The Arts & Science of Tuning HANA models for Performance
Abani Pattanayak, SAP HANA CoE
Nov 12, 2015
Disclaimer

This presentation outlines our general product direction and should not be relied on in making a purchase decision. This presentation is not subject to your license agreement or any other agreement with SAP. SAP has no obligation to pursue any course of business outlined in this presentation or to develop or release any functionality mentioned in this presentation. This presentation and SAP's strategy and possible future developments are subject to change and may be changed by SAP at any time for any reason without notice. This document is provided without a warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. SAP assumes no responsibility for errors or omissions in this document, except if such damages were caused by SAP intentionally or grossly negligent.
# SAP HANA: The Platform for All Applications

## SAP HANA Platform

### Application Services
- Web Server
- JavaScript
- Fiori UX
- Graphic Modeler
- Application Lifecycle Management

### Processing Services
- Spatial
- Graph
- Predictive
- Search
- Text Analytics
- Streaming Analytics
- Series Data
- Business Functions

### Integration & Quality Services
- Data Virtualization
- ELT & Replication
- Data Quality
- Hadoop & Spark Integration
- Remote Data Sync

### Database Services
- Columnar OLTP+OLAP
- Multi-Core & Parallelization
- Advanced Compression
- Multi-tenancy
- Multi-Tier Storage
- Data Modeling
- Openness
- Admin & Security
- High Availability & Disaster Recovery
Overview
What’s Model Performance?

- SQL Query
- Query Run Time
- Memory Usage
- CPU Usage
- Concurrency
What’s under the hood i.e. Horse Power?

- Dynamic memory (at least 50%)
- Enough CPU (1+ core/8GB)
- Single node vs. Scale-out
- Single/Multi-Tenant (Shared)
- Cloud vs. In-premise
SAP HANA: Performance Tools

- Analysis of SQL query
- Visual walkthrough of model
- Explain plan (engine used)
- Visualize plan
- HDBAdmin
Analysis of SQL query

- SELECT EMPLOYEE FROM "_SYS_BIC"."abani/CV_SALES";

- SELECT EMPLOYEE, SUM(SALES) FROM
  (SELECT * FROM "_SYS_BIC"."abani/CV_SALES")
  GROUP BY EMPLOYEE;

- SELECT Table_1.EMPLOYEE, SUM(Table_1.SALES) FROM
  "_SYS_BIC"."abani/CV_SALES" as Table_1 INNER JOIN
  "_SYS_BIC"."abani/AT_DATE" as Table_2
  ON Table_1.DATE_SQL = Table_2.DATE_SQL
  WHERE Table_2.DATE = TO_CHAR('20150101','YYYY-MM-DD');
Analysis of SQL query

- \textbf{SELECT TOP 50 DISTINCT EMPLOYEE FROM} \\
  "\_SYS\_BIC\"."abani/CV\_SALES";

- \textbf{SELECT EMPLOYEE, SUM(SALES) FROM} \\
  \{SELECT * FROM "\_SYS\_BIC\"."abani/CV\_SALES"\} \\
  GROUP BY EMPLOYEE;

- \textbf{SELECT Table\_1.EMPLOYEE, SUM(Table\_1.SALES) FROM} \\
  "\_SYS\_BIC\"."abani/CV\_SALES\_FINAL" as Table\_1 \\
  WHERE Table\_1.DATE\_NEW = '20150101' \\
  GROUP BY Table\_1.EMPLOYEE;
Visual walkthrough of model

- View properties
- Review the joins
- Calculated columns
- Scripted views
- Cardinality of dataset
- …
Explain plan

- EXPLAIN PLAN FOR
  SELECT "MANAGER", SUM("AMOUNT_SOLD")
  FROM "_SYS_BIC"."abani.efashion/AN_SALES"
  WHERE MANAGER = 'Larry'
  GROUP BY "MANAGER";

- HANA Academy: Using Explain Plans: https://www.youtube.com/watch?v=YfkGutsz5Uo
Visualize plan

SCN: Visualize Plan & Timeline
- http://scn.sap.com/community/hana-in-memory/blog/2012/12/20/show-me-the-timelines-baby
How to use HDBAdmin to analyze performance traces in SAP HANA
http://scn.sap.com/docs/DOC-51110
Performance Tuning in 30mins

- Use left outer joins
- Specify cardinality in joins (n:1 or 1:1)
- Set optimize join = true (SP09)
- Use table functions instead of scripted calculation views (SP09).
- Execute in SQL-engine* (HANA live)
Deep Dive: the science
Performance Tuning: Basics

2bn scans /second /core & 16m aggregations /sec /core

- No of columns (& rows) scanned
- No of aggregations performed
- Type of calculation required
- No of tables participating in joins
- Size of temporary (internal) tables
- Amount of data transfer between engines
- Degree of parallelization
Performance Tuning: Basics

2bn scans /second /core & 16m aggregations /sec /core

- No of columns (& rows) scanned
  - ✓ Filter & cardinality of dataset in DB
- No of aggregations performed
  - ✓ Cardinality of selected columns
- Type of calculation required
  - ✓ Sum vs. distinct count
- No of tables participating in Joins
  - ✓ Type of joins & properties
- Size of temporary (internal) tables
  - ✓ Push joins to the lowest level
- Amount of data transfer between engines
  - ✓ Inefficient use of DB engines*
- Degree of parallelization
  - ✓ Logical and physical partitioning*
Filters

• Reduce the dataset as early as possible. Use design time filters at the lowest level.

• Input Parameter: Placeholders part of the models.
  ✓ Can be used in Calculation. Can accept multiple values (SP09)
  ✓ Can be derived from table. Can be derived from store procedure (SP09)

• Ensure Variables (where clause) is pushed to the lowest level. Confirm using Visualization Plan.

• Use analytical privilege and SQL analytical privilege (SP10) to filter data.

• Avoid script-based calculation view, WHERE clause will not be pushed down.

• Using Filter is better than using Inner Join to limit the dataset.

• Avoid filters on calculated column (consider materializing these columns).
Dealing with high cardinality

- Joins are the most expensive operation. Consider pushing most of the joins (at least on the tables with high cardinality) to the lowest level.
  - ✓ Dimension: attribute views and
  - ✓ Star-schema: analytic views or the lowest-node of the calculation view.

- Columns participating in the Joins are also selected (even if not requested) during query execution. Hence direct impact on the cardinality of the dataset.

- Use aggregation node after union nodes in calculation view.

- Be careful with Keep Flag in aggregation node.
Type of calculation

• “Distinct count” is a complex operation and very CPU intensive compared to a simple aggregation like “sum” or “count”. CPU usage will exponentially high esp. with high cardinality counters like “Transaction Count”. These counters must be computed in the Analytic View level (where possible).

• Avoid calculation before aggregations (where possible).

• Watch out for currency/unit conversion.

• Attributes used in restricted or calculated measure are also requested and may increase the cardinality of the dataset.

• Calculation with data type conversion are expensive. Avoid where possible.

• Watch out for calculations (like String_Args) which execute in the row-engine.
Join pruning

- Specify “Cardinality” for all Joins (1:1, n:1 etc.). Avoid m:n (Cartesian product)
- Consider using Left Outer or Referential Joins (vs. Inner Join)
- Referential Join to attribute view (with design time filter) will behave as “Inner Join”
- Specify “Optimize Join = True” for Left Outer and Text Joins (SP09 onwards)
- Avoid Right Outer Joins
- Temporal Join to attribute views (with design time filter) will always execute.
- Avoid Joins on “Calculated Columns” (consider materializing these columns)
Model pruning

- Consider using **Constant Mapping** in UNIONs for efficient model pruning.
Use of DB engines

• Execute in SQL Engine
  ✓ recommended for HANA live* models.

• Use table function vs. scripted calculation views
  ✓ migration option in studio (SP09)
Degree of parallelization

- Consider using UNION in place of JOINS in calculation view, esp. while dealing with similar dataset (plan & actual).
- Implementing a logical partitioning model, may reduce query runtime but will increase CPU usage.
- Avoid single datasource (or node) feeding multiple nodes in calculation view.
Reduce network data transfer

- To reduce the network data transfer (i.e. to push the joins to each node) in a scale-out deployment
  - Collocate the master and fact data tables in the same node
  - Replicate master data to all nodes (if the fact table is distributed across multiple nodes)
  - Tables in _SYS_BI schema can't be replicated to all nodes

- Consider applying multi-level partitioning with hash-partitioning (on single field) in the first level. This will ensure, calculations can be computed on each node and each partition.
Caching

- Introduced in SP08. Improved in SP09.
  - for complex* queries only
  - an option to improve concurrency
Deep Dive: the art
Modeling approach: follow the measures

- Consider building models with a star-schema/cube. The fact table (measures) is surrounded by various dimensions. The star-schema can be built using analytic views or star-join calculation views (comparable performance in SP09).

- Dimensions should be built as attribute views or dimension-calculation views.

- Only business logics (very minimal joins) on top of the star-schema.

OLAP Engine is the fastest engine in HANA.
Modeling approach: follow the measures
Building it right: the first time

- Move Business Logic into HANA

- Performance tuning is not a separate activity. Keep an eye of the query performance as you model.

- Test the model early and when small (after the first join). Thoroughly test the model (all join fields & calculated columns) before moving to the next. Almost complete is not good enough.

- Ensure tables are pruned when not queried and where clause pushed down to the table-level. Check and confirm using plan visualization.
Transformation

• Degree of transformation or materialization
  • Master data (dimension)
    ✓ recommended to simplify models (fields used in joins or filters) (same or new table)

• Transaction (fact) data:
  ✓ recommended to simplify models (fields used in joins or filters) (same table)
  ✓ acceptable to simplify models (calculation before aggregation, data type conversion) (same table)
  ✓ avoid creating aggregates/materialized table (absolute last option)
SAP HANA Modeling health check

**Data transfer**
- Minimize data transfer between Analytical views, Calculation views & front end tools (or SAP HANA clients)
- Avoid executing open ended queries (using KEYs)
- Reducing the data set as early as possible
- Enforce data aggregation (GROUP BYs) using fewer (and more granular) dimensions
- Use data filters (WHERE clauses, prompts, constraint filters & Analytical privileges)

**Joins**
- Use Unions (with constants) to combine large data sets
- Join views (within Calculation Views, within Universes or using SQL) with caution. Use WHERE clauses + data filters that can be pushed down into each view
- Define cardinality N:1 fact to dimension (star schema) & enforce left-outer join where possible
- Minimize long join chains and big data set joins. Where possible move data foundation joins into Attribute views

**Calculations**
- Imbed business logic calculations into modeled views instead of within SQL statements, Procedures, Universes or front end
- Minimize the use of expensive calculations, row based expressions & data manipulation including Calculated attributes
- Transform complex expressions before hand using SLT, Data Services or Generated Columns
HANA SP11 and beyond

- **Primary use of calculation views:** In future releases (SP11 and beyond), HANA optimizer will select appropriate engines during query execution. This will eliminate the need for creating different types of views for different engines (i.e. star-join in calculation view will be comparable to analytic view). However the model design principles remains the same. We’ll still create dimensions & star-schema views, but using calculation views.

- **Support of existing Views:** HANA will be backyard compatible and will still support analytic & attribute views. Also a migration tool may be available to convert the attribute and analytic views to calculation views.
Customer Workshop
Beverage Major

• **Stats**
  - before: 5mins, after: 3sec

• **Changes (SP08)**
  - Move business logic to HANA layer
  - Efficient SQL query (some were 5000 lines)
  - Use appropriate filter
  - Avoid single node *feeding multiple* nodes.
  - Constant mapping in unions
Pension Plan Management

• Stats
  ✓ before: 30sec, after: 3sec
  ✓ before: 2min (concurrency 5), after: 7 sec (concurrency 15)

• Changes (SP08, HEC)
  ✓ Efficient SQL query
  ✓ Implement star-schema (analytical view)
  ✓ Push high cardinality joins to the lowest level (analytic & attribute views)
  ✓ Use left outer joins. Set cardinality (n:1 or 1:1)
  ✓ Convert joins to unions (esp. joins of analytical views)
  ✓ Avoid single node feeding multiple nodes.
  ✓ Avoid decision table and scripted-calculation view*
### Grocery Retailer

- **Stats**
  - ✔ before: 19sec, after: 300ms (1 user)

- **Changes (SP08)**
  - ✔ Implement star-schema (analytical view)
  - ✔ Push high cardinality joins to the lowest level
  - ✔ Convert joins to unions.
  - ✔ Use left outer joins. Set cardinality (n:1 or 1:1)
  - ✔ Avoid single node feeding multiple nodes.

---

© 2015 SAP SE or an SAP affiliate company. All rights reserved.
Audit Firm

- **Stats**
  - Firm Stats
    - before: 19min, after: 8sec (1 user)
  - My Stats
    - before: 17sec, after: 1sec
    - before: 2min (concurrency 10), after 40sec (concurrency 400)

- **Changes (SP09)**
  - Efficient SQL query
  - Use left outer joins. Set cardinality (n:1 or 1:1)
  - Set optimize join = true (SP09)
  - Scripted calculation views to table functions (SP09).
  - Push high cardinality joins to the lowest level
Questions & Answers

Please follow the blog post for Questions & Answers

Thank you

Contact information:

Abani Pattanayak,
*SAP HANA Distinguished Engineer*
Principal Consultant, SAP HANA CoE
abani.pattanayak@sap.com / 1 647-406-3392