Software Engineering 2023, 10(2): 21-27
DOI: 10.5923/j.se.20231002.01

Performance Metrics on Creating Data Via Custom OData API in SAP

Srihariram Chendamarai Kannan

SAP ABAP Lead Analyst, Edison, New Jersey, USA

Abstract This article introduces the performance testing metrics for a proposed solution that aims to generate data in SAP using a proprietary OData APIs method. The experimentation was conducted utilizing the SAP Gateway Client, specifically the transaction code /IWFND/GW_CLIENT, to assess the performance implications associated with mass creation scenarios when employing the SAP OData API. This scholarly article offers significant insights into the performance factors that necessitate attention when utilizing the SAP OData API for mass creation scenarios. The testing outcomes indicated that the suggested solution successfully satisfied the performance criteria and demonstrated its capacity to effectively manage substantial quantities of data. This article serves as a valuable resource for experts aiming to enhance their SAP performance optimization strategies.

Keywords SAP OData, SAP API, OData V4, SAP Restful-ABAP Programming Framework

1. Introduction

In today's data-driven business landscape, organizations rely heavily on efficient and high-performing data integration technologies to ensure smooth operations and optimal user experiences. ODATA (Open Data Protocol) has gained significant traction as a preferred protocol for building RESTful APIs and facilitating data exchange. OData offers a straightforward and standardized approach to creating and accessing queryable and interoperable APIs. Its metadata, which provides a machine-readable description of API data models, empowers the development of versatile client applications and tools. Numerous SAP applications and services utilize OData APIs which have become the preferred protocol for exposing data stored within SAP applications, making it accessible for external use. OData has a pivotal "connecting point" that streamlines interoperability between SAP and non-SAP applications. Any application or programming language that is capable of communicating via HTTP can consume OData services because they adhere to RESTful principles. For instance, you can create a Python, Go, or Rust application or service that interacts directly with an OData service, including integration with mobile applications.

This paper focuses on exploring the performance capabilities of ODATA and sheds light on its potential to deliver fast, scalable, and responsive data access within SAP systems.

* Corresponding author:
srihariram@hotmail.com (Srihariram Chendamarai Kannan)
Received: Aug. 19, 2023; Accepted: Nov. 22, 2023; Published: Nov. 29, 2023
Published online at http://journal.sapub.org/se

In SAP, data is well organized and saved within a database. Multiple layers of security are put in place to safeguard its data. To engage with this data, specialized tools in the form of modules, functions, or classes are crafted to enable precise interactions with this data model, whether for reading or writing. These tools are widely recognized as APIs (Application Programming Interfaces). Modern applications and ERPs are commonly equipped with APIs that streamline communication with their internal data structures. Similarly, SAP Gateway plays a crucial role in enabling the development and deployment of RESTful APIs for SAP systems. It serves as a central hub for integrating various SAP components, business processes, and external systems. By utilizing SAP Gateway, organizations can establish seamless communication channels and expose SAP data and functionality through standardized protocols, including OData.

SAP OData represents SAP's implementation of the OData protocol within its software ecosystem. It offers a comprehensive set of tools, frameworks, and services that empower developers to build OData-compliant APIs and leverage the capabilities of SAP systems.

OData V4 is the latest version of the OData protocol, introducing several enhancements and advancements over its predecessors. It provides improved support for complex data models, expanded querying capabilities, and enhanced security features.

2. Motivation and Goal

Performance plays a critical role in determining the effectiveness of any data integration solution. With the rise in data volumes and the increasing complexity of business

processes, organizations are seeking ways to optimize the speed, throughput, and efficiency of their data exchange mechanisms. ODATA, with its standardized approach and extensive tooling support, presents an opportunity to address these performance challenges and ensure smooth data access and transfer between SAP systems and external applications.

This paper aims to delve into the various factors that influence the performance of ODATA and examine best practices for optimizing its capabilities. It will explore techniques for efficient data creation and other performance-enhancing measures specific to ODATA implementations. By understanding these performance capabilities and techniques, organizations can leverage ODATA to maximize the speed and responsiveness of their data integration processes, ultimately improving overall system performance and user satisfaction.

3. Approach

The focus of this paper is to explore the slim and lightweight protocol provided by the OData and SAP gateway for efficient data consumption. While create operations are typically performed for single entities, there are valid scenarios where the creation of multiple entities simultaneously becomes necessary. To evaluate the performance and scalability of producing data in SAP through custom OData APIs, a series of experiments were conducted using SAP S4 HANA (2021 Version). The OData V4 protocol was implemented in SAP HANA through various approaches. Two commonly used methods for creating OData V4 were explored:

1. Using the RAP (Restful-ABAP Programming and CDS) Framework: The RAP Framework offers both managed and unmanaged scenarios for creating OData V4. The managed scenario provides a structured and standardized approach for generating OData V4 entities, ensuring consistency and ease of maintenance. On the other hand, the unmanaged scenario allows more flexibility and customization but may require additional effort for development and maintenance.
2. Code-Based Implementation for OData V4: This approach involves writing custom code to implement the OData V4 protocol. It provides greater flexibility and control over the implementation, allowing for fine-tuning and customization according to specific requirements. However, it may also involve more complex development and maintenance processes compared to using the RAP Framework.

Since most of the requirements in SAP need custom development, so will consider unmanaged RAP V4 API and Code based V4 API. By creating these objects in SAP S4 HANA (2021 Version) and implementing OData V4 through different approaches, the performance and scalability of staging data in SAP can be thoroughly tested. This research aims to provide insights into the effectiveness and efficiency of custom OData APIs for managing and manipulating data in SAP environments.

4. Technical Details

4.1. Prerequisite

- ABAP Programming: You should have a good understanding of SAP ABAP programming, including class and object-oriented programming (OOP) concepts. Additionally, you should be familiar with OData development and gateway configurations in SAP.
- CDS Development: You should have a fundamental understanding of SAP Core Data Service SQL queries and annotations.
- S4HANA: You should be familiar with SAP S4HANA newer versions and Restful-ABAP programming concepts.

By mastering these prerequisites, you can start creating your own OData objects in SAP.

4.2. Technical Design

In this document, it will be examined which approach is better for creating or loading data using OData V4 service.

- Step 1: Create database tables and CDS views to load data.

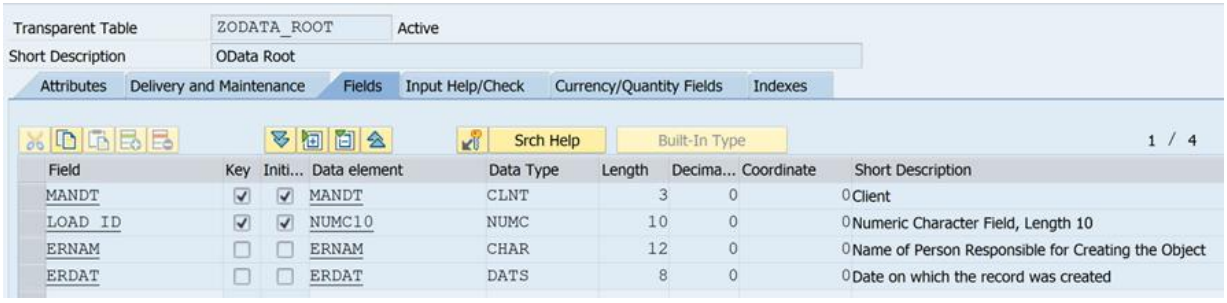

Figure 1. Header Table

Transparent Table ZODATA_LOAD Active
Short Description OData Load
Attributes | Delivery and Maintenance | Fields | Input Help/Check | Currency/Quantity Fields | Indexes
Srch Help | Built-In Type | 1 / 13

Field	Key	Initi...	Data element	Data Type	Length	Decima...	Coordinate	Short Description
MANDT	☑	☑	MANDT	CLNT	3	0	0	Client
LOAD_ID	☑	☑	NUMC10	NUMC	10	0	0	Numeric Character Field, Length 10
UNIQUE_ID	☑	☑	NUMC10	NUMC	10	0	0	Numeric Character Field, Length 10
NAME_LAST	☐	☐	BP_NAMLAST	CHAR	40	0	0	Last Name
NAME_FIRST	☐	☐	BP_NAMFRST	CHAR	40	0	0	First Name
NATION	☐	☐	BP_CNTR_N	CHAR	3	0	0	Nationality
STATE	☐	☐	BP_CNTR_ST	CHAR	3	0	0	Citizenship
BIRTH_DATE	☐	☐	BP_BIRTHD	DATS	8	0	0	Date of Birth of Business Partner
BIRTH_PLAC	☐	☐	BP_BIRTHPL	CHAR	40	0	0	Birthplace
MARITAL_ST	☐	☐	BP_MARITAL_OLD	NUMC	2	0	0	Marital Status
CHILDREN	☐	☐	BP_CHILDRN	DEC	2	0	0	Number of Children Belonging to the Business Partner
ERNAM	☐	☐	ERNAM	CHAR	12	0	0	Name of Person Responsible for Creating the Object
ERDAT	☐	☐	ERDAT	DATS	8	0	0	Date on which the record was created

Figure 2. Item Table

```
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'OData Root Unmanaged'
define root view entity ZCDS_I_UMROOT
  as select from zodata_root
  composition [0..*] of  ZCDS_I_UMLOAD as _Item
{

  key load_id,
      ernam,
      erdat,

      _Item
}
```

Figure 3. CDS View for Header

```
@AbapCatalog.viewEnhancementCategory: [#NONE]
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'OData Load data Un Managed'
@Metadata.ignorePropagatedAnnotations: true
@ObjectModel.usageType:{
    serviceQuality: #X,
    sizeCategory: #S,
    dataClass: #MIXED
}
define view entity ZCDS_I_UMLOAD
  as select from zodata_load
   association to parent ZCDS_I_UMROOT as _Header on $projection.load_id = _Header.load_id
{
  key load_id,
  key unique_id,
      name_last,
      name_first,
      nation,
      state,
      birth_date,
      birth_plac,
      marital_st,
      children,
      ernam,
      erdat,

      _Header
}
```

Figure 4. CDS View for the Item

- Step 2: Create a Restful-ABAP Unmanaged V4 OData API.

  a. Create Unmanaged Restful-ABAP Objects

```
unmanaged implementation in class zbp_cds_i_umroot unique;
strict;

define behavior for ZCDS_I_UMROOT //alias <alias_name>
//late numbering
lock master
authorization master ( instance )
//etag master <field_name>
{
  create;
  update;
  delete;
  association _Item { create; }
}

define behavior for ZCDS_I_UMLOAD //alias <alias_name>
//late numbering
lock dependent by _Header
authorization dependent by _Header
//etag master <field_name>
{
  update;
  delete;
  field ( readonly ) load_id;
  association _Header;
}
```

Figure 5. Unmanaged behavior definition

[ED5] ZSD_ODATA_UMLOAD

```
@EndUserText.label: 'OData UnManaged Load'
define service ZSD_ODATA_UMLOAD {
  expose ZCDS_C_UMROOT;
  expose ZCDS_I_UMROOT;
  expose ZCDS_I_UMLOAD;
}
```

Figure 6. Unmanaged Service Definition

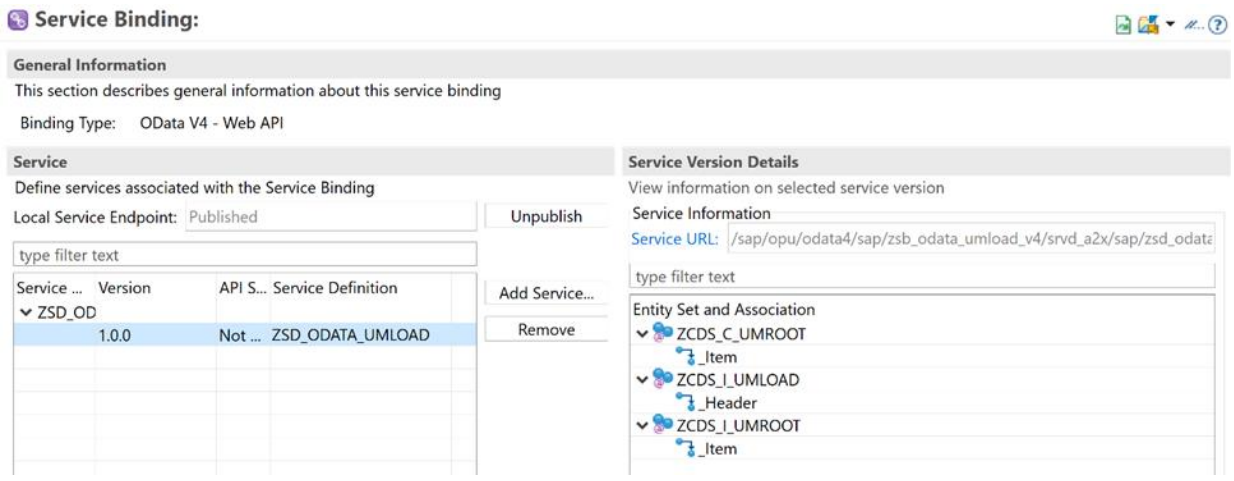

Figure 7. Unmanaged Service Binding

- Step 3: Create a Code based V4 ODATA classes using SAP provide interfaces /IWBEP/IF_V4_MP_BASIC and /IWBEP/IF_V4_DP_ADVANCED.

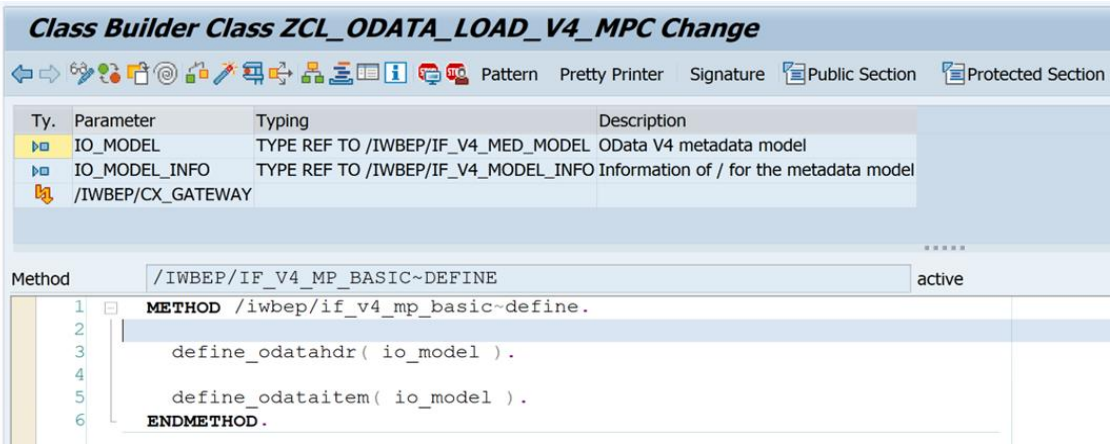

Figure 8. Model provider class for V4 OData

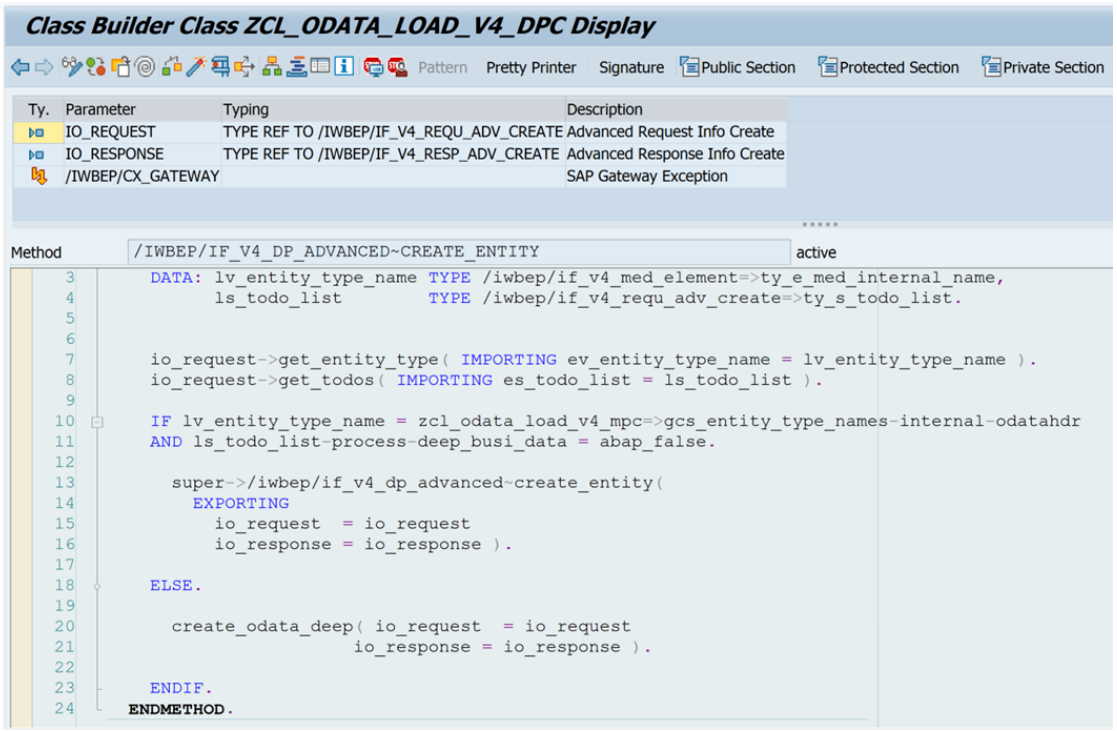

Figure 9. Data provider class for V4 OData

5. Testing and Performance Metrics

To evaluate the impact of different approaches, we conducted tests with and without utilizing changesets in the OData and SAP Gateway frameworks and tried to examine the performance and efficiency of alternative methods. To gauge the performance of the create process, the SAP Gateway framework provides comprehensive performance statistics in the HTTP response header. These statistics are structured in the following format [3]:

- HTTP header name: sap-statistics
- HTTP header value as below,
  a. total: Total processing time
  b. fw: Framework
  c. app: Application
  d. gwtotal: Total processing time of the OData request.
  e. gwhub: Processing time in SAP Gateway hub system.
  f. gwrfcoh: RFC and network overhead for communication between the hub and backend system.
  g. gwbe: Processing time in the SAP Gateway framework in the backend system (without application time).
  h. gwapp: Processing time in the application (data provider).
  i. gwnongw: Processing time of applications called (referred to as non-SAP Gateway since this processing time is not related to the SAP Gateway framework).

5.1. Testing RAP Unmanaged V4 OData API

I tried to create data in SAP using Restful ABAP programming and an unmanaged OData API with multiple sets of records.

Sample Payload

The below given JSON payload represents data for SAP OData API request. It includes information about a header and associated item details. Multiple items can be added in the “_item” section. Used similar payloads for both RAP OData and code based V4 OData APIs.


```
{
    "load_id" : "0001",
    "ernam" : "Test",
    "erdat" : "20230101",
   "_Item": [
      {
    "load_id" : "0001",
    "unique_id" : "0000000001",
    "name_last" : "Configure",
    "name_first" : "Test",
    "nation" : "USA",
    "state" : "NJ",
    "birth_date" : "19900101",
    "birth_plac" : "EDISON",
    "marital_st" : 02,
    "children" : 0,
    "ernam" : "Test",
    "erdat" : “20230101"
        }
    ]
  }
```


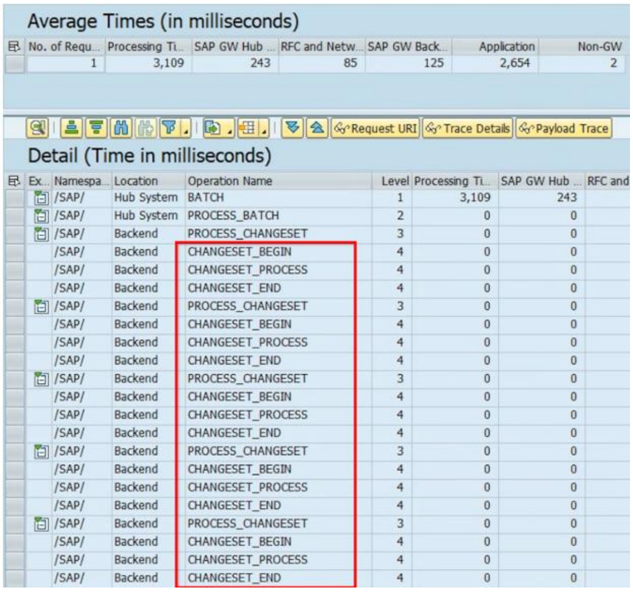

Figure 10. Evidence, Mutli Changeset approach calls multiple time to commit

All operations within a change set must be treated as a logical unit of work. This means all or nothing. Therefore, a provider must not issue COMMIT WORK or ROLLBACK WORK during change set processing. Otherwise, the framework will abandon the change set processing. If the change set contains only one operation, the check for commit or rollback is deactivated [2]. The Multi Changeset option for the create process will be considered the least favorable due to the repeated invocation of the CREATE_ENTITY method in the DPC Extension class and subsequent multiple COMMIT WORK operations. The Multi Changeset approach can result in performance degradation and increased costs. However, to optimize data staging, it is beneficial to commit multiple records simultaneously. Using the payload, The OData API was executed from the transaction code /IWFND/GW_CLIENT and measurements were taken using transaction /IWFND/TRACES.

Table 1. Performance metric (msec) for RAP Unmanaged Multi changeset

Records	gwtotal	gwhub	gwbe	gwapp	Gwrfcoh
1	139	20	18	46	52
10	368	111	19	159	76
100	2503	1010	39	1406	44
1000	25061	10442	197	14062	359
5,000	143568	50248	1824	90862	634
10,000	317927	106322	5787	204565	1225

Prepared the similar payloads for the single changeset and executed in transaction code /IWFND/GW_CLIENT, and the processing time was reduced by approximately 50%.

Table 2. Performance metric (msec) for RAP Unmanaged single changeset

Records	gwtotal	gwhub	gwbe	gwapp	Gwrfcoh
1	212	31	24	71	86
10	349	112	29	95	92
100	1873	975	31	151	713
1000	11432	10024	203	1035	157
5,000	59567	51967	1823	5127	638
10,000	127300	110473	5753	10052	1058

Although the change set approach offers advantages in terms of committing data, it is not without its limitations. One significant drawback is its potential slowness, as each changeset requires its own set of processing steps. This can create a bottleneck in the overall data staging process, impacting performance.

Another approach is testing the payload with single create requests and no changeset in the RAP unmanaged V4 ODATA API.

Table 3. Performance metric (msec) for RAP Unmanaged V4 OData API

Records	gwtotal	gwhub	gwbe	gwapp	gwrfcoh
1	230	15	21	98	96
10	161	19	11	71	60
100	322	48	33	201	40
1000	1458	349	13	1062	34
5,000	7336	1695	44	5465	132
10,000	14129	3359	48	10515	207

5.2. Testing Code Based V4 OData API

Executed the payload of data with deep create request on V4 OData code based implemented API too. And it was much faster than the RAP Framework.

Table 4. Performance metric (msec) for code based V4 OData API

Records	gwtotal	gwhub	gwbe	gwapp	gwrfcoh
1	218	35	59	39	85
10	150	36	17	26	71
100	119	52	5	27	35
1000	495	266	28	135	66
5,000	1893	1232	28	552	81
10,000	3786	2443	86	1117	140

5.3. Comparison

Performance improvement with Code Based Implementation as compared to RAP API is 75%.

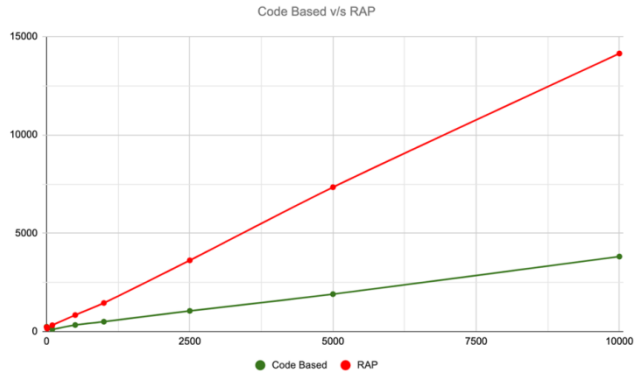

Figure 11. Performance comparison between Code based and Unmanaged RAP API

By deferring the changeset process and adapting the deep create approach in code-based development (Figures 10 and 11) helped to improve the performance. Also, OData V4 has more advantages, like supporting both XML and JSON formats, metadata query on service level only. It can be used to query both root and expanded entities and supports deep expand and query options in expanded entities.

6. Conclusions

The findings of this study highlight the significant advantages of a code based approach in achieving superior performance results. Therefore, if project timelines permit, transitioning to a code based implementation should be given a higher priority. By doing so, organizations can maximize the efficiency and responsiveness of their data staging processes, resulting in improved overall performance.

It is essential to acknowledge that the suitability of this approach may vary based on specific use cases. Different scenarios and requirements might influence the decision-making process. As such, a careful assessment of individual project needs is recommended before finalizing the choice of implementation.

In summary, our research underscores the importance of selecting the most appropriate approach for data staging, considering both without ChangeSet and with ChangeSet options. Additionally, it emphasizes the significance of adopting a code-based strategy for optimizing performance whenever feasible. With this knowledge, organizations can make well-informed decisions to enhance data staging operations and achieve higher levels of efficiency and productivity.

REFERENCES

[1] https://community.sap.com/topics/abap/rap.

[2] https://help.sap.com/doc/saphelp_nw74/7.4.16/en-us/94/a126519eff236ee10000000a445394/content.htm?no_cache=true

[3] SAP Performance Statistics. (n.d.). https://help.sap.com/doc/saphelp_nw75/7.5.5/de-DE/40/93b81292194d6a926e105c10d5048d/content.htm?no_cache=true.