Relational Sources
How to Deploy Enterprise Analytics With Relational Sources

Jochen Demuth, Senior Director Partner Engineering
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.
Dynamic Sourcing

MicroStrategy has the ability to allow queries to seamlessly and dynamically drill across multiple sources, and the server is able to auto recognize situations and intelligently direct queries against in-memory cubes when possible.

Generate Multi-pass SQL

MicroStrategy is able to easily and quickly generate multi-pass SQL queries to provide greater analytical power and minimize the amount of data that is pulled back to the mid-tier. The sophisticated SQL engine can deliver high performance for the most complex SQL computations.

Pushdown architecture

MicroStrategy leverages the database to its fullest extent by pushing data joins and analytic calculations to the database when possible. Every connector is optimized for high performance by pushing down functions to leverage the power of the database in conjunction with the Server.
MicroStrategy Pushes Analytical Workloads to Relational Data

Analytical queries have specific technical characteristics

• A typical analytical query
  • Accesses a large amount of data (up to terabytes)
  • Processes large amount of data

• MicroStrategy customers like to have interactive experience
  • Challenge is to achieve fast response times

• MicroStrategy works with partners to tackle this challenge
  • MicroStrategy formulates “good queries”
  • Relational database executes queries well
MicroStrategy Data Access Workflows
There are numerous ways for MicroStrategy to interact with relational data

- **Adhoc Schema**
  - For Analysts familiar with data in database
  - Schema is created automatically on the fly
  - Optimal time-to-value

- **Project Schema**
  - BI Architect creates logical model of data in MicroStrategy
  - Analyst or Consumers use model objects (attributes and metrics) to express their analytical needs
  - MicroStrategy generates multi-pass SQL specific to a database

- **Live Connect**
  - User actions result in interactive queries against data source
  - Good for frequently changing data

- **In-Memory Dataset**
  - Dataset is imported from database into Multi-dimensional In-Memory
  - Can improve performance and user scale accessing less frequently updated data sets
Intuitive Data Access Quickly Produces Visual Analysis of Relational Data
DB2 Is A Large Family of Relational Database Engines

IBM® DB2® is a next generation data platform for transactional and analytical operations

- This session talks specifically about IBM DB2 (for Linux, UNIX and Windows)
- Also applies to the latest cloud based offerings, such as dashDB
- IBM DB2 is a commonly used by MicroStrategy customers
- MicroStrategy and DB2 team have been working collaboratively on the technical integration for 20 years
MicroStrategy Provides Unique Optimizations for IBM DB2

• **DB2-optimized SQL syntax**
  - DB2 Analytical functions (OLAP functions)
  - CASE expressions
  - Full outer joins
  - Set operators
  - Subqueries
  - Setting Isolation levels

• **Multi-pass SQL for analytical sophistication**
  - Use of common table expressions (CTE)
  - Use of declared global temporary tables (DGTT)
  - Control over distribution keys for intermediate results
  - MicroStrategy Parallel SQL Execution

• **Seamless support for key DB2 features**
  - DB2 BLU Acceleration
  - DB2 Intrapartition Parallelism
  - DB2 Compression
  - DB2 Workload Manager
  - DB2 function library

• **Extensions to DB2 functionality**
  - Aggregate awareness with physical summary tables
  - Middle-tier computation of calculations not available in IBM DB2
  - Middle-tier caching via Intelligent Cubes
  - Report caching
  - Application-level partitioning
MicroStrategy Switches Intermediate Table Syntax Dynamically

Depending on report design a different approach provides optimal query performance

• In general, Common Table Expression (CTE) syntax provides the best execution plan

• In case of too many passes or when joining too many tables, Declared Global Temp Tables (DGTTs) are the better option

• Two settings control whether to generate CTE or DGTT syntax

<table>
<thead>
<tr>
<th>VLDB Category</th>
<th>VLDB Property Setting</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tables</td>
<td>Maximum SQL Passes Before Fallback</td>
<td>0</td>
</tr>
<tr>
<td>Tables</td>
<td>Maximum Tables in From Clause Before Fallback</td>
<td>0</td>
</tr>
</tbody>
</table>
Common Table Expression (CTE)

```sql
WITH ps1 AS
    (SELECT a12.SUBCAT_ID, a13.YEAR_ID, YEAR_ID, 
     SUM(a11.TOT_UNIT_SALES) WNXBFS1 
     FROM ITEM MTH_SLS a11 
     JOIN LU ITEM a12 
     ON (a11.ITEM_ID = a12.ITEM_ID) 
     JOIN LU.MONTH a13 
     ON (a11.MONTH_ID = a13.MONTH_ID) 
     GROUP BY a12.SUBCAT_ID, a13.YEAR_ID)

|, ps2 AS
  (SELECT a12.SUBCAT_ID, a13.YEAR_ID, YEAR_ID, 
  SUM(a11.UNITS_RECEIVED) WNXBFS1 
  FROM INVENTORY_ORDERS a11 
  JOIN LU ITEM a12 
  ON (a11.ITEM_ID = a12.ITEM_ID) 
  JOIN LU_MONTH a13 
  ON (a11.MONTH_ID = a13.MONTH_ID) 
  GROUP BY a12.SUBCAT_ID, a13.YEAR_ID)

| SELECT ps1.SUBCAT_ID, ps1.SUBCAT_DESC SUBCAT_DESC, 
  ps1.YEAR_ID, YEAR_ID, 
  ps1.WNXBFS1 WNXBFS1, 
  ps2.WNXBFS1 WNXBFS2 
  FROM ps1 
  JOIN ps2 ON (ps1.SUBCAT_ID = ps2.SUBCAT_ID AND 
  ps1.YEAR_ID = ps2.YEAR_ID) 
  JOIN LU_SUBCATS a11 
  ON (ps1.SUBCAT_ID = a11.SUBCAT_ID)
```

Declared Global Temp Tables (DGTTs)

```sql
DECLARE GLOBAL TEMPORARY TABLE SESSION.ZESP001
    SUBCAT_ID SMALLINT, 
    YEAR_ID SMALLINT, 
    WNXBFS1 DOUBLE;

PARTITIONING KEY (SUBCAT_ID, YEAR_ID) ON COMMIT PRESERVE ROWS NOT LOGGED;

INSERT INTO SESSION.ZESP001
    SELECT a11.SUBCAT_ID, SUBCAT_ID, a13.YEAR_ID, YEAR_ID, 
    SUM(a11.TOT_UNIT_SALES) WNXBFS1 
    FROM ITEM MTH_SLS a11 
    JOIN LU ITEM a12 
    ON (a11.ITEM_ID = a12.ITEM_ID) 
    JOIN LU_MONTH a13 
    ON (a11.MONTH_ID = a13.MONTH_ID) 
    GROUP BY a12.SUBCAT_ID, a13.YEAR_ID;

DECLARE GLOBAL TEMPORARY TABLE SESSION.ZESP001
    SUBCAT_ID SMALLINT, 
    YEAR_ID SMALLINT, 
    WNXBFS1 DOUBLE;

PARTITIONING KEY (SUBCAT_ID, YEAR_ID) ON COMMIT PRESERVE ROWS NOT LOGGED;

INSERT INTO SESSION.ZESP001
    SELECT a11.SUBCAT_ID, SUBCAT_ID, a13.YEAR_ID, YEAR_ID, 
    SUM(a11.UNITS_RECEIVED) WNXBFS1 
    FROM INVENTORY_ORDERS a11 
    JOIN LU_ITEM a12 
    ON (a11.ITEM_ID = a12.ITEM_ID) 
    JOIN LU_MONTH a13 
    ON (a11.MONTH_ID = a13.MONTH_ID) 
    GROUP BY a12.SUBCAT_ID, a13.YEAR_ID;

SELECT ps1.SUBCAT_ID, ps1.SUBCAT_DESC SUBCAT_DESC, 
  ps1.YEAR_ID, YEAR_ID, 
  ps1.WNXBFS1 WNXBFS1, 
  ps2.WNXBFS1 WNXBFS2 
  FROM ps1 
  JOIN ps2 ON (ps1.SUBCAT_ID = ps2.SUBCAT_ID AND 
  ps1.YEAR_ID = ps2.YEAR_ID) 
  JOIN LU_SUBCATS a11 
  ON (ps1.SUBCAT_ID = a11.SUBCAT_ID)
DROP TABLE SESSION.ZESP001
DROP TABLE SESSION.ZESP001
```
MicroStrategy Can Integrate with IBM DB2 WLM

IBM DB2 Workload Manager (WLM) enables administrators to easily monitor and control active work in the system.

- In a mixed workload environment (transactional and analytics) you need a way to prioritize queries. DB2 WLM provides this functionality.
- With DB2 with BLU acceleration or dashDB, there is built-in WLM.
- Use **WLM service classes** to set different priorities for each of your MicroStrategy workloads:
  - MicroStrategy Metadata Requests
  - MicroStrategy Element Requests
  - MicroStrategy Report Requests
- A report pre statement tags the MicroStrategy job

### VLDB Category | VLDB Property Setting | Value
--- | --- | ---
Pre/Post Statements | Report Pre Statement | CALL SYSPROC.WLM_SET_CLIENT_INFO('MSTRUser=!u', NULL, 'Project=!p', 'Report=!o', NULL)
Managing MicroStrategy Workloads in IBM DB2
Create different WLM service classes for your various MicroStrategy workloads

- **MSTR Report Requests**
  - General Reports Workload
  - Priority User Reports Workload

- **MSTR Element Requests**
  - Element Workload

- **MSTR Metadata Requests**
  - Metadata Workload

Service Class A: Reports
  - Default Subclass: Low Priority
  - Subclass A2: Medium Priority
  - Subclass A1: High Priority

Service Class B: Element Browsing

Service Class C: Metadata

*Please refer to the “Workload Management with MicroStrategy Software and IBM DB2” paper for more implementation details.
Microsoft SQL Server
Technical Integration with Microsoft SQL Server

Microsoft SQL Server is a data platform for analytical and transactional operations

Microsoft offers a rich family of relational database products in the SQL Server family

This session discusses the integration with
• Microsoft SQL Server
• Microsoft Azure SQL Database

MicroStrategy offers a different set of optimizations for the following products
• Microsoft Azure SQL Data Warehouse
• Microsoft SQL Server Analysis Services
• Microsoft Analytics Platform System
• (Microsoft SQL Server Parallel Data Warehouse)
MicroStrategy Is Most Commonly Used To Send Analytical Queries to Microsoft SQL Server

Analytical queries have specific technical characteristics that differentiate them from operational queries.

Analytical queries involve processing of massive amounts of data:
- Accessing large data volume
- Processing massive data volume

Challenge: Achieve interactive response times

Microsoft SQL Server offers some key features to help with that:
- Compression
- Partitioning
- In-memory columnstore
- In-memory OLTP, etc.
Many Integration Points Tackle Common Challenges

- SQL Server specific SQL Syntax
  - Ordered Analytic (OLAP) functions
  - CASE expressions
  - Full outer joins
  - Set operators
  - Sub queries
- Multi-pass SQL for analytical sophistication
  - Use of temporary tables or derived tables
  - Indexing
  - MicroStrategy Parallel SQL Execution
- Support for key SQL Server features
  - Parallel Query Execution
  - Indexed Views
  - Clustered Columnstore Indexes
  - Compression
  - Partitioning
  - Data level security (Pass-through auth)
  - SQL Server Linked Servers
  - Integration with Table-Valued User-Defined functions
  - Querying XML data
- Extensions to SQL Server functionality
  - Aggregate awareness with physical summary tables
  - Middle-tier computation of calculations not available in SQL Server
  - Middle-tier caching via Intelligent Cubes
  - Report caching
Multi-Pass SQL Helps Answer Complex Analytical Questions
Choice of Intermediate tables have a significant impact on Query Performance

MicroStrategy offers multiple approaches for Intermediate tables

Default: Global or True temporary table

A simple configuration setting allows switching:

VLDB: Intermediate Table Type

Intermediate result sets are created, populated and dropped by MicroStrategy.

Global temporary tables, indicated with the “##” prefix, are created in the tempdb and last for only one session
Indexes On Large Temp Tables Improve Performance

Report level setting creates secondary index on Intermediate Tables

```
select a11.CUSTOMER_ID CUSTOMER_ID
into
##ZZMQQ00
from CUSTOMER_SLS a11
where (a11.TOT_UNIT_SALES > 10.0)

create index ##ZZMQQ00_i on ##ZZMQQ00 (CUSTOMER_ID)
```

```
select a14.CATEGORY_ID CATEGORY_ID,
    max(a15.CATEGORY_DESC) CATEGORY_DESC,
    sum((a11.QTY_SOLD * (a11.UNIT_PRICE - a11.DISCOUNT))) WJXBFS1
from ORDER_DETAIL a11
    join LU_ORDER a12
    on (a11.ORDER_ID = a12.ORDER_ID)
    join session.ZZMQQ00 p1
    on (a12.CUSTOMER_ID = p1.CUSTOMER_ID)
    join LU_ITEM a13
```

In-memory columnstore Indexes

CREATE CLUSTERED COLUMNSTORE INDEX cci_?? on ??

‘???’ is a MicroStrategy wildcard that will be replaced with the name of the MicroStrategy data mart table
Executing multiple passes in parallel in MicroStrategy
Get the most out of the hardware you already have!

• Standard Edition only uses single core to process any given query – analytical queries are often CPU bound
• MicroStrategy’s “Parallel Query Execution” identifies the passes in a multi-pass SQL job that don’t depend on each other and can be safely submitted in parallel.
• By doing so the hardware resources on a Standard Edition SQL Server can be better utilized.

Required VLDB Settings to enable this:
• Query Optimizations -> Parallel Query Execution -> Enable
• Parallel Query Execution Improvement Estimate in SQL View -> Enable
• Maximum Queries per Report -> 2 (default)
## MicroStrategy and Teradata
### Partnership Strength and Value

<table>
<thead>
<tr>
<th>MicroStrategy</th>
<th>Teradata</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Industry leading BI platform</td>
<td>• Enterprise data integration</td>
</tr>
<tr>
<td>• Relationship since 1995 in enterprise Business Intelligence</td>
<td>• MicroStrategy BI performance and scalability</td>
</tr>
<tr>
<td>• BI Applications run natively on Teradata</td>
<td>• Largest number of users</td>
</tr>
<tr>
<td>• Optimized SQL for Teradata</td>
<td>• Highest level of BI complexity</td>
</tr>
<tr>
<td>• Teradata indexing, and user-defined functions</td>
<td>• Pre-defined and ad hoc query support</td>
</tr>
<tr>
<td>• High-Availability</td>
<td>• OLAP extensions</td>
</tr>
<tr>
<td>• Extended server-based computations</td>
<td>• Teradata uses MicroStrategy SQL for Optimizer testing</td>
</tr>
<tr>
<td></td>
<td>• Dedicated Engineering Resources</td>
</tr>
</tbody>
</table>

### Annual Strategy Session
- Optimized SQL for Teradata
- Extensive leverage of Teradata extensions
- High Availability Solutions
- Consistent participant in Teradata Early Adopter program
- Over 350 Joint Customers
Many Integration Points Tackle Common Challenges

• Multi-pass SQL for analytical sophistication
  • Ability to answer complex business questions inside Teradata
  • Use of volatile tables or derived tables
  • Control of primary indexes and statistics collection on intermediate results
• Integration with Teradata tools
  • Integrates with Teradata's core EDW mixed Workload Management features
  • Unity
  • TPTAPI/Export
• Extensions to Teradata functionality
  • Vast number of features that complement Teradata's architecture
  • Aggregate awareness with physical summary tables
  • Middle-tier computation of calculations not available in Teradata
  • Middle-tier caching via Intelligent Cubes
  • Report caching

• Teradata-specific SQL syntax
  • Takes advantage of Teradata's Massive Parallel Processing architecture and rich analytics
  • Ordered Analytic (OLAP) functions
  • CASE expressions
  • Full outer joins
  • Set operators
  • Sub queries
• Seamless support for key Teradata features
  • Couples with underlying Teradata optimizations for best superior query performance
  • Partitioned primary indexes
  • Aggregate join indexes
  • Teradata function library and UDFs
  • UNICODE character set
  • Columnar support
Derived Tables vs. Volatile Tables
By default MicroStrategy switches from Derived Table Syntax to using Volatile tables for reports with more than 64 passes

create volatile table ZZSP00, no fallback, no log(
YEAR_ID SMALLINT,
SUBCAT_ID BYTEINT,
WJXBFS1 FLOAT)
primary index (YEAR_ID, SUBCAT_ID) on commit preserve rows

;insert into ZZSP00
select a13.YEAR_ID YEAR_ID,
a12.SUBCAT_ID SUBCAT_ID,
sum(a11.TOT_UNIT_SALES) WJXBFS1
from ITEM_MNTH_SLS a11
join LU_ITEM a12
on (a11.ITEM_ID = a12.ITEM_ID)
join LU_MONTH a13
on (a11.MONTH_ID = a13.MONTH_ID)
group by a13.YEAR_ID,
a12.SUBCAT_ID
...
select a11.SUBCAT_ID SUBCAT_ID,
a11.SUBCAT_DESC SUBCAT_DESC,
a11.YEAR_ID YEAR_ID,
a11.WJXBFS1 WJXBFS1,
a12.WJXBFS1 WJXBFS2
from ZZSP00 a11
join ZZSP01 a12
on (a11.SUBCAT_ID = a12.SUBCAT_ID and
a11.YEAR_ID = a12.YEAR_ID)
join LU_SUBCATEG a11
on (a11.SUBCAT_ID = a11.SUBCAT_ID)
Many Teradata features Are Transperently Used

Here is but a short selection of the most commonly implemented ones

Data Distribution
- Primary Indexes are very crucial
- Physical profile of tables relates directly to response time for MicroStrategy reports

Partitioned Primary Index (PPI)
- Minimizes physical access targeting only the rows of qualifying partitions. Queries run faster.
- Helpful for queries based on range access, such as date ranges

No primary index (NoPI)
- Useful for applications that concurrently load data into a staging table
- MicroStrategy can use NoPI for intermediate table creation

Aggregate Join Index (AJI)
- Creation, maintenance, and automatic navigation of pre-aggregations and pre-joined tables
Teradata Manages Workload Using Query Bands

Query Bands assign resources to incoming queries

• Teradata allows applications to “tag” each report / SQL statement with identifying information

• MicroStrategy makes use of Query Bands

• Combined execution logs from MicroStrategy (Enterprise Manager) and Teradata (DBQL) enable deep usage analysis

---

```sql
SET QUERY_BAND
='ApplicationName=MicroStrategy;Version=9.0.1;ClientUser=!u;Source=!p;Action=!o;
StartTime=!dT!t; JobID=!j;Importance=!i;sess_id=!s;proj_id=!z;report_guid=!r;' FOR SESSION;

create volatile table ZZSP00, no fallback, no log(
  YEAR_ID INTEGER,
  SUBCAT_ID INTEGER,
  WJXBFS1 FLOAT)
primary index (YEAR_ID, SUBCAT_ID) on commit preserve rows
;insert into ZZSP00
select a13.YEAR_ID YEAR_ID, a12.SUBCAT_ID SUBCAT_ID,
... SET QUERY_BAND NONE FOR SESSION;
```
MicroStrategy 10 Offers Two Connectivity Options
Performance Considerations

• ODBC for Push-down Reports
  • Proven reliable industry standard
  • JDBC on Mac

• TPTAPI (Teradata Parallel Transporter API) for In-Memory cubes load
  • Enables effective data transfer to MicroStrategy
  • Due to API overhead this is only recommended for data volumes larger than 1GB
Parallel sessions out of Teradata into MicroStrategy Cubes

- Alternative means to load/unload data between a Teradata Database Server and Client application.
- MicroStrategy 10.1 invokes the Export Operator from TPTAPI and export data quickly out of Teradata into MicroStrategy Cubes.
- The “FastExport” protocol is capable of exporting data out of Teradata utilizing parallel sessions and therefore has a higher throughput rate than a single session traditional ODBC.
- Multiple processes launched to read data in parallel.
- TPTAPI further optimizes throughput by enabling multiple “instances”.
- For Setup/supported configuration, check out KB266840 on MicroStrategy Community website: FAQ on using Teradata Parallel Transporter API (TPTAPI)
Two steps required to enable use of TPTAPI Export

1. Enable use of TPTAPI for Teradata connection
2. Enable use of TPTAPI on Report level (typically a cube report)

- If TPTAPI is enabled for a multi-pass SQL report, MicroStrategy only retrieves the final result set via TPTAPI
- SQL View Allows Verification of TPT Use
Thank you