sources from big data lakes to personal files</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Quickly add or remove compute power as needed</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Strong security, certification, governance, and auditing</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Ability to deploy and manage at enterprise scale</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Deploy on a variety of on premises hardware and OS configurations</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Ability to physically separate installed system from all networks and the Internet</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

Overview: In-Memory Reporting
What is PRIME?

P - Parallel
R - Relational
I - In-Memory
E - Engine
## Why PRIME?

<table>
<thead>
<tr>
<th></th>
<th>PARALLEL</th>
<th>PARTITIONED</th>
<th>IN-MEMORY CUBES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ability to generate parallel queries and fetch it in parallel from the underlying source.</td>
<td>Ability to partition the data in the cube</td>
<td>Cubes with flexible schema. No pre-joins.</td>
</tr>
<tr>
<td></td>
<td>Improves the speed of cube publication</td>
<td>No 2B row limit per cube. Each cube can be divided into partitions, each partition can contain up to 2B rows</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Higher Data Throughput</strong></td>
<td><strong>Higher Capacity/Data Scalability</strong></td>
<td><strong>More Efficient, Optimized, Scalable Cubes for Building Fast Performing Dashboards</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What is PRIME?

OLAP Cubes
- Aka, the traditional Intelligent Cubes,
- Authored in Developer
- BI developers author these (not end-users)

MTDI Cubes
- Multi-Table Data Import cubes
- Authored in Web, Desktop, through data import screens
- Self-service: End-users can author these (import, wrangling, sharing)
- No necessary dependencies from project schema
OLAP In-Memory Cubes
MTDI In-Memory Cubes
## Core Differences: OLAP vs MTDI

<table>
<thead>
<tr>
<th>OLAP</th>
<th>MTDI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cube Reporting</strong></td>
<td><strong>Multiple queries</strong> (Cube Subset Instructions, CSI)</td>
</tr>
<tr>
<td></td>
<td><strong>Queries against, one single in-memory table</strong> for projection, aggregation, filtering</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Cube Data</strong></td>
<td><strong>Built with Developer</strong> as an Intelligent Cube</td>
</tr>
<tr>
<td></td>
<td><strong>Needs modeled project schema</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Supports Partitioning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Database &amp; Queries</strong></td>
<td><strong>Auto-generated</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Lookup tables</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Parallel query execution</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>One pre-joined, pre-aggregated result table is loaded only</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Parallel data fetch</strong></td>
</tr>
<tr>
<td></td>
<td><strong>DB Operations:</strong> Joins, projections, aggregations, filtering</td>
</tr>
</tbody>
</table>

* Available as a beta feature
Basic Premise: Cube Querying through CSI (SQL-like logic)

When querying MTDI or OLAP in-memory cubes internally, the dashboard in question will generate a set of instructions (Cube Subset Instructions) that are executed by the In-Memory Engine. These CSI queries are like SQL statements.
Modeling: Attribute Relationships
Applicable to MTDI Cubes
Modeling: Attribute Relationships

Types of Relationships between Attributes

- One-to-One
- One-to-Many
- Many-to-Many
Modeling: Attribute Relationships

Relevance of Relations
Modeling: Attribute Relationships

Relevance of Relations

Aggregation awareness. E.g.:

Actual metric aggregation level is Year because this metric’s level is raised to Year from the related attributes in the visualization level (Quarter).

No relationship between Year and Quarter.

Year is unrelated to Quarter; therefore, Year is not considered in the metric level resolution. It only keeps the visualization level.
On Relationships and Data Correctness...

- Data Import automatically tries to identify relationships among attributes in a table; most of the time,
- it makes sense to delete these automatically created relations, and
- define the minimum set of relations that make sense for the user.
- Having incorrect relationship can cause unexpected data.

To disable this automatic identification of relationships:

HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microstrategy\DataImport\AutoDetect

Create DWORD "NeedAutoDetect" = "0" to disable relation auto detection.
Modeling: Attribute Relationships

Invalid Relationships are Real

Relationships are validated during publication

Invalid relationships:
- Won’t have relationship table generated
- Won’t be used in CSI queries
- Are caused by
  - one-to-many relationships that have data related in many-to-many fashion.
  - one-to-one relationships are actually in the data one-to-many or many-to-many.
Many-to-Many relationships are **really NOT necessary**: 

- **Unnecessary Memory Usage**: The additional generated many-to-many relationship tables may use redundant data present already in user-uploaded tables.
- **Improved Unrelated Attribute Querying Logic**
  - **CSI auto-join** functionality will identify that if two “unrelated” attributes are in the same table, then it will use it as the relation between these two attributes.
  - **Alternative CSI** allows “joining unrelated” or many-to-many-related attributes by first doing aggregations reducing the number of rows that later are cross-joined to the unrelated attributes, avoiding very large intermediate result sets.
Aggregation Behavior
Applicable to MTDI Cubes
In-Memory Cube Reporting

Functional Considerations: MTDI Cubes – Row Count

Table “Row Count” Metrics

- MicroStrategy automatically creates a new base metric called “Row Count – [Table Name]” for every table in a MTDI cube.
- This metric is equivalent to having a column with value of 1 for every row.
- This allows to see the rows involved in any aggregation done on the grid.
- This metric is visible from Visual Insight.

"Row Count" metric: auto-generated column with 1s
In-Memory Cube Reporting

Functional Considerations: MTDI Cubes – Metric ID

Drive Aggregation on a Table

- Aggregation functions such as Count traditionally have a function parameter called “FactID” in project schema metrics; this parameter forces the SQL engine to count the given column present in the table for the provided fact.

- This can also be achieved in MTDI cubes through the “MetricID” function parameter.

Example:

![Diagram showing aggregation on a table with metrics for customers with page interactions and orders, using the MetricID parameter to count distinct months in the Order_Fact table.]

The metric counts the distinct months present in the Order_Fact table.
Performance Considerations

Joins
When querying MTDI cubes internally, the dashboard in question will generate a set of instructions (Cube Subset Instructions) that are executed by the In-Memory Engine. These CSI queries are like SQL statements.
Performance Considerations

Cube Subset Instructions are like SQL
Performance Considerations

Joins

The main factor that will affect performance when reporting off these cube is mainly the join operation. To name a few, these are the factors that can impact response time when querying MTDI cubes:

- **One-to-many relationships.** Correctly set relationships help performance avoiding doing more expensive joins between tables, and instead leveraging automatically generated relationship tables.

- **Number of tables joined in query.** Retrieving metric data from many fact tables in the MTDI cube could affect execution time as there are more join operations.

- **Security filters** will force the application of filters to ensure data is secured; if a table does not have the security attribute, the CSI/SQL engine will find a way to join necessary tables to extend the table to include the security attribute. This will cause join costs.

- **Complex filters** that will result in multiple passes and joins; if multiple filter conditions are based on multiple attributes not present in the same table, then similar to the security filter case, the filter will be applied by joining data.
Performance Considerations

Partitioning
For cube reporting, the benefit of partitioning relies on parallelizing the aggregation of data in parallel leveraging the multiple CPUs present in the Intelligence Server host machine.
Performance Considerations

Partitioning Performance Effect & Guidelines - Configure

MTDI

- **Automatic partitioning** only available for MTDI
  - Determine whether partitioning should be enabled: largest cube table is larger than 1 million,
  - If enabled, determine partition attribute
- Parallel data fetch is done per table

OLAP

- No automatic partitioning for Intelligent Cube
- Partition attribute is also used for parallel data fetch from final SQL pass: different sections of the final pass is fetched in parallel.
Performance Considerations

Partitioning Performance Effect & Guidelines

In order to take full advantage of partitioning, the **partitions of a table should be similar in size so that the processing load is distributed equally** across the parallel/map tasks when querying cubes.
Select the **correct partition attribute/table:**

- The partition attribute should be an attribute **in the largest table** (for the case of MTDI cubes); any table larger than **2 billion rows** must be partitioned.
- The partition attribute should be the **lowest-level attribute**, with the **most number of distinct elements distributed most equally across the dataset**; this attribute should be aggregated over in most or all queries, and it will almost **never be used in grids** by users directly.
- The attribute ID should be numeric, date or text.
- The partition attribute **should not be a parent attribute** of another attribute.
Performance Considerations

Partitioning Performance Effect & Guidelines – Partitioning Recommendations

Do not partition relatively small cubes

Sample case study, for an OLAP cube with 20,000 rows, a dashboard running off this cube degraded by 50% after the cube was split into 24 partitions.

- The “reduce” step cost was greater than just not parallelizing/partitioning at all.
- Follow a similar guideline to the auto-partitioning threshold: if there are more than 1 million rows in a table/cube, consider partitioning.
Performance Considerations

Partitioning Performance Effect & Guidelines – Partitioning Recommendations

Select the correct **number of partitions**:

- **Do not set partition numbers to more than half the number of (logical) CPUs.**
  - **By default**, for the cube publication process, the number of partitions **processed in parallel** is not more than half the number of CPUs in the machine.
  - More partitions **would not be processed in parallel** (as explained in section Data Processing and Storage In-Memory).

- **Increasing number of partitions ONLY does not help improve performance. By default**, to ensure performance is maintained under user concurrency, every cube query can use at most 4 CPUs (not exclusively), ensuring other CPUs have fully dedicated time for executing other tasks. This means that even if a cube is split into more than 4 partitions, no more than four “map” tasks will execute in parallel.

*There are registry settings that can modify the maximum available threads per parallelizable job; should not be modified unless there are huge Intelligence Server machines with many CPU cores. We have plans to dynamically use more threads based on host CPU configuration.*
Performance Considerations

Partitioning Performance Effect & Guidelines – Partitioning Recommendations

Identify reporting scenarios that will parallelize queries that include any of the following types of functions:

- **Additive** (e.g. Sum, Count, Min, Max) and. Additive functions can be re-aggregated.
- **Semi-additive** (e.g. Avg, StdDev, Variance, etc.) functions. Semi-additive functions are decomposed into additive functions (e.g. Avg = Sum/Count) and re-aggregated; therefore these functions can be evaluated in parallel (mapped and reduced).
- **Count Distinct** – optimizations in place.

Identify reporting scenarios that will NOT be parallelized and may see worse performance; these include cases that use non-additive functions (e.g. Median)
Performance Considerations

Dossiers
Documents have the capabilities to blend data from different datasets. This data blending or joining is done by the Document’s multi cube join engine (MCE); in addition, an MTDI cube could be a dataset of a document, and within the cube, there are joins among tables. Therefore, there two levels of joining data when running dashboards.
Performance Considerations

Dashboard Considerations: Blending vs. MTDI Joining
The general guideline is to try to **model all the data joins within the MTDI cubes**, if already using MTDI cubes, instead of relying on the data blending functionality. Data movement and staging across layers is minimized, the behavior is more consistent, and less memory is used to hold dataset relation tables at the document level.
Additional References

- **MicroStrategy Community**: Document on which this presentation was based, with more explanations: https://community.microstrategy.com/s/article/In-memory-Reporting-OLAP-MTDI-Best-Practices

Thank you 😊
Questions?