Lisbon School of Economics & Management
Universidade de Lisboa

U LISBOA | UNIVERSIDADE DE LISBOA

# MASTER
## DATA ANALYTICS FOR BUSINESS

# MASTER'S FINAL WORK
## INTERNSHIP REPORT

# CLOUD DATA WAREHOUSING SOLUTION IN THE BANKING SECTOR

JOÃO DIOGO MIGUEL SILVESTRE

MARCH – 2023

Lisbon School of Economics & Management
Universidade de Lisboa

U LISBOA | UNIVERSIDADE DE LISBOA

# MASTER
## DATA ANALYTICS FOR BUSINESS

# MASTER'S FINAL WORK
## INTERNSHIP REPORT

# CLOUD DATA WAREHOUSING SOLUTION IN THE BANKING SECTOR

JOÃO DIOGO MIGUEL SILVESTRE

**SUPERVISION:**
FACULTY ADVISOR: PROF. CARLOS J. COSTA
INDUSTRY ADVISOR: DR. RUI MIGUEL

MARCH - 2023

ABSTRACT

The Covid-19 pandemic brought about many changes in the business world, and one of the most significant was the accelerated adoption of digital technologies. Commercial banks were no exception, launching several IT projects to support remote work and digital customer experiences. However, this surge in IT projects led to an overload of project management issues. With limited resources and competing priorities, project managers struggled to keep up with the increased workload. To address this problem, banks searched for means to store and analyze important data that would allow them to improve their resource allocation and operational efficiency.

Some of Portugal's largest commercial banks substantially invested in data warehousing and analysis solutions to achieve these goals. The present study is a result of these investments and seeks to create a robust data warehousing solution in which data can be safely stored and develop a business intelligence report for data analysis purposes. The objective of this project is to combine these two components into a decision support system capable of providing up-to-date and accurate information that empowers the bank's managers to make informed business decisions.

This data warehousing solution is deployed on the cloud and involves the creation of an architecture that can handle massive amounts of data while remaining highly scalable. This architecture leverages the Data Lakehouse concept with Kimball's Data Bus principles to achieve its scalability and efficiency. Although some concerns were raised regarding its dependencies on external sources, the solution was implemented successfully and proved to be effective in this specific business context.

***KEYWORDS:*** Data Lakehouse; Data Warehouse; Business Intelligence; Cloud Computing.

***JEL CODES:*** G21; G31; M15; M54; C88; Y1.

## RESUMO

A pandemia da Covid-19 proporcionou muitas mudanças no mercado de trabalho e uma das mais significativas foi a adoção acelerada das tecnologias digitais. Os bancos comerciais que naturalmente não foram exceção, lançaram diversos projetos de TI para apoio ao trabalho remoto e às ferramentas digitais disponibilizadas aos seus clientes. No entanto, a este aumento de projetos TI acresceram problemas na sua gestão. Com recursos limitados e outras prioridades, os gestores de projeto depararam-se com um exponencial aumento de trabalho. Para ultrapassar esses problemas, os bancos procuraram formas de armazenar e analisar dados essencias que permitissem uma melhor alocação de recursos e eficiência operacional.

Assim, para atingir estes objetivos, alguns dos maiores bancos comerciais portugueses investiram de forma substancial em soluções de armazenamento e análise de dados. O presente estudo resulta desses investimentos e foca-se no desenvolvimento de um data warehouse, no qual os dados possam ser armazenados com segurança, bem como a construção de um relatório de análise de dados com recurso a um software de business intelligence. O objetivo deste projeto resulta da conjugação de ambos e tem o intuito de fornecer aos gerentes do banco informações atualizadas para a tomada de decisões.

Esta solução de data warehousing é implementada na cloud e envolve uma arquitetura de altamente escalável e com elevada capacidade de armazenar grandes quantidades de dados. Para alcançar escalabilidade e eficiência, esta arquitetura combina o conceito de Data Lakehouse com os princípios do Data Bus de Kimball. Embora algumas preocupações tenham surgido quanto à sua dependência para com fontes de dados externas, esta solução provou ser eficaz quando aplicada ao contexto de negócio em questão.



***PALAVRAS-CHAVE:*** Data Lakehouse; Data Warehouse; Business Intelligence; Cloud Computing.

***JEL CODES:*** G21; G31; M15; M54; C88; Y1.

TABLE OF CONTENTS

LIST OF FIGURES

## LIST OF TABLES

## ACRONYMS AND ABBREVIATIONS

| | |
|---|---|
| ACID | *Atomicity, Consistency, Isolation, and Durability* |
| BI | *Business Intelligence* |
| CRISP-DM | *Cross Industry Standard Procedure Data Mining* |
| CIF | *Corporate Information Factory* |
| CSV | *Comma-Separated Values* |
| DBA | *Data Bus Architecture* |
| DL | *Data Lake* |
| DLH | *Data Lakehouse* |
| DW | *Data Warehouse* |
| ETL | *Extract, Transform, and Load* |
| GDPR | *General Data Protection Regulation* |
| ITS | *IT Service Requests* |
| JSON | *JavaScript Object Notation* |
| KPI | *Key Performance Indicator* |
| RDBMS | *Relational Database Management System* |
| PaaS | *Platform as a Service* |
| SaaS | *Software as a Service* |
| SQL | *Structured Query Language* |

## ACKNOWLEDGMENTS

First and foremost, I would like to express my gratitude to the people that made this work possible. Professor Carlos J. Costa played a crucial role in guiding me through the development of the master's final work and Dr. Rui Miguel supervised my integration at Unipartner's Data & AI department and ensured it was successful.

Besides Professor Carlos, various other ISEG professors also deserve recognition for providing me with the education and expertise that has helped me become a competent professional. I am also grateful for my colleagues at Unipartner who aided me in adjusting to my first professional experience in the field of data analytics. Namely Carlos Santos, José Estevens, and Pedro Rocha, whom I am immensely grateful for. I also want to emphasize the role that Raul Araújo, and Inês Ribeiro played in the initial stages of the report, effectively bridging the gap between me, the company, and the faculty.

Lastly, and most importantly, I would like to thank my family, especially my parents and grandparents. They are the biggest contributors to my education, as well as my greatest supporters. I owe them my academic success and I could not have done my bachelor's and master's degree without them, let alone the master's final work.

# 1. INTRODUCTION

## *1.1. Contextualization*

Digital technologies play an increasingly crucial role in most business sectors and recent events have further intensified its relevance. A study conducted by the European Investment Bank indicated that even though digitalization already had an upward trend, it was the Covid-19 pandemic that accelerated the need for firms to become more digital. In Portugal, it was concluded that 42% of firms invested in digitalization, with a total of 67% having implemented at least one advanced digital solution in 2021. It is also stated that these investments are expected to not be limited to the short-term and will most likely have inevitable long-term growth (Maurin et. al., 2021).

Regarding the commercial banking sector, field experts from major Portuguese banks also stated that the pandemic did indeed unlock a digital potential that organizations in the field were often unable to fully adapt to (Jornal de Negócios, 2021). This statement was further corroborated by the Portuguese government when the secretary of treasury stated that similarly to other business sectors, the banking sector faced challenges with digital innovation (Mendes, 2022).

Given this contextualization, this master's final work focuses on implementing a digitalization project at one of the largest commercial banks in Portugal. This project is centered around the creation of a data warehousing solution in the cloud, based on the bank's specific needs. The solution was designed by the Data & Artificial Intelligence team at Unipartner IT Services. The author of this master's final work is part of this team and contributed directly to the solution's development process.

Unipartner IT Services is a privately held consulting company that works with organizations to solve their IT and business challenges. Unipartner was founded in 2015 and has grown exponentially ever since. Today, the company is ranked as a Microsoft gold partner and has won several Microsoft awards in the past few years, such as the Microsoft Partner of the Year award in 2020 (Unipartner, 2022).

The financial institution that acquired Unipartner's services is a commercial bank that operates in Portugal. In adherence to the General Data Protection Regulation (GDPR) and its provisions for privacy protection, the bank's name will never be revealed throughout

this report. Therefore, the institution's anonymity will be ensured in accordance with its conditions for consent (Regulation 2016/679 art 7). Instead, this financial institution will often be referred to as "the bank", or simply "the client".

Due to the bank's data privacy and security policies, which the author agreed to, the names of entities, attributes, azure keys, and authentication methods are given fictional names. Moreover, many Python and Structured Query Language (SQL) commands are not disclosed, and the graphical visualizations displayed in the business intelligence report are partially censored.

### *1.2.Objectives*

This work aims to create a data warehousing solution that will serve as the basis for an automated decision support system. The client intends to use it to better manage the development capacity of several teams of employees and the IT-related services they provide. To achieve this, the project was broken down into three distinct objectives:

1. Define the data warehouse (DW) architecture as the system's blueprint.
2. Create a backend for the system in the form of an extract, transform, and load (ETL) process that transfers data in a structured manner into the DW.
3. Create a frontend for the system in the form of a business intelligence (BI) report that provides an analysis of the data in the DW.

Ultimately, the ETL process is orchestrated continuously, and data is to be extracted, transformed, and loaded daily. Thus, ensuring that the reports are up to date on every matter. These will then be consumed by the bank's managers and empower them to make more informed business decisions on the bank's behalf.

### *1.3.Structure of the Master's Final Work*

This academic work starts with a brief description of its methodological approach. Afterwards, the necessary literature is reviewed based on this work's context. The business and data understanding mark the beginning of the empirical work, followed by the data preparation and a quick description of the performed data analysis. Lastly, the project's conclusion is presented with an insight into future research and issues that may arise from implementing this system.

## 2. METHODOLOGICAL APPROACH

The methodology used in this project is an adaptation of the Cross Industry Standard Procedure – Data Mining (CRISP-DM), which divides the lifecycle of a data mining project into six stages: business understanding, data understanding, data preparation, modelling, evaluation, and deployment (Shearer, 2000, Costa & Aparicio, 2020).

In the first phase, the focus was on understanding the business objectives that the client intended to achieve. Additionally, a review of the literature on data warehousing was performed and common approaches used in these types of projects were identified.

Secondly, the bank's data sources were thoroughly explored, and the relevant measures and key performance indicators (KPIs) needed to develop the project were described. At this stage, the data warehouse architecture was also defined, and the first logical data model was designed based on the previously defined KPIs.

Thirdly, the data preparation phase marked the beginning of the ETL process creation, which is a key component of the DW architecture. In the ETL, several fields of raw data were selected, cleaned, and integrated into a structured form that fits the relational model.

The modeling phase is typically related to the development of a machine learning solution (Shearer, 2000). The project at hand does not include such techniques, however, it does involve a data analysis solution, which was the focus of this project stage.

Subsequently, the BI report together with the DW architecture, were presented to the client on a weekly basis to obtain constant feedback. Based on the client's feedback, the solution suffered continuous changes until it reached its final state. Lastly, the solution was deployed and monitored to ensure it was working as intended.

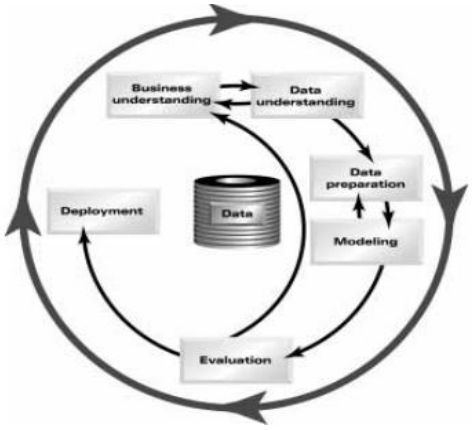


Figure 1 – Cross Industry Standard Procedure – Data Mining (CRISP-DM)

# 3. LITERATURE REVIEW

## *3.1. Data Warehouse*

A DW is typically used as an organization's central data repository which is populated through an ETL process that integrates data from a variety of sources (Harby & Zulkernine, 2022). To interact with the DW, the end user employs data analysis tools to analyze the data within it and make business decisions (Nambiar & Mundra, 2022).

There are multiple ways to design a DW for a business. Over the years, several DW architectures were created and there are two authors whose architectures stood out the most: Ralph Kimball and William Inmon. While Kimball campaigns for a data bus architecture (DBA) (Kimball & Ross, 2013), Inmon defends a corporate information factory (CIF) (Inmon, 2005).

Kimball's DBA is divided into three distinct parts: the ETL process, a dimensional data model and a BI application. This architecture is centered around a star schema which is composed of one fact table and several dimensions. The fact tables are typically large and contain the business measures, also known as facts, that a specific business department may be interested in analyzing. By contrast, the dimensions tend to be smaller and contain the descriptive features the BI application uses to filter and aggregate the facts. Furthermore, each dimension must correspond to a single ETL process and can be present in more than one star schema, thus providing a consistent view of the company's business reality (Kimball & Ross, 2013). Because of these characteristics, DBA can be quickly implemented. As such, it often applies to corporations with a limited and urgent need for a data warehouse (Ariyachandra & Watson, 2007).

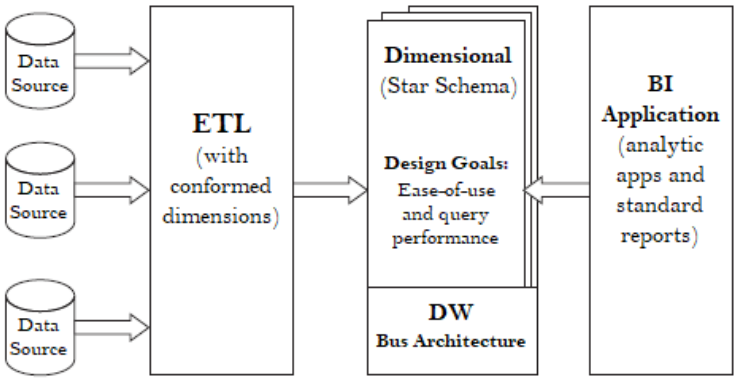


Figure 2 – Core Elements of Kimball's DW Architecture.
Adapted from "The Data Warehouse Toolkit" (Kimball & Ross, 2013)

Like Kimball, Inmon's architecture relies on an ETL process to extract, transform, and load data into a data center. However, Inmon defends a flexible relational schema based on the third form of data normalization and applies to the entire enterprise. Furthermore, this flexible relational schema does not need to follow a specific structure as the star schema does. CIF trades the advantage of having a relational schema that applies to the whole enterprise for the disadvantage of needing an added layer that takes data from the enterprise data warehouse to department-based data marts so that these can then be connected to the BI applications (Inmon, 2005). Because of these characteristics, CIF is a complex design that takes time to implement and requires more strategic planning. As such, this architecture is typically implemented at organizations that see the DW as strategic (Ariyachandra & Watson, 2010).

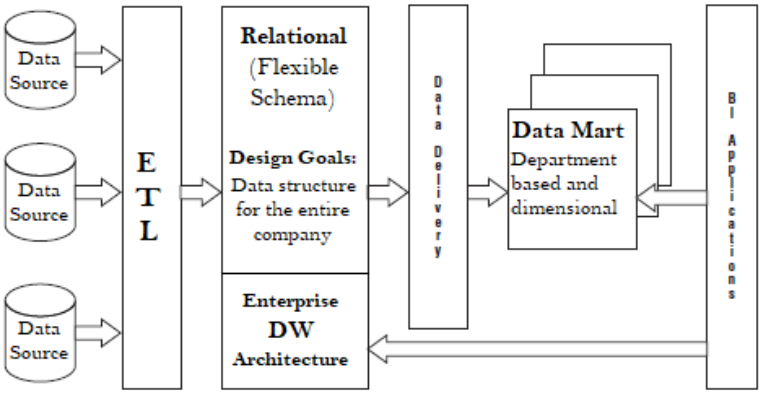


Figure 3 – Core Elements of Inmon's DW Architecture
Adapted from "The Data Warehouse Toolkit" (Kimball & Ross, 2013)

By examining the literature seen so far for both Kimball's DBA and Inmon's CIF, it is possible to identify the main differences between these two architectural approaches. The distinction between the two is summarized in the table below.

TABLE I – DIFFERENCES BETWEEN KIMBALL'S DBA AND INMON'S CIF

| **Parameters** | **Kimball's DBA** | **Inmon's CIF** |
|---|---|---|
| Approach | Follows a bottom-up approach | Follows a top-down approach |
| Data Integration | Focuses on individual business areas | Focuses on enterprise-wide data integration |
| Data Model | Star schema that applies to a specific department of the enterprise | Flexible relational schema that applies to the entire enterprise |
| Normalization | Does not need to be fully normalized | It must be in the third form of data normalization |
| Building Time | Takes less time to set up | Takes more time to set up |
| Initial Cost | Has a lower initial cost | Has a higher initial cost |
| Maintenance | Expanding the DW to more departments makes its maintenance progressively more difficult | Once the design for the whole enterprise is complete, maintaining the DW is easier |

### *3.2. Data Lake*

DW systems tend to have compatibility issues with open-source and cloud-based data engineering tools. The data lake (DL) was created to address these shortcomings and support big data processing capabilities (Armbrust et. al., 2021). Although both the DW and DL serve as data repositories, they have unique features that distinguish them from one-another. DWs are designed to store processed data that has been cleansed and organized into a single schema through its ETL process (schema-on-writing). In contrast, data lakes can ingest a large amount of unstructured data, store it efficiently and then organize it as needed (schema-on-reading). This makes them particularly useful when dealing with big data since DLs quickly ingest unstructured data and store it at a lower cost than the DW. Nonetheless, due to the complex and unstructured nature of data stored in DLs, data analysis can be challenging, often necessitating the expertise of specialized personnel to operate and manage the DL (Nambiar & Mundra, 2022).

The architecture of DLs is considerably more flexible than that of DWs. DLs are usually divided into several ponds or layers (Harby & Zulkernine, 2022), with the most common ones being raw, standardized, and cleansed layer (Nambiar & Mundra, 2022). The raw data layer acts as a landing zone and is designed to ingest raw data efficiently without any transformations. The standardized layer is designed to improve data transfer performance from the raw to the cleansed layer. Lastly, the cleansed or curated layer is where several data objects are transformed into structured data sets that the end user can interact with (Nambiar & Mundra, 2022).

Based on the literature described so far regarding DWs and DLs, their differences can be summarized into the ten different parameters that are displayed in the table below.

TABLE II – DIFFERENCES BETWEEN DWS AND DLS

| **Parameters** | **Data Warehouse** | **Data Lake** |
|---|---|---|
| Purpose | Has a business-specific purpose | Has a general purpose |
| Data Processing | Stores only highly processed data | Stores mainly unprocessed data |
| Data Type | Only structured data | Unstructured, semi-structured, structured data |
| Storage | Higher cost storage with a faster response time | Lower cost storage with a slower response time |
| Schema | Schema-on-writing (predefined schemas) | Schema-on-reading (no predefined schemas) |
| Ingestion | Has slower ingestion of new data | Has faster ingestion of new data |
| Primary Task | Optimized for data analysis | Optimized for data storage |
| Configuration | Has a limited configuration | Has a flexible configuration |
| Access Control | Allows more control over data access | Offers less control over data access |
| End Users | Typically used by business analysts | Typically used by data scientists |

### *3.3. Data Lakehouse*

The Data Lakehouse (DLH) is a new data management system that blends the features of traditional relational database management systems (RDBMS) used in DWs with the rapid ingestion, scalability, and optimized storage of DLs (Harby & Zulkernine, 2022). To achieve this, the DLH relies on a DL platform with a system that can store data in a low-cost object store (e.g., Azure Blob or Amazon S3) using file formats such as Apache Parquet (Armbrust et. al., 2021).

The practical implementation of the DLH is typically made through Databricks using the Delta Lake system, which acts as a transactional metadata layer on top of the object store that tracks which files are part of a given entity (Armbrust et. al., 2020). This unlocks the possibility of implementing ACID (atomicity, consistency, isolation, and durability) transactions within the metadata layer, while keeping most of the data in the low-cost object store of the DL (Armbrust et. al., 2021). The DLH represents the most modern storage system of the three reviewed so far and combines the benefits of both DWs and DLs that were listed in Table II.

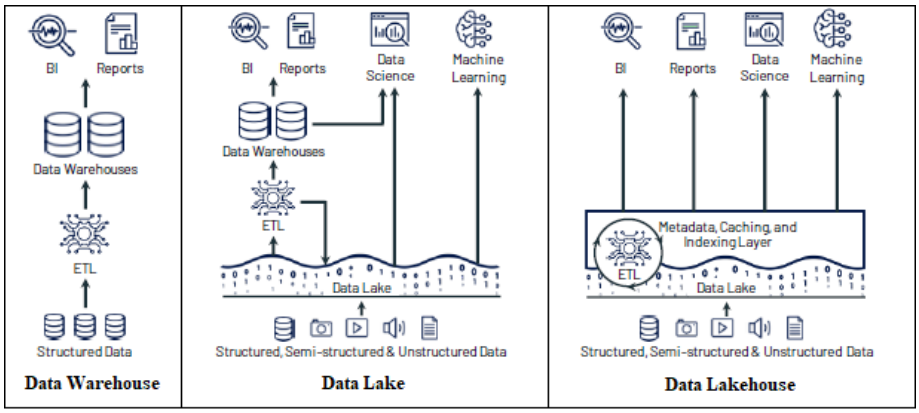


Figure 4: Evolution of Data Platform Architectures (Armbrust et. Al., 2021)

Since the DLH is a recent innovation, there is still a lack of research that exemplifies how this system can be effectively implemented. To the author's knowledge, one of the few articles that outlines a practical application of the DLH was developed for mega-biobanks where the interaction with the system was designed for data science, occasional reporting, and other biomedical research through virtual machines from which researchers access the data (Begoli et. al., 2021). Another very recent article that explores the DLH concept focuses on using it as a storage system for complex data used in deep learning, such as images or videos, represented as tensors (Hambardzumyan et. al., 2023).

These approaches differ quite significantly from the decision-support system this work intends to accomplish. However, there are some valuable insights in each research that should be taken into consideration. The first article's explanation of the data lifecycle seems relevant, particularly the strategy taken to streamline data movement between different directories by partially modifying the data file path (Begoli et. al., 2021). The second article highlights a version control methodology that could also be taken into consideration. It keeps track of a complete data lineage by storing different versions of the same dataset within the same storage but separated by sub-directories (Hambardzumyan et. al., 2023). Although the overall architecture described in both articles seem unfit for the solution this work intends to accomplish, these two specific design characteristics could be applied to a DLH designed for a bank.

As an alternative to any specific data processing technique, Databricks defines the medallion DLH architecture as the fundamental framework for organizing data within the DLH. The objective is to enhance the structure and quality of data as it traverses through each layer of the DLH, progressing from Bronze to Silver and eventually to Gold layer tables. (Azure Databricks, 2022). This design seems more aligned with the objectives this work intends to achieve.


Figure 5: Medallion DLH Architecture (Databricks Glossary, 2022)

The medallion architecture does not replace other dimensional modelling techniques, such as Kimball's DBA or Inmon's CIF. The user's requirements determine the diversity of forms and levels of normalization that datasets within each layer can adopt (Azure Databricks, 2022). Therefore, the DLH medallion architecture could prove to be useful, along with the techniques highlighted in the biobanks DLH (Begoli et al., 2021) and the deep learning DLH (Hambardzumyan et al., 2023).

# 4. EMPIRICAL WORK

## *4.1. Overview*

This work focuses on the development of a cloud data warehousing solution, the ETL process embedded in it, and a BI report that serves as a data analysis tool. The client stated that the solution should be designed around KPIs related to the number of hours spent on developing IT service requests (ITS) for the bank. These ITS include everything from small tasks to large projects developed for the bank by both internal and external teams of employees. As a result, the main metric around which the DW should be centered is time.

It was identified that the business data required to develop this project resided in three different sources: DevOps, Dataverse and File share. An Azure Data Lake Storage was created to serve as a central repository to facilitate the transformation of data from all these sources. Consequently, the data from these sources is ingested into it every day through a batch process in Azure Synapse Pipelines.

Data is regularly extracted from the DL repository and transformed into several entities through Azure Databricks to fit the schema that is analyzed in the next chapter. After all the entities are fully processed, Azure Synapse Studio is used to query them into the Azure Synapse Serverless SQL Pool. Finally, Power BI is utilized to create the report based on the table views available in the SQL Pool.

Given what was seen in the literature review, the appropriate architecture will be chosen based on the client's business necessities. The ETL process of the solution will be explained in detail and some Power BI visualizations will be displayed at the end.

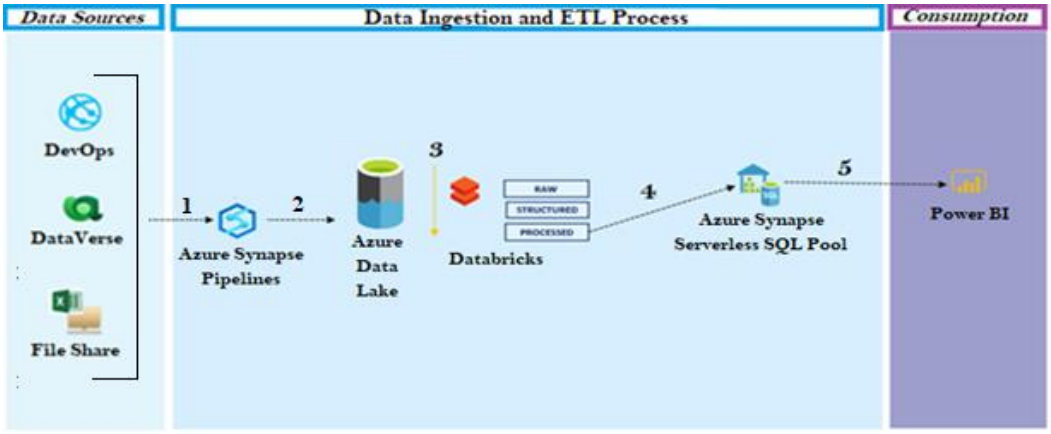


Figure 6 – Project Overview

### *4.2. Business and Data Understanding*

Following the Covid-19 pandemic, the number of IT-related services needed for the bank to improve or maintain its business activities, grew significantly. Eventually, this led to a breaking point in which project managers could no longer keep up with the vast amount of projects they had to manage simultaneously. This prompted the bank to seek out a solution that would aid its project managers in decision-making.

The client's main tool to plan its ITS is Azure Devops, which allows for a smooth collaboration between managers and developers in a single platform. This ensures the completion of projects and tasks at a faster pace. Azure DevOps itself contains several services, among which the Azure Boards. This software as a service (SaaS) allows organizations to plan their work with kanban boards that support the creation of work items corresponding to distinct small tasks or even large projects. Once a work item is created, employees can be assigned to complete it (Azure Boards, 2022).

The client at hand set up its Azure Boards with three distinct kanbans to track the development process of its ITS. The first one is the Planning Kanban, which contains the ITS that are still being planned. In this kanban, the request for an IT service is first made along with the allocation of a budget and development hours for its completion. The second one is the Execution Kanban, which contains the ITS that are already in progress. In this kanban, the development of the project or task is made and registered. Thirdly, there is the General Kanban, which aggregates all ITS being planned or already under development. As such, the General Kanban is the one that registers the full cycle of an ITS from its inception until its conclusion.

There are two other major data sources aside from DevOps: Dataverse and File share. The former is a storage system that contains data in a set of tables that can be customized according to the needs of each organization (Microsoft Dataverse, 2022). The latter is a SaaS that allows users to store and share files in the cloud by employing file sharing protocols such as server message block or network file system (Azure Files, 2022). These two sources of data are of utmost importance. The file share is used to store comma-separated value (CSV) files that list all internal and external employees and their monthly contract hours. In addition, these employees rely on Dataverse to report the work hours dedicated to completing an ITS.

It is also important to note that the client organization has a specific way of distinguishing projects from day-to-day operations. IT-related activities at the bank are grouped into three distinct activity types: planned activity, current activity, and management activity. Planned activities relate to the ITS developed through DevOps, which refer to projects and tasks that are completed as new business needs emerge. In contrast, current activities correspond to daily operations that need to be performed regardless. Lastly, there are management activities, which are only performed by managers at the bank to coordinate teams and departments.

Given the previously described sources, the bank executives and project managers highlighted some measures they would like to visualize in the frontend of this system. These measures include the available capacity, allocated capacity, used capacity and remaining capacity. All of these are essential to create the visualizations that will be seen in the BI report and some of the most important KPIs for this business.

The available capacity constitutes the sum of all the contracted hours available across all the employees in the company. This is the total capacity in hours the organization has available to complete planned, current and management activities.

$$(1)\ AVC(t) = \sum_{n} C_n(t)$$

Where variables in equation (1) are $AVC$ available capacity, $C$ the number of contracted hours, $n$ the number of employees, and $t$ the analyzed time period in months.

The allocated capacity constitutes the sum of all the allocated hours attributed to completing a specific ITS. This measure only applies to the planned activity and to future time periods, simply because there is no point in allocating capacity to the past.

$$(2)\quad AC(t) = \begin{cases} \sum_{n} A_n(t), \ \forall\, t > t_0 \\ 0, \ otherwise \end{cases}$$

Where variables in equation (2) are $AC$ allocated capacity, $A$ number of allocated hours, $n$ the number of ITS, $t$ the analyzed time period in months and $t_0$ the current date's month.

On the contrary, used capacity constitutes the sum of all the operational hours reported by the employees to complete a specific ITS. This measure only applies to the planned activity and to past time periods since it is not feasible to measure the capacity used for periods that have not yet come to pass.

$$(3)\ UC(t) = \begin{cases} \sum_{n} O_n(t),\ \forall\, t < t_0 \\ 0,\ otherwise \end{cases}$$

Where variables in equation (3) are $UC$ used capacity, $O$ number of operational hours, $n$ the number of ITS, $t$ the analyzed time period in months, and $t_0$ the current date's month.

The fourth and last measure is the remaining capacity, which represents the number of estimated hours left to complete a specific ITS. This measure allows project managers to be aware of how many additional hours should be allocated to a specific ITS, given the manager's original estimation and the progress that was made so far.

$$(4)\ RC = ED - UC - AC$$

Where variables in equation (4) are $RC$ remaining capacity, $ED$ estimated duration in hours, $UC$ total used capacity, and $AC$ total allocated capacity.

Decision makers at the bank stated that these measures should all be displayed for planned activities, whereas for current and management activities, the available capacity would suffice for now. Additionally, these measures should also be used to create some business KPIs that must be displayed on the final BI report, such as:

1. Percentages of available, allocated, and used capacity per IT department;
2. Percentages of available, allocated, and used capacity per employee origin;
3. Percentages of allocated and used capacity for each development area;
4. Difference between estimated and actual hours used for each ITS;
5. Monthly evolution of available, allocated, and used capacity.

Having in mind what was seen in the literature review, and based on the data description so far, a choice on DW architecture must be made. The client stated that this project has a high degree of urgency. Furthermore, the DW should be built specifically for the IT sector of the bank, around the four defined measures. Opting for DBA seems obvious- However, establishing the DW in a cloud environment is necessary to maintain a stable connection with the three independent sources. Therefore, the situation at hand requires a solution that combines the concept of a DLH with DBA's dimensional data model. According to the author's knowledge at the time of writing this report, this is the only academic work that has yet attempted to define the union of these two concepts. Therefore, the author decided to name it as Lakehouse DBA. This design stays true to the concept of a DLH, but is heavily influenced by Kimball's architecture.

In essence, the Lakehouse DBA can be broken down into four different segments. The first one is the ingestion of raw data from the sources into the DL. Followed by the ETL process, which uses Databricks to extract the ingested data, transform it into distinct entities, and load the entities back into a separate directory of the DL. The data entities created within the ETL process are developed in accordance with Kimball's dimensional data model. The idea is to have several conformed dimensions that can be used in several models as described in Kimball's DBA. The main difference being the fact that these dimensions are only stored in a DL with a low-cost storage. Consequently, the entities needed for a specific model are queried into an SQL Pool so that they can be analysed through a specific BI software.

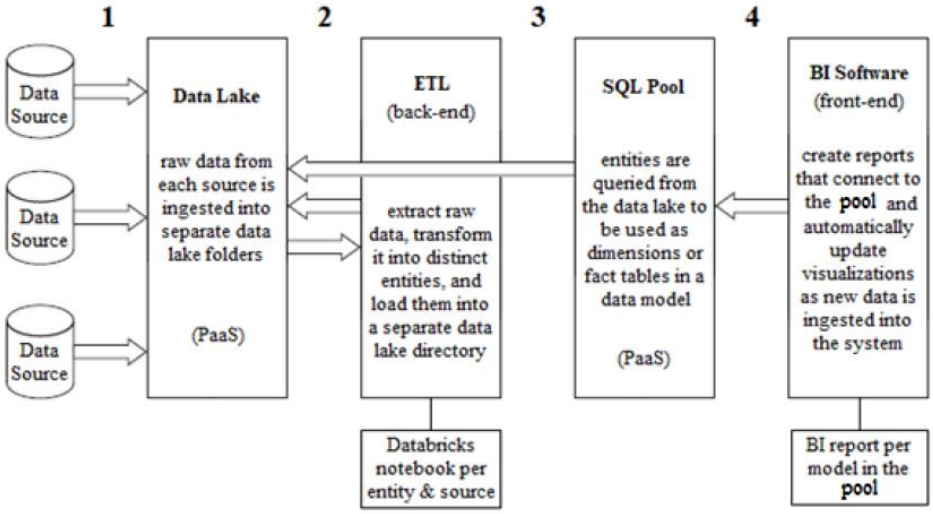


Figure 7 – Lakehouse DBA

The goal is to take advantage of DBA's benefits, namely its quick implementation with a simple but efficient dimensional data model, and combine it with DLH benefits, such as its cloud-environment efficiency with reduced storage costs. To achieve this, the ETL process in this design applies some of the concepts seen in the literature review. Specifically, it leverages streamlined data movement between directories, employs version control of datasets through sub-directories, and implements a three-layered data quality enhancement process as defined in the medallion architecture.

Based on the data description provided by the client, the relational model displayed in the following page was created. The model is a fact-constellation schema which is simillar to Kimball's star schema, but has more than one fact table. This schema is the centerpiece of the decision support system that is explored throughout this work and aims

to analyze the capacity KPIs that were previously defined. There is a second model in this DW that focuses on the cost analysis of ITS based on the number of hours dedicated to developing them. Although this second model shares some of the same dimensions as the first one, it is still under development and will not be discussed in this work.

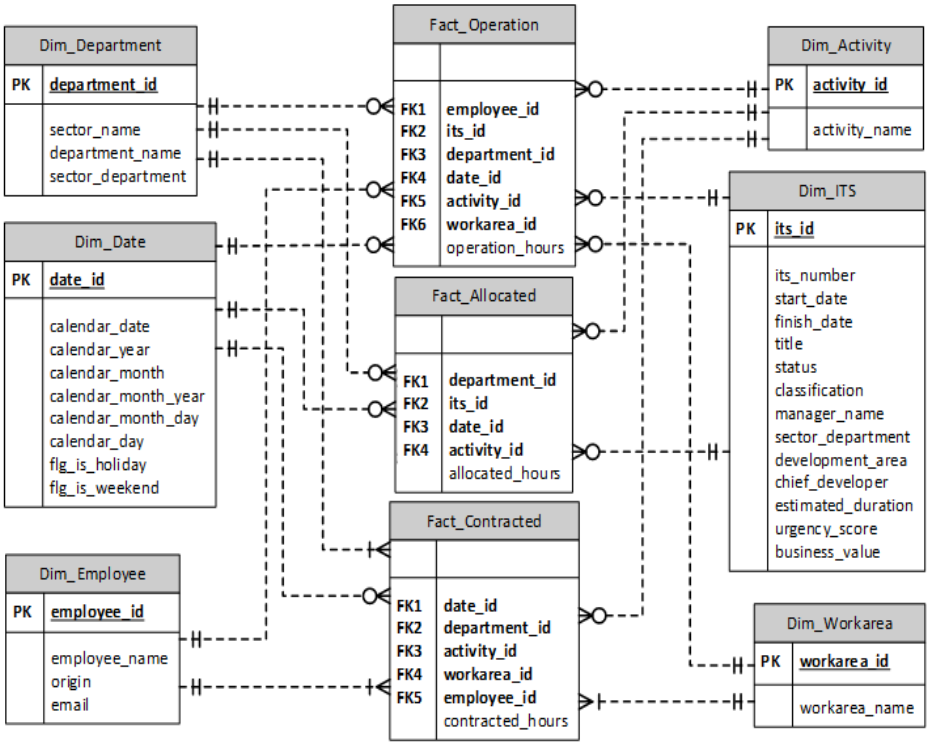


Figure 8 – Logical Data Model

This data model has a total of three fact tables and six dimensions. The fact tables contain the facts or metrics that were previously discussed, and each dimension enables the grouping of the facts based on specific attributes. This grouping allows for the organization and categorization of data, thus providing a structured framework for analysis and reporting of each of the KPIs which will be elaborated upon in a later chapter.

The tables constituting this logical data model will be thoroughly discussed in the data preparation phase, focusing on the creation of each table in a delta format, and elucidating the meaning associated with every attribute within them. However, the ones for the date and activity dimensions will not be shown, mainly because these dimensions are static. Therefore, they are not subject to changes over-time, unlike all the other dimensions. The "dim_date" only contains dates up to the year 2100 and the "dim_activity" only contains the three activity types mentioned in the data understanding chapter: the planned, current, and management activities.

### *4.3. Data Preparation*

This section of the report focuses on the data preparation that occurs in the ETL process, which is the backend of the architecture. This ETL interacts with two cloud services, one platform as a service (PaaS) for the DL and another for the SQL Pool.

The DL PaaS chosen for this project was Azure Data Lake Storage Gen2, which is ideal for big data analytics. It is built on top of Azure Blob Storage, thus making it very cost-efficient and easily scalable. Furthermore, it is Hadoop compatible since it allows data access through the Hadoop Distributed File System and enables a direct connection to Hadoop environments such as Azure Databricks (Azure Data Lake, 2022).

The second PaaS is Azure Synapse Serverless SQL Pool, which provides a query service that can create table views of the entities stored within the DL. It supports large-scale data functions and because it is a serverless service, there is no infrastructure to setup. Moreover, this is an entirely pay-per-use model since costs are only incurred for the data processed by the queries (Azure Synapse Serverless SQL Pool, 2022).

First, it is necessary to populate the Azure DL with data from the sources. The connection between both is made through the Azure Synapse workspace, where pipelines can be created to perform data movement activities that import datasets into a DL (Azure Synapse Pipelines, 2022). For this project in specific, the only pipeline activity used for each source was the copy activity, which essentially just moves data from the sources to a DL directory named “RAW”. Once data is ingested into “RAW”, Azure Databricks combines the power of Apache Spark with Delta Lake and effectively creates the ETL using programming languages such as Python and SQL.

The ETL in Databricks is broken down into two different stages. In the first one, datasets from each source are extracted from the “RAW” directory, and the necessary fields are selected, structured, and moved to the “Structured” directory. In the second stage, the datasets in “Structured” are then accessed and transformed into each table of the model in a delta format, ensuring that new batches of data can be continuously merged into the tables. Each delta table is then loaded into a third and last directory in the DL named “Processed”. Data movement between each directory occurs daily by linking the source and destination file paths. Each directory contains several sub-directories that correspond to the day, month and year in which the dataset was loaded.

Lastly, two transact-SQL scripts are run in the Azure Synapse Workspace to query the facts and dimensions stored in the Processed directory of the DL, so that they can be viewed in the SQL Pool and analyzed by the user in the BI report.

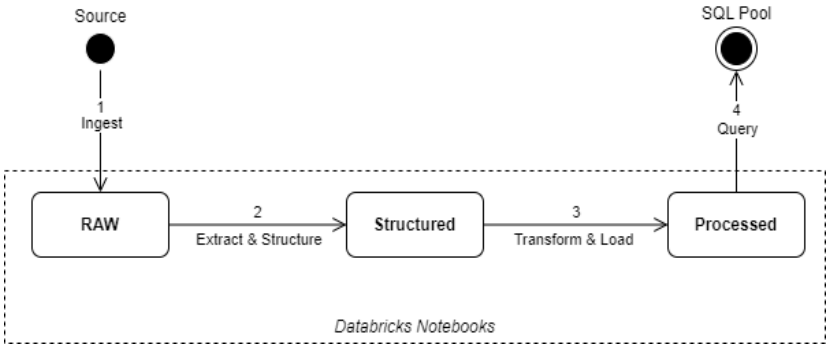


Figure 9 – ETL State Diagram

TABLE III – DATABRICKS NOTEBOOKS AND CORRESPONDING ENTITIES

| Notebook | Entity |
|---|---|
| Common_Functions | - |
| Extract_Devops | - |
| Extract_Dataverse | - |
| Extract_Fileshare | - |
| Transform_Load_DimDate | dim_date |
| Transform_Load_DimActivity | dim_activity |
| Transform_Load_DimWorkarea | dim_workarea |
| Transform_Load_DimDepartment | dim_department |
| Transform_Load_DimEmployee | dim_employee |
| Transform_Load_DimITS | dim_its |
| Transform_Load_FactContracted | fact_contracted |
| Transform_Load_FactOperation | fact_operation |
| Transform_Load_FactAllocated | fact_allocated |
| ETL_Master | - |

All the databricks notebooks used to construct the ETL process can be seen in Table III. The “Common_Functions” notebook is outlined in the appendix and is run at the beginning of every other notebook with the command “%run ./Common_Functions”, so that all the necessary functions can be called. The command “dbutils.notebook.exit()” is also run at the end of every notebook so that it can be exited successfully.

The “Extract” notebooks execute step 2 of the ETL State Diagram and effectively extract the necessary data and ensure that they follow a specific structure. The “Transform_Load” notebooks execute step 3 of the ETL State Diagram and, therefore, transform and load the data in a structured manner into each one of their corresponding entities. Lastly, the “ETL_Master” is used solely to execute all the other notebooks daily, thus ensuring that all the data is up to date.

### *4.3.1. Extract Notebooks*

```
set_spark_adls_conf()
load_date()

# Source & Destination Variables
source_path   = f"""RAW/DEVOPS/Services/{LoadDateFolder}"""
adls_source_path = get_adls_path(source_path)
destination_path = f"{source_path}".replace("RAW", "Structured")
adls_destination_path = get_adls_path(destination_path)

col_list = ["start_date", "finish_date", "classification", "estimated_duration", "sector_department", "title",
            "kanban_name", "its_number", "manager_name", "urgency score", "development_area", "status",
            "allocation_date", "allocated_hours", "chief_developer", "business_value"]

col_renames = {"Custom.fb39cf79-dea3-4112-8bd5-9db08a195594": "estimated_duration",
               "Custom.2e45ec6e-2b41-45c7-a641-e153224b47e5": "urgency_score",
               "Custom.6ded2b9f-075b-4d58-abb7-3ae010550305": "its_number"}

# Read JSON and Structure Columns
if check_path_exists(adls_source_path):
    try:
        df_devops = spark.read.option("multipleline","true").json(adls_source_path)
        for i in range(len(df_devops.columns)):
            if df_devops.columns[i].startswith("Microsoft.VSTS.") or df_devops.columns[i].startswith("System."):
                df_devops = df_devops.withColumnRenamed(df_devops.columns[i], df_devops.columns[i].split(".")[-1])
            elif df_devops.columns[i].startswith("Custom.") and "-" not in df_devops.columns[i]:
                df_devops = df_devops.withColumnRenamed(df_devops.columns[i], df_devops.columns[i].split(".")[-1])

        for key, value in col_renames.items():
            df_devops = df_devops.withColumnRenamed(key, value)
        df_devops.select(*col_list)
    except Exception as e:
        raise e
else:
    print("Path does not exist: ",adls_source_path)
    dbutils.notebook.exit("Extract_Devops")

df_devops.repartition(1).write.option("mergeSchema", "true").format("parquet").mode("overwrite").save(adls_destination_path)
dbutils.notebook.exit("Extract_Devops")
```

Figure 10 – Extract_Devops Notebook

The Extract_Devops notebook starts by calling the functions in figures 32, 33 and 34 to establish the connection with Azure DL, create a load date for the subdirectory path and define the source and destination paths. After invoking the function in figure 35 to ensure the path exists, the JSON file containing all the DevOps ITS is read into a spark data frame. However, the data ingestion pipeline imports the source fields into the DL RAW directory with incorrect names. To fix this issue, each column must be renamed. Some column renames are very straightforward, as can be seen in the first for loop. However, others require a manual rename, as can be seen in the second for loop. After these commands are run, the data frame is stored in its destination path in the Structured directory. It is also worth mentioning that the if/else and try/except statements used in this extract notebook, and the subsequent ones, play a crucial role in ensuring the azure DL source path is correct and that any issue in reading the files is raised as an exception.

```
set_spark_adls_conf()
load_date()
# Source & Destination Variables
hours_source_path    = f"""RAW/DATAVERSE/ReportedHours/{LoadDateFolder}"""
hours_adls_source_path = get_adls_path(hours_source_path)
employees_source_path = f"""RAW/DATAVERSE/EmployeeList/{LoadDateFolder}"""
employees_adls_source_path = get_adls_path(employees_source_path)
destination_path = f"{hours_source_path}".replace("RAW", "Structured").replace("ReportedHours", "Timesheets")
adls_destination_path = get_adls_path(destination_path)

col_list = ["calendar_date","its_number","email","operation_hours", "employee_name"]

# Read JSON of Reported Hours
if check_path_exists(hours_adls_source_path):
    try:
        df_hours = (spark.read.json(hours_adls_source_path).select("employee_code", "calendar_date", "operation_hours", "its_number"))
    except Exception as e:
        raise e
else:
    print("Path does not exist: ",hours_adls_source_path)
    dbutils.notebook.exit("Extract_Dataverse")

# Read JSON of List of Employees
if check_path_exists(employees_adls_source_path):
    try:
        df_employees = (spark.read.json(employees_adls_source_path).select("email", "employee_name", "employee_code"))
    except Exception as e:
        raise e
else:
    print("Path does not exist: ",employees_adls_source_path)
    dbutils.notebook.exit("Extract_Dataverse")

df_timesheets = (df_hours.join(df_employees, df_hours.employee_code == df_employees.employee_code, "inner").select(*col_list))

df_timesheets.repartition(1).write.option("mergeSchema", "true").format("parquet").mode("overwrite").save(adls_destination_path)
dbutils.notebook.exit("Extract_Dataverse")
```

Figure 11 – Extract_Dataverse Notebook

The Extract_Dataverse notebook follows the same logic as the one that was shown before. Upon arrival, data is stored in JSON files in the RAW directory of the DL. In this case, there are two datasets, each stored in its respective folder. One of the folders stores the Dataverse table containing the employees' names and email, while the other contains the reported hours each employee took to develop a specific ITS.

This time, because there are two different datasets in different folders, the azure DL source path is set separately for each one of them. Therefore, the datasets are also read into two distinct spark data frames. Lastly, a single data frame named "df_timesheets" is created by inner joining these two spark data frames through the "employee_code" field. Once this process is completed, the columns defined in the list "col_list" are selected and the "df_timesheets" data frame is saved to its destination path as an apache parquet file that overwrites any file that was previously saved to the same destination.

```
set_spark_adls_conf()
load_date()

# Source & Destination Variables
internal_source_path   = f"""RAW/FILESHARE/InternalCapacity/{LoadDateFolder}"""
external_source_path   = f"""RAW/FILESHARE/ExternalCapacity/{LoadDateFolder}"""
adls_internal_source_path = get_adls_path(internal_source_path)
adls_external_source_path = get_adls_path(external_source_path)
destination_path = f"{internal_source_path}".replace("RAW", "Structured").replace("Internal", "")
adls_destination_path = get_adls_path(destination_path)

internalSchemaString = ("""initial_date string, end_date string, sector_department string, email string,
                        workarea_name string, planned_hours double, current_hours double, management_hours double""")

externalSchemaString = ("""initial_date string, end_date string, sector_department string, contract_type string,
                          workarea_name string, planned_hours double, current_hours double, management_hours double""")

col_list = ["initial_date","sector_department","workarea_name","email", "end_date"
           "planned_hours","current_hours","management_hours","origin","contract_type"]

# Read CSV Internal Capacity
if check_path_exists(adls_internal_source_path):
   try:
      internal_df = spark.read.csv(adls_internal_source_path, sep=';', enforceSchema = False,  schema=internalSchemaString,header=True)
      internal_df_1 = (internal_df.withColumn("origin", lit("Internal"))
                                  .withColumn("contract_type", lit(None))
                                  .select(*col_list))
   except Exception as e:
      raise e
else:
  print("Path does not exist: ",adls_internal_source_path)
  dbutils.notebook.exit("Extract_FileShare")

# Read CSV External Capacity
if check_path_exists(adls_external_source_path):
   try:
      external_df = spark.read.csv(adls_external_source_path, sep=';', enforceSchema = False,  schema=externalSchemaString,header=True)
      external_df_1 = (external_df.withColumn("origin", lit('External'))
                                  .withColumn("email", lit(None))
                                  .select(*col_list))
   except Exception as e:
      raise e
else:
  print("Path does not exist: ",adls_external_source_path)
  dbutils.notebook.exit("Extract_FileShare")

df_fileshare = internal_df_1.union(external_df_1)
df_fileshare.repartition(1).write.option("mergeSchema","true").format("parquet").mode("overwrite").save(adls_destination_path)
dbutils.notebook.exit("Extract_FileShare")
```

Figure 12 – Extract_Fileshare Notebook

The Extract_Fileshare notebook is the one responsible for reading the CSV files that arrive in the DL from the bank's file share. After setting the connection to the Azure DL, defining the load date and getting the Azure DL source and destination paths, the schemas for each of the CSV files are defined. This process is unique to this notebook, because both files contain data that is manually written. As such, it is necessary to ensure that all of the entries follow a specific structure.

After the CSV files are read into spark data frames, the necessary columns for each of them are created and/or selected. Once this process is completed, both data frames are unionized into one and saved as a parquet file to the "Structured" directory of the DL.

### *4.3.2. Transform and Load Dimension Tables*

TABLE IV – WORK AREA DIMENSION ATTRIBUTES

| Attribute | Data Type | Description | Source |
|---|---|---|---|
| workarea_id | int | Primary key of dim_contract | - |
| workarea_name | nvarchar(200) | Type of contract signed with employee | File Share |

```
set_spark_adls_conf()
load_date()

# Source & Destination Variables
fileshare_source_path = f"""Structured/FILESHARE/Capacity/{LoadDateFolder}"""
fileshare_adls_source_path = get_adls_path(fileshare_source_path)
destination_path = 'Processed/Dim_Workarea'
adls_destination_path = get_adls_path(destination_path)

# Create the Entity
spark.sql(f"""
        CREATE TABLE IF NOT EXISTS DIM_Workarea(
            workarea_id long
            ,workarea_name string
            )
         USING DELTA
         LOCATION '{adls_destination_path}'
     """)

default_value_insert ("dim_workarea", "-1", "n/a")
max_id ("dim_workarea", "workarea_id")

#read the files in the structured folder for each source into a dataframe
df_workarea = (spark.read.parquet(fileshare_adls_source_path)
                  .select("workarea_name").dropDuplicates().na_drop())
df_workarea.createOrReplaceTempView("vw_load_workarea")
```

Figure 13 – Transform_Load_DimWorkarea Notebook Part 1

The “dim_workarea” is one of the simplest dimensions since it only contains two attributes, its primary key, and the work area name. The purpose of this dimension is to store all the work areas that the bank’s employees can be contracted to work in, such as web development, business intelligence, or system administration.

Initially, the connection to the Azure DL is established and the load date is defined once again by calling the functions in Appendix A figure 32 and 33, respectively. This table’s fields originate from a single source. As such, only two Azure DL paths are made using the function in Appendix A figure 34, one for the source and one for the destination.

The destination path is called during the creation of the delta table, which is essential to enable the use of the Delta Lake system that was described in the literature review. The “default_value_insert” function that is presented in figure 37 of Appendix A is then used to insert a new instance with values “-1” and “n/a”. This instance will serve as a reference

for all the fact table entries left as “null”. This is vital to ensure that the principle of referential integrity is effectively upheld.

Last of all, the maximum id of “dim_workarea” is obtained using the “max_id” function displayed in Appendix A figure 36. Then, the file in the “Structured” directory is read into a spark data frame, the “workarea_name” field is selected, duplicates and null values are removed, and a temporary view named “vw_load_workarea” is created.

```
%sql
CREATE OR REPLACE TABLE vw_dim_workarea AS (
  select v.workarea_name
  ,case when workarea_id is null then row_number()
  over (partition by workarea_id order by workarea_id)+max_id end as workarea_id
  from vw_load_workarea v
  left join dim_workarea w on w.workarea_name = v.workarea_name
  inner join vw_max_id on 1=1
  where workarea_id is null)

MERGE INTO dim_workarea t
USING vw_dim_workarea s ON t.workarea_name = s.workarea_name
WHEN NOT MATCHED THEN INSERT *
```

Figure 14 – Transform_Load_DimWorkarea Notebook Part 2

The last part of this notebook focuses on an SQL query that creates a new virtual table named “vw_dim_workarea”. This query involves another three distinct tables: the “dim_workarea” (which is the delta table), the “vw_max_id” (which was created with the “max_id” function), and the “vw_load_workarea” (which was created as a temporary SQL view from the spark data frame in part 1 of this notebook).

The select statement of this query returns the “workarea_name” from the temporary SQL view that was previously made, as well as a derived column by the name of “workarea_id”. Because there is a left join with the “dim_workaera” and an inner join with the “vw_max_id”, it is possible to restrict the queried instances to those where the id is “null”. This way, the derived column will only be assigned to a row number for new or unique “workarea_name” values. This results in a new identifier being given to each of these new values, which is incremented by the maximum id of the delta table.

After the “vw_dim_workarea” is created, the “workarea_name” values in that table that do not match those that are already present in the delta table are inserted into the “dim_workarea” as new instances with an id higher than the previous row.

TABLE V – DEPARTMENT DIMENSION ATTRIBUTES

| Attribute | Data Type | Description | Source |
|---|---|---|---|
| department_id | int | Primary key of dim_department | - |
| sector_name | nvarchar(200) | Name of the business sector | Fileshare & Devops |
| department_name | nvarchar(200) | Name of the department | Fileshare & Devops |
| sector_department | nvarchar(200) | Sector & Department Names | Fileshare & Devops |

```
set_spark_adls_conf()
load_date()

# Source & Destination Variables
fileshare_source_path   = f"""Structured/FILESHARE/Capacity/{LoadDateFolder}"""
fileshare_adls_source_path = get_adls_path(fileshare_source_path)
devops_source_path = f"""Structured/DEVOPS/Services/{LoadDateFolder}"""
devops_adls_source_path = get_adls_path(devops_source_path)
destination_path = 'Processed/Dim_Department'
adls_destination_path = get_adls_path(destination_path)

# Create the Entity
spark.sql(f"""
        CREATE TABLE IF NOT EXISTS DIM_DEPARTMENT(
            department_id long
           ,sector_name string
           ,department_name string
           ,sector_department string
            )
         USING DELTA
        LOCATION '{adls_destination_path}'
      """)

default_value_insert ("dim_department", "-1", "n/a", "n/a", "n/a")
max_id ("dim_department", "department_id")

#read the files in the structured folder for each source into a dataframe
df_capacity = (spark.read.parquet(fileshare_adls_source_path).select("sector_department"))
df_services = (spark.read.parquet(devops_adls_source_path).select("sector_department"))
df_dept = df_capacity.union(df_services).dropDuplicates().na.drop()

df_dept = (df_dept.withColumn("sector_name", split(df_dept["sector_department"], "-").getItem(0))
                  .withColumn("department_name", split(df_dept["sector_department"], "-").getItem(1)))
df_dept.createOrReplaceTempView("vw_load_department")
```

Figure 15 – Transform_Load_DimDepartment Notebook Part 1

Unlike the previous table, the “dim_department” lists all the department names and their respective business sector. This dimension takes data from two distinct source paths: the File share source path and the DevOps source path. Once they are both defined, the “dim_department” is created with four different attributes: the “department_id”, “sector_name”, “department_name”, and “sector_department”. The last field is the only one that comes directly from both sources and contains both the sector and the department name separated by a hyphen.

Each source’s “sector_department” values are selected into separate data frames and then unionized into one where duplicates and null values are dropped. The “sector_name” and “department_name” attributes are then created by splitting the “sector_department”

field in two and assigning the first half to the "sector_name" and the second half to the "department_name". A temporary view named "vw_load_department" is created based on the "df_dept" data frame containing all three fields.

Once again, it is important to highlight that the "default_value_insert" function is called to insert an instance with the negative id "-1" into the "dim_department" table. This time however, there are more "n/a" values being inserted because this dimension has 4 attributes instead of two.

```
%sql
CREATE OR REPLACE TABLE vw_dim_department AS (
  select c.sector_name, c.department_name, c.sector_department
  ,case when department_id is null then row_number()
  over (partition by department_id order by department_id)+max_id end as department_id
  from vw_load_department c
  left join dim_department e on e.sector_department = c.sector_department
  inner join vw_max_id on 1=1
  where department_id is null)

MERGE INTO dim_deparment t
USING vw_dim_department s ON t.sector_department = s.sector_department
WHEN NOT MATCHED THEN INSERT *
```

Figure 16 – Transform_Load_DimDepartment Notebook Part 2

Following the same logic as the previous notebook, a new virtual table named "vw_dim_department" is created using three different tables: the "dim_department", the "vw_max_id" and the "vw_load_department". However, this time the "vw_max_id" was created with the "max_id" function that took as argument the "dim_department" table. Therefore, this new "vw_max_id" refers to the maximum id of the department dimension.

Like the previous dimension, the "department_id" is a derived column that is calculated using a "case when" statement that assigns a row number to each instance in the "vw_load_department" table partitioned by the "department_id" column. The value of this column is then incremented by the maximum id value that was obtained from the "max_id" function. This is necessary to guarantee that when new values are merged into this delta table, they are inserted with a new identifier that starts a unit above the maximum id of the values already in the table.

The merge of the "vw_dim_department" into the "dim_department" delta table is then performed using a match on the attribute "sector_department". When the values are not matched, it is inserted into the table as a new instance.

TABLE VI – EMPLOYEE DIMENSION ATTRIBUTES

| Attribute | Data Type | Description | Source |
|---|---|---|---|
| employee_id | int | Primary key of dim_employee | - |
| employee_name | nvarchar(200) | Name of the employee | Fileshare & Dataverse |
| origin | nvarchar(200) | Origin of the employee | Fileshare & Dataverse |
| email | nvarchar(200) | Email of the employee | Fileshare & Dataverse |

```
set_spark_adls_conf()
load_date()

# Source & Destination Variables
fileshare_source_path   = f"""Structured/FILESHARE/Capacity/{LoadDateFolder}"""
fileshare_adls_source_path = get_adls_path(fileshare_source_path)
dataverse_source_path   = f"""Structured/DATAVERSE/Timesheets/{LoadDateFolder}"""
dataverse_adls_source_path = get_adls_path(dataverse_source_path)
destination_path = 'Processed/Dim_Employee'
adls_destination_path = get_adls_path(destination_path)

# Create the Entity
spark.sql(f"""
         CREATE TABLE IF NOT EXISTS DIM_EMPLOYEE(
              employee_id long,
              employee_name string,
              origin string,
              email string
              )
          USING DELTA
          LOCATION '{adls_destination_path}'
      """)

default_value_insert ("dim_employee", "-1", "n/a", "n/a", "n/a")
max_id ("dim_employee", "employee_id")

#read the files in the structured folder for each source into a dataframe
df_capacity = spark.read.parquet(fileshare_adls_source_path).select("employee_name","origin","email")
df_timesheets = spark.read.parquet(dataverse_adls_source_path).select("employee_name","email")

df_employee = (df_capacity.join(df_timesheets, ["email"], "fullouter").na.drop().dropDuplicates(["email"])
                          .withColumn("origin", when(col("email").contains(".ext@"), lit("External")).otherwise(lit("Internal"))))
df_employee.createOrReplaceTempView("vw_load_employee")
```

Figure 17 – Transform_Load_DimEmployee Notebook Part 1

The importance of the employee dimension can be attributed to two key factors. Firstly, it holds crucial information such as the employee's origin, which may be internal or external and must be prominently exhibited in the BI report. Secondly, the BI software used to analyze this data model utilizes the employee's email to restrict access to all the pages that make up the report. Managers are granted access to the entirety of the report, which contains sensitive employee data, such as their reported work hours. However, other employees cannot view confidential information about their peers.

As shown in figure 17, the employee dimension corresponds to the "dim_employee" delta table and comprises values from two source paths: File share and Dataverse. This dimension contains four attributes and aims to store the names and emails of all the

employees that are contracted by the bank and/or report work hours in Dataverse, as well as identify an employee's origin.

The function in figure 34 of Appendix A is used to obtain the source path for both the "Capacity" parquet file and the "Timesheets" parquet file that were saved to the "Structured" directory in the extract notebooks. Both sources are read into different data frames and then joined into one through the "email" field. Then, a new column named "origin" is created, which classifies employees as "Internal" or "External" based on the email used to report the operation hours. External employees are given an email that contains the expression ".ext@". Hence, it is possible to distinguish the employees' origin based on their email addresses. After dropping null values and duplicate emails, the "df_employee" is used to create the temporary SQL view "vw_load_employee".

```
%sql
CREATE OR REPLACE TABLE vw_dim_employee AS (
  select v.employee_name, v.origin, v.email
  ,case when employee_id is null then row_number() over (partition by employee_id order by employee_id)+max_id end as employee_id
  from vw_load_employee v
  left join dim_employee c on c.email = v.email
  inner join vw_max_id on 1=1
  where employee_id is null)

MERGE INTO dim_employee t
USING vw_dim_employee s ON t.email = s.email
WHEN NOT MATCHED THEN INSERT *
```

Figure 18 – Transform_Load_DimEmployee Notebook Part 2

The table "vw_dim_employee" is created with the same four attributes defined in the "dim_department" delta table. The "employee_name", "origin" and "email" are all selected from the "vw_load_employee" SQL view that was previously created. An inner join with the "vw_max_id" and a left join with the "dim_department" are performed so that the derived column "employee_id" can be partitioned over the values that do not yet exist in the delta table. As was the case for all the other dimensions, only the new instances are selected because the "where" statement that indicates the condition under which a row will be displayed in the query result.

As before, the instances in "vw_dim_employee" that are not already in the delta table are merged into it using the "email" attribute. When there is no match, it is considered as a new instance and inserted into the "dim_employee" table.

TABLE VII – ITS DIMENSION ATTRIBUTES

| Attribute | Data Type | Description | Source |
|---|---|---|---|
| its_id | int | Primary key of dim_its | - |
| its_number | int | Number of the ITS | Devops |
| start_date | date | Start date of the ITS | Devops |
| finish_date | date | Finish date of the ITS | Devops |
| title | nvarchar(200) | Title of the ITS | Devops |
| status | nvarchar(200) | Status of the ITS | Devops |
| classification | nvarchar(200) | Classification of the ITS | Devops |
| manager_name | nvarchar(200) | Manager that emitted the ITS request | Devops |
| sector_department | nvarchar(200) | Department responsible for the ITS | Devops |
| development_area | nvarchar(200) | Development area of the ITS | Devops |
| chief_developer | nvarchar(200) | Chief developer responsible for the ITS | Devops |
| estimated_duration | float | Estimated time needed to complete the ITS | Devops |
| urgency_score | int | Score that classifies the urgency of an ITS | Devops |
| business_value | int | Business value associated with an ITS | Devops |

```
set_spark_adls_conf()
load_date()

# Source & Destination Variables
devops_source_path   = f"""Structured/DEVOPS/Services/{LoadDateFolder}"""
devops_adls_source_path = get_adls_path(devops_source_path)
destination_path = 'Processed/Dim_ITS'
adls_destination_path = get_adls_path(destination_path)

# Create the Entity
spark.sql(f"""
         CREATE TABLE IF NOT EXISTS DIM_ITS(
              its_id int
              ,its_number int
              ,start_date date
              ,finish_date date
              ,title string
              ,status string
              ,classification string
              ,manager_name string
              ,sector_department string
              ,development_area string
              ,chief_developer string
              ,estimated_duration double
              ,urgency_score int
              ,business_value int
              )
          USING DELTA
          LOCATION '{adls_destination_path}'
      """)

default_value_insert ("dim_its", "-1", "-1", "1900-01-01", "2100-12-31", "n/a", "n/a", "n/a", "n/a", "n/a", "n/a", "n/a", "0", "0", "0")
max_id ("dim_its", "its_id")

#read the files in the structured folder for each source into a dataframe
df_its = (spark.read.parquet(devops_adls_source_path).filter(col("kanban_name") == "General Kanban")
               .dropDuplicates(["its_number"]).na.fill("")
               .select("its_number", "start_date", "finish_date", "title", "status",
                       "classification", "manager_name", "sector_department", "development_area",
                       "chief_developer", "estimated_duration", "urgency_score", "business_value"))
df_its.createOrReplaceTempView("vw_load_its")
```

Figure 19 – Transform_Load_DimITS Notebook Part 1

The ITS dimension is the largest one since it contains 14 different attributes, as seen in Table VII. This dimension's only source is DevOps, and it covers the attributes of each ITS as they progress through the kanbans in Azure Boards.

Some of the most important attributes for this dimension are the "its_number", "classification", "development_area", and "estimated_duration". The ITS number is unique and carries the same value throughout the ITS development process. This number is also given as input in Dataverse when employees report their hours. The classification is the category given to an ITS by the manager that emitted the request, and can contain one of three values: "run", "grow", or "transform". The development area differs from the work area since the latter is the one that is specified on an employee's contract and the former refers only to an ITS. Most importantly, the estimated duration is one of the variables needed for equation (4) described in the previous chapter.

According to what was seen in the business and data understanding chapter, the general Kanban is the one that aggregates all the ITS that are either still in the planning phase or already being executed. This Kanban tracks the entire progress of the ITS from the moment it is created until the moment it is concluded. Therefore, the spark data frame that is used to read the source has a filter that restricts it to the ITS in the General Kanban. The necessary fields are selected, the ITS duplicates are dropped based on the values of the column "its_number" and the temporary view "vw_load_its" is created.

```
%sql
CREATE OR REPLACE TABLE vw_dim_its AS (
  select v.its_number, v.start_date, v.finish_date, v.title v.status,
  v.classification, v.manager_name, v.sector_department,  v.development_area,
  v.chief_developer, v.estimated_duration, v.urgency_score, v.business_value
  , case when its_id is null then row_number()
  over (partition by its_id order by its_id)+max_id end as its_id
  from vw_load_its v
  left join dim_its w on w.its_number = v.its_number
  inner join vw_max_id on 1=1
  where its_id is null)

MERGE INTO dim_its t
USING vw_dim_its s ON t.its_number = s.its_number
WHEN MATCHED AND (t.finish_date <> s.finish_date) THEN UPDATE SET t.end_date = s.finish_date
WHEN NOT MATCHED THEN INSERT *
```

Figure 20 – Transform_Load_DimITS Notebook part 2

The second part of this notebook follows the same logic as all the other dimension notebooks. The relevant attributes from the temporary view are selected and new rows are merged into the delta table based on the field "its_number".

### *4.3.3. Transform and Load Fact Tables*

TABLE VIII – FACT CONTRACTED ATTRIBUTES

| Attribute | Data Type | Description | Source |
|---|---|---|---|
| date_id | int | Foreign key of dim_date | - |
| employee_id | int | Foreign key of dim_employee | - |
| department_id | int | Foreign key of dim_department | - |
| workarea_id | int | Foreign key of dim_workarea | - |
| activity_id | int | Foreign key of dim_activity | - |
| contracted_hours | float | Hours contracted per employee | Fileshare |

```
set_spark_adls_conf()
load_date()

# Source & Destination Variables
fileshare_source_path   = f"""Structured/FILESHARE/Capacity/{LoadDateFolder}"""
fileshare_adls_source_path = get_adls_path(fileshare_source_path)
destination_path = 'Processed/Fact_Contracted'
adls_destination_path = get_adls_path(destination_path)

# Create the Entity
spark.sql(f"""
  CREATE TABLE IF NOT EXISTS FACT_CONTRACTED (
        date_id int
      , employee_id int
      , email string
      , department_id int
      , sector_department string
      , workarea_id int
      , workarea_name string
      , activity_id int
      , activity_name string
      , contracted_hours double
    )
  USING DELTA
  LOCATION '{adls_destination_path}'
""")

#read the files in the structured folder for each source into a dataframe
df_contracted = (spark.read.parquet(fileshare_adls_source_path).na.fill(value=0).na.fill("n/a")
                    .select ("initial_date","end_date","sector_department","workarea_name",
                             "email","planned_hours","current_hours","management_hours","origin"))
df_contracted.createOrReplaceTempView("vw_load_contracted")
```

Figure 21 – Transform_Load_FactContracted Notebook Part 1

To some extent, the structure of the fact notebooks is similar to that of the dimension notebooks. In this data model there is one fact table per data source. The fact contracted focuses specifically on data from file share. As a result, it contains the "contracted_hours" attribute that is needed for the AVC measure shown in equation (1).

The fact delta tables have additional attributes beyond what is shown in their respective attribute tables. These additional columns can be seen in the "spark.sql" statement above and are required for merging new data into the delta table. However, when a view of the table is created in the SQL Pool, these added columns are not queried.

```
%sql
CREATE OR REPLACE TABLE vw_fact_contracted AS (
  select activity_name, date_id, v.email, v.sector_department, v.workarea_name
    , coalesce(employee_id,-1) as employee_id
    , coalesce(department_id,-1) as department_id
    , coalesce(workarea_id,-1) as workarea_id
    , coalesce(activity_id,-1) as actvitity_id
    , case
        when activity_name = 'Planned' then planned_hours
        when activity_name = 'Current' then current_hours
        when activity_name = 'Management' then management_hours
      end as contracted_hours
  from vw_load_contracted v
  inner join dim_date d on (calendar_date between initial_date and (case when end_date <>'2100-12-31' then end_date
  else to_date(dateadd(month,12,current_date())) end) and calendar_month_day = 1)
  left join dim_workarea w on w.workarea_name = v.workarea_name
  left join dim_department e on e.sector_department = v.sector_department
  left join dim_employee c on c.email = v.email
  left join dim_activity m on 1 = 1 and activity_name in ("Planned","Current","Management"))

MERGE INTO fact_contracted t
USING (select * from vw_fact_contracted where horas > 0) s ON t.date_id = s.date_id
AND t.email = s.email AND t.sector_department = s.sector_department
AND t.workarea_name = s.workarea_name AND t.activity_name = s.activity_name
WHEN MATCHED AND (t.contracted_hours <> s.contracted_hours) THEN UPDATE SET (t.contracted_hours = s.contracted_hours)
WHEN NOT MATCHED THEN INSERT *
```

Figure 22 – Transform_Load_FactContracted Notebook Part 2

The second part of this fact table's notebook focuses on an SQL query that creates the table "vw_fact_contracted", which is a result of a select statement that joins 6 different tables: "dim_workarea", "dim_department", "dim_employee", "vw_load_contracted", "dim_date", and "dim_activity". Once the "vw_fact_contracted" is created, it is merged into the delta table based on all the foreign attributes that compose it.

The select statement in this query retrieves several columns from each of the joined tables and includes a coalesce function that ties in with the "default_value_insert" function that was used in each dimension. There is also a case statement within the select statement that calculates the contracted hours based on the activity name and the integer values from the columns "planned_hours", "current_hours" and "management_hours". As such, when the "activity_name" is "Planned", the contracted hours will equate to the value from the "planned_hours" column, and the same applies for the other two activities.

The only "inner join" applies to the "dim_date" dimension and has an "on" clause that specifies the join condition. This condition checks if the "calendar_date" from the "dim_date" falls within the interval between the "initial_date" and "end_date" from "vw_fact_contracted". If the "end_date" column is not equal to the default date "2100-12-31", then the end date is not altered. Otherwise, the "end_date" is set to exactly one year after the current date. This way, employees without a specific end date to their contract are not assumed to be part of the available workforce indefinitely.

TABLE IX – FACT OPERATION ATTRIBUTES

| Attribute | Data Type | Description | Source |
|---|---|---|---|
| employee_id | int | Foreign key from dim_employee | - |
| workarea_id | int | Foreign key from dim_workarea | - |
| department_id | int | Foreign key from dim_department | - |
| activity_id | int | Foreign key from dim_activity | - |
| its_id | int | Foreign key from dim_its | - |
| date_id | int | Foreign key from dim_date | - |
| operation Hours | float | Hours used to develop a specific ITS | Dataverse |

```
set_spark_adls_conf()
load_date()

# Source & Destination Variables
dataverse_source_path   = f"""Structured/DATAVERSE/TimeSheets/{LoadDateFolder}"""
dataverse_adls_source_path = get_adls_path(dataverse_source_path)
destination_path = 'Processed/Fact_Operation'
adls_destination_path = get_adls_path(destination_path)

# Create the Entity
spark.sql(f"""
  CREATE TABLE IF NOT EXISTS FACT_OPERATION (
        employee_id int
      , email string
      , workarea_id int
      , workarea_name string
      , its_id int
      , its_number
      , department_id int
      , department_name string
      , activity_id int
      , activity_name string
      , date_id int
      , operation_hours double
   )
  USING DELTA
  LOCATION '{adls_destination_path}'
""")

#read the files in the structured folder for each source into a dataframe
df_operation = (spark.read.parquet(dataverse_adls_source_path)
                    .filter(col("its_number").isNotNull())
                    .withColumn("activity_name", lit("Planned"))
                    .select("calendar_date", "its_number", "email", "activity_name", "operation_hours"))
df_operation.createOrReplaceTempView("vw_load_operation")
```

Figure 23 – Transform_Load_FactOperation Notebook Part 1

The fact operation focuses on the "operation_hours" originating from Dataverse and needed for the UC measure shown in equation (2). The business and data understanding chapter explains that the UC applies only to the planned activity. As a result, the spark data frame that reads the source path data is filtered to limit the instances to the ones that correspond to an ITS. Then, the "activity_name" column is created with the string "Planned" as a constant value throughout all the rows. Like before, a new temporary SQL view is created from the spark data frame to merge new unique instances into the fact delta table stored in the specified destination folder.

```
%sql
CREATE OR REPLACE TABLE vw_fact_operation AS (
  select date_id, v.email, v.its_number, v.activity_name
  , coalesce(employee_id,-1) as employee_id
  , coalesce(its_id,-1) as its_id
  , coalesce(v.its_number, -1) as its_number
  , coalesce(activity_id,-1) as activity_id
  , -1 as department_id, 'n/a' as sector_department
  , -1 as workarea_id, 'n/a' as workarea_name
  , sum(operation_hours) as operation_hours
  from vw_load_operation v
  inner join dim_date d on d.calendar_date = v.calendar_date
  left join dim_its i on i.its_number = v.its_number
  left join dim_employee c on c.email = v.email
  left join dim_activity m on m.activity_name = v.activity_name
  group by date_id, its_id, v.its_number, employee_id, v.email, activity_id, v.activity_name)

MERGE INTO fact_operation t
USING vw_fact_operation s ON t.date_id = s.date_id AND t.its_number = s.its_number AND t.email = s.email
AND t.activity_name = s.activity_name AND t.workarea_name = s.workarea_name AND t.sector_department = s.sector_department
WHEN MATCHED AND (t.operation_hours <> s.operation_hours) THEN UPDATE SET t.operation_hours = s.operation_hours
WHEN NOT MATCHED THEN INSERT *
```

Figure 24 – Transform_Load_FactOperation Notebook Part 2

Once again, based on the SQL view created in part 1 of the notebook, the table "vw_fact_operation" is created via an SQL query that combines multiple other tables: "dim_workarea", "dim_department", "dim_employee", "dim_date" and "dim_activity". The "vw_fact_operation" is then merged into the "fact_operation" delta table utilizing each dimension's relevant foreign attributes.

Similarly to the previous fact table, the foreign key null values obtained in the query result are replaced by "-1". This is essential to safeguard the principle of referential integrity and ensure that null keys in the fact table refer to the "-1" dimension keys. However, unlike the previous fact table, the main metric of the fact operation is the sum of "operation_hours" grouped by all the attributes of each dimension.

It is also worth noting that the attributes "department_id" and "sector_department" from the department dimension are given a "-1" identifier and a "n/a" string for all the query results. The same happens for the "workarea_id" and "workarea_name" of the workarea dimension. This is because in part 1 of this notebook, the spark data frame used to create the "vw_load_contracted" does not contain any of these columns since they simply do not exist in the source. When the bank's IT staff first set up Dataverse a few years ago, they just created a gateway for the employees to access through their email that accepted only three inputs: the calendar date, the ITS number, and the number of hours dedicated to developing it. Thus, these commands are a temporary placeholder until the bank's system administrators add the work area and department input fields to Dataverse.

TABLE X – FACT ALLOCATED ATTRIBUTES

| Attribute | Data Type | Description | Source |
|---|---|---|---|
| department_id | int | Foreign key of dim_department | - |
| activity_id | int | Foreign key of dim_activity | - |
| its_id | int | Foreign key of dim_its | - |
| date_id | int | Foreign key of dim_date | - |
| allocated_hours | nvarchar(200) | Hours allocated to develop a specific ITS | Devops |

```
set_spark_adls_conf()
load_date()

# Source & Destination Variables
devops_source_path   = f"""Structured/DEVOPS/Services/{LoadDateFolder}"""
devops_adls_source_path = get_adls_path(fileshare_source_path)
destination_path = 'Processed/Fact_Allocated'
adls_destination_path = get_adls_path(destination_path)

# Create the Entity
spark.sql(f"""
         CREATE TABLE IF NOT EXISTS FACT_ALLOCATED(
              ,date_id int
              ,its_id int
              ,its_number int
              ,department_id int
              ,sector_department string
              ,activity_id int
              ,activity_name string
              ,allocated_hours double
              )
          USING DELTA
          LOCATION '{adls_destination_path}'
      """)

#read the files in the structured folder for each source into a dataframe
df_allocated = (spark.read.parquet(devops_adls_source_path)
                    .withColumn("activity_name", lit("Planned")).filter(col("kanban_name") == "Execution Kanban")
                    .select("its_number","kanban_name","workarea_name", "activity_name",
                            "allocation_date", "allocated_hours","sector_department"))
df_allocated.createOrReplaceTempView("vw_load_allocated")
```

Figure 25 – Transform_Load_FactAllocated Notebook Part 1

The fact allocated is the last of the fact tables and contains another key metric, the "allocated_hours". This attribute is used to calculate the AC measure covered in the last chapter, shown in equation (3). Like the UC measure, the AC only applies to the planned activity. Hence, the spark data frame that reads the dataset in the DevOps source path has the "activity_name" values set to "Planned".

Moreover, the ITS selected in the "df_allocated" data frame must be present in the Execution Kanban. This relates to the way the bank's Azure Boards were set up. For an ITS to be present in the Execution Kanban, it already went through the planning stage. Therefore, the ITS at that point in time already has an estimated duration and is ready to receive an allocation of hours.

```
%sql
CREATE OR REPLACE TABLE vw_fact_allocated AS (
  select date_id, p.its_number, p.sector_department, p.activity_name
  , coalesce(its_id,-1) as its_id
  , coalesce(p.its_number,-1) as its_number
  , coalesce(department_id,-1) as department_id
  , coalesce(activity_id,-1) as activity_id
  , sum(allocated_hours) as allocated_hours
  from vw_load_allocated p
  inner join dim_date d on calendar_date = allocation_date
  left join dim_its i on i.its_number = p.its_number
  left join dim_workarea w on w.workarea_name = p.workarea_name
  left join dim_department e on e.sector_department = p.sector_department
  left join dim_activity m on m.activity_name = p.activity_name
  group by date_id, its_id, p.its_number, activity_id, p.activity_name, department_id, p.sector_department
  having allocated_hours > 0)

MERGE INTO fact_allocated t
USING vw_fact_allocated s ON t.date_id = s.date_id AND t.its_number = s.its_number
AND t.sector_department = s.sector_department AND t.activity_name = s.activity_name
WHEN MATCHED AND (t.allocated_hours <> s.allocated_hours) THEN UPDATE SET (t.allocated_hours = s.allocated_hours)
WHEN NOT MATCHED THEN INSERT *
```

Figure 26 – Transform_Load_FactAllocated Notebook Part 2

The “vw_fact_allocated” is created by selecting attributes from “vw_load_allocated”, made in the first part of this notebook, as well as attributes from the dimensions in the join clauses. This time, only the dimensions “dim_department”, “dim_activity”, “dim_its” and “dim_date” were joined. After coalescing the ids from each dimension, the “allocated_hours” fact is created similarly to the “operation_hours”.

There are plenty of ITS projects that are already planned out but have not received any hour allocations yet due to a lack of resources. To prevent these cases from being loaded into the delta table, the “having” clause operates on the grouped rows and excludes the groups that do not meet the specified condition. Moreover, all the dimensions are left-joined with the fact table, except for the “dim_date”. The rows from both tables that match the condition “calendar_date = “allocation_date” are combined, creating a new intermediate table that is then left-joined with other tables.

As was seen in each fact notebook, the merge statement is made using the main attributes of all the dimensions that connect to the fact. When there is a match in the “merged into” statement and the value of the hours is different from the one that is already in the delta table, then an update is performed. This is particularly important in this notebook because the allocated hours fact is much more volatile than any other fact. Additionally, all the fact tables seen so far lack a primary key because the combination of all the foreign keys from each dimension act as a composite primary key for the fact tables. This is the reason why there is no need to call the “max_id” in any fact notebook.

### *4.3.4. ETL Orchestration*

The notebooks that were seen so far must be executed regularly so that new instances get merged into the model's entities automatically. To achieve this, the ETL_Master notebook was created. The code in this notebook calls the "run_notebook" function from Appendix A figure 38, to run the "Extract" and "Transform_Load" notebooks on a loop.

```
extract_notebooks = ["Extract_Devops", "Extract_Dataverse", "Extract_Fileshare"]

for item in extract_notebooks:
  run_notebook (execution_id, item, "-", {"LoadDate":f"{LoadDate}"})


dimension_dict = {"Transform_Load_DimDate": "dim_date",
                  "Transform_Load_DimActivity": "dim_activity",
                  "Transform_Load_DimWorkarea": "dim_workarea",
                  "Transform_Load_DimDepartment": "dim_department",
                  "Transform_Load_DimEmployee": "dim_employee",
                  "Transform_Load_DimITS": "dim_its"}

for key, value in dimension_dict.items():
  run_notebook (execution_id, key, value, {"LoadDate":f"{LoadDate}"})


fact_dict = {"Transform_Load_FactContracted": "fact_contracted",
             "Transform_Load_FactOperation": "fact_operation",
             "Transform_Load_FactAllocated": "fact_allocated"}

for key, value in fact_dict.items():
  run_notebook (execution_id, key, value, {"LoadDate":f"{LoadDate}"})
```

Figure 27 – ETL_Master Notebook

First and foremost, the extract notebooks for each one of the sources are executed. Consequently, the transform and load notebooks are run for each one of the dimensions and fact tables that integrate the relational model. Last of all, the orchestration of the ETL is set up in Azure Synapse Studio by creating an orchestration pipeline. This makes it possible to trigger the system chain that was initially presented.

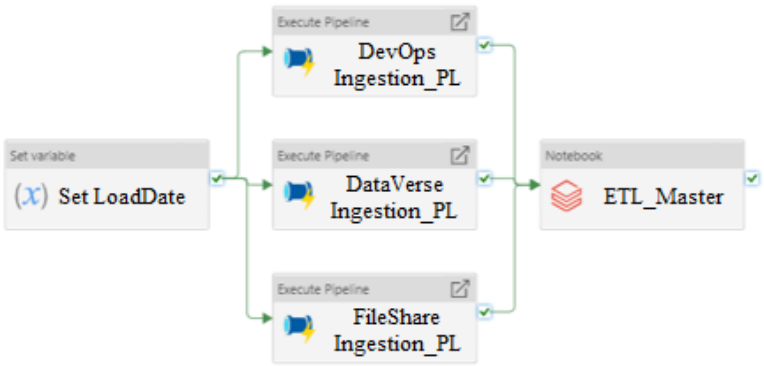


Figure 28 – Synapse ETL Orchestration Pipeline

This entire process is set to trigger every day at 4 AM. As such, this master pipeline runs the data ingestion pipelines for each of the three sources. After that, the ETL_Master Databricks notebook is executed and consequently, the notebooks that transform and load the model's entities are executed as well. Therefore, new batches of data continuously travel from Raw to Structured and from Structured to Processed.

Once this process is completed, all the table entities that comprise the logical data model are present in the DL's "Processed" folder. Therefore, it is possible to query all of them into the SQL Pool so that they can be seen in Power BI. This was achieved by creating transact-SQL scripts for each entity in Azure Synapse Studio. An example of how these scripts look for a given dimension and fact table can be seen in the figure below. Naturally, the data types for each attribute of each entity will match the attribute tables defined in Tables IV, V, VI, VII, VIII, IX and X.

**SQL Script 1**

```
CREATE EXTERNAL TABLE Dim_Name (
    Primary Key      int,
  Dim_Attribute 1    int,
  Dim_Attribute 2    nvarchar(200),
  Dim_Attribute 3    nvarchar(200)
  )
  WITH (
  LOCATION = "Processed/Dim_Name"
  DATA_SOURCE = [Client_Data_Lake],
  FILE_FORMAT = [SynapseDeltaFormat]
  )
GO

CREATE EXTERNAL TABLE Fact_Name (
  Foreign Key 1  int,
  Foreign Key 2  int,
  Foreign Key 3  date,
  Fact_Attribute  float
    )
    WITH (
    LOCATION = "Processed/Fact_Name/"
    DATA_SOURCE = [Client_Data_Lake]
    FILE_FORMAT = [SynapseDeltaFormat]
    )
GO
```

**SQL Script 2**

```
CREATE VIEW VW_Dim_Name
AS
SELECT
 [Primary Key],
 [Dim_Attribute 1],
 [Dim_Attribute 2],
 [Dim_Attribute 3]

FROM Dim_Name
GO



CREATE VIEW VW_Fact_Name
AS
SELECT
 [Foreign Key 1],
 [Foreign Key 2],
 [Foreign Key 3],
 [Fact_Attribute]

FROM Fact_Name
GO
```

Figure 29 – Azure Synapse Transact-SQL Script Example

Upon revisiting the ETL State Diagram depicted in figure 9, it becomes evident that every step illustrated therein has been successfully executed. The ingestion pipelines presented in figure 28 guarantee the completion of the initial step (ingest), while the "Extract" notebooks handle the second step (extract and structure). The subsequent step (transform and load) is effectively accomplished by the "Transform_Load" notebooks, and finally, the SQL scripts ensure the fulfillment of the fourth and final step (query).

### *4.4. Data Analysis*

This section covers the frontend of the architecture, where Power BI is used to access the table views made available in the Synapse Serverless SQL Pool. As such, by importing all the tables in the relational model, it is possible to create a business intelligence (BI) report with visualizations that can then be published in the Power BI service of the client (Microsoft Power BI, 2022). Because the BI report connects to Synapse, and in turn Synapse runs the system's backend, all the visualizations in the report are automatically updated as new batches of data go through the ETL process.

Although the report is still under development, some visualizations can already be seen in the figure displayed in the next page, which had to be heavily censored due to GDPR articles invoked by the bank. Nevertheless, the visualizations in the report were created based on the measures and KPIs defined in the Data Understanding chapter. Thus, the measures defined in equations (1), (2), (3), and (4) had to be implemented in Power BI as Data Analysis Expressions which can be seen in the figure below.

```
AVC = SUM(Fact_Contracted[contracted_hours])
UC = SUM(Fact_Operational[operational_hours])
AC = SUM(Fact_Allocated[allocated_hours])
Total UC = CALCULATE(SUM(Fact_Operational[operational_hours]), ALL(Data_Aux[Data]))
Total AC = CALCULATE(SUM(Fact_Allocated[allocated_hours]), ALL(Dim_Data[Data]))
RC = SUM(Dim_ITS[Estimated Duration])-[Total AC]-[Total UC]
```

Figure 30 – AVC, AC and UC Measures

The report has an initial menu that allows the user to access three different pages, one for each type of activity. After selecting one of them, such as the "Planned Activity", the user arrives to the menu displayed in the first frame of figure 31. This, in turn, gives the user a drill-through access for each capacity: Available (1), Allocated (2), and Used (3).

The time constraints of measures (2) and (3) were ensured by the time filter in the upper-right corner of their respective frames in figure 31. The filter in the Allocated Capacity page restricts visualizations to the future specified number of months $(t > t_0)$. Whereas the filter in the Used Capacity page restricts visualizations to the past specified number of months $(t < t_0)$. Each measure and KPI is reflected in the graphs displayed throughout the pages, except for KPI 4 and measure 4. These are only shown in a private table accessed through the "details" tab in both the Allocated and Used Capacity pages.


Figure 31 – Power BI Report Pages

# 5. CONCLUSION AND FUTURE RESEARCH

The immense amount of data that enterprises deal with daily, resulted in a paradigm shift that affects how companies operate and commercial banks were no exception. To avoid the substantial capital expenditure of maintaining on-premises DWs, many organizations began implementing cloud-base solutions due to its modus operandi. The cloud's on-demand self-service, high scalability, and low operational expenditures enabled the creation of data warehousing and data analytics solutions that satisfied businesses' fast-paced environment. This attracted large corporations with vast amounts of data that need to be analyzed, such as the client mentioned in this work.

Despite its many advantages, cloud-based projects can be very complex and require constant communication between the development team and the business representatives that will be the system's end-users. In this specific case, all the development team members had weekly meetings with the bank's executives and daily interactions with other bank employees. This was vital to ensure that data was consistent and that the visualizations in the report matched the client's business reality. If this were not the case, every decision the bank's decision-makers took would be based on false premises.

The decision to opt for the Lakehouse DBA proved to be critical to developing a project of this size in a timeframe that satisfied the client's urgency. Furthermore, this design fit perfectly because it was entirely made for a specific section of this large enterprise. Nevertheless, this architecture should not be interpreted as a completely new architecture. It is primarily an updated version of an older architecture tailored to function in a modern cloud environment. As a result, some of the limitations associated with Kimball's DBA are still applicable to this modern design.

If the bank keeps building upon this architecture in an even larger scale across multiple departments, compatibility issues are likely to arise. Distinct departments are organized differently and may register capacity in a different way. As such, having dimensions that need to fit multiple business sections will require a constant change in their data structure and ETL process. Alternatively, if an adaptation of CIF were implemented right from the start, all these concerns would be addressed during its design stage. However, the time it would take to create an efficient company-wide data warehousing solution would be significantly higher due to its increased complexity.

Furthermore, this work demonstrates that building a DW in the cloud can be advantageous due to the possibility of integrating data from multiple sources online. This gives rise to another conclusion: cloud data warehouses are very source dependent. This is mainly because of the ETL process that populates it with data. If these independent data sources, such as DevOps and Dataverse, are subject to any updates, it is not certain that the ETL process based on the non-updated versions will keep working as intended.

The bank is also bound to these platforms for the time being. If, for whatever reason, it is necessary to change from DevOps or Dataverse to other platforms, the ETL process will undoubtedly break. Consequently, the Power BI report will stop matching the client's business reality, and project managers will no longer be able to use the system to make informed business decisions.

It is also essential to note that this work has two limitations that may induce some bias in the conclusions drawn. Firstly, this data warehousing solution was entirely developed with a specific set of cloud services. There are several other PaaS and SaaS offered by Azure and other cloud providers that may have different functionalities. Secondly, data warehousing solutions depend on the business context of the enterprise in which they are applied. Organizations with different structures are likely to store data differently, resulting in distinct DWs that are not interchangeable.

There are numerous opportunities for future research in this field, mainly because data warehousing in the cloud is a relatively recent subject that is still largely unexplored. Most of the existing DW architecture literature dates back to when cloud computing was not prevalent. Thus, there is an opportunity to devise innovative approaches that adapt earlier designs to the cloud. Additionally, there is still much to learn about the interactions between various platforms within the cloud, particularly regarding SaaS and PaaS interactions. As such, further research in this area is highly encouraged.

Overall, cloud data warehousing is just one of many digital innovations that can be used to modernize businesses. Keeping up with technological advances is crucial for companies to remain competitive and commercial banks are no different. The banking sector is heavily influenced by disruptive technologies which spark the need for innovative solutions such as the one that was described throughout this work.

## REFERENCES

Ariyachandra, T., & Watson, H. (2010). Key organizational factors in data warehouse architecture selection. *Decision support systems*, 49(2), 200-212

Armbrust, M., Das, T., Sun, L., Yavuz, B., Zhu, X., Murthy, M., …, Zaharia, M. (2020, August). Delta Lake: high-performance ACID table storage over cloud object stores. *Proceedings of the VLDB Endowment, Vol. 13, No. 12*

Armbrust, M., Ghodsi, A., Xin, R., & Zaharia, M. (2021, January). Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics. *In Proceedings of CIDR* (p. 8).

Azure Boards (2022). *Azure Boards*. Available from: https://azure.microsoft.com/en-us/products/devops/boards/ (Accessed: 22/11/2022)

Azure Data Lake (2022). *Introduction to Azure Data Lake Storage Gen2*. Available from: https://learn.microsoft.com/en-us/azure/storage/blobs/data-lake-storage-introduction (Accessed: 22/11/2022)

Azure Databricks (2022). *What is the medallion lakehouse architecture?* Available from: https://learn.microsoft.com/en-us/azure/databricks/lakehouse/medallion (Accessed: 22/11/2022)

Azure Files (2022). *File Share*. Available from: https://learn.microsoft.com/en-us/previous-versions/windows/desktop/mscs/file-share (Accessed: 22/11/2022)

Azure Synapse Pipelines (2022). *Pipelines and activities in Azure Synapse Analytics.* Available from: https://learn.microsoft.com/en-us/azure/data-factory/concepts-pipelines-activities?tabs=data-factory (Accessed: 22/11/2022)

Azure Synapse Serverless SQL Pool (2022). *Serverless SQL pool in Azure Synapse Analytics*. Available from: https://learn.microsoft.com/en-us/azure/synapse-analytics/sql/on-demand-workspace-overview (Accessed: 22/11/2022)

Begoli, E., Goethert, I., Knight, K. (2021). A Lakehouse Architecture for the Management and Analysis of Heterogeneous Data for Biomedical Research and Mega-biobanks. *IEEE International Conference on Big Data 2021*

Costa, C. J., & Aparicio, J. T. (2020). POST-DS: A methodology to boost data science. In *2020 15th iberian conference on information systems and technologies (CISTI)* (pp. 1-6). IEEE.

Databricks Glossary (2022). Medallion Architecture. Available from: https://www.databricks.com/glossary/medallion-architecture (Accessed: 22/11/2022)

Hambardzumyan, S., Tuli, A., Ghukasyan, L., Rahman, F., Topchyan, H., Isayan, D., …, Buniatyan, D. (2023, January). Deep Lake: a Lakehouse for Deep Learning. *Annual Conference on Innovative Data Systems Research*. Amsterdam, Netherlands

Harby, A. & Zulkernine, F. (2022). From Data Warehouse to Lakehouse: A Comparative Review. *IEEE International Conference on Big Data 2022*. Osaka, Japan.

Inmon, W.H. (2005). *Building the Data Warehouse*. 4th Edition. Indianapolis, Indiana: Wiley Publishing, Inc.

Jornal de Negócios (2021). *Estará o setor da banca a cumprir a promessa do digital.* Available from: https://www.jornaldenegocios.pt/negocios-em-rede/deloitte/detalhe/estara-o-setor-da-banca-a-cumprir-a-promessa-do-digital (Accessed: 10/10/2022)

Kimball, R. & Ross, M. (2013). *The Data Warehouse Toolkit*. 3rd Edition. Indianapolis, Indiana: John Wiley & Sons, Inc.

Maurin, L., Santos, R., Delanote, J., Rizzoli, I. (2021). *EIB Investment Survey 2021 – Portugal Overview.* European Investment Bank

Mendes, J.N (2022). *Intervenção do Secretário de Estado do Tesouro no plenário da Assembleia da República.* Available from: https://www.portugal.gov.pt/pt/gc23/comunicacao/intervencao?i=intervencao-do-secretario-de-estado-do-tesouro-no-plenario-da-assembleia-da-republica (Accessed: 10/10/2022)

Microsoft Dataverse (2022). *What is Microsoft Dataverse?* Available from: https://learn.microsoft.com/en-us/power-apps/maker/data-platform/data-platform-intro (Accessed: 22/11/2022)

Microsoft Power BI (2022). *What is Power BI Desktop?* Available from: https://learn.microsoft.com/en-us/power-bi/fundamentals/desktop-what-is-desktop (Accessed: 22/11/2022)

Nambiar, A. & Mundra, D. (2022) An Overview of Data Warehouse and Data Lake in Modern Enterprise Data Management. *Big Data Cognitive Computing*, 6, 132.

Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) [2016] OJL 119 4.5.2016

Shearer, C. (2000) The CRISP-DM Model: The New Blueprint for Data Mining. *Journal of Data Warehousing* 5 (4), 13-22.

Unipartner (2023). *About us*. Available from: https://www.unipartner.com/about (Accessed: 10/10/2022)

## APPENDICES

### *Appendix A – Common Functions*

```
def set_spark_adls_conf():
  spark.conf.set("fs.azure.account.auth.type", "#########")
  spark.conf.set("fs.azure.account.auth.provider.type", "org.apache.hadoop.fs.azure_datalake.auth.Client_Credentials")
  spark.conf.set("fs.azure.account.auth.client.id", dbutils.secrets.get(scope = "client_databricks_key_vault",
                                                                        key = "#########"))
  spark.conf.set("fs.azure.account.auth.client.secret", dbutils.secrets.get(scope = "client_databricks_key_vault",
                                                                            key = "#########"))
  spark.conf.set("fs.azure.account.auth.client.endpoint",dbutils.secrets.get(scope = "client_databricks_key_vault",
                                                                             key = "#########"))
```

Figure 32 – Function set_spark_adls_conf

```
def load_date ():
  dbutils.widgets.text("LoadDate", "","")
  dbutils.widgets.get("LoadDate")

  global LoadDateFormat, LoadDateFolder, LoadDate
  LoadDate = getArgument("LoadDate")
  LoadDateFormat = datetime.strptime(str(LoadDate), '%Y-%m-%d')
  LoadDateFolder = LoadDateFormat.strftime('%Y/%m/%d')
```

Figure 33 – Function load_date

```
def get_adls_path(path):
    if not path:
      raise Exception("Exception: get_adls_path : parameter path is empty")
    adls_url = dbutils.secrets.get(scope = "client_databricks_key_vault", key = "#########")
    return adls_url + path
```

Figure 34 – Function get_adls_path

```
def check_path_exists(path):
  try:
    dbutils.fs.ls(path)
    return True
  except Exception as e:
    return False
```

Figure 35 – Function check_path_exists

```
def max_id (table_name, table_id):
    assert isinstance(table_name, str), "The table name must be a string"
    assert isinstance(table_id, str), "The table id must be a string"
    try:
        df_id = spark.sql(f"select coalesce(max({table_id}),0) as max_id from {table_name} where {table_id}>0")
        df_id.createOrReplaceTempView("vw_max_id")
        return("A temporary view named 'vw_max_id' with the column 'max_id' was successfuly created")
    except Exception as e:
        raise e
```

Figure 36 – Function max_id

```
def default_value_insert (table_name, *inserts):
  assert isinstance(table_name, str), "All the function arguments must be strings"
  string_ints = ["-", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
  insert_list = []

  for i in inserts:
    assert isinstance(i, str), "All the function arguments must be strings"
    if i.startswith(tuple(string_ints)):
      insert_list.append(i)
    else:
      insert_list.append(f'"{i}"')

  sql_insert = ', '.join(insert_list)

  try:
    df_dim = spark.sql(f"select * from {table_name}")
    table_id = df_dim.columns[0]

    if df_dim.filter(col(f"{table_id}").contains(f"{insert_list[0]}")).count() == 1:
      return (f"The default value {insert_list[0]} is already present in the table {table_name}")
    else:
      spark.sql(f"INSERT INTO {table_name} VALUES ({sql_insert})")
      return (f"The default values: ({sql_insert}), were inserted as a new instance in the table {table_name}")

  except Exception as e:
    raise e
```

Figure 37 – Function default_value_insert

```
def run_notebook (execution_id, notebook, entity, args):
  start_time = datetime.now()
  message = ""
  status = ""
  try:
    status  = dbutils.notebook.run(notebook,0,args)
  except Exception as e:
        end_time = datetime.now()
        message=e
        insert_log(execution_id, notebook, entity, -1 ,message, start_time, end_time)
        raise e
  end_time = datetime.now()
  insert_log(execution_id, notebook, entity, status ,message, start_time, end_time)
```

Figure 38 – Function run_notebook

```
def insert_log(execution_id, notebook, entity, status ,message, start_date, end_date):
   spark.sql(f""" INSERT INTO LOG VALUES
               ('{execution_id}', '{notebook}', '{entity}', '{status}' ,'{message}', '{start_date}', '{end_date}')
           """)
```

Figure 39 – Function insert_log