POLITECNICO DI TORINO

Master's Degree in ICT for smart societies

Master's Degree Thesis

A new normalization data platform on Oracle ADB using a Google service

Supervisors

Prof. PAOLO GARZA

ALESSANDRO DUGO

Candidate

MATTEO BOGONI

December 2023

Abstract

The goal of this thesis is to propose a new alternative to the current normalization software, Trillium, used inside the database of an automotive company. The idea is to develop a new product that is able to maintain the same quality standard and that can be sold separately to the entire flow of the database.
To do that, the new available technologies on the market were considered: for the normalization component it was used the Google Address Validation, a tool provided by Google where by passing an address, it is returned normalized. All the system is developed on the Oracle cloud using the Autonomous database, a fully automated service based on the AI.

Acknowledgements

Vorrei ringraziare per primo il mio relatore Paolo Garza, che mi ha seguito nella stesura di questo elaborato rendendosi sempre disponibile per ogni chiarimento.
Inoltre ringrazio Alessandro Dugo per avermi dato l'opportunità di svolgere la tesi all'interno dell'azienda e di avermi inserito all'interno del progetto del Central Customer Database. A questo punto vorrei anche ringraziare tutti i miei colleghi all'interno del team che mi hanno accolto ed aiutato durante il mio percorso. In particolare vorrei ringraziare Marcello che mi ha seguito dall'inizio e mi ha spiegato tutto, dalla A alla Z.
A questo punto vorrei ringarziare i miei genitori che mi hanno sempre sostenuto e mi hanno dato l'opportunità di affrontare questi cinque anni e diventare la persona che sono. Una dedica anche a mio fratello Pietro con cui ho convissuto per più di 24 anni e con cui ho un legame speciale.
Un ringraziamento anche a tutti i miei nonni, zii e cugini che fanno parte della mia famiglia e a cui sono molto legato.
Finendo la magistrale è doveroso ringraziare tutti i miei colleghi di corso che oltre ad essere tali sono diventati anche amici con cui, oltre a subire insieme le pene degli esami e scadenze dei progetti, condivido anche momenti di divertimento al di fuori del Politecnico. In particolare un ringraziamento al team del ping pong con il quale ho passato più tempo giocare che a lezione.
Colgo l'occasione per ringraziare anche tutti gli amici conosciuti i tre anni precedenti che mi hanno accompangato in questo viaggio e con i quali condivido bellissimi momenti.
Un ringrziamento poi a tutti gli amici che ho avuto e che ho, da chi ho conosciuto

più recentemente a chi conosco da sempre, per essermi sempre stati accanto ed aver reso questi anni indimenticabili.
Infine per gli amici veri, quelli che ci sono sempre stati e su cui posso sempre contare, con cui ho passato mille avventure e con cui ne passerò altrettante. Grazie di tutto!

'Per tutta la gente che c'è dall'inizio
E per tutti quelli che ho perso lungo il cammino
È stato un lungo viaggio'
∞ *Love by Guè*

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the digital age, where data collection and management constitute a fundamental pillar for every sector, the importance of integrity and precision in information emerges as a crucial element. This context highlights the key role of databases, which form the foundation for the creation, preservation, and analysis of essential data critical for the effective functioning of various industries and services.
Within this informational landscape, one of the critical aspects in data management pertains to the need for ensuring a uniform and accurate representation of postal addresses within databases. Addresses, considered vital elements for identification and localization, play a central role in a wide range of applications, from e-commerce to healthcare and the management of public resources.
This thesis, conducted in collaboration with the consulting firm "Technology Reply", aims to thoroughly explore the issue of address normalization within an extensive database of a significant automotive company. Address normalization, a complex and crucial process, aims to achieve uniformity in data, reducing ambiguities, and improving the overall quality of stored information. Focusing on this theme is imperative, as associating a postal address or comprehensive contact information with a user allows for fully harnessing the potential of databases, contributing significantly to the improvement of operational efficiency and analytical precision.
Throughout this work, an initial examination of the structure and overall functioning of the database in question is undertaken, providing a detailed overview of the

various phases and processes that guide an address from its entry into the database to the actual storage phase. A deeper analysis will be dedicated to the normalization phase, highlighting its execution and the software involved.
Subsequently, a concrete challenge encountered in the use of external software for normalization is addressed, emphasizing the need to implement a new solution. The related tests and regression analyses necessary to ensure the proper functioning of the process will also be presented.
Finally, through the analysis of case studies and exploration of services available on the market, a new solution for the data normalization phase based on emerging technologies is proposed. This solution will be subject to a comparison evaluating its effectiveness, with the prospect of potentially replacing the current software in the future. Despite this possibility, the primary objective remains the search for optimal solutions to enhance data integrity and quality.

1.1 State of the art

The address data normalization is an important topics which has been dealt with at length in recent years. An interesting solution has been presented in the paper titled "A Hybrid Approach to Address Normalization" by Wing Shing Wong (Chinese University of Hong Kong) and Mooi Choo Chuah (AT&T Bell Laboratories)[1], where they explores three distinct approaches in the landscape of address normalization. The first involves the creation of an extensive dictionary containing all conceivable combinations of address elements, leading to a resource-intensive and slow normalization process. The second approach adopts a rule-based system that extracts street name, city name, and ZIP code based on keywords or word positions. However, this method faces challenges in terms of rule modification and updates, potentially causing conflicts. The third and innovative approach involves a learning system that is initially trained on a subset of addresses, demonstrating continuous self-learning similar to neural network systems. The contribution to this domain is a hybrid system, incorporating the third approach along with spelling correction, aiming to normalize mailing addresses accurately and efficiently. The system integrates an address dictionary and a scoring system, that is inspired by

analogue neural network. The system's key processes include learning, further divided into dictionary creation/updating and system parameter training. During the learning phase, the system is trained to recognize a set of addresses, leading to the construction and updating of an address dictionary. This dictionary classifies grouped address components, referred to as tokens, assigning weights based on their frequency in the learning set. In the normalization of a new address, the system consults the address dictionary to recognize various components, and in cases of ambiguity, a scoring function resolves multiple meanings.
Another method for the address standardization is outlined in the technical report called 'Address Standardization' [2], with a specific focus on the USC WebGIS Open Source Geocoding Platform. Address standardization involves various techniques and is integral to the address data cleaning process. The report delves into different levels of address standardization and its implementation within the USC WebGIS platform, emphasizing its significance in creating high-quality spatial data for subsequent analyses.
The document details the procedures involved in address data cleaning, including parsing, normalization, and other essential processes. Address validation and normalization, crucial components of the address cleaning process, are explored extensively.

In order to provide a clear and cohesive organizational structure, the following chapters will address specific key aspects of this topic, offering a comprehensive overview of the normalization phase within a database.
Chapter 2 will focus on a detailed analysis of the functioning of the database in question, examining each stage through which data passes before being permanently stored. Special attention will be given to the normalization phase, providing an in-depth explanation of its operation and the external software on which it is based.
In Chapter 3, real issues related to the normalization phase will be addressed, presenting tangible solutions. Targeted tests will be conducted to ensure the effectiveness of the changes implemented.
Chapter 4 will introduce a new proposal for address normalization based on emerging

technologies, evaluating its effectiveness in comparison to the current methodology. Subsequently, future developments will be suggested to enhance the efficiency of the new solution.
Finally, Chapter 5 will conclude the thesis by examining the work undertaken and proposing potential practical applications for the proposed solution.

Chapter 2

Control Customer Database structure

2.1 Introduction

In the automotive sector, "Technology Reply" has developed the Central Customer Database, also known as CCDB, for a renowned multinational company. This cutting-edge platform is designed for the acquisition, consolidation, and utilization of information derived from user interactions with the company, its products, and services, such as automobiles and maintenance, over time and in diverse ways.
The CCDB aims to address various critical needs, such as evaluating the performance of the sales network, providing an indicator of the effectiveness of undertaken marketing actions, and expediting network contact actions for those expressing interest.
Operating in Europe, South America, the Middle East, and some countries in Africa, the CCDB can process, analyze, recognize, control, enrich, and integrate data from various sources while respecting the specificity of each country. Given its multi-market nature, the process is handled uniquely for each market due to potential differences in formats, codifications, and specificity. In this thesis, all proposed activities are focused on the market '1000', identifying the market in Italy.

Among the CCDB's key functionalities there are data input control and standardization, assignment of uniqueness to the subject through deduplication algorithms, definition of data quality KPIs, nominal name search, and corporate management of personal data and data retention processes.
The CCDB has proven to be the primary source of information for operational marketing activities, post-sales support, and surveys. The resulting benefits include increased efficiency and effectiveness of processes based on this information, performance analysis of processes, immediate actions triggered by collected information, and a detailed view of the customer's history during interactions.

CCDB environments

It is crucial to delve into the 'internal' structure and management of the CCDB. There are two distinct development environments, namely Certification and Production, which broadly contain the same elements, the same files, tables, etc. The substantial difference lies in the fact that the first environment is exclusively used to test new modifications and conduct tests. In contrast, the Production environment actively runs the data flow from various markets, populating what will eventually become the complete database used for analyses and subsequent decision-making. Understanding the significance of the Certification environment becomes evident, as it is the space where modifications are tested before being introduced to Production. This practice aims to prevent any disruptions to the flow and potential damages, which could significantly impact the concerned company.
It is important to note that all modifications, tests, and implementations proposed in this thesis have been carried out in the Certification environment.

2.2 Data model

As mentioned earlier, the database contains records related to both users, i.e., individuals, and products, such as automobiles, as well as services like maintenance or insurance.
As will be explained later, depending on the type of data, the record undergoes

various operations and populates different tables, only to be recompiled in the final step.

2.2.1 Master data

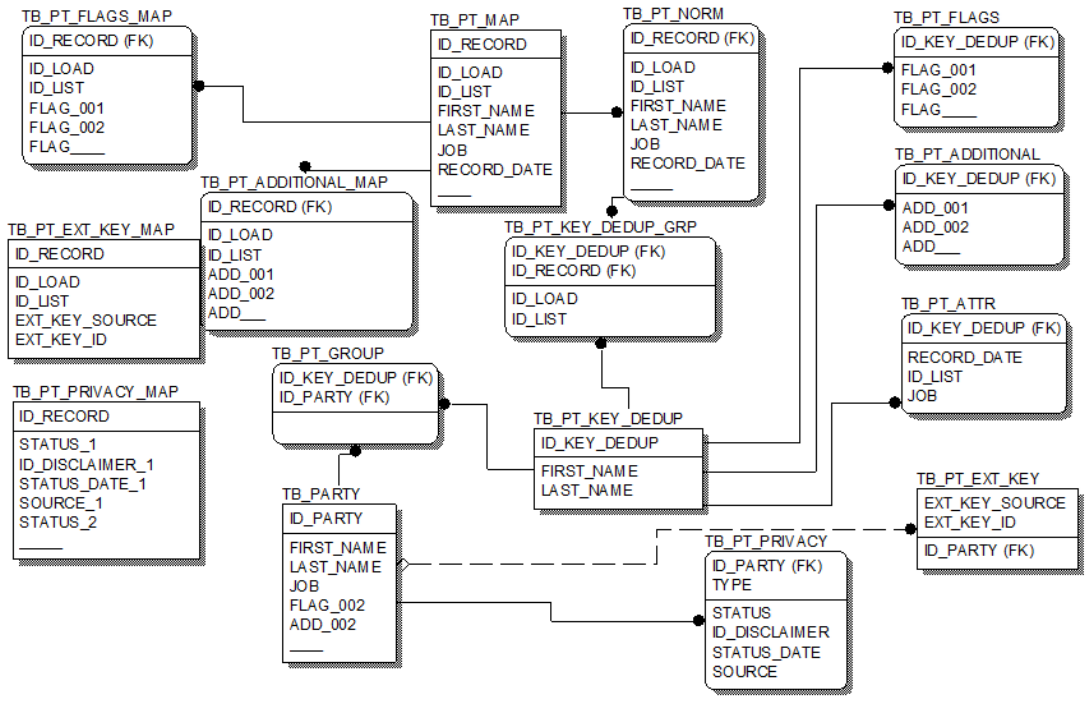

Figure 2.1: Master data tables flow

Figure 2.1 represent the entity-relationship model and it shows the main tables involved in the master data management process.

- The ***TB_PT_MAP*** table contains the master information: it should allow the storage of all the master information received from the different input systems, regardless of the source market. The *TB_PT_MAP* table contains the data as provided by the input systems, unless data deleted due to consistency and transcoding checks. The data in the table represent the input of the normalization process.

- The ***TB_PT_FLAG_MAP*** can be seen as an extension of the TB_PT_MAP table, and it is intended to contain additional flags that are not present within TB_PT_MAP but are necessary during the mapping of new input flows.
- The ***TB_PT_NORM*** table contains the master data that are subjected to normalization process. The latter process takes as input the contents of the TB_PT_MAP table while the output is stored inside the *TB_PT_NORM*. The relationship between the two tables is 1:1.
- The ***TB_PT_KEY_DEDUP*** table contains the dedup "secca" keys, so the master columns that, if they are equal in two different records, determine automatically the equality of the two records.
- The ***TB_PT_KEY_DEDUP_GRP*** tables represent grouping tables, that is, many to one relationship. In this case, the table allows for the relationship of individual master movements on the TB_PT_NORM that are subjected to dedup "secca" and then are converged to a single key in the TB_PT_DEDUP table.
- The ***TB_PT_GROUP*** table maintains the relationship between two or more dedup keys (TB_PT_KEY_DEDUP) that were found to be associated with the same subject as a result of the algorithms applied in the dedup phase. The grouping key assigned to each record group is the code that will be assigned to the golden record, that is, the unique code by which the master subject will be identified within the database.
- The ***TB_PARTY*** contains the master golden record, that is, the summary of the best of the information received as input for that specific master subject.

2.2.2 Product data

In Figure 2.2 are shown the main tables involved in the process of product data management.

- The ***TB_PARTY_CAR*** table maintain the relationship between customer and car. For products for which there is a single, non-transferable customer

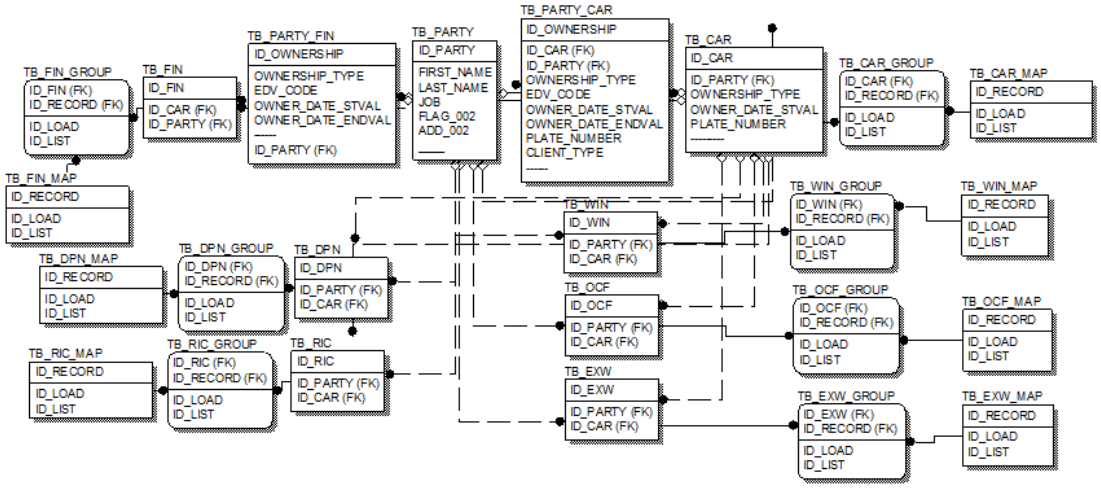

Figure 2.2: Product data tables flow

(e.g., warranty work), the associated customer is in the table itself while for products for witch is possible to have more customers, it is provided another table. Specifically, the TB_PARTY_CAR table maintains the link between a car and its owner. We may have, for the same car, the one who bought it new and all subsequent changes of ownership.

- The ***TB_OCF_RECORD_MAP*** table contains information about car orders, such as requested model, requested version, etc.

- The ***TB_CAR_RECORD_MAP*** table contains information about both new and old cars.

- The ***TB_OCF_RECORD_GROUP*** table allows to relate single orders in the TB_OCF_RECORD_MAP that are subjected to dedup phase and thus associated with a single TB_OCF table element.

- The ***TB_CAR_RECORD_GROUP*** table has a similar function to TB_OCF_RECORD_GROUP, as it relate car data of TB_CAR_RECORD_MAP that are subjected to dedup phase and thus associated with a single TB_CAR table element.

- The ***TB_OCF*** table contains the golden records of the car orders, that is, the best information received for that specific order. The table also contains the relation between the master subject matched to the order and the car to which the order relates.

- The ***TB_CAR*** table contains the golden records of the cars and also in this case it is composed by the best available information.

In general, within the CCDB, it is possible to identify three main type of tables, each of them with a specific purpose:

- the tables with the suffix "**_MAP**" represent the mapping tables, so, tables that fill the staging area during the transfer of information from the input files to the rest of the database.

- The tables with the suffix "**_GROUP**" represent the grouping tables, in which the relation is many : 1.

- The tables without any particular suffix, such as TB_OCF and TB_CAR, contains the golden record of their respective areas.

2.3 CCDB structure

In figure 2.3 is represented the main structure of CCDB system and, thus, the entire process to which an input record is subjected in the system. The process is composed by several blocks, each of which represent a specific operation. As result of these operation a table is populated with updated information and the follow block will use this table as input and it will populate another table in turn.
According to the type of data inside the record, product or master data, it follows a different branch and it is subjected to different procedures.

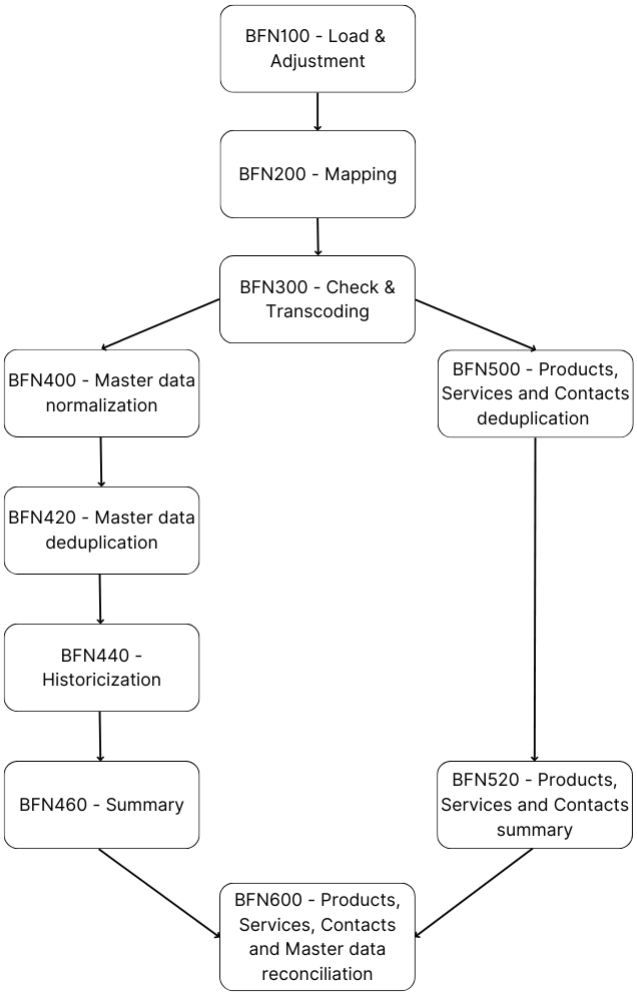

Figure 2.3: CCDB phases

2.3.1 Load & Adjustment

The *Load* phase has the function of insert inside the database the information coming from supply flows. Since the flows are not always generated for this specific task, due to the fact that they comes from different sources and they are used also in other system with different requirements, it may be necessary to adjust the format and the content to the system requirements.
The *Adjustment* operation is apt to delete possible records that are not useful for the application purposes and to delete possible abnormal fonts.

This activity can be done:

- Before the *Load* phase: in this case the operation will be done directly on the input flux and the output will consist of one or more new flux which will be subsequently processed;
- During the *Load* phase: the adjustment phase is an integral part of the Load procedure and occurs concurrently with the data loading phase in the appropriate staging areas

The Load operation represents the first step in the process of populating the database following the receipt of one or more input streams.
Each new processed movement is assigned a unique identifier (id_record) that will be carried over to all staging tables on which the movement can be distributed. Through the id_record it is then possible to relate data that are part of the same input movement.
Each Load operation that is performed, has a unique identifier (id_load) that is associated with each individual record entered on the database within the same load cycle. Thus, transactions that are part of the same load cycle will have different id_record but will be grouped under a single id_load.
Within the same Load operation, information from different sources may be loaded; data processed in the same load cycle are subject to the same processing rules, which are specific through a code provided at the start of the load cycle (Load_group code).

2.3.2 Mapping

The *Mapping* phase allows the different formats used by different input streams to be adapted to the standard structure of the information as it is managed within the CCDB system. Thanks to the Mapping component, new input streams can be integrated into the system without intervention to the software.
The procedures for loading the master and product data require that the information are present in specific tables (target_map), where the information has already been broken down on the various fields. These tables contain all the information

manageable by the system, in the expected format.
Mapping rule configuration can be multi-market or market-specific.
Specific rules for the input list need to be configured before the mapping step is executed. A key element in list configuration is defining the list type, that is, defining the characteristics of the populating operation being performed. The different types of lists are:

- Standard: This is the normal list type with which periodic uploads are made. It can contain only master data or also product data. Master data are subjected to a specific completeness check, since they must contain a minimum set of information to be accepted, and then they are subjected to normalization.

- Anagraphic recycling: In this case, it is about targeted updates to master records already in the database. Normally they are used for master data, but product data can also be associated. Data processing is similar to the processing done for standard lists.

- Master enrichment lists: are lists used to supplement what is already on the system with new punctual information, without specifying the master data in its entirety (e.g., updating phones, e-mail, etc.).

- Product recycling lists: the list only involves updating product data while master data operations are performed

- Phone / e-mail cancellation and annulment list: list allows you to annul or delete phones and e-mails of specific clients.
  In case of cancellation phone number and e-mail address will be marked as invalid and will not be used in the creation of the final record. If the numbers or e-mails are later submitted again by an input stream, the data will become valid again.
  In case of annulment phone number and e-mail address will be marked as invalid and will not be used in the creation of the final record. If the numbers or e-mails are later submitted again by an input stream, they will be considered invalid.

Once the list type is defined, the distribution of the input data to the various mapping tables (target_map) should be done. A mapping table is provided for each entity in the database.
To move a data from input record to a mapping table, it is sufficient to indicate the input record's offset, length, table name, and target field.

2.3.3 Check & Transcoding

When defining mapping rules, it is possible to include checks on the input data. If data fails to pass the check it will not be totally used or it will be limited to master or product part. The possible controls provided are:

- Obligation of a single piece of data: it is possible to require the discard of a record, or a part of it, if one or more fields are not valued.
- Format control: it is possible to request the discard of a record or a part of it, if one or more fields do not meet the required format.
- Domain control: it is possible to require validation of a field in the input record. For codes, is checked the presence into a lookup table and for date is checked the inclusion or exclusion within given period.
- Transcoding: domain check can be extended with a transcoding, in this case the check will not be performed using a lookup table but using a transcoding table. Failure the check can result in discarding the entire record (or a part of it), forcing it to null or to a default value for the specific field.

2.3.4 Normalization

The normalization phase consists of a series of operations aimed at data cleaning, a subsequent syntactic analysis, and an application of a standardization phase.
This crucial phase is carried out in part directly on the database while for some components, such as name and address, it is entrusted to an external software called Trillium
Since it is used an external software, a portion of normalization process is integrated

via shell-unix in batch mode, which takes care of extracting the information from the database, starting the normalization process, and, then, loading the normalized data.
This phase will be deepened in the following section since it is the phase most involved in the development of the thesis.

2.3.5 Deduplication

The deduplication component is aimed at parsing new input master subjects to check whether they already exist on the database and thus avoid entering the same master subject multiple times (i.e., the same person or company multiple times). Usually, data from the input streams do not have a unique key that allows simple and certain identification of the subject existence in the database. Identification is based on a certain amount of information that, when it reaches a level of similarity above a certain threshold, makes it possible to declare that two master records refer to the same person.
The deduplication operation, within the CCDB system, identifies all activities carried out aimed at identifying:

- The presence of master records referring to the same subject among the master records during the upload. Indeed, it is possible that in the same stream or set of streams that is uploaded, there are already master records referring to the same subject.
- The presence of input master records that are already in the CCDB database
- The presence of master records already on the CCDB that due to updating are found to have reached a level of similarity whereby they can be considered to belong to the same individual.

The deduplication operation then operates both on new input master records, looking for duplicates within the input stream and from the input stream to the database, and on master records already in the database and subject to update.

Figure 2.4 shows the process of searching for any duplicate master record performed inside CCDB system. It is possible to identify two possible type of

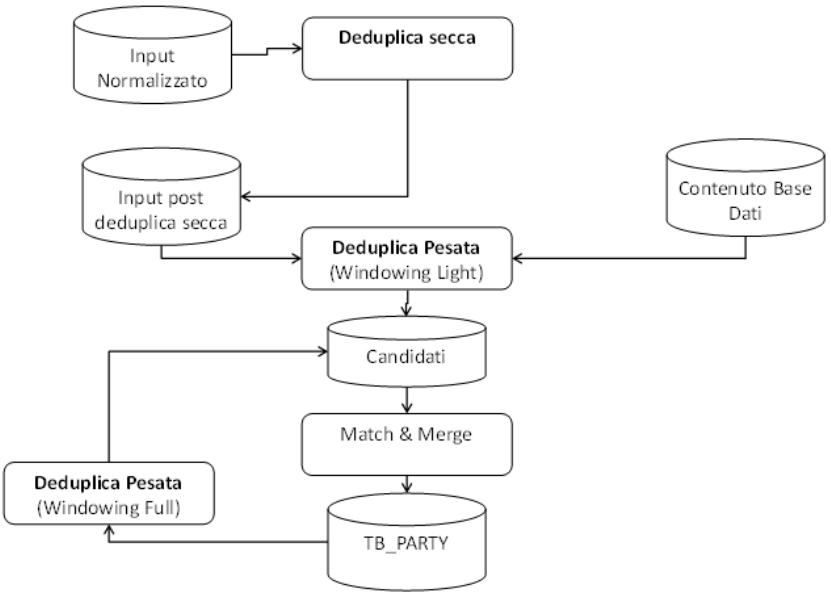

Figure 2.4: Deduplication flow

deduplication: "secca" and "pesata".
The deduplication "secca" is referred to the identification of master records belonging to the same subject on the basis of complete equality of all information identified as deduplication keys. Deduplication keys are the set of all information that can lead to the identification of duplicates. This duplicate search phase does not apply any recognition algorithm but, based on the total equality of all deduplication keys, assumes that two master records belong to the same subject. In this way it is possible to avoid applying complex algorithms when two master record are clearly the same, in order to make the operation faster.
The deduplication "pesata" involves the usage of recognition algorithms, since the process take place among sets of deduplication keys that are not completely equal (if they were equal they would have already been deduplicated in the previous phase). The algorithm has an high computational complexity, so it is applied on a subset of records that have common characteristics, like same address or same phone number. The search for possible candidate records for the deduplication is

called "windowing".
Once the candidates are determined, their similarity are checked. Through a specific configuration table, it is possible to define which combinations of deduplication keys are the same or have a level of similarity above a certain threshold such that it can be determined that the records belongs to the same subject.

2.3.6 Historicization

Once records belonging to the same subject have been identified, it is necessary that the different information are organized in such a way to maintain all the data. The goal is to always be able to identify the most recent, thanks to date, and possibly most reliable information, thanks to source. Indeed some feeder systems provide certified information that are therefore more reliable than others.
Each list used for uploading data to the system has a priority level, through which is determined the greater or lesser reliability of the data relative to other lists.
The updating of the information follows these two rules:

- the list is updated only if the one of the new record has an higher priority with respect to the one inside the database. The priority is expressed through a number, where 0 is the lower.
- the data is updated only if the one of the new record is more recent with respect to the one inside the database.

2.3.7 Master Data summary

Once information that are part of the same master record have been identified, it is necessary for all these information to be associated to single record for every master subject, that contains the best data. This operation, called synthesis, must be done whenever new master information goes into updating the database.
The synthesis of a master data is done in two distinct steps

- An initial summary is done in the previous phase (Historicization) where the received data are subject to a first level of synthesis, aimed at aggregating all

the master information, received at different times and from different sources, that report exactly the same information

- The second step is aimed to obtaining the single master record, called golden record: among all the records available for the master subject (as a result of the activity performed during the historization phase), the most recent information present on the record will be selected.

The data obtained from the synthesis is the data visible outside the CCDB system: this means that all the features that have access to the CCDB have access to the summarized data.

2.3.8 Products, services and contacts deduplication

Like master data, also vehicle, service and contact data require a dedup operation, which is aimed at analyzing new products being acquired to see if they are already in the database and thus avoid entering the same product multiple times.
The management of product data is simpler than for master data, since for the product data there is a unique natural key that makes it possible to identify in a simple and certain way the existence or non-existence of the subject on the database. In addition, it is not required any normalization activities for product data.

2.3.9 Products, services and contacts summary

Also in this case, after dedup, a synthesis operation takes place to collect all the information on the individual object received at different times from different sources, so as to obtain a single record per product, containing the best data.
The summary procedure aimed at obtaining the single product record (golden record), consists of selecting from all the available records of the single product, the most recent data. Thus, for each column the value present in the most recently received record is extracted.
Each time new information about an existing product is uploaded to the system,

the summary process is repeated to ensure that the information received becomes part of the golden record.

2.3.10 Master data and products reconciliation

The purpose of the reconciliation phase is to reconcile master and product information that in the mapping phase has been dissected and distributed on the different tables and that, once distributed on the different target tables need to be related. The aim of this phase is to match each product with the master subject to whom it is referred and to match each non-auto product with the auto to which it is referred.

2.4 Normalization phase

As introduced before, the normalization phase is a key point in the process of data storage. Indeed, the records that are passed as input into the database can have very different origins from each other: they may, for example, come from dealerships because an individual has shown interest in buying a car, or from surveys via telephone or taken online. In addition, since it is a very large and globally known automaker and includes several brands within, it has markets that span many countries.
As a result, data have different formats depending on the market and source and they may be subject to various errors caused by missing data or incorrect data entry.
Therefore, it is necessary a phase in which there is an enrichment and cleansing of data so that we have values that are as clean as possible within the database and so that, at a later time, it is possible to work and perform analysis on truthful data.
The normalization is a single phase but it can be divided in is three main conceptual steps:

- Pre-Trillium

- Trillium
- Post Trillium

2.4.1 Pre-Trillium

This is the first step of the normalization phase where some fields are cleaned and adjusted in order to have a readable and standardized data according to the database specifications.
This step takes as input data from "TB_PT_MAP" table where the initial record is split inside the various fields and its goal is to generate the input file for Trillium, containing the data to be normalized.
Firstly, a series of cleaning operations are done on various fields in order to provide the cleanest possible data to Trillium. Some components such as email, phone, tax codes (nat_id) and vat number (company_id) are normalized and validated directly on the database through custom validation functions and they will no longer be normalized by the software, while, for components such as name and address, in this step are performed only cleaning operations.
The input file for Trillium is composed, therefore, of some already cleaned and normalized fields on which the software will not act and other cleaned fields on which it will perform normalization.

```
1  function fn_get_first_name(p_cod_market varchar2, p_first_name
       varchar2) return varchar2 is
2
3          v_first_name varchar2(4000 char) := p_first_name;
4
5      begin
6
7          if regexp_like (upper (trim (v_first_name)), '^X([[:space:]]*
       X)*X$', 'i')
8              or regexp_like (trim (v_first_name), '^([[:punct:]]{1}[[:
       space:]]*)+$', 'i')
9          then
10             return null;
11         end if;
```

```
12
13          return v_first_name;
14
15      end fn_get_first_name;
```

Listing 2.1: Example of pre-Trillium normalization function

The code 2.1 shows an example of normalization function: in particular, it is referred to the field called "first_name".
The function takes as input the value referred to the field "first name" and check if it is composed only by punctuation marks followed by blank space or if it starts with the letter X followed by other letter "X" or blank space and ends again with letter X, to avoid values such as "XXX" or "X XX" that are not valid.
If at least one of this condition is satisfied the function return a null as value to avoid the storing of wrong data.

2.4.2 Trillium

The second step of the normalization is done relying on an external software called Trillium.
The Trillium Software System (TSS) is a widely solution for managing data quality. It specializes in providing tools and services for cleansing, standardizing, enriching, and improving the quality of data within an organization. TSS is able to handle diverse types of data, including structured and unstructured information, and it offers features like data profiling, data cleansing, data enrichment, and data matching. These functionalities help organizations in maintaining high-quality data, which is essential for making informed decisions and achieving operational efficiency [3].
In particular Trillium Software System is used to perform the normalization of the street address components, such as street name, street number and geographical coordinates and of name components, such as first name and the assignment of gender and party type.
For the street address component, TSS base its operation on postal files, that are files containing all the address information. Note that these files are extremely

important as without them it is not possible to perform the address normalization. Indeed, they have to be periodically updated, in order to have the newest and updated version of them.
Instead the name components rely on dictionaries, that are file containing information and patterns about master information.
There are two types of dictionaries: the standard one provided by Trillium and the custom one that is the result of multiple modifications and interventions made over time, both by consultants and by Trillium's own suppliers.
Trillium receives the input file created appropriately, as described in the previous step, and based on its internal components it processes an output file containing the normalized data.
The process from input to output is nothing more than a series of sequential steps in which are performed transformations, cleaning, and normalization of data (address and name).

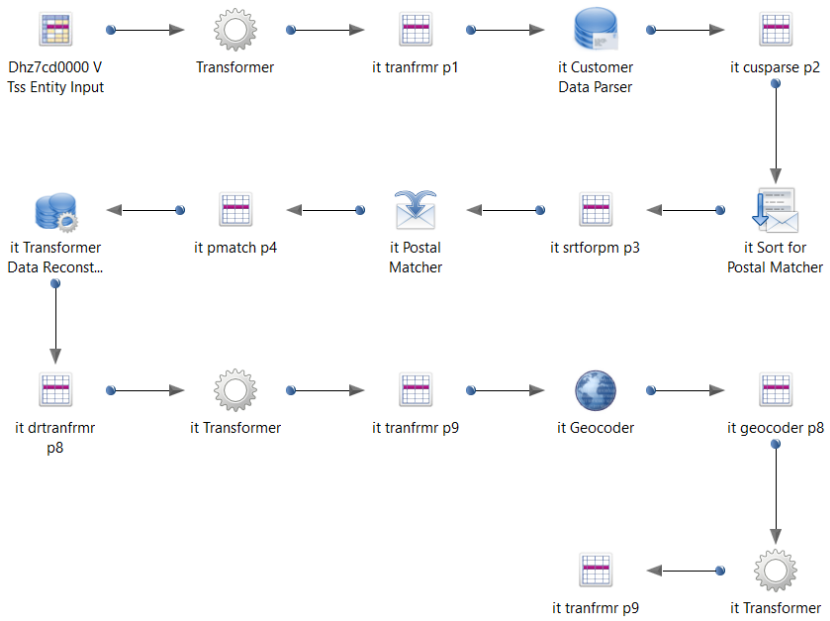

Figure 2.5: Example of Trillium project

Figure 2.5 shows an example of the process that occurs inside Trillium. As it can be noticed there are some steps: firstly the block called "Customer Data

Parser" is a dictionary and it is used to normalize the name component, splitting if necessary the record in name and surname and assigning the gender and party type. The "Postal Matcher" refers to the address component and contain the postal table on which is based the address normalization. Then the "Transformer Data Reconstruction" is used to reconstruct the entire record as it is divided into the various components during the process.
It is important to underline that what happen inside this process is a sort of "black box", in that it is possible to modify and customize the flow but the normalization itself is managed by the software.

2.4.3 Post-Trillium

Once Trillium performs the normalization of the input data, the results are upload inside the CCDB, thank the external tables. Before that the records are stored inside the target table, TB_PT_NORM, additional check and adjustments arose over time, are made on the database and not on Trillium since it is easier working directly on the tables and it impact less the normalization phase from a performance point of view.

```
function fn_get_hno(p_cod_market varchar2, p_hno_map varchar2,
    p_hno_norm varchar2, p_addr_match_level number) return varchar2 as

        v_hno varchar2(50 char) := '';

    begin

        v_hno := trim(upper(p_hno_norm));
        if v_hno is null then
            v_hno := trim(upper(p_hno_map));
        end if;

        return v_hno;

    end fn_get_hno;
```

Listing 2.2: Example of post-Trillium normalization function

The code 2.2 reports an example of post normalization adjustment: in this case the field under review is "hno", that correspond to the street number. The value received as output by Trillium is transformed in upper case and possible leading and trailing blank spaces are removed.
Then if the value is null, it is replaced by the one present on the "TB_PT_MAP", also in this case in upper case and without leading and trailing blank spaces.
Finally, once all post normalization adjustment are performed, the data are uploaded on the "TB_PT_NORM" table and begins the deduplication phase.

Chapter 3

New normalization environment solution

3.1 Introduction

As mentioned in the previous chapter, a part of the normalization phase is entrusted on an external software called Trillium. This implies that there are shells used to handle the extraction of data from the database, to manage the call the software, and finally to send back the output to the database.
Both CCDB system and Trillium software run on IBM's AIX servers, but, unfortunately, for the latter the company has not granted development and service/support, so no more updates have been brought in.
For this reason, it was necessary to find an alternative to ensure the proper functioning of the software and the normalization phase. The idea was to propose a new solution that is relied on a new server, RedHat, which led to the development and implementation of Trillium on the new server, while the entire CCDB remained on AIX servers.

AIX

AIX, which stands for Advanced Interactive eXecutive, is a significant operating system in the enterprise computing domain that it is developed by IBM. AIX is built upon the UNIX operating system and it brings together reliability, scalability, and security.
IBM AIX is usually used for corporate servers and workstations. It allows to distribute access to memory, processor, and disk between different tasks and it is possible to configure various security options like dynamic secure tunnel authentication [4].

Redhat

RedHat is a software company that provides open source software products to enterprises. One of their flagship products is Red Hat Enterprise Linux (RHEL) that is a powerful and widely used operating system designed for enterprise-level computing environments. One of the key strengths of RHEL lies in its stability and reliability. It undergoes rigorous testing and quality assurance processes to ensure it meets the stringent standards required for enterprise-grade computing environments. It also incorporates robust security features, including encryption protocols, access controls, and compliance tools [5].

Not being completely familiar with the version of Trillium on Red Hat, it was necessary to adapt some processes, such as the implementation and testing of the new interface between AIX and Red Hat for what concern the call to Trillium software. Another necessary action was to export existing projects on the old version and import them in the new one, making sure that they work properly.
Taking advantage of this migration, the postal files used for correct address normalization were updated. Since the CCDB system can handle different markets, the mailboxes will have different formats and standards for each of them, and it is therefore necessary to install postal files for each of them.

Finally, the non-regression tests were performed in which it was checked that the processes did not perform significantly worse than the previous version. These checks were performed on the following components: Name, Address, and Geocoding.

3.2 Uploading group shell

As just mentioned, it was necessary to adapt some of the components inherent to the management of the software for normalization, since during the transition many things changed, such as directories and file names, in addition to the very fact of having the software on another server.
Since this phase was shell-unix based, it was tested that the various commands would run correctly also on the new machine and that the shells would work properly.
The first shell that was modified is the one called "start_batch_load.sh" which is used after the check and transcoding phase to upload data streams to the Trillium software and initiate normalization. Moreover it is also used by technicians to manually launch some uploads in order to perform tests and/or analysis.
Going into more detail, the shell receives as input some parameters such as "market code", "group code" and "id load" based on the type of load you want to run.
After an initial phase in which the input parameters are checked, there is the normalization phase: the ".dat" file containing all the data to be normalized for that specific upload is moved from the 'datadir' directory of the database into the batch in the 'data' section. The batch in question is the project of Trillium, so the one responsible for the normalization, that has been exported in batch form on the server. In the following section the structure of this batch will be explained in more detail.
At this point, depending on the casuistry, the file "debug_norm_failure.sh", "parallel_start.sh" or "start.sh" is run to kick off the normalization phase on Trillium.
Then the normalized data are moved within the database on target tables and the Trillium directories are cleaned to avoid contaminating subsequent operations.
Changes have been made in the new version of the shell to ensure proper functioning. For examples, some of the directories changed and had to be updated.

As mentioned before, the Trillium software was moved to the RedHat servers while the CCDB remained on the AIX server. The most substantial difference was in developing and handling the connection to another server. In fact, previously, the call to the trillium software was done by simply moving to the batch directory with the command `cd $CCDB_NORMDIR/$1/n1/p$3/batch/scripts` and then running the appropriate file, such as "debug_norm_failure.sh."
With the new version, however, a call to an external server is required. To do this, it is used the command **ssh** which provides a ssh connection.
Shh stands for "Secure Shell" and it is a cryptographic network protocol that provides a secure way to access and manage remote machines over a network. SSH applications are based on a client–server architecture, connecting an SSH client instance with an SSH server [6].

```
1  trillium_path=$RH_NORMDIR/$1/n1/p$3/batch/scripts
2  cd $trillium_path
3
4  if [[ ("$1" == "3106" || "$1" == "3123") && -f debug_norm_failure.sh
       ]]
5  then
6
7      ssh tss15adm@XX.XX.XX.XXX << EOF
8  cd $trillium_path
9  ./debug_norm_failure.sh
10 exit
11 EOF
12
13 elif [[ -f parallel_start.sh ]]
14 then
15
16     ssh tss15adm@XX.XX.XX.XXX << EOF
17 cd $trillium_path
18 ./parallel_start.sh
19 exit
20 EOF
21
22 else
```

```
23
24      ssh tss15adm@XX.XX.XX.XXX << EOF
25 cd $trillium_path
26 ./start.sh
27 exit
28 EOF
29
30 fi
```

Listing 3.1: Ssh implementation in "start_batch_load" shell

In code 3.1 is shown an excerpt from the new shell. As it can be noticed after an initial `"if"` where is checked if exist the file, is performed the ssh connection. The ssh command is composed by two parts: the one before the '@', in this case tss15adm, is the user that we want to use for the authentication in the server while the part after '@' is the IP address or the name of the server. In this case, for privacy reasons, the IP has been covered.
Once the ssh command successfully connect to RedHat server, we moved into the script path and then the normalization file is run.

3.3 Project uploading

When Trillium software was moved to the new server, pre-existing configurations and assignments were not retained. As a result, they had to be recreated, including repositories and related users.
In addition, also the projects that were on the old server have had to be managed since they were not automatically imported on the new one. Therefore, it was necessary to export projects from the old server and import them on the new one. This process ensured that project definitions, configurations, and cleanup rules were replicated in the new environment.
In order to manage the various projects, it was used the Control Center, also called the CC [7]. The CC is the main interface of the software and it is the central point from which users can access and manage all aspects of their data quality projects, such as initiating, monitoring, and managing data cleaning and normalization

processes.
At first, through the CC of the software installed on AIX, the projects were exported. In particular, as already mentioned, the thesis is focused on market number 1000, that is the one referred to the Italian market.

3.3.1 Repository and user

Another operation to be done was to create the "repository - user" association, since, initially, Trillium on the new server was completely empty.
As a result, it was necessary to recreate these associations to ensure that users have access to the appropriate projects and project definitions in the new environment. This step is critical to enable users to continue to work effectively with Trillium projects in the new server while maintaining proper control over access and permissions.
Specifically, each market has been associated with its user, who is then the only one able to operate on the project within its repository.
To do this, it was necessary to use a specific component of the software called the Repository Manager. The Repository Manager serves as a central repository where all project definitions, configurations, cleanup rules and other information related to Trillium projects are stored. It also allows users to manage permissions to access and modify project definitions, providing adequate control over access to data and configurations.

To get a more complete overview of how the normalization phase works through Trillium, let's now take a closer look at the structure of the projects and their respective batches.
When a project is on Trillium, it is a "client-side" project, in that definitions and configurations are stored and managed within the Trillium environment, either locally or on the server. In this case, users have the ability to open, edit and run projects directly in the Trillium interface.
This approach is useful when you want to work interactively and have direct control

over data cleaning operations. It is particularly suitable for tasks that require active user intervention and supervision.
But when a project is integrated within an automated flow, as in the case of CCDB, it is necessary to export them in batch form so that they can be executed without the need for direct user interaction.
In paticular, a batch of a Trillium project is structured in the following folders (directories):

- data: as suggested by the name in this directory there are the data. There are those that have to be normalized and those after normalization. Initially this directory is empty and it is dynamically filled each time with the new data, which once normalized, are deleted so as to leave it clean for the following loading.

- ddl: ddl stands for Data Definition Language and it contains the files that define the structure and format of the data within the project.

- logs: this folder is filled during normalization with any logs. Having this folder is very useful in case there are some errors so that it is possible figure out in which step they occurred and act accordingly.

- scripts: this directory contains the files that initiate the actual normalization. Specifically, the files in question are "debug_norm_failure.sh", "parallel_start.sh" and "start.sh" which are run by the shell "start_batch_load.sh", as explained earlier.

- settings: in this folder there are configuration files and specific definitions for the project.

- tables: the directory contains files or resources that are relevant to the tables and data that are used or generated during project execution.

Notice that inside the batch, also the contents of some files such as the "start.sh" and "debug_norm_failure.sh" were changed. In particular, the "start.sh" shell has been updated with the directories of the new server, and some configuration files

have been changed, such as the "config_batch" file, which became "config64."
But it is important to underline that in the "start.sh" shell, not only the file names and directory have changed, but the table and data configuration has been completely redesigned, as well as the management of the various normalization processes.

3.4 Export shell

The second shell that was modified is the one called "export_batch.sh". This file is very important in that it is responsible for exporting a project from the Control Center and importing it in the server in the form of a batch so that it can be used, for example, running the shell "start_batch_load.sh", or by manually inserting the data within the batch into the 'data' directory and running the file "start.sh".
Specifically, once the shell is run with the relevant market as a parameter, it is chosen from a list which of the projects in that market you want to export. Once selected the batch is extracted from the project. At this point the ".sh" file in the scripts directory is renamed to "start.sh" so that it is aligned with the batch specification. Then the contents of some files is modified, including the "start.sh" file itself, updating the directories with those of the server so as to ensure proper operation.
Finally, the commands shown in the code 3.2 are printed on the screen, and they must be executed manually by the user:

```
1 echo "*Backup* the previous batch and execute the following command
    with the user ccdb0000"
2 echo "  $ mv ${batch_dir_final} ${batch_dir_final}_bck_$(date +%Y%m%d
    ) && cp -r $batch_dir ${batch_dir_final%%/batch}"
3 echo "To import the current batch in Production execute the following
     from production"
4 echo "  $ mv ${batch_dir_final} ${batch_dir_final}_bck_$(date +%Y%m%d
    ) && scp -r XX.XX.XX.XX:${batch_dir} ${batch_dir_final}"
```

Listing 3.2: batch moving commands

As anticipated by the comment, the first part of the first command, `$ mv $batch_dir_final $batch_dir_final_bck_$(date +%Y%m%d)`, makes a backup of any previous project, renaming it with the current date, while with the second part, `cp -r $batch_dir $batch_dir_final%%/batch`, the batch is physically moved from the Control Center directory to the server.
Similar to what has just been described, the second command performs the same operations as the previous one with the difference that the batch is saved in the "Production" environment and not the "Certification" environment, as it can be seen from the IP present before the directory (obscured in this case for privacy reasons).

In this case, a new shell called "wrapper.sh" had to be created on the RedHat server that would be able to manage calls to the "export_batch.sh" shell.
Indeed, as already mentioned, many directories changed in the new server, and, in addition to changing from one server to another, a substantial difference was that in this case they also changed from the Certification environment to the Production one. This was not happening on the AIX server, in which there were no directory differences between the two environments.
Specifically, the directory inherent to the Trillium software, the one from where you take batches to export, is called `trilliumQA` in certification while in production it is `trilliumPROD`.
For this reason it was necessary to create the shell "wrapper.sh", shown in part in Code 3.3, which run the shell "export_batch.sh" in the two different environments, passing them as a parameter.

```
1 ./export_batch.sh $cod_market $encoding CERT $suffix
2
3 RCcert=$?
4 if [[ $RCcert -gt 0 ]]
5 then
6     echo "Error in CERT batch"
7     exit $RCcert
8 fi
```

```
9
10 ./export_batch.sh $cod_market $encoding PROD $suffix
11
12 RCprod=$?
13 if [[ $RCprod -gt 0 ]]
14 then
15    echo "Error in PROD batch"
16     exit $RCprod
17 fi
```

Listing 3.3: Wrapper shell

Once the shell "wrapper.sh" was created, the shell "export_batch.sh" has been modified so that it could handle the two different environments.
In addition to the changes already described due to the move to the new server, an example of a change is that depending on the environment passed as input, there is a different directory, which value is insert inside a variable that is subsequently used as shown in the code 3.4.

```
1
2 if [[ $environment == 'CERT' ]]
3 then
4     ts_env=/trilliumQA
5
6 elif [[ $environment == 'PROD' ]]
7 then
8     ts_env=/trilliumPROD
9
10 else
11     echo "Unsupported environment"
12     exit 1
13 fi
```

Listing 3.4: New environment management in "export_batch" shell

Similar checks are also performed within some files where, depending on the environment, some paths are updated. In these cases, the files are modified, updating path an name of other components, in order to make them executable for

the specific environment.
Additionally, a new logic has been developed to ensure that the same project is always exported in the two different environments. When the "start_batch_load.sh" shell is first runs with the CERT parameter, the user chooses which project within a specific market they want to export. Note that there can be more than one project, and in that case, the user selects it. At this point, thanks to the developed enhancements, when the shell is runs for the second time with the PROD parameter, the initially selected project is automatically exported to avoid potential user errors and streamline the process.

3.5 Non regression analysis

During the software development cycle, a key phase is the non-regression analysis. This process is a set of tests, both automatic and manual, that are performed after implementing changes in an application's code. The main functionality is to detect and correct regressions [8].
Within the CCDB, non-regression analyses are performed whenever the postal files and dictionaries of the Trillium software are changed. Indeed, as explained in the previous chapter, these files are updated approximately every six months, so that they have the most recent and reliable versions and, consequently, the normalization is more accurate.
The analyses that are performed are used to make sure that the normalization results have not undergone a regression, that is a change in the behaviour of the software that leads to a result that does not conform to expectations, or in this case leads to a result that is worse than the previous version.
Usually, the non-regression analyses are performed over three different component:

- Postal component: related to the customer address, such as city and street name
- Custom component: related to the master information such as the gender
- Geocoder component: related to the geographical coordinates

In this case, having developed a new environment for the software, in addition to the tests just described, further analyses were conducted to assess the performance of the new solution.
The first test concerned computational time, during which the execution times of the two different machines were compared. In the second test, it was verified whether or not the addition of the `ssh` call to an external server led to a significant increase in execution time.

3.5.1 Implementation

In the non-regression analyses has been made changes so as to improve the accuracy of the analyses. In particular, these improvements were made in the postal component.
The analysis are launched by a procedure that initially creates two different external tables that are filled with the contents of two files, respectively: 'tss_output_asis' and 'tss_output_tobe'.
These files represent the outcomes of the normalization process using the Trillium software with two versions of the postal files. The "asis" designation is referred to the old version, while the "tobe" designation corresponds to the updated postal files.
Once these tables are obtained, the analysis are performed comparing the data, and the output consists in the generation of two files: 'postal_file_detail' and 'postal_file_general'.
In the "detail" file there are all the records in which something has changed, from one version to another. This provides the capability to have the entire record in both versions and compare them. Notice that, being highly detailed, the file is sent to the ICT team, which is responsible for analizying any errors in singles records.
In the "general" file, on the other hand, there are general statistics, such as, the total number of records where something changed in normalization and it is used to get an overview of how the results have changed since the files were updated.
The implementation was performed in the creation of the "general" file. Among the possible statistics, are also presented information concerning the address match

levels within the file. This metric is a number that normally varies between 0 and 5 (but it depends from market to market) and indicates how accurately the address has been normalized: a value of 0 indicates that the address has had a positive match in all normalized fields and, therefore, it is possible to locate the position on a map with high accuracy. A value of 5 still suggests relatively good accuracy, even if with some limitations. As the value decreases, the quality of the address deteriorates, until it reaches 1 where the information are so poor that make the identification of the building impossible.
One of the results of this analysis is a table in which there are the number of records that changed level of quality of the address in the two version of postal file. In this case there are two abbreviations: "OK" indicates that the level is 0 and "KO" indicates that the level is one of the other cases, so that the address has not been perfectly normalized.
Since the CCDB is multi-market, each market may have its own particular address match level management, dictated by the fact that each State has a different management of addresses and, consequently, of postal files. In fact, in some cases, an address match level of "4" is sufficient to identify a building and it is therefore considered good as well as level 0, which shows up as "OK" on the table.
The idea for the modification was to include in the results table the cases where values other than 0 are considered optimal, hence "OK". Indeed, previously, all the values different form 0 were considered "KO".
Initially, a function called `fn_check_valid_aml()` was created, taking the market code and the address match level as input. Subsequently, it checks if there are additional accepted values for that specific market and returns "0" if the level passed as input is considered optimal and "1" if it is not.
After creating the function, the table generation was also modified, adding two rows that represent the cases where a record changes the value of address match level but it is still considered "OK". For example if a record pass from a level of 0 to 4 it is considered in the row "OLD OK > NEW OK" in the table 3.6, while without this modification it would be considered in the row "OLD OK - NEW KO". Thus, this modification corrects the whole distribution in the table, as it add a new casuistry.

This concept will be further explored in the "Postal Components" section, dedicated to a more detailed analysis of these tables.

3.5.2 Results

The results of the non-regression analyses and of the runtime tests are shown below. The results obtained from the Trillium software in the new solution, with updated postal files, are compared with those from the software in the old version.
Notice that, as explained, the thesis is focused on the market 1000, the Italian one, and consequently, also the analysis are performed on this market.
The non-regression analysis results display the "general" files and they are divided by components, while at the end are presented the runtime test results.

Custom component

The first results are inherent in the customer master component, specifically on the gender and party type values.
As it can be seen, the file is composed by three different tables: a black, an orange and a blue one.
The black table shows the distribution of party type and gender values of the normalized records. In particular, the party type represents the type of record and it takes the value "C" if the record corresponds to a company, "P" if it is a private person, and "U" if it is undefined.
On the other hand, the gender applies only to person and takes the value of "M" if the person is a man, "F" if it is a woman, "U" if it is undefined, and "N" if the field is null.

In Figure 3.1 are shown the statistics of values just described, in three different situations: "INPUT" indicates the value of the record before it was normalized, so as input to Trillium, "OLD" indicates the record normalized with the old postal files while "NEW" indicates the record normalized with the updated ones. For each of these there is both the number of records and the percentage of the total.
As it can be seen there is a difference only between the input and output values while nothing has changed between the old and new versions. This result is actually

Party_type/Gender Code	INPUT Dictionary num records	INPUT Dictionary %	OLD Dictionary num records
C	35552	9.85%	27396
PF	42976	11.91%	106908
PM	96553	26.76%	218194
PN	0	0%	44
PU	185645	51.46%	1070
U	0	0%	7114

(a) *Caption*

OLD Dictionary %	NEW Dictionary num records	NEW Dictionary %
7.59%	27396	7.59%
29.63%	106908	29.63%
60.48%	218194	60.48%
.01%	44	.01%
.29%	1070	.29%
1.97%	7114	1.97%

(b) *Caption*

Figure 3.1: Party type and gender distribution

perfectly normal since, during the update only the new postal files, containing the address component, were updated while the dictionaries were not changed.
However, it can be seen a massive decrease in records that were initially identified as private but gender was undefined, passing from more than half of the records, 51.46%, to a negligible percentage, 0.29%. This result is a clear sign that the normalization phase is extremely important and it is also very effective.
In the orange table, the one at Figure 3.2, are shown the number and percentages of record distributions that from the input to the output of the normalization phase maintained the same party type value.
Here we have the comparison between one version and another, but as already mentioned, in this case there are no differences.

The last table is the blue one, shown in Figure 3.3, and it is the opposite of the previous one. Indeed, it shows the number and percentages of records that have changed the value of party type. As it can be seen, in the "OLD vs NEW" section, representing precisely the comparison between the old and the new version, the percentage of records that changed is 0%.

Party_type/Gender Equal Results	Num Records (OLD vs NEW)	Num Records (%) (OLD vs NEW)	Num Records (INPUT vs OLD)
Total Records	360726	100%	161459
Both Private with same gender	326216	90.43%	136520
Both Company	27396	7.59%	24939
Both Unknown	7114	1.97%	0

(a)

Num Records (%) (INPUT vs OLD)	Num Records (INPUT vs NEW)	Num Records (%) (INPUT vs NEW)
44.75%	161459	44.75%
84.55%	136520	84.55%
15.44%	24939	15.44%
0%	0	0%

(b)

Figure 3.2: Same party type / gender distribution

Party_Type/Gender Different Results	Num Records (OLD vs NEW)	Num Records (%) (OLD vs NEW)	Num Records (INPUT vs OLD)
Total Records	**0**	**0%**	**203139**
Different Party Type	0	0%	17183
Equal Party Type - Different Gender	0	0%	185956

(a)

Num Records (%) (INPUT vs OLD)	Num Records (INPUT vs NEW)	Num Records (%) (INPUT vs NEW)
56.31%	**203139**	**56.31%**
8.45%	17183	8.45%
91.54%	185956	91.54%

(b)

Figure 3.3: Different party type / gender distribution

Postal components

The second set of results is associated to addressing component and also in this case we have three different tables refining the overall statistics.
The first is the black one in Figure 3.4, which shows the distribution in numbers and in percentages of the address match level values of the records before and after the postal file update. As it can be seen there are differences between the OLD column and the NEW one which shows that indeed there have been changes using the new postal files.
Looking more closely it can be seen that level 0 has remained more or less unchanged while high values, which correspond to good addresses, have decreased in favour of low values, which are associated to incomplete addresses. At first this behaviour

might seem negative, as address match levels have worsened. In reality, updating the postal files results in more accurate address normalization, which leads to more careful level assignment. In this case, some records that had previously been assigned as discrete, or at least with a higher level, were reevaluated in favour of greater veracity of the data. In this way there are fewer cases of false positives, that are records that were initially decreed as good but actually were not.
However, as mentioned before, more detailed analyses are left to a specialized team that analyze individual cases.

AML code	OLD PFile num records	OLD PFile %	NEW PFile num records	NEW PFile %
0	59388	55.69%	59067	55.39%
1	11869	11.13%	14389	13.49%
10	0	0%	0	0%
2	18222	17.09%	16894	15.84%
3	14	.01%	14	.01%
4	6320	5.92%	5913	5.54%
5	10810	10.13%	10346	9.7%
6	0	0%	0	0%
7	0	0%	0	0%
8	0	0%	0	0%

Figure 3.4: Address match level distribution

Also in this case, the orange table in figure 3.5 depicts the number of records that maintained the same address match level value.
This data is important because it can quickly give useful information. In fact knowing that with a postal file update about 93% of the records did not change level is good since if they had changed for example half of the records there would have been something strange.
If that had happened, it should have been investigated what might have been the cause of such a large change.

AML Equal Results	Num Records	Num Records (%)
Total Records	99056	92.9%
Both OK	56056	56.59%
Both KO with same failure	43000	43.4%

Figure 3.5: Same address match level

The last table, depicted in Figure 3.6, shows the number of records that changed address match level value.
In this case the value of "OK" indicates a high level of accuracy, thus 0 and in some markets other values as well, while "KO" indicates a not optimal level.
As it can be seen the values in the two versions are compared, and in each row a different case history is represented. The first shows the records that pass from a value of 0 or considered good, to a not good level.
In the second row there are the records that were instead considered not optimal but, thanks to the postal file update, were normalized correctly.
In the third row there are the records that remained non-sufficient (KO), worsening also their level, e.g., passing from a value of 3 to a value of 2.
The next row shows the opposite, i.e., all those records that remained KO but that improved their level of accuracy, e.g., passing from 3 to 4.
The last two lines were introduced recently, thanks to the implementation made in the code of non-regression analyses. In fact, initially it was not expected that there would be multiple levels considered OK, since the only acceptable value was 0. With the addition of the changes instead, in some markets, even values such as 4 are considered good. It is therefore necessary to keep track of all the records that, although they are still considered good, pass from a level of 0 to a lower one, for example 4 (fifth line) and vice versa (last line).
In this case, the analysis was performed on market 1000, which, however, does not accept as OK any values other than 0.

AML Different Results	Num Records	Num Records (%)
Total Records	**7567**	**7.09%**
OLD OK - NEW KO	3332	44.03%
OLD KO - NEW OK	3011	39.79%
OLD KO > NEW KO	984	13%
OLD KO < NEW KO	240	3.17%
OLD OK > NEW OK	0	0%
OLD OK < NEW OK	0	0%

Figure 3.6: Different address match level

Geocoder component

Finally there are the results of the non-regression analyses for the geocoder component.
In the first table in Figure 3.7 the distribution of records in the various accuracy levels are shown. Again, OLD values are compared with NEW values.
Note that in this case, as shown in the first column, the best value, so the one with the highest accuracy, is level 5, which identifies the roof of the building. As the value approaches 1 the accuracy becomes worse.

Geographic Code 3	OLD Geocoder num records	OLD Geocoder %	NEW Geocoder num records	NEW Geocoder %
null	6808	3,4%	6946	3,47%
1 - region	190	,09%	22	,01%
2 - city	2265	1,13%	1920	,96%
3 - postcode	50738	25,36%	49477	24,73%
4 - street	30602	15,3%	29417	14,7%
5 - rooftop	109397	54,69%	112218	56,1%

Figure 3.7: Geographic code distribution

Three different tables are represented in Figures 3.8: the first represents the number and percentage of records that maintained the same geographic code value (89.56%) and those that changed it (the remaining 10.44%).
The second table shows the number of records who maintained the same geogrfice coordinates and the number of those who changed them, among those that maintained the same level of geographic code.
The third table shows the records that, by changing the geographic code, had an improvement (increase in level) and the records that, instead, had a decrease in precision.

Finally, the last table in Figures 3.9, represents the distribution of how accurate the coordinates of the records are. In particular, it can be seen that most of the results, about 94%, have good accuracy, with an uncertainty range of less than 1km. On the other hand, less than 5% are in a range between 1 and 5km and only 2% have an uncertainty greater than 5km.

This last tables contain the results of the non-regression analyses that are normally performed when postal files and dictionaries are updated.
As already mentioned, it is very important to understand these results in the correct

same/diff geographic_code_3	# record	%
Same geographic_code_3	179120	89,56 %
Different geographic_code_3	20880	10,44 %

Same geographic_code_3		

same geographic_code_3	# record	%
Same geographic coordinates	12616	7,04 %
Different geographic coordinates	166504	92,96 %

Different geographic_code_3		

different geographic_code_3	# record	%
geographic_code_3 greater	12785	61,23 %
geographic_code_3 lower	8095	38,77 %

Figure 3.8: Same and different geographic code distribution

Average Distance	0,5408	

distance report	#	%
< 1 km	179340	93,04 %
>= 1 km and < 5 km	9122	4,73 %
>= 5 km	4298	2,23 %

Figure 3.9: Coordinates uncertainty

way so that it is possible to follow the evolution of this delicate phase and know in which part of normalization phase act in case of any unexpected results.

Computational time component

As explained, since it has been developed a new normalization solution using a new environment it was so necessary to perform analyses inherent to the execution time so as to compare the runtime in the two cases.
Two different tests were performed: in the first one it is evaluated the simple machine execution time, that is, the speed at which the Trillium software runs on the two different servers.
To perform this test, three different files containing different numbers of records

were used so as to subject the servers to various case histories and file sizes: the files have respectively 1000, 10000 and 100000 samples each.
Once the files were created they have been inserted, one at a time, inside the Trillium project batch, in the 'data' section. Then, the shell called 'debug_norm_failure' was executed, which initiates normalization using Trillium. In particular, the command `date && ./debug_norm_failure.sh && date` was run, which allows to takes the time before and after the shell execution, so as to take the total execution time.
The table 3.1 shows the results of this test. As it can be seen, for small files the time is almost the same while for larger sizes, Trillium performs normalization faster on RedHat servers than on AIX ones coming to be, in the case of large files, even 30% faster.

File dimension	**Server**	**Elapsed time**
1000	Red Hat	00:00:08
	Aix	00:00:09
10000	Red Hat	00:00:14
	Aix	00:00:41
100000	Red Hat	00:02:22
	Aix	00:03:20

Table 3.1: Machine execution time

The second test was performed taking into account the entire flow to which the record sets are subjected in the normalization phase. Thus, an entire batch was loaded using the shell `start_batch_load.sh` in order to test the new changes and also take into account the time it takes to execute an ssh call on another server.
Also in this case, three different files of sizes 1000, 10000 and 100000 samples were generated but, instead of placing them inside the batch, they were left in the directory inside the CCDB on the AIX server.
Once this was done, the old shell was run on the AIX server, while on RedHat the updated one was run, as previously described, adding the commands `date` before and after so as to have the total execution time.

File dimension	Server	Elapsed time
1000	Red Hat	00:00:31
	Aix	00:00:34
10000	Red Hat	00:03:18
	Aix	00:03:17
100000	Red Hat	00:30:04
	Aix	00:30:27

Table 3.2: Group uploading execution time

In table 3.2 the results obtained are shown. As it can be seen, in all three case histories, the execution times are practically the same.
This result is very important since developing a new solution, based on the new environment where Trillium software runs does not involve changes in timing and thus does not lead to slowdowns in the normalization phase.
This means that the new changes are valid and they can therefore be implemented in the production environment. Subsequently, they will be gradually extended to all markets so that the entire database relies on the new solution.

Chapter 4

Novel normalization data platform architecture

4.1 Introduction

Once the Trillium software had been moved to new server, developing a new solution, and verified the correct functioning and fit within the CCDB flow, an alternative to the normalization software was considered. Indeed, the need for this migration, due to the failure to guarantee support and updates from AIX, gave raise to the development of an alternative product that could be sold separately from the entire data acquisition, cleansing and storage flow and that, in the future, could also replace Trillium itself within the CCDB.

The implementation of the new normalization system required a cutting-edge approach, harnessing the potential offered by leading technologies available on the market.

By basing the data platform on Oracle's Autonomous Database (ADB) and integrating the Google Address Validation API service, a robust infrastructure for advanced address management was established.

Subsequently, the features of Google Address Validation API, Oracle Cloud Infrastructure (OCI), and Oracle Autonomous Database (ADB) were thoroughly examined. These technologies represent the core of innovation, with each making a

distinctive contribution to optimizing the address normalization process.
This chapter focuses on the in-depth analysis of integrating these technologies within the new normalization system, highlighting their impacts on performance and operational efficiency.

Google Address Validation

Google Address Validation API is a service offered by Google that provides address normalization and validation functionality. This API enables the efficient management of large volumes of address-related data by offering a reliable mechanism to ensure that addresses are correct and valid [9].
The API accepts an address as input and returns the normalized and enriched address, along with additional information about the quality of the normalization. Google Address Validation API leverages an extensive database of addresses to ensure that the provided address is valid and accurate. This helps in avoiding delivery or shipping errors caused by incorrect or incomplete information. Using this service, businesses can enhance operational efficiency and accuracy in address management.

Oracle Cloud Infrastructure - OCI

Oracle Cloud Infrastructure (OCI) is a cloud computing platform provided by Oracle Corporation. It is a highly scalable and reliable cloud environment that offers a wide range of IT services and resources on-demand [10].
One of the distinguishing features of OCI is its cloud architecture, designed to deliver superior performance and security. OCI provides a diverse range of services, including computing, storage, databases, networking, and many others.
A fundamental aspect of OCI is its flexibility in adapting to various business needs. It enables organizations to create, deploy, and manage applications and workloads in a highly customizable manner, while also allowing for resource optimization and operational cost reduction.
Additionally, OCI offers advanced tools for security, monitoring, and resource management, ensuring a secure cloud environment. This includes features such as

data encryption, access control, and advanced management tools.

Autonomous Database

Oracle Autonomous Database is a fully automated service that simplifies the development and deployment of application workloads for organizations, regardless of complexity, scalability, or criticality. The service's converged engine supports various data types, streamlining application development and deployment, from modeling and coding to ETL, database optimization, and data analysis. With machine learning-based optimization, scalability, and patching, Autonomous Database delivers top-notch performance, availability, and security for OLTP, analytics, batch, and Internet of Things (IoT) workloads. Built on Oracle Database and Oracle Exadata, Autonomous Database is available on Oracle Cloud Infrastructure (OCI) for serverless or dedicated implementations [11].

4.2 The structure

The idea is to redesign the normalization phase again, creating a new package "pg_party_norm", that replace the one currently in use within the CCDB. This package contains the procedures used to initiate and execute the normalization phase. Also in the Oracle environment the package has been named "pg_party_norm" and within it contains the procedure "pr_populate_tmp_tmp_pt_map" which is used to initiate the normalization phase.

```
1
2 procedure pr_populate_tmp_pt_map (p_cod_market varchar2, p_id_load
    number) is
3       begin
4
5           delete from tmp_pt_map;
6
7           insert into TMP_PT_MAP
```

```
8           select cod_market, null, id_record, id_load, id_list,
    record_date, first_name, first_name_short, last_name_prefix,
    last_name_main, last_name_suffix, generic_name, company_name,
    party_type, gender, care_of, address_lang_code, sub_add_1,
    sub_add_2, street_name, street_type, hno, hno_addition,
    nuts_lv1_name, nuts_lv2_name, nuts_lv3_name, nuts_lv4_name,
    nuts_lv5_name, post_code, geographic_code_1, geographic_code_2,
    geographic_code_3, longitude, latitude, pobox, nat_id1, nat_id2,
    nat_id3, nat_id4, company_id, email_1, email_2, website,
    phone_1_int_prefix, phone_1_full_num, phone_1_type,
    phone_2_int_prefix, phone_2_full_num, phone_2_type,
    phone_3_int_prefix, phone_3_full_num, phone_3_type,
    phone_4_int_prefix, phone_4_full_num, phone_4_type,
    phone_5_int_prefix, phone_5_full_num, phone_5_type,
    phone_6_int_prefix, phone_6_full_num, phone_6_type,
    phone_7_int_prefix, phone_7_full_num, phone_7_type,
    company_form_code, company_type_code, company_cat1, company_cat2,
    company_cat3, company_cat4, company_turnover, company_emp_num,
    company_reference, ref_name, ref_gender, ref_role, ref_note,
    email_ref, email_ref_valid, birthdate, birthcountry, salutation,
    academic_title, nobility_title, other_title, profession_code,
    preferred_language, schooling, marital_status, household_members,
    financial_risk_level, pref_contact_code, flag_fga_employee,
    flag_dac, flag_nomail, flag_leasing, flag_renting, flag_seller,
    add_field1, add_field2, add_field3, id_party_input,
    date_endval_record, id_key_feed, flag_hidden
9           from TB_PT_MAP
10          where cod_market = p_cod_market and id_load = p_id_load;
11
12          pr_normalize_address(p_cod_market, p_id_load);
13          pr_normalize_name(p_cod_market, p_id_load);
14          pr_normalize_phone(p_cod_market, p_id_load);
15          pr_normalize_email(p_cod_market, p_id_load);
16          pr_normalize_natid(p_cod_market, p_id_load);
17          pr_normalize_companyid(p_cod_market, p_id_load);
18          pr_populate_tss_elab_norm(p_cod_market, p_id_load);
19
20      end pr_populate_tmp_pt_map;
```

Listing 4.1: pr_populate_tmp_tmp_pt_map procedure

As can be seen in the code 4.1, once the procedure is run, it is populated the table "TMP_PT_MAP", which contains all the data that must be normalized. These fields are taken from the "TB_PT_MAP" which is the table that, in the CCDB flow, is populated at the end of the Check and Transcoding phase, i.e., the table from which the normalization is drawn.
As always, the 'id_load' and 'cod_market' values are passed as parameters, which are used in the `WHERE` condition to prevent mixing records belonging to different markets, and therefore having different characteristics, or from other loads. Furthermore, this operation is important as it reduces the size of the table, focusing only on records of interest, and avoiding the need to work directly with the "TB_PT_MAP" that may contains millions of data.
Tables with the prefix 'TMP_' indicate temporary tables, which are created and used temporarily during the execution of a session. In this case, global temporary tables were employed, which, once created, are visible to all sessions within the database instance. After data are inserted into a global temporary table, they are only visible in the current session and they are deleted at the end of the session or when the process that inserted them terminates.
Despite being automatically cleared at the end of each session, it is a good practice to clean the table before use.
Once populated the table, the procedures 'pr_normalize_address', 'pr_normalize_name', 'pr_normalize_phone', 'pr_normalize_email', 'pr_normalize_natid', 'pr_normalize_companyid' and finally 'pr_populate_tss_elab_norm' are run.
As the name implies, most of these procedures are used to perform the normalization of various component and, currently, they are executed sequentially because, as shown below, they do not contain any normalization operations yet. However, the idea of splitting the normalization of various components into separate procedures is done in order to be able to run them in parallel, reducing the execution time of the normalization phase.

```
procedure pr_normalize_name (p_cod_market varchar2, p_id_load number)
    is
        begin
            delete from tmp_pt_name_map ;

            insert into tmp_pt_name_map
            select  cod_market, id_com_filter, id_record, id_load,
    id_list, record_date, first_name, FIRST_NAME_SHORT,
    LAST_NAME_PREFIX, LAST_NAME_MAIN, LAST_NAME_SUFFIX, GENERIC_NAME,
    COMPANY_NAME, PARTY_TYPE, GENDER
            from tmp_pt_map
            where cod_market = p_cod_market and id_load = p_id_load;

            -- hypothetical normalization phase for name component

            insert into tmp_pt_name_output_map
            select *
            from tmp_pt_name_map
            where cod_market = p_cod_market and id_load = p_id_load;

        end pr_normalize_name;
```

Listing 4.2: pr_normalize_name procedure

In this case, the normalization procedures are empty since they are not the focus of this thesis. In code 4.2, an example of an empty procedure is shown. As it can be observed, the structure for a future normalization phase has been set up. Specifically, all fields related to that component, in this case the name, are inserted into an input table. After potential normalization, they are then inserted into an output table, which will contain the normalized and enriched data.
The only complete procedure is pr_normalize_address, which initiates address normalization via Google address Validation.

4.2.1 Address component normalization

As just said, the goal of the pr_normalize_address is to kick off the address normalization.

Indeed, in this procedure, it is again populated a temporary table, the tmp_pt_address_map, with the following code 4.3.

```
1
2 insert into tmp_pt_address_map
3     select  cod_market, id_com_filter, id_record, id_load, id_list,
    record_date, care_of, sub_add_1, sub_add_2, street_name,
    street_type, hno, hno_addition,  nuts_lv1_name, nuts_lv2_name,
    nuts_lv3_name, nuts_lv4_name, nuts_lv5_name,  post_code,
    geographic_code_1, geographic_code_2, geographic_code_3, longitude
    , latitude, pobox,
4     ROW_NUMBER() over (partition by sub_add_1, sub_add_2, street_name
    , street_type, hno, hno_addition, nuts_lv1_name, nuts_lv2_name,
    nuts_lv3_name, nuts_lv4_name, nuts_lv5_name, post_code,
    geographic_code_1, geographic_code_2, geographic_code_3, longitude
    , latitude ORDER by id_record) as rn,
5     min(id_record) over(partition by sub_add_1, sub_add_2,
    street_name, street_type, hno, hno_addition, nuts_lv1_name,
    nuts_lv2_name, nuts_lv3_name, nuts_lv4_name, nuts_lv5_name,
    post_code, geographic_code_1, geographic_code_2, geographic_code_3
    , longitude, latitude ORDER by id_record) as partition_id_record
6     from tmp_pt_map
7     where cod_market = p_cod_market and id_load = p_id_load;
```

Listing 4.3: Query for tmp_pt_address_map population

As it can be observed, the table is populated with fields related to the address component, such as street name and street number, in addition to the fields inherent to general and identifying information, such as market code and id record. These values are retrieved from the previously populated tmp_pt_map, that contains all the data that have to be normalized.
In addition to the fields just described, two new columns are also added, called 'rn' and 'partition_id_record', which were not present in the previous table. These columns are respectively created using the commands `row_number() over (partition by ... ORDER BY id_record)` and `min(id_record) over (partition by ... ORDER BY id_record)`.
In the first column, the command `row_number() over (...)` assigns a unique

sequential number to each row returned by a query. In this case, the content within the 'OVER' clause specifies the criteria by which the rows should be grouped and ordered. The `partition by` command divides the result sets into partitions based on the fields that follow, and finally, `ORDER BY` defines the order in which the rows should be numbered.
As a result of the query, to each record within a set, composed by all records that have the fields inside `OVER` equal to each other, will be assigned an increasing number starting from 1. The order in which the numbers are assigned is based on the order of the 'id_record' fields: the record with the smallest value will be assigned *1*, and so on in ascending order.
The commands in the second column are similar but instead of assigning a different value to each record in the group, they assign the smallest 'id_record' to all of them in that group. This allows for the unique identification of sets of equal records.
After populating the 'tmp_pt_address_map' table, the following procedure is called 'pr_call_api'. Within this procedure, there is a cursor that points to the just described table. For each group of identical records, it retrieves the first one, that is the one with the 'rn' column equal to 1. The cursor iterates through all the groups, and the corresponding record is passed as a parameter to the subsequent procedure, the 'pr_request_address'.
Note that by doing this, there is an initial 'screening' of records. This means that, if within the same load there are multiple records that can be traced back to the same address, only one of them is processed. This results in a reduction of workload for the system, which translates to a decrease in execution time.

Google Address Validation request

The 'pr_request_address' is the procedure used to handle requests to Google Address Validation.
Firstly, variables are declared, and in this step, some fields are subjected to an initial cleaning phase. Specifically, the fields 'nuts_lv1_name' (corresponding to the name of city), 'street_name', 'post_code', and 'hno' (corresponding to the street number) are passed as input to their respective cleaning functions, such as

'fn_nuts_lv1_name_cleaning', 'fn_street_name_cleaning', and so forth.

```
1
2  function fn_nuts_lv1_name_cleaning(p_cod_market varchar2,
       p_nuts_lv1_name varchar2) return varchar2 is
3
4              v_nuts_lv1_name tb_pt_map.nuts_lv1_name%type :=
       p_nuts_lv1_name;
5
6          begin
7              -- campo nuts_lv1_name:
8              if length (trim (v_nuts_lv1_name)) = 1  or regexp_like (
       trim (v_nuts_lv1_name), '^X([[:space:]]*X)*X$', 'i') then
9                  v_nuts_lv1_name := null;
10             elsif regexp_like (trim(p_nuts_lv1_name), '^[[:punct:]]',
        'i') then
11                 v_nuts_lv1_name := substr (trim(v_nuts_lv1_name), 2);
12             end if;
13
14             -- Pulizia su nuts_lv1_name:
15             v_nuts_lv1_name := regexp_replace(v_nuts_lv1_name, '[[:
       space:]]+', ' ');
16
17             if replace(v_nuts_lv1_name, '*', '') is null then
18                 v_nuts_lv1_name := null;
19             end if;
20
21             v_nuts_lv1_name := trim(regexp_replace(replace(
       v_nuts_lv1_name, '*', ' '), '[[:space:]]+', ' '));
22
23             return v_nuts_lv1_name;
24
25         end fn_nuts_lv1_name_cleaning;
```

Listing 4.4: Example of cleaning function

Above, the code 4.4 shows an example of a field cleaning function related to the street number. Initially, are removed any unwanted characters, such as the letter 'n' before the number and its respective punctuation marks, as well as the character

'°', which could interfere with the actual normalization phase. Additionally, any leading and trailing blank space and leading zeros are also eliminated.
After executing all the functions for the initial field cleaning, the variable 'composed_address' is created. This variable is composed of the fields that have just been normalized, in addition to others taken from the input record: these fields together form a complete address.
Subsequently, a query is executed in which the variable 'composed_address' is compared with the 'input_address' field in the table named 'TB_ADDRESS_COMPONENT'. This table contains all the responses received from Google Address Validation, and specifically, the 'input_address' field contains the string sent to the Google tool for normalization.
In this way, it is checked whether the address of the record in question has already been normalized or not. In case this field is not present in the table, the request to the Google API proceeds.

```
1
2  v_input_json := '{
3                  "address": {
4                      "addressLines": ["' || v_composed_address || '"]
5                      }
6              }';
7
8  v_response := APEX_WEB_SERVICE.MAKE_REST_REQUEST(
9              p_url          => url,
10             p_http_method  => 'POST',
11             p_body         => v_input_json
12             );
13
14 pr_read_response(v_response, v_composed_address);
```

Listing 4.5: Requesto for Google Address Validation

The code snippet 4.5 illustrates the structure required for normalization. In the variable 'v_input_json', it is created a JSON containing the address to be normalized and populated, while the response from the API is stored in the 'v_response' variable. Specifically, the command `APEX_WEB_SERVICE.MAKE_REST_REQUEST` is a

function that enables a PL-SQL application to communicate with external web services through HTTP requests, such as RESTful requests.
The function takes parameters including the URL, which facilitates communication with the API, the HTTP method (in this case POST), and finally, the content of the JSON.
Then it is run the 'pr_read_response' procedure, which is aimed to read and manage the response from the Google API.
Instead, if the address contained within the variable 'composed_address' is found in the 'TB_ADDRESS_COMPONENT', it means that the address has already been normalized in the past. It would be redundant and time-consuming to query the API again, as it would incur unnecessary costs.

Google Address Validation response

The procedure 'pr_read_response' is run only when a request is made to the Google service and it is used to handle the response. It takes as input the response and the address before normalization (the one inside the 'composed_address' variable) and, as mentioned before, it inserts them into the 'TB_ADDRESS_COMPONENT' table, which maintains a history of all the normalized addresses.
However, the response is received in JSON format, so it is necessary to extract the relevant data [12]. Below is an example of the format received from Address Validation [13]:

```
{
"result": {
      "verdict": {},
      "address": {
            "formattedAddress": '',
            "postalAddress": {},
            "addressComponents": [{}],
            "missingComponentTypes": [],
            "unconfirmedComponentTypes": [],
            "unresolvedTokens": []
      },
      "geocode": {},
```

```
13          "metadata": {},
14          "uspsData": {},
15      },
16      "responseId": "ID"
17  }
```

Listing 4.6: Google Address Validation response structure

As shown in the code 4.6, the JSON is divided into two sections: 'result' and 'responseID', which respectively contain the normalization results and a code associated with the made request.
The 'result' section is composed of sub-sections, each containing groups of information.
The first section is 'verdict', which provides fields indicating the quality of the output address in string form. This field can be seen as the equivalent of the 'address_match_level' field present in the CCDB, as it indicates how well the address has been normalized.
The second section is named 'address', and there are the details of the processed address, including 'formatted_address' that contains the fully normalized address, or 'postal address' which includes all information representing a postal address such as country, city, ZIP code, etc...
Another very important field within the 'address' section is called 'addressComponents', which contains all the various components of an address. This field has a structure similar to the one shown in code 4.7.

```
1       {
2           "componentName": {
3             "text": "24"
4           },
5           "componentType": "street_number",
6           "confirmationLevel": "CONFIRMED"
7       },
```

Listing 4.7: Address component structure

As it can be seen, there is the type of component which can be, for example,

'route', 'street_number', 'country', etc., along with their respective values. Additionally, there is a field indicating whether the component has been verified correctly or not.
Finally, the data related to geocoging are placed within the 'geocode' section, where, in particular, it is possible to find the geographical coordinates of the address.
An example of response received from Google Address Validation is shown in the appendix in A.1. In this case the input was 'corso duca degli abruzzi 24, Torino'.

Within the 'pr_request_address' procedure, whether the record has already been normalized or a request has been made to the Google service, the 'tmp_pt_address_output_map' table is populated.
This table combines data from the 'TB_ADDRESS_COMPONENT' table, which contains the normalized data, and from the 'tmp_pt_address_map', which contains the input address components grouped by `partition by`. For each record within the group, the same normalized and enriched values are assigned, as it has been observed that they represent the same address.
The 'tmp_pt_address_output_map' table adopts the same format and column names as the table within the CCDB, making it compatible with the entire flow. For this reason, functions were created in order to adapt certain fields to the established format. An example is the 'quality_address' function, which takes the quality value of the address normalization as input, which, as a reminder, is in string format, and converts it to a numerical value ranging from 0 to 5, similar to the 'address_match_level' field.
Once this table is populated, the address component normalization phase is concluded. As shown in code 4.1, the other normalization procedures are run, but they are currently empty.
Finally, the 'pr_populate_tss_elab_norm' procedure is run. Within this procedure, a query is executed to populate the 'tmp_tss_elab_norm' table. This table reconstructs the initial record by combining the output tables from the various normalization phases. In this case as well, the table adopts the same format and columns as the corresponding table in the CCDB: the 'tb_pt_norm'.

4.3 Analysis, comparison with Trillium

After developing the new data platform and ensuring its capability to handle address normalization, the performance was compared with the current system based on Trillium, installed on the RedHat server.
The first test conducted pertained to the execution time for normalizing a set of records. Three different files were created, containing respectively 10, 100, and 1000 records. In this case, a reduced number of samples were tested compared to the Trillium test between the AIX and RedHat servers. This decision was made because using Google Address Validation incurs a cost for each request made to the service. Therefore, it was chosen a number that allowed for multiple tests to be conducted.
For the new data platform, the 'pr_populate_tmp_pt_map' procedure was executed with the respective cod_market and id_load, using the following command before and after to calculate the elapsed time: `DBMS_UTILITY.GET_TIME`.
Regarding the Trillium software within the CCDB, the 'start_batch_load' shell was run, which encompasses the entire normalization phase, including the SSH call to the RedHat server.

File dimension	Approach	Elapsed time
10	Trillium	00:00:00,27
	Google Address Validation	00:01:23
100	Trillium	00:00:05
	Google Address Validation	00:10:40
1000	Trillium	00:00:32
	Google Address Validation	01:55:48

Table 4.1: Group normalization execution time

In Table 4.1 the results related to the execution time are shown. As it can be observed, for the same file size, the current normalization system based on Trillium is significantly faster than the one relying on the Google service.

In the first case, with just 10 samples, normalization with Trillium is almost immediate, while the one with Google Address Validation it already takes over a minute. However, the most striking case is the one with the file containing 1000 records, where using Trillium takes about 30 seconds, whereas with the Google service, it takes almost two hours.
Obviously, these results are very unfavourable as the time difference is enormous, and the two services are not even remotely comparable.
However, it is important to note that the test with the Google service was repeated a second time, using the same set of 1000 records. In this case, no API requests were needed as the addresses had been previously saved in the 'TB_ADDRESS_COMPONENTS' table and, surprisingly, the execution time was just 4.88 seconds, considerably less compared to the Trillium case.
From this data, it can be understood that the issue regarding excessive execution time is caused by the request and response from the API.

After testing the performance in terms of execution time speed, they were conducted test regarding the quality of normalized data. Specifically, the results of normalization through Trillium and the Google service were analyzed.

AML different resultes	**Num Records (%)**
OLD KO - NEW OK	3 %
OLD OK - NEW KO	28 %
OLD OK - NEW OK	62 %
OLD KO - NEW KO	7 %

Table 4.2: Address match level distribution

Table 4.2 shows the distribution of values in the 'address_match_level' field, particularly comparing the numbers in the Trillium version (OLD) with those from the Google Address API (NEW).
As observed, 62% of the records already had an excellent level of normalization and maintained it when using the Google address validation API. Only 3% improved, while 27% decrease in the level.
Although these results may initially appear negative, it is important to consider

the context in which we are working. Since two different software solutions were used, they will also have different validation standards. As previously explained, the Google service returns a string value to assess the quality of normalization, which is then converted to an integer to be compatible with the corresponding field currently in use on the CCDB.
However, this conversion is based on the original 5 levels (from 0 to 5) proposed by Trillium, which may not necessarily align with those of the Google service.

STREET_NAME	STREET_TYPE	FOR_DEDUP_STREET	HNO	HNO_ADDITION
V VENEZIA	V	V VENEZIA SANTO STEFANO DI CADORE	14	B
VIA VENEZIA	null	VIA VENEZIA Santo Stefano di Cadore	14 B	B
NUTS_LV1_NAME	NUTS_LV2_NAME	NUTS_LV3_NAME	NUTS_LV4_NAME	NUTS_LV5_NAME
SANTO STEFANO DI CADORE			BL	ITALY
Santo Stefano di Cadore	null	null	BL	Italia
NUTS_LV5_CODE	POST_CODE	GEOGRAPHIC_CODE_1	LONGITUDE	LATITUDE
	32045	25050	12,558961	46,561749
IT	32045	25050	12,5496237	46,5580169

Figure 4.1: Comparison between two normalized record

An example highlighting the importance of a more in-depth evaluation is illustrated in Figure 4.1, where two records normalized using Trillium and Google Address Validation services are compared. The two records have an address match level of 0 for the first record and 1 for the second.
As it can be observed, the majority of fields are identical in both records. Note that in the 'nuts_lv5_code' column, the record normalized with the Google service has a correctly populated value, unlike the other record where it is empty.
This example clearly demonstrates that relying solely on the comparison of the "address_match_level" could be misleading. Despite the two records having different values for this specific field, the reality is that the fields related to address component are very similar.

4.4 Possible future developments

As shown above, the main issue related to processing time is caused by the request to the Google service. For this reason, initial logic has already been implemented

in order to reduce the number of request, such as grouping records with similar addresses using 'partition by' and normalizing them only once, as well as saving the response in 'TB_ADDRESS_COMPONENTS' table to avoid re-normalizing a previously normalized record.
Since the system is based on Oracle's Autonomous Database, a possible future implementation could be to develop an AI capable of more accurately recognising if an address has already been normalized. Currently, the system relies only on the exact match of the "composed_address" field between two different records.
Among the various AI possibilities, one could be the implementation of a Natural Language Processing (NLP) model trained to recognise common patterns and similarities between addresses, extracting the key information from the address.
This approach would not only reduce execution time but also decrease costs associated with the request themselves.
Another possible implementation, once the normalizations of the other components are completed, is to execute the respective procedures in parallel instead of sequentially.
Indeed, Oracle Cloud Infrastructure provides features for distributing workloads across different sessions or processes, for example, using job management functions or parallel sessions. This function allows procedures to be executed simultaneously rather than sequentially.
Furthermore, it would be possible to develop a graphical interface based on Oracle APEX. Oracle Application Express is a web application development platform based on databases that allows for the rapid creation of dynamic applications integrated directly with Oracle databases.
In this way, it would be possible to create a user-friendly interface for managing the address component normalization process and given an input address, it could be also able to display key information about the normalized data, such as identifying its location directly on a map.
This would allow for individual record usage, in addition to managing the entire loading process.

Chapter 5

Conclusion

Throughout this thesis, the crucial issue of address normalization within a vast database of a major automotive company has been extensively explored. This involved an established system and the design and implementation of an innovative solution. In an era where accuracy and integrity of information are paramount, it has been demonstrated the vital role of addresses as key elements for identification and localization in various contexts.

The detailed analysis of the database structure and processes involved has highlighted the complexity of the path an address takes, from initial entry to the storage phase. The normalization phase, which received particular focus, represented the heart of this research, aiming to ensure data uniformity, reduce ambiguities, and enhance the overall quality of stored information.

During the study, a concrete challenge in the use of external software for normalization was addressed, leading to the need for implementing a new solution. Crucial roles were played by the tests and analyses conducted, demonstrating that, despite the changes made, the new solution is capable of operating correctly and maintaining performance on par with the previous version.

These results are very promising, as they provide a solution to the initial problem and will gradually allow for the implementation of these changes in the production environment. In addition, the phase of updating postal files has been explained, crucial to ensure increasingly accurate and up-to-date data.

Subsequently, through case study analysis and exploration of emerging technologies, an innovative solution based on new technologies for data normalization was proposed. In particular, the new address validation service offered by Google was utilized.

This solution, thoroughly compared with the Trillium software discussed in the first part of the thesis, introduces new perspectives in address validation. However, the tests conducted highlighted a significant difference in execution times, favouring the currently used software. These prolonged timelines are attributed to the Google Address Validation service and are therefore challenging to improve.

Despite these challenges, the proposed solution should not be completely dismissed. Although it cannot currently replace the Trillium software within the CCDB, the product could be advantageous in situations where the data to be processed are not extremely voluminous or for small businesses geographically confined. As indicated by the results, if an address has already been normalized and, thus, it is present in the database, the execution time is significantly faster.

Furthermore, it is crucial to consider also the economic aspect. The proposed solution incurs costs only in the case of API requests. In scenarios with a limited amount of data, it might not be cost-effective to recurrently pay licenses for the use of Trillium software but to pay only when the normalization of data is actually needed.

In conclusion, the new data platform presented emerges as a potential solution for address normalization. However, it is important to note that in order to maximize its adaptability and potential applications, it is crucial to consider the previously discussed additional developments. These future implementations could significantly expand the capabilities of the data platform, paving the way for broader utilization in diverse scenarios.

Appendix A

Code

```
{
  "result": {
    "verdict": {
      "inputGranularity": "PREMISE",
      "validationGranularity": "PREMISE",
      "geocodeGranularity": "PREMISE",
      "addressComplete": true,
      "hasInferredComponents": true
    },
    "address": {
      "formattedAddress": "Corso Duca degli Abruzzi, 24, 10129 Torino
    TO, Italia",
      "postalAddress": {
        "regionCode": "IT",
        "languageCode": "it",
        "postalCode": "10129",
        "administrativeArea": "TO",
        "locality": "Torino",
        "addressLines": [ "Corso Duca degli Abruzzi, 24"]},
      "addressComponents": [
        {"componentName": {
            "text": "Corso Duca degli Abruzzi",
            "languageCode": "it"
          },
```

```
            "componentType": "route",
            "confirmationLevel": "CONFIRMED"
          },
          {
            "componentName": {
              "text": "24"
            },
            "componentType": "street_number",
            "confirmationLevel": "CONFIRMED"
          },
          {
            "componentName": {
              "text": "Torino",
              "languageCode": "it"
            },
            "componentType": "locality",
            "confirmationLevel": "CONFIRMED"
          },
          {
            "componentName": {
              "text": "Italia",
              "languageCode": "it"
            },
            "componentType": "country",
            "confirmationLevel": "CONFIRMED"
          },
          {
            "componentName": {
              "text": "TO",
              "languageCode": "it"
            },
            "componentType": "administrative_area_level_2",
            "confirmationLevel": "CONFIRMED",
            "inferred": true
          },
          {
            "componentName": {
              "text": "Torino",
```

```
            "languageCode": "it"
          },
          "componentType": "administrative_area_level_3",
          "confirmationLevel": "CONFIRMED",
          "inferred": true
        },
        {
          "componentName": {
            "text": "10129"
          },
          "componentType": "postal_code",
          "confirmationLevel": "CONFIRMED",
          "inferred": true
        }
      ]
    },
    "geocode": {
      "location": {
        "latitude": 45.0624757,
        "longitude": 7.6623485
      },
      "plusCode": {
        "globalCode": "8FQ93M66+XW"
      },
      "bounds": {
        "low": {
          "latitude": 45.0624757,
          "longitude": 7.6623485
        },
        "high": {
          "latitude": 45.0624757,
          "longitude": 7.6623485
        }
      },
      "placeId": "ChIJR22IE3FtiEcRoCXXwNR3T14",
      "placeTypes": [
        "street_address"
      ]
```

```
100       }
101     },
102     "responseId": "e61b3d15-9a2c-492d-91b2-e1dcd09a07e3"
103   }
```

Listing A.1: Example of Google Address Validation JSON response

Bibliography

[1] Wing Shing Wong and Mooi Choo Chuah. «A hybrid approach to address normalization». In: *IEEE Expert* 9.6 (1994), pp. 38–45 (cit. on p. 2).

[2] Daniel W Goldberg, Jennifer N Swift, and John P Wilson. *Address standardization*. Tech. rep. Technical report 12. Los Angeles, CA: GIS Research Laboratory, University of . . ., 2014 (cit. on p. 3).

[3] Precisely. *Trillium Software's data quality*. URL: `https://www.precisely.com/about-us/trillium-software` (cit. on p. 21).

[4] IBM. *IBM*. URL: `https://www.ibm.com/products/aix` (cit. on p. 26).

[5] RedHat. *RedHat*. URL: `https://www.redhat.com/en` (cit. on p. 26).

[6] Wikipedia. *Wikipedia*. URL: `https://en.wikipedia.org/wiki/Secure_Shell` (cit. on p. 28).

[7] Trillium Software System. *Control Center*. URL: `https://docs.precisely.com/docs/sftw/trillium_series7/7.16/en-us/pdf/trillium-software-system-v7-16-00-control-center.pdf` (cit. on p. 29).

[8] *La scuola dei dati*. URL: `https://www.yimp.it/il-test-di-non-regressione/` (cit. on p. 35).

[9] Google. *Google Maps Platform*. 2022. URL: `https://developers.google.com/maps/documentation/address-validation/overview` (cit. on p. 48).

[10] Oracle. *Oracle Cloud Infrastructure Platform Overview*. 2021. URL: `https://www.oracle.com/a/ocom/docs/cloud/oracle-cloud-infrastructure-platform-overview-wp.pdf` (cit. on p. 48).

[11] Oracle. *Using Oracle Autonomous Database Serverless*. Anno. URL: https://docs.oracle.com/en/cloud/paas/autonomous-database/serverless/adbsb/autonomous-intro-adb.html#GUID-8EAA5AE6-397D-4E9A-9BD0-3E37A0345E24 (cit. on p. 49).

[12] Oracle. *Oracle*. URL: https://docs.oracle.com/en/database/oracle/oracle-database/12.2/adjsn/using-PLSQL-object-types-for-JSON.html#GUID-F0561593-D0B9-44EA-9C8C-ACB6AA9474EE (cit. on p. 57).

[13] Google. *Google Maps Platform*. 2022. URL: https://developers.google.com/maps/documentation/address-validation/understand-response (cit. on p. 57).