

Performance in SAP HANA Databases

A technical workshop session/CSA Know-How Webinar

Lars Breddemann, SAP Customer Solution Adoption (CSA)
June 2013, SAP HANA SPS 5

Watch the webinar recording!

<http://scn.sap.com/community/webinars/technology-rig-know-how/content>



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.

Webinar Agenda

- Statement processing
- How can I know what the DBMS does?
- Understand how SAP HANA processes your requests
- How does a join work?
- What are Star Queries?

Statement processing

- **Idea of DB as a service (client/server)**
- Describe what data you want, not how to get it
- Developer doesn't need to care about where data is stored and how to get it to/from there
- Developer doesn't need to care about how transactional consistency is maintained
- Developer doesn't need to care about backup, recovery, privilege management, multi-user access ...
- Well, she needs to care, but doesn't need to implement it herself :)
- Relational algebra as universal foundation for data structures used (relations $\approx$ tables)
- SQL as incomplete but accepted language $\rightarrow$ gold standard for DB queries

Problems?

- **Fundamental problem:**
If I don't tell the DB what to do, how can I control performance?
- Indexes?
- DB hints?
- Funny SQL coding?
- DB parameters? (`force_fast_processing = true`)?
- Creating/deleting/updating/manipulating DB statistics?
- “Super advanced”- techniques like plan management/evolution, cardinality feedback based optimizers?

Problems?!

- **This is not what we were promised!**
- This kills developer productivity
- This is maintenance hell (are the same parameters/hints/indexes... still required in DB version x+1?)
- **So, let's code SQL however we want and let the DBMS figure it out?**
- Kinda, but bad queries will lead to bad performance
(and maybe give answers you didn't want to get in the first place)

The way it is...

- Obviously, understanding your data and your data design is key to be able to ask the kind of questions that can be answered:
 - a) at all
and
 - b) quickly.
- *"Not invented here!"* doesn't apply to data models.
- **"You touch it, you buy it...."** - it's your data model as soon as you are using it.

You touch it, you buy it...

- “You touch it, you buy it....” - it's your data model as soon as you are using it.
 - pony business...

"But, I'm just *reading* from the DB..!?"

- Reading data is **the** main use case.
(besides, this is also true for your SQL/SQLScript/application source code!)
- It's high time data processing.
- It **changes the state** of your database **instance** (but **not** of your **database**, though. Your data will stay unaltered (hopefully))
 - Data will be placed in memory (maybe be removing other data first)
 - Intermediate results will be created and stored
 - Data will be handed over to other computing nodes and eventually to the client
 - Tables and rows will be prevented from being altered
and/or
 - Copies of the changed data will be created

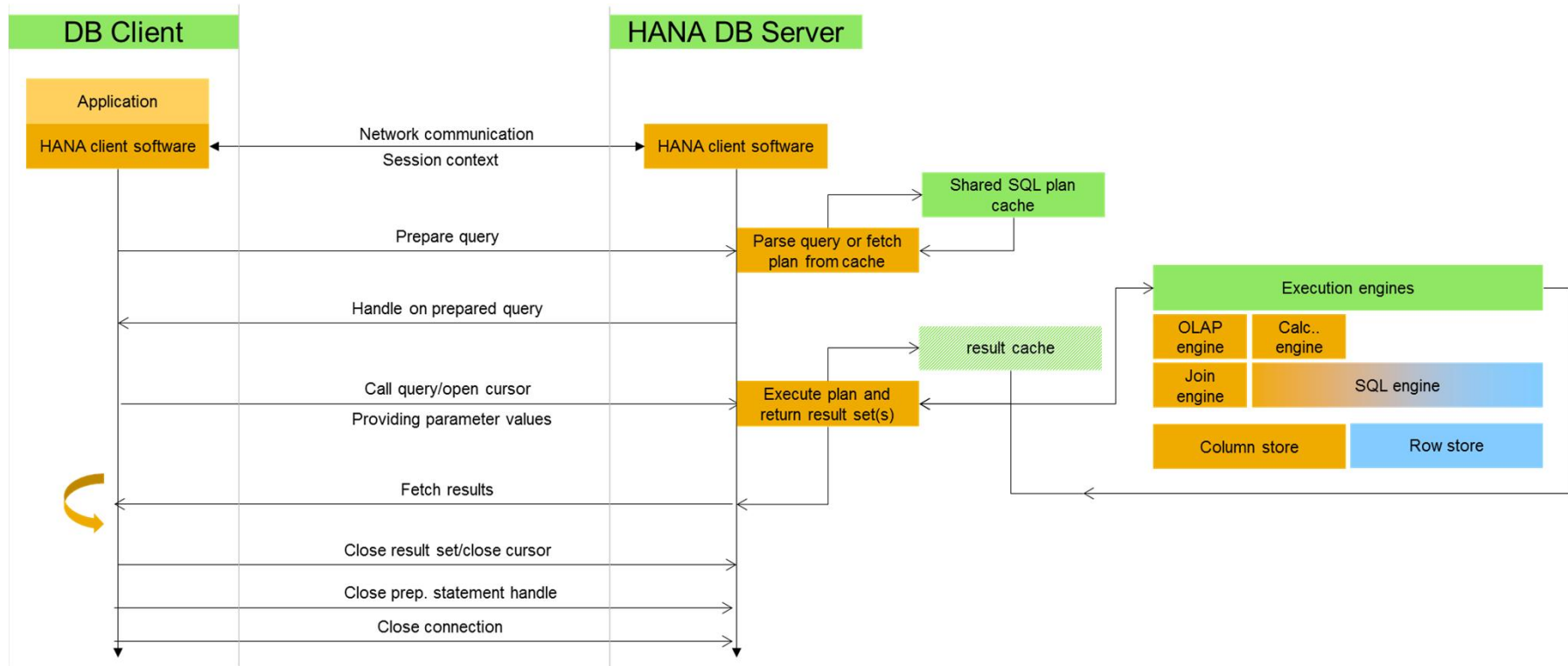
Wrapping it up...

- You better know, what you ask the DBMS to do for you
- This is where you get your expectations about possible performance from
- This is where you can start to make reasonable assumptions on what could be done

But how should I know what the DBMS does?

- Is this documented somewhere?
 - No, it's not (not really, yet)
 - And probably never will (completely).
 - Query processing techniques are the crown jewels of DBMS vendors.
 - Some of them fit only a very specific use case (maybe not *that* important to know how they work)
 - The exact implementation should and will change.
That shouldn't my SQL code.
- However, we can make some really good guesses on how data is processed
 - Information is meanwhile available...
 - SAP HANA Documentation (<http://help.sap.com>)
 - SCN forums (also saphana.com)
 - SAP Education trainings (HA100/HA200/HA300/HA360...)

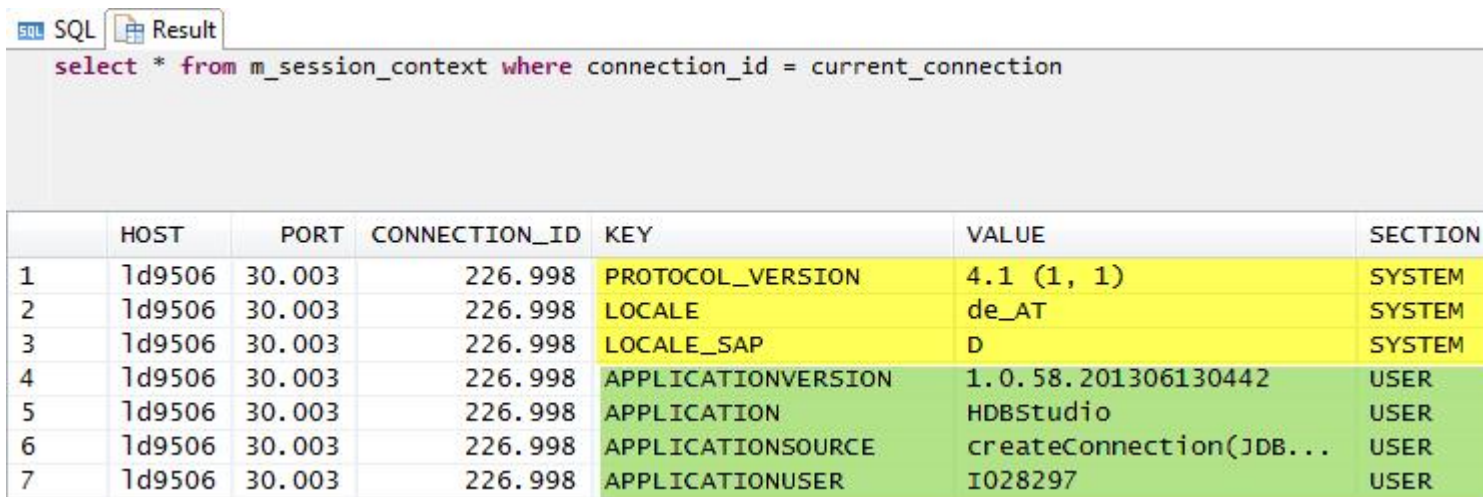
Processing a request



Understand how SAP HANA processes your requests

- It's good old client/server communication!
- Sessions, connections and context
 - Sessions seen as the "conversation" between client/server
 - A conversation can be held via one or several technical connections
 - even to different nodes
- Context defines what the conversation is about and how commands should be "understood"
 - autocommit yes/no?
 - user language
 - user privileges
 - session metadata
 - namespace → current schema (allows for short addressing of DB objects)

DEMO Session context



The screenshot shows the SQL console in SAP HANA Studio. The SQL editor contains the query: `select * from m_session_context where connection_id = current_connection`. Below the editor, the results are displayed in a table with 7 rows and 7 columns: HOST, PORT, CONNECTION_ID, KEY, VALUE, and SECTION. The first three rows (KEY: PROTOCOL_VERSION, LOCALE, LOCALE_SAP) are highlighted in yellow and labeled as 'SYSTEM' variables. The remaining four rows (KEY: APPLICATIONVERSION, APPLICATION, APPLICATIONSOURCE, APPLICATIONUSER) are highlighted in green and labeled as 'USER' variables.

	HOST	PORT	CONNECTION_ID	KEY	VALUE	SECTION
1	1d9506	30.003	226.998	PROTOCOL_VERSION	4.1 (1, 1)	SYSTEM
2	1d9506	30.003	226.998	LOCALE	de_AT	SYSTEM
3	1d9506	30.003	226.998	LOCALE_SAP	D	SYSTEM
4	1d9506	30.003	226.998	APPLICATIONVERSION	1.0.58.201306130442	USER
5	1d9506	30.003	226.998	APPLICATION	HDBStudio	USER
6	1d9506	30.003	226.998	APPLICATIONSOURCE	createConnection(JDB...	USER
7	1d9506	30.003	226.998	APPLICATIONUSER	I028297	USER

- Context variables set per session
- **SYSTEM** variables are set at login/automatically and cannot be changed during the session
- e.g. session language is set via LOCALE (ISO lang code)/LOCALE_SAP (SAP internal language encoding)
- **USER** variables can be freely defined (up to 50 per session) and accessed from outside the session (`m_session_context` view, SAP HANA studio)
- **APPLICATION*** variables not set automatically but used by convention to allow easier system monitoring
- **APPLICATIONUSER** should show the „real“ user (NetWeaver puts in ABAP user)
- **APPLICATIONSOURCE** shows source location of database call (automatically in NetWeaver)
- **APPLICATIONVERSION** should provide versioning information of the DB client

DEMO Session context – Sessions screen

The screenshot shows the SAP Sessions screen for the system 'WR1 (DEVMASTER) WR1 - Warroom Testsystem'. The table displays session data with columns for ID, USER_NAME, APPLICATION, APPLICATIONSOURCE, APPLICATIONVERSION, and APPLICATION. Callouts provide context for specific fields:

- DB user, not end user:** Points to the USER_NAME column.
- ABAP module: source code line:** Points to the APPLICATIONSOURCE column.
- Applicationversion allows for overview on used client applications, e.g. SAP HANA Studio version:** Points to the APPLICATIONVERSION column.
- User as seen from the application:** Points to the APPLICATION column.

ID	USER_NAME	APPLICATION	APPLICATIONSOURCE	APPLICATIONVERSION	APPLICATION	TIME
218.417	SAPWRN	NULL	SAPMSSY2:14480	NULL	DDIC	17.06.2013 00:30
218.480	SAPWRN	NULL	SAPMSSY2:14480	NULL	DDIC	17.06.2013 00:30
218.482	SAPWRN	NULL	SAPLRSPC_BACKEND:9700	NULL	BWREMOTE	17.06.2013 01:00
218.485	SAPWRN	NULL	SAPLRSA:3741	NULL	BWREMOTE	17.06.2013 01:00
218.486	SAPWRN	NULL	ABAP module: source code line	NULL	SAPSYS	17.06.2013 01:00
218.487	SAPWRN	NULL	SAPLRSA:3741	NULL	BWREMOTE	17.06.2013 01:00
219.117	SYSTEM	HDBStudio	createConnection(JDBCPlug...	1.0.56.201305141923	I024795	17.06.2013 06:39
219.119	SYSTEM	HDBStudio	createConnection(JDBCPlug...	1.0.56.201305141923	I024795	17.06.2013 06:39
219.135	SYSTEM	HDBStudio	createConnection(JDBCPlug...	1.0.56.201305141923	I024795	17.06.2013 06:41
219.148	SYSTEM	HDBStudio	createConnection(JDBCPlug...	1.0.56.201305141923	I024795	17.06.2013 06:41
219.257	SYSTEM	HDBStudio	createConnection(JDBCPlug...	1.0.56.201305141923	I024795	17.06.2013 06:48
219.268	SYSTEM	HDBStudio	createConnection(JDBCPlug...	1.0.56.201305141923	I024795	17.06.2013 06:48
219.273	SYSTEM	HDBStudio	createConnection(JDBCPlug...	1.0.56.201305141923	I024795	17.06.2013 06:48
219.280	SYSTEM	HDBStudio	createConnection(JDBCPlug...	1.0.56.201305141923	I024795	17.06.2013 06:50
219.288	SYSTEM	HDBStudio	createConnection(JDBCPlug...	1.0.56.201305141923	I024795	17.06.2013 06:51
219.291	SYSTEM	HDBStudio	createConnection(JDBCPlug...	1.0.56.201305141923	I024795	17.06.2013 06:51
219.299	SYSTEM	HDBStudio	createConnection(JDBCPlug...	1.0.56.201305141923	I024795	17.06.2013 06:51
219.304	SYSTEM	HDBStudio	createConnection(JDBCPlug...	1.0.56.201305141923	I024795	17.06.2013 06:51
219.321	SYSTEM	HDBStudio	createConnection(JDBCPlug...	1.0.56.201305141923	I024795	17.06.2013 06:56
222.624	SYSTEM	HDBStudio	NULL	1.0.51.201302171030	NULL	17.06.2013 07:56
222.626	SYSTEM	HDBStudio	NULL	1.0.51.201302171030	NULL	17.06.2013 07:56

DEMO Session context – english language

SQL SQL Result

select * from m_session_context where connection_id = current_connection

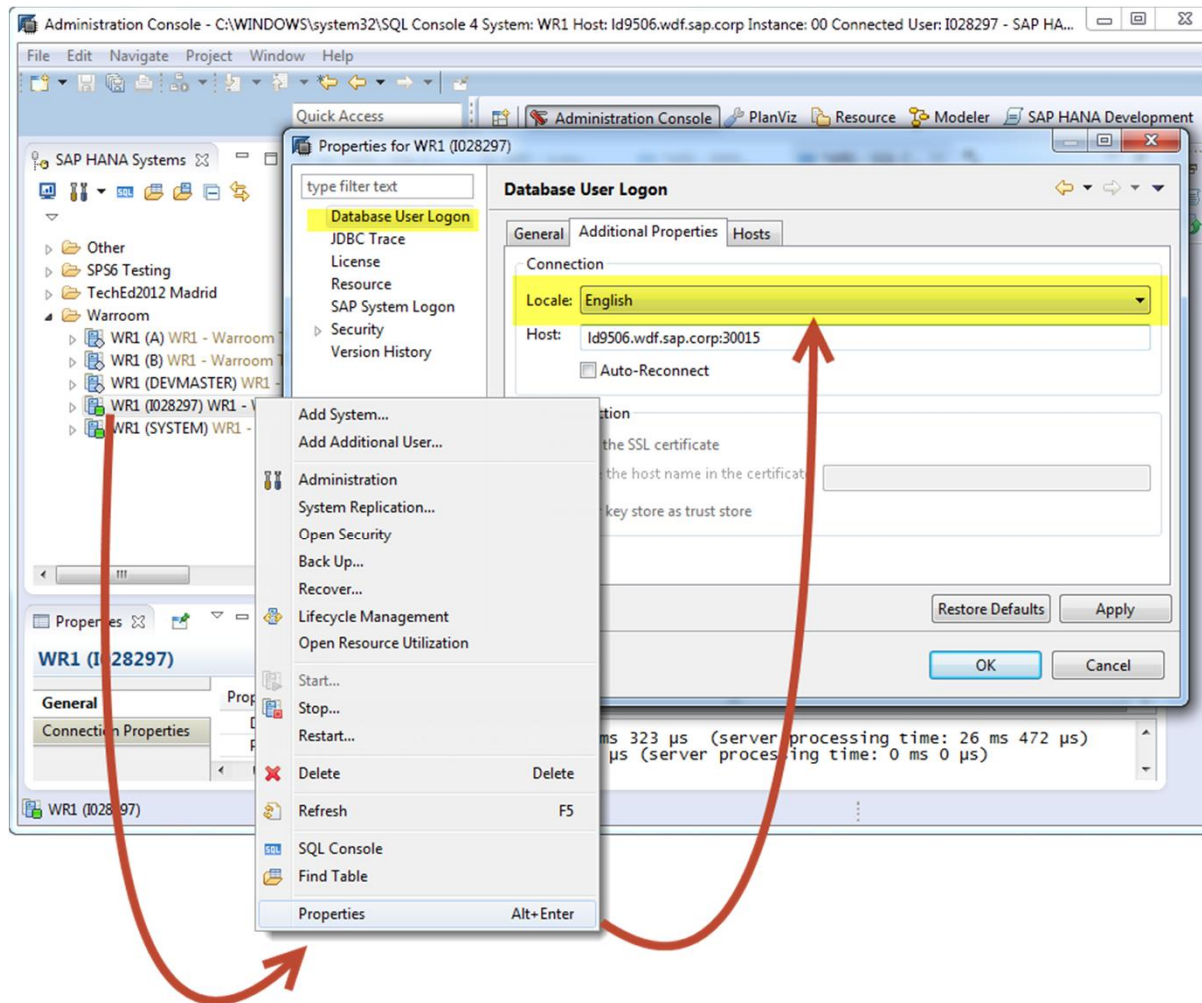
	HOST	PORT	CONNECTION_ID	KEY	VALUE	SECTION
1	1d9506	30.003	203.716	PROTOCOL_VERSION	4.1 (1, 1)	SYSTEM
2	1d9506	30.003	203.716	LOCALE		SYSTEM
3	1d9506	30.003	203.716	APPLICATIONVE...	1.50...	USER
4	1d9506	30.003	203.716	APPLICATION	HDBStudio	USER
5	1d9506	30.003	203.716	APPLICATIONSO...	run(SQL...	USER
6	1d9506	30.003	203.716	APPLICATIONUSER	i028297	USER

SQL SQL Result

SELECT TOP 20
"ARTICLE_LABEL_LOCALE",
sum("MARGIN") AS "MARGIN",
sum("AMOUNT_SOLD") AS "AMOUNT_SOLD",

	ARTICLE_LABEL_LOCALE	MARGIN	AMOUNT_SOLD	QUANTITY_SOLD
1	side slit Long skirt	64.437,700000000004	130.569,09999999992	598
2	Lycra Trousers	45.848,600000000006	105.164,09999999992	600
3	Corduroy shorts	26.852,000000000015	76.395,49999999994	429
4	Quilted Lycra skirt	23.65,100000000002	68.215,100000000003	346
5	Long Lace skirt	53,3000000000003	54.964,800000000002	414
6	Goat wool skirt	1.625,9000000000005	49.195,600000000006	267
7	Burlington Trousers	20.461,900000000005	40.095,300000000001	173
8	Canvas shorts	7.329,399999999998	28.775,399999999994	188
9	Printed Apple Tre...	9.790,199999999997	25.044,499999999993	117
10	Suede Lycra skirt	5.685,9000000000001	19.785,9	153

DEMO Session context – set session language in HANA Studio



DEMO Session context – german language

SQL

Result

select * from m_session_context where connection_id = current_connection

LOCALE = ISO
LOCALE_SAP = SAP language
encoding (1 char)

	N_ID	KEY	VALUE	SECTION
1	690	PROTOCOL_VERSION	4.1 (1, 1)	SYSTEM
2	690	LOCALE	de_AT	SYSTEM
3	690	LOCALE_SAP	D	SYSTEM
4	1d9506 30.003	203.690 APPLICATIONVE...	1.50.0....	USER
5	1d9506 30.003	203.690 APPLICATION	HDBstudio	USER
6	1d9506 30.003	203.690 APPLICATIONSO...	run(SQL...	USER
7	1d9506 30.003	203.690 APPLICATIONUSER	i028297	USER

SQL

Result

SELECT TOP 20
"ARTICLE_LABEL_LOCALE",
sum("MARGIN") AS "MARGIN",
sum("AMOUNT_SOLD") AS "AMOUNT_SOLD",

	ARTICLE_LABEL_LOCALE	MARGIN	AMOUNT_SOLD
1	IN DEUTSCH: Side slit Long skirt	64.437,7000000000004	130.569,09999999992
2	IN DEUTSCH: Lycra Trousers	45.843,6000000000006	105.164,09999999992
3	IN DEUTSCH: Corduroy Shorts	26.802,0000000000015	76.395,49999999994
4	IN DEUTSCH: Quilted Lycra skirt	27.859,1000000000002	68.215,10000000003
5	IN DEUTSCH: Long Lace skirt	3.103,3000000000003	54.964,80000000002
6	IN DEUTSCH: Goat wool skirt	19.625,9000000000005	49.195,600000000006
7	IN DEUTSCH: Burlington Trousers	20.461,9000000000005	40.095,300000000001
8	IN DEUTSCH: Canvas Shorts	7.329,3999999999998	28.775,399999999994
9	IN DEUTSCH: Printed Apple Tre...	9.790,1999999999997	25.044,499999999993
10	IN DEUTSCH: suede Lycra skirt	5.685,9000000000001	19.785,9

Processing a request

- **Server:** Check Shared SQL cache
- **Server:** Parse and optimize → execution plan (compile query program)
 - Compiled program consists of base data access operations, like "read data from column x with filter abc"
- **Client:** Request execution of prepared statement
 - Setting parameter values before
- **Server:** Execute compiled query program
- **Server:** Create result set and store it internally (materialize)
- **Server:** Notify client that result is ready
- **Client:** Reads (fetches) the result set until full result set had been read
- **Client:** Asks for closing the result set and "forgets" the handle to it

ATTENTION: a single query can create multiple result sets (e.g. procedures or grouping set queries, a.k.a. guided navigation)

Trace settings in SAP HANA Studio

The screenshot shows the 'Trace Configuration' tab in SAP HANA Studio. The left sidebar contains a navigation pane with tabs: Overview, Landscape, Alerts, Performance, Volumes, Configuration, System Information, Diagnosis Files, and Trace Configuration. The main area displays several trace configuration sections:

- Global Database Trace**: Configuration: Default. Description: The global database trace configuration overrides the default configuration for the database traces. Note: Only database traces with the default configuration are overridden. If a particular database trace is configured, this configuration will be used for that database trace.
- SQL Trace**: Status: Inactive. Description: If the SQL trace is active, the database calls for the specified database or application users are traced. The trace data is stored in files starting with sqltrace_<servername>.
- Database Trace**: Configuration: Default. Description: If the database trace is configured, the traces for the trace components (for example, INDEXSERVER and NAMESERVER) of the system are written to files named <servicename>.trc. Some of these traces are always activated by default.
- Performance Trace**: Status: Inactive. Description: If the performance trace is running, the system performance is traced. The trace data is saved to the file specified.
- User-Specific Trace**: Configuration: Not Specified. Description: If the user trace is specified, the traces for the trace components (for example, INDEXSERVER and NAMESERVER) for a specific database or application user are written to files named <servicename>_<servername>_<contextname>.trc.
- Expensive Statements Trace**: Status: Active. Description: If the expensive statements trace is active, all statements that last longer than the specified threshold are traced. You can analyze the results on the Performance tab under Expensive Statements Trace.
- End-to-End Traces**: Description: The predefined end-to-end traces are used by applications to record the steps through all the available trace components (for example, INDEXSERVER and NAMESERVER) in a configuration. When an end-to-end trace is used, the traces for the trace components are written to files named <servicename>_<servername>_<end-to-end_trace_name>.trc.
 - sap_passport_high**: Configuration: Default
 - sap_passport_medium**: Configuration: Default
- Kernel Profiler**: Status: Inactive. Description: The kernel profiler helps you to analyze performance issues with systems on which third-party software cannot be installed. It is also a useful tool for investigating issues with customer systems, as it allows you to analyze issues in parts of the SAP HANA database where performance trace and function profiler tools are not available.

Annotations (yellow boxes) point to specific settings:

- Default settings for traces**: Points to the Global Database Trace section.
- Actual system wide trace settings**: Points to the Database Trace section.
- Actual user specific trace settings**: Points to the User-Specific Trace section.
- Setup for NetWeaver E2E-traces via SAP PASSPORT**: Points to the End-to-End Traces section.
- system wide or user specific SQL trace**: Points to the SQL Trace section.
- SAP internal performance trace & profiler**: Points to the Performance Trace section.
- Expensive statement trace**: Points to the Expensive Statements Trace section.
- SAP internal sampling profiler**: Points to the Kernel Profiler section.

Alternatively, let the HANA tell you how it processes your queries

- **SQLTRACE (PYTHON TRACE)**
- **PERFORMANCE TRACE + FUNCTION PROFILER**
- **KERNEL PROFILER**
- **OPTIMIZER TRACES**
- **LOAD GRAPH**
- **STATISTICS SERVER**
- **EXPENSIVE STATEMENTS TRACE**
- **SQL PLAN CACHE**
- **EXPLAIN PLAN**
- **PLAN VISUALIZATION**

SQLTRACE (PYTHON TRACE)

- Incomprehensible w/o SAP HANA development knowledge
- Is like “*taking a sledgehammer to crack a nut*”
- doesn't provide what *happens* during the execution (rather input → output trace)

PERFORMANCE TRACE + FUNCTION PROFILER

- TREX heritage
- unsupported for non-SAP-Development usage
- requires HANA server access + X11-Window client
- requires special privileges
- doesn't cover row store activity
- tool is barely documented and complex
- function profiler is useless w/o source code

KERNEL PROFILER

- stack code sampling
- profiler tells you how often a specific part of the SAP HANA code was executed and how long that took (approx.)
- again, useless w/o source code

OPTIMIZER TRACES

- `sqlopttime` (runtime information for the query optimization phase)
- `sqloptStep` (single query optimization/transformation steps)
- `trex_qo` (TREX query optimizer → column store plans)

Alternatively, let the HANA tell you how it processes your queries

- SQLTRACE (PYTHON TRACE)

- PERFORMANCE TRACE + F

- KERNEL PROFILER

- OPTIMIZER TRACES

- **LOAD GRAPH**

- **STATISTICS SERVER**

- **EXPENSIVE STATEMENTS TRACE**

- **SQL PLAN CACHE**

- **EXPLAIN PLAN**

- **PLAN VISUALIZATION**

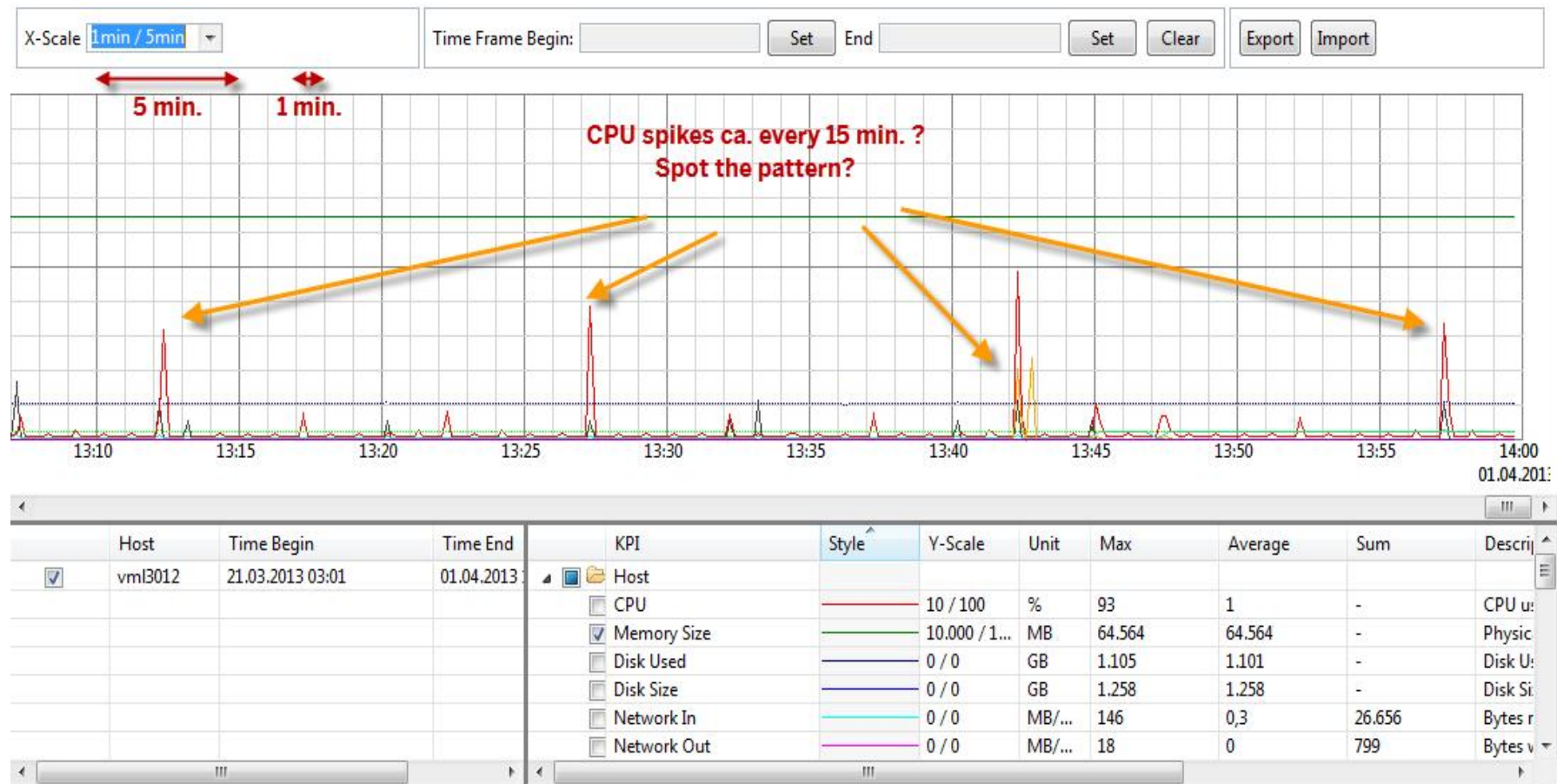


pretty useless for mere mortals like us...

LOAD GRAPH

- Nameserver data (nameserver_history.trc)
- TREX heritage
- Highly aggregated data - cannot (easily) be mapped to a specific user action
- Yet, give a "feeling" of how workload "looks like"
- Could be used to give a rough baseline
- No function to compare to time frames side-by-side or by overlay
- MIN/MAX/AVG numbers are highly volatile and fully depend on the chosen time frame

LOAD GRAPH - Screenshot



STATISTICS SERVER

- `_SYS_STATISTICS` schema
- Lots of data, gathered by automatic snapshots
- Are purged automatically
- Is used as the data source for solution manager (performance warehouse)
- Provide alerts for SAP HANA

STATISTICS SERVER

The screenshot displays the SAP Statistics Server interface. The main window is titled "WR1 (SYSTEM) WR1 - Warroom Testsystem" and shows a list of alerts. Below the alerts, there is a "Check Information" table. To the right, a list of statistics tables is shown under the "_SYS_STATISTICS" folder.

Alerts:

Time	Description
Last 15 min (0/1/0)	
Last 30 min (0/1/0)	
Last hour (1/1/2)	
06.06.13 01:06	BIG_DATA.Z_FACTS contains 1770337637 records.
06.06.13 01:06	BIG_DATA.Z_FACTS partition 0 contains 1770337637 records. A table partition cannot contain more than 1770337637 records.
06.06.13 01:06	There are currently 152 diagnosis files. This might indicate a problem with trace file rotation.
06.06.13 02:00	The last data backup is 9 days old. This will increase downtime in a recovery situation.
Last 2 hours (2/1/4)	
06.06.13 00:06	BIG_DATA.Z_FACTS contains 1770337637 records.
06.06.13 00:06	BIG_DATA.Z_FACTS partition 0 contains 1770337637 records. A table partition cannot contain more than 1770337637 records.

Check Information:

ID	Check	Max Priority	Last Run
0	Internal statistics server problem		<not available>
1	Host physical memory usage		06.06.2013 02:04:35
2	Disk usage		06.06.2013 02:01:34
3	Inactive services		06.06.2013 02:04:35
4	Restarted services		06.06.2013 02:04:35
5	Host CPU Usage		06.06.2013 02:04:35
10	Delta merge (mergedog) configuration		06.06.2013 01:06:54
12	Memory usage of name server		06.06.2013 02:04:35
16	Lock wait timeout configuration		06.06.2013 01:06:54

Statistics Tables:

- Column Views
- Functions
- Indexes
- Procedures
- Sequences
- Synonyms
- Tables
 - GLOBAL_COLUMN_TABLES_SIZE
 - GLOBAL_CPU_STATISTICS
 - GLOBAL_DEC_EXTRACTOR_STATUS
 - GLOBAL_DISKS
 - GLOBAL_INTERNAL_DISKFULL_EVENTS
 - GLOBAL_INTERNAL_EVENTS
 - GLOBAL_MEMORY_STATISTICS
 - GLOBAL_PERSISTENCE_STATISTICS
 - GLOBAL_ROWSTORE_TABLES_SIZE
 - GLOBAL_TABLES_SIZE
 - GLOBAL_TABLE_PERSISTENCE_STATISTICS
 - HOST_BLOCKED_TRANSACTIONS
 - HOST_COLUMN_TABLES_PART_SIZE
 - HOST_CONNECTIONS
 - HOST_CONNECTION_STATISTICS
 - HOST_CS_UNLOADS
 - HOST_DATA_VOLUME_PAGE_STATISTICS
 - HOST_DATA_VOLUME_SUPERBLOCK_STATISTICS
 - HOST_DELTA_MERGE_STATISTICS
 - HOST_HEAP_ALLOCATORS
 - HOST_LONG_IDLE_CURSOR
 - HOST_LONG_RUNNING_STATEMENTS

EXPENSIVE STATEMENTS TRACE

- Provides information about runtimes of single statement executions
- Captures the parameter values for parameterized statements
- Can be easily configured to focus on the real long running statements
- What long running statements are, clearly depends on your use case.
- Sometimes this means milliseconds, sometimes it's minutes

EXPENSIVE STATEMENTS TRACE – Setup screen

The screenshot displays the 'Expensive Statements Trace' setup screen in the Warroom Testsystem. The main window shows a table of SQL statements with columns: CONNECTION_ID, STATEMENT_ID, DB_USER, APP_USER, STATEMENT_STRING, and DURATION_MICROSECONDS. A configuration dialog box is open, titled 'Expensive Statements Trace', with the following options:

- Trace Status: ☐ Inactive ☒ Active
- Threshold Duration (µs): 500000
- Buttons: Restore Defaults, < Back, Next >, Finish, Cancel

12	CONNECTION_ID	RB	STATEMENT_ID	RB	DB_USER	RB	APP_USER	RB	STATEMENT_STRING	12	DURATION_MICROSECONDS
232.183			997218825607679		SYSTEM		1028297		SELECT "THREADS","HOST","TH...		600
232.915			1000362368870435		_SYS_STATISTI...				select HOST PORT SERVICE...		840
232.915			1000362321656404		_SYS_STATISTI...				select HOST, ROUND(sum(USED...		810
232.914			1000359220064382		SYS_STATISTI...				select HOST PORT SERVICE...		840
232.914			1000359359955960								770
232.912			1000350855578915								800
232.912			1000349977262957								730
232.911			1000346372483504								840
232.911			1000346029942957								730
232.909			1000337129935564								870
232.909			1000336744086561								800
232.908			1000333938465779								830
232.908			1000332261656446								740
232.906			1000324602749189								9.650
232.904			1000316445543360								500
232.906			1000325234641888								9.940
232.904			1000315995220495								4.430
232.904			1000315995220495								4.430
232.904			1000316262756914								1.120
232.904			1000316527966304		_SYS_STATISTI...				select reason, count(*) as numb...		1.210
232.904			1000315538118792		_SYS_STATISTI...				select "SCHEMA_NAME" ':' "		3.240
232.904			1000315538118792		SYS_STATISTI...				select "SCHEMA_NAME" ':' "		3.230

EXPENSIVE STATEMENTS TRACE - Navigation

Overview | Landscape | Alerts | Performance | Volumes | Configuration | System Information | Diagnosis Files | Trace Configuration

Threads | Sessions | Blocked Transactions | SQL Plan Cache | Expensive Statements Trace | Job Progress | Load

Enter your filter Visible rows: 1800/1800 Configure Add filter Save as file

NT_ID	AB	DB_USER	AB	APP_USER	START_TIME	DURATION...	AB	OBJECT_NAME	12	RECORDS	AB	STATEMENT_STRING
62		BACKUP_AD...			20.06.2013 22:15:02	1.487.749.527				0		BACKUP DATA ALL USING FILE ('/usr/sap/
13		BACKUP_AD...			22.06.2013 22:15:01	1.487.749.527				0		BACKUP DATA ALL USING FILE ('/usr/sap/
50		BACKUP_AD...								0		BACKUP DATA ALL USING FILE ('/usr/sap/
32		BACKUP_AD...								0		BACKUP DATA ALL USING FILE ('/usr/sap/
70	I815462	I815462	I815462							365		SELECT "FIELDA", "FIELDDB", "KEYDATE", SU
70	I815462	I815462	I815462							32		SELECT "FIELDA", "FIELDDB", "KEYDATE", SU
77	SYSTEM	d0316								0		{ call SYS.REPOSITORY_REST (? ?) }
70	SAPWRN	DDIC								1		SELECT * FROM "TSP0
34	SYSTEM	d0316								0		{ call SYS.REPOSITORY
105	I815462	I815462								0		{ call SYS.REPOSITORY
70	SYSTEM	d031643			20.06.2013 17:04:22	45.489.971		SYS.REPOSITORY_REST		0		{ call SYS.REPOSITORY
46	SYSTEM	d020788			21.06.2013 14:00:18	42.109.527				0		grant "DATA_ADMIN
18	SYSTEM	d020788			21.06.2013 13:59:28	41.329.651				0		grant "CONTENT_ADMIN" to "D020788"
145	SYSTEM	d031643			20.06.2013 17:03:39	40.374.016		SYS.REPOSITORY_REST		0		{ call SYS.REPOSITORY_REST (? ?) }
54	I833916				20.06.2013 15:51:02	37.069.820		SYS.REPOSITORY_REST		0		{ call SYS.REPOSITORY_REST (? ?) }
95	SYSTEM	d031643			20.06.2013 17:08:37	36.684.323		SYS.REPOSITORY_REST		0		{ call SYS.REPOSITORY_REST (? ?) }
3	SYSTEM	I028297			24.06.2013 15:35:49	34.747.379				-1		alter system reclaim log
79	I815462	I815462			20.06.2013 17:41:36	33.624.550		SYS.REPOSITORY_REST		0		{ call SYS.REPOSITORY_REST (? ?) }

double click

right click

STATEMENT_STRING

SELECT "FIELDA", "FIELDDB", "KEYDATE", SUM("COUNT") FROM BIG_DATA.Z_FACTSGROUP BY "FIELDA", "FIELDDB", "KEYDATE"

Copy Save As... Close

Column filters

DB_USER (17/18) - Not Equal

_SYS_STATISTICS

Quick filter on [SELECT "FIELDA", "FIELDDB" ...]

Distinct values for [STATEMENT_STRING]

Copy row

Visualize Plan

SQL PLAN CACHE

- Contains two kind of information:
 1. parsed statements
 2. call and runtime statistics for these statements
- Can answer questions like:
 - "How much time was spent on actually executing the statement?"
 - "Can my statement be reused or does it need to be optimized over and over again?"
- Depending on the use case, the default cache size can be much to small
 - We're gaining experience with that right now

Alternatively, let the HANA tell you how it processes your queries

- SQLTRACE (PYTHON TRACE)
- PERFORMANCE TRACE + FUNCTION PROFILER
- KERNEL PROFILER
- OPTIMIZER TRACES
- **LOAD GRAPH**
- **STATISTICS SERVER**
- **EXPENSIVE STATEMENTS TRACE**
- **SQL PLAN CACHE**
- **EXPLAIN PLAN**
- **PLAN VISUALIZATION**



**System wide monitoring –
the BIG picture**

EXPLAIN PLAN

- Can't look "into" modelled views
- Poor formatting
- Focusses on "costs" and estimated cardinalities (no. of rows)
- First information on which engine is used

EXPLAIN PLAN – Demo 1

```
SELECT TOP 20
  "ARTICLE_LABEL_LOCALE",
  sum("MARGIN") AS "MARGIN",
  sum("AMOUNT_SOLD") AS "AMOUNT_SOLD",
  sum("QUANTITY_SOLD") AS "QUANTITY_SOLD"
FROM "_SYS_BIC"."i028297.demo/AN_SALES"
WHERE "FAMILY_NAME" IN ('City Trousers', 'City Skirts')
GROUP BY
  "ARTICLE_LABEL_LOCALE"
ORDER BY "AMOUNT_SOLD" desc;
```

ARTICLE_LABEL_LOCALE	MARGIN	AMOUNT_SOLD	QTY_SOLD
Side Slit Long Skirt	64.437,70000000000004	130.569,099999999992	598
Lycra Trousers	45.848,60000000000006	105.164,099999999992	600
Corduroy Shorts	26.852,00000000000015	76.395,499999999994	429
...			

EXPLAIN PLAN – Demo 2

The screenshot shows the SAP SQL Editor interface. On the left, the 'SQL' tab is active, displaying the following query:

```
1 SELECT TOP 20
2     "ARTICLE_LABEL_LOCALE",
3     sum("MARGIN") AS "MARGIN",
4     sum("AMOUNT_SOLD") AS "AMOUNT_SOLD",
5     sum("QUANTITY_SOLD") AS "QUANTITY_SOLD"
6 FROM "_SYS_BIC"."i028297.demo/AN_SALES"
7 WHERE "FAMILY_NAME" IN ('City Trousers', 'City Skirts')
8 GROUP BY
9     "ARTICLE_LABEL_LOCALE"
10 ORDER BY "AMOUNT_SOLD" desc;
11
```

A red arrow points to the end of the query with the text "right click". On the right, a context menu is open, listing various actions. The "Explain Plan" option is highlighted in yellow.

Action	Shortcut
Undo Typing	Ctrl+Z
Revert File	
Save	Ctrl+S
Save As...	
Show In	Alt+Shift+W ▶
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Open File...	
Preferences...	
Execute	F8
Prepare Statement	Ctrl+Shift+P
Explain Plan	Ctrl+Shift+Z
Visualize Plan	Ctrl+Shift+V
Choose Connection	
Clear	F2
Close All Results	
Format	Ctrl+Shift+O
Show Definition	

EXPLAIN PLAN – Demo 3

SQL

Result

EXPLAIN PLAN FOR
SELECT TOP 20
"ARTICLE_LABEL_LOCALE",
sum("MARGIN") AS "MARGIN",
..

	OPERATOR_NAME	OPERATOR_DETAILS	EXECUTION_ENGINE	SCHEMA_NAME	TABLE_NAME	TABLE_TYPE	TABLE_SIZE
1	COLUMN SEARCH	i028297.demo/AN_SALES.ARTICLE_LABEL_LOCALE, SUM(i028297.d...	COLUMN	NULL	NULL	NULL	NULL
2	LIMIT	NUM RECORDS: 20	COLUMN	NULL	NULL	NULL	NULL
3	ORDER BY	SUM(i028297.demo/AN_SALES.AMOUNT_SOLD) DESC	COLUMN	NULL	NULL	NULL	NULL
4	AGGR...	GROUPING: i028297.demo/AN_SALES.ARTICLE_LABEL_LOCALE, AGG...	COLUMN	NULL	NULL	NULL	NULL
5	CO...	FILTER CONDITION: i028297.demo/AN_SALES.FAMILY_NAME = 'ci...	COLUMN	_SYS_BIC	i028297...	OLAP VIEW	89.17



Which Engines are used to process the query ?

EXPLAIN PLAN – Demo 4

OPERATOR_NAME	OPERATOR_DETAILS
COLUMN SEARCH	i028297.demo/AN_SALES.ARTICLE_LABEL_LOCALE, SUM(i028297.demo/AN_SALES.MARGIN), SUM(i028297.demo/AN_SALES.AMOUNT_SOLD), SUM(i028297.demo/AN_SALES.QUANTITY_SOLD) (LATE MATERIALIZATION)
LIMIT	NUM RECORDS: 20
ORDER BY	SUM(i028297.demo/AN_SALES.AMOUNT_SOLD) DESC
AGGREGATION	GROUPING: i028297.demo/AN_SALES.ARTICLE_LABEL_LOCALE, AGGREGATION: SUM(i028297.demo/AN_SALES.MARGIN), SUM(i028297.demo/AN_SALES.AMOUNT_SOLD), SUM(i028297.demo/AN_SALES.QUANTITY_SOLD)
COLUMN VIEW	FILTER CONDITION: i028297.demo/AN_SALES.FAMILY_NAME = 'City Trousers' OR i028297.demo/AN_SALES.FAMILY_NAME = 'City Skirts'

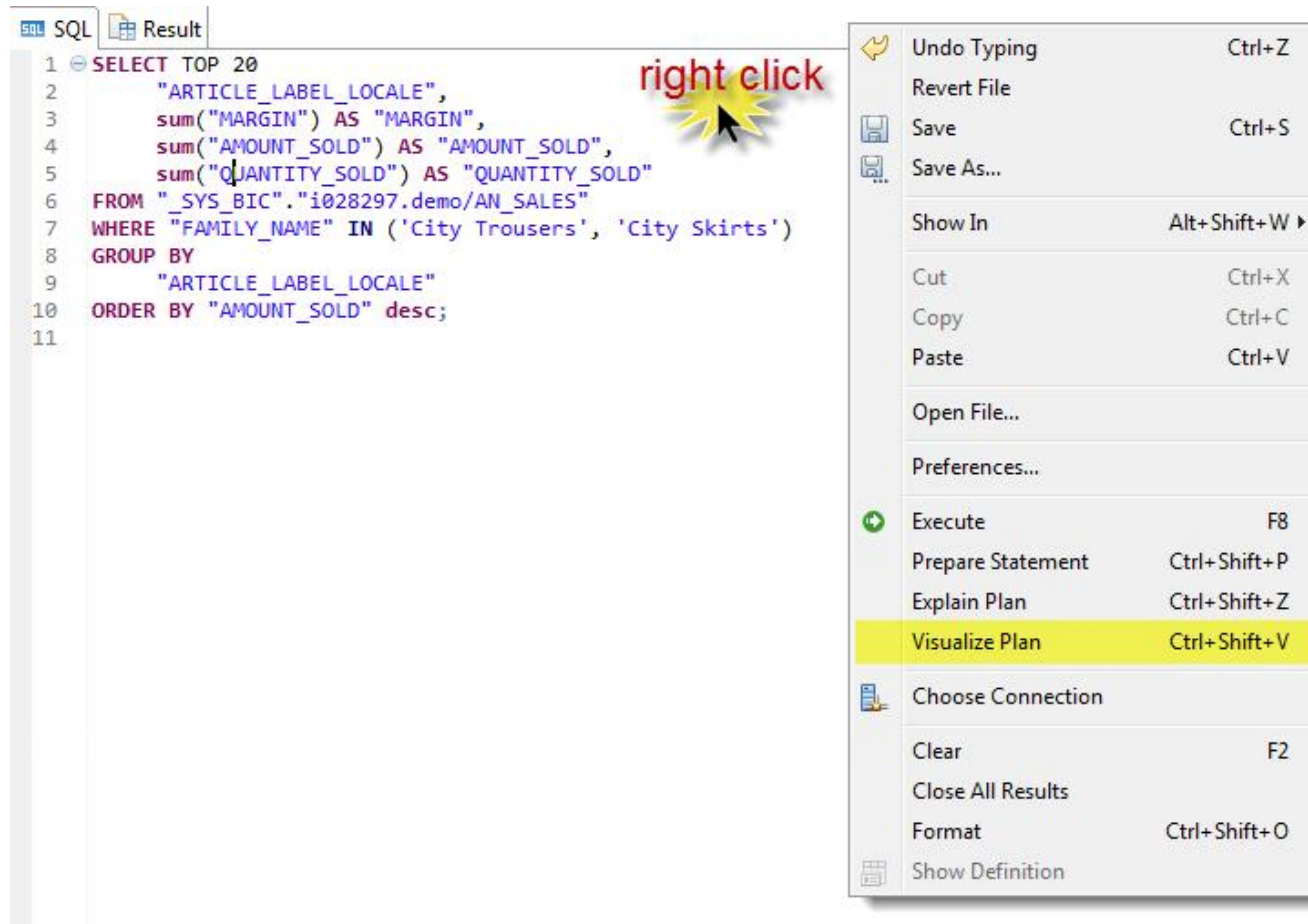
EXPLAIN PLAN – Demo 5

OPERATOR_NAME	EXEC_ENGINE	SCHEMA_NAME	TABLE_NAME	TABLE_TYPE
COLUMN SEARCH				
LIMIT	COLUMN	NULL	NULL	NULL
ORDER BY	COLUMN	NULL	NULL	NULL
AGGREGATION	COLUMN	NULL	NULL	NULL
COLUMN VIEW	COLUMN	NULL	NULL	NULL
	COLUMN	_SYS_BIC	i028297.demo/AN_SALES	OLAP VIEW

PLAN VISUALIZATION

- Relatively easy to use
- Graphical tool, build into SAP HANA studio
- Provides estimation and actual runtime statistics for memory, CPU time, parallelism and total runtime.
- Can be access via SQL Editor, SQL statement cache view, Expensive statement view
- Can be saved (XML file) and reviewed later on
- THE TOOL OF CHOICE!

Plan Viz – Demo 1



The screenshot shows the SAP SQL Editor interface. The left pane displays an SQL query, and the right pane shows a context menu after a right-click. A red arrow points to the right-click action.

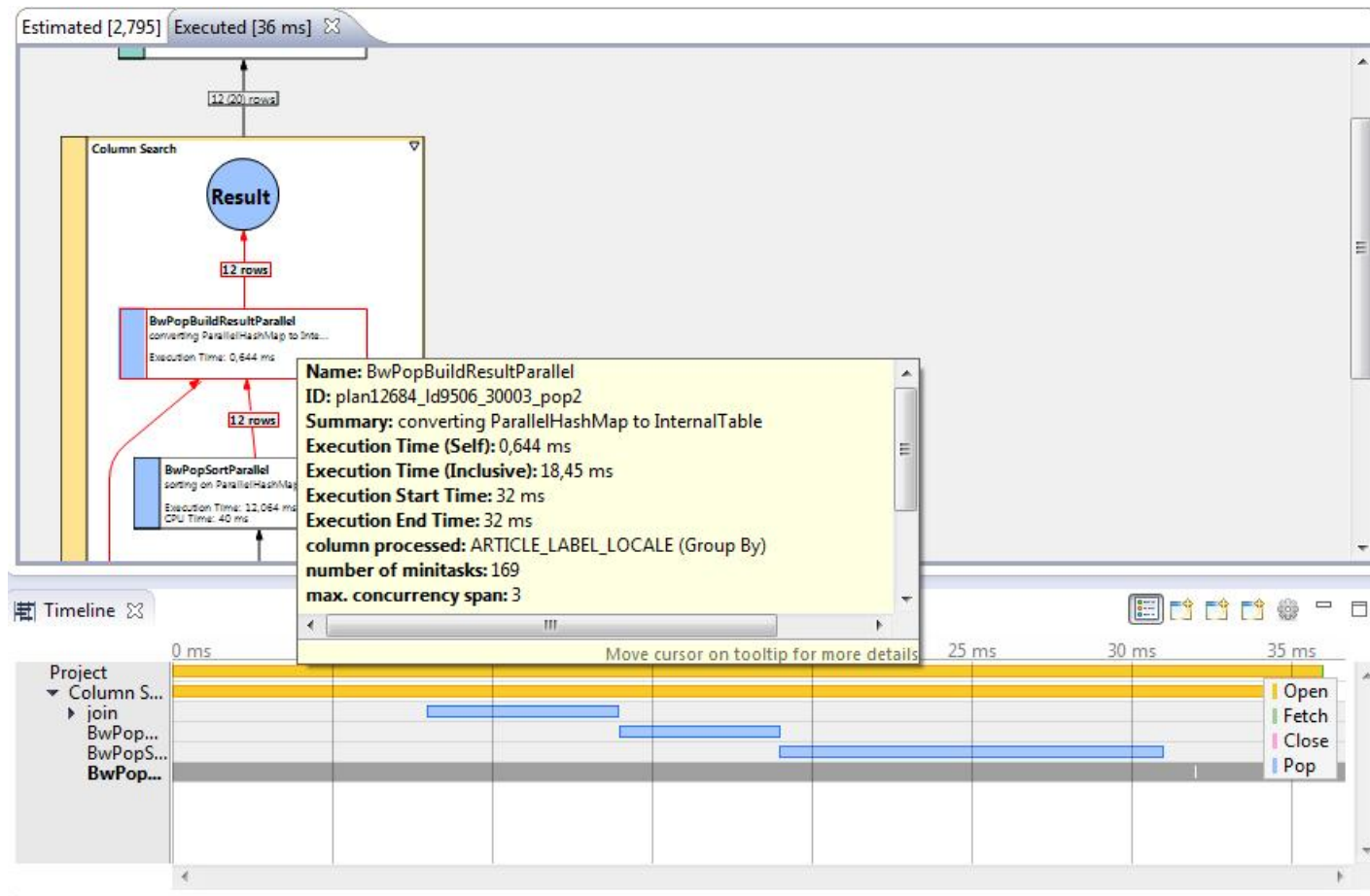
SQL Query:

```
1 SELECT TOP 20
2     "ARTICLE_LABEL_LOCALE",
3     sum("MARGIN") AS "MARGIN",
4     sum("AMOUNT_SOLD") AS "AMOUNT_SOLD",
5     sum("QUANTITY_SOLD") AS "QUANTITY_SOLD"
6 FROM "_SYS_BIC"."i028297.demo/AN_SALES"
7 WHERE "FAMILY_NAME" IN ('City Trousers', 'City Skirts')
8 GROUP BY
9     "ARTICLE_LABEL_LOCALE"
10 ORDER BY "AMOUNT_SOLD" desc;
11
```

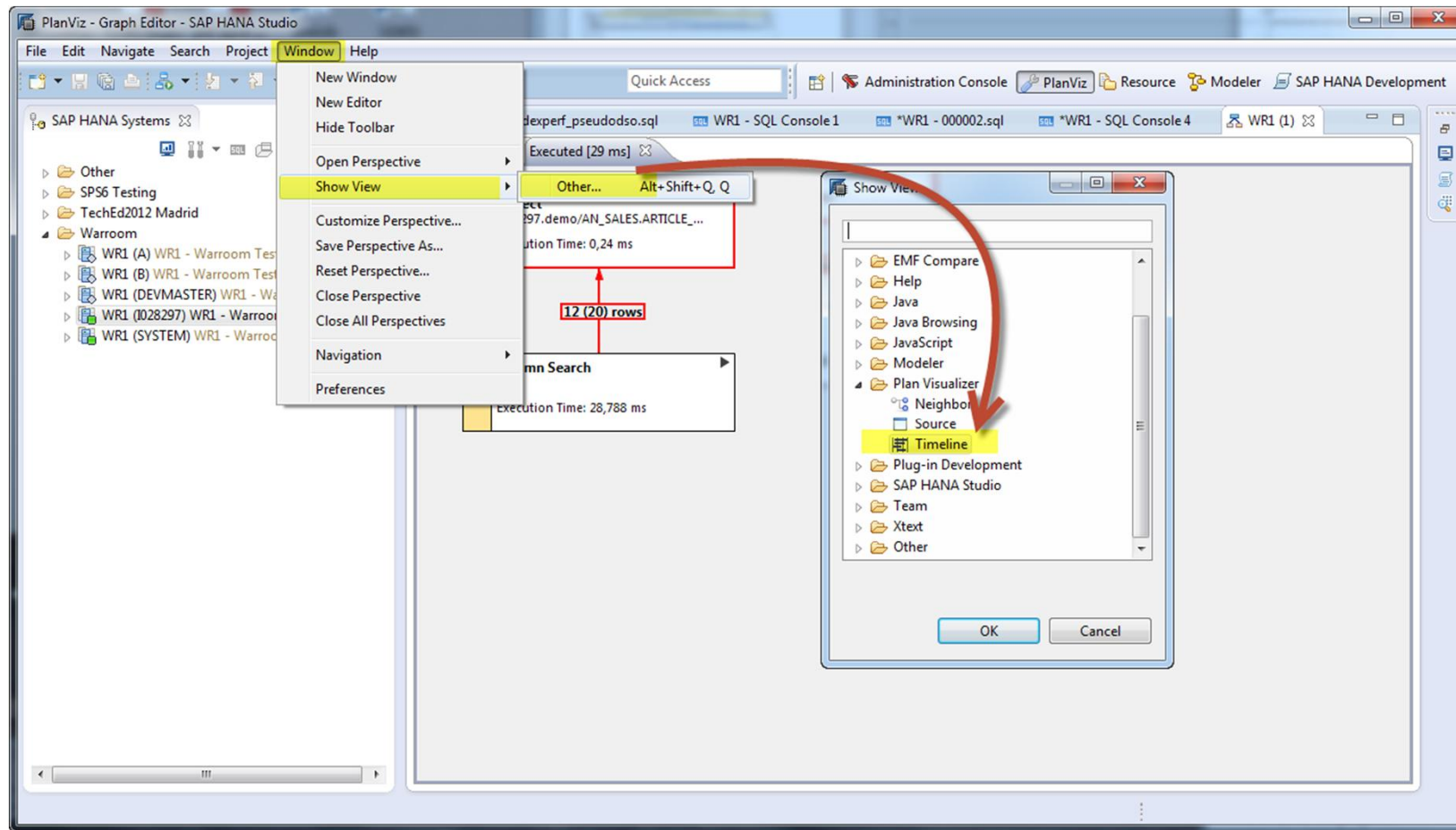
Context Menu Options:

Action	Shortcut
Undo Typing	Ctrl+Z
Revert File	
Save	Ctrl+S
Save As...	
Show In	Alt+Shift+W ▶
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Open File...	
Preferences...	
Execute	F8
Prepare Statement	Ctrl+Shift+P
Explain Plan	Ctrl+Shift+Z
Visualize Plan	Ctrl+Shift+V
Choose Connection	
Clear	F2
Close All Results	
Format	Ctrl+Shift+O
Show Definition	

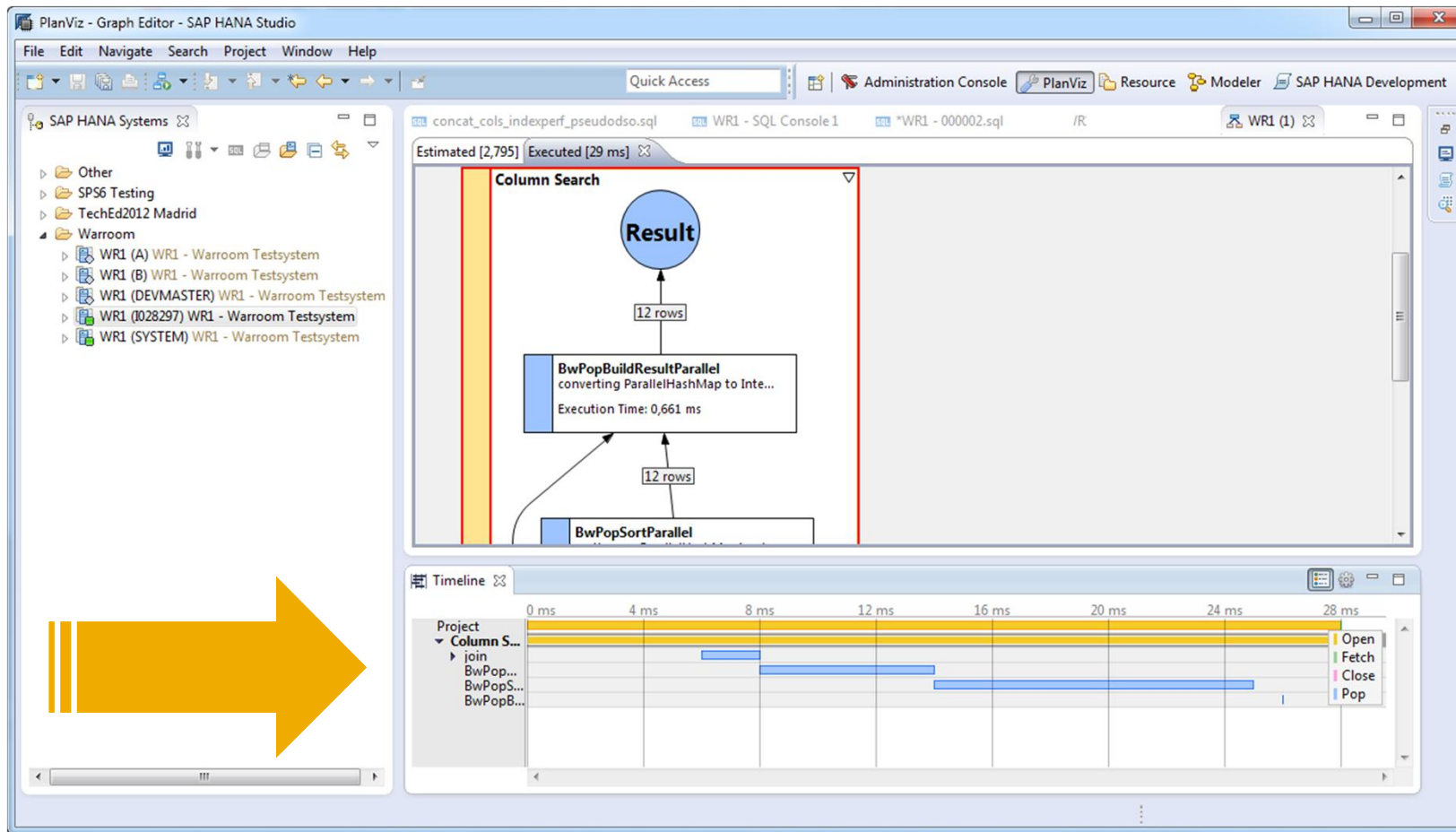
Plan Viz – Demo 2



Plan Viz – Demo 3 – display timelines

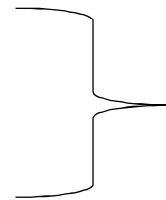


Plan Viz – Demo 4 – display timelines



Alternatively, let the HANA tell you how it processes your queries

- SQLTRACE (PYTHON TRACE)
- PERFORMANCE TRACE + FUNCTION PROFILER
- KERNEL PROFILER
- OPTIMIZER TRACES
- **LOAD GRAPH**
- **STATISTICS SERVER**
- **EXPENSIVE STATEMENTS TRACE**
- **SQL PLAN CACHE**
- **EXPLAIN PLAN**
- **PLAN VISUALIZATION**



**Single statement
analysis – what you were
looking for...**

How does a join work?

- Step back: what's a join anyhow?
 - binary operation!
 - Thunderdome rule: *"Two relations enter, one leaves"*
- How can be then join more than two tables?
 - Easy: one after the other :) (→ commutative)
- How does classic join processing look like?
- Tuple/record based...
 - Nested Loop
 - Sort/Merge
 - Hash Join

Column Store data structure (stupid simple version)

ID	Value
1	Claudia
2	Mary
3	Peter

Dictionary (sorted)

Offset	ValueID
1	1
2	1
3	2
4	3
5	1

Main Index (ValueID-Vector)

- Optimized for efficient read access
- Inserting new values would require update of dictionary and re-assignment of ValueIDs
- Changed data is therefore written to the Delta Store
- 2 Level data staging, similar to F- and E-fact table in classic SAP BW
- Base operation for searches: scan through the whole Main Index (much faster than classic looping approach as Vector-CPU instructions are used)
- Multiple columns are scanned by multiple CPU cores in parallel
- Rows are „glued back together“ via the offset of the entries in the Main Vector

Join in column store?

- Remember basic structure of column in column store:
 - Value dictionary (containing all distinct values)
 - Bit vector (referring to the Dictionary, one reference for every row)
 - Finding rows with a specific value means:
 - find reference id in dictionary of the column
 - Scan bit vector for occurrences of this/these reference IDs
 - Joining means: find specific values in **another** column of another table
 - Same values in dictionary have different reference IDs in different columns
 - To compare, the reference IDs for equal dictionary values need to be mapped
 - Say hello to the transition tables!
 - With the transition tables (equality) joins can be restated as column searches

Back to multi table join processing...

- What would be a good strategy on this?
- Start somewhere and pick the next table for which we have a join condition?
 - should lead to the same results (and does) but runtime may vary big time
- Goal: find the order of joins that executes fastest
- Next best thing to do: try to reduce the amount of data to be worked on as early as possible
- How can we do that?
 - Look for very efficient filters in our query (it's all about cardinality)
 - Understand: joining is filtering
 - So, what join(-transition) has the best filter effect?
 - Which one throws away most of the data?

Multi table join

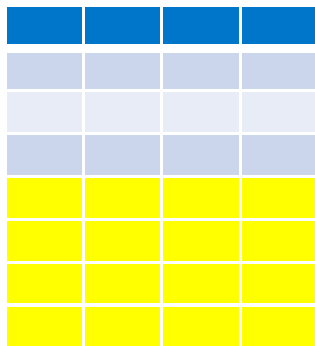


Table A
2000 rows, 1000 rows

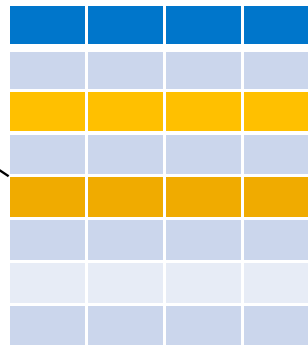


Table B
500.000 rows, filtered
10.000 rows

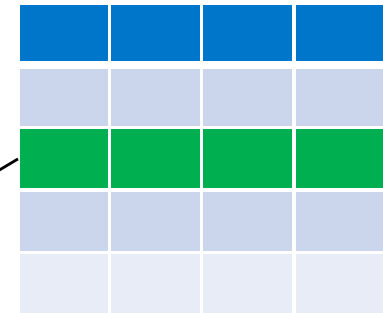


Table C
100 Mio rows, filtered ca.
1000 rows

Join order

$(A \rightarrow B) \rightarrow C$

or

$(C \rightarrow B) \rightarrow A$

or

$(B \rightarrow A) \rightarrow C$

or

$(B \rightarrow C) \rightarrow A ?$

Back to multi table join processing...

- General idea here:
 - use the defined join conditions and reduce data early
 - All nice and good, this problem is fairly well understood and boils down to a optimization problem on n permutations of join orders
- Modern DBMS create thousands of permutations and evaluate the expected processing effort (the 'costs') and pick the permutation with the lowest expected efforts
- But how do you handle STAR QUERIES...?

What are Star Queries?

- (extended) star schema
- Designed for OLAP queries
 - That's **aggregated** transactional data (SUM/AVG/MIN/MAX/LEAST/FIRST...), KEY FIGURES!, which is filtered by dimensions (sales area, channel, division, time of transaction).
 - It's specifically **NOT** a "*show me all transaction lines!*" query
- That's a story on it's own...
 - General mismatch here of expectations vs. technology
 - Often the requirement is "business needs everything"

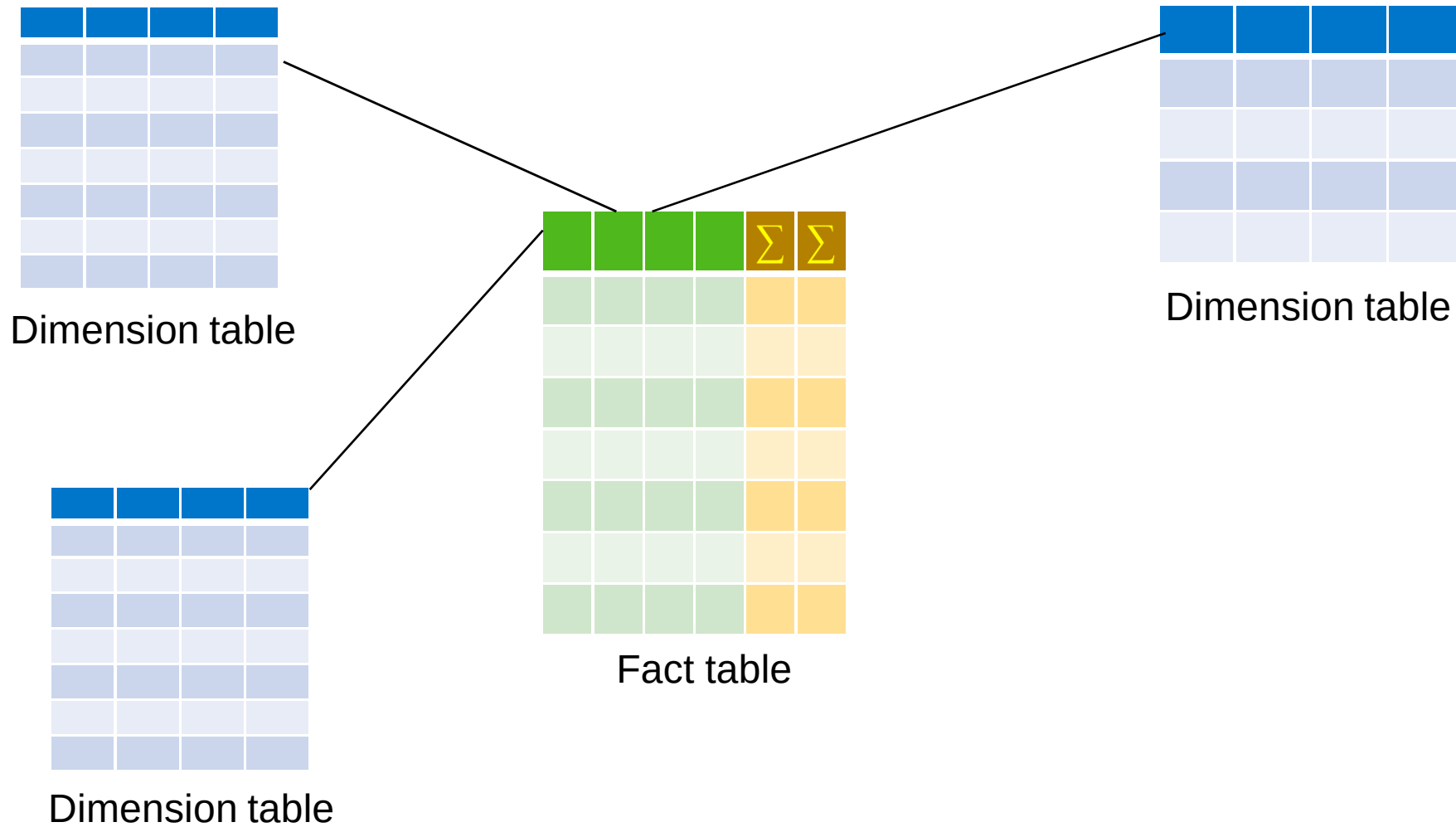
What are Star Queries?

- Does "business" really look at "everything" (thing 1.5 million line item transactions)?
 - Of course not
- So, what's the point here? Why is everything requested?
 - Defer the decision about what to focus at
 - Be independent from the IT department to change the reports
 - Be able to deal with the data in a familiar environment (EXCEL, MatLab, SAS, SPSS, R...)
- SAP tries to address this by providing UI tools that are relatively easy to use and that integrate with e.g. MS EXCEL
- AND by reducing the (background) dependencies for query processing
 - no need for aggregates or pre-defined indexes
 - complex built-in functions (fuzzy search, window functions, currency conversion, etc.)

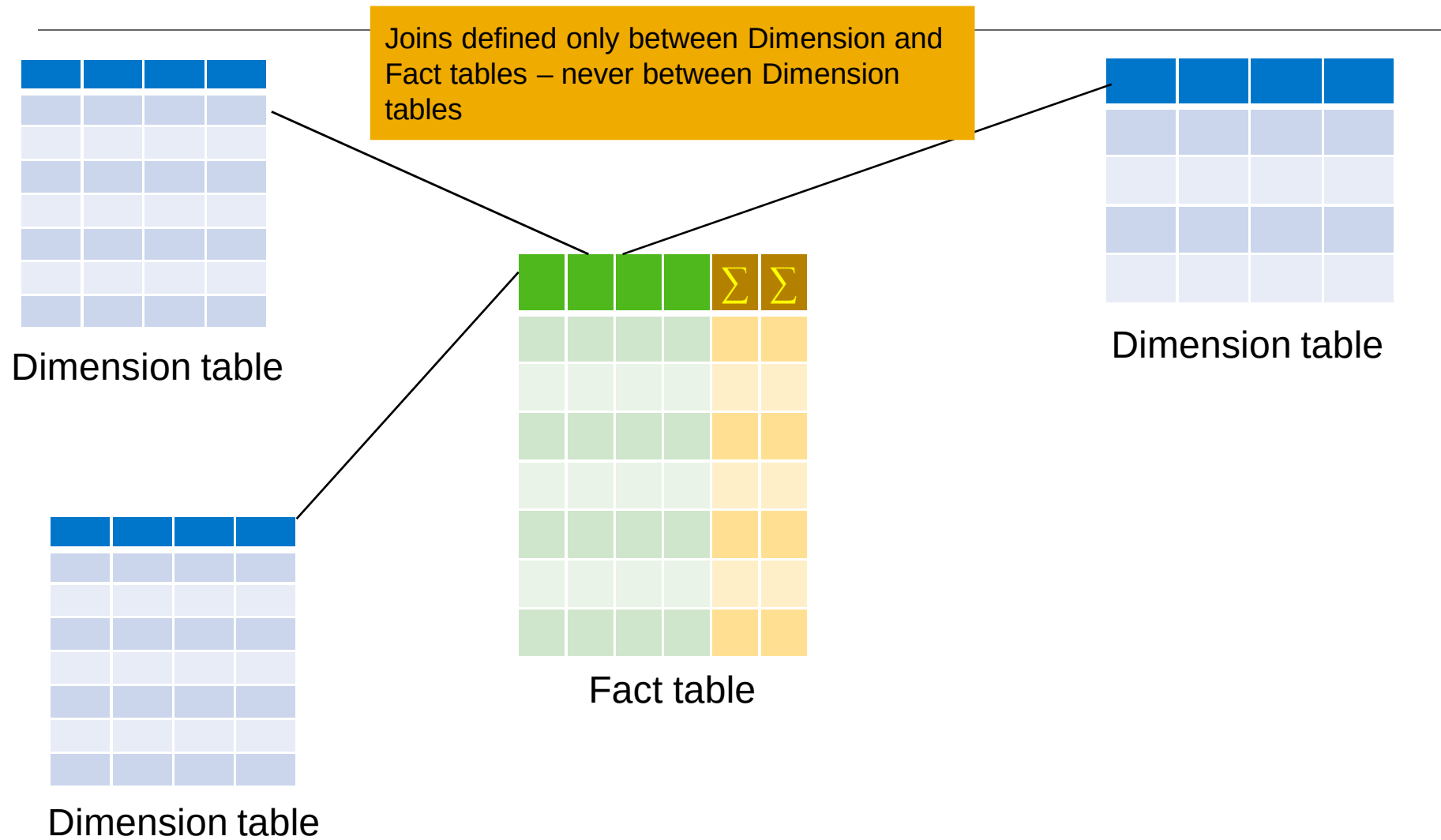
What are Star Queries?

- Still, any DBMS is there to support the user requests as good as possible, so additional features are added to support even this "wrong" kind of usage.
- Nevertheless, this will invariably lead to longer runtimes. (PERIOD)
- However, back to star query!
- What's special about star schema when it comes to query processing?
 - The data that should be filtered is in the central facts table
 - The facts table contains the vast bulk of data to be processed (line item dimensions are a famous exception to this)
 - The actual filters are applied indirectly by filtering on the dimension tables and joining those to the facts table
 - There are only join transitions between the facts table and the dimension tables
 - There are NO join transitions between the dimension tables...

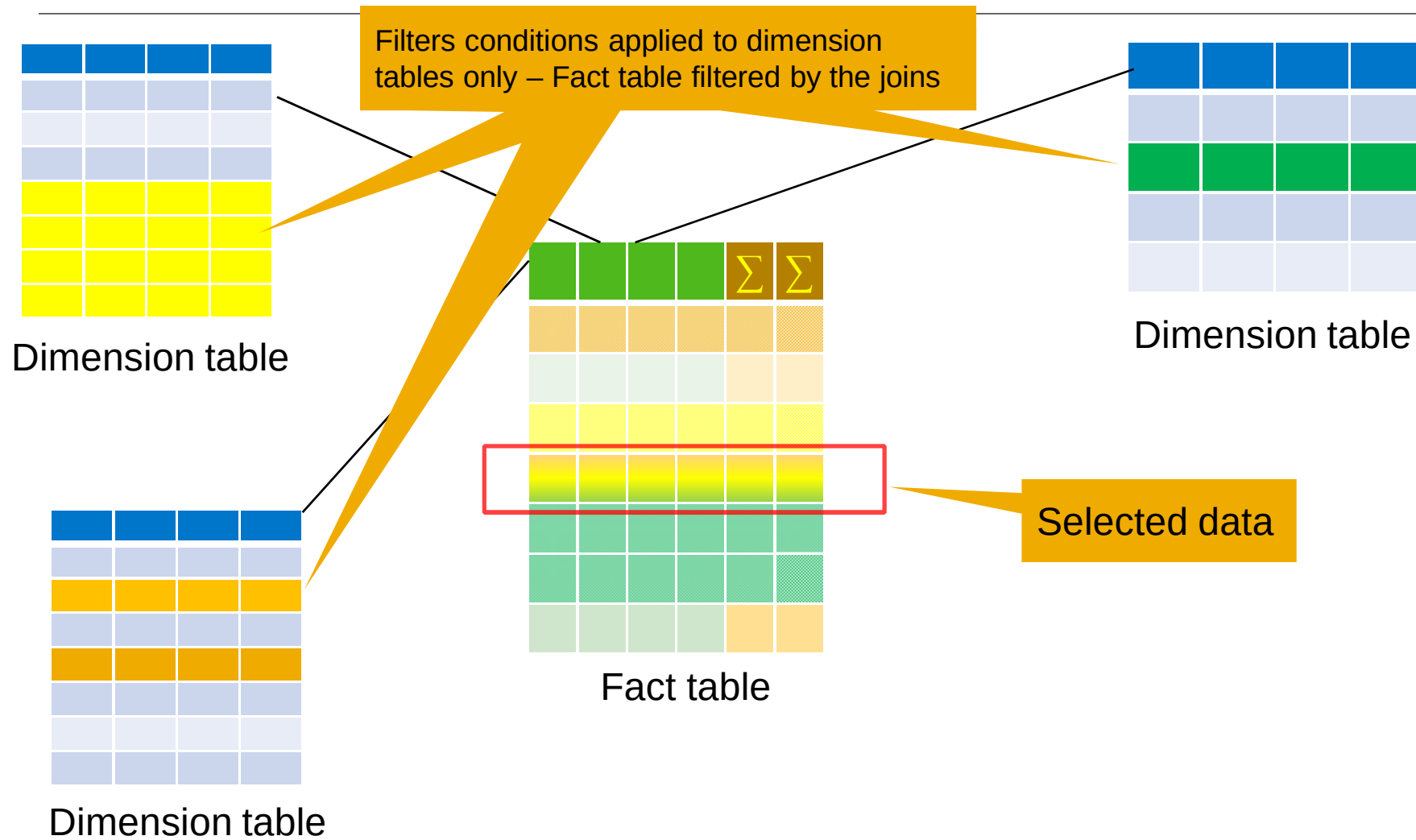
Star Schema



Star Schema



Filtering in Star Schema



What are Star Queries?

- What does that mean for the available join order permutations?
- CORRECT: fact table is either first or second!
- So, with classic join processing, the best choice is to find the dimension-fact join that reduces most of the facts table data
- Not too difficult (except for the estimation of join cardinalities), but is this sufficient?
- Wouldn't we be better off to collect all filters first and read the data from the facts table then?
- Bet we would!

Star Query optimizations

- Oracle's approach: STAR TRANSFORMATION
 - 1. Bitmap indices (bit-vectors that can be combined with logic operators AND/OR/XOR)
 - 2. Multi-Step-Processing
 - Get values to be filtered from dimension tables
 - Apply filters to the facts table via bitmap indices
 - Aggregate and collect data from facts table
 - Read required data from the dimension tables (join back)
- That's AWESOME!
- Other DBMS have similar optimizations:
 - SAP MaxDB: Clustered tables + Star query optimization
 - IBM DB2/6: Clustered indexes (multidimensional indexes, combining filter conditions)
 - MS SQL Server: parallel reading of tables

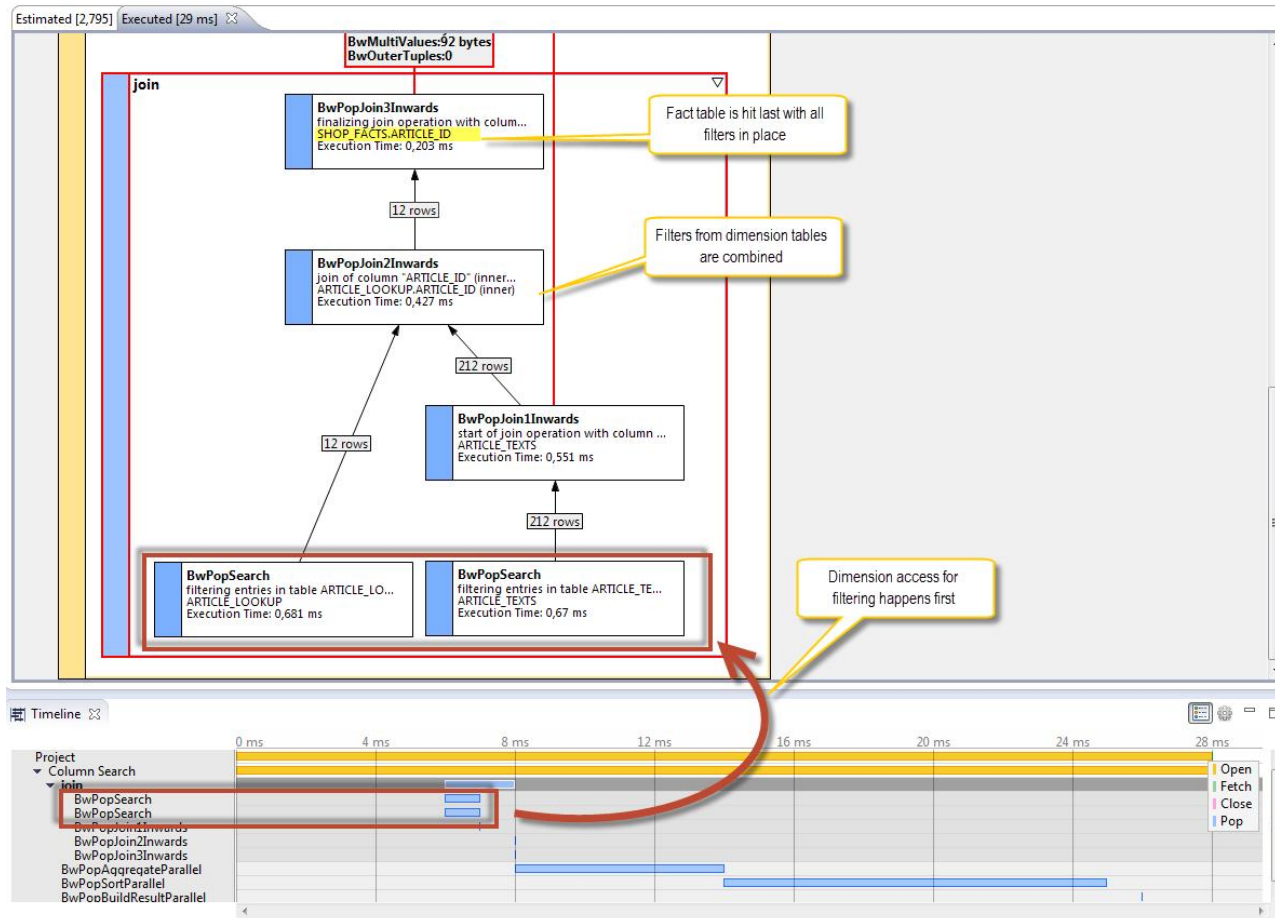
Star Query optimization

- Downside: the DBMS must somehow **know** that the query is a STAR QUERY
- Can be done by inferring (could fail)
- Or by hints (could be wrong -> remember maintenance hell?,
 - although: intentional hints may still be OK to have...
 - BW on Oracle suffered when STAR TRANSFORMATION + FACT hints were removed)
- Alternative ?

SAP HANA approach to star queries

- Define OLAP cubes on HANA level
- That's **ANALYTIC** Views
 - In Plan Visualisation we find:
 - BWPopJoin...
 - Dimension tables are accessed first, filters are applied to the facts table at once and necessary data from the dimension table is fed to the aggregated result
- remark: the SQL optimizer in recent revisions in SAP HANA is sometimes clever enough to recognize Star Schema queries w/o an analytic view.

Star Query demo – Analytic view



Analytic views – calc. cols – plain SQL

Analytic view without calculated column

Statement 'SELECT TOP 20 "ARTICLE_LABEL_LOCALE", sum("MARGIN") AS "MARGIN", sum("AMOUNT_SOLD") AS ...'

successfully executed in 71 ms 500 µs (server processing time: 26 ms 739 µs)

successfully executed in 69 ms 917 µs (server processing time: 24 ms 280 µs)

successfully executed in 66 ms 854 µs (server processing time: 25 ms 944 µs)

Analytic view with calculated column

Statement 'SELECT TOP 20 "ARTICLE_LABEL_LOCALE", sum("MARGIN") AS "MARGIN", sum("AMOUNT_SOLD") AS ...'

successfully executed in 117 ms 532 µs (server processing time: 73 ms 225 µs)

successfully executed in 98 ms 992 µs (server processing time: 55 ms 764 µs)

successfully executed in 144 ms 241 µs (server processing time: 71 ms 85 µs)

→ easily doubled server processing time - nearly a third more for total runtime.

SQL equivalent

Statement 'SELECT TOP 20 at.article_label as "ARTICLE_LABEL_LOCALE", sum("MARGIN") AS "MARGIN", ...'

successfully executed in 98 ms 712 µs (server processing time: 36 ms 213 µs)

successfully executed in 104 ms 154 µs (server processing time: 39 ms 560 µs)

successfully executed in 104 ms 754 µs (server processing time: 33 ms 687 µs)



Thank you

*We value your opinion. Let us know how you like this CSA presentation and how we can make adopting SAP innovations a more beautiful experience. **Please send questions and comments to csa_feedback@sap.com.***

Contact information:

Lars Breddemann

RIG Expert, Customer Solution Adoption (CSA)
SAP Österreich GmbH, Lassallestraße 7b, 1021 Wien, Österreich
<mailto:lars.breddemann@sap.com>

© 2013 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

National product specifications may vary.

These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries.

Please see <http://www.sap.com/corporate-en/legal/copyright/index.epx#trademark> for additional trademark information and notices.