An Overview of SAP Core Data Services

Add Belati
Corporate Application Department
Saudi Aramco
Dhahran, Saudi Arabia
email: add.belati@aramco.com

Firas Alomari
Corporate Application Department
Saudi Aramco
Dhahran, Saudi Arabia
email: firas.alomari@aramco.com

***Abstract*—Increasing amount of data and the diversity of available data structures enable applications that can make timely decisions based on live data that, in most cases, can be evaluated without traditional application layer processing. SAP introduced the Core Data Service (CDS) framework as a data modeling approach in which Virtual Data Models (VDM) are defined at the database layer. CDS improve applications performance by pushing data processing from the application to the database layer to reduce data movements. In this paper, we present some technical insights into SAP's CDS, including new application patterns supported by CDS and the motivations behind it. We also discuss some practical considerations and challenges that may arise with CDS adoption and implementation.**

***Index Terms*—ERP; Programming; Database; Code Pushdown**

I. Introduction

Enterprise Resource Planning (ERP) and Business Intelligence (BI) systems are a fundamental part of today's enterprise IT applications portfolio. They provide a set of standardized software packages that capture interdisciplinary business processes across the entire value chain of an enterprise in a streamlined fashion [1]. ERP and BI systems integrate business functions with a centralized data repository shared by all business processes in the enterprise. The information provided by these systems drive daily business operations and provides for the development of new business ideas.

ERP systems were designed to capture daily operational and transactional business data. This kind of data processing typically referred to as Online Transactional Processing (OLTP) [2], uses row-based operations to efficiently process transactions, instantly recording business events (e.g., payments) and reflecting changes as they occur. However, they are not efficient at performing set-wide operations on entire tables. BI or Data Warehouse systems, on the other hand, were designed as Online Analytical Processing (OLAP) systems, to provide analytical and trend reporting from the growing data in the ERP systems. They leverage column-oriented tables to speed up operations over a huge volume of data at the expense of efficiency in executing OLTP workload.

OLTP and OLAP systems evolved separately to prevent the long-running and resource-intensive OLAP workload from decreasing the transactional throughput of the OLTP system [3]. Specifically, OLAP solutions were running analytical queries on a copy of the transactional data (i.e., views) from OLTP data stores [4]. This enabled companies to efficiently address a growing number of conflicting business needs, albeit, at the expense of increased complexity to link, orchestrate and synchronize multiple systems in the IT infrastructure. Therefore, unified Analytical Transaction Processing (ATP) systems were proposed to perform fast analytical processing coupled with transactional data management [4], preferably, by merging operational and analytical systems into one single system. ATP systems run the analytical queries directly on top of the transactional data to enable real-time data operations, reduce IT complexity and lower the total cost of ownership.

Systems, Applications and Products (SAP) introduced the High-Performance Analytic Appliance (HANA) [5] to provide a unified ATP system that fulfils the aforementioned requirements of business applications. It provides a data management platform to support efficient processing of both transactional and analytical workloads on the same physical database. It supports a code pushdown approach to execute data-intensive processing in the database close to the raw data (i.e., code-to-data) to reduce expensive data movement and improve applications performance. To take advantage of this code pushdown concept, SAP introduced CDS framework as a data modeling infrastructure to enable data reusability, extensions, and integration in HANA. Later, CDS were also introduced to the SAP ABAP application stack to enable developers to take advantage of the code pushdown with other databases.

In this paper, we introduce CDS and the motivation behind it. We also discuss some of CDS implications on application development strategy. The paper is organized as follows: CDS are described in Section II. In Section III we discuss CDS advantages and present some practical considerations related to code-push-down with CDS. We conclude the paper and present future research directions in Section IV.

II. SAP Core Data Services (CDS)

In this section we present a brief background of code pushdown and introduce the data services layer provided by CDS. Additionally, we describe CDS technical features and enhancements.

A. Background

The availability of multi-model databases that support complex data structures such as spatial, text, graph, and time series data enables built-in advanced analytics capabilities, such as

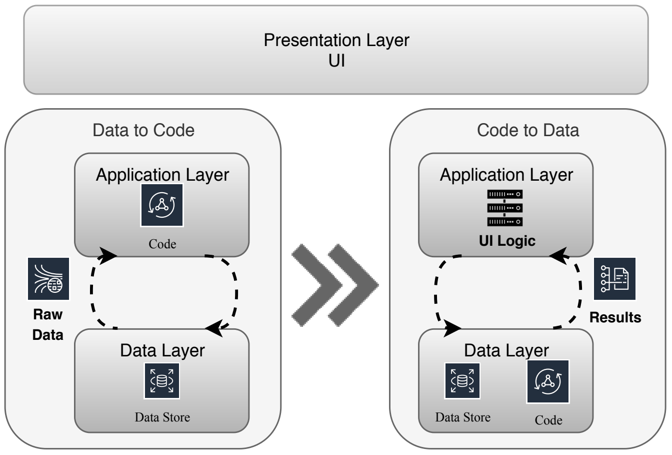


Fig. 1. Code Pushdown Model.

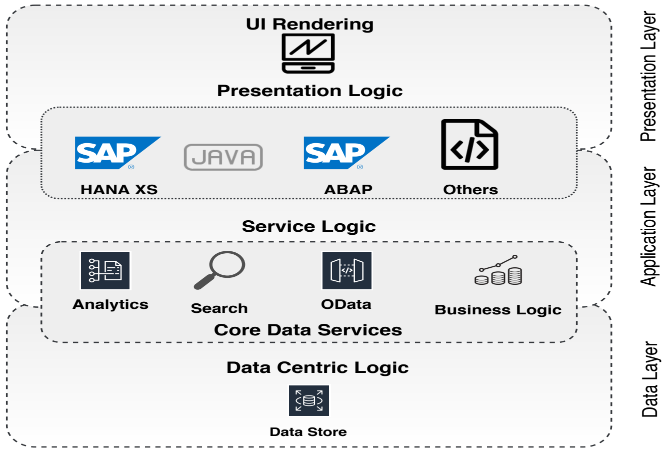


Fig. 2. Core Data Services Model.

text mining, spatial analysis and predictive analytics.These capabilities support diverse application patterns ranging from transactional systems to decision and analytic support systems in real-time and on live data [6]. However, as the size of data grows, moving data becomes the bottleneck and the cost of moving data around becomes prohibitive. Yet, in many cases, the data can be evaluated and processed on the data storage without the need for traditional application layer processing.

Let us consider the example of one million values, representing different currencies, that need to be calculated. Typically, the individual values are transferred to the application server from the database to only be discarded after the conversion and calculation has been done. Therefore, it's advantageous to move the data less by running workloads in the data storage or as close to it as possible [7]. This concept of code pushdown or code-to-data (see Figure 1) improves the applications performance by executing application logic inside the database, thus, decreasing the amount of data transfer and round-trips between the database and application layers.

B. Data Services Layer

Technically, applications communicate with databases through a number of various interfaces such as Standard Query Languages (SQL), Stored Procedures, SQL Scripts, or specific APIs. In the code pushdown concept, data services such as search or predictive analytics use these interfaces to push the computation to the database layer and only move results to application layer. Therefore, a common abstraction layer for integrating database interfaces and data services together is necessary. To this end, SAP introduced a data service layer (i.e., CDS) as a common abstraction layer for integrating these interfaces and data services together. This layer provides data models for defining and formatting data sources consistently across multiple systems, enabling different applications to share the same data, thus reducing development costs and time as well as improving the quality and performance of the applications.

The CDS data modeling infrastructure uses specific domain language to define different data services in a unified data definition and query language [8]. These data models are defined and consumed on the database server rather than on the application layer. They can be further enriched with semantics to allow developers to define entity types (e.g., orders or products) and the semantic relationships between them, which correspond to key relationships in traditional entity relationship (ER) models.

In particular, CDS offer central definitions that can be used in many different application context, such as transactional and analytical applications, to interact with data in the database in a unified way. They provide a cross platform unified data abstraction layer that sits between the database and client applications (see Figure 2). They function as proxy that drives data intensive computation to the database and exposes only relevant results to be consumed by different applications. Similar to Open Data Protocol (OData) for User Interface (UI) abstraction where UI rendering is pushed up to the client, CDS push down data-intensive calculations to the data layer to get the benefits of the underlying databases high-performance capabilities such as fast in-memory column operations, query optimization, and parallel execution.

C. CDS Description

CDS models are expressed in a Data Definition Language (DDL). DLL is based on standard SQL with some enhancements, such as associations, extensions, and annotations. The CDS Query Language (QL) is an extension to SQL used to consume CDS data. The QL includes enhancements such as defining views within the CDS data model. It also introduces the use of associations defined by the DDL. Instance-based authorization to CDS entities are defined using the Data Control Language (DCL). DCL can leverage literal conditions that compare elements of a CDS entity with literal values such as organization code, thus, pushing granular authorization filters and restrictions to the database for better performance.

They can also integrate with traditional authorization concept in SAP to check against existing authorization objects.

There are two variants of CDS: ***ABAP*** and ***HANA***. ***HANA CDS*** are HANA database dependent entities residing on the database itself. They enable the creation of database tables, views and data types using the native HANA DB SQL statements, enriching them with semantical properties, and using HANA native functions to perform data intensive computations. The HANA CDS does not require a specific application stack and therefore can support a variety of technologies and programming languages.

On the other hand, ***ABAP CDS*** provide a framework for defining and consuming semantic data models on the central database of the ABAP application stack. One can use SQL like statements to create and deploy the corresponding CDS entity on the target database automatically. Simillar to HANA CDS the models are based on the DDL and DCL, which are managed by ABAP Dictionary. However, unlike HANA CDS, ABAP CDS are database independent.

Virtually, design principles for both ABAP and HANA CDS are the same but due to differences in the respective environments, some technical differences between these flavors evolved. One clear distinction is that ABAP CDS access control and authorization can support traditional ABAP-based authorizations or defined in the DCL of the CDS entities. Both methods can be used independently or together, however the traditional authorization concept allows for the reuse of existing authorizations in the ABAP system.

Technically, CDS are an enhancement of the standard SQL that provides a data modeling framework for developers to define CDS entities, such as tables, views, and user defined data structures in the database. The CDS entities capture the semantics of the data to join the data needed for the application into one single model. There are two types of CDS entities: ***views*** and ***table functions***.

The CDS ***views*** are defined for existing tables or views in the database. They can be used to rearrange table fields based on the application needs. The views can have additional input parameters to filter the data during selection process at database level itself. So there is no need for a where condition in the application layer code.

Alternatively, CDS ***table function*** views include computations for database tables that are used by other CDS views, such as date and time calculation and conversion functions. Similar to CDS views, table functions can have additional parameters. Both type of views provide a number of major capabilities and enhancements over standard SQL, such as:

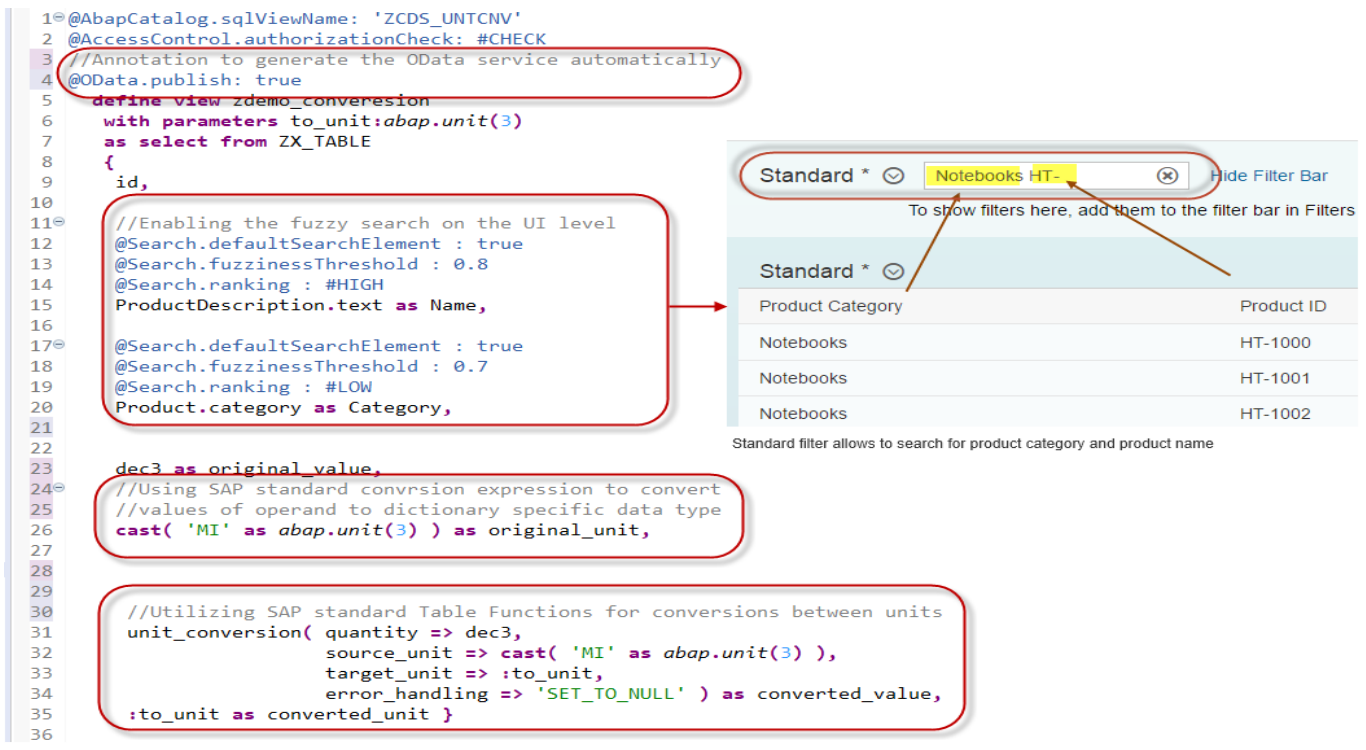
- *Joins*: are used to group fields from one or more different tables or views. Joins such as INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, or UNION are supported.
- *Associations*: are joins on demand that get executed when specifically used in a query. They are reusable, and allow a CDS view to be linked with other data sources, such as classical tables and views or another CDS entity, with a varying degree of cardinality. Cardinality and simple path expression in queries are the most important benefits of associations views.
- *Annotations*: define properties and behavior at run time. For example, one can save views or link another column during run time using the "@" character in the CDS view definitions. It further allows parameters to be used or meta data to be added to the CDS view data.
- *Expressions*: are used for calculations such as aggregation and mathematical computation in the data model queries.
- *Extensions*: are views with additional information that is not in the original table. They reduce data movement by adding expressions, associations and additional columns to an existing view.

D. Example

We show two examples of using CDS in Figure 3. The CDS are used to build a data model and service definitions on a conceptual level. Specifically, CDS models are translated into native database artifacts (e.g., schema) and interpreted to services to be exposed to applications. Specifically, at line 4 the OData annotation generates OData service automatically. The OData is then consumed by the application to enable the fuzzy search shown in the example. Further, at line 11 the example shows additional annotations that are used to enable other visual elements such as default values and enabled UI controls. In lines 23, the example also shows the currency conversion scenario mentioned in Section II-A using CDS table function. Besides the performance enhancement with this approach, the data abstraction layer makes it easier to define semantically rich data models. These models can be used for easy data access and allow for reuse in different types of application.

III. Discussion

Code pushdown or code-to-data concept is not a new one. In fact, some would argue that database stored procedures offer a similar performance advantage to CDS entities. However, stored procedure languages offer abstractions close to the database layer and they often lack concepts to express the application semantics. Moreover, with stored procedures one would lose the advantages of the CDS unified development environment by introducing code into the system that is more difficult to maintain, challenging to test, often frustrating to debug and needs to be integrated and managed through diverse database interfaces [9]. With CDS, one can make use of SAP's available tools to assist developers during CDS development, testing, and in their deployment to the production systems. Practically, CDS objects are integrated in the repository of SAP applications artifacts for complete life cycle management. They are like any other SAP development objects, subject to the transport system within SAP so that they can be easily deployed or transported from development via test to a production system consistently.



```
@AbapCatalog.sqlViewName: 'ZCDS_UNTCNV'
@AccessControl.authorizationCheck: #CHECK
//Annotation to generate the OData service automatically
@OData.publish: true
  define view zdemo_converesion
   with parameters to_unit:abap.unit(3)
   as select from ZX_TABLE
   {
    id,

    //Enabling the fuzzy search on the UI level
    @Search.defaultSearchElement : true
    @Search.fuzzinessThreshold : 0.8
    @Search.ranking : #HIGH
    ProductDescription.text as Name,

    @Search.defaultSearchElement : true
    @Search.fuzzinessThreshold : 0.7
    @Search.ranking : #LOW
    Product.category as Category,


    dec3 as original_value,
    //Using SAP standard convrsion expression to convert
    //values of operand to dictionary specific data type
    cast( 'MI' as abap.unit(3) ) as original_unit,



     //Utilizing SAP standard Table Functions for conversions between units
     unit_conversion( quantity => dec3,
                      source_unit => cast( 'MI' as abap.unit(3) ),
                      target_unit => :to_unit,
                      error_handling => 'SET_TO_NULL' ) as converted_value,
     :to_unit as converted_unit }
```

Standard * | Notebooks HT- | Hide Filter Bar

To show filters here, add them to the filter bar in Filters

Standard *

Product Category	Product ID
Notebooks	HT-1000
Notebooks	HT-1001
Notebooks	HT-1002

Standard filter allows to search for product category and product name

Fig. 3. Core Data Services Examples.

Applications based on CDS framework implies a major shift in development practices. Specifically, CDS shift the development paradigm from a process-centric activity to a data-centric activity [4]. For example, applications need to be designed to take advantage of CDS features and define new data hierarchies that can be readily used by applications. Since development is driven by the need to eliminate or reduce data movement from the database into application servers, which, in fact, is the main bottleneck for data-intensive applications in traditional three-tier architecture. Developers have to identify data-intensive parts of their application [10]. These parts can then be redesigned -leveraging CDS views- to push down the computation into the data layer. However, identifying data-intensive parts is not necessarily a trivial exercise. Therefore, Intuitive techniques for describing and modeling data are necessary [3]. One can guide this by best practices, design patterns, code analysis, performance profiling, etc.

Furthermore, a data-centric approach reduces the applications code footprint. Specifically, with consistent data models across systems, different applications can reuse the same models [7], [3]. Models can also be extended or built on top of other models (i.e., stacked) to meet the application needs. Service definitions such as OData and REST APIs can then be designed and exposed using the necessary models. Subsequently, user interfaces can be developed independently of the data and service definitions. In fact, with this approach application development becomes more of a service orchestration activity rather than a programming activity.

Concisely, the development practices should be guided by the following three principles: First, UI Rendering with presentation logic should be pushed up to the Client (i.e., browser, mobile apps). Second, data-intensive computation is done in the database and only the results are moved to the application server. Specifically, application server should handle control-flow and procedural logic only. Third, development artifacts from all layers are managed in a central repository so they can be easily created, tested, integrated and deployed.

This will also introduce a number of challenges that should be considered in our development methodology [7], [11]. For example, requirements should not only capture how the process works but they should also describe the data acquisition process. Similarly, analysis activities need to carefully consider data sources, transformation and context as well as the traditional analysis of business processes. Finally, data-centric development requires additional effort to ensure that effective data quality controls are in place. Therefore, testing should expand from verifying application functionality to ensuring application data quality are validated, and validation rules are carried over to operations and maintenance of the application. These are only some of the challenges that must be considered in the development life cycle.

IV. Concluding Remarks

CDS offer easy-to-understand, reusable tools to help realize code push-down (i.e., Code-to-Data) model. With CDS, it is possible to build applications that integrate application control logic in the database layer to achieve real-time performance. It may require additional effort to move logic down to the database level. However, it reduces complexity and leads to simplification in data models and applications, redefining application development practices. Developers of business applications must therefore make careful choices about what data operations to include and what to omit. Certainly, it's not feasible or economically viable to model all existing applications data operations in CDS. Alternatively, if too little is included, the models may not support application needs adequately, which pushes more development effort and cost onto the application developers. This shift in application development practice presents new challenges, however, it provides opportunities to support new kinds of interactive applications, which were not possible before.

In future research, we plan to investigate processes that are appropriate for the CDS concept in our ERP system. Furthermore, we plan to evaluate costs and benefits of transforming existing business application to make use of the CDS concept. Specifically, we plan to develop criteria to guide the developers in identifying which applications or parts of an application are more appropriate for CDS. One idea we have is to utilize performance-profiling tools to prioritize applications for further evaluation. Furthermore, we plan to evaluate our development methodology and identify how it needs to be adapted to meet the expectations of this transformation.

References

[1] Z. P. Matolcsy, P. Booth, and B. Wieder, "Economic benefits of enterprise resource planning systems: some empirical evidence," *Accounting & Finance*, vol. 45, no. 3, pp. 439–456, 2005.

[2] V. Sikka, F. Färber, W. Lehner, S. K. Cha, T. Peh, and C. Bornhövd, "Efficient transaction processing in sap hana database: the end of a column store myth," in *Proceedings of the 2012 ACM SIGMOD Int. Conf. on Management of Data*. ACM, 2012, pp. 731–742.

[3] J.-H. Boese, C. Tosun, C. Mathis, and F. Faerber, "Data management with saps in-memory computing engine," in *Proceedings of the 15th Int. Conf. on Extending Database Technology*. ACM, 2012, pp. 542–544.

[4] H. Plattner, "A common database approach for oltp and olap using an in-memory column database," in *Proceedings of the 2009 ACM SIGMOD Int. Conf. on Management of data*. ACM, 2009, pp. 1–2.

[5] N. May, A. Bohm, and W. Lehner, "Sap hana– the evolution of an in-memory dbms from pure olap processing towards mixed workloads," *Datenbanksysteme für Business, Technologie und Web*, 2017.

[6] F. Färber, S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner, "Sap hana database: data management for modern business applications," *ACM Sigmod Record*, vol. 40, no. 4, pp. 45–51, 2012.

[7] H. Plattner, "The impact of columnar in-memory databases on enterprise systems: implications of eliminating transaction-maintained aggregates," *Proc. of the VLDB Endowment*, vol. 7, no. 13, pp. 1722–1729, 2014.

[8] J. Hrastnik, R. Dentzer, and R. Colle, *Core Data Services for ABAP*. SAP PRESS, 2019.

[9] N. May, A. Böhm, M. Block, and W. Lehner, "Managed query processing within the sap hana database platform," *Datenbank-Spektrum*, vol. 15, no. 2, pp. 141–152, 2015.

[10] F. B. Alomari and D. A. Menascé, "Self-protecting and self-optimizing database systems: Implementation and experimental evaluation," in *Proc. of the 2013 ACM Cloud and Autonomic Comp. Conf.*, 2013, pp. 1–10.

[11] A. Boehm, "In-memory for the masses: enabling cost-efficient deployments of in-memory data management platforms for business applications," *Proc. of the VLDB Endowment*, vol. 12, no. 12, pp. 2273–75, 2019.