



**UNIVERSITY OF NAIROBI  
FACULTY OF SCIENCE AND TECHNOLOGY  
DEPARTMENT OF COMPUTER SCIENCE**

**USING IN-MEMORY COMPUTING TO PROVIDE REAL-TIME  
AND ACTIONABLE SALES INSIGHTS**

**JACKSON ODONGO ODUOR**

**P54/35143/2019**

**SUPERVISOR:**


**CHRISTOPHER A. MOTURI**

**A research project report submitted to the Department of Computer Science Faculty of Science and Technology in partial fulfillment of the requirement for the award of Masters of Science Degree in Information Technology Management of the University of Nairobi.**

**August 2021**

## DECLARATION

This project report, to the best of my knowledge, is my original authorial work except as acknowledged therein and has not been submitted for any other degree or professional qualification award in this or any other University.

Signature:  98E6FC1F9F8F44C... \_ \_ \_

Date: 4<sup>th</sup> August 2021

Oduor Jackson Odongo  
P54/35143/2019.

This report has been submitted as partial fulfillment of the requirements for the degree of Master of Science in Information Technology Management at the University of Nairobi with my approval as the University supervisor.

Signature



Last modified: 21:46

Date: *August 4, 2021*

Christopher A. Moturi  
Department of Computer Science  
Faculty of Science and Technology  
University of Nairobi.

## **ACKNOWLEDGEMENT**

I thank God for Mr. Christopher Moturi for his inordinate guidance throughout this research project. I'm grateful to Professor Daniel Orwa for his review. My special thanks to my family for all the endless support.

## **DEDICATION**

I dedicate this research study to my daughter, Shalom.

## **ABSTRACT**

The adoption of enterprise resource planning systems in both the public and private sectors aims at having business transactions effectively captured, processed and stored. However, for most businesses, running real-time models on transactional and historical data is time-consuming, often happening overnight to prevent system contention. In order to gain a competitive edge, instant information is key to organizations. It empowers decision-making and improves the quality of the decisions made. This research implemented an analytics dashboard prototype that uses in-memory computing in the cloud foundry environment to leverage the parallel computing offered by the cloud environment. The prototype was simulated in a multicore environment with 16 and 32 core processing unit cores. The response times for 1, 2, 4, 16, 64 and 128 cores were calculated using Amdahl's law of response times. A survey on the effectiveness of the IMC-based analytics dashboard in the business context was conducted with 20 key business users. The research findings revealed that save for when concurrent load is required and the CPU bandwidth to the memory bus notwithstanding, organizations intending to use in-memory computing technology such as HANA do not need to spend in core processing units acquisition of more than 16 cores. This research also established that real-time analytics and reporting is realized by in-memory technology's high-end computing performance. Real time information ensures continuous business transparency. It also empowers decision-making at strategic and operational levels as well as improves the quality of the decisions made.

# **TABLE OF CONTENTS**

<b>CHAPTER ONE</b>	11
<b>INTRODUCTION</b>	11
<b>1.1 Background</b>	11
<b>1.2 Problem Statement</b>	12
<b>1.3 Objectives</b>	13
<b>1.4 Research Questions</b>	13
<b>1.5 Significance</b>	13
<b>CHAPTER TWO</b>	14
<b>LITERATURE REVIEW</b>	14
<b>2.1 Introduction</b>	14
<b>2.2 OLTP vs OLAP in ERP Design</b>	14
<b>2.3 The Capabilities of In-Memory Computing</b>	16
<b>2.4 The New Programming Model with IMC</b>	24
<b>2.5 Literature Summary</b>	25
<b>2.6 Overall System Architecture</b>	26
<b>CHAPTER THREE</b>	27
<b>RESEARCH METHODOLOGY</b>	27
<b>3.1 Introduction</b>	27
<b>3.2 System Development Methodology</b>	27
<b>3.3 Evaluation &amp; Analysis</b>	32
<b>CHAPTER FOUR</b>	35
<b>IMPLEMENTATION, RESULTS &amp; DISCUSSION</b>	35
<b>4.1 Introduction</b>	35
<b>4.2 Specifications and Analysis</b>	35
<b>4.3 Inputs and Outputs</b>	36
<b>4.4 The Design of the Open CDS-based Virtual Data Modelling in HANA</b>	37
<b>4.5 Runtime Statistics of the Prototype</b>	38
<b>4.6 Effectiveness of the In-Memory Computing in the organizational context</b>	41
<b>CHAPTER FIVE</b>	43
<b>SUMMARY, CONCLUSION AND RECOMMENDATIONS</b>	43

<b>5.1 Introduction</b>	43
<b>5.2 Linking the Study Findings to Objectives</b>	43
<b>5.3 Conclusion</b>	46
<b>5.4 Limitations of the study</b>	47
<b>5.5 Recommendations for future research</b>	47
<b>REFERENCES</b>	48
<b>APPENDICES</b>	49
<b>Appendix I: Prototype Survey</b>	49
<b>Appendix II: The Analytics Dashboard</b>	50
<b>Appendix III: The CDS Views</b>	51
<b>Appendix IV: The Business Logic</b>	53
<b>Appendix V: The OData Project</b>	59

**TABLE OF FIGURES**

Figure 1: Transaction-consistent snapshot of the OLTP data (Funke et al., 2012). ..... 18

Figure 2: The count of queries executed in two sets of ERP systems (May et al., 2017)..... 20

Figure 3: NFAE and FDA optimizations in (May et al., 2017). ..... 21

Figure 4: Nested CDS views relationship (May et al., 2017). ..... 23

Figure 5: Traditional Data-to-Code model vs New Code-to-Data model (Heilman et al., 2013). 24

Figure 6: Code Pushdown with Application Server ABAP (Heilman et al., 2013)..... 25

Figure 7: Overall System Architecture (Pattanayak, 2017). ..... 26

Figure 8: Design-Led Development Process (SAP SE.2020.<https://experience.sap.com>)..... 27

Figure 9: The Prototype Dashboard Design..... 28

Figure 10: The Proposed Analytics User Interface..... 29

Figure 11: Development and Deployment Process Steps ..... 31

Figure 12: Persona for the dashboard prototype ..... 36

Figure 13: The Design of the CDS-based Virtual Data Model..... 37

Figure 14: Response time with an IMC on 32 CPU Cores ..... 38

Figure 15: Response time with an IMC on 8 CPU Cores ..... 39

Figure 16: The Analytics Dashboard’s Average Response Time ..... 40

Figure 17: The Industry Distribution in the Survey on using the IMC-based Analytics Dashboard  
..... 41

Figure 18: Results of the Effectiveness of the In-Memory Computing in the organizational  
context..... 42



## **TABLE OF TABLES**

Table 1: Inputs & Outputs.....	36
Table 2: Response times with an IMC on 8 & 32 CPU Cores.....	39
Table 3: Response times with IMC on CPU Cores between 1 and 128 .....	40

## **ABBREVIATIONS**

ABAP	Advanced Business Application Programming
ANSI	American National Standard Institute
API	Application Programming Interface
BI	Business Intelligence
BPM	Business Performance Management
DBMS	Database Management System
DML	Data Manipulation Language
CDS	Core Data Services
CFO	Chief Financial Officer
CRM	Customer Relationship Management
CSS	Cascading Style Sheets
DDL	Data Definition Language
ERP	Enterprise Resource Planning
FDA	Fast Data Access
HANA	High performance Analytics Appliance
HTML5	Hyper Text Markup Language version 5
IMC	In-Memory Computing
JSON	JavaScript Object Notation
KPI	Key Performance Indicators
LLVM	Low Level Virtual Management
MMU	Memory Management Unit
MVC	Model-View-Controller
NFAE	Native For All Entries
ODATA	Open Data Protocol
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
SAP	System Applications and Products in Data Processing

# CHAPTER ONE

## INTRODUCTION

### 1.1 Background

The adoption of ERP Systems in both the public and private sectors aims at having business transactions effectively captured, processed and stored. A number of Enterprise Resource Planning systems have emerged to automate business or organizational processes such as Financials, Sales & Distribution, Procurement, Human Capital Management, Enterprise Asset Management, Production Planning, Quality Management and Project Systems. Properly implemented ERPs have been effective as far as driving the business operations are concerned. Since the beginning of the century, there has been an explosion in the volume of data from business operations. This has birthed two types of Information Systems namely, Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP).

OLTP is effectively delivered with traditional ERPs. As part of the project cutover, all the data from legacy systems are migrated for system Go-Live preparations. Post system Go-Live, more data continues to be captured in the form of business transactions. Over time, this creates a huge but significant data warehouse necessary for providing business intelligence. The need to have real-time data analytics and the never-ending demand for better and faster performance has been limited by the architectural designs of the ERP systems. The most recent designs of the traditional ERPs comprises either a 2-tier architecture or a 3-tier architecture. Both architectures have the application layer separate from the database layer, and the code (business logic) resides in the application server but interfaces with the database server through I/O processing. Most of the database servers used in these architectures are disk-based and row-oriented (Farber et al., 2012). This leads to slow performance especially with the need to ingest millions of records in less than a second required for action and/or decision-making. The solution to such reporting and query requirements is the OLAP.

In OLAP, analytical processes mostly run overnight to prevent system contention hence getting analytical report is a process that can take more than a day. Moreover, much as data warehouse

systems are an established component of the information systems landscape in most companies, many have failed due to architectural concepts and data modeling. OLAP has been deployed and applied in many industries such as manufacturing (for order shipment and customer support), retail (for user profiling and inventory management), financial services (for claims analysis, risk analysis, credit card analysis, and fraud detection), transportation (for fleet management), telecommunications (for call analysis and fraud detection), utilities (for power usage analysis), and healthcare (for outcomes analysis) (Reddy et al., 2010).

In recent developments in OLAP, column-stores have become more popular leading to In-Memory Computing (IMC) as is the case with Hyper System and SAP High Performance Analytic Appliance (HANA) database. IMC's capability to execute the business logic inside the database kernel as opposed to inside the application server is what makes it more powerful in handling data analytics with lower latency. Leveraging cloud computing and mobile technologies provides even a more powerful user experience by offering faster data analytics irrespective of device or operating system platform. The advent of cloud computing and mobile technologies has prompted for access to real-time, actionable and relevant insights from mobile devices. Real-time analytics is an enabler to revenue growth and operational efficiencies (Agostino, 2004). This research is relevant for businesses running ERPs especially with Sales & Distribution modules and in particular for Chief Commercial Officers, Chief Information Officers, Chief Technology Officers, as well as Head of Information and Communications Technology. It is also important for System Analysts, System Developers, and Solution Architects who have an interest in code-to-data paradigm, where data intensive operations are pushed down to the database instead of to the application server.

## **1.2 Problem Statement**

With the steady growth of e-commerce, there is need for real-time analytics on data especially in sales performance management. For most businesses, running real-time models on transactional and historical data is time-consuming, often happening overnight to prevent system contention. It is a significant concern for organizations to get real-time queries at lower latency.

### **1.3 Objectives**

- a) To examine the capabilities of In-Memory Computing
- b) To design and develop a prototype dashboard that leverages an IMC for real-time analytics
- c) To experiment the application of code-to-data paradigm
- d) To evaluate the latency in data processing from the prototype in the business context

### **1.4 Research Questions**

- a) What are the capabilities of In-Memory Computing?
- b) How much of a paradigm shift does In-Memory Computing has on traditional computer programming?
- c) What is the CPU core requirement for real-time data visualization in In-Memory Computing?
- d) How applicable is the In-Memory Computing in the business context?

### **1.5 Significance**

In order to gain a competitive edge, instant information is key to organizations. It empowers decision-making at operational levels and improves the quality of the decisions made. It also strengthens the relationship between the organization and its customers, suppliers, partners and shareholders.

# CHAPTER TWO

## LITERATURE REVIEW

### 2.1 Introduction

This chapter reviews relevant literature on ERP designs in the context of Online Transaction Processing (OLTP) vs Online Analytical Transaction Processing (OLAP), the capabilities of In-Memory Computing with both OLTP and OLAP capabilities, and the new Code-to-Data programming paradigm brought about by In-Memory Computing.

### 2.2 OLTP vs OLAP in ERP Design

Online operational systems are used for transaction and querying processing. They are therefore termed as Online Transaction Processing (OLTP) Systems. Data warehouse systems are used for data analysis and decision-making, organizing and presenting data in various formats. They are therefore termed as Online Analytical Processing (OLAP) Systems. OLAP is a combination of analytical processing procedures and graphic presentation (the user's interface). OLAP instruments enable complex computations, provide users with the possibility to access, and analyze large data volumes, the relations among them, and to present the data from various views (a multidimensional outlook of data) (Reddy et al., 2010) (Butuza et al., 2011). Reddy et al. (2010) highlight the following as the major distinguishing features between OLTP and OLAP: **User and system orientation**: While OLTP is used for transaction and query processing, OLAP is used for data analysis. **Data content**: OLTP system manages current data to a detailed format. OLAP manages a huge amount of historical data with summarization and aggregation. **Database design**: The OLTP database design has an entity-relationship data model and is application-oriented, whereas OLAP database design has a star or snowflake model and is subject-oriented. **Access**: access to OLTP systems is in short and atomic transactions whereas access to OLAP systems are read-only operations.

### The need for OLAP despite OLTP

OLTP is the prerequisite for OLAP. While centralized warehouses also handle the integration of data from many sources, it is still desirable to have OLTP and OLAP capabilities in one system, which could make both components more valuable to their users.

Seres & Medacovic (2014) observes that an operational database is designed and tuned from known tasks and workloads, such as indexing using primary keys, searching for particular records and optimizing 'canned queries'.

As data warehouse queries are often complex, they involve the computation of large groups of data at summarized levels and may require the use of special data organization, access and implementation methods based on multidimensional views. Processing OLAP queries in operational databases would substantially degrade the performance of operational tasks (Seres & Medacovic, 2014). While an OLAP query often needs read-only access of data records for summarization and aggregation, an operational database supports the concurrent processing of multiple transactions (Seres & Medacovic, 2014). Concurrency control and recovery mechanisms, such as locking and logging are required to ensure the consistency and robustness of transactions (Seres & Medacovic, 2014).

Execution of concurrent transactions may be adversely affected if recovery mechanisms and concurrency control were to be applied for OLAP operations (Seres & Medacovic, 2014). Whereas historical data is needed for decision support, it is not usually maintained in the operational databases. Therefore, the information from operational databases is incomplete for decision-making, which requires consolidation of information from heterogeneous sources (Seres & Medacovic, 2014). Keith Gile of Giga confirms that, "Enterprise wide data dissemination requirements are increasing significantly: analytic functions are no longer the domain of a few elite power-users. Companies need to deliver analytic functionality to a broad user community, targeted at the specific business needs of users via dashboards, key performance indicators, dynamic reporting, and real-time analytics." All the aforementioned relevant literature are based on architectural designs where the database server is separate from the application servers. This kind of ERP design leads to significant latency during analytics occasioned by I/O processing between the database layer and the business logic layer. In a bid to improve the speed of business analytics with lower latency, the In-Memory Computing technology is revolutionizing the design of ERP systems.

### **2.3 The Capabilities of In-Memory Computing**

Generally, the traditional data warehousing used for ERP reporting extracts, transforms and loads data from OLTP systems into OLAP systems at specified time intervals (Baboo et al., 2013). This delay of a day or less is not helpful for businesses especially with the current changing market environments. There is a need to analyze the business as it happens, with real-time information. Some organizations today cannot even afford an hour's delay to remain competitive. The question has always been whether it is possible to have both OLTP and OLAP in the same system, leveraging In-Memory technology to enable real-time analytics on transactional data. Baboo et al. (2013) correctly observes that tuples in OLTP are stored in blocks (arranged in rows) which reside in the disk and are cached in the main memory in the database server. Even though indexing allows faster access to single tuples, the access becomes slower as the number of requested tuples increases. OLAP, on the other hand, has data organized in star schemas, where the attributes (columns) are compressed with the help of dictionaries to provide for optimization. Faster processing is realized after the attributes are converted into integers. Optimization of the hardware, particularly the main memory is the antidote to having real-time analytics (on millions of records) on transactional data. This has been made more feasible by the fact that main memory is becoming cheaper and growing larger (Baboo et al., 2013). The traditional ERP systems store the data in the disk drives. When it comes to analysis, accessing the data from the disk has a significant impact on performance. Accessing data from the Random Access Memory (RAM) improves the performance of analytics. Having all the data required for analytics residing in RAM is the game changer that many businesses need to actualize real-time data visualization.

Moreover, modern CPUs (Central Processing Units) have emerged with tremendous computing power at an average of 8 CPUs with each CPU having about 16 cores. That results in 128 units of computing power with about 500 GB of main memory. Such CPU multicore architecture together with comparatively huge main memory is also a significant enabler to in-memory computing. Baboo et al. (2013) describe the key features of OLTP and OLAP in terms of memory storage of data tables. OLTP mainly uses row-store to store rows consecutively in the memory. It is recommended in cases where: a) the table has a small number of rows, for example, the configuration tables in ERP systems b) there is need to process a single record at a time c) there is need to fetch a complete set of a record like all the details of a customer master d) aggregation and fast searching are not required.



OLAP mainly uses column-store where columns are stored consecutively in the memory. It is optimal for attribute-focused access with aggregations like SUM, GROUP BY. It is recommended in cases where: a) there is need to fetch any single column based on any descriptive value b) an application has many tables and there is a requirement to search the table based on the values of only a few columns c) the table has more columns than rows d) there is need to aggregate values in the columns.

Baboo et al. (2013) suggest two advantages of column-store databases for OLTP. Firstly, since in column stores a compression factor of 10 can be realized compared to the traditional row-oriented database. Compressed data can be loaded into the CPU cache more quickly. Secondly, dictionary coding enables columns to be stored as sequences of bit-coded integers. During scans or join operations, checks for equality can be done on integers making it faster than comparing string values. With in-memory columnar storage, the column scanning speed and the dictionary compression allows read operations with tremendous performance. With this, no additional index structures are required as is the case with row-oriented storage where an effort for maintaining metadata is needed.

We look at two cases of in-memory databases capable of mixed OLTP and OLAP workloads namely, the Hyper System and SAP HANA. These two next-generation database systems offer OLAP capabilities operating directly on transaction data thus enabling real-time business intelligence.

### **2.3.1 The HyPer System**

Hyper allows queries to be run in parallel to transactions with very low overhead. It has a separate process forked from the OLTP process and which constitutes a transaction-consistent snapshot of the OLTP data. Queries are then executed in this separate process as shown below:

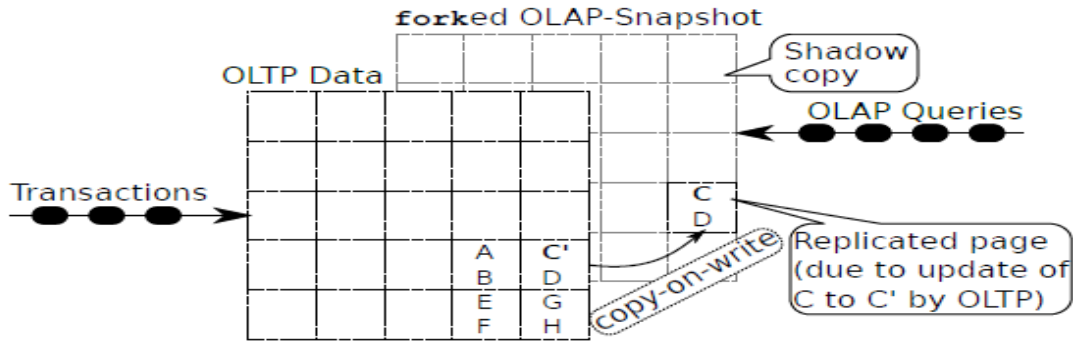


Figure 1: Transaction-consistent snapshot of the OLTP data (Funke et al., 2012).

The Operating System, with the help of the Memory Management Unit (MMU), keeps the snapshot consistent.

In order to maintain the memory state from the time of the snapshot creation, the original page is replicated for OLAP process before the OLTP process updates a page. This is the power of the HyPer's high performance, allowing the fastest dedicated online transactional systems and the fastest online analytics even if both workloads run in parallel on the same data. This is a hardware-based snapshotting, which has proven superior to the traditional software-based snapshotting.

HyPer writes transactions as stored procedures, which are compiled to native code by leveraging the Low-Level Virtual Machine (LLVM) compiler back-end. This allows for super-fast transactions in the order of thousands of transactions per second. HyPer's query engine built on Query Compilation Technique together with its sophisticated query optimizer, enables HyPer to realize milli-second query response times on common business intelligence queries like top items, top sales etc.

Funke et al. (2012) observe a number of compression characteristics about HyPer. Firstly, the small portion of data that is accessed frequently remains uncompressed to remain accessible for transactions. Dictionary stores and run-length encoding, which is favorable for columnar storage, is used for data that is rarely read or updated (cold data). Another reason for compressing cold data is that it allows for the use of a single dictionary, with smaller memory footprint, for all partitions of a relation thereby eliminating lock contention with OLTP processes.

This enables for greater compression rates and faster query execution because for every table scan there is a single dictionary scan.

Run-length encoding, in addition to dictionary compression, reduces the size of the dictionary-key columns and increases the scan performance (Funke et al., 2012).

Ordered-preserving dictionary compression found in the traditional analytical databases speeds up execution but is very difficult to maintain. Because the HyPer DBMS has to enable high-frequency updates and inserts, it uses a secondary index on the compressed attribute that consumes less memory than the traditional indexes on string attributes. Ordered dictionaries are powerful for range and prefix filter conditions e.g. *attribute LIKE 'prefix%'*. For equality filter conditions e.g. *attribute = 'value'*, the scan operator directly uses the secondary index to establish the qualified tuples hence preventing a full table scan. Filter conditions other than prefixes or equality, are evaluated by scanning the dictionary and selecting the qualifying keys into a hash table.

Hash table is usually small for many queries allowing it to fit into cache making an efficient hash join (Funke et al., 2012).

In conclusion, the Hyper DBMS has demonstrated that compression can be embedded on OLTP systems by compressing tuples that are rarely updated without affecting the transaction processing. This is realized by hardware improvements and achieving compression at lower overheads. Cold data can be easily compressed and optimized for effective and efficient query processing by relocating them to bigger memory pages. The DBMS, however, does not demonstrate how it can be used for ERPs, which is why it is important to have a look at another in-memory database, namely the SAP HANA.

### **2.3.2 The HANA Database**

The innovation of the SAP's High Performance Analytic Appliance (HANA) system was inspired by the increasing main memory capacity and the advent of multi-core systems that combines columnar storage and row-based query engine. HANA has evolved from HANA for analytical workloads in 2009, to HANA for transactional workload in 2012, to HANA for mixed workloads (i.e. both OLAP and OLTP) in 2015 (May et al., 2017).

Firstly, we look at the HANA for analytical workloads, which was informed by the inefficiencies in the Business Warehouse around architectural choices, aggregation queries and concurrency handling. To address these challenges, parallelization at all levels, from hardware to query execution formed the main design principle of HANA. The ever-growing number of CPUs in the market motivated this. Just like the HyPer database, HANA uses dictionary compression. Unlike Hyper, however, ordered dictionaries are used in HANA and buffers operations that update the database in a delta index which keeps the main table static. The buffered entries and the static portion of the main table are fused in an explicit merge operation to create a new version of the main part. Dynamic Tiering is a technique of HANA for analytical workloads that enables transparent aging of data where the main memory keeps the hot data (i.e. actuals) while the data warehouse keeps the cold data (i.e. historical data) (May et al., 2017).

Secondly, in order to support SAP ERP product, which is built on Advanced Business Application Programming (ABAP) language, HANA for OLTP was born. Column-store technique is optimized for workloads that leverages OLTP for thousands of statements per second, consisting of queries and concurrent updates. Since ERP systems have thousands of transactional tables, it is costly to replicate transactional data it increases the memory. Column store-based approach, therefore, has to be utilized to cater for the insert, update and delete operations. To avoid compilation of queries for OLTP, HANA employs a plan cache to store optimize SQL statements by linking the compiled query execution to the SQL statement. This prevents the parsing of the SQL (May et al., 2017). May et al. (2017) further illustrate the plan caching with a survey. Figure 2 shows a plotting of top 100 statements against the execution count. Only a fraction of the main memory was consumed by the plan cache in these systems to guarantee 99% hit ratio. Meaning that only about 1% of all the queries required compilation.

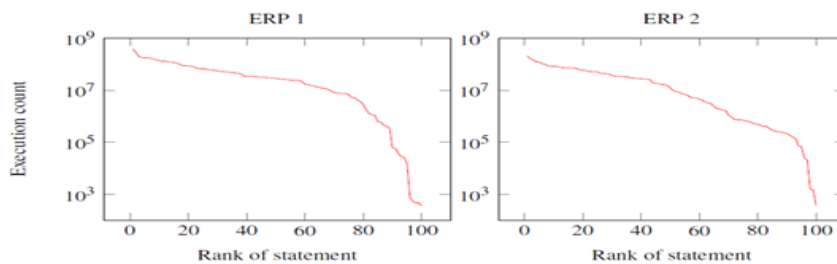


Figure 2: The count of queries executed in two sets of ERP systems (May et al., 2017).

Fast Data Access (FDA) and Native For All Entries (NFAE) are the two main features that have optimized the Application Server – DBMS co-design. FDA enables direct read/write from/into a special internal data representation shareable between the database system and the application server. This bypasses the SQL connectivity stack traditionally required for data transfer resulting in a 20% increase in speed as shown in Figure 3. NFAE, on the other hand, converts disjunctions and conjunctions into a semi-join from the modification of the ABAP runtime which is optimized for effective execution by the database system (May et al., 2017).

The following ABAP statement extracts customer information for all customers with active sales order and the specific customer numbers given within the run time table `it_vbak` residing in the application server:

```
SELECT c.kunnr , c.name1, c.ktokd , c.stceg FROM kna1 c INTO CORRESPONDING  
FIELDS OF TABLE it_data FOR ALL ENTRIES IN it_vbak s WHERE s.kunnr = s.kunnr.
```

NFAE passes the runtime table of the application server, `it_vbak`, to HANA and then does a semi-join between the table `kunnr` and the previous runtime object `it_vbak`. This is done by leveraging the FDA. Without changing a single line in the business logic, higher speed in the order of a magnitude can be realized.

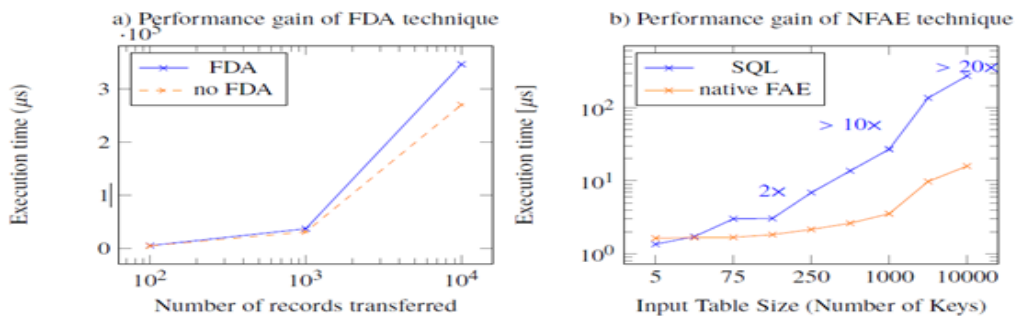


Figure 3: NFAE and FDA optimizations in (May et al., 2017).

From the above discussion, it is evident that HANA provides admirable throughput for OLTP workloads for ERP scenarios. Finally, in discussing the evolution of HANA, there is need to analyze the capability of HANA for mixed workloads.

## **SAP HANA for OLTP and OLAP Workloads**

Motivated by the need to have a tight coupling of OLAP and OLTP under a common data management system and the need to overcome the costs of maintaining and operating two separate systems, HANA extended its capability to handle mixed workloads under the same database. May et al. (2017) presents a number of optimization techniques in the HANA for mixed workloads:

**Resource Management:** Owing to the fact that the analytical queries demand significant memory and CPU usage in order to deliver real-time reporting, thousands of concurrent transactional queries, which ordinarily require low response times, are bound to suffer resource starvation (May et al., 2017). Real-time OLAP should not be realized at the expense of real-time OLTP needed for critical business transacting. HANA overcomes this by resource pooling whereby instead of resource overprovisioning, it discerns instances of where the memory is heavily consumed and terminates the subject OLAP statements. At the same time, HANA uses NUMA (Non Uniform Memory Access) architecture to establish concurrency with respect to the latest utilization of the CPU. If the server has less load, the OLAP query uses in full the CPU resource at its disposal, otherwise there is no parallelization with a huge workload (May et al., 2017). HANA also prioritizes the OLTP-style statements to achieve guaranteed response times required for interactive ERP modules. This is realized by classifying statements during compilation phase into either OLTP or OLAP by getting hints from the application server (May et al., 2017).

**Heavy on-the-fly aggregation:** A big number of aggregate redundant tables are maintained in the traditional ERP which corresponds the business objects like ‘customers per country’ due to limited database performance. This, however, makes the application more complex by maintaining both the aggregates and the actual business logic leading to high maintenance overhead especially with large tables. HANA for mixed workloads replaces the aggregate redundant tables with database views on the parent tables. This reduces the business logic in the code significantly because the views do not have to be maintained during write transactions. As a result, OLAP-based ERP System can now compute the aggregated values on-the-fly. HANA employs Dynamic and Static Caching strategies. The former keeps the static parent version of a database view, then calculates the delta regarding the parent version then consolidates it into a snapshot which is transaction-consistent.

The application code designer is also able to pass hints of caching opportunities to the database by using declarative specification of cached views. The latter, on the other hand, uses a retention time to refresh the cache. It is meant for requests that can bear with data that is a couple of minutes old (May et al., 2017).

**Complex Analytics on Technical, Normalized Schemas:** Executing both OLTP and OLAP transactions on the same physical database schema implies that there is no data warehousing. Especially with the elimination of aggregates in the business logic in the application server, a layered architecture of database views is necessary. Here database views are built on top of lower-level views representing higher-level business objects upon which SQL queries are executed for OLAP systems.

The reporting queries work on the technical, normalized database schema. The layered architecture of views is referred to as Core Data Services views (May et al., 2017). Figure 4 shows the layered architecture of CDS views as nodes and relationship associations and relationships as edges (May et al., 2017).



Figure 4: Nested CDS views relationship (May et al., 2017).

May et al. (2017) argue that in order to overcome the challenge of performing analytical queries on highly complex CDS views where hundreds of base tables have been referenced, applying caching strategy as already discussed in this paper optimizes and efficiently evaluates the queries. The main challenge with the complex queries that seems to remain unresolved is how to characterize complex views and analyze the performance issues. Relying only on the number of tables accessed may be misleading because it does not take into account the size of the accessed table.

**Data Aging:** Data accessed through SQL queries reside in memory and are usually loaded therein by default upon first access ( in full columns). Because keeping data that is rarely accessed in the main memory is expensive, the HANA database classify data as either warm or cold data in what is known as aging run in order to optimize storage.

Cold data is stored in disk and loaded into memory on demand. To detect historical data and prevent it loaded into main memory, HANA employs table partitions with page-loaded columns, relying on application hints indicating which queries need hot data and pruning partitions informed by the runtime statistics of the parameters in the query and columns accessed (May et al., 2017).

## 2.4 The New Programming Model with IMC

In order to get the full benefits of In-Memory database systems such as HANA, there has to be a paradigm shift in programming, from the traditional application code (Data-to-Code) to the new Code-to-Data model as shown in Figure 5. HANA realizes this by the availability of SQL Script interface. It includes SELECT queries, Data Definition Language (DDL), and Data Manipulation Language (DML). Heilman et al. (2013) identify two main reasons for SQL Script as the desire to prevent the transfer of huge amounts of data from the database to the application layer and the need to perform calculations in the database to leverage HANA's query optimization, parallel execution and fast column operations. Compared to the traditional SQL queries, SQL Script is capable of the following capabilities: modular programming by breaking down complex functions into smaller ones enabling reuse and functional abstraction, returning multiple results, availability of control logic for instance if/else as well as support for local variables for intermediate results.

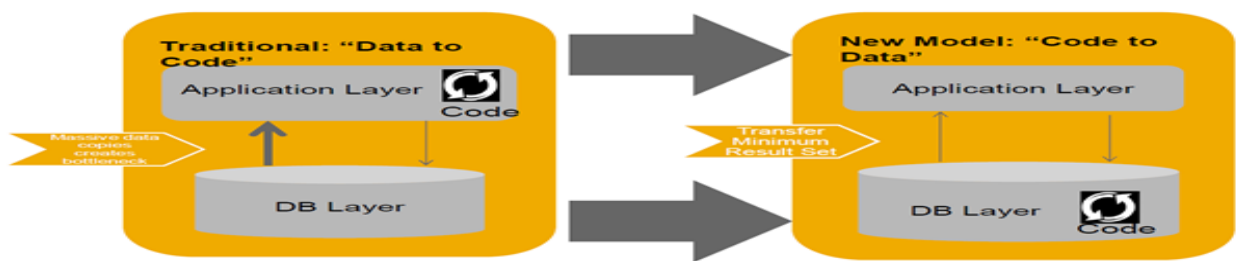


Figure 5: Traditional Data-to-Code model vs New Code-to-Data model (Heilman et al., 2013).

SAP ABAP, the main programming language for SAP ERP, has been optimized for HANA to ensure maximum benefits are reaped from the new Code-to-Data model. SAP ABAP code (needed for data processing) is pushed down to HANA and less data is transferred between HANA and ABAP. Figure 6 shows the scenario in comparison classic databases.



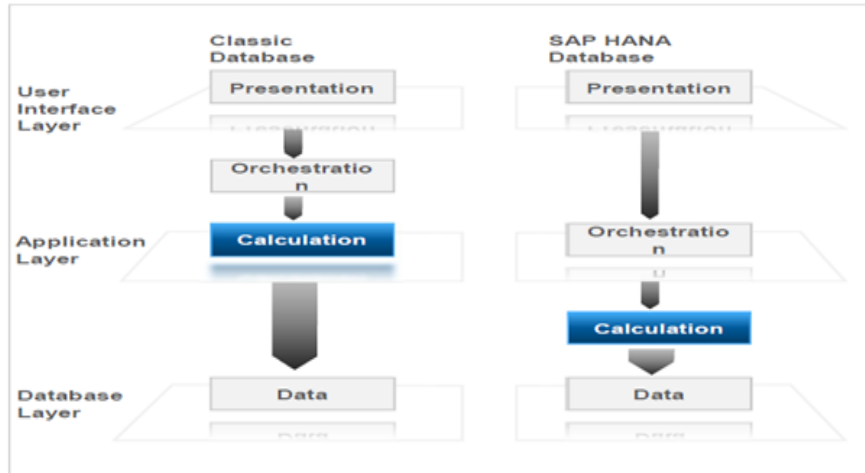


Figure 6: Code Pushdown with Application Server ABAP (Heilman et al., 2013).

## 2.5 Literature Summary

With the advent of multicore CPUs and prices of main memory increasingly dropping, organizations using ERP systems have to adopt these technologies if they want to perform real-time analytics on transactional data. A number of in-memory databases have emerged in the last few years reducing the current problem of higher response times with the disk I/O processing. We have analyzed Hyper and HANA in-memory databases, which both use columnar storage to enable OLAP queries without compromising the OLTP queries. They both employ data tiering, classifying data into clusters of hot, warm and cold in order to manage the memory resources. Owing to the fact that Hyper's case has not been proven to work efficiently with ERP systems, this paper took a deeper look at HANA developed by SAP. Since SAP is the world's leader in ERP solutions, HANA comes as a big relief for its global customers who have had the perennial problem working with stale information with the classical databases. The HANA innovation has also introduced a new programming paradigm, namely the Code-to-Data model, that allows code pushdown to HANA for complex calculations and processing.

## 2.6 Overall System Architecture

The following system design architecture represents the researcher's view of the problem and the solution:

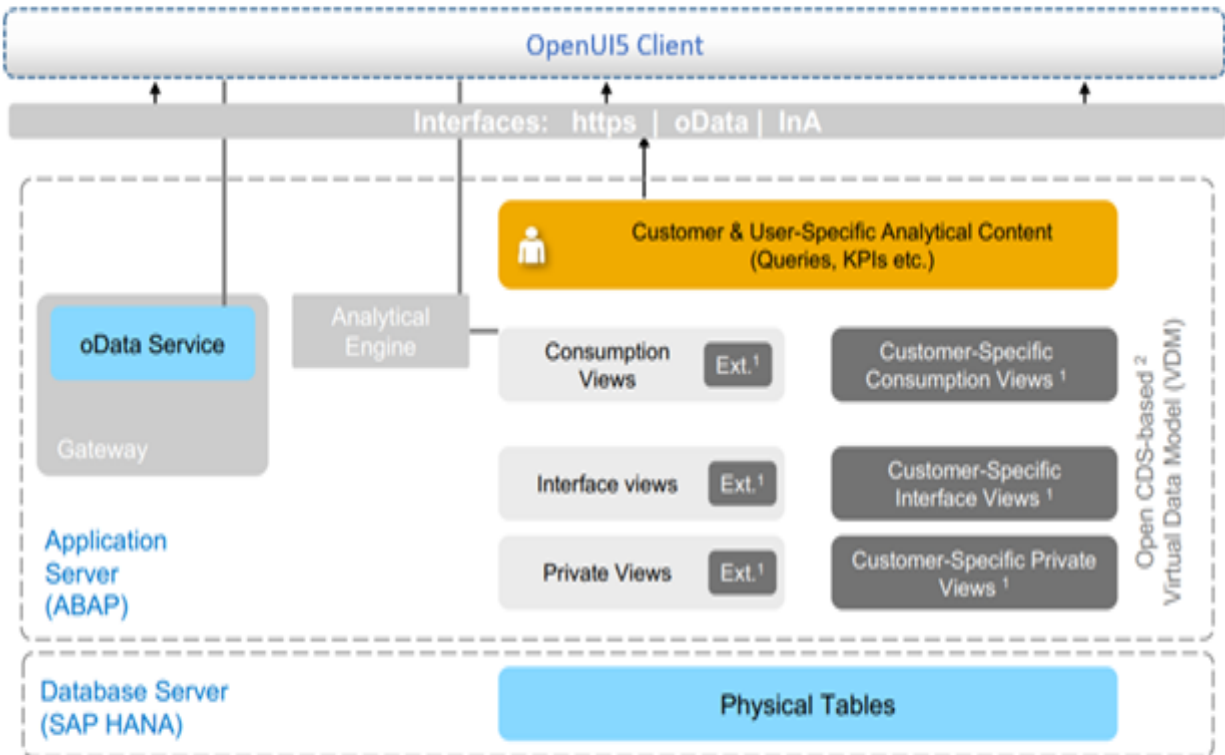


Figure 7: Overall System Architecture (Pattanayak, 2017).

The real-time analytics dashboard is based on the HANA database where database tables reside and data is persisted. In the Application Server, Virtual Data Models are created in form of CDS views (Pattanayak, 2017). At the lowest level is the Private Views, which interact with the physical tables by fetching data directly from the HANA database. On top of the Private Views are the Interface Views, which interact with the database through a set of Private Views through database Joins or Associations (Pattanayak, 2017). It is also at this point where calculations on the retrieved data is done. The ultimate views are the Consumption Views, which are created on top of the Interface Views (Pattanayak, 2017). They are used to expose the data to OpenUI5 client. OpenUI5 is an open source MVC Framework comprising HTML5, JavaScript, CSS and JSON tools. The OpenUI5 client represents the prototype analytics dashboard to be developed. OData services are then created from the Consumption Views and exposed to the OpenUI5 client through the Secure HTTP protocol.

# CHAPTER THREE

## RESEARCH METHODOLOGY

### 3.1 Introduction

This chapter discusses the research design to be used in conducting the study. It gives details on research strategy.

### 3.2 System Development Methodology

This is a problem solving approach that entails the recognition and articulation of the research problem, a suggestion of how the problem might be addressed, and the implementation of the prototype. This study engaged the Design-Led Development Process (DLD) methodology. It has four main phases – Discover, Design, Develop and Deliver:

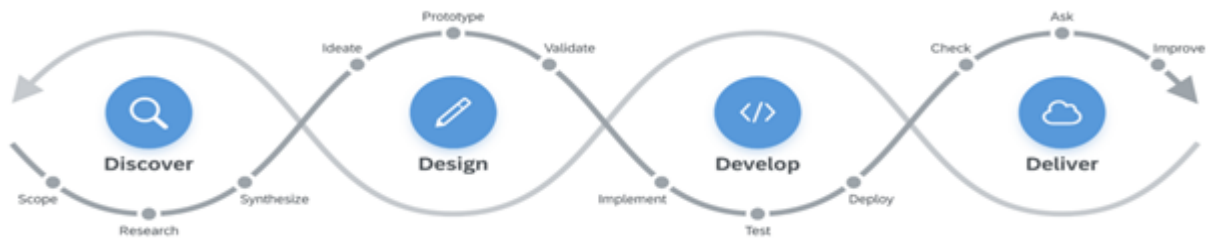


Figure 8: Design-Led Development Process (SAP SE.2020.<https://experience.sap.com>)

This is the best methodology for prototyping a solution to our problem because it is not necessary to fully understand the problem before exploring alternative solutions. It would also give more confidence to have a tangible implemented prototype early on. In designing and implementing the prototype dashboard, the following iterative steps were followed: Discover, Design, Develop, and Deliver.

#### Discover

This stage assumes knowledge of what the business wants and needs. To get relevant insights, an onsite visit to the analytical users and everyone involved and observing their typical daily work is needed. At the end of this activity, it should be clear what exact dashboard should be developed for the prototype.

## Design

With the findings from the discover phase, this phase requires brainstorming in a team as well as designing an initial prototype to be validated with the end users (SAP SE.2020.<https://experience.sap.com>). By the end of this phase, there is a design ready to be implemented. At this phase, control flow diagrams, flowcharts and pseudocodes are created. The analytics dashboard displays, in various chart layouts, four main sales Key Performance Indicators (KPIs) namely: a) Total Sales per Customer in a Column Chart b) Total Sales per Product in a Bar Chart c) Total Sales per Country in a Map View and d) Total Sales per Currency in a Pie Chart

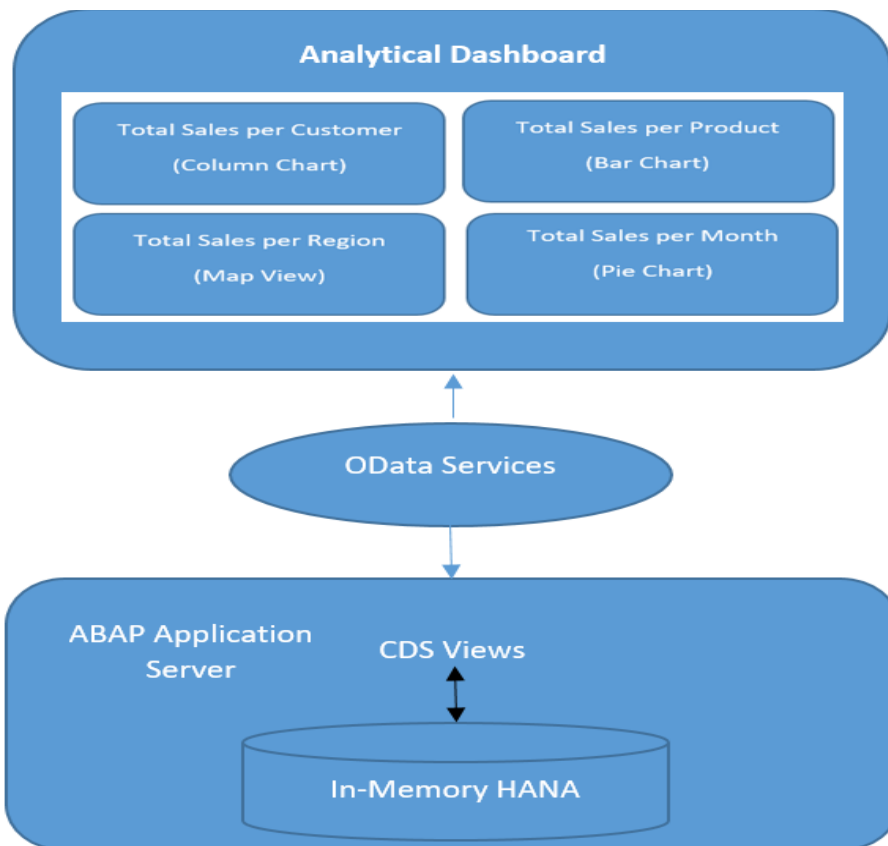


Figure 9: The Prototype Dashboard Design

The real-time analytics dashboard is based on the HANA database where database tables reside and data persisted. In the Application Server, Virtual Data Models are created in form of CDS views (Pattanayak, 2017). At the lowest level is the Private Views, which interact with the physical tables by fetching data directly from the HANA database. On top of the Private Views are the Interface Views, which interact with the database through a set of Private Views through database Joins or Associations (Pattanayak, 2017).

It is also at this point where calculations on the retrieved data is done. The ultimate views are the Consumption Views, which are created on top of the Interface Views (Pattanayak, 2017). They are used to expose the data to OpenUI5 client. OpenUI5 is an open source MVC Framework comprising HTML5, JavaScript, CSS and JSON tools. The OpenUI5 client represents the prototype analytics dashboard to be developed. OData services are then created from the Consumption Views and exposed to the OpenUI5 client through the Secure HTTP protocol.

The way CDS annotations translates to business semantics in a metadata driven analytics user interface is as shown below:

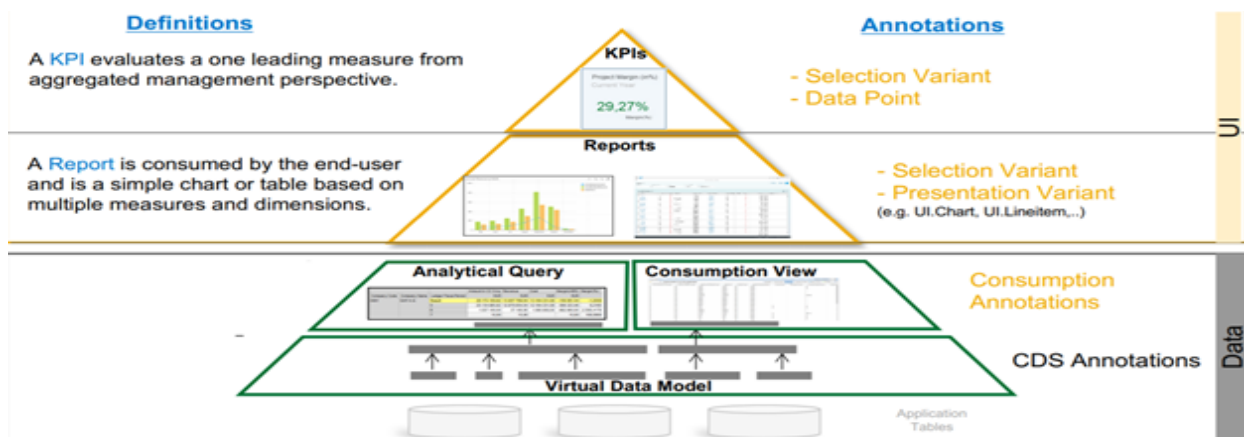


Figure 10: The Proposed Analytics User Interface

## Develop

This phase starts as soon the initial prototype dashboard design is finished and entails the development, testing and deployment of the prototype. Some aspects of the design may still need to be changed in light of ongoing users' feedback, to optimize the dashboard or to allow for technical constraints. For the delivery of the dashboard, a verification that the final implementation and design delivers the intended user experience is carried out. By the end of this phase, the prototype is ready for delivery and evaluation.

**Development, Integration & Deployment Tools:** The project is developed and tested in Cloud Foundry – an Open Source Cloud Platform using the trial version of SAP HANA in-memory database. It is developed using the SAP Business Application Studio, a Platform-as-a-Service tool that provides a unified multi-cloud environment to develop, test, build, run and deploy the

prototype. Eclipse IDE (Integrated Development Environment) is used for data modelling as well as for the creation of Core Data Services and exposing them through the OData services via the gateway. The front-end dashboard is developed using OpenUI5 framework comprising HTML5, JavaScript, CSS and JSON.

The Business Application Studio gives provision for a virtual machine known as a Dev Space. This is the development environment. The user interface of the analytics dashboard is developed using the UI Development Toolkit for HTML5 (SAPUI5). This is an extensible JavaScript-based HTML5 control rendering library for business applications that run in the browser (SAP SE.2020.<https://experience.sap.com>). Web and Eclipse-based tools are available to support SAPUI5 development. SAPUI5 is optimized to make use of SAP metadata exposed in the OData stream. It provides an implementation of the MVC design pattern. The Model manages the application data, the View defines and render the UI and the Controller modifies the view and model.

The database CDS views are rendered through OData. This is an Open Data Protocol based on HTTP, Atom Pub format, and JSON. It enables provision of data services-based REST principles and defines data queries using URLs constructed with specific rules (SAP SE.2020.<https://experience.sap.com>). It was released under “open specification promises” by Microsoft.

**OData URL structure:**

[http://services.odata.org/OData/OData.svc/Category\(1\)/Customers?\\$top=2](http://services.odata.org/OData/OData.svc/Category(1)/Customers?$top=2)

**Service Document:**

The service document (returned at the OData service root) gives the titles and URLs for each service feed:

[http://services.odata.org/V3/OData/OData.svc/\\$metadata](http://services.odata.org/V3/OData/OData.svc/$metadata)

The \$metadata entry for an OData service returns an EDMX file (Entity Data Model XML) that contains a complete description of the feeds, types, properties, and relationships exposed by the OData service.

### **OData - System Query Options:**

**\$orderby** - order entries by the entity supplied, example: `/Products?$orderby=Rating,desc`

**\$stop** - selects only the first n items in a collection, example: `/Products?$stop=5`

**\$skip** - selects entries starting with n+1, example: `/Products?$skip=2`

**\$filter** - filters entries based on the criteria, example: `/Suppliers?$filter=Address/City eq 'Redmond'`

**\$expand** - expanded entries are loaded quickly and presented inline, `/Categories?$expand=Products`

**\$format** - defines the format that the server must return, example: `/Products?$format=json`

**\$select** - returns the subset of the specified properties, example: `/Products?$select=Price,Name`

The dashboard web application runs on all the major web browsers as well as all mobile devices irrespective of the underlying operating systems.

**Testing Techniques:** In order to establish whether the prototype is capable of functioning in a real-life context, white box and black box testing techniques is carried out.

White box testing checks for the correctness of code and the OData APIs as well as adherence to best practices. Black box on the other hand, tests the data input and output including the response times, CPU utilization and the usability of the user interface. The prototype is tested for its latency in providing real time analytics. The performance metrics to be used for this experiment are Average Response Time , Average Throughput and the CPU Time of SQL statements execution.

**Deployment:** The prototype dashboard is deployed in the Cloud Foundry – an Open Source Cloud Application Platform using the trial SAP HANA in-memory database. It is deployed as Software as a Service (SaaS) in form of a web application, accessible through mobile, laptop and desktop devices.



Figure 11: Development and Deployment Process Steps

## **Deliver**

This phase makes the dashboard available to the end-users. It also involves an observation and investigation of how the dashboard makes analytical faster and easier, if it is user friendly and if there are any functions and features missing. The outcome of this observation and investigation feeds back into the Design-Led Development process and informs the next iteration of the solution.

### **3.3 Evaluation & Analysis**

Evaluating the prototype involves a case study and a survey within the chosen case.

#### **3.3.1 Sampling Procedure and Sampling Size**

Of the businesses that run ERPs and with the need for real-time analytics, Janus Continental Group has been selected for this research project. The choice of Janus Continental Group as a case study is based on convenience because the company has agreed to give us access and is convenient as far as time and resources are concerned. The sales department has confirmed willingness and contribution of data for use in this project. This case is, however, typical of other organizations that run enterprise resource planning systems with a sales & distribution module. Therefore, the findings here are generalized to other similar organizations.

#### **3.3.2 A Case Study of Janus Continental Group**

The final prototype is evaluated in a real-world context in form of a short-term case study at the Janus Continental Group, within the sales department with 20 users. These are users mandated to conduct sales planning and performance management.

#### **3.3.3 Data Generation Methods**

This study uses the secondary data in the form of the sales data already captured in the ERP Test system through the Sales and Distribution OLTP module. The data include customer master data, sales orders, and delivery particulars. Real-time analytics is done on this data, which is estimated at millions of records. The analytics result set is then rendered on the prototype dashboard. Using data modeling, the CDS views extract the operational data into analytic cubes from where they can be exposed through the OData APIs. The CDS views collect data in three main categories :a) Measures: are quantifiable values that can be subjected to mathematical functions.



For example, sales and sales price b) Dimensions: are qualitative values that do not have aggregate requirements. For example, sales country, product name, and customer c) Cubes: are a combination of both dimensions and measures. For example, salesperson, sales price, product name, country and currency. The CDS views based on the above is used as data sources for real-time analytics.

User evaluation of twenty potential end-users is conducted to establish the conclusion of the design process and to determine if further modification to the prototype is needed. This is done through observations, interviews and questionnaires to evaluate functionality, performance, reliability, usability, and accessibility. **Observations:** is used to watch how the sales users work .The prototype may then be adjusted accordingly. An observation of how the prototype is used is also necessary. **Interviewing & structured questionnaires:** is used to gather requirement specifications as well as for the evaluation of the prototype dashboard. **Documents analysis and creation:** a study of the organization's standard operating procedures and the job descriptions gives insights to what extent the prototype works in the organization. Analysis and design models, testing logs, user guides and a personal journal are also created and maintained.

The evaluation of the developed prototype is carried out in the business context in order to assess the effectiveness of the analytics dashboard in the business environment. The evaluation metrics focus on the following key areas: the simplicity and ease of use, the adaptability to all the devices and web browsers, enrichment of the potential user's work by providing useful sales insights as well as the provision of real-time and actionable sales insights

The methodology for assessing the above is qualitative in nature. Questionnaires is administered with the focus of assessing the functional requirements of the prototype.

The study is conducted for one week. The questionnaire to be used in the interview can be found in under Appendix III.

The responses to the Likert-type survey questions are coded as follows:

- a) Strongly Agree - 5
- b) Agree - 4
- c) Not sure - 3
- d) Disagree - 2
- e) Strongly Disagree - 1

The survey questions are grouped into four main categories:

- a) Simplicity: evaluates the ease of use
- b) Adaptability: evaluates how adaptable the prototype is to all the devices and web browsers
- c) Delightfulness: assesses how the prototype enriches of the potential users' work by providing useful sales insights
- d) Latency: evaluates the prototype's response time from the business point of view

### **3.3.4 Specifications and Analysis**

Requirement specifications collected from face-to-face interviews with the end-users and documents analysis informed the iterative design of the prototype. These are analyzed qualitatively resulting in sketches and mock-ups that guided the discussions and further design. The responses to Likert-typed questions are summarized by use of mean (average) to measure the central tendency.

The HANA Studio is used to analyze the runtime statics of the prototype in evaluating:

The Average Response Time - database query response time (in milliseconds) and the CPU Time of SQL statements execution (in microseconds). The runtime statistics obtained are used as input to the mathematical calculation in predicting the response time using Amdahl's law, Response time  $(s) = S + P/N$ .

Tables, pie charts and line graphs are used to visually present the responses and results of the prototype evaluation.

### **3.3.5 Target Population**

The target population for this research project is the businesses or organizations running ERPs, especially those with Sales & Distribution modules. It is also important for System Architects who have an interest in code-to-data paradigm, where data intensive operations are pushed down to the database instead of to the application server.

## **CHAPTER FOUR**

### **IMPLEMENTATION, RESULTS & DISCUSSION**

#### **4.1 Introduction**

This chapter demonstrates the results and discussion from the dashboard prototype applicable to this research project. In order to carry out the evaluation of the In-Memory Computing, the following specific objectives were set:

- a) To measure the performance of the IMC by evaluating the average database query response time in milliseconds.
- b) To assess the functionality, usability, and accessibility of the dashboard prototype in the business context

#### **4.2 Specifications and Analysis**

The specification for the analytical dashboard followed the Design-Led Development methodology. The objective was to design and develop an analytical dashboard that leverages HANA, an in-memory database. The dashboard was meant for the E-Commerce industry, specifically organizations operating on ERPs with a Sales Module. It was designed to improve sales planning and performance by providing real-time and actionable sales insights to the Sales Team. The potential users of the dashboard were engaged to get more clarity on the business reality. Figure 13 is a persona that represents the main stakeholder, his business objectives and the tools he uses to achieve the objectives. It guided the design decisions by representing the goals and desires of the dashboard users.

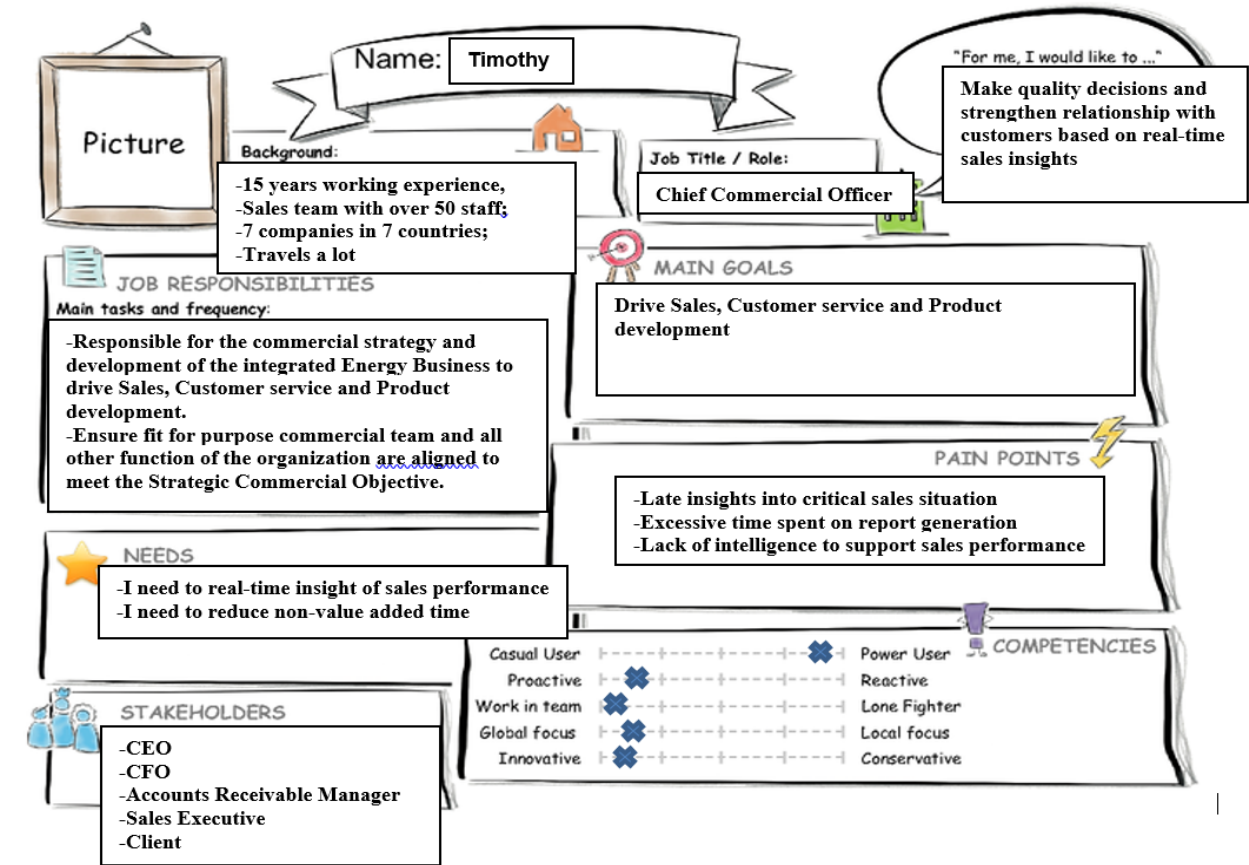


Figure 12: Persona for the dashboard prototype

### 4.3 Inputs and Outputs

The following table represents the data input used for analytics as well as the various dimensions and measures of the analytic dashboard:

Table 1: Inputs & Outputs

Data Type	Data Example	Particulars	Access Frequency
Master Data	Customers		Medium
	Products		
	Countries		
	Address		
Transaction Data	Sales Orders		High

#### 4.4 The Design of the Open CDS-based Virtual Data Modelling in HANA

The standard operational HANA database tables VBAK, VBAP, BUT000, ADR2 and MAK2 were leveraged in building the virtual data model. VBAK contains all the sales header information, VBAP contains all the sales items information, BUT000 contains the customer information, ADR2 contains the country information and MARA contains the product details.

The CDS-based Virtual Data Model is as shown below:

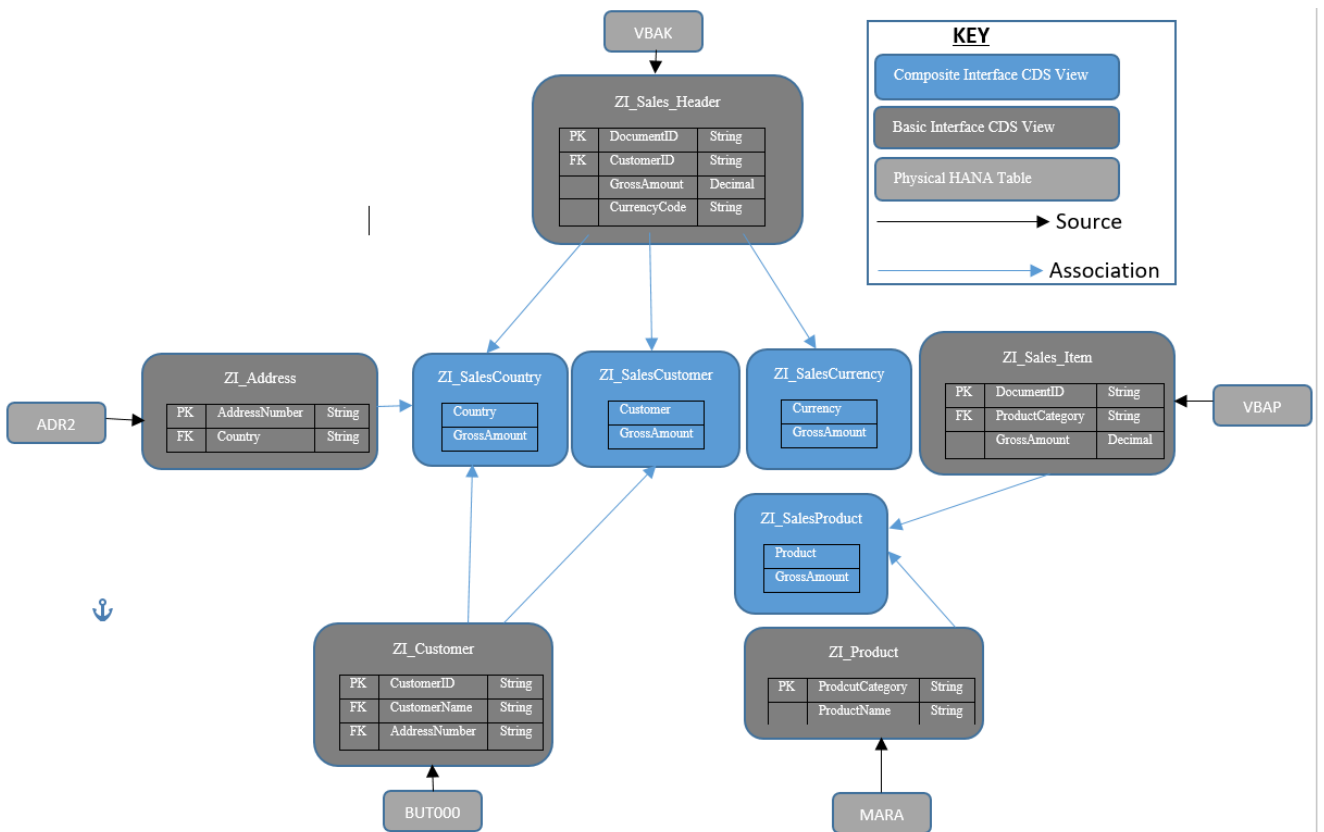


Figure 13: The Design of the CDS-based Virtual Data Model

ABAP Development Tools (ADT) in Eclipse is used to model the CDS views as well as to connect to the ERP system.

The OData service entity is designed in the S/4 HANA system using the Gateway Service Builder tool. This result in four entity sets namely; Total Sales per Customer, Total Sales per Product, Total Sales per Country and Total Sales per Currency. These entity sets are populated by data from the CDS views created from the physical HANA tables.

## 4.5 Runtime Statistics of the Prototype

The performance tests were implemented using the Google Browser Performance and Network Analysis tool in combination with the HANA CPU Cores Sizing tool. Whereas the latter was used to size the HANA database into instances of 8 CPU cores and 32 CPU cores, the former was used to monitor the performance logs that give specific response times for each sized instance.

### Average Response Time

This is the database query response time in milliseconds. Shorter response time is made possible by the IMC's ability to constantly cache countless amounts of data. Data stored in cached memory has lower latency. Response time is analyzed as a distribution as HANA interacts with other dynamic systems like the Operating System, the Network as well as the Application Servers. The response time is measured on two critical conditions: no other queries are running in parallel and the HANA CPU capacity is available to the analytical query being measured at the execution time. The query was measured with simulation of 32 and 16 CPU Cores, with the following results:

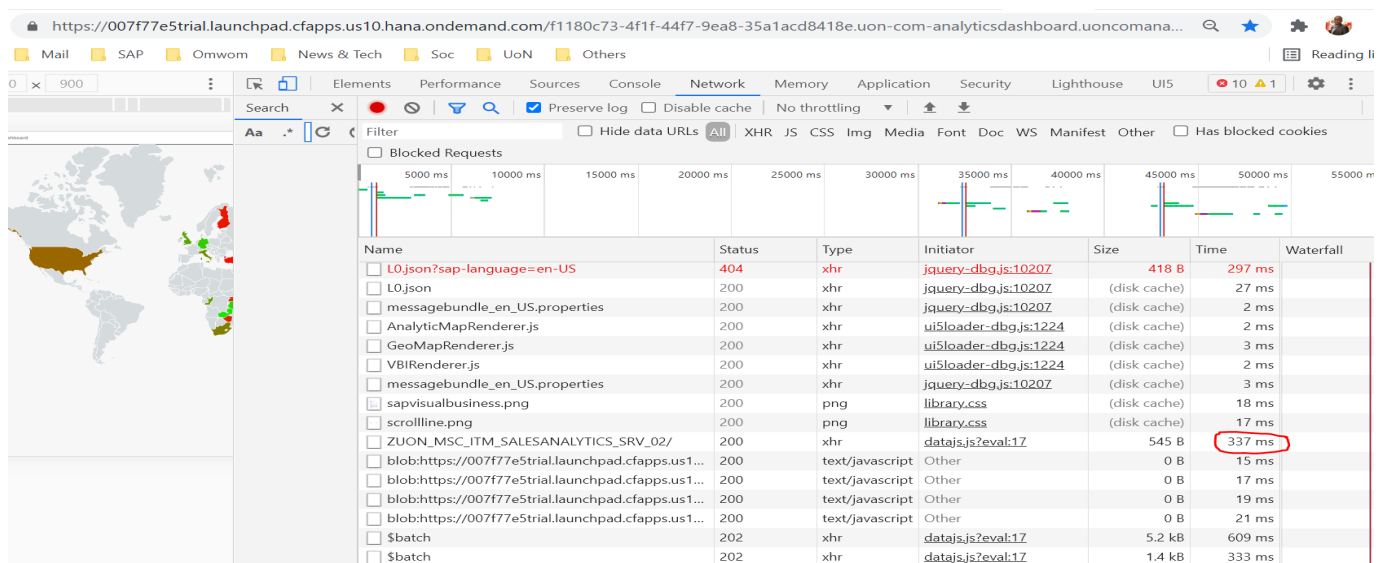


Figure 14: Response time with an IMC on 32 CPU Cores

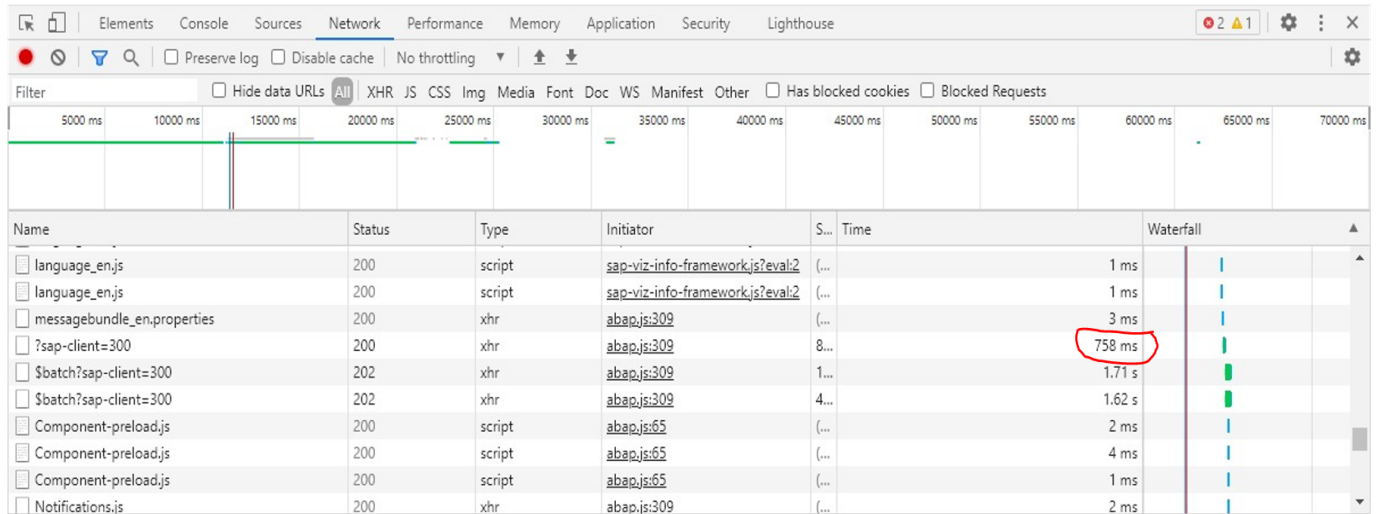


Figure 15: Response time with an IMC on 8 CPU Cores

Table 2: Response times with an IMC on 8 & 32 CPU Cores

CPU Cores	Measured Response Time	Measured Response Time
	(ms)	(s)
8	758	0.758
32	337	0.337

Using Amdahl's law, the response times for 1, 2, 4, 16, 64 and 128 CPU Cores can be predicted.

Amdahl's law states that;

$$\text{Response time (s)} = S + P/N$$

Where S is the sequential part of the analytical query execution flow, P is the parallelizable part of the analytical query and N is the number of simulated CPU Cores.

Applying the formula to 8 CPU Core gives;  $0.758 = S + P/8$

Applying the formula to 32 CPU Core gives;  $0.337 = S + P/32$

From the equations,

$$S = 0.197$$

$$P = 4.488$$

With the values of S and P, the calculated estimated response times for 1, 2, 4, 16, 64 and 128 CPU cores as shown below:

Table 3: Response times with IMC on CPU Cores between 1 and 128

CPU Cores	Calculated Response Time (s)
1	4.685
2	2.441
4	1.319
8	0.758
16	0.478
32	0.337
64	0.267
128	0.232

The results of how response times varies with respect to the number of CPU cores can therefore be plotted as follows:

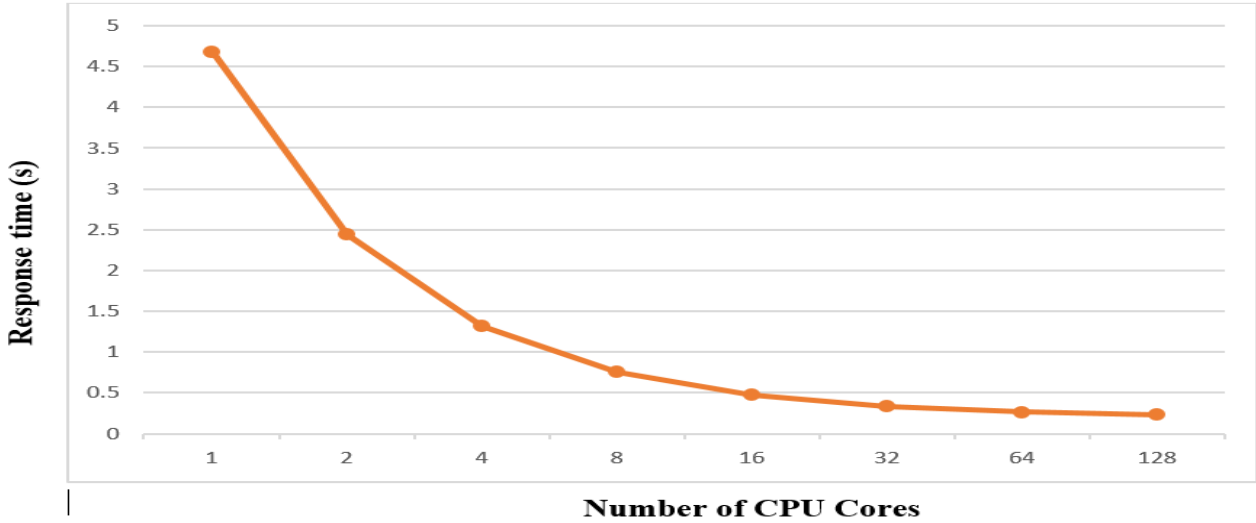


Figure 16: The Analytics Dashboard’s Average Response Time



From the results, it is evident that with the CPU Cores of more than 16 the response times do not significantly change. Therefore, businesses do not need to spend more in CPU acquisition except for when concurrent load is required and the CPU bandwidth to the memory bus notwithstanding.

#### 4.6 Effectiveness of the In-Memory Computing in the organizational context

The evaluation of the functionality, usability and accessibility of the prototype was done to establish its relevance in the business context. A survey was conducted using google forms to collect both qualitative and quantitative data.

The evaluation commenced by introduction and making known to the business the overall objective of the research project. The prototype was configured and deployed for business use. The business units targeted for the survey included Oil & Gas, Real Estate, Hospitality, Energy, Construction, Investment as well as Shared Services as shown below:

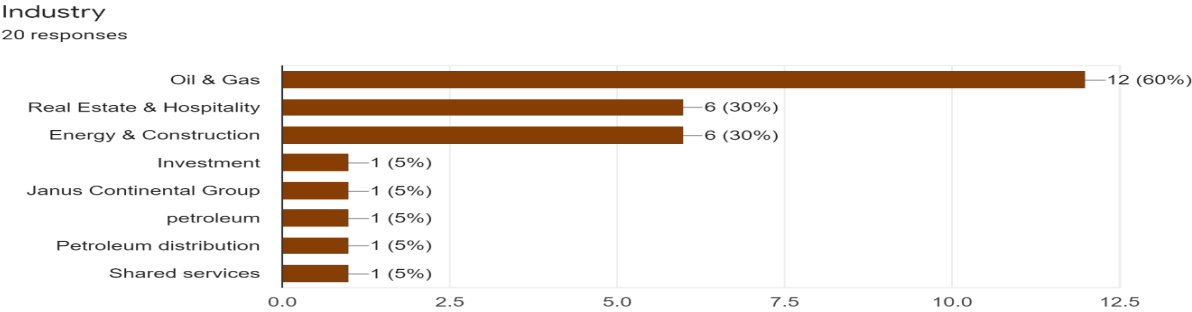


Figure 17: The Industry Distribution in the Survey on using the IMC-based Analytics Dashboard

The sales department used the dashboard for a period of 2 weeks after which the survey questionnaires were issued out. It took 5 days to have the responses back.

Out of the 33 targeted users, 20 participated in the survey. Over 90% corroborated that the analytics dashboard provides real-time & actionable sales insights, that the analytics dashboard enriches their work and that the analytics dashboard is easy and simple to use. 80% of the users agreed that the dashboard could be accessed on all devices and web browsers.

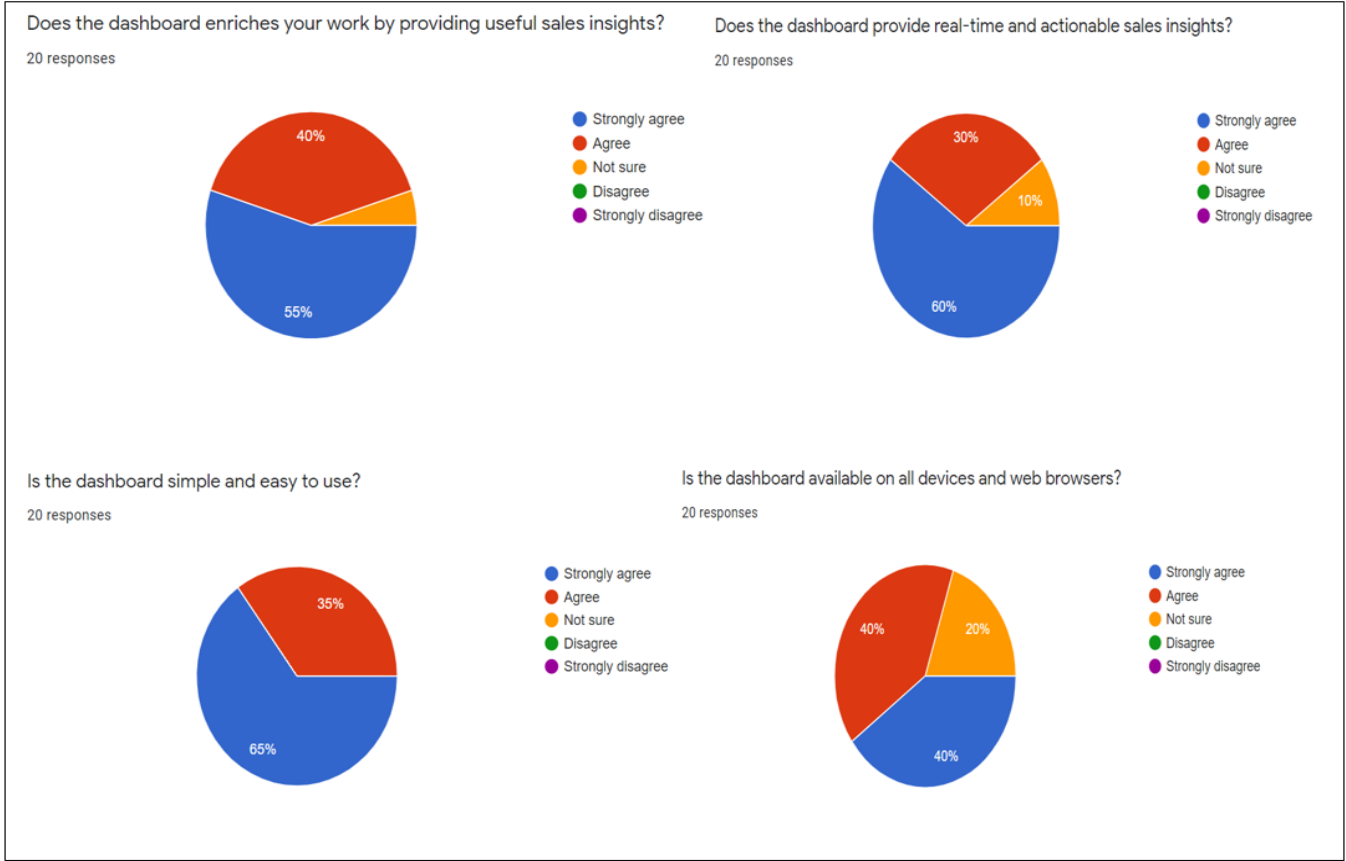


Figure 18: Results of the Effectiveness of the In-Memory Computing in the organizational context

The highest ranked feature was the dashboard’s simplicity in providing real-time and actionable sales insights. The ‘Not Sure’ Likert-type was ranked the lowest in all the categories, pointing to a very small number of users who may not have devoted enough time to evaluate the prototype. Whereas all the questions that the potential users were asked to respond to are found in the appendices section of this report, some of them are as shown in the figure 18 above.

# **CHAPTER FIVE**

## **SUMMARY, CONCLUSION AND RECOMMENDATIONS**

### **5.1 Introduction**

This research project had four objectives. Firstly, to examine the capabilities of In-Memory Computing. Secondly, to design and develop a prototype dashboard that leverages an IMC for real-time analytics. Thirdly, to experiment the application of code-to-data paradigm. Fourthly, to evaluate the latency in data processing from the prototype in the business context.

### **5.2 Linking the Study Findings to Objectives**

#### **Objective 1: To examine the capabilities of In-Memory Computing**

With the advent of multicore CPUs and prices of main memory increasingly dropping, organizations using ERP systems have to adopt these technologies if they want to perform real-time analytics on transactional data. A number of in-memory databases have emerged in the last few years reducing the current problem of higher response times with the disk I/O processing. This research project has analyzed the HANA in-memory database which uses columnar storage to enable OLAP queries without compromising the OLTP queries. Owing to the fact that the analytical queries demand significant memory and CPU usage in order to deliver real-time reporting, thousands of concurrent transactional queries, which ordinarily require low response times, are bound to suffer resource starvation. Real-time OLAP should not be realized at the expense of real-time OLTP needed for critical business transacting. HANA overcomes this by resource pooling whereby instead of resource overprovisioning, it discerns instances of where the memory is heavily consumed and terminates the subject OLAP statements. . If the server has less load, the OLAP query uses in full the CPU resource at its disposal, otherwise there is no parallelization with a huge workload(May et al., 2017).. HANA also prioritizes the OLTP-style statements to achieve guaranteed response times required for interactive ERP modules. This is realized by classifying statements during compilation phase into either OLTP or OLAP by getting hints from the application server.

HANA for mixed workloads overcomes maintenance of a big number of aggregate redundant tables in the traditional ERP which corresponds the business objects like ‘customers per country’ due to limited database performance (May et al., 2017).

This reduces the application code significantly because the views do not have to be maintained during write transactions. As a result, OLAP-based ERP System can now compute the aggregated values on-the-fly (May et al., 2017). With the elimination of aggregates in the business logic in the application server, a layered architecture of database views is necessary. Here database views are built on top of lower-level views representing higher-level business objects upon which SQL queries are executed for OLAP systems. The reporting queries work on the technical, normalized database schema. The layered architecture of views is referred to as Core Data Services views.

In order to overcome the challenge of performing analytical queries on highly complex CDS views where hundreds of base tables have been referenced, applying caching strategy optimizes and efficiently evaluates the queries.

HANA employs Dynamic and Static Caching strategies. The former keeps the static parent version of a database view, then calculates the delta regarding the parent version then consolidates it into a snapshot which is transaction-consistent. The application code designer is also able to pass hints of caching opportunities to the database by using declarative specification of cached views. The latter, on the other hand, uses a retention time to refresh the cache. It is meant for requests that can bear with data that is a couple of minutes old (May et al., 2017). Data ageing is another capability of the HANA IMC. Data accessed through SQL queries reside in memory and are usually loaded therein by default upon first access ( in full columns). Because keeping data that is rarely accessed in the main memory is expensive, the HANA database classify data as either warm or cold data in what is known as aging run in order to optimize storage. Cold data is stored in disk and loaded into memory on demand. To detect historical data and prevent it loaded into main memory, HANA employs table partitions with page-loaded columns, relying on application hints indicating which queries need hot data and pruning partitions informed by the runtime statistics of the parameters in the query and columns accessed. In order to get the full benefits of In-Memory database systems such as HANA, there has to be a paradigm shift in programming, from the traditional application code (Data-to-Code) to the new Code-to-Data model. HANA realizes this by the availability of SQL Script interface which includes SELECT queries, Data Definition Language (DDL), and Data Manipulation Language (DML). The SQL Script prevents the transfer of huge amounts of data from the database to the application layer and the need to perform calculations in the database to leverage HANA's query optimization, parallel execution and fast column operations.

Compared to the traditional SQL queries, SQL Script is capable of modular programming by breaking down complex functions into smaller ones enabling reuse and functional abstraction, returning multiple results, availability of control logic for instance if/else as well as support for local variables for intermediate results.

**Objective 2: To design and develop a prototype dashboard that leverages an IMC for real-time analytics**

The IMC-based analytics dashboard was designed and developed using the agile-based Design-Led Development Process (DLD) to ensure it provides a great user experience. The virtual data modelling and the algorithms were implemented in Cloud Foundry – an Open Source Cloud Platform using the trial version of SAP HANA in-memory database. The development was done using the SAP Business Application Studio, a Platform-as-a-Service tool that provides a unified multi-cloud environment to develop, test, build, run and deploy the prototype. Eclipse IDE (Integrated Development Environment) was used for data modelling as well as for the creation of Core Data Services and exposing them through the OData services via the gateway. The front-end dashboard is developed using OpenUI5 framework comprising HTML5, JavaScript, CSS and JSON.

**Objective 3: To experiment the application of code-to-data paradigm**

An experiment was carried out to establish the IMC response time as well as the CPU core requirement for real-time data visualization. The average response time was measured on the conditions that no other queries were running in parallel and the HANA CPU capacity was available to the analytical query being measured at the execution time. The query was measured with simulation of 16 and 32 CPU Cores. The response times for 1, 2, 4, 16, 64 and 128 CPU Cores were calculated using Amdahl's law of response. Plotting response times against the corresponding the number of CPU cores corroborated that the CPU Cores of more than 16 the response times do not significantly change. As such, organizations intending to use In-memory Computing technology such as HANA do not need to spend more in CPU acquisition except for when concurrent load is required and the CPU bandwidth to the memory bus notwithstanding.

#### **Objective 4: To evaluate the latency in data processing from the prototype in the business context**

The IMC-based analytics dashboard was subjected to an evaluation exercise from the sales department. The evaluation involved a test of the prototype's functionality, usability and accessibility to establish its relevance in the business context. The prototype was configured and deployed for business. The sales department used the dashboard for a period of 2 weeks after which the survey questionnaires were issued out. It took 5 days to have the responses back. Out of the 33 targeted users, 20 participated in the survey. Over 90% corroborated that the analytics dashboard provides real-time & actionable sales insights, that the analytics dashboard enriches their work and that the analytics dashboard is easy and simple to use. 80% of the users agreed that the dashboard could be accessed on all devices and web browsers. The highest ranked feature was the dashboard's simplicity in providing real-time and actionable sales insights. The 'Not Sure' Likert-type was ranked the lowest in all the categories, pointing to a very small number of users who may not have devoted enough time to evaluate the prototype.

#### **5.3 Conclusion**

This research study established that real-time analytics and reporting is realized by in-memory technology's high-end computing performance. With in-memory computing there is one single source of data i.e. no separation between the operational and analytical data. Real time information ensures continuous business transparency. Except for when concurrent load is required and the CPU bandwidth to the memory bus notwithstanding, organizations intending to use in-memory computing technology such as HANA do not need to spend in CPU acquisition of more than 16 CPU cores. The study corroborates that the computing response times do not significantly change beyond the aforementioned number of CPU cores.

#### **5.4 Limitations of the study**

While undertaking this research project, a number of limitations were face. Firstly, the research was done using the HANA in-memory technology and as such we did not look at other models of in-memory computing. Secondly, the in-memory computing was simulated with only 16 and 32 CPU cores due to limited resources.

#### **5.5 Recommendations for future research**

Owing to the aforementioned limitations, there are a few recommendations for future research from this study. It would be important to evaluate the average response times of the real-time analytics based on other models of in-memory computing other than HANA, for instance the Hyper System. Another area of interest is the assessment of the average response times of the in-memory computing in a multicore and parallel computing environment. The main challenge with the complex queries that seems to remain unresolved is how to characterize complex views and analyze the performance issues. Relying only on the number of tables accessed may be misleading because it does not take into account the size of the accessed table.

## REFERENCES

1. Agostino, R. (2004). Business intelligence: Solving the ERP overload.
2. Baboo, S. S., & Kumar, P. R. (2013). Next generation data warehouse design with OLTP and OLAP systems sharing same database. *International Journal of Computer Applications*, 72(13).
3. Butuza, A., Hauer, I., Muntean, C., & Popa, A. S. (2011). Increasing the Business Performances using Business Intelligence. *Analele Universitatii" Eftimie Murgu" Resita Fascicola de Inginerie*, 3(XVIII), 67-72.
4. Färber, Franz & May, Norman & Lehner, Wolfgang & Große, Philipp & Müller, Ingo & Rauhe, Hannes & Dees, Jonathan. (2012). The SAP HANA database - An architecture overview. *IEEE Data Eng. Bull.* 35. 28-33.
5. Funke, F., Kemper, A., & Neumann, T. (2012). Compacting transactional data in hybrid OLTP & OLAP databases. arXiv preprint arXiv:1208.0224.
6. Heilman, R., & Product, S. H. (2013). SAP HANA SQLScript Basics, Debugging, and ABAP Connectivity. SAP HANA Product Management, SAP Labs, LLC, 1-41.
7. Hill, M. D., & Marty, M. R. (2008). Amdahl's law in the multicore era. *Computer*, 41(7), 33-38.
8. May, N., Böhm, A., & Lehner, W. (2017). SAP HANA—The Evolution of an In-Memory DBMS from Pure OLAP Processing Towards Mixed Workloads. *Datenbanksysteme für Business, Technologie und Web (BTW 2017)*.
9. Pattanayak, Abani. (2017). SAP S/4HANA Embedded Analytics: An Overview. *Journal of Computer and Communications*. 05. 1-7. 10.4236/jcc.2017.59001.
10. Reddy, G. S., Srinivasu, R., Rao, M. P. C., & Rikkula, S. R. (2010). Data Warehousing, Data Mining, OLAP and OLTP Technologies are essential elements to support decision-making process in industries. *International Journal on Computer Science and Engineering*, 2(9), 2865-2873.
11. Šereš, L., & Medaković, Đ. 2014. Embedding OLAP into ERP Systems from the View of User Adoption: The Example of Serbian SMEs. *Management*, 9(1), 03-10.



# APPENDICES

## Appendix I: Prototype Survey



### UNIVERSITY OF NAIROBI

#### AN EVALUATION OF AN IN-MEMORY COMPUTING-BASED SALES ANALYTICS DASHBOARD FOR ORGANIZATIONS RUNNING ERP SYSTEMS

This survey is being undertaken as part of a research at the University of Nairobi, SCI Department to evaluate how using In-Memory Computing can help businesses realize real-time and actionable sales.

##### 1. General Information

- a) Email: \_\_\_\_\_
- b) Business Name: \_\_\_\_\_
- c) Industry:
  - Oil & Gas
  - Real Estate & Hospitality
  - Energy & Construction

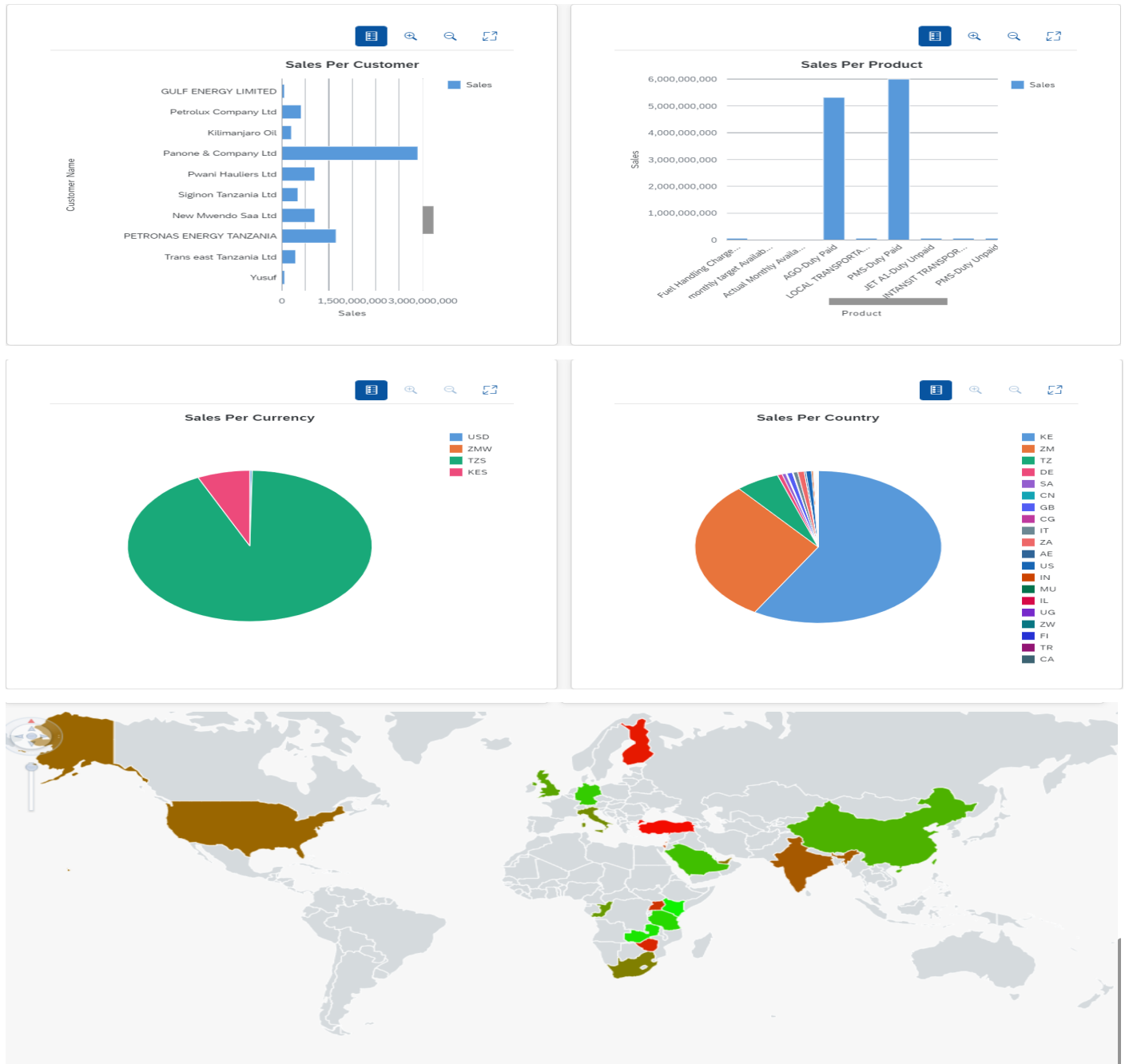
##### 2. Effectiveness of the IMC-based analytics dashboard in the business context

- a) Simplicity  
Is the dashboard simple and easy to use?
  - Strongly agree
  - Agree
  - Not sure
  - Disagree
  - Strongly disagree
- b) Adaptability  
Is the dashboard available on all devices and web browsers?
  - Strongly agree
  - Agree
  - Not sure
  - Disagree
  - Strongly disagree
- c) Delightfulness  
Does the dashboard enriches your work by providing useful sales insights?
  - Strongly agree
  - Agree
  - Not sure
  - Disagree
  - Strongly disagree
- d) Latency  
Does the dashboard provide real-time and actionable sales insights?
  - Strongly agree
  - Agree
  - Not sure
  - Disagree
  - Strongly disagree

**Thank you for your participation!**

## Appendix II: The Analytics Dashboard

Below is the screenshot for the analytics dashboard. It shows in real-time the total sales in four categories; Sales per Customer in a column graph, Sales per Product in a bar graph, Sales Per Currency & Country in a pie chart. Additionally, it displays Sales per country in an Analytic Map where the amount of sales is directly proportional to the intensity of the green color code.



## Appendix III: The CDS Views

### Customer Master Basic CDS View:

```
define view ZI_CUSTOMER
as select from but000 {
key partner as CustomerID,
    name_org1 as CustomerName,
    persnumber as AddressNumber
} where bu_group between 'Z001' and 'Z002';
```

### Product Master Basic CDS View:

```
define view ZI_PRODUCT
as select from makt {
key spras as language,
key matnr as ProductCategory,
    maktx as ProductName
};
```

### Sales Header Basic CDS View:

```
define view ZI_SALESHEADER
as select from vbak {
key vbeln as DocumentID,
    kunnr as CustomerID,
    netwr as GrossAmount,
    waerk as CurrencyCode
};
```

### Sales Items Basic CDS View:

```
define view ZI_SALESITEM
as select from vbap {
key vbeln as DocumentID,
key posnr as DocumentItem,
    matnr as ProductCategory,
    netwr as GrossAmount
};
```

Sales Per Country Composite View:

```
define view ZI_SALESCOUNTRY as select from ZI_AGGRSALESCOUNTRY as _AggrSalesCountry
{
  Country as Country,
  cast( sum(GrossAmount) as abap.dec( 20, 2 ) ) as GrossAmount
} group by Country;
```

Sales Per Customer Composite View:

```
define view ZI_SALESCUSTOMER as select from ZI_SALESHEADER as SalesHeader
association [0..*] to ZI_CUSTOMER as _Customer
  on SalesHeader.CustomerID = _Customer.CustomerID {
  Customer.CustomerID as CustomerID,
  Customer.AddressNumber as AddressNumber,
  Customer.CustomerName as CustomerName,
  sum(GrossAmount) as GrossAmount
} group by _Customer.CustomerID, _Customer.AddressNumber, _Customer.CustomerName;
```

Sales Per Product Composite View:

```
define view ZI_SALESPRODUCT as select from ZI_SALESITEM as SalesItem
association [1..*] to ZI_PRODUCT as _Product
  on SalesItem.ProductCategory = _Product.ProductCategory {
  Product.ProductCategory as ProductCategory,
  Product.ProductName as ProductName,
  sum(GrossAmount) as GrossAmount
} group by _Product.ProductCategory, _Product.ProductName;
```

Sales Per Currency Composite View:

```
define view ZI_SALESCURRENCY as select from ZI_SALESHEADER as SalesHeader
{
  CurrencyCode as Currency,
  sum(GrossAmount) as GrossAmount
} group by CurrencyCode;
```

## Appendix IV: The Business Logic

### UI Controller:

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/suite/ui/commons/ChartContainer",
    "sap/suite/ui/commons/ChartContainerContent",
    "sap/m/MessageToast"
],
function (Controller, ChartContainer, ChartContainerContent, MessageToast) {
    "use strict";
    return Controller.extend("uon.com.analyticsdashboard.controller.Main", {
        onInit: function () {
            var oView = this.getView();
            this.adjustMyChartBox(oView, "idVizFrame1", "Cell1");
            this.adjustMyChartBox(oView, "idVizFrame2", "Cell2");
            this.adjustMyChartBox(oView, "idVizFrame3", "Cell3");
            this.adjustMyChartBox(oView, "idVizFrame4", "Cell4");
            this.getOwnerComponent().getRouter().attachRoutePatternMatched(this.maptheMap, this);
        },
        maptheMap: function () {
            var oDataModel = this.getView().getModel();
            var that = this;
            oDataModel.read("/SalesCountrySet", {
                success: function (data) {
                    data.results.sort(function (a, b) {
                        return b.value - a.value;
                    });
                    var green = 255, red = 1;
                    var aMapData = [];
                    var facto = 255/data.results.length;
                    for (var i = 0; i < data.results.length; i++) {
                        var item = data.results[i];
                        var newGreen = green - facto * (i + 1);
                        var newRed = red + facto * (i + 1);
                        var color = 'rgb('+parseInt(newRed) + ', ' + parseInt(newGreen) + ',0)';
```

```

        item.color = color;
        aMapData.push(item);
    }
    var oModel = new sap.ui.model.json.JSONModel();
    oModel.setData({
        "mapData" : aMapData
    });
    that.getView().setModel(oModel,"viewModel");
    }
});
},
adjustMyChartBox: function (oView, sChartId,sBlockId) {
    var oVizFrame = oView.byId(sChartId);
    var oChartContainerContent = new ChartContainerContent({
        content: [oVizFrame]
    });
    var oChartContainer = new ChartContainer({
        content: [oChartContainerContent]
    });
    oChartContainer.setShowFullScreen(true);
    oChartContainer.setAutoAdjustHeight(true);
    oView.byId(sBlockId).addContent(oChartContainer);
    }
});
});

```

### Configuration File:

```

{
  "_version": "1.32.0",
  "sap.app": {
    "id": "uon.com.analyticsdashboard",
    "type": "application",
    "i18n": "i18n/i18n.properties",
    "applicationVersion": {

```

```
"version": "1.0.0"
},
"title": "{{appTitle}}",
"description": "{{appDescription}}",
"resources": "resources.json",
"ach": "ach",
"dataSources": {
  "mainService": {
    "uri": "/sap/opu/odata/sap/ZUON_MSC_ITM_SALESANALYTICS_SRV_02",
    "type": "OData",
    "settings": {
      "odataVersion": "2.0",
      "localUri": "localService/metadata.xml"
    }
  }
}
},
"sap.ui": {
  "technology": "UI5",
  "icons": {
    "icon": "sap-icon://task",
    "favIcon": "",
    "phone": "",
    "phone@2": "",
    "tablet": "",
    "tablet@2": ""
  },
  "deviceTypes": {
    "desktop": true,
    "tablet": true,
    "phone": true
  }
},
"sap.ui5": {
  "flexEnabled": false,
  "rootView": {
```

```
"viewName": "uon.com.analyticsdashboard.view.App",
"type": "XML",
"async": true,
"id": "App"
},
"dependencies": {
  "minUI5Version": "1.66.0",
  "libs": {
    "sap.ui.core": {},
    "sap.m": {},
    "sap.ui.layout": {}
  }
},
"contentDensities": {
  "compact": true,
  "cozy": true
},
"models": {
  "i18n": {
    "type": "sap.ui.model.resource.ResourceModel",
    "settings": {
      "bundleName": "uon.com.analyticsdashboard.i18n.i18n"
    }
  },
  "": {
    "dataSource": "mainService",
    "preload": true
  }
},
"resources": {
  "css": [
    {
      "uri": "css/style.css"
    }
  ]
},
},
```



```
"routing": {
  "config": {
    "routerClass": "sap.m.routing.Router",
    "viewType": "XML",
    "async": true,
    "viewPath": "uon.com.analyticsdashboard.view",
    "controlAggregation": "pages",
    "controlId": "app",
    "clearControlAggregation": false
  },
  "routes": [
    {
      "name": "RouteApp",
      "pattern": "RouteApp",
      "target": [
        "TargetApp"
      ]
    },
    {
      "name": "default",
      "pattern": "",
      "titleTarget": "",
      "greedy": false,
      "target": ["default"]
    }
  ],
  "targets": {
    "TargetApp": {
      "viewType": "XML",
      "transition": "slide",
      "clearControlAggregation": false,
      "viewId": "App",
      "viewName": "App"
    },
  },
}
```

```
"default": {
  "viewType": "XML",
  "transition": "slide",
  "clearAggregation": "true",
  "viewName": "Main",
  "title": "Dashboard",
  "viewId": "Main",
  "viewLevel": 1,
  "controlId": "app",
  "controlAggregation": "pages"
}
},
"sap.cloud": {
  "public": true,
  "service": "uon-com-analyticsdashboard"
}
}
```

## Appendix V: The OData Project

### OData Logic:

The screenshot displays the SAP Gateway Service Builder interface. The left pane shows the project structure for 'ZUON\_MSC\_ITM\_SALESANALYTICS' under the 'IMC-based Sales Analytics' project. The 'Data Model' section is expanded to show 'Entity Types' with 'SalesAnalytics' containing 'Text' and 'Value' properties. Below are 'Navigation Properties', 'Associations', 'Entity Sets' (SalesCountrySet, SalesCurrencySet, SalesCustomerSet, SalesProductSet), and 'Service Implementation' (Create, Delete, GetEntity, GetEntitySet, Update). The 'Runtime Artifacts' section lists several classes and a service. The right pane shows a table of these artifacts:

Name	Generated Artifact Type
ZCL_ZUON_MSC_ITM_SA_02_DPC	Data Provider Base Class
ZCL_ZUON_MSC_ITM_SA_02_DPC_EXT	Data Provider Extension Class
ZCL_ZUON_MSC_ITM_SA_02_MPC	Model Provider Base Class
ZCL_ZUON_MSC_ITM_SA_02_MPC_EXT	Model Provider Extension Class
ZUON_MSC_ITM_SALESANALYTICS_MDL1	Registered Model
ZUON_MSC_ITM_SALESANALYTICS_SRV_02	Registered Service

### OData URL & Metadata:

[http://jcgdsx.dalbitpetroleum.com:50100/sap/opu/odata/sap/ZUON\\_MSC\\_ITM\\_SALESANALYTICS\\_SRV\\_02/\\$metadata](http://jcgdsx.dalbitpetroleum.com:50100/sap/opu/odata/sap/ZUON_MSC_ITM_SALESANALYTICS_SRV_02/$metadata)

```
<?xml version="1.0" encoding="UTF-8" ?>
<edmx:Edmx xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
  xmlns:sap="http://www.sap.com/Protocols/SAPData" Version="1.0">
  <edmx:DataService m:DataServiceVersion="2.0">
    <Schema xmlns="http://schemas.microsoft.com/ado/2008/09/edm" Namespace="ZUON_MSC_ITM_SALESANALYTICS_SRV_02" xml:lang="en" sap:schema-version="1">
      <EntityType Name="SalesAnalytics" sap:content-version="1">
        <Key>
          <PropertyRef Name="Text"/>
        </Key>
        <Property Name="Text" Type="Edm.String" Nullable="false" sap:unicode="false" sap:creatable="false" sap:updatable="false" sap:sortable="false" sap:filterable="false"/>
        <Property Name="Value" Type="Edm.Double" Nullable="false" sap:unicode="false" sap:creatable="false" sap:updatable="false" sap:sortable="false" sap:filterable="false"/>
      </EntityType>
      <EntityContainer Name="ZUON_MSC_ITM_SALESANALYTICS_SRV_02_Entities" m:IsDefaultEntityContainer="true" sap:supported-formats="atom json xlsx">
        <EntitySet Name="SalesCurrencySet" EntityType="ZUON_MSC_ITM_SALESANALYTICS_SRV_02.SalesAnalytics" sap:creatable="false" sap:updatable="false" sap:deletable="false"
          sap:pageable="false" sap:addressable="false" sap:content-version="1"/>
        <EntitySet Name="SalesCountrySet" EntityType="ZUON_MSC_ITM_SALESANALYTICS_SRV_02.SalesAnalytics" sap:creatable="false" sap:updatable="false" sap:deletable="false"
          sap:pageable="false" sap:addressable="false" sap:content-version="1"/>
        <EntitySet Name="SalesProductSet" EntityType="ZUON_MSC_ITM_SALESANALYTICS_SRV_02.SalesAnalytics" sap:creatable="false" sap:updatable="false" sap:deletable="false"
          sap:pageable="false" sap:addressable="false" sap:content-version="1"/>
        <EntitySet Name="SalesCustomerSet" EntityType="ZUON_MSC_ITM_SALESANALYTICS_SRV_02.SalesAnalytics" sap:creatable="false" sap:updatable="false" sap:deletable="false"
          sap:pageable="false" sap:addressable="false" sap:content-version="1"/>
      </EntityContainer>
      <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="self"
        href="http://jcgdsx.dalbitpetroleum.com:50100/sap/opu/odata/sap/ZUON_MSC_ITM_SALESANALYTICS_SRV_02/$metadata"/>
      <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="latest-version"
        href="http://jcgdsx.dalbitpetroleum.com:50100/sap/opu/odata/sap/ZUON_MSC_ITM_SALESANALYTICS_SRV_02/$metadata"/>
    </Schema>
  </edmx:DataService>
</edmx:Edmx>
```