TABLEAU CONFERENCE
Best practices for Tableau on SAP HANA

Jason Burns
Product Manager
Tableau

Tom Slee
Product Manager
SAP HANA

Track: Data
Session: 18BI-068 (breakout session)
Location: MCCNO – L2 – 286
Time: Wednesday 24 October, 3:30pm—4:30pm
Best practices for Tableau on SAP HANA

Jason Burns
Product Manager
Tableau

Tom Slee
Product Manager
SAP HANA

Track: Data
Session: 18BI-068 (breakout session)
Location: MCCNO – L2 – 286
Time: Wednesday 24 October, 3:30pm—4:30pm
Best practice #1: Stay current with SAP HANA and Tableau software
Agenda

Orientation

- Use cases

Problem areas

- Custom SQL

- Query routing

- Calculation view semantics

- Views

- Query processing internals
Tableau / SAP HANA use cases
SAP HANA: The only all-in-one, in-memory-first data platform

Accelerate with simplicity

Benefits

- High performance and reliable operations for traditional and new applications
- Simplified application architectures and IT landscapes with minimal data movements
- Faster time to value by designing and building virtually for immediate business results
- Lower costs with effective management of large data sets

SAP HANA
The business data platform for the intelligent enterprise

Real-time analytics on live transactions without data duplication

Built-in advanced analytics and multi-model processing

In-memory-first dynamic data storage management

Virtually or physical connect to all data from any source

Modern application development

Enterprise-ready security and reliability 24/7

© 2018 SAP SE or an SAP affiliate company. All rights reserved. | PUBLIC
Tableau / SAP HANA scenarios I
Operational reporting and real-time analytics

- Operational reporting and real-time analytics on data from Business Suite or S/4HANA

- **Business Suite on HANA** provides SAP HANA Live Calculation Views, accessible through ODBC

- **S/4HANA** provides S/4HANA Embedded Analytics CDS views, accessible through OData

Tableau accesses Business Suite operational data from HANA

Accesses S/4HANA operational data through OData to the app server
Tableau / SAP HANA scenarios II
Strategic and tactical reporting

• BW on HANA and BW/4HANA provide access through BW and through HANA Calculation Views

• HANA access is particularly useful for those who build custom views on BW views
Tableau / SAP HANA scenarios III
HANA data mart scenarios

Source Systems

ERP

CRM

HR

ETC..

SAP HANA

IT-driven reporting

Tableau

Business unit and end-user driven reporting

SAP HANA

SAP HANA

SAP HANA

SAP HANA

SAP HANA

Source Systems = 2TB
Exposing data from SAP HANA
Flavors of HANA views

• SQL views
  • CREATE VIEW
  • Define in HANA as design-time artifacts
  • Created by ABAP CDS views

• HANA CDS views
  • Define in HANA using Core Data Services
  • ORM-like constructs
  • Generates SQL views

• HANA 1.0 Calculation Views
  • Define as design-time artifacts in the HANA repository using HANA Studio
  • Query from _SYS_BIC
  • Metadata in _SYS_BI
  • Generated by BW
  • Supported in HANA 2.0

• HANA 2.0 Calculation Views
  • Define as design-time artifacts using Web IDE and deploy to HDI container
  • Metadata in _SYS_BI
  • Query from HDI container
HANA Calculation Views
Building views in layers

SAP HANA

- Base tables
- Global views
- LOB-specific views

Central / IT
- Business Units

Tableau

HANA Data Mart
HANA Calculation Views
Building views in layers

BW/4HANA
Tableau

SAP HANA views

LOB-specific views
Business Units

Suite on HANA
Tableau

SAP HANA

Customer views
Business Units

Re-use views
Private views
Base tables

HANA Live
Suite

SAP Business Suite on HANA with HANA Live
Internals
HANA indexserver

Parse to build a parse tree

Optimize

• SQL Query transformation and cost-based optimization

• Calculation Engine instantiation

Execute

• Multiple “engines” operate semi-autonomously
SAP HANA performance

Compilation time
- Can be a significant part of the overall execution time
- A problem when prepared statements cannot be re-used (when bind parameters are not provided)
- Statement routing can cause queries to be compiled on more than one node.

Recent improvements
- Reduce compilation time for identified problematic cases
- Lightweight compilation to route requests quickly
SAP HANA performance

Plan cache

- Failing to use prepared statements with bind parameters leads to plan cache misses
- Plan cache limitations can lead to a high miss rate

Recent improvements

- Plan cache sharing among users was turned on by default starting from SAP HANA 2.0 SPS00.
- Plans can now be cached for users with analytical privileges
Custom SQL: performance
Identifying slow queries

At Tableau:
Use Tableau Performance Recording or analyze Tableau logs

At HANA:
Enable the Expensive Statements trace, and access it by querying the view M_EXPENSIVE_STATEMENTS

Connect the dots
Analyze slow performing/errors in your Tableau dashboards, by filtering on user or time, or statement...

For example:

```sql
SELECT TOP 10 DB_USER, START_TIME, DURATION_MICROSEC, STATEMENT_STRING
FROM M_EXPENSIVE_STATEMENTS
WHERE DB_USER = 'MYUSER'
AND STATEMENT_STRING LIKE '%SQL SNIPPET FROM TABLEAU%'
```
HANA SQL Queries on Calculation Views

- Join cardinality setting
- Implicit type casting
- JOIN predicates
- EXISTS/IN predicates

These tips may change over time

Improving custom SQL

At HANA:
Design Calculation Views with appropriate options

At Tableau:
Override if necessary in SQL query hints
Join cardinality in Calculation Views

Join cardinality
- can be set in calculation views.

Join pruning
- OUTER join
- No column requested from the dimension table
- Join cardinality is “… to one” for the dimension table

Benefits
- Faster query execution
- Less memory consumption

```
create view my_view as
(select *
from orders
left outer many to one join customers
on orders.cid = customers.id
left outer many to one join products
on orders.pid = products.id
);
```
Incorrect join cardinality may supply duplicated rows. See https://blogs.sap.com/2017/10/27/join-cardinality-setting-in-calculation-views/

Available public hints permit you to override cardinality settings in a query

```
SELECT ...
FROM ...
WITH HINT ( HINT(<value>) )
```

- **NO_JOIN_CARDINALITY**
  - EXPLICIT: discard all cardinality settings given by users
  - IMPLICIT: skip to add join cardinality by SQL Optimizer
  - ALL: EXPLICIT + IMPLICIT

- **CHECK_JOIN_CARDINALITY**: If any join violates its cardinality setting, the query returns an error and disables join cardinality-relevant rewriting rules

- **NO_JOIN_REMOVAL**: Remove all kinds of join removal rules

- **NO_JOIN_SIMPLIFICATION**: Disable join simplification
Avoid implicit type casting

- Explicit type casting to carry out the more efficient cast
  
  Problematic query
  
  ```sql
  SELECT *
  FROM T
  WHERE date_string < CURRENT_DATE;
  ```

  Workaround
  
  ```sql
  SELECT *
  FROM T
  WHERE date_string < TO_VARCHAR(CURRENT_DATE, 'YYYYMMDD');
  ```

- Add generated columns to avoid type casting entirely
  
  Problematic query
  
  ```sql
  SELECT *
  FROM T
  WHERE varchar_value = 1;
  ```

  Workaround
  
  ```sql
  ALTER TABLE T
  ADD (num_value DECIMAL GENERATED ALWAYS AS varchar_value);
  SELECT *
  FROM T
  WHERE num_value = 1;
  ```
JOIN predicates

- Avoid non-equijoin predicates in outer joins
- Avoid non-equijoin predicate in inner joins

Example:

- rewrite a non-equijoin predicate into an equijoin predicate

Problematic query:

```
SELECT M.year, M.month, SUM(T.ship_amount)
FROM T
LEFT OUTER JOIN M
ON T.ship_date BETWEEN M.first_date
AND M.last_date
GROUP BY M.year, M.month;
```

Workaround:

```
SELECT M.year, M.month, SUM(T.ship_amount)
FROM T
LEFT OUTER JOIN M
ON EXTRACT(YEAR FROM T.ship_date) = M.year
AND EXTRACT(MONTH FROM T.ship_date) = M.month
GROUP BY M.year, M.month;
```
JOIN predicates

- Avoid join conditions not supported natively in the column engine
  - Calculations with parameters in join predicates
  - Expressions which refers to both child tables
- Hints: [NO_]CS_EXPR_JOIN suggests to prefer or avoid column store calculations

Predicate with parameter

```
SELECT *
FROM T
JOIN M
ON T.ship_amount + ? = M.target_amount;
```

Calculation involving multiple tables

```
SELECT *
FROM T
JOIN M
ON IFNULL (T.ship_amount, M.pre_order_amount) = M.target_amount
```
JOIN predicates

Avoid cyclic JOIN predicates

- Outer join: cyclic joins not supported by column engine
- Inner join: can be handled by column engine but very inefficient in some cases

Example:

- move the supplier.nation column to the lineitem table.

Cyclic join

```sql
SELECT *
FROM supplier S,
     customer C,
     lineitem L
WHERE L.supp_key = S.key
AND L.cust_key = C.key
AND S.nation = C.nation;
```

Acyclic join

```sql
SELECT *
FROM supplier S,
     customer C,
     lineitem L
WHERE L.supp_key = S.key
AND L.cust_key = C.key
AND L.supp_nation = C.nation;
```
JOIN predicates

Avoid subquery filter predicates accessing multiple tables in outer joins

Example:

- maintain the priority column in the lineitem table instead of orders table.

<table>
<thead>
<tr>
<th>Problematic query</th>
<th>Workaround</th>
</tr>
</thead>
</table>
| SELECT *  
FROM customer C  
LEFT OUTER JOIN (  
SELECT * FROM orders O  
JOIN lineitem L  
ON O.order_key = L.order_key  
WHERE L.shipmode = 'AIR'  
OR O.priority = 'URGENT')  
ON C.cust_key = L.cust_key; | SELECT *  
FROM customer C  
LEFT OUTER JOIN (  
SELECT * FROM lineitem L  
WHERE L.shipmode = 'AIR'  
OR L.priority = 'URGENT')  
ON C.cust_key = L.cust_key; |
Avoid disjunctive EXISTS and IN predicates

- When an EXISTS or NOT EXISTS predicate is connected with other predicates through OR, it is internally mapped to left outer join.

<table>
<thead>
<tr>
<th>Problematic query</th>
<th>Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT * FROM T WHERE EXISTS (SELECT * FROM S WHERE S.a = T.a AND S.b = 1) OR EXISTS (SELECT * FROM S WHERE S.a = T.a AND S.b = 2);</td>
<td>SELECT * FROM T WHERE EXISTS (SELECT * FROM S WHERE S.a = T.a AND (S.b = 1 OR S.b = 2));</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problematic query</th>
<th>Workaround 1</th>
<th>Workaround 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT * FROM T WHERE a IN (SELECT a FROM S WHERE S.b = 1) OR EXISTS (SELECT a FROM S WHERE S.b = 2);</td>
<td>SELECT * FROM T WHERE a IN (SELECT a FROM S WHERE S.b = 1 OR S.b = 2);</td>
<td>SELECT * FROM T WHERE a IN ((SELECT a FROM S WHERE S.b = 1) UNION ALL (SELECT a FROM S WHERE S.b = 2));</td>
</tr>
</tbody>
</table>
Avoid inefficient EXISTS/IN predicates

- **[NOT] IN predicate**
  - For NOT IN, entire subquery is processed first before the processing the query as a whole, matching up based on the condition provided.
  - For NOT EXISTS, true or false is returned while checking for the provided condition.
  - Unless subquery results is very small, using NOT EXISTS is much faster than NOT IN. (Same applies for EXISTS/IN)
  - Side effects: The result from the child including the predicate is materialized before executing the join.

<table>
<thead>
<tr>
<th>NOT IN query</th>
<th>Possibly equivalent query in some cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT * FROM T WHERE a NOT IN (SELECT a FROM S);</td>
<td>SELECT * FROM T WHERE NOT EXISTS (SELECT * FROM S WHERE S.a = T.a);</td>
</tr>
</tbody>
</table>
Query routing
Every session has an ANCHOR node, to which the client is connected at the beginning of the session.

Set the global client distribution mode with an INI parameter:

- connection: ANCHOR connections are allocated in a round-robin fashion. In environments where queries are executed directly, this avoids overloading a single node.
- statement: by default prepared statements are routed based on table locations (distribution mode “statement”).
- all: enables connection and statement-level connection assignment

```
indexserver.ini
[distribution]
client_distribution_mode
= [ off | connection | statement | all ]
```
Query routing and execution: performance

- Each query is executed at a single node
- The execution node fetches data as needed from other nodes
- Fetching data over the network creates the possibility of performance problems
Query routing for faster execution: prepared statements

- **Execution options**
  - Direct: execute at the current node
  - Prepare & execute: preparation permits communication before execution

- Prepared statements can be routed to the best node
• Every transaction has a PRIMARY node. Requests will be executed at this node unless a prepared statement is routed to another node.

• In HANA 2.0 SPS02 the PRIMARY node for a transaction is configurable via a connection property PRIMARYCONNECTION.

• A heuristic improves the chance that the primary node is the best node on which to execute queries, and to avoid the expense of query re-routing.
  - ANCHOR (default): the primary is the anchor connection
  - LASTPRIMARY: the primary connection is the primary connection of the previous transaction
  - LASTEXECUTE: the primary connection is the connection on which a prepared statement was executed
HANA 1.0 Rev122.15: “Lightweight query preparation” avoids expensive duplicate query preparation in the case of routed queries.

- A query routed from one node to another is compiled twice: once at the primary node and once at the node to which it is routed. The end-to-end elapsed time experienced by the user is (2x preparation + 1x execution).
- New “lightweight query preparation” avoids a full compilation before deciding whether to route the query to another node.
- Enable by a server setting

```
indexserver.ini
[sql]
minimal_compilation_for_routing = true
```
Calculation views

- Originally implemented in the application server in SAP BW / BOBJ
- Closer to OLAP cubes than SQL views: include ideas of dimension and measure
- Pushed down to the database with HANA
- Rich features, including currency conversion, restricted columns, ranking, anonymization

- Access through internal interfaces, MDX or SQL
- Moving towards SQL standardization
- Still some mismatches between SQL and Calculation View semantics…
Calculation View semantics

- Often stacked many layers deep
- Data “flows” from bottom to top
- Calculation and aggregation happens at specific layers
- Order-dependent operations require care
  - AVG of AVG is usually meaningless
  - COUNT DISTINCT, VAR, and other non-additive operations are also order-dependent
Unlike SQL views, the level of aggregation depends on the columns referenced in the query.

Use explicit aggregation in your queries

```
SELECT COUNTRY, CITY, SUM(SALES)
FROM CV
GROUP BY COUNTRY, CITY
```

Be careful about level of aggregation…
Calculation view semantics

• Ideal solution
  • Put all the logic into either calculation views or SQL queries, and avoid mixing the two.

• When that is not possible…

• Non-additive functions
  • Do not mix aggregations across SQL / Calculation Views (eg SUM, MAX)
  • Do not aggregate previously-aggregated measures using non-additive functions (AVG, COUNT DISTINCT, COUNT(*), VAR etc)

• Watch for joins and other operations that may introduce unintended column references
  ```sql
  SELECT F1
  FROM CV1 JOIN CV2
  ON CV1.F2 = CV2.F2
  ```

• In your calculation view, use these options where appropriate
  • keep flag – always aggregate even if column is not referenced
  • transparent filter flag – do not aggregate even if the column is referenced
Calculation view semantics: push-down

• Evaluation of many SQL functions can be “pushed down” into the Calculation View. In this case there is no mixing of logic at runtime across the SQL / Calculation View domains.

• SQL functions that cannot be pushed down must be evaluated after the Calculation View, bringing complexity.

• Use EXPLAIN PLAN to identify operations that cannot be pushed down.
  • COLUMN SEARCH operations (SQL) should be above (after) AGGREGATION operations.
  • Multiple COLUMN SEARCH operations also indicates incomplete push-down, and may influence aggregation levels.

• Individual functions that can be pushed down are included in

```
CALL PUBLIC.GET_FUNCTION_DICTIONARY_FROM_RS_TO_CS_ENGINE();
```
Useful Documents


SAP HANA Troubleshooting and Performance Analysis Guide

SAP HANA Administration Guide

SAP HANA SQLScript Reference

- SAP Note 2223597 - Implicit SQL optimization of SAP HANA Calculation Views
- SAP Note 2000002 - FAQ: SAP HANA SQL Optimization
- SAP Note 2142945 - FAQ: SAP HANA Hints
- SAP Note 2441054 - High query compilation times and absence of plan cache entries for queries against calculation views
- SAP Note 1872746 - Model cardinality affects query results
- SAP Note 2618790 – Graphical View Modeling in SAP HANA - How to Avoid Unexpected Results
- SAP note: [2180165](#) – Expensive Statements trace
Thank you!

Tom Slee  
tom.slee@sap.com
What about Tableau?
Enable the Tableau Metadata Cache during SSO sessions
A tale of two caches

Tableau Server

SAP HANA

Query Cache

Metadata Cache

Non-SSO connection
A tale of two caches

SSO connection (Kerberos/SAML)

Tableau Server
- Query Cache
- Metadata Cache

SAP HANA
A tale of two caches

SSO connection

Tableau Server
Query Cache
Metadata Cache

SAP HANA
How to enable the shared metadata cache for SSO scenarios

```
tsm configuration set -k enable_metadatcache --force-keys -v shared
tsm pending-changes apply
```

- With the default metadata cache behavior, the sequence for the first user looks like some misses, followed by some hits, then some misses and more hits. The sequence for the second user is the same.
- With the cached disabled, the sequence is all misses for both users.
- With shared, the sequence for the first user is the same as for the default behavior, but the sequence for the second user is almost all hits

For detailed instructions on setting Tableau Server configuration options using TSM, see https://onlinehelp.tableau.com/current/server/en-us/cli_configuration-set_tsm.htm
Enable SAML Assertion Encryption via .TDC
The .TDC Change

For detailed instructions on using .TDC files with Tableau, see [https://kb.tableau.com/articles/howto/using-a-tdc-file-with-tableau-server](https://kb.tableau.com/articles/howto/using-a-tdc-file-with-tableau-server)

* SSL must be enabled on SAP HANA for this to work
Extracts & Calculations
The Scenario

- IT has set up a published Tableau connection to SAP HANA for users to consume.
- The views that support the published connection do not contain the calculations we need for analysis.
- We create a workbook based on that connection and model our calculations in our workbook.
The Problems

• Tableau Data Server doesn’t know about your calculations upon refresh & optimization.
• Any calculations that exist on the published data source are materialized in the extract, but the workbook calculations are not.
• The extract performance is diminished because Tableau must calculate these are runtime.
The Solutions

• Work with IT to ensure that your calculations are included in the published connection. (Not always easy)

• Build the workbook based on an embedded connection that is stored within the workbook. (Easier)
QnA
Appendix: list of improvements
SAP HANA performance enhancements
Listing of individual improvements
SAP HANA performance enhancements
Compilation time for routed requests

• Enhancements
  • HDB-4912 and incident 183210 (client) – Add a “re-prepare flag” for routed statements
  • HDBSESSION-417, HDBSQL-2034 & HDBSQL-2205 & Incident 105001 - support “re-prepare flag” in session layer and optimizer

• Requested by
  • CONA, NTT DoCoMo

• Status
  • Open, planned for SPS04

• Description
  • If a query is routed from the primary connection node to a different node, skip the light-weight compilation step and proceed with full compilation.

• Explanation
  • A feature called light-weight compilation was introduced recently. When light-weight compilation is in operation, query compilation proceeds as follows:
    1. Node 1 (first request): light-weight compilation to make a routing decision (not cached)
    2. Node 2 (routed): light-weight compilation (not cached)
    3. Node 2 : real compilation (cached)
  • The purpose of this enhancement is to skip Step 2, and so speed up query compilation on Node 2.
SAP HANA performance enhancements
Auto-parameterization

- **Enhancements**

- **Requested by**
  - Apple, CONA

- **Status**
  - Decommitted for SPS04

- **Explanation**
  - Instead of clients executing a separate “Prepare” request, parse the statement on the server, identify the parameters, and so avoid repeated preparation.
  - The best plan may be different for different parameter values. The optimizer must be extended to handle this problem.
SAP HANA performance enhancements
Statement compilation time

• Enhancements
  • Incident 148585, incident 155273 – reduce high compilation times with XML Analytical Privileges or SQL-based analytical privileges
  • Apple BCP incident: 344783 / 2017 high parsing and execution times with SQL-based analytical privileges.

• Status
  • Fixed in Rev122.12.

• Explanation
  • These changes have reduced compilation times for some cases from over 20 seconds (148585) or over 50 seconds (155273) for a query which stacks 466 views.
  • For results, see later slide.
SAP HANA performance enhancements
Statement compilation time

• Enhancements
  • Incident 148619: high compilation time at each execution, optimization in Calc engine. Fix in Rev122.12. Apple BCP issue (248657 / 2017)
  • Reduce compile time for:
    • HDBSQL-1676: many IN predicates
    • HDBSQL-1608: self-key join removal
    • HDBSQL-1685: large expressions.
    • HDB-3826: "unfolding calc scenario"

• Status
  • Implemented

• Explanation
  • For some complex queries, compilation time can be many seconds, and may be longer than execution time. The problem is especially important when plans cannot be reused. These changes have reduced query compilation times significantly for some major customers.