Dipartimento di Informatica, Bioingegneria,
Robotica e Ingegneria dei Sistemi (DIBRIS)

Reengineering the Analytical Back-end of an Energy Distribution Company

by

Davide Ponzini

Master Thesis

DIBRIS, Università di Genova

Via Opera Pia, 13 16145 Genova, Italy http://www.dibris.unige.it/

Università degli Studi di Genova

Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi (DIBRIS)

MSc Computer Science
Data Science and Engineering

Reengineering the Analytical Back-end of an Energy Distribution Company

by

Davide Ponzini

Advisors: Giovanna Guerrini, Alessandro Vassalli
Examiner: Gianna Reggio

April, 2024

Table of Contents

Abstract

Migrating the analytical back-end of a company from on-premise to the Cloud can be a very complex task. It is important not only to choose an appropriate **software architecture**, but also to pay attention to the **organization** of the development process.

A proper **Data Warehouse structure** must also be chosen to allow efficient storage and retrieval of information. The back-end also needs to keep its data constantly updated, so **custom downloaders** need to be developed. These downloaders, using a wide array of different techniques, retrieve information from a large number of data providers with various frequencies.

This thesis contributes to the migration on the Cloud of the Data Warehouse of an energy market company.

In addition, to assess the quality of the migration, two kinds of tests have been performed. **Data quality tests** determine the reliability of the custom downloaders developed, while **performance tests** estimate the improvements offered by the new back-end.

Multiple **issues** specific to the energy market domain, which heavily influenced the whole development process, are analyzed in this document.

Part I

Background and Related Work

Chapter 1

Introduction

The thesis is concerned with the migration of an energy company Data Warehouse to the cloud.

This chapter will provide a description of the reasons that made this migration needed, as well as the desired outcome. My own role, as well how I contributed to the process will also be explained in this chapter.

1.1 Context

This project has been developed at the Italian head quarters of **Axpo Italia SpA**, located in Genoa.

Axpo Italia SpA is the Italian subsidiary of Axpo Group, providing energy products and solutions to the Italian market [1]. Axpo Group has its headquarters in Switzerland, being the largest producer of renewable energies and one of the leading energy traders worldwide [2].

Axpo needs to store a large amount of heterogeneous data coming from several different sources, as well as to be able to access these information to perform a wide range of operations, ranging from Business Intelligence analyses to retrieving specific sets of data to execute different algorithms.

The needs of different company departments constitute a complex scenario, composed of a large number of tools and processes, as well as a very complex interaction network between these tools and existing databases.

Axpo has hired **Power Reply**, a company that specialises in consulting services and development for the Energy and Utilities departments, for the task of migrating the Data Warehouse from on-premise databases to the cloud.

I have worked closely with both Axpo employees and the Reply team assigned to the migration process.

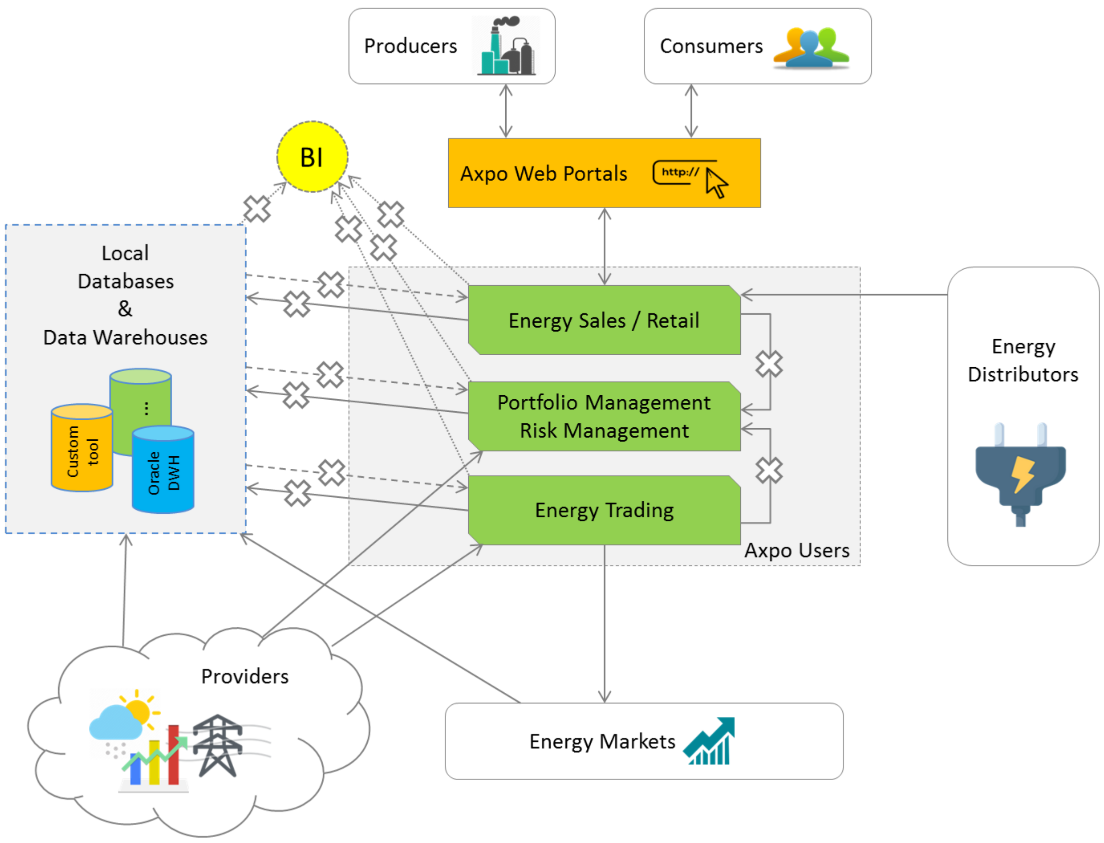


Figure 1.1: Current data communication structure among Axpo departments and data sources.
X's represent integration software.

1.2 Initial State

Different sources provide different data to the company. Some information are stored in specific databases, while other data first need to be processed by specific tools before being stored. In several cases, databases store both the raw data and the results of the computations performed by specific tools.

Figure 1.1 shows an overview of data communication among Axpo departments and the different external data sources.

As we can see, each department communicates with multiple local databases, as well as with other departments. All these communications are mediated by several pieces of integration software (represented by X's in the figure).

The data is stored in a few databases, which are used by several departments and contain several kinds of information. All these databases are on-premise. This brings several advantages, such as a great degree of availability and ease of management, but has also some problems, which have become more and more relevant in the last years. These problems are related to database performances and data quality as well as to the management of the processes used to communicate with the databases.

1.2.1 Integration Software

One of the most apparent issues is the high quantity of integration software used by the company.

Each department needs data in a specific format. Often the information are stored differently. During retrieval, custom programs are used to manipulate data in advanced ways, for example by merging results from a query with information from a different source or by performing cleaning operations too complex to be expressed in SQL.

Management A few years ago, the quantity of integration software was low and easy to manage. However, as the company grew, more programs have been added and the situation has become harder and harder to manage. The company does not have some kind of documentation or schema which can be consulted to understand which programs are used and what they do.

Sometimes subscriptions are paid for tools which are no longer needed, since a new tool could do the same things. However, the high complexity of the system and the lack of documentation can make this problem difficult to notice.

Performance Having a high number of tools means that data must go through several steps. Sometimes multiple tools are required for the needs of specific users.

This has an obvious impact on performance, since a large number of computations is performed on the data.

Bug tracing Given the high number of tools, if an issue is found with some data, it is extremely difficult to trace its cause, since it could reside in any of the tools used in the process.

As a consequence, debugging may become a very complex and time-consuming activity.

1.2.2 Database Performance

The databases has been slowing down over the last years, given the constantly increasing amount of data. Some tables contain hundreds of millions of records, which sometime make even performing simple queries a long operation.

In some cases, performing a simple query (for example with two equality checks on the `WHERE` clause) can take up to a few minutes.

These long execution times have become problematic in some critical processes, which have to be completed daily before a specific time.

1.2.3 Data Quality

An additional problem was related to data quality.

There were no data quality assurance processes in-place, which means there was no way to validate the quality of new data.

Actual name	Correct name
IM_020****	IM_20****
IM_0S1****	IM_S1****
IM_036****	PVI_036****_001
IM_036****_01	PVI_036****_001
UP_PARCOELICO_1	UP_PARCOEOLICO_1
UP_PARCOEOLICO_1	UP_PARCOEOLICO_1

Table 1.1: Some UP naming inconsistencies.

This led to data cleaning operations executed directly into the queries, which severely impacted performances negatively.

For example, some users needed to extract data related to the energy production of some UPs[1]. The information are stored on a single table, but presented many problems related to data quality.

The main issues are:

Inconsistent plant names There is no quality assurance control when receiving plant names from providers. These names are most likely manually input by clients, since we noticed they sometimes contain typos.

For example a specific UP, called *UP_ParcoEolico_1* was one time named *UP_ParcoElico_1*.

Table 1.1 shows some naming inconsistencies.

Different date conventions Each provider handles DST[2] changes in a different way[3]. The main issues are that there are no normalization actions in-place and that data from different providers are stored in the same table.

This causes a problem when extracting data from the table, since each provider needs *ad hoc* logic in the query, which not only makes it harder to read, but also negatively impacts performance.

As an example, let us consider a query needing to extract information from a table containing data from two weather forecast providers, say *Provider A* and *Provider B*[4]. However, the two providers handle DST in different ways. Upon further analysis, we discovered that even a single provider doesn't have a consistent approach across all years.

In detail, we have:

[1] *Unità di produzione* (Production Unit). An UP is a plant which produces electric energy.

[2] *Daylight Saving Time*. Several countries advance clocks during summer months so that evening daylight lasts longer, while sacrificing normal sunrise times.

[3] During DST changes we can either have 23 or 25 hours in a single day. In the first case, some provider name the hours from 0 to 23, while others may keep the standard 24 hour format but skip the hour 2. In the second case, some providers use a 25 hours notation, while others call the hours when the change occurs 3A and 3B. There are also several providers which don't use any of the above notations but have their own specific format.

[4] The actual names of the providers are trade secrets and, as a consequence, cannot appear in this document.

- Lack of consistency across providers
- Lack of consistency across different years of the same provider

For example, the day when DST ends we are supposed to have 25 hours, since the hour between 2 and 3 a.m. is repeated twice.

The data provided by *Provider A* are ranged 1 to 25, which is what mathematical models expect, while *Provider B* only provides 24 hours and discards the 25th. In this case it is necessary to remap all the hours after 2 a.m., which is when the DST change occurred.

The behavior of *Provider B* isn't however consistent across all years, since we noticed that the data prior to 2017 have 25 hours. As such it is important to take into account the year when deciding whether to remap the hours or not.

Data scattering These tables are populated with data from several different sources.

For example, a consistent number of information are extracted from a custom tool which acts as a database, but has many more functionalities. However, the data contained in this custom tool comes from a different one, *Trimp*, which is a metering tool[5] developed by a third-party organization.

This high interaction between tools adds an additional layer of complexity to the system, impacting negatively both performance and understanding.

Long execution times Some queries contain several `CASE ... WHEN ...` instructions, sometimes even with multiple nested `SUBSTRING` and `CHARINDEX` operations. These data cleaning operations severely impact performances, since the database has to perform long operations for each record.

Figure 1.2 shows an example of cleaning operations performed in the query.

On the other hand, if there had been some data quality guarantee, these queries could have been greatly simplified and their performance would have improved greatly.

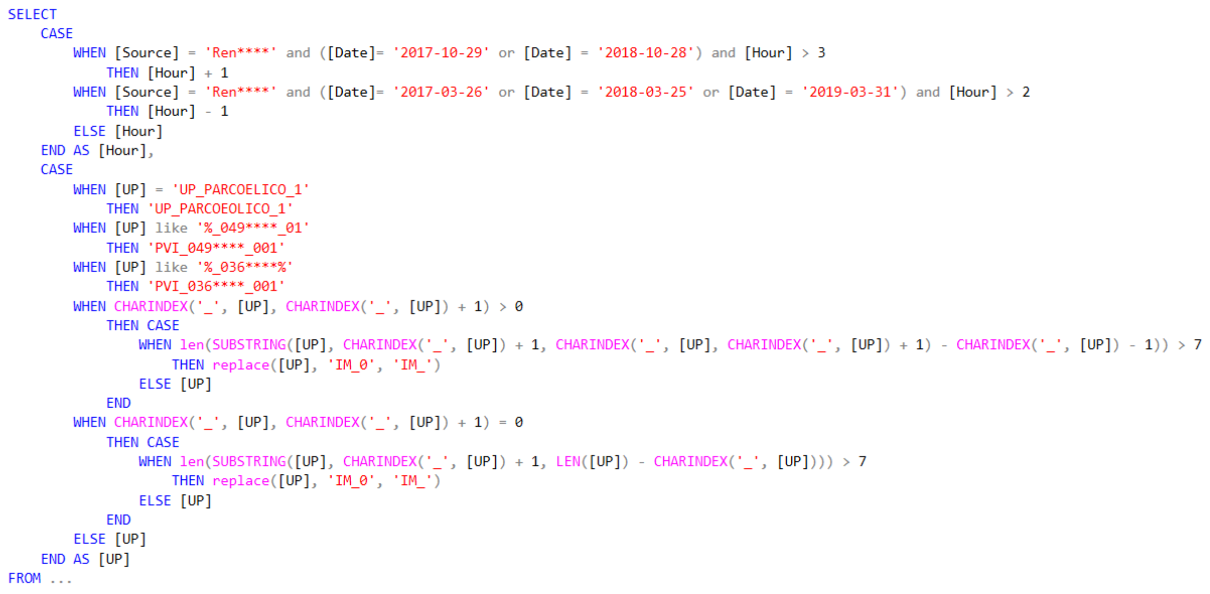
For example, the query shown in Figure 1.2 would have simply become `SELECT [Hour], [UP] FROM ..` In this case, this query could also benefit from indexing, while in the current state this situation is impossible, since indexes do not work when data are manipulated (for example by a `SUBSTRING` operation).

1.3 Goal

The problems described in the previous section lead to the decision of migrating the existing structure to the cloud.

The goal is to create a single cloud-based data warehouse, which can be directly queried by multiple departments both for their needs as well as for performing Business Intelligence analyses.

[5] A metering tool regularly receives values from energy meters and performs some basic operations on them depending on the meter type. For example old meters send a cumulative count of how much energy has been consumed since the meter was installed, while newer models only send the amount consumed since the last reading.



```sql
SELECT
    CASE
        WHEN [Source] = 'Ren****' and ([Date]= '2017-10-29' or [Date] = '2018-10-28') and [Hour] > 3
            THEN [Hour] + 1
        WHEN [Source] = 'Ren****' and ([Date]= '2017-03-26' or [Date] = '2018-03-25' or [Date] = '2019-03-31') and [Hour] > 2
            THEN [Hour] - 1
        ELSE [Hour]
    END AS [Hour],
    CASE
        WHEN [UP] = 'UP_PARCOELICO_1'
            THEN 'UP_PARCOEOLICO_1'
        WHEN [UP] like '%_049****_01'
            THEN 'PVI_049****_001'
        WHEN [UP] like '%_036****%'
            THEN 'PVI_036****_001'
        WHEN CHARINDEX('_', [UP], CHARINDEX('_', [UP]) + 1) > 0
            THEN CASE
                WHEN len(SUBSTRING([UP], CHARINDEX('_', [UP]) + 1, CHARINDEX('_', [UP], CHARINDEX('_', [UP]) + 1) - CHARINDEX('_', [UP]) - 1)) > 7
                    THEN replace([UP], 'IM_0', 'IM_')
                ELSE [UP]
            END
        WHEN CHARINDEX('_', [UP], CHARINDEX('_', [UP]) + 1) = 0
            THEN CASE
                WHEN len(SUBSTRING([UP], CHARINDEX('_', [UP]) + 1, LEN([UP]) - CHARINDEX('_', [UP]))) > 7
                    THEN replace([UP], 'IM_0', 'IM_')
                ELSE [UP]
            END
        ELSE [UP]
    END AS [UP]
FROM ...
```

Figure 1.2: Data cleaning operations performed in-query.

The communication with the data warehouse is handled by a single could-based integration layer, which is in charge of performing a wide range of operations, from data cleaning to scheduled data retrieval from various sources.

Figure 1.3 shows the desired outcome of the migration.

Each department is now expected to communicate only with the data warehouse and, in case information from different departments are needed, to extract these results from the data warehouse and not from the department itself.

This means that the data warehouse will always contain all the information and that they will always be up-to-date.

A similar situation applies also to data retrieve from external providers, which now will be retrieved by the cloud platform and not by specific users.

Business Intelligence As we can see, Business Intelligence operations are much simpler compared to Figure 1.1.

In the old scenario, multiple sources need to be queried, and each source has its own integration layer. With the new architecture, all the data resides in one place, which means that analyses can be directly performed on the data warehouse.

Moreover, the new architecture is expected to have higher performances, since operations can be executed on distributed and scalable machines.

1.4 Contribution

My role in this project was to supervise the development process of the Data Warehouse, making sure that the result would suit the needs of Axpo employees.

Activities such as downloading, cleaning and loading the data into the Data Warehouse were done exclusively by Reply, given their very technical nature. Nonetheless, I needed

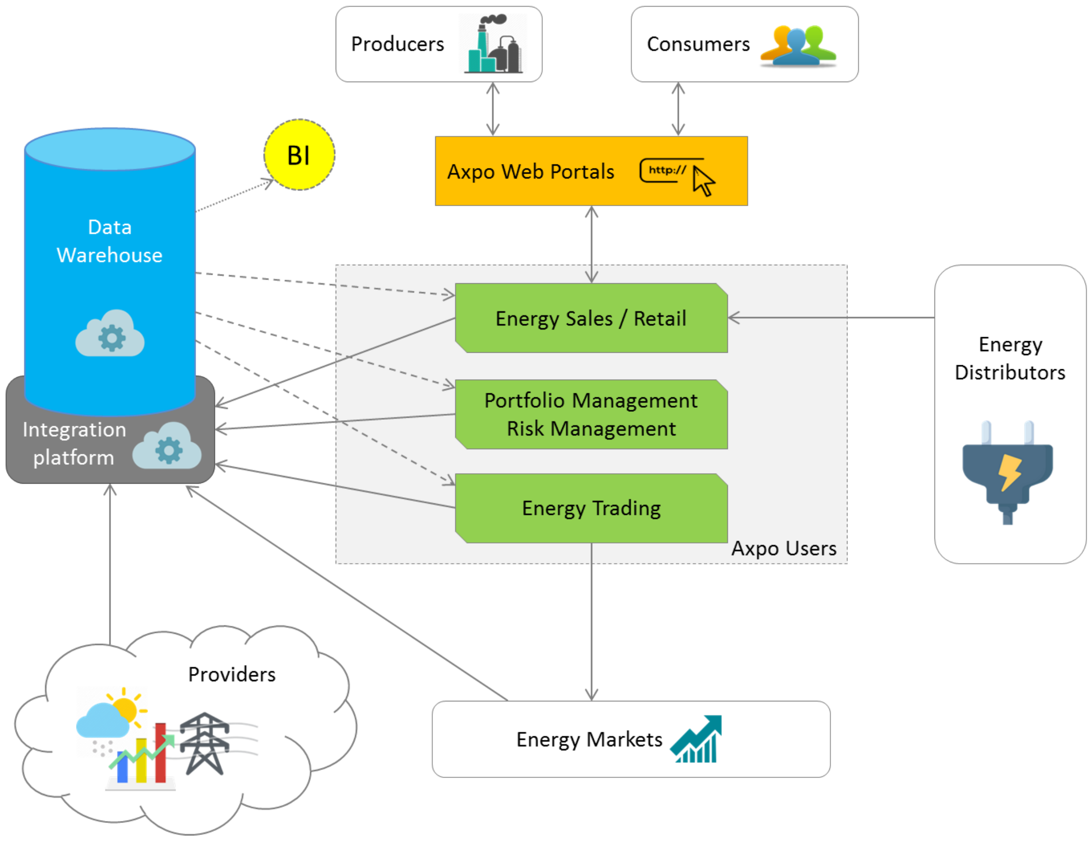


Figure 1.3: Desired data communication structure among company departments and external data sources.

to analyze their work to understand if the process was done correctly.

I was also tasked with asserting the correctness of the data stored on the Data Warehouse, as a result of the ETL process developed by Reply.

1.4.1 Supervision

I played a role in supervising the work done by Reply during the migration process.

This task presented many different aspects.

Requirements I was tasked with making sure that the requirements given by Axpo to Reply were consistent with the actual needs of Axpo employees.

The latter had already created a document detailing which information the Data Warehouse will need to download. This document was originally used by Reply for planning their work.

However, while I was analyzing several queries used by Axpo employees, I happened to notice some data sources which were not part of the planning given to Reply.

I proceeded to notify Axpo and Reply, and adjustments to the plan were made.

Organization I also worked closely with the Axpo Project Manager tasked with supervising Reply, helping him organize meetings to check the work progress.

During development, some problems with the workflow chosen by Reply became apparent to us, and we requested some changes.

The description provided in chapter 4 is the result of this mediation between us and Reply.

Data Warehouse structure I also had to make sure that the Data Warehouse structure was suitable for storing efficiently the data downloaded.

To this goal, I analyzed the various solutions proposed by Reply. These analyses were carried out together with Axpo employees, when their technical knowledge was also required.

The solutions proposed by Reply, apart from a few small cases, proved to be adequate to the requirements, and no major changes have been requested.

1.4.2 Testing

My second main task was assessing the quality of the data stored in the Data Warehouse, as a result of the ETL process.

I performed several tests, comparing the information stored in the Data Warehouse with the ones present in the on-premise databases.

Multiple problems were encountered, and I worked closely with Reply employees to solve these issues. When it was needed, I also requested the help of Axpo specialists, especially for understanding some complex scenarios.

This process will be described in detail in chapter 7.

Performance I also decided to carry out some extra tests to assess the performance of the Data Warehouse, comparing it with the on-premise databases.

These tests highlighted the effectiveness of the migration process.

Chapter 2

Related Work

A large number of migration processes have already been carried out in different scenarios outside of the company.

The migration process discussed in this document presents, however, three main peculiarities:

- A huge amount of data required to migrate.
- The need to create custom extractors to download new data every day.
- A high downloading complexity.

This chapter discusses the state-of-the-art techniques used to perform migration processes, as well as various problems related to cloud migrations. A final part will focus on different testing approaches for making sure the information stored on the cloud are identical to those already present on-premise.

2.1 State of the Art

The amount of data Axpo required to migrate is huge and highly heterogeneous and not much information are available for such amount and complexity. A large number of papers has been published regarding migrations to the Cloud, but few are relevant with this project.

The following publications are the most relevant regarding the migration process carried out by Axpo.

Architecture design Some publications focus on systematic architecture design techniques, starting from the target domain [3]. The design proposed is, however, too much formal and high-level: the document focuses on understanding the needs of the company to choose which components are needed in the Cloud architecture. The needs are however described in a high-level notation, as shown in Figure 2.1. Depending on the identified requirements, different rules are applied. These rules are used for specifying software architecture dependencies. For example, one of the rules is:

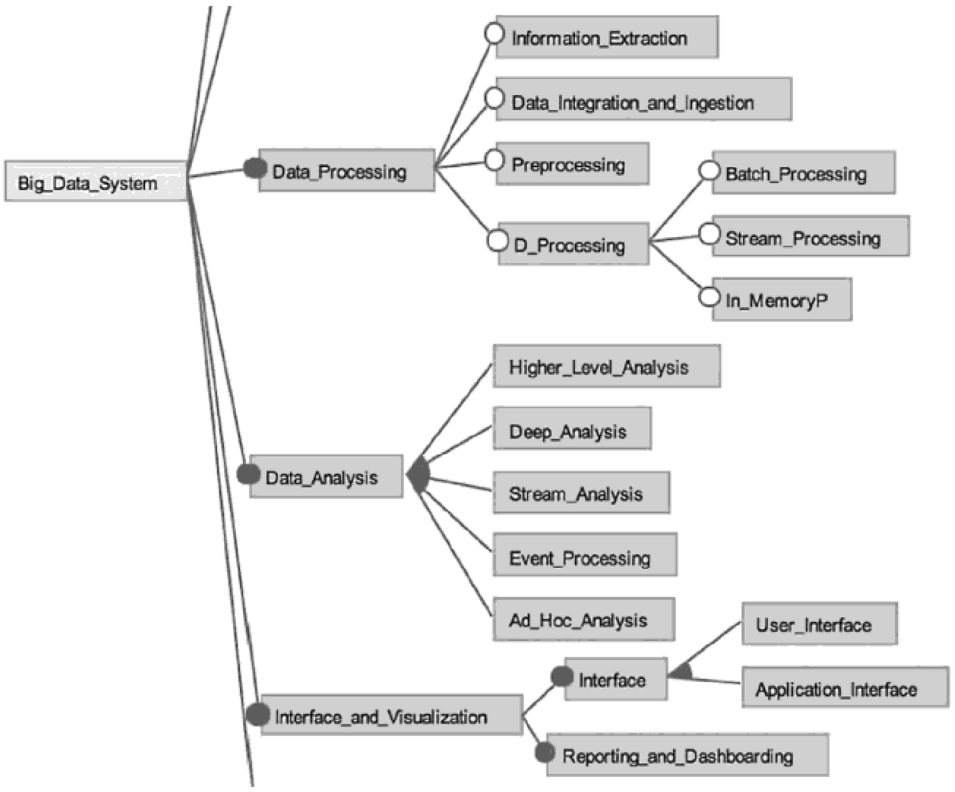


Figure 2.1: High-level architecture design [3].

```
If AD HOC ANALYSIS is selected then enable INFORMATION VIRTUALIZATION on
                component INFORMATION MANAGEMENT SERVER.
```

This design is meant for engineering a whole software architecture from scratch, not just for re-engineering a single component (in this case, the analytical back-end) while leaving the others unchanged. Moreover, the company was already aware of their needs, since they were the same of the on-premise structure.

Issues relevant to the re-engineering process required by Axpo are also mentioned, such as the need to perform cleaning operations on data, but are not investigated or described in detail.

Migration process Other publications focus on the re-engineering process itself, as well as some expected issues [4]. The document, however, presents only a brief description of each problem, inviting readers to carry out further analyses on their own. The issues exposed in the publication have been also encountered during the migration process carried out by Axpo, and will be described in more details in the chapter 5 and in section 6.2.

The publication also contains some remarks about potentialities offered by the Cloud, even though they are all discussed in general. Some of these aspects are also described in more details in this section and in chapter 3.

2.2 Moving to the Cloud

Moving workloads and data to the Cloud is becoming a reality for a very large number of companies.

According to a survey carried out in 2018 by *Druva*, a cloud data management and security company, 90% of the 170 companies interviewed have either already moved to the Cloud or intend to do so in the near future [5].

The main drivers of these migrations are disaster recovery, ease of management, and archival.

Data in the Cloud is stored in multiple places around the world, meaning that in case of disasters no data will be lost, thanks to redundancy protocols. The architecture itself, on the other hand, is directly managed by the Cloud provider, meaning that companies will have one less problem to worry about.

There are also other factors which make Cloud migrations more appealing, including:

- Automatic scalability.
- Reliability.
- High availability.
- Remote access.
- Easier cost forecasting.

2.3 Choosing the Right Process

Before beginning a Cloud migration, it is important to take decisions with respect to several factors, which are going to influence the whole process. The most relevant are described in what follows.

2.3.1 Cloud Platform Choice

The first choice is whether to use a single Cloud provider or to split the application across multiple providers. Both choices have their own advantages and disadvantages [6].

2.3.1.1 Single provider

Using a single Cloud provider is relatively simple. The development team needs to learn just one set of cloud APIs, and the application can take advantage of everything the chosen Cloud provider offers.

The downside to this approach is vendor lock in. Moving the application to a different provider could require just as much effort as the original cloud migration. Additionally, having a single cloud provider might negatively impact the company ability to negotiate important terms, such as pricing and SLAs[1], with the cloud provider.

2.3.1.2 Multiple providers

Using multiple providers can provide several advantages, depending on the chosen model.

Different applications on different Clouds The simplest approach is to run a set of applications in one Cloud provider and a different set in another. This approach gives the company increased business leverage with multiple providers, as well as flexibility for where to put applications in the future. Each application can also be optimized for the provider on which it runs.

Same application on multiple Clouds A different approach consists of running some parts of a single application on one provider, and other parts on a different one.

In this way, it is possible to effectively use the strengths of each provider. Let's assume, for example, that *Provider A* has better AI capabilities than the other providers, while *Provider B* offers cheaper and more efficient storage. A solution using this approach would store all data on *Provider B*, and transfer only the information needed for AI models to *Provider A*.

There is, however, a risk tied to this approach. The performance of the whole application depends on the performance of each provider, meaning that the slowest provider could become a bottleneck for the whole application.

Similarly, downtime of a single provider could result in the whole application not working as intended.

Cloud-agnostic applications A third possibility is building an application which can run on *any* provider.

There are multiple advantages, such as a higher flexibility when negotiating with providers, as well as the ability to run the same application on multiple providers at the same time. The latter allows more dynamic workload balancing, since it is possible to easily shift work between different providers.

The downside is the inability to use each of the strengths of each provider, reducing the benefits of hosting the application in the Cloud.

2.3.2 Migration Techniques

After having decided which and how many providers use, it is necessary to choose how to migrate the application to the Cloud.

There are three main approaches, depending on the integration level of the application with the Cloud [5, 6].

[1] *Service Level Agreement.* It is the minimum level of service that a carrier will deliver to the company per agreement. When the service dips below that level, the company can open a repair ticket.

Lift-and-shift The simplest and fastest approach is moving the application *as-is* to the cloud.

With this option, however, the application cannot benefit from each of the Cloud unique services, meaning that the improvements will be reduced.

Lift-and-refit An improved version of the *lift-and-shift* approach.

After the application has been moved to the Cloud, some modifications are made where possible, enabling the application to leverage the strengths of the Cloud provider.

The downsides are that this approach requires more time than *lift-and-shift*, as well as in-depth knowledge of the application. Moreover, there is also the risk of increasing the complexity of the application.

Cloud native The best solution, although also the most time expensive, is to rebuild the application from scratch on the Cloud.

In this way it is possible to use each of the Cloud features to the fullest, maximising both performance and readability.

The downside is, however, the large amount of time needed to rewrite the whole application.

2.3.3 Deployment

The last important decision is how and when to switch over the production system from the legacy on-premise solution to the new cloud version.

There are two possible solutions.

All at once The switch is performed once the whole application has been successfully ported to the Cloud.

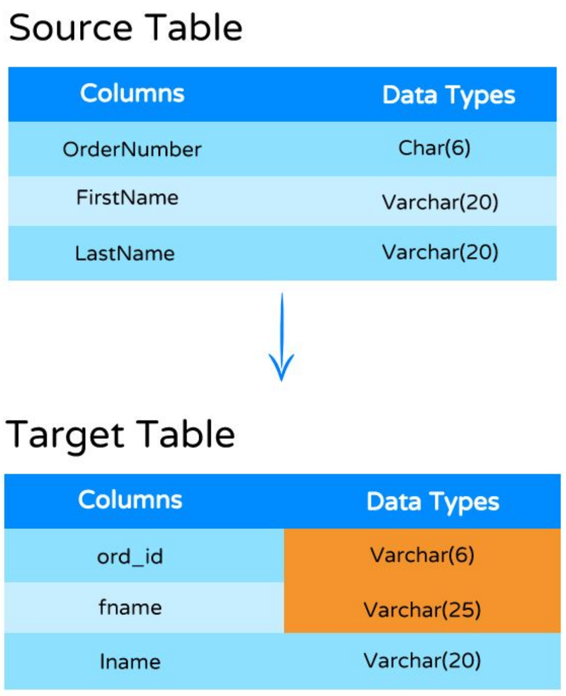
All services are switched at the same moment from on-premise to Cloud.

Incremental After each part of the application has been migrated, some users move their workload to the Cloud. This process is then repeated until all users have successfully moved to the Cloud.

For very large applications, this is the best solution, since it isn't necessary to wait for the whole migration to be complete (which could require years) before using it.

2.4 Testing

Once the source database has been migrated to the Cloud, it is important to verify the correctness of the data contained in it. Having wrong information may cause poor business decisions, as well as other issues.



Source Table

Columns	Data Types
OrderNumber	Char(6)
FirstName	Varchar(20)
LastName	Varchar(20)

Target Table

Columns	Data Types
ord_id	Varchar(6)
fname	Varchar(25)
lname	Varchar(20)

Figure 2.2: Different data types during migration.

A wide array of tests can be performed on a Data Warehouse to assess the correctness of the data it contains [7, 8].

Most state-of-the-art tests, however, are not suited for the energy market domain, due to the nature of the data required.

We will now analyze some of the various data quality tests widely used in the world, providing a final comparison between state-of-the-art testing techniques and the particular requirements of the energy market domain.

2.4.1 Test Types

Data type coherency When performing a migration, it is important to pay attention to the format chosen for the table, not only in terms of table structure but also regarding data types used for each field.

As we can see from Figure 2.2, it is possible to have mismatches between the source and destination tables.

This can lead to unexpected problems. For example, if a `VARCHAR` field supports less characters on the destination table, the string may get truncated.

NULL values Some columns are not supposed to contain `NULL` values.

It is important to make sure that any `Not NULL` constraints are respected, both by reproducing the constraint in the destination table and by specifically looking for `NULL` values.

The presence of `NULL` values is often indicative of problems in the ETL process.

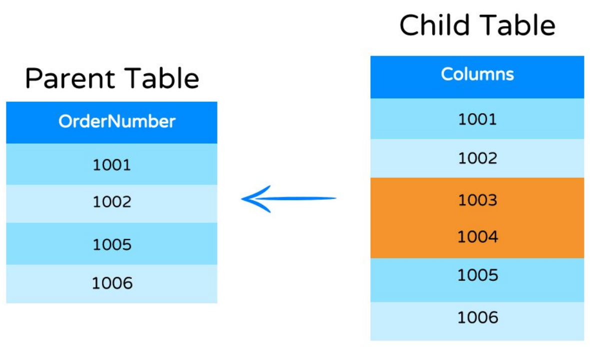


Figure 2.3: Orphan records.

Duplicate records Another important test is controlling whether the destination table contains any duplicate entries where it's not supposed to, such as IDs.

Having multiple instances of the same record may result in inaccurate results, leading to poor business decisions.

Orphan records Orphan records are often indicative of missing data or problems in the ETL process.

An orphan record can be define as an incomplete foreign key: on a child table we have an ID which doesn't appear in the parent table. For example, we may have billing information for a specific user, but no information about the user himself.

Figure 2.3 shows an example of orphan records.

Unknown data If possible, it is also a good idea to make sure values are between a reasonable range.

For example, having a client over 150 years old is surely indicative of some problems. The same can be said for negative ages.

This test can however only be applied on specific types of information: in some cases all possible values may be acceptable, such as profits or losses generated from a particular energy plant.

Min/max validations Min/max validation can assess the range of particular kinds of data.

Having different ranges between the two databases can indicate either missing data (if the range is too small), or unexpected data (if the range on the cloud exceeds the local one).

Timeliness The same information may change in time. Downloading the same information in two different moments may give different results, if that value is updated over time.

When performing a migration, it is important to pay attention to these different versions and to retrieve the correct one.

2.4.2 Energy Market Domain Issues

Some of the tests described above are not suited for the energy market domain for a variety of reasons.

Forecast updates Energy forecasts are often updated, meaning that *Timeliness* test types would fail.

However, this is not a problem, since more recent versions are more accurate and are the only ones actually needed.

As a consequence, these tests cannot be applied on this kind of data.

Date ranges The local database contains some data related to several years ago, which are no longer needed by any user.

As such, *min/max validation* tests need to be slightly tweaked. Instead of comparing the exact ranges, the cloud is allowed have less data, as long as these information are very old.

Unexpected data values For most values used in the energy market domain, all possible values are acceptable. Values outside of an “expected” range may indeed be indicative of unexpected market events and must as such be treated as correct.

This kind of tests, as a consequence, cannot be applied on this particular process.

Chapter 3

Cloud Services

Moving the analytical back-end to the cloud requires choosing a cloud service provider. Axpo and Reply concorded on relying on Microsoft Azure Cloud Computing Platform & Services.

This chapter will provide a description of the main cloud technologies used for the project.

3.1 Introduction

Microsoft Azure is a cloud computing platform offering several services created by Microsoft. Many different programming languages, tools and frameworks, including both Microsoft-specific and third-party software and systems are supported[9].

Tools, called *Resources* can be organized under a *Resource Group*, a sort of "container" which allows to set configurations (e.g. permissions) at a global level.

In this chapter we will analyze the most important tools used for the ETL and analysis processes under the *ToyCase* enviroment. We will also provide a description of the various environments used.

Figure 3.1 shows all resources available in the *ToyCase* environment.

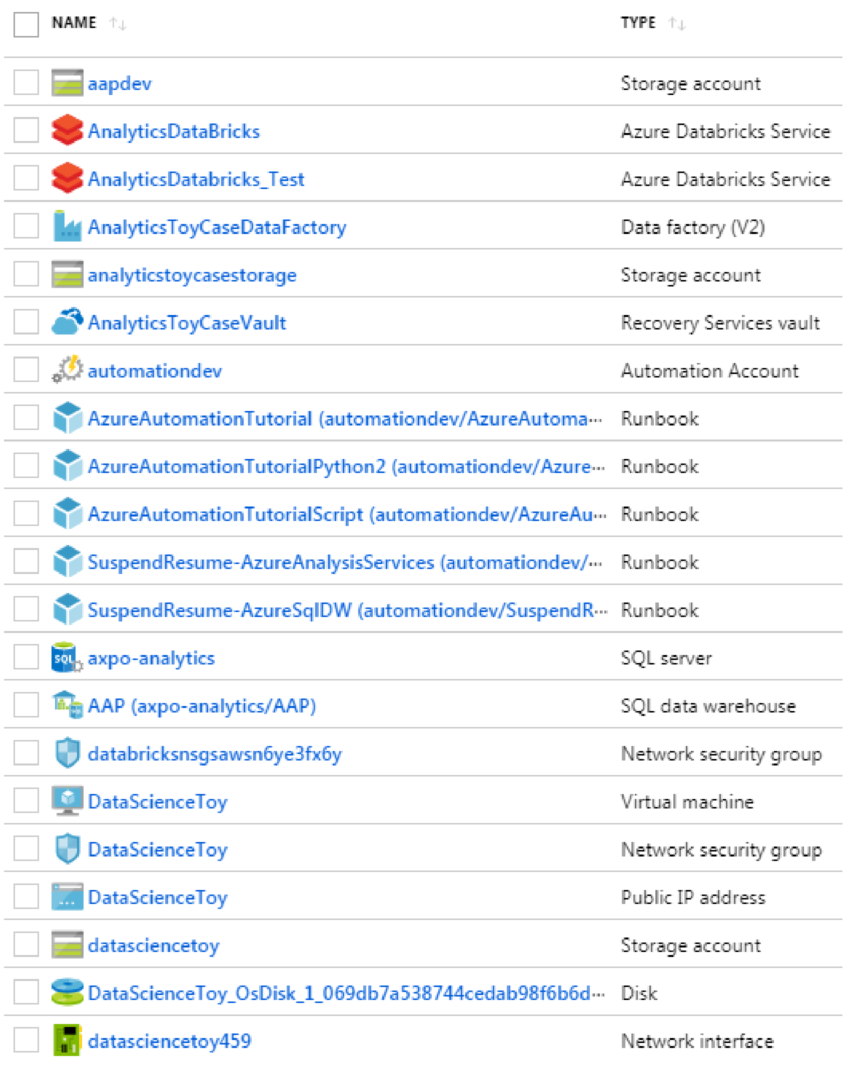
3.2 Enviroments

We define as *environment* a `resource group`, i.e. a collection of resources that can be managed together.

We created two environments in order to answer to the different needs of the project. These environment are used respectively for development and production.

All environments have the same tools and configuration, the only difference is how they are used.

Development The development environment, nicknamed *Toycase*, is where Reply develops new functionalities.



NAME	TYPE
aapdev	Storage account
AnalyticsDataBricks	Azure Databricks Service
AnalyticsDatabricks_Test	Azure Databricks Service
AnalyticsToyCaseDataFactory	Data factory (V2)
analyticstoycasestorage	Storage account
AnalyticsToyCaseVault	Recovery Services vault
automationdev	Automation Account
AzureAutomationTutorial (automationdev/AzureAutoma···	Runbook
AzureAutomationTutorialPython2 (automationdev/Azure···	Runbook
AzureAutomationTutorialScript (automationdev/AzureAu···	Runbook
SuspendResume-AzureAnalysisServices (automationdev/···	Runbook
SuspendResume-AzureSqlDW (automationdev/SuspendR···	Runbook
axpo-analytics	SQL server
AAP (axpo-analytics/AAP)	SQL data warehouse
databricksnsgsawsn6ye3fx6y	Network security group
DataScienceToy	Virtual machine
DataScienceToy	Network security group
DataScienceToy	Public IP address
datasciencetoy	Storage account
DataScienceToy_OsDisk_1_069db7a538744cedab98f6b6d···	Disk
datasciencetoy459	Network interface

Figure 3.1: Microsoft Azure *ToyCase* Environment Resources.

This environment may contain work-in-progress scripts, along with some tests on functionalities or data. Moreover, the data warehouse contains only a portion of the data needed by the algorithms, since the purpose of this environment is to test whether the downloader works.

Production The production environment is architecturally identical to the development environment, but it is actively used by end-users and their algorithms.

This environment contains up to several years of data history for each provider and constantly downloads new information when scheduled.

All the code on this environment must always work without errors. To guarantee this, all the code has to be previously tested on the development environment.

3.3 Resources

This section will describe the main Azure resources used for downloading, storing and processing the data.

3.3.1 Databricks

Azure Databricks is an Apache Spark-based analytics platform optimized for the Microsoft Azure cloud services platform [10].

Databricks allows full management of Spark clusters, along with online editing of scripts, which are used to create and submit jobs to the cluster.

3.3.1.1 Folder structure

Databricks provides a single workspace shared across all users. It is possible to create and manage folders and files under this workspace.

By default, there are two folders with special permissions already set-up.

Shared All objects stored under the `Shared` folder are public, meaning that all users have read, write and execute permissions on them.

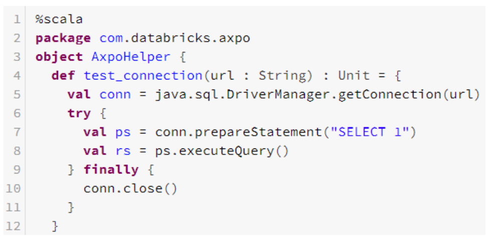
This folder can be used as a quick way to share items between users, especially if they have different or incompatible permissions.

Users The `Users` folder contains a subfolder for each user who has access to Databricks.

By default, each user is the administrator of their own folder, but they can also give permissions to other users.

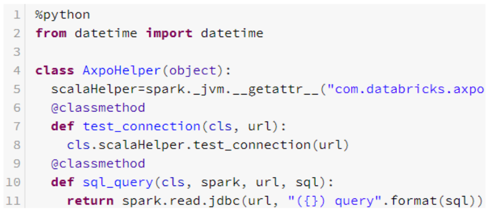
3.3.1.2 Coding

Databricks allows programming in the following languages:



```scala
%scala
package com.databricks.axpo
object AxpoHelper {
  def test_connection(url : String) : Unit = {
    val conn = java.sql.DriverManager.getConnection(url)
    try {
      val ps = conn.prepareStatement("SELECT 1")
      val rs = ps.executeQuery()
    } finally {
      conn.close()
    }
  }
```

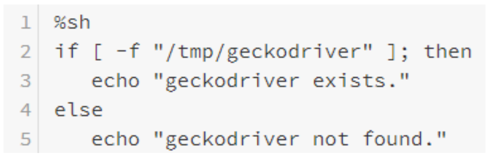
(a) Scala



```python
%python
from datetime import datetime

class AxpoHelper(object):
  scalaHelper=spark._jvm.__getattr__("com.databricks.axpo
  @classmethod
  def test_connection(cls, url):
    cls.scalaHelper.test_connection(url)
  @classmethod
  def sql_query(cls, spark, url, sql):
    return spark.read.jdbc(url, "({}) query".format(sql))
```

(b) Python



```bash
%sh
if [ -f "/tmp/geckodriver" ]; then
   echo "geckodriver exists."
else
   echo "geckodriver not found."
```

(c) Bash

Figure 3.2: Different scripting languages in Databricks.

- Python
- Scala
- R
- SQL
- Bash

Scripts are divided into several cells, similarly to a Python notebook.

Each cell must be written in a single programming language, but a script can contain cells in different languages.

In order to specify how a script should be interpreted it is sufficient to write the language used in the first line of the cell.

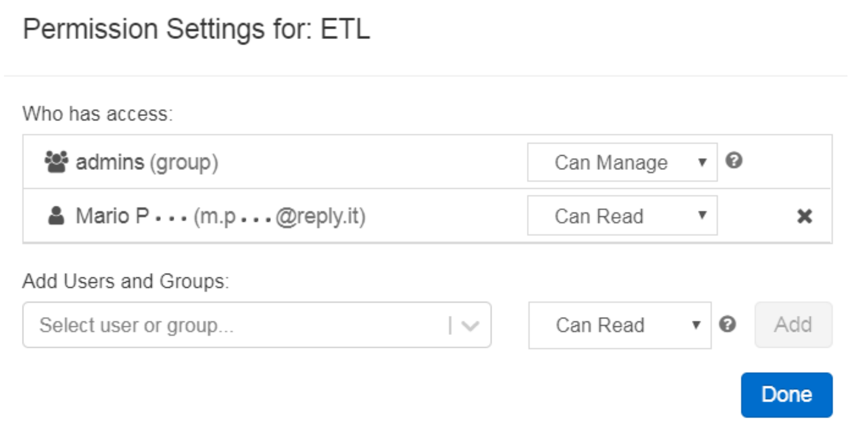
Figures 3.2a, 3.2b and 3.2c show a cell in Scala, one in Python, and one in Bash, respectively. These different cells can even belong to the same script.

3.3.1.3 Permissions

Admins can specify advanced permissions for each folder or even file.

It is possible to assign permissions to both single users and groups.

Figure 3.3 shows an example of permissions, which are set on a folder called ETL. All users belonging the the group “admins” can manage the folder and its content, while the user “Mario P***” can only read its content.


Permission Settings for: ETL

Who has access:

admins (group)	Can Manage	
Mario P . . . (m.p . . . @reply.it)	Can Read	✖

Add Users and Groups:

Select user or group... — Can Read

Figure 3.3: Azure Databricks folder permissions.

Options There are different permission options. These are:

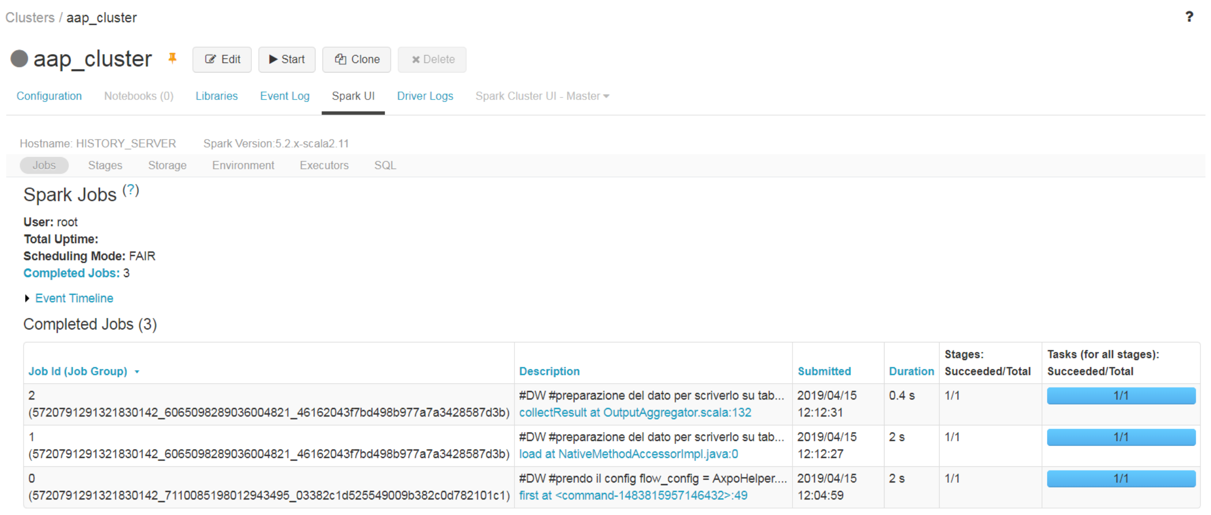
1. No permissions
    - Users cannot do anything with this object
2. Can Read
    - Users can read and comment notebooks in the folder
3. Can Run
    - Users can read, comment, attach/detach and run commands in notebooks in the folder
4. Can Edit
    - In addition to the above, users can also modify notebooks in the folder
5. Can Manage
    - Users can do all of the above, as well as assign permissions to others

Groups Permissions can also be assigned at group-level. Groups are a collection of users which share the same permissions.

A default group, called "admins" has manage permissions on all objects in Databricks. Other groups can be freely created and managed.

Default permissions By default, admins have *manage* permissions on all items in Databricks. They have full access to any file, which means that they can also see proprietary algorithms or other company secrets.

Each user has also *manage* permissions on their personal folder, under the directory "Users". This place acts as their personal workspace, in which they can perform any action.



Hostname: HISTORY_SERVER   Spark Version:5.2.x-scala2.11

Spark Jobs (?)

User: root
Total Uptime:
Scheduling Mode: FAIR
Completed Jobs: 3

Completed Jobs (3)

Job Id (Job Group)	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2 (5720791291321830142_6065098289036004821_46162043f7bd498b977a7a3428587d3b)	#DW #preparazione del dato per scriverlo su tab... collectResult at OutputAggregator.scala:132	2019/04/15 12:12:31	0.4 s	1/1	1/1
1 (5720791291321830142_6065098289036004821_46162043f7bd498b977a7a3428587d3b)	#DW #preparazione del dato per scriverlo su tab... load at NativeMethodAccessorImpl.java:0	2019/04/15 12:12:27	2 s	1/1	1/1
0 (5720791291321830142_7110085198012943495_03382c1d525549009b382c0d782101c1)	#DW #prendo il config flow_config = AxpoHelper.... first at <command-1483815957146432>:49	2019/04/15 12:04:59	2 s	1/1	1/1

Figure 3.4: Databricks SparkUI interface.

There is also a shared folder in which all users have by default all permissions. This folder is useful as a quick way of sharing code between users, especially if they have different permissions, or if they want to share only specific files.

3.3.1.4 Execution

All scripts create a job identified by an UUID[1]. These jobs are executed on an Apache Spark cluster.

All standard operations of a cluster are available, such as viewing running, queued and completed jobs, inspecting job stages and inspecting and changing cluster configuration.

All of these options can be selected through a standard SparkUI interface, such as the one shown in Figure 3.4.

Cluster specifications The cluster is composed of two nodes: one is both master and worker, the other serves only as worker. Each node has 14 GB RAM available and 4 cores.

One of the most important aspects of Databricks is called *autoscaling*. If the cluster is currently executing a computationally demanding task, Databricks may choose to add some workers to increase performance. It is possible to set both the minimum and the maximum amount of workers.

Autoscaling is useful when facing variable loads, since it allows to save up some costs by shutting off some workers that are not needed, whereas in a statically-sized cluster all workers would be constantly active.

3.3.1.5 Importing/Exporting scripts

Databricks allows exporting and importing scripts by directly interacting with the user interface, from which it is possible to export a whole folder in `zip` format. Similarly, it

[1]Universally Unique Identifier

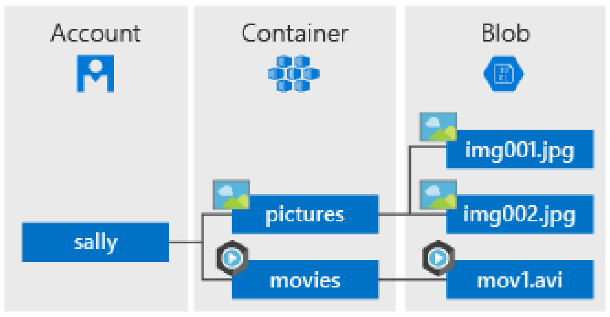


Figure 3.5: Relationship between DataLake resources.

is also possible to import a zipped folder into a different environment. Alternatively, a single file can also be directly exported in their native format.

3.3.2 DataLake

Azure DataLake Gen2 is a cloud storage service. It is possible to store and retrieve files of any format and up to 8TB size.

3.3.2.1 Blob storage

Files are stored under a Blob storage format, which is an Azure-developed format optimized for storing massive amounts of unstructured data [11].

Blob storage offers three types of resources:

- The storage account.
- A container in the storage account.
- A blob in a container.

Figure 3.5 shows the relationship between these resources. Files are memorized in a blob, i.e, a particular distributed storage format which can contain pieces of multiple files.

Permissions can be set at container level or at file level. Under each container it is possible to upload or download files, or to organize them into sub-folders.

Data can be accessed through HTTP or HTTPS. It is also possible to mount the Blob storage as a file system through the `abfss`[2] protocol.

Blob types Azure Storage supports three types of blobs [11]:

- Block blobs store text and binary data, up to about 4.7 TB. Block blobs are made up of blocks of data that can be managed individually.
- Append blobs are made up of blocks like block blobs, but are optimized for append operations. Append blobs are ideal for scenarios such as logging data from virtual machines.

[2] *Azure Blob File System*. The last `s` means the protocol is *Secure*, i.e., it requires SSL authentication.

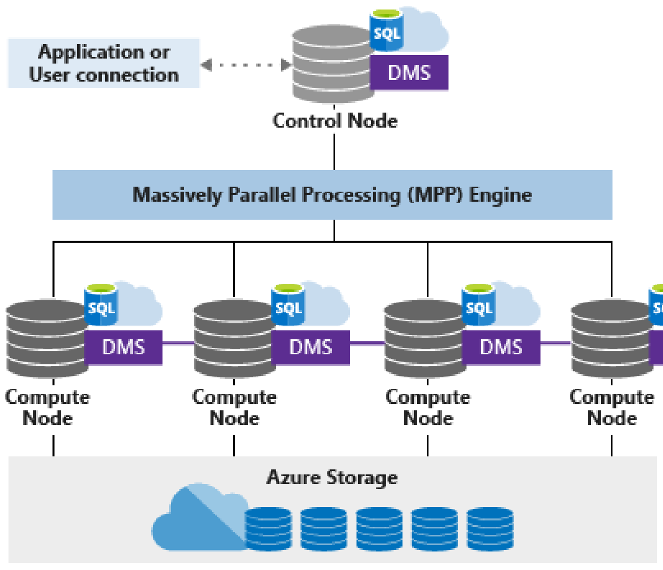


Figure 3.6: Azure SQL Data Warehouse MPP architecture.

- Page blobs store random access files up to 8 TB in size. Page blobs that store the virtual hard drive files serve as disks for Azure virtual machines.

All files used by Databricks, given their relatively small size (a few MB per file) are stored as block blobs.

3.3.3 SQL Data Warehouse

Azure SQL Data Warehouse is a cloud-based data warehouse that uses Massive Parallel Processing (MPP).

MPP allows a high number of processors, even located on different machines, to work on different parts of the same task simultaneously.

Thanks to MPP it is possible to run complex queries across petabytes of data.

3.3.3.1 Data storage

The data warehouse data is stored separately on Azure Storage.

The data itself is sharded into several **distributions** across different nodes, to improve performance.

3.3.3.2 Massive Parallel Processing

Massive Parallel Processing is managed in Azure SQL Data Warehouse by a node-based architecture [12], as shown in Figure 3.6.

Queries are sent to the Control node, which optimizes the work for parallel processing, before sending it to Compute nodes, which execute it.

Control node The control node can be defined as the brain of the data warehouse. It receives commands and runs a MPP engine, which optimizes and coordinates the work.

Queries are split into 60 smaller queries[3] which can all be run in parallel.

The smaller queries are then assigned to a number of Compute nodes, which execute them.

Each smaller query is executed on a single data distribution.

Compute nodes A compute node is a working unit which executes queries received by the Control node.

An Azure SQL Data Warehouse can have up to 60 compute nodes. Depending on how many nodes are available to the system[4], the Control node assigns them from 1 to 60 distributions.

For example, if there are 60 nodes available each one will have to compute a single task, whereas if only a single node is available it will have to execute all 60 tasks.

3.3.3.3 Limitations

MPP allows a low number of parallel queries, which are a fundamental requirement, given that several users will be working the data warehouse at the same time.

Azure SQL data warehouse offers different capacity plan, with different costs and computational power. The plan currently chosen by Reply allows up to 12 queries in parallel, which, although is enough for development, it is much too low for the actual needs of the trading and big data departments combined.

The highest plan tier offers up to 128 queries in parallel at more than ten times the actual cost [13]. Even ignoring the economic effects of this choice, the number of queries offered is still not high enough, which means that the data warehouse might not be able to answer all queries in acceptable times.

The solution to this problem is to create a replica of a portion of the data warehouse and to perform the most computationally demanding queries there. These replicas will reside on Azure Analysis Services.

3.3.4 PolyBase

PolyBase is a tool which allows the SQL Data Warehouse to process SQL queries on data from external sources [14].

The most common external source is Azure Blob Storage. PolyBase is able to create a connection layer between the Data Warehouse and Blob files. These files appear to the user as normal external tables. These files can then be directly queried through SQL.

PolyBase is natively available on Microsoft Azure, so there's no need to perform a manual installation. The tool is also often transparent to the users, so most times it is actually

[3] This value appears to be fixed and independent from which the query executed [12].

[4] Their number depends on the chosen plan tier. A higher number of compute nodes comports an higher subscription cost.

used without even realizing it.

PolyBase relies on Hadoop to perform cross-source SQL operations.

Advantages Without PolyBase, querying a SQL Data Warehouse along with an external source would require either transferring half of the data available, bringing all data in a single format, or performing two separate queries and writing custom client-side logic to join the data obtained.

With PolyBase this problem is removed, since users can directly query both sources at once using SQL, without having to transfer anything beforehand.

Performance PolyBase can push computation to Hadoop to improve performance. The decision is handled by PolyBase's own query optimizer, which uses statistics on external tables to make the cost-based decision. Computations on Hadoop create MapReduce jobs which leverage distributed computational resources.

3.3.5 Other resources

3.3.5.1 Data Factory

Azure Data Factory can be defines as a cloud-based data integration service that allows users to create data-driven workflows in the cloud for orchestrating and automating data movement and data transformation [15].

Users can create workflows that ingest data from different sources. These workflows can apply operation of the data using different cloud services, such as Hadoop or Spark. The results can be published to cloud data stores, such as Azure SQL Data Warehouse or Azure Data Lake.

Advantages and Limitations This service is very effective for performing basic operations, such as reading from a data source and writing on a different one. Basic transformation operations can also be applied.

These simple workflows can be developed in just a few minutes using the graphical interface.

However, this tool isn't suited for more complex operations, since it isn't possible to write code directly, but it is necessary to use the user interface. In these cases, coding in Databricks proves to be easier and more versatile.

3.3.5.2 Virtual Machine

A virtual machine can be run on the Azure cloud services.

The machines can be customized depending on the user needs, with options ranging from the operating system to vCPUs[5] and RAM available.

[5] *Virtual CPU.* A vCPU is a share of a physical CPU that is assigned to a virtual machine. It is possible to assign multiple *vCPUs* to a single Azure Virtual Machine.

The chosen virtual machine runs Windows Server, with a graphical environment installed.

Users can install their own applications on the machine, running code in languages or environments not supported by Azure Databricks.

3.3.5.3 Analysis Services

Azure Analysis Services allows users to combine and analyze data from multiple sources.

Users can apply operations or filters on data sources, as well as display the results in multiple graphical formats, such as maps, charts or reports.

This service has not been used during development, but will be used once the entire migration process will be completed to perform Business Intelligence analysis on the data contained in the data warehouse.

3.3.5.4 Automation Account

Azure Automation Account allows user to schedule management operations for the whole environment.

Users can create *Runbooks*, which are a set of scheduled commands, and execute them.

Runbooks can be created either through a graphical interface or by using Python or Powershell.

A single runbook can contain multiple jobs, each one performing a single task. Each job can be scheduled to be executed at a specific time.

Part II

Proposed Solution

Overview

Considering all the problems explained previously, Reply proposed a solution for developing the Data Warehouse, as shown in figure 3.7. Their proposal also included a software architecture for performing analyses on the Data Warehouse.

1. Several scripts, executed by Databricks, download the data using different techniques, such as API calling, FTP downloading or web scraping.
2. This data is then stored on DataLake, under the `rawdata` folder.
3. Those files are recovered by Databricks, which applies some transformations on the data, for example normalization or column removal.
4. The result is saved again on DataLake, in *csv* format, under the `srcdata` folder.
5. The values from the *csv* files are loaded into the data warehouse. The communication between the blob storage and the data warehouse is handled by Polybase.
6. Some values stored in the data warehouse are accessed by Databricks and used as configuration settings.
7. The data warehouse is queried by reporting tools, such as Excel or PowerBI, as well as algorithms developed by Axpo.
8. In case of heavy workloads, it is also possible to query a copy of the data warehouse, located on Azure Analytics Services.

I analyzed their proposal, paying particular attention to the ETL process and Data Warehouse structure, as well as the workflow chosen for developing such structures.

In what follows, it will be discussed:

- The workflow used by Reply.
- How the ETL process works.
- The structure of the Data Warehouse and the reasons for that choice.

Problems and difficulties identified during each part of the process will also be discussed. Several examples will also be provided.

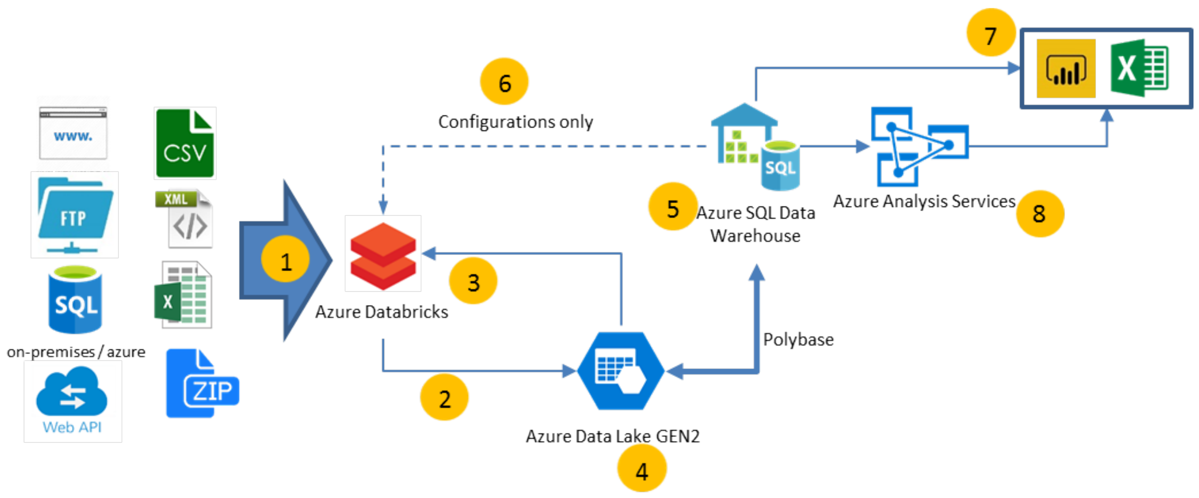


Figure 3.7: Interaction between Azure components.

Chapter 4

Migration Process

In this chapter we will analyze how the migration has been organized.

Several aspects will be seen, ranging from how the whole project has been split into smaller activities, called *Sprints*, to the workflow used by Reply to develop each Sprint.

4.1 Sprints

Reply proposed to organize its migration work into Sprints.

A **sprint** can be defined as a set of activities needed to migrate an existing process. A sprint is comprised of several data streams from multiple providers.

A **provider** is a data source, usually external, from which data is retrieved and stored on the Axpo Data Warehouse.

A **data stream** is a specific kind of data exposed by a given provider.

4.1.1 Introduction

The data warehouse needs to contain data originating from a large amount of providers. Each provider exposes information related to a specific aspect (for example, stock prices or gas consumption).

The amount of data needed is very large, reaching up to a few thousand different data streams across all providers. Considering the size of the project, it was necessary to organize the ETL development of all data streams. This organization was driven by two factors.

First of all, it was necessary to migrate multiple existing processes, prioritizing specific ones. Some processes played a critical role in the company and having them on a more robust and efficient structure would have provided a greater benefit.

Secondly, each tool requires a different set of data streams, with an almost non-existent overlap. As such, each process could be migrated independently.

Example A specific tool extracted data from existing on-premise databases to produce market prices reports. Since the databases were going to be migrated to the cloud, it was necessary to create a cloud-based ETL process for all the data used by these tool.

The development of a new ETL process was necessary to keep the Data Warehouse populated with the most recent data. Having recent data was a critical requisite, since up-to-date information are needed on a daily basis by multiple Axpo employees.

As such, we planned a Sprint with the goal of enabling the tool to fetch all of its data from the Data Warehouse, instead of having to query multiple on-premise databases.

For this sprint it was necessary to download a total of 372 data streams from 9 different providers.

Each provider exposes multiple data streams. For example, GME[1] provides data about different market sessions, ranging from MGP to all the MI sessions. Moreover, different kinds of information are provided for each session, such as prices, quantities, and transit limits.

Each file has been considered a separate data stream.

4.1.2 Balancing

Sprints have been designed to be balanced, i.e., each sprint requires roughly the same amount of effort.

It is necessary to take into account different factors, in order to estimate the effort required for a sprint, as well as the expected sprint deadline.

This balancing activity depends on the following aspects:

- Provider source
- Technique required to download data
- Intrinsic provider complexity

These factors are fundamental from an organizational point of view, since they allow Axpo employees to coordinate their work with the Data Warehouse development process.

For example, some employees have processes currently running on local machine, which they want to transfer to the cloud.

In order to transfer they processes they need, however, to retrieve data from the Data Warehouse. As a consequence, they need to wait for the ETL process to be fully developed before they can start migrating their processes onto the Cloud.

4.1.2.1 Provider source

An important aspect to consider when estimating the complexity of a provider is whether it is an internal or external source.

[1] GME is the manager of the Italian Spot Electricity Market. For more information, see appendix A.

	Internal	External
Already known	Yes, in depth	Sometimes, not in depth
Can be modified	Yes	No
Direct access to DB	Yes	No
Possible data inconsistency	Yes, but known	Yes, unknown

Table 4.1: Difference between internal and external data sources.

Internal Internal sources are directly owned, and often also developed, by Axpo. They can be either databases or tools.

Internal sources are the easiest to extract data from, for a variety of reasons.

First of all, they are already well-known, since they have been developed or chosen (in case of a third-party tool) by someone working at Axpo.

Secondly, they can be modified to accommodate the Data Warehouse needs. For example, if the Data Warehouse needs to extract data in a specific format, it is possible to modify the tool to expose it in that given format. As such, the Data Warehouse extraction process can be simplified.

Another important aspect is that most of these tools rely on its own database, which can also be directly queried to extract information directly, thus further simplifying the process.

External Most providers can be defined as external sources, that is websites or services not owned by Axpo.

Some of these services are public, while others provide data under a paid subscription.

These websites present several difficulties, compared to internal sources.

First of all, they are less known that internal sources, since no-one at Axpo directly worked on them. As a consequence, it is necessary to spend more time studying the tool. There is also a consistent possibility of unexpected issues coming up during development[2].

Secondly, there is a lack of standardization between each source: each company provides data in its own format and way, making the extraction process more complex.

Moreover, there could also be some problems with the data provided. For example, some providers use different date conventions depending of the type of data required. These different formats need to be normalized into a common notation before inserting the data into the Data Warehouse.

Table 4.1 sums up the main differences between internal and external data sources.

[2] For example, a provider exposes data pertaining to the current year in `csv` format, while data related to previous years are stored in `gz` format. There is however no documentation about this difference and the problem can only be noticed by analyzing when a custom downloader stops working.
For a more in-depth description of the issues encountered, see section 5.2.

4.1.2.2 Download technique

Another important aspect to consider when estimating the complexity of a provider is the technique required for extracting data from it.

A more complex extraction process will take more time and resources, and needs to be accounted for in advance.

Also, some developers are more experienced than others on a particular extraction technique. To maximise efficiency, it is best to have each developer working on the technique they are more skilled in.

This approach requires however having in a single sprint several different extraction techniques. Otherwise, some developers would be forced to work with technologies they are less familiar with, increasing the effort required for implementing the process.

Database Databases are the fastest and easiest sources to extract data from.

Data are already in a structured format, since they are stored in a relational table. This structure can also be easily altered by writing a query, for example for aggregating or de-aggregating some data.

The only difficulty is understanding the structure of the tables. However, since all databases are internal sources, this problem can easily be simplified by asking directly the Axpo employees who work on that database.

FTP Extracting data from FTP shares has a small degree of complexity.

In most cases, the FTP shares store data in structured formats, such as `csv` or `xml`.

To download data it is necessary to connect to these share points and retrieve the files required.

These files are then parsed and normalized depending on the data structure used by the provider.

In some cases, these files require some special processing for the first lines, which are used as headers.

API Some websites offer an API service which can be called to extract data.

Depending on the amount of documentation available, this process can either be considered as difficult as extracting data from an FTP share or can prove some serious challenges.

In some cases, the required queries required are complex and require an extensive analysis of all parameter combinations, which can often only be carried out empirically.

Web scraping The most difficult and time consuming ETL approach is web scraping, which simulates human web site navigation.e

Given its complexity, web scraping is chosen only as a last resort, when a website does not expose any service for directly extracting the data.

Web scrapers not only require a large amount of time to be developed, but are also very susceptible to website changes: a single element moved a different page can break the download process for the whole provider.

4.1.2.3 Provider complexity

An important factor to consider is also the intrinsic complexity of a given provider.

Some providers consistently provide data in formats which are hard to parse or to extract.

In some cases, some websites are difficult to navigate, requiring navigation of multiple pages just to find the information required, while others may require some functionalities (such as scrolling a page with a web crawler) which are hard to implement.

In other cases, the data extracted may be in a format which is difficult to parse or requires complex transformation operations for obtaining a format suited for the Data Warehouse.

These considerations, which apply at provider-level, are used to estimate a coefficient, which is applied to all the data streams of a given provider. This coefficient helps in correcting the time estimation for a given sprint.

4.2 Process

This chapter will provide a description of how Reply organized the development of each sprint.

The initial organization presented a few problems, for which both I and the Project Manager requested some changes.

Both these problems and the changes will be analyzed as well as the limitations of the chosen workflow.

4.2.1 Phases

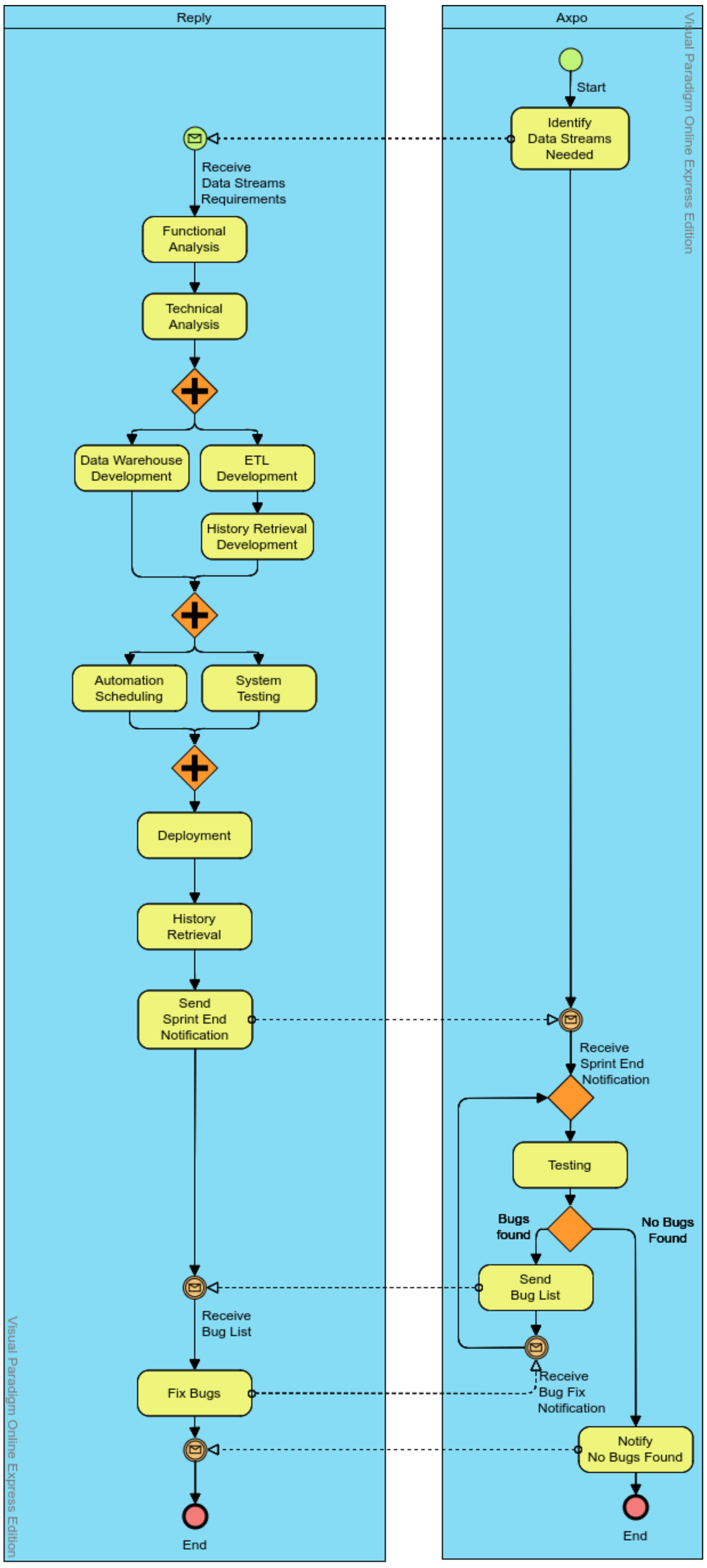
The development of a sprint is composed by multiple phases, as shown in figure 4.1.

Functional analysis The goal of functional analysis is to get a high-level understanding of the data needed for the sprint.

This role requires to analyze each provider, learning which data streams to download, where to download them from and how often the data needs to be downloaded.

The result of this work is an Excel file, containing information such as:

- A detailed list of each data stream required for each provider.
- The update frequency of each data stream.
- A download schedule for each data stream.
- A download URL per data stream.



Reply
Axpo
Start
Identify Data Streams Needed
Receive Data Streams Requirements
Functional Analysis
Technical Analysis
Data Warehouse Development
ETL Development
History Retrieval Development
Automation Scheduling
System Testing
Deployment
History Retrieval
Send Sprint End Notification
Receive Sprint End Notification
Testing
Bugs found
No Bugs Found
Send Bug List
Receive Bug List
Receive Bug Fix Notification
Fix Bugs
Notify No Bugs Found
End
End
Visual Paradigm Online Express Edition
Visual Paradigm Online Express Edition

Figure 4.1: Workflow used by Reply.

Technical analysis The next step is to take these high-level information and provide some very detailed instructions on how to download the data.

These information are very technical in nature, such as which method (for example, an *HTTP GET* call) or parameters need to be used to download a specific datum, as well as which operations must be applied to transforming the downloaded data properly.

These information are then passed on to the developers, who write the actual implementation.

ETL development Developers receive the low-level specifications written during the technical analysis and implement the actual ETL process.

This process, implemented on Azure Databricks, needs to retrieve data from multiple providers using different techniques. Several cleaning and normalization operations are then applied on the data downloaded. Finally, the results are loaded into the Data Warehouse.

History retrieval development The ETL procedure downloads data for a single day[3] but, during the Data Warehouse initialization, it is necessary to download data up to a few years before.

This process creates a wrapper for the ETL procedure, calling it several times, each time downloading data for a different day. The procedure is iterative, meaning that a single day will be extracted, transformed and loaded before processing the next day.

Days are downloaded backwards. This allows to handle errors, such as file structure changes, more efficiently, since they will not affect any file downloaded up to that point.

Data Warehouse design The ETL process needs to load the transformed data on the Data Warehouse.

Its design is done in parallel, by a different team, with the ETL process. The design process is responsible for the following actions:

- Creating some tables onto which the ETL process will load the data *as-is*.
- Creating a procedure for performing some operations on the data, such as remapping or adding constants.
- Loading the data onto the actual end-user tables, as well as checking if the data actually needs to be loaded.
  - Some values may already be present if the ETL process for a given time range is executed multiple times, for example due to errors.

[3] To be more precise, the process downloads data for a single time unit, i.e,. the scheduled download interval for a given stream. Most streams are downloaded daily, but some have a different download frequency (varying from hourly to two or three times a day).

Automation scheduling Each website provides new data at regular intervals. It is thus necessary to execute the ETL process each time the provider has new data.

During the functional analysis, each provider has been marked with its update frequency. This process schedules, for each flow, when the ETL process will be executed, ranging from once a month to multiple times a day.

Since it is necessary to download multiple data streams from each provider, Reply decided to schedule a single job per provider which executes all jobs required for that provider.

A secondary function of the scheduling process is to pause the whole Data Warehouse at night, since it is not going to be used, to reduce cloud maintenance costs.

System testing Functional and technical analysts must also make sure that the whole ETL process is working as intended and that the data downloaded into the Data Warehouse is identical to that present in the original websites.

These tests are done by both analysts together by hand. They manually check some data samples from the data warehouse and compare them with the same values in the websites.

This testing phase is focused mainly on the following aspects:

Number of rows Testers manually assert that the number of imported values in the Data Warehouse (i.e. table rows) is the same than those available on the website.

As such, they can be sure the ETL process didn't lose any values.

Conversion errors Testers check columns on which conversion operations have been applied, to make sure the process completed successfully.

These checks for example assert that decimal numbers have been correctly interpreted, dates have been properly remapped and that each column contains the right kind of data.

Sample checking Lastly, testers take same sample rows and compare all their values with the data available on the website.

If all the values are identical, then it's likely the ETL process is working properly and that all the other values will be correct too.

Deployment to production environment The last step is to deploy the whole structure to the production environment.

The deployment is done by exporting all the scripts used (for both the ETL process and the Data Warehouse configuration) and re-importing them into the new environment. All scripts are then executed, in order to:

1. Initialize (or modify, if they already exist) all tables and procedures in the Data Warehouse.

2. Download the first set of data (typically up to a few years before).

This phase is the most delicate, since this environment is used for production, so it is imperative that the deployment must not cause any errors.

Particular attention must also be paid to the update of existing structures. For example, if a table structure has to be changed, it is not possible to simply drop and recreate it, since all data would be lost. It is then necessary to compute the difference between the two versions, and to update as needed.

History retrieval After deployment, the Data Warehouse is empty and needs to be initialized.

Initialization is done by invoking the apposite procedures previously developed.

The process takes a long time to execute, usually a few hours per provider, but only needs to be executed once.

User Acceptance Test The system is finally tested by the various Axpo departments users.

If they notice any problem, they notify Reply, which develops and applies the appropriate fixes.

Some are not specific or predefined, but rather mainly consist of the users trying to use the Data Warehouse for their actual needs, such as running their algorithms.

Other tests are more systematic, such as the ones described in chapter 7.

4.2.2 Issues

Several issues with the workflow have been found during development. This section will list them, along with the chosen solutions.

4.2.2.1 History Retrieval

Initially, the history retrieval process suffered from many unexpected issues.

This phase was originally planned after the deployment process. However, during the first sprint, the retrieval process for many providers broke.

Upon further analysis we discovered that several providers changed data format over the course of the years. As such, it was necessary to modify the history retrieval process to handle these special cases.

For example, a specific provider exposes `csv` files for the current year and `gz` archives for the previous years. This difference had been overlooked during development since the developers only focused on recent files.

After deployment, when the process had been scheduled to retrieve data for multiple years, the downloader crashed since it was expecting a `csv` file but retrieved an archive. Developers had to manually analyze the error log, as well as the provider website to understand the problem and implement a solution.

Some other providers maintained instead the `csv` format, but changed their internal structure, for example adding or removing columns which were expected by the downloader.

Optimization As an additional safety measure, Reply decided to further modify the history retrieval process, making it download each day independently.

Originally, the process downloaded all days required from the provider, before storing them in bulk onto the Data Warehouse. In case of errors all data downloaded would be lost.

Since the errors related to different formats for old data have no impact on more recent data, it is reasonable to store each day independently onto the Data Warehouse. In this way, in case of errors, all the other information would still be available onto the Data Warehouse.

This optimization also prevented the downloader from having to retrieve the same data multiple times in case of failure.

Workflow change This additional work caused some delays in the original planning, since they hadn't been originally taken into account.

In order to take these delays into account, Reply decided to execute a preliminary history retrieval process during development, testing the process not on just a few days but on several years. In this way, this kind of errors would be noticed before deployment, ensuring the deadlines would be respected.

4.2.2.2 Initial delays

The development of the first sprint took far more time than originally planned.

This was caused by a misestimation of the ETL development complexity. This error was mainly caused by two factors.

Technology The team was not yet familiar developing on *Microsoft Azure Cloud Computing Platform & Services.*

This aspect had been underestimated by Reply, which led to a too much optimistic deadline. Much more time than expected had been spent trying to solve problems related to Azure tools, which led to delays in the original planning.

Providers Another aspect which had been originally underestimated was the high complexity of some providers.

Some providers required very complex methods to extract data from them. These methods required more time to develop than originally estimated, which led to more delays.

Since we requested Reply to respect the original deadlines even considering their delays, they had to add some team members to catch up, as well as grouping together some smaller sprints, to reduce the overhead.

4.2.2.3 Automated tests

Reply decided not to create automated tests but to rely on user tests instead.

They stated that in the energy market scenario, users tests are more appropriate than automated tests for a variety of reasons.

First of all, these data are very domain-specific, so writing a proper testing suite would require extensive domain knowledge.
Secondly, most values come from external providers, so it is not possible to perform many actions if these values are detected as incorrect.
Lastly, there is a notification system in place in case of critical errors (i.e. processes breaking and failing to complete). Reply has considered this notification system enough for this project.

Further tests, such as the ones described in section 7, have clearly shown that relying only on notifications and System Tests is not enough to assure the quality of the Data Warehouse. As a consequence, Reply has been requested to develop these tests during development, to shorten the delay between the deployment and the actual beginning of the data quality tests.

4.2.2.4 Data correctness definition

An interesting aspect is the impossibility to determine if the data received are correct or not.

Most data received are measurement taken by either operators or automated tools. In some other cases, data are the result of forecasting algorithms, developed by the provider themselves.

Since there is no way to assert the correctness of the data received by the providers, the values shown on their websites are assumed to be correct.

The only test that can be done is asserting that the information stored in the Data Warehouse, at the end of the download process, is identical to the information present on the websites.

4.2.2.5 Priority vs non-priority data streams

All data streams have been categorized as priority or non-priority.

Priority streams contain data needed by tools currently in use by the company. Having this data available is fundamental for running these tools on the cloud.

On the other hand, some data has been marked as non-priority. These information are not currently used by any tool, but may be used in the future or for exploratory analysis.

Given the low importance of having these data available in a short time, the only operation performed on them has been functional analysis, to identify if there are any issues with the data. A development date has, however, not been fixed: the ETL processes for these data stream will be developed during spare time or after all other priority data streams have been completed.

The downside of this categorization is the lack of a precise time estimation for all the data streams. This is likely not to cause a consistent delay for a few data streams, but could become a problem for a high number of streams.

Moreover, in several occasions Axpo employees had erroneously marked some data streams as non-priority, causing some confusion and forcing Reply to re-plan their work schedule. Changing these requirements naturally comports some delays, since we are requesting more work to be done before a deployment.

4.2.2.6 Lack of formal documentation

A problem present in all development phases is the lack of a clear and formal documentation. Most of the information are in the heads of the developers, meaning that other people have either to ask them or to look directly at the code. The documentation present is represented in a non-clear way, usually using Excel sheets to annotate different types of information and comments.

A more formal notation has been used, on the other hand, to represent the Data Warehouse structure. This notation, shown in Figures 4.2 and 4.3, describes, however, only the columns present in each table, without specifying any relationship between them. Moreover, there is no clear distinction between fact tables (all the large boxes shown at the center of the Figure) and dimension tables (some of the small ones on the edges).

An additional issue is that only the column names are reported, as shown in Figure 4.3. Other relevant information, such as the data type for each column, are not shown.

This design can, as a consequence, lead to confusion.

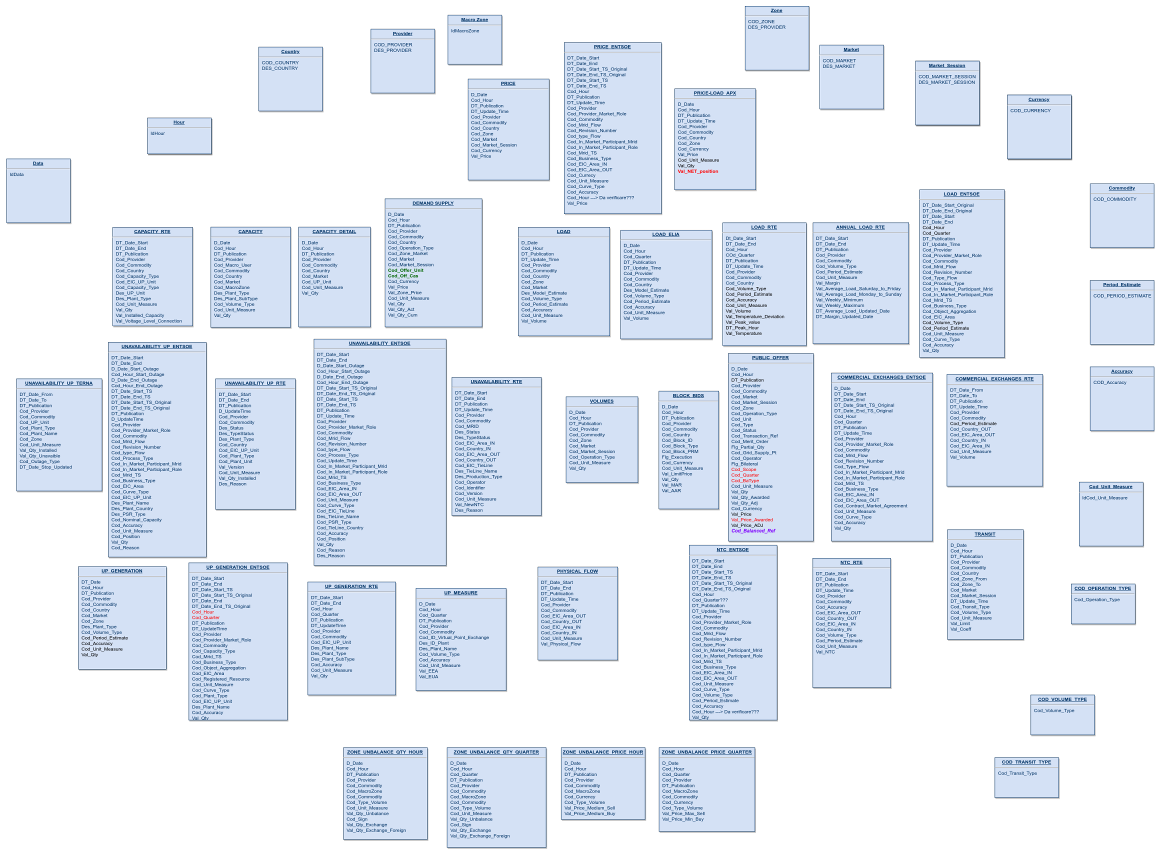


Figure 4.2: Diagram used for representing Data Warehouse structure.

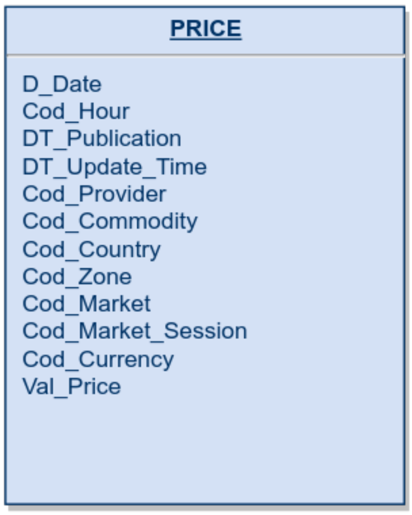


Figure 4.3: A single table from the Data Warehouse structure diagram.

Chapter 5

ETL Process

In this chapter we will describe the process used for extracting, transforming and loading data into the Data Warehouse.

I analyzed the multiple problems encountered during the ETL development as well as the process employed by Reply. Most problems were however solved by Axpo specialists, since their specific knowledge was required.

I also asked Reply to keep track of all the issues encountered as well as their status.

This can be used both by Reply, to better organize its work, and by Axpo, to understand if a delay is due to an unexpected issue and how well Reply is performing.

The resulting document can be used both as a small layer of documentation and to produce some tests for the more vulnerable parts of the ETL processes.

5.1 ETL Operations

This section will provide a description of the main operations performed by the ETL process. An exhaustive list is not provided, since describing each operation of them would be extremely long and repetitive.

5.1.1 Extraction

Most processes are executed daily, so the downloaders usually retrieve files only for the current day. However the downloaders can also be easily configured to retrieve a range of dates, in case of history retrieval (which is only used to initialize the Data Warehouse).

Each downloaded file is saved to the `rawdata` folder on DataLake *as-is*.

5.1.1.1 Provider types

Different download techniques are used, depending on the kind of services exposed by a particular provider.

FTP Several providers provide an FTP share, where they upload a few files each day, which contain the data for that given day.

In this case the downloader just needs to connect to the FTP server and retrieve the files.

Files stored here are usually either in `csv`, `xml` or `xls` format.

API Other websites provide an API which can be queried for data.

The resulting datasets are usually in `json` or `xml` format.

In this case the downloader makes several calls to the API, each one with different parameters.

SQL databases These providers provide the easiest interface, since it is possible to directly perform queries and extract the data in nearly any format.

Since most of these providers are internal services owned by Axpo, their number is limited, compared to all the public providers.

Web scraping Some websites do not expose their data through any particular service.

In this case the only solution is to build a web scraper, which simulates human web navigation retrieving information manually.

This approach is the most complex and time consuming, since it has to be build *ad hoc* for the website. Moreover, its re-usability across different websites is low, if non-existent, so it needs to be developed each time from scratch.

5.1.2 Transformation

Databricks retrieves the files saved on `rawdata` and performs some operations on them, before saving the resulting datasets in `.csv` format on the `srcdata` folder in DataLake.

The most commonly executed operations are:

- Reading the dataset: depending on its format different methods are needed
- Renaming columns
- Dropping unneeded columns
- Appending the file download date to each row, where not already present
- Unpivoting the dataset, if needed
- Normalizing column names

Unpivoting A consistent number of datasets provide a value for each hour in a day in an apposite column.

It is necessary to change the format, displaying on one column the hour and in another one the value. This operation is called *unpivoting* or *melting*.

Zone	H1	H2	H3	...	H25
North	12	15	14	...	0
South	32	29	28	...	0

Zone	Hour	Value
North	H1	12
North	H2	15
North	H3	14
North	...	...
North	H25	0
South	H1	32
South	H2	29
South	H3	28
South	...	...
South	H25	0

Table 5.1: Normal (left) and unpivoted (right) tables.

Table 5.1 provides an example of unpivoting. In the left table we have 26 columns, one representing the zone and a column for each hour[1], while in the right table we have 3 columns: zone, hour and value.

Column names normalization Some columns do not respect the naming convention decided for the Data Warehouse.

Some columns may contain special characters or spaces, which cause problems when inserted into a database table.

Several operations are performed to remove or replace these characters.

For example, spaces and dashes are replaced with an underscore, parentheses of any type are removed and the character `&` is replaced with its literal representation `and`.
Also, all names are converted into lowercase.

5.1.3 Loading

Databricks retrieves the files from `srcdata` and loads the data onto the Data Warehouse.

The Data Warehouse, through several tables, views and procedures, performs some final operations on the data, before appending the new values onto the final tables.

5.1.3.1 Data Warehouse procedures

The Data Warehouse contains three schemas:

- `src` *(source)*
- `stg` *(staging)*
- `dm` *(data mart)*

[1]There are 25 hours to account for DST.

***Source* schema** Databricks loads the data directly into the `src` schema tables, without performing any operations in it.

This schema contains a table for each flow, with the same structure as the `csv` file. All values loaded in these tables are initially represented as strings but they are then casted to their actual data type through apposite views.

These views are still located in the same schema, and there is a view for each table.

Each time new data are inserted into those tables, the old values are erased.

***Staging* schema** Through some stored procedures, the views present in the `src` schema are read and some additional operations are applied on the data.

The result is then saved on some apposite tables stored under the `stg` schema.

As with the `src` schema, when inserting into these tables, the previously existing data are erased.

The most significative differences between the two schemas are the structure and the amount of tables.

In the `src` schema, there are as many tables as flows and their structure is identical to the `csv` files stored on DataLake.

The tables on the `stg` schema are, on the other hand, much less and resemble the structure of the final tables. Moreover, flows from different providers are stored together in these tables.

The operations performed by the stored procedures are in charge of changing the data structure, from the one used in the `csv` to the one needed by the final tables, and to add some constants: for example, if we have a `src` table containing information about *MSD*, its provider will surely be *GME*[2].

***Data Mart* schema** This schema contains the final tables, which will be queried by the end-users.

Through apposite stored procedures, the content of the `stg` tables are appended into the `dm` tables.

These procedures pay special attention to checking if the values to be inserted are already present in the Data Warehouse and, if they are, they do not perform any action.

This has been done to prevent duplicate data, which may be created by running more than once the ETL process (which may happen in case of errors).

5.1.3.2 Logging

All loading operations are traced to a log table, which is analyzed in case of alerts to understand when and how a process failed.

Logging Logging is performed by invoking several stored procedures on the Data Warehouse, which write information to apposite tables. These information contain:

[2] *MSD* information are only provided by *GME*. For more details, see appendix A.

- The id of the process
- The provider name
- The data stream name for that given provider
- Two timestamps, for insertion and update respectively
- Some flags to indicate the operations already performed on the data

Logs are written during each phase of the ETL process for each flow.

Alerts Alerts are generated by a monitoring system set in Databricks. Jobs can be set to send mails when a jobs starts, ends or fails.

Upon job failure, Databricks sends a mail to specific users. The mail contains a link to the cell which raised an error, along with the id of the process which failed.

It is possible both to analyze the code and to query the database logging tables (using the id contained in the mail) to retrieve additional information.

Monitoring tools Currently there are no tools actively monitoring the quality of the data present in the Data Warehouse apart from the alert system.

This is done under the assumption that if the data are inserted into the Data Warehouse they are already of acceptable quality. However, if some errors happen to elude the monitoring system, there is not currently any way to notice the problem.

5.2 Issues

During development of the ETL process, there have been many difficulties. This section will list the most relevant problems encountered.

5.2.1 File Encoding

This problem is related mainly to `csv` files. Most files are encoded in `ANSI` format, but a few providers used special characters which required a different encoding.

For example, some energy plants or technical description contain accented characters, which are not supported by the `ANSI` format.

Spotting this kind of problems isn't easy, since not all files may present special characters.

The solution, on the other hand is easy, since it is sufficient to use `utf-8` encoding.

5.2.2 Wrong File Extension

We also encountered an issue related to a wrong file extension. Terna exposes its data in Excel format, but the websites presents a low level of consistency in its files.

While most files were in `xls` format, a few were in `xlsx` format, which requires a different method for reading it. However, upon further analysis, the files proved to be regular `xls` with a wrong file extension. As a result, extensions are ignored when reading files.

5.2.3 Aggregation/Deaggregation

Some providers offer data with granularity different from the needs of Axpo users.

In this case it is necessary to either aggregate or deaggregate the data downloaded, before inserting it into the Data Warehouse.

These processes present their own difficulties however.

Aggregation The difficulty in aggregating is understanding which operation is more appropriate for each kind of data.

Let's assume we want to aggregate into hourly data information provided every 15 minutes.

Energy consumptions[3], for example, need to be summed, while temperature forecasts need to be averaged, since it would make no sense to sum them.

Deaggregation Splitting aggregated data into a finer granularity presents additional problem to those of aggregating.

For example, let's assume we want to know the hourly energy consumption for a given user. If we have a single value for the whole day, it is not reasonable to divide that value by 24, since it is very unlikely the consumption has been constant during the whole day.

In this case the correct operation is infer a consumption model from other available data and to assume this user will behave similarly. Then the total value will be divided proportionately to the model.

This is however a specific example, in general it is impossible to give a rule of thumb, since each case presents different properties and must be analyzed individually.

Time shifts and time formats Choosing the correct operation is not the only problem however.

Different time formats, such as gas day, and time shifts, caused by DST, make these operations even more complex.

The first 6 hours (*0-6am*) of a "standard" day are part of the previous gas day. However, during DST, these hours become 7 (*0-7am*), since gas days always have 24 hours, while standard days can have 23, 24 or 25 hours[4].

In these cases more complex queries are required to obtain the correct data before applying aggregation or deaggregation operations.

[3] Energy consumption refers to the amount of electricity used in a given time range, expressed in MW.

[4] For more details, see sections 6.2.2.1 (Gas day) and 6.2.2.1 (DST).

Session	Scheduled Start
Daily A	0 30 13,14 ? * * *
Daily B	0 30 8 * * ?
Daily C	0 10 18 ? * * *
Daily D	0 30 8,17 ? * * *
Daily E	0 30 11 1,2,3,4,5 * ? *
Intra-daily 1	0 30 15 ? * * *
Intra-daily 2	0 0 17 ? * * *
Intra-daily 3	0 30 8 ? * * *
...	...

Table 5.2: Jobs scheduled for a specific provider.

5.2.4 Different Update Frequencies

Each provider updates data with its own frequency.

Update frequencies vary greatly, ranging from hourly to monthly updates. There are also some particular scenarios, for example a certain provider updates its data on Monday, either Tuesday or Wednesday (with no apparent discerning factors) and Friday.

This causes issues when scheduling jobs, since it is necessary to schedule each job independently.

Table 5.2 shows an example of jobs scheduled for a specific website providing market data. All the start times are in *crontab* notation[5].

Each one of these jobs executes multiple Databricks notebooks, each one downloading a specific data stream.

5.3 Examples

In this section we will list a few examples to give a more hands-on idea of some of the problems encountered during development.

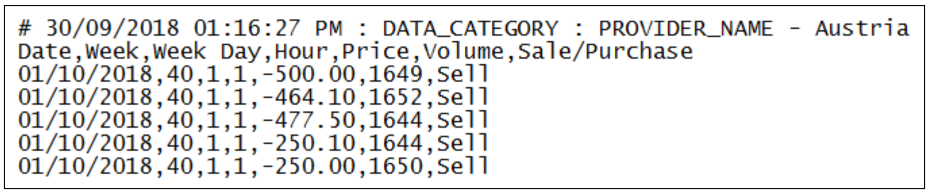
5.3.1 CSV Parsing

Reading from a `csv` file is not always a trivial operation.

Several files contain some heading lines with additional information. When reading these files it is necessary to apply additional operations to extract these information.

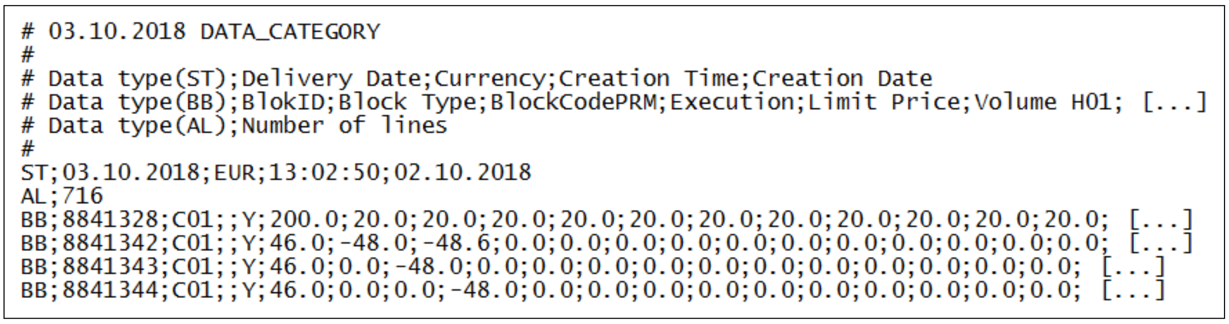
For example, a certain provider exposes market information in different files. Depending on the file, different operations must be applied.

[5] The values indicate respectively seconds, minutes, hours, day of month, month, day of week and year. For more information about *crontab* notation, see [16].



```
# 30/09/2018 01:16:27 PM : DATA_CATEGORY : PROVIDER_NAME - Austria
Date,Week,Week Day,Hour,Price,Volume,Sale/Purchase
01/10/2018,40,1,1,-500.00,1649,Sell
01/10/2018,40,1,1,-464.10,1652,Sell
01/10/2018,40,1,1,-477.50,1644,Sell
01/10/2018,40,1,1,-250.10,1644,Sell
01/10/2018,40,1,1,-250.00,1650,Sell
```

Figure 5.1: Metadata in first rows of a `csv` file.



```
# 03.10.2018 DATA_CATEGORY
#
# Data type(ST);Delivery Date;Currency;Creation Time;Creation Date
# Data type(BB);BlokID;Block Type;BlockCodePRM;Execution;Limit Price;Volume H01; [...]
# Data type(AL);Number of lines
#
ST;03.10.2018;EUR;13:02:50;02.10.2018
AL;716
BB;8841328;C01;;Y;200.0;20.0;20.0;20.0;20.0;20.0;20.0;20.0;20.0;20.0;20.0;20.0; [...]
BB;8841342;C01;;Y;46.0;-48.0;-48.6;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0; [...]
BB;8841343;C01;;Y;46.0;0.0;-48.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0; [...]
BB;8841344;C01;;Y;46.0;0.0;0.0;-48.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0; [...]
```

Figure 5.2: `Csv` with multiple data formats.

First rows metadata As we can see from Figure 5.1, the first row of the `csv` file requires special attention.

From it we can extract information about[6]:

- Creation date *(30/09/2018 01:16:27 PM)*
- Category *(DATA_CATEGORY)*
- Provider *(PROVIDER_NAME)*
- Country *(Austria)*

These information are then appended to each row of the dataset and stored on the Data Warehouse.

Multiple data formats For a different data stream, the first 6 rows of the file define some metadata of the dataset, and need to be treated separately. Figure 5.2 shows the format of the data.

This file contains three different types of data, each with its own format and structure. The first rows define the columns available for different types of data. The first column of the dataset specifies the data type.

As we can see from Figure 5.2, the first three rows (of type, respectively, `ST`, `AL` and `BB`) have a different number of columns. This causes problems when reading the dataset, and requires splitting the file in three datasets, each for a data type.

The first line is, instead, handled similarly to the situation described in the previous example.

[6] Some information are under a non-disclosure clause and as such have been anonymized.

Date	Filename	Extension
11/04/2019	73	XLS
10/04/2019	52	XLSX
09/04/2019	31	XLS
08/04/2019	03	XLS

Table 5.3: Filenames and extensions for files downloaded from `Terna.it`. All files are related to the same category.

5.3.2 Unpredictable File Names

The website of Terna S.p.A.[7] is an example of technical problems which heavily influenced the extraction process.

Premise Terna provides each day an Excel file containing several data. An intuitive approach would be to download directly these files.

However, due to the inconsistencies described in the next paragraph, it became necessary to navigate the website with a web scraper, which is far more complex to develop than a simple downloader.

Inconsistencies The main problem with the Terna website is that there isn't any naming convention for the Excel files.

As we can see from Table 5.3, files related the same data category for consecutive days have both very different names and different extensions. Moreover, some files marked as `.xlsx` are actually just `.xls` with a wrong file extension.

Web scraper Since it is impossible to predict which files to download, the only possible approach is to navigate the website and download the files manually or through a web scraper. The downloader uses Selenium to navigate and interact with Terna website, and to retrieves the URLs of several `.xls` and `.xlsx` files, which will be downloaded by Databricks.

This process is by far slower than a normal downloader, but allows us to bypass the filename issue.

5.3.3 Website Navigation

Damas[8] is an example of a website which requires complex navigation logic.

The website offers a simple interface for selecting a time range and showing the data for that given range. However, this system presents two limitations.

[7] Terna S.p.A. is the most important Transmission System Operator in Italy. It manages almost all of the Italian energy transmission grid.

[8] Damas is a private web portal managed by Terna S.p.A. It provides data related to energy transfers between Italy and bordering countries. Information from this website are private, meaning that each company can only see their own energy transfers.

Limited date range First of all, the maximum date range allowed is 31 days. If a user tries to perform a query on a bigger date range, the system shows an error message.

This means that it is necessary to perform 12 queries to download a whole year.

No DST The other problem is that if a date range contains a DST change, the website shows the following error message:

```
One of the days in the interval BUSDAY(30.03.2019) and
  BUSDAYTILL(31.03.2019) contains clock change days.
```

As shown by the message, it is impossible to download even a small amount of days, if a DST change occurs between them.

The solution would be to download these dates individually, while downloading the rest of the data in ranges smaller than 31 days. However, since each year has different dates for DST changes, it would be necessary to set up these date manually for each year in the downloader.

In the end, the best solution, even though it's computationally less effective, is to download each day individually. In this way there is no need to group dates into ranges and to pay attention to DST changes. The download process, on the other hand, is slower than downloading data in a batch.

Chapter 6

Data Warehouse Structure

The data warehouse has been built with a particular structure, suited for both storing and processing the huge amount of data needed, as well as dealing efficiently with its high heterogeneity.

The various schemas created, as well as how they contribute to processing and exposing all the information, will be described in this chapter. A final section will elaborate on a particular type of tables, which plays a very important role in the Data Warehouse.

6.1 Schemas

The data warehouse contains several schemas, each one serving a specific purpose.

Two schemas, called *Source* and *Staging*, are used for loading the data, while a different one, called *Data Mart*, contains the final data.

The first two schemas behave differently from the latter, since their purpose is to process the content of each `csv` file when inserting new data into the Data Warehouse. These tables are emptied each time a different file is processed.

Several additional schemas, are also present in the data warehouse, but they have a less important role.

6.1.1 Source

The *Source* schema, named `src`, is where Databricks directly loads the `csv` data at the end of the ETL process.

It contains a table for each data stream downloaded by Databricks. The structure of these tables is identical to the `csv` structure. All fields are of type `VARCHAR`, since all values loaded from `csv` are by default interpreted as strings.

Each table in this schema has associated a view. These views are used to cast the data to their correct types as well as rename the fields into a standardized format.

6.1.2 Staging

The *Staging* schema, called `stg`, loads the data from the `stg` schema views and performs some more operations on them.

The number of tables in this schema is much lower than the one in the `src` schema. This is because the data in this schema are aggregated together.

Example Let us take for example weather data. Information about weather come from several providers. Each one has different information: some providers may contain data about temperature and humidity, while others may have information about gas or power demand[1]. In other cases, for example, a provider may be related to a single zone (e.g., energy consumption in Austria), so multiple providers are needed to get the whole European picture.

In all these cases, it is necessary to group together data from different providers, since the information they provide are strictly correlated.

This aggregation is done in this schema by apposite procedures. These procedures read data from the views present in the `src` schema.

6.1.3 Data Mart

The *Data Mart* schema, called `dm`, contains the first layer of data to be consulted by users.

Data are loaded into this schema from the *staging* stables.

Tables in this schema are meant to be consulted by end-users for their daily needs. Information contained in this schema are both up-to-date and complete (i.e., each table contains data related to several years back and there are no missing values).

6.1.4 DBO

The `dbo` schema contains remappings, which are used to normalize some data.

Remappings will be described in section 6.2.

6.1.5 Configuration

An interesting schema is the one called `configuration`, which contains some parameters which influence the ETL process executed by Databricks.

Amongst all the parameters, two are of particular interest: the download time range, as well as retry information, in case of download failure. Table 6.1 is an example of configuration parameters stored in this schema.

[1] These information are correlated. For example, on a cold day there is more need for heating and, as a consequence, a higher gas demand.

Provider	Stream	From	To	Retries	Retry Delay
Weather_A	Temperature	01/01/2017	NULL	3	10
Weather_A	Humidity	01/01/2017	NULL	3	10
Weather_B	Temperature	01/01/2019	NULL	2	15
Weather_B	Pressure	01/01/2018	NULL	5	5

Table 6.1: Some configuration parameters.

Download time range These information are used to specify when to stop when downloading data from a given provider.

This parameter has two uses: first of all, in case the provider has more historical information than what needed, it limits the amount of data recovered. This not only reduces history download time, but also prevents too much unneeded data from being loaded into the data warehouse. In the opposite case, this parameter is also needed not to attempt to download historical data from a provider if we know it is not available. This prevents the downloader from encountering errors caused by non-existent files.

A `NULL` value specifies that there is no time range limit.

Retries The downloaders have been programmed to attempt again to download data in case the provider timeouts.

These parameters specify both how many attempts to perform, as well as the required delay between each attempt.

Given the different importance and update frequency of the different providers, these values differ depending on which data is being downloaded.

Remarks It is important to notice that these parameters are not specified for each provider, but for each data stream.

This decision has been made considering two factors.

First of all, sometimes data from a single provider is recovered from multiple sources (e.g. different websites belonging to the same organization). Each source can present a different level of availability.

Secondly, some data streams are more important than others. It is reasonable to spend more effort downloading this kind of data if they happen to be unavailable, otherwise some critical processes may stop working.

6.1.6 Other Schemas

The Data Warehouse contains also other schemas, which play, however, a less important role in the data processing procedures.

For example, the schema `trd` (*Trading*) contains some views explicitly requested by the trading team. These views are just a simple manipulation of the data stored in the *Data Mart* schema. Their purpose is exposing some data in a particular format, which is required by some tools used by the trading department.

6.2 Remappings

Some information can be expressed in different ways.

For example, the same hour can have different values depending on the timezone used. Some providers also count hours from 0 to 23, while others use the most common 1-24 notation.

When dealing with this kind of data, it is important to choose a consistent and uniform way of these information.

This action is done through additional tables, used to remap the original data to the uniform notation used.

6.2.1 Logic vs Materialized Remappings

Remappings can be either logical or materialized.

The former are created by joining any table with a remapping table, producing the results in the query output. In the latter case, on the other hand, the results are physically stored on the data warehouse and do not require any join operation.

Let's now analyze the advantages and disadvantages of each remapping technique.

6.2.1.1 Query performance

The type of remapping chosen can directly influence the performance of a query.

Logical remappings Logical remappings require to join each value of a given table with the remapping. This operation is applied to a large amount queries used by Axpo, since they require some information in a specific notation.

The cost of the join is however low, since these remapping tables are usually very small, ranging from tens to a couple hundred rows. As such a single join can be computed very quickly, but it can become a problem if these operations need to be applied for almost each query.

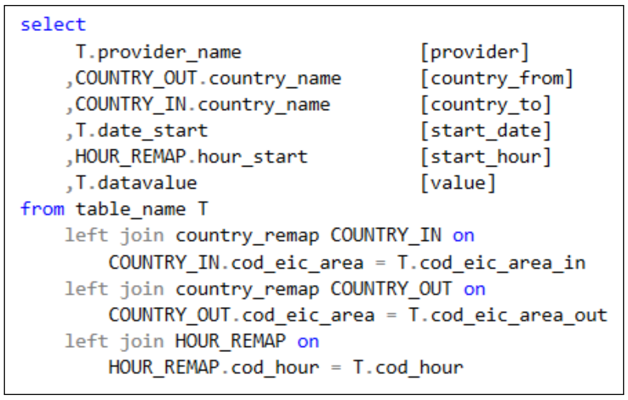
Materialized remappings Materialized remappings require no additional overhead, since the remapped information are physically stored along the other data.

This choice is ideal for values which are always used in their remapped notation. For example, each provider uses its own hour notation, while all Axpo tools use a standard numeric notation ranging from 1 to 25.

Instead of having to perform a join for _every query, it is more efficient to memorized this value directly into the table, removing the join overhead.

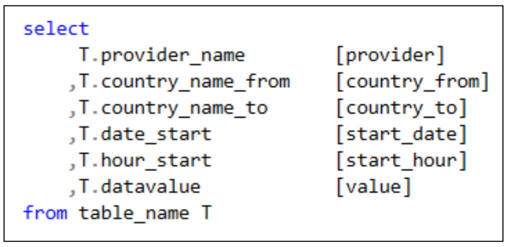
6.2.1.2 Query complexity

Queries become more complex since they require join operations with additional tables. This scenario is even more complex when multiple remappings are needed for the same



```
select
     T.provider_name                   [provider]
    ,COUNTRY_OUT.country_name          [country_from]
    ,COUNTRY_IN.country_name           [country_to]
    ,T.date_start                      [start_date]
    ,HOUR_REMAP.hour_start             [start_hour]
    ,T.datavalue                       [value]
from table_name T
    left join country_remap COUNTRY_IN on
        COUNTRY_IN.cod_eic_area = T.cod_eic_area_in
    left join country_remap COUNTRY_OUT on
        COUNTRY_OUT.cod_eic_area = T.cod_eic_area_out
    left join HOUR_REMAP on
        HOUR_REMAP.cod_hour = T.cod_hour
```

(a) Logical



```
select
     T.provider_name          [provider]
    ,T.country_name_from      [country_from]
    ,T.country_name_to        [country_to]
    ,T.date_start             [start_date]
    ,T.hour_start             [start_hour]
    ,T.datavalue              [value]
from table_name T
```

(b) Materialized

Figure 6.1: Query complexity for different remappings. Both queries produce the same output.

table.

Logical remappings For local remappings, each remapping needed results in a join operation.

Sometimes, multiple columns referring to the same type of information need to be remapped. This leads to a more complex query, in which it is necessary to remember the name of each remap in order to avoid confusion.

As an example, the query shown in Figure 6.1a represents a query using logical remappings. As we can see, multiple join operations are needed, making the query harder to read.

Materialized remappings Materialized remappings do not need any join operation, since the data is directly present in the table. As a consequence, queries are both easier to write and to read.

The query shown in Figure 6.1b represents the same query as above, but uses materialized remappings. The query is certainly easier to read and to understand, and the possibility of making errors is minimized.

6.2.1.3 Data insertion

When a new row is inserted additional operations have to be performed depending on the remapping type.

Logical remappings Logical remappings are more convenient during insertion, since they do not require any additional operation on the values inserted.

The only check needed is that each value to be remapped is present in the remapping table.

Otherwise an alert must be raised and an appropriate remapping has to be created.

Materialized remappings In case of materialized remappings, it is necessary to compute each remapping prior to the insertion operation.

In case of missing values, there are two possibilities: either the row is not inserted or it is inserted with a temporary remapped value of `NULL`. In both cases an alert must be raised.

6.2.1.4 Remapping changes

Remapping tables can be changed if errors are detected. Depending on the remapping type used for a specific values, different scenarios may occur.

Logical remappings Logical remappings handle changes to remapping tables very efficiently.

Since the remapped value is computed at query time, no additional operation are needed on the table.

Materialized remappings Materialized remappings need to be updated each time the remapping table is modified.

All remapped values are computed again by an apposite procedure (the same used during insertion). In case of incomplete remappings (e.g., a value from the remapping table has been removed), an alert must be raised.

6.2.2 Examples

In this section, I will provide a few examples of the remappings created by Reply, to give a more hands-on idea of the notations used by different providers, as well as how they have been normalized.

Since I was constantly analyzing the Data Warehouse, I often noticed several small problems with remappings, such as wrong associations or missing values. Each time I noticed a problem, I analyzed the cause and told Reply how to fix the issue.

In other cases, I directly asked Reply to remap additional data, which could be used to improve query performance or to simplify existing queries.

6.2.2.1 Hours

The energy market presents different problems related to hour handling. We will see them in detail, along with how they have been remapped.

Provider	Hour format
EU Market	7
EU Market	Hour 7
EU Market	Volume H07
IT Market	7
EU Transfer	06
IT Transfer	06
IT Transfer	6
IT Transfer	H07

Table 6.2: Different notations for the same hour (7 am).

Notation The most apparent issue is the lack of a standard notation amongst all providers.

Table 6.2 shows an example of how the same hour is referred to by different providers.

These different notations cause many problems during queries, since the same value has different meanings depending on the provider.

As such, it is necessary to devise a solution to normalize all the hours into a single notation, independently of the provider.

Daylight Saving Time One of the problems most commonly encountered is caused by Daylight Saving Time (DST).

On the last Sunday of March, time is shifted one hour forward at 2am while, on the last Sunday of October, time is shifted one hour back at 3am.

As a consequence, in March we have a day with 23 hours, since the hour between 2 and 3am is skipped, while in October we have a day with 25 hours, since the hour between 2 and 3am is repeated twice.

The main problem is that each provider handles these hours in their own way, using their own notation. As a consequence, the same value may have different meanings depending on the provider.

Some providers, for example, start numbering the hours from 0, while others start from 1. In this case, it becomes clear that a value of `1` can indicate either `0am` or `1am`.

An additional problem is also given by the lack of a standard notation for expressing the hours between 2am and 3am on the last Sunday of October, which are repeated twice. The second repetition needs however to be distinguished with a different notation. Table 6.3 shows how these hours are referred to by different providers.

Gas Day Gas providers handle dates in a different way, using a particular notation called *Gas Day*. Gas days start at 6am and end at 6am of the next day.

Differently from standard dates, there are no DST changes, which means that each day across the whole year has 24 hours. During DST, each day is as a consequence shifted by an hour (gas days start and end a 7am).

As a consequence, each file retrieved from gas providers contains data about to two

Provider	0am	1am	2am	2am (2)	3am
Forecasts	0	1	2	3	4
Weather	0	1	2A	2B	3
Market A	1	2	3	4	5
Market B	1	2	3	4	5
Market B	Volume H01	Volume H02	Volume H03A	Volume H03B	Volume H04
Market C	1	2	3	3B	4
Market C	Hour 1	Hour 2	Hour 3A	Hour 3B	Hour 4
Market C	Volume H01	Volume H02	Volume H03A	Volume H03B	Volume H04
Market D	1	2	3	4	5
Transfer A	1	2	3	4	5
Transfer A	ORA01	ORA02	ORA03	ORA04	ORA05
Transfer B	0	1	2	3	4
Transfer C	0	1	2A	2B	3
Remapped to	1	2	3	4	5

Table 6.3: Different hour notations during DST changes (on last Sunday of October). The last row specifies the remapped value for all occurrences.

different days: the hours 6-24 are related to a given day, while the remaining (24-6), are related to the following one.

Timezones Some websites provide dates in local time, which means that these dates must take into account GMT.

Additional problems are presented by DST changes. For example, DST changes in the UK occur between 1am and 2am (local time), which, taking into account the GMT difference, is the same time as 2-3am (the moment DST changes in Italy).

This means that the same hour not only changes meaning depending on the provider, since each one has its own notation, but also on the geographical zone.

Solution The solution is to create a table which specifies how to remap each hour into their actual value.

Considering that most queries will need this kind of data, as well as the fact that remaps are likely not to change for existing data, it has been chosen to materialize the remapped hours directly into the tables.

In this way, users will expect faster computational times, as well as simpler queries.

6.2.2.2 Countries

Countries can be referred to in different ways. As such it is necessary to normalize these information.

Problems Providers refer to countries in two different granularities.

Original	Remapped
Belgian	Belgium
Belgium	Belgium
Germany	Germany
germany_luxembourg	Germany
Germany/Luxembourg	Germany
netherlands	Netherland
Netherland	Netherland
PT	Portugal
Portugal	Portugal

Table 6.4: Country remappings.

The most generic information indicates the country itself, while the most detailed specifies a smaller zone, which belongs to a specific country. These zones are specific to the energy sector and divide a country based both on geographical and energetic factors.

Not all countries are divided in zones.

Countries Most providers use the same notation for expressing countries.

However there are a few exceptions. For example, some providers spell countries in a different way or using a different case. Some other providers refer to countries using a two-character ISO notation[2].

As such, the data needs to be remapped into a common notation, to preserve consistency amongst all data. This remapping is done through an apposite table, such as the one shown in table 6.4.

Country zones Country zones are either referred to directly or by using a particular notation known as `EIC`[3].

In the first case no normalization operations are needed, while in the second, it is necessary to remap these codes to both a normalized version of the zone name as well as to the country it belongs to.

Table 6.5 shows an example of `EIC` codes, as well as their remapped values.

Solution Two additional tables have been created, containing remapping information for both countries and zones.

These tables are similar to tables 6.4 and 6.5.

[2] Standard notation commonly used worldwide. For example, Italy is abbreviated 'IT' and United States 'US'.
For a comprehensive list of country codes, see [17].

[3] *Energy Identification Codes*. These coding scheme is used across Europe to facilitate cross-border exchanges and to efficiently and reliably identify different objects and parties relating to the Internal Energy Market and its operations [18].

EIC	Zone	Country
10YIT-GRTN——B	Italy	Italy
10Y1001A1001A73I	Italy_NORD	Italy
10Y1001A1001A788	Italy_SUD	Italy
10Y1001A1001A75E	Italy_SICI	Italy
10Y1001A1001A83F	Germany	Germany
10YDE-ENBW——N	Germany_TransnetBW	Germany

Table 6.5: EIC codes and remapped values.

Provider	Original	Remapped
Provider_A	P	Purchase
Provider_A	S	Sell
Provider_B	Purchase	Purchase
Provider_B	Sell	Sell
Provider_C	BID	Purchase
Provider_C	OFF	Sell
Provider_D	C	Purchase
Provider_D	V	Sell

Table 6.6: Bidding operation type remappings.

Since not all queries require this kind of data, the remapping are performed only at logical level. As such, new queries require to perform join operation with the remap tables.

On the other hand, several views created for specific departments already perform these join operations, simplifying user interaction.

The size of these tables are small (less than 100 rows), so the performance impact is negligible.

6.2.2.3 Bidding Operation Types

Some providers use different notations when referring to bidding operations type.

Market bids can refer either to purchases of sales. Each provider represents this information in its own format.

As such, it is necessary to create a remapping table to normalize this notation. The table needs to know both the original format as well as the provider, since different providers may use the same value for different things.

The resulting table is similar to table 6.6.

The table hasn't been materialized given its very small size (less than 10 elements).

Part III

Experimental Evaluation

Overview

Several different tests have been executed to assess the quality of the migration.

I decided to focus on two main aspects:

- Data Quality
- Performance

Data quality is the most important requirement asked by Axpo: the accuracy of their business choices is strictly dependant on the quality of the data they are working with.

Performance is also a very important factors, since several critical business processes need to access data in short amounts of time.

This part will describe how each test has been carried out, as well as the results obtained. Some problems encountered when assessing data quality or performances will also be discussed, as well as their impact on the testing phase.

Chapter 7

Data Quality Tests

Several tests have been performed on the Data Warehouse to assess the correctness of the data stored in it.

7.1 Test Types

7.1.1 Range Tests

This kind of test asserts that each database has data related to the same date range.

The minimum and maximum dates for each table are extracted from each database and compared. Results are acceptable if the Data Warehouse contains at least the whole range present in the on-premise database.

These tests initially recovered the date range over the whole dataset, but were later modified to retrieve more detailed information, computing the total number of hours[1] downloaded for each category[2].

Exceptions It is necessary to analyse all the results manually, since several exceptions can be made for the above rule.

Very old data One possible exception is that of very old data. Some information related to several years ago are not currently used by the company and as such are not strictly necessary.

Moreover, it may be currently impossible to retrieve these information from their providers, since they may no longer be accessible.

On the other hand, if these information are needed it may be possible to perform a one-time manual data import from the on-premise database to the Data Warehouse.

[1] The number of hours is an important metric since most data are timeseries at hourly granularity. If the hour count is equal to the actual number of hours in a given time period, it means there are no “holes” in the dataset. These missing values could not be noticed with the initial test.

[2] With *category*, we mean all the relevant attributes needed to distinguish between two different values. This is important especially for forecasts, since multiple publications of the same hour must count as one (since the last value is always the most accurate).

Scheduling If a table contains all data required except for the current day (or the next day in case of forecasts) it may be a problem related to the scheduling chosen for downloading data.

For instance, the on-premise database may download data in the morning while the Data Warehouse downloads the same information during the afternoon. These test are thus dependent on the time they are executed. To avoid this limitation, the last two days are not taken into account when performing these tests.

Company needs The most discerning element is, as always, the actual needs of the company.

For example, if only the last year of some type of data is actually used, it is irrelevant if the on-premise database stores more years than the Data Warehouse, as long as the latter has the range required by the company.

Failure In case of failure, if no exceptions can be made for the data, Reply is notified and tasked with solving the issue.

Solutions are specific to the kind of data of the provider. In some cases they may be as simple as changing a few parameters in the history retrieval settings, while in other cases, they may require more complex interactions and analyses of the provider, as well as additional development of the ETL procedures.

7.1.2 Key Tests

This kind of test asserts that each database contains data related to the same information.

It is a comparison based on the keys which identify a specific datum. Specific queries extract all the keys from each table in both databases and compare them.

The most basic comparison is to perform an intersection between the two key datasets and count the number of obtained elements. This number, compared to the element count in the on-premise database, gives us a percentage of how many rows are in common between the two databases.

More advanced tests are done manually, since they require in-depth knowledge of the domain, and are used to adjust the queries used to extract the keys.

For example, it is sometimes necessary to aggregate or deaggregate some data, to use particular remappings or to rename some values (for instance, some values may be written all uppercase in a database and camel-case in the other; in other cases there may be additional characters, such as underscores).

Exceptions Usually the tests are repeated until all the data in the on-premise database is present in the Data Warehouse.

However, depending on the company needs, it may be enough to settle for a lower value, depending on how the data are actually used.

Failure In case of test failure, Reply is notified with the problems encountered and will provide a fix. These tests will then be repeated until all keys are correctly present in both databases.

7.1.3 Value Tests

The last kind of tests asserts that the two databases not only contain data related to the same information, but that they also contain the same data.

This test is similar to a Key Test, but takes also into account the values associated with each key set. This kind of tests is able to expose problems related for example to wrong aggregation or deaggregation operations.

The tests are considered successfully passed if we obtain the same common data percentage as the final results of the Key Tests (which is usually 100%, unless it has been decided a lower value is enough for the company needs).

Failure In case of failure, Axpo and Reply work together to understand the cause of the problem.

This collaboration is necessary since Axpo can provide technical domain information, while Reply has more knowledge related to the ETL process. Once the problem is identified, Reply will provide a fix.

These tests will be then run again, until no further problems are found.

7.2 Testing Suite

Each test needs to join together results obtained from different databases.

This operation is not trivial, since a query cannot be directly performed across multiple databases.

The solution I opted for was creating a script on Databricks, from which each database can be queried independently, the results can be put together and additional operations can be performed.

7.2.1 Why Databricks?

The test suite has been developed on Databricks for several reasons.

Already accessible First of all, this environment was already accessible, since it has been used for the ETL process. Using an existing environment requires little to no additional configuration, and its costs are already included in the project budget.

Multiple languages Secondly, Databricks supports multiple languages in the same workbook. The query results have been merged using Python, while additional analy-

```python
JDBC_URL_ON_PREMISE_DB = "<db address>"
JDBC_URL_DATA_WAREHOUSE = "<db address>"

def exec_query(query, conn, tablename):
    df = spark.read.jdbc(conn, "({}) query".format(query))
    sqlContext.registerDataFrameAsTable(df, tablename)
    return df

# make sure the query has the DISTINCT clause, otherwise Key tests will not be accurate!
query_db = '''<insert query>'''
DB = exec_query(query_db, JDBC_URL_ON_PREMISE_DB, 'DB')
DB.cache()
count_db = DB.count()

# no need to use DISTINCT, EXCEPT takes care of duplicates
query_dwh = '''<insert query>'''
DWH  = exec_query(query_dwh, JDBC_URL_DATA_WAREHOUSE, 'DWH')

diff = sqlContext.sql('select * from DB except select * from DWH')
diff.cache()
sqlContext.registerDataFrameAsTable(diff, 'DIFF')
count_diff = diff.count()

print('On-premise:    {}'.format(count_db))
print('Correct:       {} ({:.3%})'.format(count_db-count_diff, 1-(count_diff/count_db)))
print('Wrong/Missing: {} ({:.3%})'.format(count_diff, count_diff/count_db))
```

(a) Python

```sql
%sql

select
   DIFF.*
  ,DWH.value as dwh_value
from DIFF
  inner join DWH on (...)
```

(b) SQL

Figure 7.1: Testing suite.

ses have been performed on the resulting dataset in SQL, taking full advantage of the strengths of both language.

Spark A third important aspect is that Databricks natively supports Spark, meaning that each database could be queried at the same time. Moreover, each query is automatically parallelized to further improve their performance.

7.2.2 Script Description

The script created, shown in Figure 7.1, requires as input two queries.

The first query will be performed on the on-premise database, while the second on the Data Warehouse. These queries must return data in the same format, since the results are going to be merged.

By using Spark, the scripts performs a join between both results, taking all values from the on-premise database which have not been found in the Data Warehouse.

From this new dataset it is possible to compute the data match percentage between the two databases.

SQL A separate cell, using SQL, allows users to perform *ad-hoc* queries on the mismatching values, which has been used abundantly to analyze the problems found.

For example, the query shown in 7.1b compares the values stored in each database, excluding the rows which are present only in the local database. This query proved to be extremely useful for understanding issues related to Value tests.

7.3 Test Results

Several tests have been executed on the Data Warehouse.

Each time, the testing suite has been modified to increase the result accuracy.

7.3.1 Preliminary Results

Preliminary tests have shown many problems related with the ETL process.

Range tests Range tests have shown that 50% of the data streams on the Data Warehouse have at least the same date range than the on-premise databases.

Out of 38 tests executed, we had the following results:

- 17 data streams had a greater date range on the Data Warehouse than on the on-premise databases.
- 2 data streams had the same range on both databases.
- 16 data streams on the Data Warehouse had a smaller date range compared to the local databases.
- 3 Data Warehouse data streams were empty

Key tests Key Tests results were more disappointing at first. Out of 38 possible tests, we were able to perform only 16. The other tests could not be written because there were information (for example, columns or remapping tables) missing.

Out of these 16 tests, only two initially gave acceptable results, with 98% and 92% of the data contained in the on-premise database being available on the Data Warehouse. All the other results were pretty low, with 5 of them around 30%, 2 lower than 10% and the remaining 8 at 0%.

Further analysis These results, especially the ones at 0% meant it was necessary to further analyze the situation, to understand if the issue laid within the Data Warehouse or the testing suite itself.

We began exploring both databases manually, with the support of Axpo domain specialists. Several differences in the data formats had been found. For example, some fields contained manually generated data, while in other cases an hour notation different from the standard format was used in the on-premise databases. In some other cases, the data was supposed to be different, so this kind of test could not be applied (see section 7.4.2 for details).

This lead to an improvement of the testing suite, now taking into account these differences.

7.3.2 Test Suite Improvements

After some improvements in the test suite, the tests were then executed again, since the first results were proven not to be representative of the Data Warehouse quality.

Range tests Range tests were changed to obtain more detailed information. Overall, the results obtained were very similar to the previous ones, but a few additional issues were discovered.

For example, we noticed that in some cases date ranges varied depending not only on the provider, but also on the country for the same provider.

These results indicated that some problems were related only to particular cases of the ETL process.

Key tests Key tests results became much more promising, meaning that the main problem didn't reside in the Data Warehouse but in the testing suite itself.

Out of the initial 16 tests, we obtained a much higher consistency level, as shown in Figure 7.2.

A high number of tests were successful, with 3 tests having an accuracy greater than 99%. A value of 100% is very difficult to achieve, since the data in the on-premise databases is downloaded at a different moment than that on the Data Warehouse. This difference can produce some false negative results, but they will always be related to a single day (usually the current one). As such, a value greater than 99% can be considered an acceptable threshold.

It was also shown that there were some inconsistencies between the two databases, but we were also able to identify their cause. Two tests failed because the required information had not been downloaded at all, one because hours had been remapped incorrectly, while the other failures were caused by a smaller date range.

In the first case, the on-premise database contained information about specific countries or models, while the Data Warehouse contained information about different countries or models. This issue can usually be solved by a simple change in the downloader configuration.

The two other problems were equally easy to solve. For the remappings, it was necessary to fix some values in the remapping table and rerun the procedure to materialize

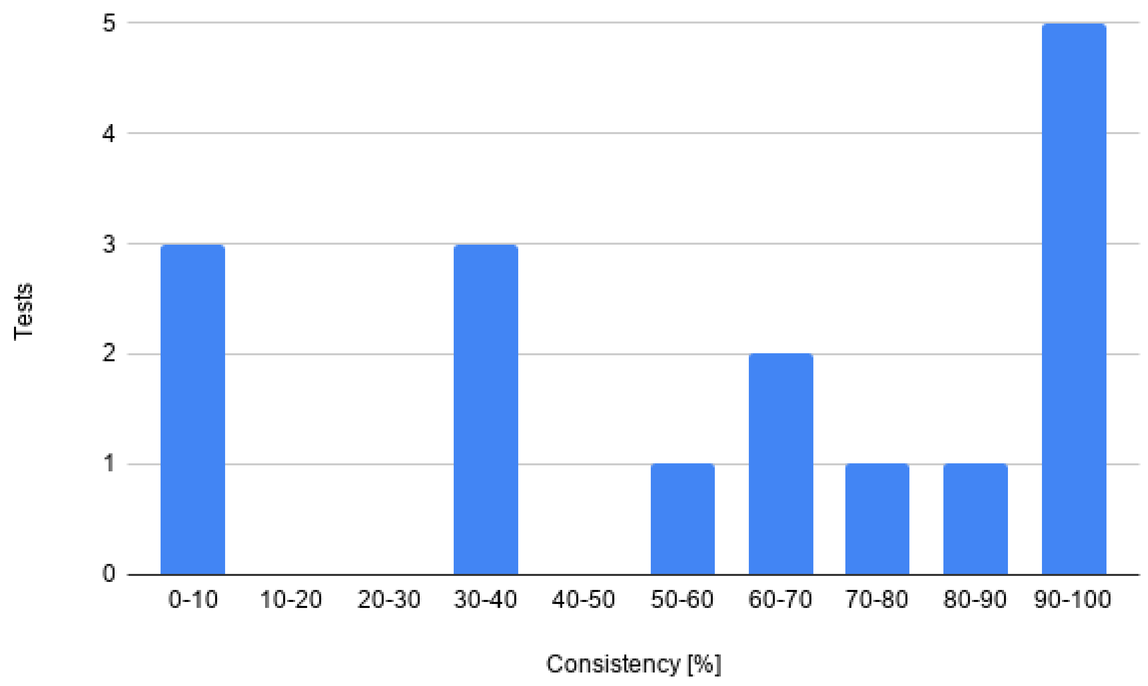


Figure 7.2: Consistency level histogram on key tests.

the correct values, while for the data range it was required to change the downloader configuration.

Value tests Considering the acceptable results obtained with the Key tests, we decided to assess the correctness of the values. Out of 16 queries, 7 were related to forecasts, for which this kind of tests could not be applied. As a consequence, only the remaining 9 tests have been executed.

The results, as shown in Figure 7.3 were very positive, with 6 tests (66% of the testing suite) having 100% accuracy, while the others gave respectively 64%, 47% and 5%.

It should also be noted that in some cases the values stored in the on-premise databases were wrong, while those present in the Data Warehouse were correct. In these cases, it is impossible to perform an accurate analysis, and lower values are also accepted. This process is however carried out manually and these kind of exceptions are made for specific cases after a careful analysis.

7.3.3 ETL Fixes

The identified issues have been communicated to Reply, along with examples of incorrect values.

At the time of writing, Reply has only fixed issues related to Value tests, meaning that the overall data quality is still going to improve with time.

The tests have been executed once more, obtaining excellent results, as shown in Figure 7.4.

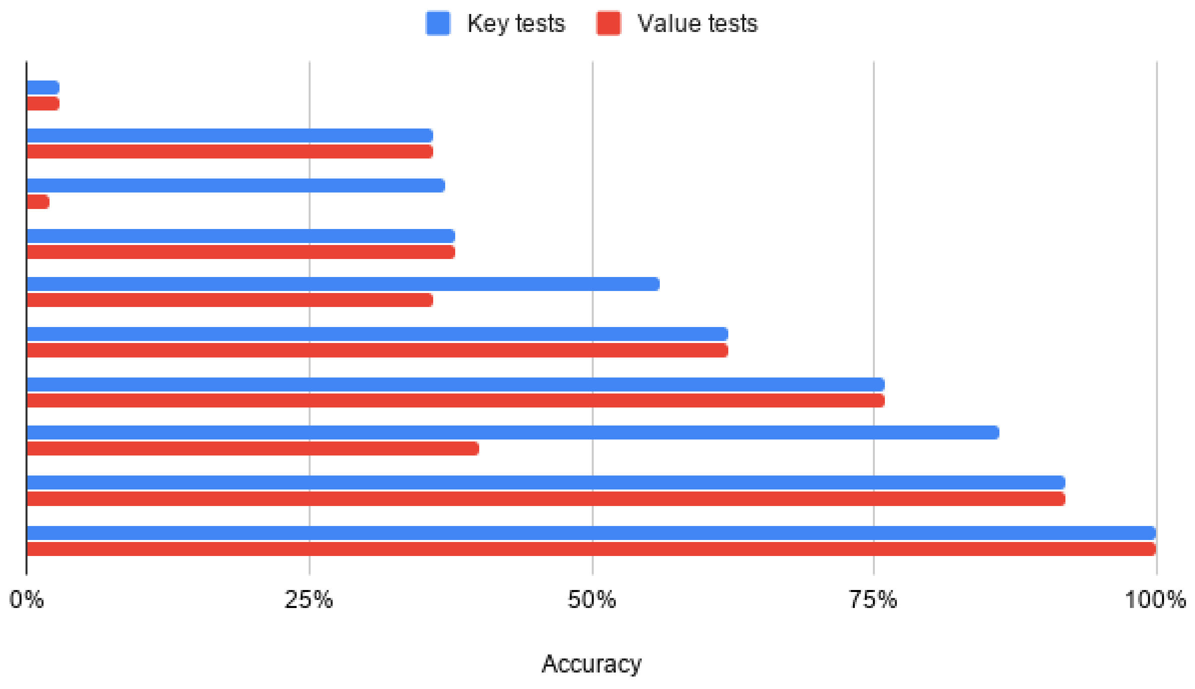


Figure 7.3: Key-Value test results.

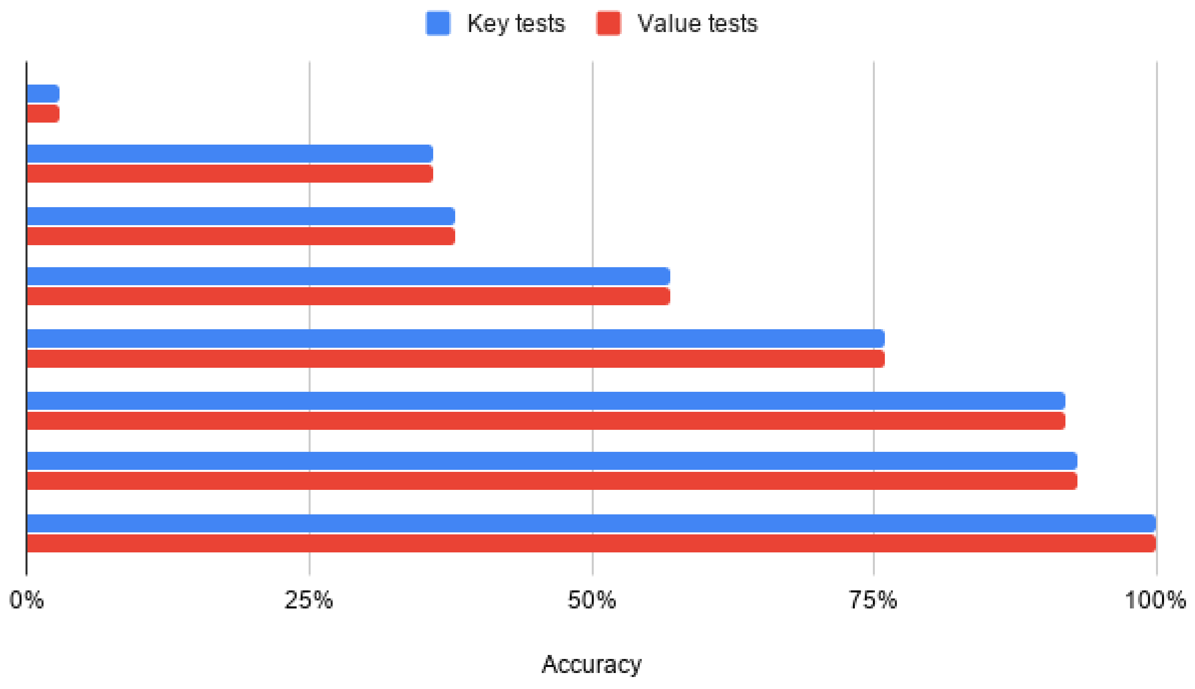


Figure 7.4: Key-Value tests after Reply fixes.

We can see that each Value test has the same accuracy level as their respective Key test. Value tests, on the other hand are still incomplete.

From these results, we can understand that the ETL process is not yet able to download all the required data, but each downloaded information is correct.

7.3.4 Conclusions

The results discussed above show that the ETL process, once properly tuned, is effective at downloading data and that the downloaded information are correct.

Some problems, however, are still present in the Data Warehouse, which, once fixed, will certainly increase the results of the tests.

It has been shown that building a proper testing suite is a very complex matter, and a poorly constructed one can prove ineffective at detecting issues either by giving false negative or false positive results.

False negatives A false negative result happens when a given datum appears to be different from the on-premise database but in reality is correct. This can happen for two main reasons.

In some cases, values have been processed in the on-premise database, but this process has not been replicated correctly by the test. In this case, it is necessary to apply the same kind of data transformation in the test.

Another possible scenario is caused by a different handling on `NULL` values. For example, some coefficients for specific hours are sometimes not specified by the providers. The Data Warehouse stored these missing values as `NULL`s, while in the on-premise database they were assigned default values, which were different depending on the type of coefficient.

False positives A test can also produce false positive results, especially if the test is too much generic. These results can hide existing problems in the Data Warehouse, showing that there are no problems.

For example, let us consider a table containing information about multiple countries. Let us also assume that for Austria we have data ranging from 2015 to 2019, while for Germany (for which we should have the same range) we are missing data related to 2019.

A generic test, computing the minimum and maximum date ranges across all data would show a range of 2015-2019, which means that there are no problems. This result is, however, a false positive. A more in-depth test, testing the minimum and maximum date ranges for each country would however notice the issue.

Most tests are however more complex than this example however, since it is necessary to aggregate information across a large number of dimensions.

7.4 Problems

Several problems have been encountered as a result of the tests carried out. After each problem has been solved, we executed the tests once again, obtaining better results.

Perfect results have not been achieved yet, meaning that more problems need to identified and solved. The results obtained are however enough to show the effectiveness of the testing suite.

This section will provide a description of the main problems identified.

7.4.1 Pivoting

In some cases, tables on the on-premise database are pivoted differently than those on the Data Warehouse.

If the company requires a specific output format the table on the Data Warehouse will be unpivoted accordingly, meaning that it will then be possible to run the tests on the new table. On the other hand, if both formats are acceptable and no unpivoting operation is to be performed, it is impossible to execute Key and Value Tests, since the structures are too much different.

In this case a few manual tests will be executed to assert at least a degree of correctness, even though a comprehensive testing suite has not been developed.

7.4.2 Forecast Publications

Forecasts are very difficult to compare.

The values downloaded from a provider are the results of a specific run of a certain algorithm. Different runs usually give different results, since the input data is different. Moreover, these algorithms are proprietary, so it isn't possible to investigate their behaviour.

Forecasts are published multiple times a day. Depending on when the data is downloaded, different values are obtained.

The main problem arises from different download schedules between the on-premise tools and the ETL processes developed by Reply. Different download schedules mean different values obtained, which invalidate most tests. As a consequence, this kind of data can only be tested on download ranges. Additional tests have to be performed by domain experts during real data application scenarios (i.e., running a tool with the new data and manually seeing how well it performs).

7.4.3 Floating Points

Floating point values present some limitations when performing comparisons. This kind of data is by design approximate, meaning that it won't store the exact value but an approximation of it [19].

This is not usually an issue, since the approximation is extremely close to the actual value, but it can present some problems when performing equality checks, since they are based on exact values. In this case, values which may appear to be the same are considered different, resulting in wrong metrics.

The solution is to convert all floating point values to the `DECIMAL` data type, which can support a specified number of decimal digits.

Decimal data types are stored differently from floating points, the latter being based on a value multiplied by an exponent, while the former being just a plain representation of the number, with the possibility of specifying how many digit are decimal.

Since decimal data types represent exactly all the digits in the number, operations such as equality checks and intersections (which internally rely on equality checks) can be performed without fear of getting wrong results.

7.4.4 Different Values

In multiple cases, Key tests indicated a high consistency level but Value tests gave poor results.

I analyzed each issue with the help of Reply and identified the proper solution. A consistent number of problems was caused by operations applied on the data in the on-premise database. These operations were specific to the type of data, so there wasn't a unique or common solution.

For example, I notice that the quantities of energy transferred between different zones were different, although all the keys were correct. This meant that the Data Warehouse contained the correct datum but with the wrong value, compared to the on-premise database.

Website data The first check we did after having identified the problem was controlling that the values present on the website were identical to those stored in the Data Warehouse.

The website had two different categories of data, named *Day Ahead* and *Total*. The specifics given by Axpo indicated that the value to download was *Total*.

We compared these value with the results obtained from the Value test and noticed that the data in Data Warehouse was correct, while the on-premise database contained different numbers.

Data operations After having identified this problem, I asked the Axpo specialist who used this kind of data if the local database performed some operations on the data. It turned out that the quantities stored in the on-premise database were the net transferred amount and not the whole amount.

Table 7.1 shows the difference between whole and net transfers amount. In the first case, we record how much energy each zone has transferred, while in the second we only store the difference between the two values. This behaviour has been replicated in the Value test.

Different data stream Even with this normalization operations in place, the Value test was still indicating the presence of several errors.

While comparing the newly computed results obtained from the test with the provider's website, we noticed that the values of the on-premise database matched the *Day Ahead* column from the website.

Amount	From	To	Quantity
Whole	A	B	100
	B	A	35
Net	A	B	65
	B	A	0

Table 7.1: Whole vs net energy transfer amounts between zones.

This problem has been presented to Axpo, and we understood it was necessary to change the initial requirements, adding this kind of data to the original planning.

Chapter 8

Performance Tests

Multiple tests have been performed to assess the performance of the Data Warehouse, comparing it with the on-premise databases.

8.1 Test Types

8.1.1 Metrics

The first approach has been to analyze some metrics internal to the database.

These metrics are automatically managed by the database itself and show a huge variety of information about the current configuration of the database.

Dynamic Management Views In SQL Server, Dynamic Management Views (*DMVs*) are system views which return server state information useful for monitoring the health of a server instance, diagnosing problems and tuning performance.

The amount of information shown is huge (several thousands of metrics), but only a few have been useful for the performance analyses I carried out.

Other metrics Most metrics are related to advanced aspects (e.g. the number of locks performed per second or the number of pages for the whole database) which have proved not to be useful for the performance analyses.

These metrics are indeed used for an in-depth analysis or for advanced tuning of the Data Warehouse.

In my case, however, I wanted to show the difference in query execution times, since it is the most relevant aspect for Axpo employees.

8.1.1.1 Metrics Analyzed

One of the most important metric analyzed is the execution time of each query submitted to the database.

This value can be retrieved by querying the DMV `sys.dm_exec_query_stats` [20].

A lot of information are available, such as:

- The query itself.
- The execution plan chosen for the query.
- How many times the query has been executed.
- Execution time.
- Physical / Logical reads performed.
- Physical / Logical writes performed.
- Time used to compile the execution plan.
- Memory used to compile the execution plan.
- Degree of parallelism (*DOP*).

The only information I decided to analyze is the execution time, since the other were too much specific and low-level.

Values available There are four different values related to execution time:

- Last elapsed time.
- Total elapsed time.
- Last worker time.
- Total worker time.

The first two refer to the total time used for executing the query, while the other are specific to CPU usage.

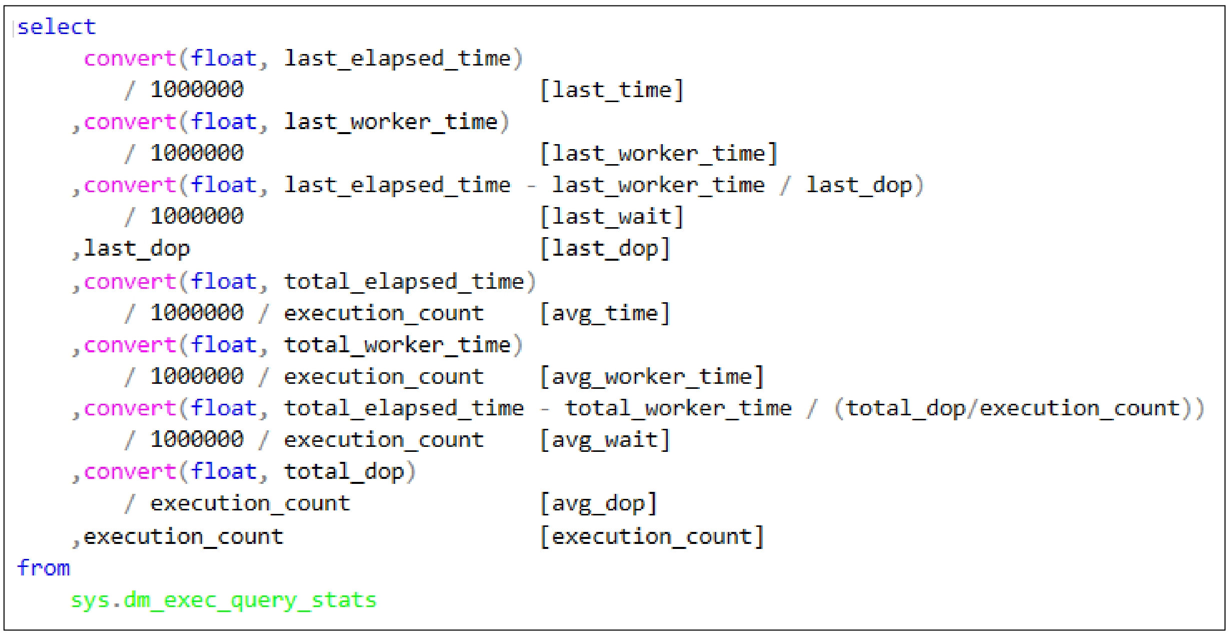
Both these values are important, since we can compare CPU time to the total elapsed time to understand the impact of locks.

Complications There are however a few complications in this computation.

First of all, if the query has been parallelized (DOP $>$ 1) worker times are aggregated [21]. As a consequence, it is necessary to divide this value by the DOP.

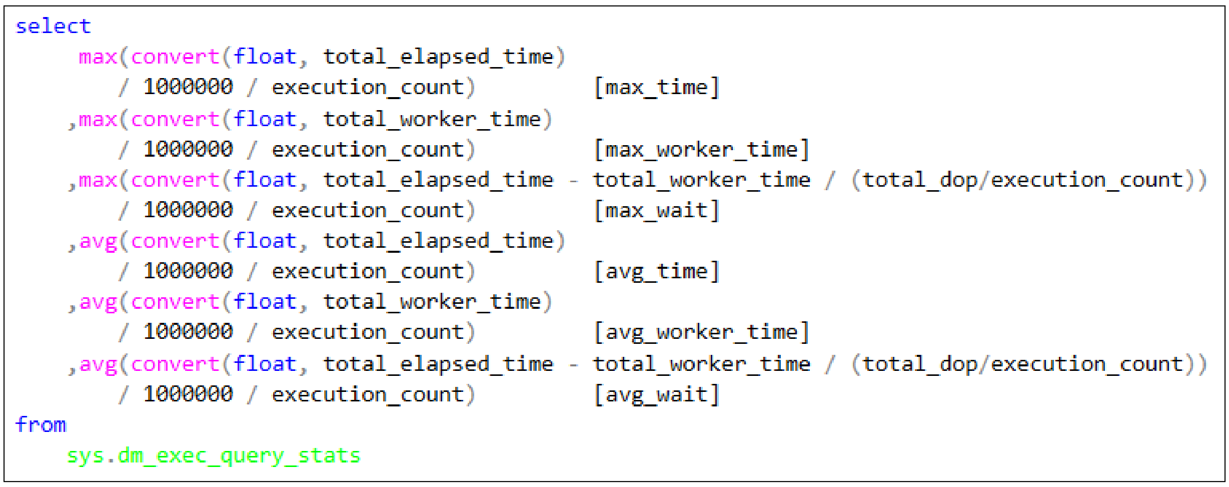
Secondly, all total values need to be divided by the execution count, since we need to compare the execution times independently of how many times a query has been called. These values are however stored as integers (without any decimal digits), so they all need to be converted to a decimal type, otherwise we would lose information about decimal digits.

Lastly, values are expressed in microseconds (even though they are accurate up to milliseconds [20]). These values have been converted to seconds to provide better readability.



```
select
     convert(float, last_elapsed_time)
        / 1000000                       [last_time]
    ,convert(float, last_worker_time)
        / 1000000                       [last_worker_time]
    ,convert(float, last_elapsed_time - last_worker_time / last_dop)
        / 1000000                       [last_wait]
    ,last_dop                           [last_dop]
    ,convert(float, total_elapsed_time)
        / 1000000 / execution_count     [avg_time]
    ,convert(float, total_worker_time)
        / 1000000 / execution_count     [avg_worker_time]
    ,convert(float, total_elapsed_time - total_worker_time / (total_dop/execution_count))
        / 1000000 / execution_count     [avg_wait]
    ,convert(float, total_dop)
        / execution_count               [avg_dop]
    ,execution_count                    [execution_count]
from
    sys.dm_exec_query_stats
```

(a) Detailed information.



```
select
     max(convert(float, total_elapsed_time)
        / 1000000 / execution_count)         [max_time]
    ,max(convert(float, total_worker_time)
        / 1000000 / execution_count)         [max_worker_time]
    ,max(convert(float, total_elapsed_time - total_worker_time / (total_dop/execution_count))
        / 1000000 / execution_count)         [max_wait]
    ,avg(convert(float, total_elapsed_time)
        / 1000000 / execution_count)         [avg_time]
    ,avg(convert(float, total_worker_time)
        / 1000000 / execution_count)         [avg_worker_time]
    ,avg(convert(float, total_elapsed_time - total_worker_time / (total_dop/execution_count))
        / 1000000 / execution_count)         [avg_wait]
from
    sys.dm_exec_query_stats
```

(b) Average and max metrics.

Figure 8.1: Queries used for analyzing query execution times.

Query Two different queries have been used, as shown in figures 8.1a and 8.1b. The first query produces a detailed output of the execution of each query, while the second shows average and peak execution times across all queries.

Aggregate information Aggregate information give a quick but effective overview of the database.

Maximum values show the worst case scenarios, while average values are representative of the database performance. We can also understand if there is a problem with locking by comparing elapsed and waiting times. If the two values are close, it means that most time was spent waiting.

Detailed information Detailed information allow to analyze each query independently.

By ordering the results in descending order, we can easily find how many queries present problems (such as high wait times) or are computationally heavy (which could mean they are not optimized). By showing an additional field, called `sql_handle`, a unique identifier for that query can be obtained. By joining this value with other DMVs, it is possible to perform additional analyses on that specific query.

Last execution times are also shown, since they could differ from average times. A large difference can imply there are moments in which the database is under a heavy load and performs poorly.

8.1.2 Queries

Several generic queries have also been executed on both the Data Warehouse and the on-premise databases. These queries present different properties, and are representative of the actual workload.

The execution times of these queries have been recorded and compared.

8.1.2.1 Query types

Specific queries have been developed to test different usage scenarios.

These queries are:

Count This query needs to count how many rows are in a table. It requires a scan of the whole table, but access to no values.

Generic selection This query is expected to perform a sequential scan of the table and return all values. The speed of this query depends solely on the database power.

Low selectivity This query has a simple `WHERE` clause, which is expected to retrieve around 50% of the data.

High selectivity This query has a more complex `WHERE` clause ranging over multiple attributes. It is expected to retrieve a small amount of data.

Ordering This query retrieves a medium-sized dataset and is tasked with sorting it according to a given value.

Aggregation This test divides the data into groups and computes an aggregated value for each group.

Duplicate removal This query selects a large number of rows and tries to remove all duplicate from multiple columns.

Insertion / Deletion Two similar queries insert a medium amount of rows in a table and then remove them. Times are recorded independently for insertion and deletion.

8.1.2.2 Table types

Azure SQL Data Warehouse supports three different table types, offering different indexing options [22].

Tests have been carried out on all three types, to serve both as a comparison between different options and to give a more precise estimation of the improvements provided by the Data Warehouse.

Heap tables Heap tables are optimized for insertion and deletion operations.

Rows are stored in the order in which they are inserted into the table, although the Database Engine can move data around in the heap to increase the storage efficiency [23].

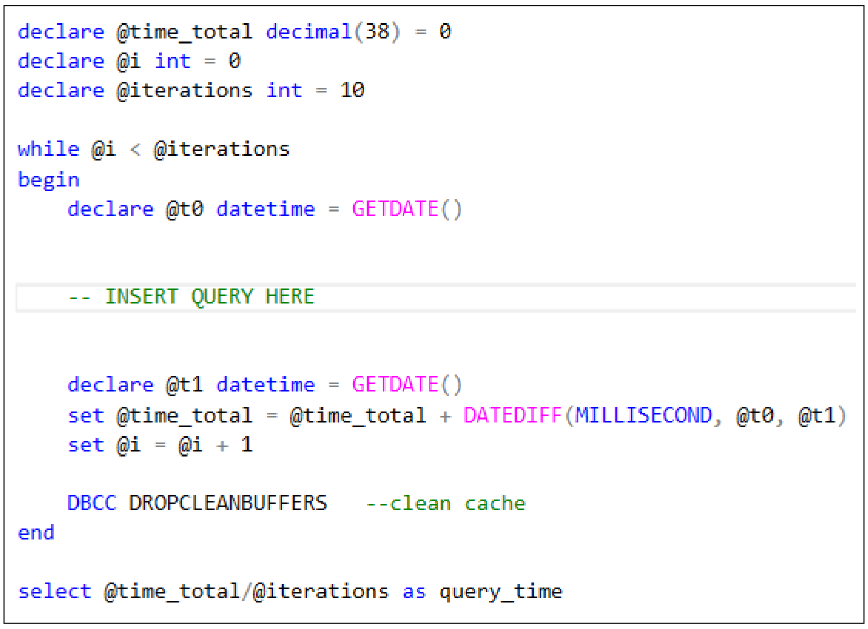
As a consequence, the data order cannot be predicted. Clustering indexes cannot be applied to heap tables.

Column-store index tables Column-store tables store data as columns, as opposed to the traditional row storage [24].

This storage type allows very fast access to multiple rows of a single column, since they are stored in the same memory block. Additionally, only the selected columns are fetched, avoiding retrieval of unnecessary data.

Column-store tables can be either clustered or non-clustered. The former allows specifying only a single clustering column.

Clustered index tables Clustered tables store data in sorted order, depending on the clustering index chosen. They are very efficient for retrieving sorted data and for performing selection operations on the cluster index.



```sql
declare @time_total decimal(38) = 0
declare @i int = 0
declare @iterations int = 10

while @i < @iterations
begin
    declare @t0 datetime = GETDATE()


    -- INSERT QUERY HERE


    declare @t1 datetime = GETDATE()
    set @time_total = @time_total + DATEDIFF(MILLISECOND, @t0, @t1)
    set @i = @i + 1

    DBCC DROPCLEANBUFFERS   --clean cache
end

select @time_total/@iterations as query_time
```

Figure 8.2: SQL wrapper for collecting accurate query execution time.

On the other hand, sorting operation of different attributes may be penalized, compared to other table type. Insertion and deletion operation need to update the clustering index each time, creating additional overhead.

8.1.2.3 Setup

The queries have been executed on both databases on identical tables, having the both same structure and data.

Measurements A small SQL wrapper for the query, shown in Figure 8.2, has been created in order to collect accurate measurements.

This wrapper records both the starting and ending time of the query, from which the actual execution time can be computed.

Multiple executions of the same queries are performed in order to improve result accuracy. The resulting execution time is the average of all the times obtained.

Caching Databases can cache query results to speed up successive executions of the same query.

Although caching improves query performance, it can produce misleading results in our tests, since we are going to execute the same query many times in a row.

It is impossible to disable result caching for a specific query, but it is possible to clean up the whole cache, preventing it from invalidating test results.

Database	Max elapsed	Max CPU	Max wait	Avg elapsed	Avg CPU	Avg wait
On-premise	3147.153	1108.736	3130.507	4.789	2.602	4.379
Cloud	150.743	48.563	126.462	0.342	0.115	0.269
Proportion	4.79%	4.38%	4.04%	7.14%	4.43%	6.13%

Table 8.1: Aggregated query execution times metrics.

8.2 Test Results

8.2.1 Metrics

This kind of tests has been performed on both the main on-premise database and the cloud Data Warehouse.

8.2.1.1 Aggregated metrics

From the results of this test, shown in Table 8.1, we can see that the cloud Data Warehouse can produce, on average, query results in 7.14% of the time required by the on-premise database (more than 12 times faster). Wait times are a bit smaller on the Data Warehouse (keeping in mind the proportion), but are not considerably reduced.

More interestingly, the highest execution times recorded are smaller on the Data Warehouse by a large amount (around than 20 times lower). Average CPU times have similar proportions considering both the highest and the average values recorded.

A more detailed analysis is however required before coming up to some conclusions.

8.2.1.2 Detailed metrics

The metrics analyzed are shown in Tables 8.2 and 8.3. These tables contain only the ten queries with the highest total execution time. A larger number of queries, however, has been analyzed, even though they are not listed for readability purposes.

One of the first things which can be noticed is that the last execution times on the on-premise database are often different from the average values (as expected, since they depend on the current database usage), while on the Data Warehouse the last execution times are, in most cases, identical to the average values.

Execution Count Upon analyzing the execution count for each query, it was noticed that on the Data Warehouse most queries had been executed just once. As a consequence, the average query execution count for both databases have been computed for both databases.

The results showed that the Data Warehouse had not been used enough yet: each query had been executed on average 9 times, compared to an average of 5129 in the on-premise database.

Data Warehouse Usage Moreover, I knew that only a restricted subset of Axpo users where actively using the Data Warehouse, since the migration was still in development and the data needed by the other users had not been migrated yet. This smaller number of users meant that the average workload of the Data Warehouse was lower than in the on-premise database.

Having a smaller workload, meant that the Data Warehouse could use more resources to compute the query results.

As a result, we can conclude that these metrics can only provide a generic overview of the Data Warehouse performance at the current time.

8.2.1.3 Problems

Tests on metrics, although properly structured, proved not to be representative of the Data Warehouse.

Workload The large difference in active usage caused the data to be too much influenced by noise, in the form of different workloads between the two databases.

On the one hand, the local database was actively used by a large number of users, which meant that the database had to deal with a high amount of queries at the same time.

On the other hand, the Data Warehouse was used by a restricted number of users, leaving more resources available for computing the results.

Different queries These metrics analyzed the execution times of all the queries executed on each database. It is important to keep in mind that the queries analyzed are not the same however. It is very likely that a consistent number of complex and computationally demanding queries has been run only in the on-premise database, since the Data Warehouse is newer and still incomplete.

Some users, who usually execute complex queries, still haven't had the possibility of computing on the cloud, since the data they need hasn't been migrated yet. These queries will very likely increase the metrics obtained from the Data Warehouse.

8.2.2 Queries

8.2.2.1 Setup

This test has been performed on actual data used by Axpo. The view selected contains forecasts from multiple providers related to multiple categories. It has been chosen since it is one of the largest and slowest tables used by the company. The view contains 106 million rows.

Materialization Executing tests on a view would have produce inaccurate results, since multiple operations (mostly joins with dimensional table) are performed each time. To avoid this problem, the data has been copied to a separate table, created appositely for testing.

Last execution			Average values			
Total	CPU	Wait	Total	CPU	Wait	DOP
6839.531	224.368	6820.834	3131.482	197.520	3115.022	12
718.741	163.319	555.421	718.741	163.319	555.421	1
484.972	99.821	385.151	484.972	99.821	385.151	1
272.092	186.531	256.547	258.351	165.731	244.540	12
210.750	94.374	202.885	210.750	94.374	202.885	12
266.322	64.247	202.074	266.322	64.247	202.074	1
247.747	52.608	195.138	247.747	52.608	195.138	1
250.279	466.570	211.398	221.347	425.721	185.870	12
183.070	42.352	140.717	183.070	42.352	140.717	1
148.721	120.985	138.639	111.562	108.767	102.498	12

Table 8.2: Detailed metrics for on-premise database.

Last execution			Average values			
Total	CPU	Wait	Total	CPU	Wait	DOP
150.742	48.562	126.461	150.742	48.562	126.461	2
140.425	45.220	117.814	140.425	45.220	117.814	2
134.712	35.646	116.889	134.712	35.646	116.889	2
134.385	46.860	110.955	134.385	46.860	110.955	2
120.121	37.765	101.238	120.121	37.765	101.238	2
117.977	36.492	99.731	117.977	36.492	99.731	2
114.978	34.205	97.875	114.978	34.205	97.875	2
116.704	38.060	97.674	116.704	38.060	97.674	2
115.938	37.497	97.189	115.938	37.497	97.189	2
116.585	39.543	96.813	116.585	39.543	96.813	2

Table 8.3: Detailed metrics for Data Warehouse.

In this way, we also made sure no one else would use this table while the tests were being executed.

Excessive table size The size of the table was still however excessive for some tests; the two main problems encountered were:

- Excessive computational times
- Not enough RAM to store the results

The first issue prevented running an appropriate number of tests, since a single query execution could take more than half a day. It was necessary to perform multiple runs of each query to obtain more reliable results, so each test would have taken several days.

The second problem was related to the PC used for executing the tests: asking to retrieve a huge number of rows would have required a very large amount of RAM on the machine to temporarily store the results. Since the machine was limited to 8GB RAM, it often encountered *Out of Memory* errors, which prevented the tests from completing.

As a consequence, I decided to leave only the most recent 10 million records, and perform the tests on this smaller set. Since all data in the original table is updated daily, taking the last n days leaves all the value distributions intact.

8.2.2.2 Results

The execution times measured from the queries mostly confirmed the results obtained from the metrics.

The times obtained are shown in Figure 8.3.

Figure 8.4 shows the execution time ratio between the Data Warehouse and the on-premise database, for each query type.

Initial considerations As we can see, the Data Warehouse performs better than the on-premise database on almost all test cases.

Count operations, as well as insertions and deletions can be computed in a very short amount of time, especially when compared to the on-premise database.

Selection operations, on the other hand, are directly dependant on the selectivity degree of the query. A very selective query is very performing, while a completely non-selective query (such as `SELECT * FROM <table>`) is even outperformed by the on-premise database.

All the other operations are generally more performing than the on-premise database, but their performance is strictly related to the table type.

Heap tables Heap tables present the lowest execution time for deletion operations, and a very low time for insertions. This is because of the structure of heap tables, which don't need to update any index when or removing data from a table.

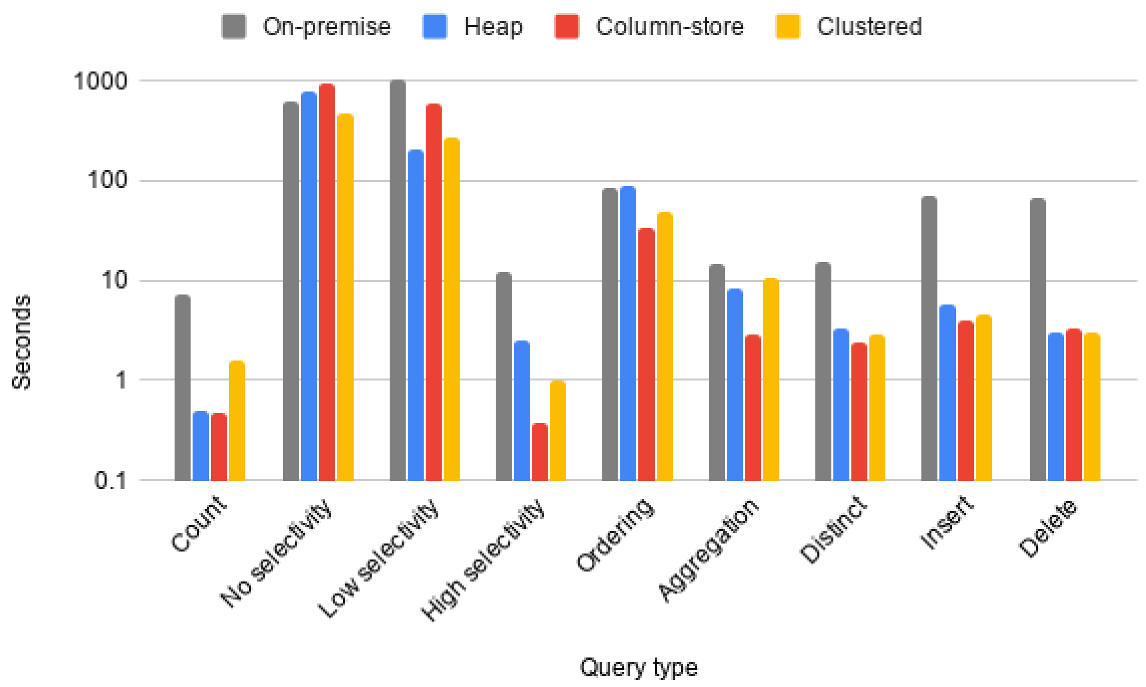


Figure 8.3: Query execution times for both databases. Different table options have been tested.

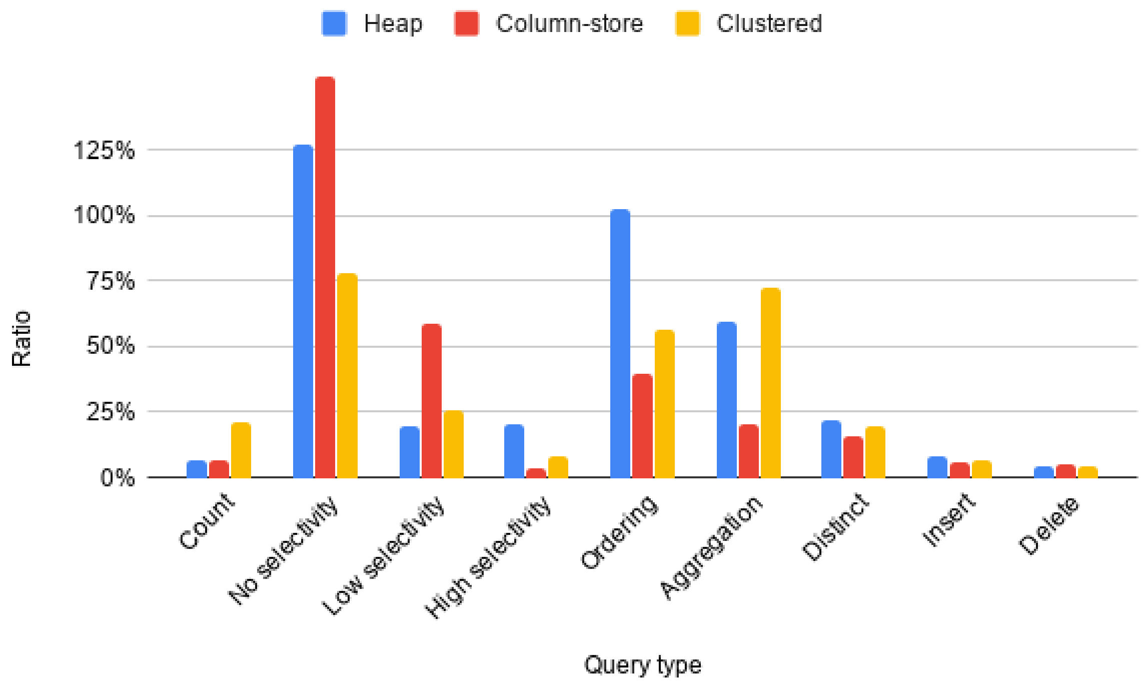


Figure 8.4: Execution time ratio between Data Warehouse and on-premise database, for each query type.

Ordering operations, however, perform poorly, since this table structure stores the values in unsorted order. As a consequence, each page needs to be accessed multiple times to sort all values.

Column-store tables Column-store tables are the fastest option for high-selectivity queries, as well as aggregation and duplicate removal operations.

Since data is stored in column format, selection operations on specific attributes can be performed directly on the column, without retrieving unneeded data.

The same principle applies to aggregation and duplicate removal: the columns necessary for the operation are stored in the same memory blocks, decreasing the amount of read operations needed to compute the result.

On the other hand, low-selectivity queries are penalized, since the particular column structure created an additional reading overhead when multiple columns are required for a lot of records.

Clustered tables Clustered tables offer the best performance overall, although they are very dependant on the workload.

For instance, we can notice that the aggregation operation, which has not been performed on the clustering index, performs poorly.

As a consequence, this table is a very good choice if most operations have conditions on the same attributes.

8.2.2.3 Considerations

From the results, it is clear that the Data Warehouse offers higher performances than the on-premise database.

Insertion and deletion operations are very efficient on all table formats, while other operations are dependent on the attributes present in the query, as well as the table format chosen.

There is, however, no general solution. As we have shown, for example, a column-store table can be very efficient on operations requiring access to a small number of attributes, but can be outperformed even by the on-premise database if a large number of columns and rows is requested.

As a consequence, it is always important to keep in mind the workload when creating a table. Choosing the wrong option can worsen performance, in the worst cases making the Data Warehouse even slower than a local database.

On the other hand, if the right choices are made, users can notice a large improvement in execution speed, up to 33 times in the best scenario (a high-selectivity query on a column-store table).

Chapter 9

Conclusions

Moving from a complex structure of on-premise databases and tools is not an easy task. Many problems are encountered in each phase of the development, starting from understanding the various data sources needed to developing an appropriate structure.

First of all, finding a suitable **workflow** is not an easy matter. As it has been described, the original workflow presented by Reply had several issues and limitations. A part of these problems have been solved by making some changes to the process, but a few still remain. For example, the history retrieval process is now tested more in depth before deployment of a Sprint.

The **ETL process** also encountered many difficulties, caused by different factors. Each provider, for example, exposes data in different formats and requires specific data normalization operations.

Testing the **quality of the data** is also a complex matter, since energy market data present domain-specific difficulties. For example, it is impossible to automatically assess the correctness of forecast values, since they are expected to change each time.

Even considering all these problems, we can see a great improvement in both **simplicity** and **performance**.

The architecture has become more simple, since now all the data is stored on a centralized Data Warehouse, instead of multiple databases, each with its own insertion logic. As a consequence, most of the integration software previously used are no longer required.

Performance is, however, the most important aspect, since it was the main cause for the migration. From the test results, we can notice a consistent reduction in query execution times, as long as the appropriate table structures are chosen.

We can conclude by stating that migrating from an on-premise database to a Cloud-based architecture is a good choice, especially in terms of expected performance, although the process is very complex.

Appendices

Appendix A

GME

The Italian Power Exchange (IPEX) is the exchange for electricity and natural gas spot trading in Italy. It is managed by *Gestore del Mercato Elettrico* or GME [25].

IPEX comprises the following spot markets:

- Day-ahead Market
- Intra-day Market
- Ancillary Services Market

Day-ahead Market The Day-ahead Market, or *Mercato Giorno Prima* (MGP), hosts most of the electricity sale and purchase transactions.

Energy blocks, at hourly granularity, are traded for the next day. The trading session starts 9 days before the day of delivery and closes at 12 p.m. the day before the day of delivery, hence the name "day-ahead market".

Bids are valued at the National Single Price, or *Prezzo Unico Nazionale* (PUN), which is the average of the prices of geographical zones, weighted by the quantities purchased in these zones[1].

In the case some constraints are violated, zonal prices are used instead[2].

Intra-day Market The Intra-Day Market, or *Mercato Infragiornaliero* (MI), allows Market Participants to modify the schedules defined in the MGP by submitting additional supply offers or demand bids.

Sales are performed similarly to MGP, while purchases are executed exclusively at zonal prices.

The MI is comprised of seven sessions, named MI1 to MI7.

[1]PUN: $\sum_{i=1}^{n\_zones} \frac{consumption_i * price_i}{consumption_i}$

[2]Most constraints are physical limitations, e.g. a cable line cannot physically transfer more than a certain amount of electricity. If it is necessary to transfer more than this amount, it is necessary to use more than one line, which is usually located in a different zone (hence the need to use zonal prices).

Ancillary Services Market The Ancillary Services Market, or *Mercato Servizi di Dispacciamento* (MSD), is the venue where Terna S.p.A. procures the resources it requires for managing and monitoring RTN[3].

Terna acts as a central counterparty and accepted offers are remunerated at the price offered (*pay-as-bid*).

The MSD is comprised of two submarkets, called ex-ante MSD and Balancing Market, or *Mercato Bilanciamento* (MB). Both market are comprised of 6 sessions, named MSD1 to MSD6 and MB1 to MB6.

[3] *Rete Trasmissione Nazionale* (National Transmission Network). It covers almost all the energy transmission network in Italy. Around 98% of RTN is managed by Terna.

Bibliography

[1] Axpo Group. *Axpo Portrait (IT)*. URL: https://www.axpo.com/axpo/it/en/about-us/portrait.html.

[2] Axpo Group. *Axpo Portrait (CH)*. URL: https://www.axpo.com/axpo/ch/en/about-us/portrait.html.

[3] Ivan Mistrik et al. *Software Architecture for Big Data and the Cloud*. 2017. URL: https://books.google.it/books?hl=en&lr=&id=zvPtDQAAQBAJ&oi=fnd&pg=PP1&dq=+datawarehouse+cloud+migration+process.

[4] Mahdi Fahmideh Gholami et al. "Cloud migration process - A survey, evaluation framework, and open challenges". In: *Journal of Systems and Software* 120 (2016), pp. 31–69. URL: https://www.sciencedirect.com/science/article/pii/S0164121216300966.

[5] DZone. *Cloud Migration: How And Why Business Is Moving to the Cloud*. URL: https://dzone.com/articles/cloud-migration-how-and-why-business-is-moving-to.

[6] New Relic. *Cloud Migration Checklist: 10 Steps You Need to Follow*. URL: https://blog.newrelic.com/engineering/cloud-migration-checklist/.

[7] Codoid. *https://codoid.com/data-quality-checks-data-warehouse-etl/*. URL: https://codoid.com/data-quality-checks-data-warehouse-etl/.

[8] Codoid. *ETL Data Quality Testing Best Practices — Codoid — Best Testing Company*. URL: https://codoid.com/etl-data-quality-testing-best-practices/.

[9] Wikipedia. *Microsoft Azure*. URL: https://en.wikipedia.org/wiki/Microsoft_Azure.

[10] Microsoft documentation. *What is Azure Databricks*. URL: https://docs.microsoft.com/en-us/azure/azure-databricks/what-is-azure-databricks.

[11] Microsoft documentation. *Introduction to Azure Blob storage*. URL: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction.

[12] Microsoft documentation. *Microsoft SQL Data Warehouse - MPP architecture*. URL: https://docs.microsoft.com/en-us/azure/sql-data-warehouse/massively-parallel-processing-mpp-architecture.

[13] Microsoft documentation. *Memory and concurrency limits for Azure SQL Data Warehouse*. URL: https://docs.microsoft.com/en-us/azure/sql-data-warehouse/memory-and-concurrency-limits#gen2.

[14] Microsoft documentation. *What is Polybase?* URL: https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-guide?view=sql-server-ver15.

[15] Microsoft documentation. *Introduction to Azure Data Factory.* URL: https://docs.microsoft.com/en-us/azure/data-factory/introduction.

[16] NetIQ. *Understanding Cron Syntax in the Job Scheduler.* URL: https://www.netiq.com/documentation/cloud-manager-2-5/ncm-reference/data/bexyssf.html.

[17] Wikipedia. *List of ISO 3166 country codes.* URL: https://en.wikipedia.org/wiki/List_of_ISO_3166_country_codes#Current_ISO_3166_country_codes.

[18] ENTSO-E. *Energy Identification Codes (EICs).* URL: https://www.entsoe.eu/data/energy-identification-codes-eic/.

[19] Microsoft documentation. *Using decimal, float, and real Data.* URL: https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms187912(v=sql.105)#using-float-and-real-data.

[20] Microsoft documentation. *sys.dm_exec_query_stats.* URL: https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-query-stats-transact-sql?view=sql-server-2017.

[21] DBA StackExchange. *What is difference between last_worker_time and last_elapsed_time in DMV sys.dm_exec_query_stats?* URL: https://dba.stackexchange.com/questions/31002/what-is-difference-between-last-worker-time-and-last-elapsed-time-in-dmv-sys-dm.

[22] Microsoft documentation. *Indexing tables in SQL Data Warehouse.* URL: https://docs.microsoft.com/en-us/azure/sql-data-warehouse/sql-data-warehouse-tables-index.

[23] Microsoft documentation. *Heaps (Tables without Clustered Indexes).* URL: https://docs.microsoft.com/en-us/sql/relational-databases/indexes/heaps-tables-without-clustered-indexes?view=sql-server-2017.

[24] Microsoft documentation. *Clustered Columnstore Index in Azure SQL Database.* URL: https://azure.microsoft.com/es-es/blog/clustered-columnstore-index-in-azure-sql-database/.

[25] Gestore Mercato Energetico. *Mercato elettrico a pronti.* URL: http://www.mercatoelettrico.org/it/Mercati/MercatoElettrico/MPE.aspx.