Bridging the Gap Between Relational Data and Application-Level Business Objects with Core Data Services (CDS)

Stefan Bäuerle, **Alexander Böhm** SAP SE
September 2017

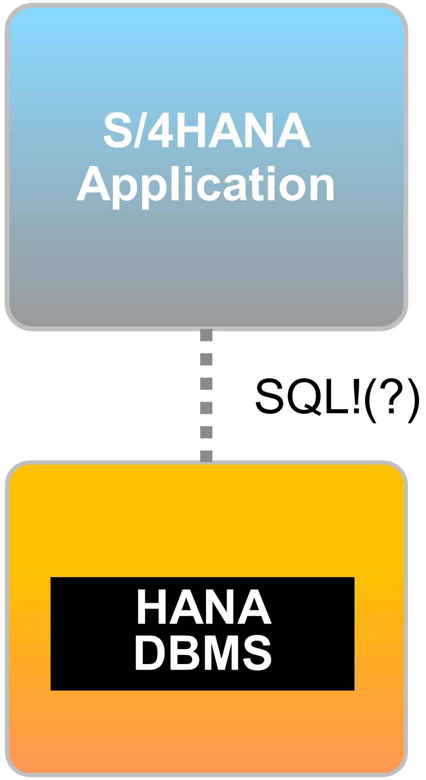
PUBLIC

Agenda

- Hybrid Transactional/Analytic Processing (HTAP)
  - Challenges
  - Motivation for CDS

- CDS as a possible solution
  - Core CDS concepts
    - Types
    - Entities
    - Associations
    - Annotations
  - Results from S/4HANA

HTAP Challenges



HTAP in S/4HANA

- S/4 builds on the classical business suite data model
  - Normalized (usually 3NF or higher)
  - Lots of small tables with configuration data, translated texts, etc.
  - Lots of legacy to deal with:
    - German column names difficult for international customer base (MANDT, BUKRS, …)
    - Heavy use of NVARCHAR for non-textual data (DATE, TIMESTAMP, BOOLEAN, numbers, …)
    - Abuse of fields (e.g. number encoded as NVARCHAR[1], but need 20 values -> “X”)
- Database views are at the core of the data model
  - Translate technical, legacy data to modern representation
    - Meaningful column names
    - Get rid of “hacks”, proper types
  - Enable analytics by creating meaningful business objects (e.g. invoice) from database tables
    - Business objects span multiple database tables
    - Business objects often need to be combined to derive insights

A simple view from DBMS perspective

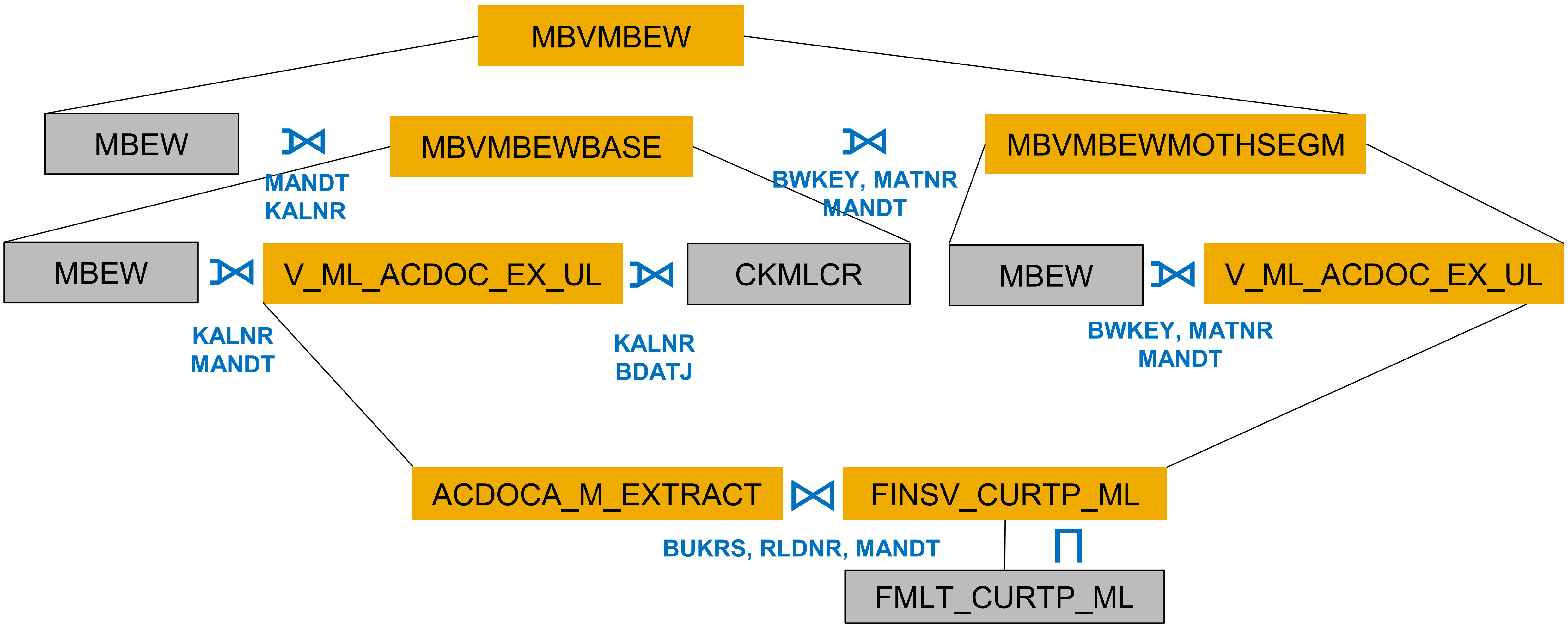
```
CREATE VIEW "SAPQM7"."MBVMBEW"  AS
  SELECT "B"."MANDT", "B"."MATNR", "B"."BWKEY", "B"."BWTAR", "B"."LVORM",
```

Type adjustment

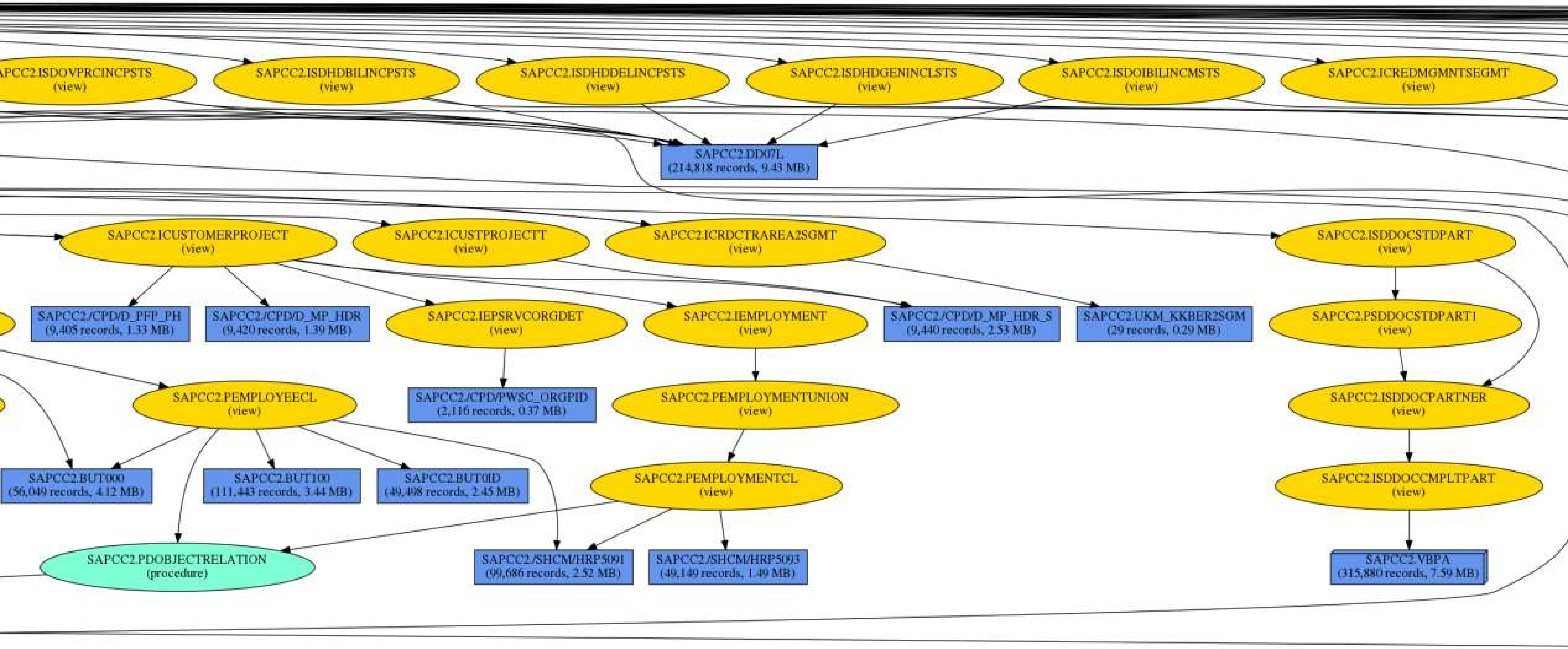
Field selection from input tables based on dependent fields

```
  FROM ( "MBEW" "MBEW" LEFT OUTER MANY TO ONE JOIN "MBVMBEWBASE" "B" ON ( "MBEW"."KALNR" =
"B"."KALNR" AND "MBEW"."MANDT" = "B"."MANDT" AND "MBEW"."MANDT" = "B"."MANDT" ) )
  LEFT OUTER MANY TO ONE JOIN "MBVMBEWMOTHSEGM" "MOTHER" ON ( "MOTHER"."MATNR" = "B"."MATNR" AND
"MOTHER"."BWKEY" = "B"."BWKEY" AND "MOTHER"."MANDT" = "B"."MANDT" AND "MBEW"."MANDT" =
"MOTHER"."MANDT" ) WITH READ ONLY
```

Reverse-engineering MBVMBEW



Adding complex analytics

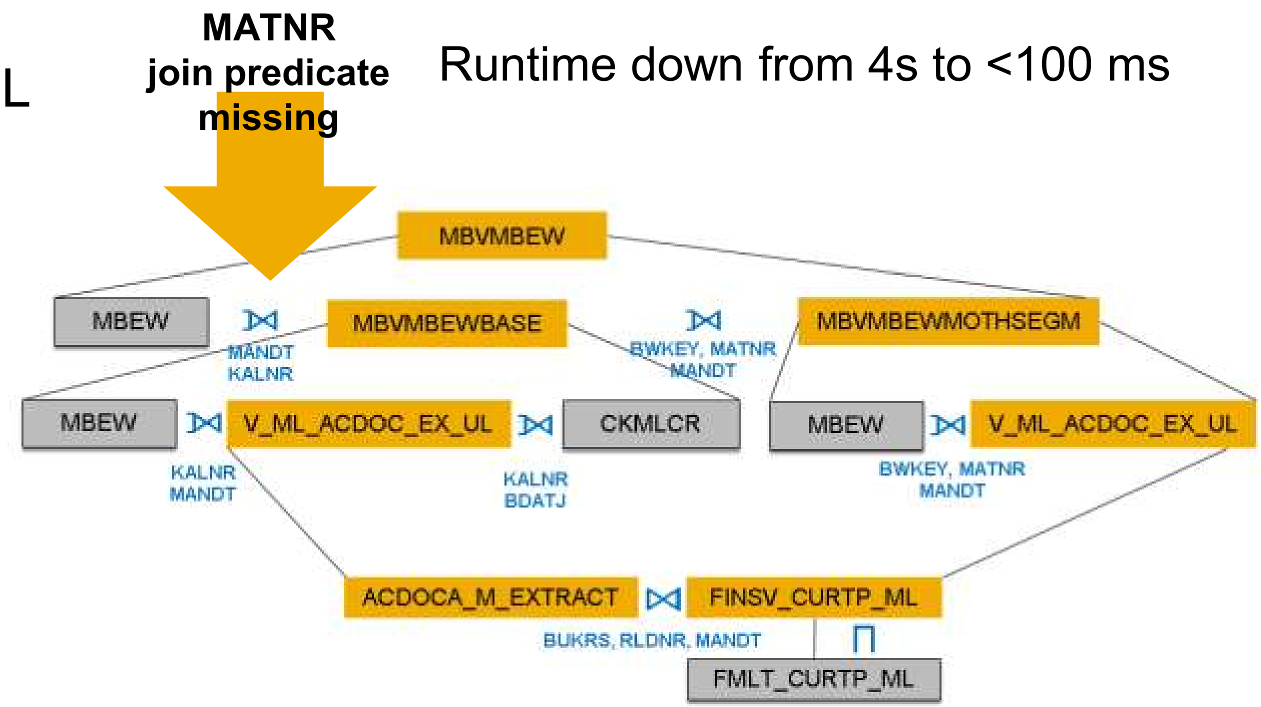


Core Data Services (CDS)

Motivation

- Application Developers are skilled domain experts
- Imperative programming languages but not SQL
  - Writing SQL is hard
  - Often no clue about relational algebra
  - Not familiar with DBMS-level optimization
  - Danger to miss important details
  - Example: MBVMBEW (again)
- Necessity for simpler DBMS interface

Runtime down from 4s to <100 ms



Core Data Services

- Pull data modeling, retrieval, and processing to a semantic level close to the domain experts

CDS = SQL *(+ careful extensions)*

- Key concepts
  - **Entities** with structured types (instead of flat tables)
  - Custom-defined/Semantic **Types** (instead of primitive types)
  - **Associations** for foreign key relations with cardinalities and simple path filter expressions
  - **Annotations** to enrich the data models with additional metadata – e.g. for Analytics

CDS Concepts: Data Types

- Supported Types:
  - „built-in“ Primitive Types (like String, Integer, DecimalFloat, Date)
  - Custom-defined Simple- and Structure Types
- Examples:

```
type Derived : String(111);

type AddressType : String(7) enum {
      home; business = ‘biz’;
}

type Structured {
      descr : Derived;   // reusing a custom-defined type
      amount : Decimal(10,2);
      grossAmount : Decimal(10,2) = amount * (1.00 + taxrate()); // calculated element
      kind : AddressType default home;
}
```

CDS Concepts: Entities

- Entities
  - Define the persistence layer of an application
  - Structured types with an underlying persistency and a uniquely identifying key
  - Entity is defined like a structured type, just with a leading keyword entity instead of type

- Examples:

```
type Amount {
      value : Decimal(10,2);
      currency : String(3);
}

entity Address {
      key streetAddress : String(77);
      key zipCode : String(11);
      city : String(44);
}

entity Employee {
      key ID : UUID;
      name : String(77);
      salary : Amount;                    // Amount is a structured type
      addresses : Association to Address[0..*] via entity Employee2Address;
}
```

CDS Concepts: Associations

- Associations define relationships between entities
  - Which key to use
  - Additional filter conditions (up to a complete join condition)
  - Information regarding cardinality
- Examples:

```
entity Address {
    owner : Association to Employee; // can be used for :m associations
    streetAddress; zipCode; city; // snipped type defs
    kind : enum { home, business };
}

entity Employee {
    addresses : Association[0..*] to Address via backlink owner;
    homeAddress = addresses[kind=home]; // → using XPath-like filter.
}
```

CDS Concepts: Annotations

- Domain-specific annotations to enrich/extend objects without changing the core model
- Example:

CDS view example

```
@EndUserText.label: 'Financial Statement
sFIN'
@Analytics: { dataCategory: #FACT }

define view WSFinancialStatementQuery as
select
  from WSFinancialStatement
  {
    key ChartOfAccounts,
    key GLAccount,
    …,
    @Semantics.currencyCode: true
    key CompanyCodeCurrency,
    @Semantics.amount.currencyCode:
'CompanyCodeCurrency'
    @DefaultAggregation: #SUM
    @EndUserText.label: 'Amount In
                         Company Code
Currency'
    AmountInCompanyCodeCurrency,   …  }
```

@EndUserText.label = <label> label for visualization/UI

@Analytics: { dataCategory: #FACT } fact table for BI tools

@Semantics.currencyCode element is a currency code
@Semantics.amount.currency: indicates where to find the currency

@DefaultAggregation: #SUM default aggregation behavior for BI tools (other options are AVG, MIN, MAX, …)

Example: Data Model in CDS DDL and Mapping to Standard SQL

- CDS DDL

```
type Name {
  first : String(30);
  last : String(77);
};
entity Person {
  key ID : Integer;
  name : Name;
  job : Association to Job;
};
entity Job {
  key ID : Integer;
  description : String(255);
};
```

- Standard SQL

```
CREATE COLUMN TABLE "Person" (
  "ID" INTEGER CS_INT NOT NULL ,
  "name.first" NVARCHAR(30),
  "name.last" NVARCHAR(77),
  "job.ID" INTEGER CS_INT,
  PRIMARY KEY ("ID"));

CREATE COLUMN TABLE "Job" (
  "ID" INTEGER CS_INT NOT NULL ,
  "description" NVARCHAR(255),
  PRIMARY KEY ("ID"));
```

Example: Queries (QL)

- Superset of standard SQL (SQL + QL extensions)
- QL extensions to leverage enhancements provided by the data models
- Examples: (structured Types (salary.value) & Associations (orgunit.costcenter))

```
SELECT name, salary.value, orgunit.costcenter FROM Employee;
```

Which would be equivalent to the following standard SQL statement:

```
SELECT e.name, e."salary.value", ou.costcenter FROM Employee e
JOIN OrgUnit ou ON e.orgunit_ID = ou.ID;
```

More Examples

```
SELECT ... from Emloyee WHERE orgunit='4711';
SELECT ... from Emloyee WHERE homeAddress.zipCode='76149'
   AND homeAddress.streetAddress='Vermontring 2';

SELECT ... from Emloyee WHERE address[kind=home].city = 'Walldorf';
SELECT ... from Emloyee WHERE homeAddress = addresses[kind=home];

SELECT FROM Employee {
     name,
     addresses[kind=home].city AS homeTown,
     addresses[kind=business].city AS businessTown
}

SELECT DISTINCT FROM OrgUnit[boardarea='TIP'] .employees[salary>'$100.000'] {
     addresses [kind=home].city, count(*)
}
```

Example: Queries using QL enhancements and Standard SQL

- Retrieving a list of all ordered materials per companies

- CDS QL (heavy use of associations)

```
SELECT FROM BSEG {bkpf.mandt,
bukrs.butxt, mara.matxt, SUM((menge))
AS menge2 }
WHERE bkpf.txkrs <> 0 AND menge > 0
GROUP BY bkpf.mandt, bukrs.butxt,
mara.matxt;
```

- Standard SQL

```
SELECT BKPF.MANDT, T001.BUTXT, MARA.MATXT,
        SUM(BSEG.MENGE)
FROM BKPF
JOIN BSEG ON BKPF.MANDT = BSEG.MANDT
        AND BKPF.BUKRS = BSEG.BUKRS
        AND BKPF.BELNR = BSEG.BELNR
        AND BKPF.GJAHR = BSEG.GJAHR
JOIN MARA ON BSEG.MANDT = MARA.MANDT
        AND BSEG.MATNR = MARA.MATNR
JOIN T001 ON BSEG.MANDT = T001.MANDT
        AND BSEG.BUKRS = T001.BUKRS
WHERE BSEG.MENGE > 0 AND BKPF.TXKRS <>0
GROUP BY BKPF.MANDT, T001.BUTXT,MARA.MATXT
```

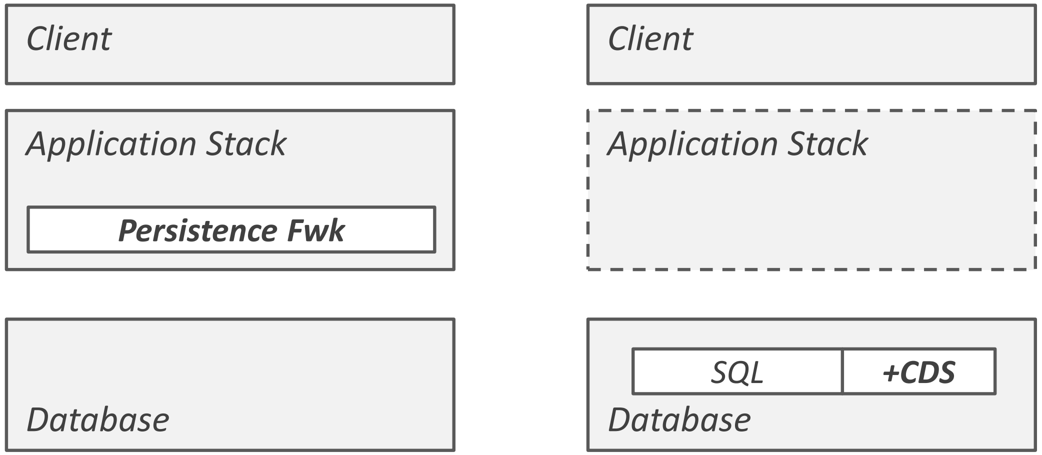
What are others doing?

- Many platforms ease the use of SQL through some kind of persistence framework
  - Microsoft's Entity Framework (EF) and LinQ
  - Apple's Core Data (CD)
  - Force.com's SOQL
  - JPA / Hibernate in JEE
  - Active Records (AR) in Ruby on Rails
- As CDS, most borrow and combine concepts from Entity-Relationship Modeling, XPath, …

CDS	Also supported in…
Parameterized Views	HANA, SQL Server, … via table functions
Annotations of Data Models	EF, JPA, OData
Associations	*– all –, OData*
Path Expressions + Infix Filters	*– all –, OData, XPath*
Calculated Attributes	EF, CD, JPA
Custom-defined & Struct.Types	*– all –, SQL:1999*
Structured Result Sets	*(SQL:1999 impls)*
Intrinsic Extensibility	SOQL
Predicated Privileges	JPA, …, Sybase ASE

Where are the Differences to other Approaches?

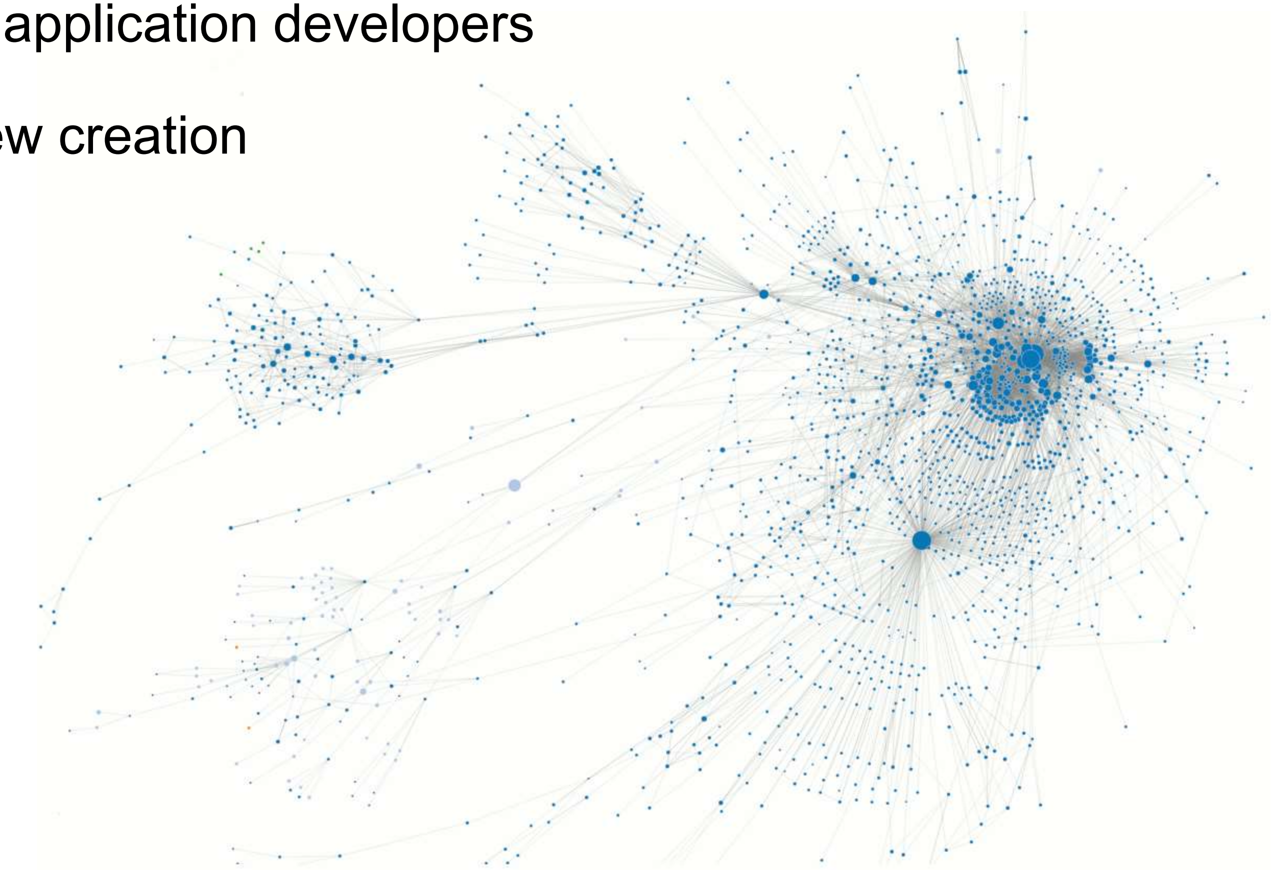
- Other frameworks are bound to a particular application stack
- CDS is designed as an extension to SQL, independent of the application stack
- CDS stays in the relational model, instead of hiding it behind object-relational mappers
  - Preserves advantages such as declarative/functional approach



Object-Relational	Relational / SQL
Entity Framework, LinQ	
Core Data	SQL + CDS
JPA / Hibernate	Force.com SOQL
Active Records	

Summary

- HTAP creates data model / query challenges
- Need to increase productivity / ease of use for application developers
- CDS can be used to simplify modelling and view creation
- Results so far in S/4HANA application stack
  - Over 20.000 CDS views
  - Over 2,1 mio associations
  - Average complexity:
    - 12 tables reference (max: 1593)
    - 3 levels of view stacks (max: 31)
    - 96 associations (max: 30.236)



Thank you.

Contact information:

Dr. Alexander Böhm

alexander.boehm@sap.com