In-memory Cubes - How to fine-tune performance with parallel in-memory processing and partitioning: OLAP and MTDI Cubes

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: PRIME, MTDI, OLAP

Data Query & Data Load

Data Modeling

Data Processing & Storage

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>


#mstrworld
Overview: PRIME, MTDI, OLAP
PRIME

What is PRIME?

Parallel Relational In-Memory Engine
## Why PRIME?

<table>
<thead>
<tr>
<th>PARALLEL</th>
<th>PARTITIONED</th>
<th>IN-MEMORY CUBES</th>
</tr>
</thead>
</table>
| - Ability to generate parallel queries and fetch it in parallel from the underlying source.  
  - Improves the speed of cube publication | - Ability to partition the data in the cube  
  - No 2B row limit per cube. Each cube can be divided into partitions, each partition can contain up to 2B rows | - Cubes with flexible schema. No pre-joins. |

| Higher Data Throughput | Higher Capacity/Data Scalability | More Efficient, Optimized, Scalable Cubes for Building Fast Performing Dashboards |
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
## Online Analytical Processing (OLAP) and Multi-Table Data Import (MTDI) cubes

<table>
<thead>
<tr>
<th></th>
<th>OLAP</th>
<th>MTDI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Version</strong></td>
<td>Existed before MicroStrategy 10.0</td>
<td>New in MicroStrategy 10.0</td>
</tr>
<tr>
<td><strong>Data Volume</strong></td>
<td>In 9.x, limited to 2 billion rows</td>
<td>Each partition can have up to 2 billion rows</td>
</tr>
<tr>
<td><strong>Schema</strong></td>
<td>A single table dataset, pre-joined</td>
<td>Multiple tables, not pre-joined</td>
</tr>
<tr>
<td></td>
<td>Closer to raw data staged in-memory;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Multiple in-memory fact tables, including varying grains, many-to-many</td>
<td></td>
</tr>
<tr>
<td></td>
<td>relationship tables, and entity-relation model semantics</td>
<td></td>
</tr>
<tr>
<td><strong>Definition</strong></td>
<td>User specifies the definition of the cube on a Report Template with</td>
<td>User specifies the tables to load into memory; Attributes/Metrics are</td>
</tr>
<tr>
<td></td>
<td>Attributes, Metrics, and Filter</td>
<td>mapped to these tables</td>
</tr>
<tr>
<td><strong>Query Generation</strong></td>
<td>Intelligence Server generates the SQL to execute against the data</td>
<td>Queries submitted to the RDBMS are simple SELECT statements against</td>
</tr>
<tr>
<td></td>
<td>source (can use Multi-Source)</td>
<td>the tables—no joins among tables</td>
</tr>
<tr>
<td></td>
<td>By default, final pass of SQL joins lookup tables to fact table and</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or metric temp tables to retrieve attribute descriptions</td>
<td></td>
</tr>
<tr>
<td><strong>Partition</strong></td>
<td>Available</td>
<td>Available</td>
</tr>
<tr>
<td><strong>Parallel</strong></td>
<td><strong>Parallel Data Fetch Option</strong> set at <strong>Intelligent Cube level</strong></td>
<td><strong>Maximum Parallel Queries Per Report</strong> set at <strong>Project level</strong></td>
</tr>
</tbody>
</table>
Core Differences: In-Memory OLAP vs MTDI

**OLAP**
- Multiple queries (Cube Subset Instructions, CSI)
- Queries against, **one single in-memory table** for projection, aggregation, filtering

**MTDI**
- **Multiple** in-memory queries (CSIs) against multiple in-memory tables for **joins**, projection, aggregation, filtering
- **Auto-Join** functionality (use imported tables as relationship tables, e.g.): No explicit need to define relationships, but useful to optimize queries.
- **Not available**: Transformations, *begin/end-on-hand metrics using metric dimensionality

**Cube Reporting**
- Built with **Developer** as an **Intelligent Cube**
- Needs modeled project schema
- Supports Partitioning

**Cube Data**
- Built with **Data Import** from MicroStrategy Web
- **Does not need modeled project schema**: Self-service enabler
- Attributes can be mapped to project schema (e.g. to leverage sec filters, stand-alone filters).
- Supports Partitioning & auto-partitioning

**Database & Queries**
- One pre-joined, pre-aggregated result table is loaded only
- **Parallel query execution**
- **Parallel data fetch**

**DB Operations**
- Joins, projections, aggregations, filtering

**MTDI**
- **Auto-generated** lookup tables
- **Auto-generated** relationship tables
- **Indices**
- **Parallel query execution**
- **Parallel data fetch**
- **Raw table data is loaded to memory**

*Available as a beta feature*
Data Query and Load
Data Query stages of Intelligent Cubes:

- **Query generation and execution**: Outside of all the SQL generation VLDB settings available for all report/cube executions, in particular the following can help optimize this step:
  - **Parallel Query Execution** (VLDB setting under Query Optimizations): See KB44052: How does the feature in the MicroStrategy 9.3.x SQL engine: "Parallel Query Execution" work.

- **Data fetch from the data source**: this involves the actual movement of data from the data source to the Intelligence Server. Of relevance, these settings can help improve memory usage or overall data load performance:
  - **Parallel Data Fetch** (Data > Configure Intelligent Cube... > Data Partition): Allows fetching slices of the final result table in parallel through different database connections into separate in-memory partitions.
Data Query and Load

OLAP Cubes – Parallel Data Fetch

Without Parallel Data Fetch

```
select a11.QUARTER_ID, QUARTER_ID_2, 
sum(a11.TOT_COST) WJXBFS1 
from QTR_CATEGORY_SLS a11 
group by a11.QUARTER_ID
... 
```

With Parallel Data Fetch

```
select a11.QUARTER_ID ... 
where MOD(ABS(pa11.QUARTER_ID_2), 2) = 1 
select a11.QUARTER_ID ... 
Where MOD(ABS(pa11.QUARTER_ID_2), 2) = 0
... 
```
Data Query and Load

MTDI Cubes

- All tables are loaded into memory in parallel.
- The following settings control the degree of parallelism for data loading (how many tables to load at the same time):
  - **Parallel Query Execution** (VLDB setting under Query Optimizations): See KB44052: How does the feature in the MicroStrategy 9.3.x SQL engine: "Parallel Query Execution" work.
- There are no joins or aggregations when selecting data from these relational sources (unless table is a SQL query or a Query Builder table).

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.

*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 in upcoming releases.*
Data Modeling
Data Query and Load

MTDI Cubes: Modeling Differences vs Project Schema

- OLAP: Intelligent Cubes are based on existent application logic
  - OLAP: Parallel Data Fetch with multiple connections
  - OLAP: Final Result Table

- MTDI: Application logic built on top of cubes in dashboards (with derived attributes, derived metrics)
  - MTDI: Parallel Query execution
  - MTDI: Database
    - Database: Raw table data is loaded to memory
    - Database: DB operations: filtering, projection

- Intelligent Cubes: Application logic built within (through simple attributes, metrics) of MTDI cubes
  - Intelligent Cubes: MTDI Cube

- Connection 1, Connection 2, Connection 3
# Data Query and Load

## MTDI Cubes: Modeling Differences vs Project Schema

<table>
<thead>
<tr>
<th>Logical Representation of Data Source objects</th>
<th>MTDI Schema</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tables</td>
<td>Tables</td>
<td></td>
</tr>
<tr>
<td>Fact columns</td>
<td>Metrics</td>
<td>Metrics directly represent aggregable columns.</td>
</tr>
<tr>
<td>Dimensional columns</td>
<td>Attributes &amp; Attribute forms</td>
<td></td>
</tr>
</tbody>
</table>

### Attributes Modeling

<table>
<thead>
<tr>
<th>Attribute Form Expressions</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute Lookup Table</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Multiple Forms</td>
<td>Yes</td>
<td>Restricted</td>
</tr>
</tbody>
</table>

### Other Schema Functionality Support across “Types of Schemas”

<table>
<thead>
<tr>
<th>Fact Expressions</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition Mappings</td>
<td>Transparently supported; SQL Engine, brings data from multiple tables result table that is loaded into OLAP cube</td>
<td>Multiple Tables with same columns can be loaded in parallel into a single in-memory table</td>
</tr>
<tr>
<td>Transformations</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Logical Tables</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Table Aliases</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

### Partitioning Configuration

<table>
<thead>
<tr>
<th>Data Partitioning</th>
<th>Yes</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto-partitioning</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

In MTDI, any column-level transformation happens through Data Wrangling, or in Visual Insight with derived attributes; alternatively, use a freeform SQL table, a Query Builder table in MTDI.

For MTDI, Attributes do not have an explicitly defined lookup table; the lookup table is auto-generated.

MTDI only allows defining a few forms for an attribute: ID, DESC, DATE, SORT, Latitude, Longitude. It also supports which are display forms by default.

In MTDI, any column-level transformation happens through a freeform SQL table, a Query Builder table, Data Wrangling, or in Visual Insight with derived attributes.

In MTDI multiple horizontally partitioned tables can be loaded into a single in-memory table.

A table in MTDI can be a freeform SQL pass, for example.

Useful for attribute roles; in MTDI the same table can be imported multiple times to achieve a similar effect, but with potential performance issues.

Allow loading more than 2 billion rows of data in memory; leverage multi-core architectures to query/calculate data in parallel.

Automatically determine whether partitioning should be enabled (largest cube table is larger then 1 million), and if enabled, determine partition attribute.
Relation and Look-up tables are generated automatically to improve query performance.

- **Lookup tables** contain Attribute Element form data and indices
- **Relation tables** contain relation between two attribute indices
- **Fact tables** (user-uploaded tables) contain metric data related to attribute indices.
Data Processing and Storage in Memory

Lookup and Relation Tables
Only tables containing partitioned attribute (including lookup and relation tables) are partitioned in memory.

For MTDI and OLAP cubes, each individual partition or table is processed in parallel within each publication stage.
Data Processing and Storage in Memory

About Partitioning – How to enable for MTDI?
Data Processing and Storage in Memory

Cube Publication Stages and General Performance Profile

- **Create Lookup tables for attribute data:**
  - identify set of unique data elements per attributes,
  - assign numeric indices for these data elements for performance purposes, and
  - attach lookup forms (description, etc.); these indices are used in metric indices later.
  - Lookup table creation is done in parallel, for every attribute or **attribute partition**.

- **Create Relationship tables:** relationship tables relate attributes elements to improve performance when joining data.
  - This only applies to MTDI cubes.
  - Relationship table generation is relatively fast (just indices of multiple attributes based on data present in one or more tables).
  - Relationship table building is done in parallel by relationship table.

- **Build fact tables:** Fact represent the actual tables the users upload; these tables only contain attribute indices along with references to the metric data.
  - fact table index,
  - populating metric columns and generating of additional indices that help improve element filtering.
  - Fact table generation is also done in parallel, by table or **table partition**.

- **Persist the Cube:** This step creates the corresponding cube files to allow loading the cubes without the need to republish.

To visualize detailed logs on the full publication process, enable the Engine > Perf log in the MicroStrategy Diagnostics tool.
Data Processing and Storage in Memory

Data Storage Optimizations

<table>
<thead>
<tr>
<th>Stage</th>
<th>Factors Affecting Performance</th>
<th>Main task in stage</th>
</tr>
</thead>
</table>
| Query Execution and Data Fetch | • Avoid loading a lot of redundant data for large datasets *(normalize data)*; repeated data may be loaded in memory for some time until attribute lookup tables are finished.  
  • Use the smallest data type possible in the schema configuration for every column to minimize memory usage in memory in this stage. | Raw data is loaded in memory in this stage.                                        |
| Lookup Table Generation      | • String or dates as Data type of ID attributes are slower to process.  
  • Very high cardinality of individual attribute elements in combination with string/date data types can slow down the lookup generation phase. | Identifying unique elements for columns and sorting them. Sorting string           |
| Relationship Table Generation| • The number of the automatically detected or user defined relationships tables.                                                                                                                                               | Identifying unique tuples of two related attributes.                               |
| Fact Table Generation        | • Larger fact tables will take longer to index and populate  
  • Larger number of metrics in a single fact table can delay the full fact table generation stage.  
  • Refresh operations on single fact table will cause full table metric population; if only small set of metrics change it may make sense to partition into smaller vertical sections of the table. | Main index generation and metric population is done for every loaded table.        |
Data Processing and Storage in Memory

Data Storage Optimizations

• **Metric value compression**: This was introduced since 9.x, and this avoids keeping in memory duplicate metric values. This applies to metrics of type double, strings (basically, all types that are larger than integer). The current rule in the code is if all rows for a metric have the same value, or if the row count is greater than the distinct number of rows times 10, then we do not repeat metric values for all rows; we save it once in memory.

<table>
<thead>
<tr>
<th>Attribute Index</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&lt;refer to val 1&gt;</td>
</tr>
<tr>
<td>1</td>
<td>&lt;refer to val 1&gt;</td>
</tr>
<tr>
<td>2</td>
<td>&lt;refer to val 2&gt;</td>
</tr>
<tr>
<td>…</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.383</td>
</tr>
<tr>
<td>1827.8898</td>
</tr>
<tr>
<td>12.3829</td>
</tr>
</tbody>
</table>

• **Index compression**: Besides the fact that data is referred to by indices, the space used to hold indexes in memory is also optimized by using smaller data types to hold the index values, depending on the number of elements, for example, for an attribute. So, instead of using integers (4 bytes) to represent an index value, we use a short (2 bytes), which reduces the index space by half, to say an example.
Data Processing and Storage in Memory

Data Storage Optimizations

- **Metric data type compression**: This optimization changes the data types of columns internally, as in, if we can hold all the values in a column in a smaller data type, we use a smaller data type to hold data in cubes in-memory (e.g. All decimal/big decimal numeric values can be represented with an integer, then we just use integer, to say an example).

  ![Diagram showing metric data type compression from Big Decimal to int](image)

- **Normalization of data**: Data is normalized in memory so that redundant data is not saved (such as attribute descriptions, etc.) in memory; and we use indexes to relate lookup data and fact data, not relying necessarily on original attribute primary keys (which may not be as efficient to store or relate data among tables, e.g. strings).

  ![Diagram showing normalization of data](image)
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

• **Product Manuals**:
Thank you

Questions?