DEV200 – Core Data Services
Next-Generation Data Definition and Access on SAP HANA
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.
Speakers

**Las Vegas, Oct 19 - 23**
Kai Baumgarten – Senior Developer
Philipp Degler - Developer

**Barcelona, Nov 10 - 12**
Kai Baumgarten – Senior Developer
Kilian Kilger – Developer
Core Data Services (CDS)
Overview and Concepts

SAP TECHED
The SAP HANA Platform - More than just another database

Key technology aspects of SAP HANA

- In-Memory computing
  Allowing OLTP and OLAP in real-time

- Row and column based data store
  Very fast aggregation and search

- High data compression
  Make use of real-life / sparse fill of tables

- Support for multi-core architecture
  Benefit from massive parallelization

Key impacts on ABAP apps for SAP HANA

- No aggregates
  On-the-fly data models without duplicates

- Less indices
  Flexible and fast retrieval of the dataset

- Less code lines
  Less complexity in data models and code

- Mass data analysis
  Partitioning and data distribution
Paradigm Changes in Application Programming

Data-To-Code: Intensive computations in APPLICATION layer

Application Programming Models

Code-To-Data: Intensive computations in DATABASE layer

Database

<code>

Classic Approach

Data Centric Approach
**Core Data Services (CDS): Building Block for Timeless Software**

<table>
<thead>
<tr>
<th>Clients</th>
<th>Next generation data definition and access for database centric applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI Abstraction</td>
<td>Cross-platform unified abstraction layer – similar to OData for UI abstraction</td>
</tr>
<tr>
<td>Programming Models</td>
<td>Maximum transparency for different programing models</td>
</tr>
<tr>
<td>Core Data Services</td>
<td>Integrates with platform lifecycle-management</td>
</tr>
<tr>
<td>Database</td>
<td></td>
</tr>
</tbody>
</table>
Availability of CDS in SAP Platforms

- SAP NetWeaver 7.4 SP05
- SAP HANA SPS6
- SAP Business Suite EHP7 (Suite on HANA)
- S/4HANA
- SAP Business Warehouse 7.3
Example of ABAP Core Data Services Usage at SAP

Graph Representation of a Data Model using the Association as Edges

---

**S/4Hana Development System**[1]

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of CDS Views</td>
<td>6,359</td>
</tr>
<tr>
<td>Number of Associations</td>
<td>8,005</td>
</tr>
<tr>
<td>Lines of Code in CDS Views</td>
<td>316,603</td>
</tr>
</tbody>
</table>

[1] on 2015-07-22
Motivation for Core Data Services

Can’t everything be done with SQL

In principle yes, but …
Issue 1: Large Semantic Gap between Task and SQL Code Today

Task: Get the id, name and the respective zip code of the home address for all employees in org-unit 4711

SQL: SELECT e.id, e.name, a.zipCode FROM Employee e
    LEFT OUTER JOIN Employee2Address e2a ON e2a.employee = e.id
    LEFT OUTER JOIN Address a ON e2a.address = a.id
    AND a .type='homeAddr'
WHERE orgunit=4711.
Issue 2: SQL Complexity Leads to Imperative Code

Issue: Due to the complexity of SQL, developers are using code instead of SQL.

Performance gaps: Loops in loops, nested queries with many round trips, ...

Code:

```sql
SELECT * FROM Employee INTO TABLE it_empl WHERE orgunit = 4711.
LOOP AT it_empl.
  WRITE it_empl-id.
  WRITE it_empl-name.
  SELECT * FROM Addresse INTO TABLE it_addrs WHERE id = employees-id.
  LOOP AT it_addrs.
    IF it_addrs-type = 'HOMEADD'.
      WRITE it_addrs-zipcode.
    ENDIF.
  ENDLOOP.
ENDLOOP.
```
CDS Brings Conceptual and Implementation Level Closer Together

**Task:** Get the id, name and the respective zip code of the home address for all employees in org-unit 4711

Small Semantic Gap, No Coding required

**CDS:**

```
SELECT id, name, homeAddress.zipCode FROM Employee
WHERE orgunit=4711.
```
CDS ABAP Integration

Leverage DDIC semantics

Infrastructure ensures Any DB support

Consistent Lifecycle Management and extensibility as with all other ABAP artifacts

Highly reusable and extensible CDS artifacts

CDS also available in native SAP HANA
(see: DEV 160 - Encounter Core Data Services with SAP HANA)
Core Data Services Details
Core Data Services – A Family of Domain Specific Languages

**DDL**
Data Definition Language
Data modelling and retrieval on a higher semantic level
Extends native SQL means for higher productivity

**QL**
Query Language
Consume CDS entities via Open SQL in ABAP
Fully transparent SQL extensions

**DCL**
Data Control Language
Define authorizations for CDS views
Modelled and declarative approach
Integrates with classic authorization concepts
CDS Development Environment

Based on Eclipse platform and therefore integrated in ABAP in Eclipse

Textual and graphical editor

Rich feature set for fast development
Code completion
Enhanced data preview
Quick-Fix function
Syntax highlighting
Code Push-Down Scenarios with CDS

1. **Built-in SQL-functions / SQL-expressions**
   Push non trivial logic to *any SAP supported DB* via CDS*

2. **CDS Table Functions**
   Use natively implemented SAP HANA-DB functions from CDS

* Might or might not change in the future
CDS Built-in (SQL) Expressions
Types of CDS Built-in Expressions

- Date and Time Functions
- Generic SQL Expressions
- Arithmetic Functions
- Conversion Functions
- String Functions
# CDS Built-in Expressions Overview

<table>
<thead>
<tr>
<th>Generic SQL Expressions</th>
<th>Conversion Functions</th>
<th>String Functions</th>
<th>Arithmetic Functions</th>
<th>Date and Time Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple- / Searched-Case Coalesce</td>
<td>Unit Conversion Currency Conversion Cast Hex2Bin Bin2Hex Decimal Shift</td>
<td>Concat Instr Left / Right Length Lpad / Rpad Ltrim / Rtrim Replace</td>
<td>Abs Ceil Div (Integer) Division (Dec) Floor Mod Round</td>
<td>Days between Add Days Add Month Current Tmstp Date is Valid Tmstp is Valid Seconds btw. Tmstps Add seconds</td>
</tr>
</tbody>
</table>
Example: Currency Conversion

View with parameters

```abap
define view zddl_currconv
  with parameters
    target_currency: abap.cuky,
    @Environment: systemField: #SYSTEM_DATE
    exchange_rate_date: abap.dats
as select from SEPM_I_SalesOrderItem as Item
{
  currency_conversion(
    amount => Item.GrossAmountInTransacCurrency,
    source_currency => Item.TransactionCurrency,
    target_currency => $parameters.target_currency,
    exchange_rate_date => $parameters.exchange_rate_date
  ) as GrossAmountInTargetCurrency
}
```

Default parameter for exchange date

Currency Conversion

Named Parameters
CDS Built-in Expressions Demo
Code Push-Down with Table Functions
What do CDS Table Functions do?

CDS Table Functions allow to use natively implemented database-functions in SAP HANA DB directly from CDS.
Why do we need Table Functions?

Some scenarios require selective measure

Highest performance requirements e.g. with complex calculations
Use of database / analytical engine specific functions required
Open SQL and CDS views are not sufficient to solve problem

Often referenced as ‘Breakout Scenarios’

But!

Database specific
Currently SAP HANA DB only
No automatic fallback for Any-DB
What do you need?

1. **DDL Source**
   - Contains definition of CDS table function
   - Specifies parameter list, return parameters and a reference to the implementing method

2. **ABAP Managed Database Procedure**
   - Implementation of an ABAP Managed Database Procedure

3. **SAP HANA Database**
   - Database and runtime for CDS Table Functions
   - Runs stored SQL-script procedure generated from AMDP on database tables
Definition of a Table Function

- **Stored in a DDL source**
- **Name of table function**
- **List of input parameters**
- **List of return parameters**
- **Link to implementation**

```abap
@ClientDependent: false
define table function zprimes_cds
  with parameters
    rank : abap.int4
  returns
  {
    prime_number : abap.int4;
    text_representation : abap.string;
  }
.
  implemented by method zcl_primes=>get_prime;
```
Implementation of a Table Function (1)

Stored in a global ABAP class

```abap
CLASS zcl_primes DEFINITION
PUBLIC FINAL CREATE PUBLIC.
PUBLIC SECTION.
INTERFACES:
  if_amdp_marker_hdb.
CLASS-METHODS:
  get_prime FOR TABLE FUNCTION zprimes_cds.
PROTECTED SECTION.
PRIVATE SECTION.
ENDCLASS.
```
Implementation of a Table Function (2)

Implementation is done in a special tagged method

Table functions are read-only

Used ABAP managed objects

Implementation in SQLScript

METHOD get_prime BY DATABASE FUNCTION FOR HDB LANGUAGE SQLSCRIPT OPTIONS READ-ONLY

USING zprimes.

declare offs integer;
offs := rank - 1;
return select * from zprimes
    order by prime_number
    limit 1 offset :offs;

ENDMETHOD.
Usage of a Table Function

Table functions can be used in Open SQL

Table functions work on SAP HANA DB only
Pragma confirms the successful execution of a feature mode check
(See class cl_abap_dbfeatures)

```sql
SELECT SINGLE FROM zprimes_cds( rank = 5 )
FIELDS *
INTO @DATA(prime)
##DB_FEATURE_MODE[AMDP_TABLE_FUNCTION].
```
Client Handling - Definition

Activate client handling

Client input parameter
Optional
Filled automatically by OpenSQL

Client return parameter
Mandatory
Located in first column

@ClientDependent: true
define table function zprimes_clnt_cds
  with parameters
    @Environment.systemField: #CLIENT
    clnt_in : abap.clnt,
    rank    : abap.int4
  returns
  {
    clnt    : abap.clnt;
    prime_number : abap.int4;
    text_representation : abap.string;
  }
  implemented by method
    zcl_primes=>get_prime_clnt;
METHOD get_prime_clnt BY DATABASE FUNCTION FOR HDB LANGUAGE SQLSCRIPT OPTIONS READ-ONLY

USING zprimes_clnt.

```
declare offs integer;
offs := rank - 1;
return select * from zprimes_clnt
    where clnt = :clnt_in
    order by prime_number
    limit 1 offset :offs;
```

ENDMETHOD.

Usage of client input parameter to select the correct values from the database
Associations
SQL Joins as Means for Combining Fields from Two or More Tables

How we work with SQL joins

```sql
define view zddl_join

as select from SEPM_I_SalesOrderItem as I

inner join SEPM_I_Currency as C

on I.TransactionCurrency = C.Currency

{ ... }
```

But SQL Joins …

… are not reusable

… have a difficult syntax

… do not contribute to the data model of an application
Associations as Reusable Relationships

Define Association with Cardinality

Use Association => select a column

Expose Association for reuse

Reuse association => select another column

```sql
define view zddl_assoc
as select from SEPM_I_SalesOrderItem
association [0..1] to SEPM_I_Currency as _Currency
  on $projection.TransactionCurrency = _Currency.Currency
{
  _Currency.CurrencyISOCode,
  _Currency
}
```

```sql
define view zddl_use_assoc
as select from zddl_assoc
{
  _Currency._Text.CurrencyLongName,
}
```
Used (!) associations are implicitly translated into SQL joins

Reuse of generated joins when semantically identical
Filter Expressions

Filter Expression with Cardinality Annotation

```sql
define view zddl_bpa
as select from SEPM_I_BusinessPartner
{
   _Address[1: Country = 'DE'].CityName
}
```

Filters can be used to restrict the result set when using [0..n] or [1..n] associations.

Filters can be used to change the cardinality, e.g. in the WHERE clause only [0..1] associations are allowed.

Same filters can be configured to only generate a single join by using annotations.
Usage in Open SQL

Association with leading backslash

```
SELECT FROM sepm_i_businesspartner
FIELDS \_Address-PostalCode
INTO @DATA(result).
```

Associations can directly be used in Open SQL when using select on a CDS entity

They are translated into joins from the Open SQL compiler
Associations Demo
Extending Views
Hierarchy of Views and Extensions

- **Hierarchical view-on-view concept**
  - Optimized result-sets with minimum data transfer
  - Extensions allow to append additional columns, arithmetic expressions or literals to result set

---

**Basic View**

- **Consumer**
  - C1
  - C2
  - C3

- **View on View**
  - V1
  - V2

- **Extensions**
View Extends

```
extend view SEPM_I_BusinessPartner
    with zddl_bp_extend

{
    _Address.cityname
}
```

Views can be extended to add additional fields

Extend is stored and transported in separate DDL source

Even „union“ and „group by“ clauses can be extended
Extending Views Demo
Annotations
CDS: Common Basis for Domain-Specific Frameworks

Reusable and unified view model for all use cases

Annotations enabling flexible usage in different contexts

Efficient development
Annotations

Some „core“ annotations

```sql
@AbapCatalog.sqlViewAppendName: 'zddl_v_bpa'
@EnduserText.label: 'Business Partner with street'
define view zddl_bpa
as select from SEPM_I_BusinessPartner
{
  @Semantics.address.street: true
  _Address[1: Country = 'DE'].StreetName
}
```

Domain specific annotation

Annotations can be used to further enrich the meta model with consumer-specific vocabularies

Self defined annotations can be used and queried by an annotation API

Own frameworks exist around third party annotations
Annotations Demo
Authorizations with DCL
Data Control Language Overview

Declarative approach instead of coded

Based on CDS modeling objects and therefore part of the data-model

Authorizations are also pushed down to DB by extending the Open SQL SELECT statement
Usage of Declarative DCL Authorizations – Example Scenario

Scenario

Enable READ-access only to Sales Orders that belong to an Org Unit for which the user is authorized (according to the PFCG authorization object)

CDS-View (to be authorized)

define view SalesOrders as select from SalesOrder_Table
{
    SalesOrder as SalesOrderNumber,
    GrossAmount,
    SalesOrg,
    _Items, // This is an association
    _Buyer // This is an association
}
**Classic Authority Checks**

**Coded authority check**

```
SELECT * FROM SalesOrders INTO CORRESPONDING FIELDS OF @wa_SalesOrder.

AUTHORITY-CHECK OBJECT 'S_SALES'
   ID 'ORGUNIT' FIELD @wa_SalesOrder-SalesOrg
   ID 'ACTVT' FIELD '03'.
   IF sy-subrc = 0.
      APPEND wa_SalesOrder TO lt_SalesOrders.
   ENDF.
ENDIF.
ENDSELECT.
```

**Rating**

- Performance is critical
- Every access to the view has to be secured separately
Usage of Declarative DCL Authorizations – DCL Role and Consumption

**DCL Role**

```java
role DCL_SalesOrders {
    grant select on SalesOrders
    where (SalesOrg) = aspect pfcg_auth(S_SALES, ORGUNIT, actvt='03');
}
```

**Open SQL Consumption**

```sql
SELECT * FROM SalesOrders
    INTO CORRESPONDING FIELDS OF TABLE @lt_SalesOrders.
```

**Rating**

- Performance: Only authorized entries are retrieved from the DB. Statement is not executed without authorization
- Authorizations are only defined once and automatically (re-) used everywhere
CDS and SAP Gateway
CDS Integration with Gateway and Service Adaptation Description Language (SADL)

1. Define view in CDS editor or reuse existing CDS views
2. Import CDS view to Gateway Service Builder (SEGW)
3. Activate / Register Gateway Service
4. Consumption

More Information:
DEV106 - The ABAP Programming Model in SAP S/4HANA
Core Data Services Summary
Core Data Services at a Glance

1. Semantically Rich Data-Models
   Domain specific languages (DDL, QL, DCL)
   Declarative, close to conceptual thinking

2. CDS is completely based on SQL
   Any ‘Standard SQL’ features directly available like joins, unions, build-in functions, …

3. Fully Compatible with Any DB
   Generated and managed SQL Views
   Native integration in SAP HANA

4. Common Basis for Domain-Specific Frameworks e.g. UI, Analytics, Odata, BW,…
   @AnalyticsDetails.aggregationBehaviour: SUM

5. Built-in Functions and Code Pushdown
   Table Functions for Breakout Scenarios
   Rich Set of Built-in SQL Functions

6. Extensible
   On model level thru extensions
   On meta-model level thru annotations
Further Information

Related SAP TechEd sessions:

DEV160 - Encounter Core Data Services with SAP HANA
DEV201 - Overview of Development with ABAP for SAP HANA
DEV260 - Building Applications with ABAP Using Code Pushdown to the Database
DEV265 - Building an End-to-End App from SAP HANA to SAP Fiori via ABAP (Hands-On)
DEV301 - Building an End-to-End App from SAP HANA to SAP Fiori via ABAP (Lecture)

SAP Public Web

http://scn.sap.com/community/abap
http://go.sap.com/solution/platform-technology.html

SAP Education and Certification Opportunities

www.sap.com/education

Watch SAP TechEd Online

www.sapteched.com/online
Feedback

Please complete your session evaluation for DEV200.

Thanks for attending this SAP TechEd session.
Thank you

Contact information:

Martin Huvar
Product Management of SAP Technology Platform
martin.huvar@sap.com
No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. Please see http://global12.sap.com/corporate-en/legal/copyright/index.epx for additional trademark information and notices.

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

National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP SE or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP SE or SAP affiliate company 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.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE’s or its affiliated companies’ strategy and possible future developments, products, and/or platform directions and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, which speak only as of their dates, and they should not be relied upon in making purchasing decisions.