Politecnico di Torino

POLITECNICO DI TORINO

Master's Degree in Computer Engineering
Data Science

Master's Degree Thesis

Design and Implementation of an application Data Lake to support Repair Process in Baker Hughes

Supervisors

Prof.ssa Elena Baralis

Tutor Francesco Tamberi

Candidate

Matilde Pulidori

October 2021

A Nonna Checca

Acknowledgements

Arrivata a questo punto, mi sembra tutto incredibile. Non solo perché la fine di questo percorso è anche il traguardo che ho sempre avuto paura di non riuscire a raggiungere, ma anche perché nel frattempo ho avuto la fortuna di vivere tutte quelle emozioni che hanno reso il tragitto fino a qui uno dei più intensi e importanti della mia vita. In questi anni sono cresciuta, mi sono conosciuta di più, ho incontrato persone nuove e ho coltivato gli affetti di una vita, ho imparato e ho scoperto con una fame che non pensavo di possedere.

I miei ringraziamenti più sinceri vanno ai miei genitori, Antonella e Stefano, che per primi hanno sempre lottato per aprire davanti a me la porta delle possibilità, permettendomi di scegliere cosa studiare, come e dove studiarlo, sostenendomi anche nei momenti più duri, dandomi la forza di non arrendermi di fronte alle difficoltà. Insieme a loro, ringrazio Irene, mia sorella, la mia compagna di vita e la persona più importante per me.

Ringrazio Francesco Tamberi e Marco Perrone, i colleghi con cui ho svolto questo progetto di Tesi. Mi hanno dato fiducia, guidandomi e insegnandomi più di quello che pensavo fosse possibile. Con loro ho affrontato tutta la realizzazione di questo lavoro, con i suoi innumerevoli imprevisti, ma soprattutto ho condiviso con loro la gioia di aver portato a termine questo progetto. Estendo il mio ringraziamento anche a Baker Hughes e alle persone che ho conosciuto con quest'esperienza, per avermi dato modo di mettermi in gioco e contribuire nel mio piccolo in un'iniziativa cruciale.

Inoltre, ringrazio la mia Relatrice Elena Baralis, per avermi fatto conoscere e appassionare alla sua materia nel miglior modo possibile e avermi saputo dare consigli indispensabili nella realizzazione di questa Tesi.

Ci sono poi un'infinità di persone che mi sento di dover ringraziare di cuore, per aver fatto parte della mia vita durante questi anni di studio.

La prima persona, quella che mi ha dato il coraggio di intraprendere questo tortuoso percorso, che sento mi abbia dato quella spinta finale per convincere me stessa che dovevo buttarmi, inseguire ciò che mi piaceva e cambiare corso di studi, è stato Rodolfo: anche se le nostre strade si sono separate da molto tempo, non posso non serbargli il grazie più significativo.

Ringrazio le mie amiche di sempre, Camilla e Flavia, per volermi bene come se avessimo lo stesso sangue, per essere i miei fari ovunque io sia, la mia casa e il posto in cui mi diverto di più, per avermi vista arrivare fino a qui, attraverso ogni fase della mia vita.

Ringrazio Gabriele, il mio primo compagno di studi a Ingegneria Informatica, per non aver mai lasciato la mia mano in questi anni, accompagnandomi nei periodi più bui e più difficili che abbia mai passato, assicurandosi sempre che non fossi lasciata indietro. Con lui ho condiviso esami, progetti universitari, paure, gioie, e non potevo trovare un amico migliore con cui condividerle.

Ringrazio Bibi, la persona che Torino mi ha regalato come compagna di avventure, l'amica insostituibile che ho avuto la fortuna di aver sempre accanto, la persona che non ha mai esitato un attimo a farmi sentire a casa.

Ringrazio Matteo, con lui ho vissuto gran parte di questo percorso, lo ha reso speciale e senza di lui non sarei la ragazza che sono adesso.

Ringrazio poi tutti i miei amici più cari: Claudia, Valerio, Simona, Elena, Silvio, Letizia. Non posso escludere da questi ringraziamenti anche i miei compagni di corso, che hanno vissuto insieme a me lezioni, progetti, esami: Riccardo, Francesco, Matteo, Vito. Con loro, allargo questo pensiero a tutte le altre persone che ho avuto occasione di incontrare al Politecnico.

Ringrazio tutta la mia famiglia e specialmente le mie nonne, Anna e Francesca, per avermi sempre profuso l'affetto più puro e sincero che conosco. In particolare, dedico questo momento alla mia Nonna *Checca*, la luce che mi manca ogni giorno, che se ne è andata in silenzio, lasciandomi il ricordo dell'allegria che ha sempre emanato.

Infine, ringrazio il Politecnico di Torino e tutti i professori che ho conosciuto in questi anni, per aver sempre cercato di trasmettere, attraverso l'insegnamento, la passione di cui ogni sfida ha bisogno, e con cui, anche io, ho imparato ad affrontare le mie.

Summary

Digital Revolution leads companies to invest and focus their attention on leveraging the potential of digital technologies. Baker Hughes Company, an Energy Technology company, is driving its digital transformation investing in new technologies or improving the existing ones.
The objective of this thesis project is the study and the development of an application Data Lake to support the Repair Service Team of Baker Hughes. Repair One Portal (ROP) is a custom web application that tracks Repair Process and most of its data are stored in a relational database.
Data Lake is an emerging alternative system to the Data warehouse. It answers the need of extracting information from different kinds of data in a flexible and not expensive way. It decouples operational database from the repository for the analysis, maintaining a raw data copy on the Data Lake instance.
The implemented Data Lake provides a report layer for Repair data, a starting point for data integration with other data sources, improves logics and information accessibility, allowing the monitoring of an important area of the company.
The thesis aims to describe the case study, the choices that drove the solution design phase, the challenges faced during the implementation, with the ambition of providing a solution model for similar initiatives.

Table of Contents

List of Figures

Acronyms

ACID
Atomicity, Consistency, Isolation, e Durability

AI
artificial intelligence

API
Application Programming Interfaces

AWS
Amazon Web Service

BH
Baker Hughes

BI
Business Intelligence

CF
Check Forms

ComOp
Commercial Operator

CRUD
Create Read Update Delete

CTO
Chief Technology Officer

CWD

Customer Want Date

DB

Data Base

DBA

Data Base Administrator

RDBMS

Data Base Management System

DL

Data Lake

DDL

Data Definition Language

DSS

Decision Support System

DWH

Data Warehouse

ELT

Extract Load Transform

ETL

Extract Transform Load

ETO

Engineered To Order

ER

Entity-Relationship

ERP

Enterprise Resource Planning

FnP

Forecast & Planning

HDFS

Hadoop Distributed File System

ILM

Information Lifecycle Management

IoT

Internet of Things

ITR

Inquiry to Remittance

JSON

JavaScript Object Notation

JV

Joint Venture

KDD

Knowledge Discovery in Database

KPI

Key Process Indicator

LM

Land Marine

LNG

Liquified Natural Gas

MD5

Message Digest 5 - Widely used hash function producing a 128-bit hash value

MIT

Massachusetts Institute of Technology

ML

Machine Learning

MS

Micro Service

NoSQL

Non-SQL - Non tabular Database that stores data differently than relational tables/SQL relational databases

OBIEE

Oracle Business Intelligence Enterprise Edition

OLAP

On-Line Analytical Processing

OLTP

On-Line Transaction Processing

PM

Project Manager

PO

Purchase Order

RDS

Relational Database System

RDBS

Relational Data Base System

RDBMS

Relational Data Base Management System

RFI

Request For Induction

RFQ

Request For Quotation

ROP

Repair One Portal

SOW

Scope Of Work

SSC

Service Shop Cockpit

SQL

Structured Query Language

TAT

Turn around Time

TCP

Transmission Control Protocol

TPS

Turbomachinery & Process Solutions

WAL

Write-Ahead Logs

Chapter 1

Introduction

1.1 Digitalization in companies

Digitalization is the use of digital technologies to change a business model and provide new revenue and value-producing opportunities; it is the process of moving to a digital business [1]. The increasing widespread adoption of digital technologies in the last decades, like Big Data, Internet of Things (IoT), Cloud Computing, and Mobile Computing, changed the way people interact and how business is organized. Indeed, productivity growth in developed countries is strongly impacted by how much these economies invest in technological innovation [2].

In particular, digital technologies provide a significant contribution to make an organization rise and remain competitive. Companies that want to succeed in the digital age must find the best technologies for their needs (*digital disruption*), design a new business model allowed by digital technologies (*digital business*) and finally embrace the change of the company architecture (*digital transformation*) [3].

Manufacturing industry, for example, is the basic industry of all the modern economic systems and is now living the era of Industry 4.0. Investing in Internet of Things (IoT), Artificial Intelligence and Machine Learning, 3D Printing and Augmented Reality, gives a big contribution in reducing human error and provides various advantages, like [4]:

- Possibility to choose location based on technical capabilities and proximity to consumer demand, rather than on low-wage regions
- Quicker production
- Possibility to do predictive and preventive maintenance, which results in lower downtime and less capital expenditure, over time
- More efficiency and optimization, thanks to high-connectivity between machines, shared data and advanced analytics

- Better archiving and search capability

Another interesting example is AirBnB. AirBnB can be identified as one of the major cases that guided the revolution in the accommodation services. Connecting hosts and guests facilitates a trust-worthy communication and the attractiveness of the experience. While hotels invest in physical location, staff training and integrated services, AirBnB is highly data-driven and its success was mainly due to the fact that investments were concentrated on the platform as social business, in order to create the best match between guests and hosts, using smart interfaces and algorithms [5].

To better explain the concept of digitalization, it's necessary to highlight its differences with respect to two other similar concepts: digitization on one side, digital transformation on the other side. *Digitization* is the process of changing from analog to digital form [6]. When talking about digitization, it is the single information that is transformed: no more dealing with analog or 'paper-based' information is the objective. Instead, *Digitalization* changes a process, and consequently the people's job around it. For this reason, digitalization of a process need to be followed by the acquisition of digital skills [7]. *Digital transformation* of a company not only requires to undertake multiple digitalization projects, but to embrace a custom-oriented change competency as one of the core characteristic of the organization [8]. As Jason Bloomberg from Forbes says:

> "In the final analysis, therefore, we digitize information, we digitalize processes and roles that make up the operations of a business, and we digitally transform the business and its strategy [8]."

1.2 The Big Data Challange

> "Big data is *high-volume*, *high-velocity* and *high-variety* information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making [9]."

Big Data is one of the main characters in the era of digitalization. The above definition means that Big Data represents large amount of data coming continuously from various sources. Volume, Velocity and Variety are called the three Vs of Big Data [10]:

- **Volume**: High volumes of low-density and unstructured data, from tens of terabytes to hundreds of petabytes.
- **Velocity**: Fast rate at which data is received and streams into memory, often in real time or near real time.

- **Variety**: Availability of many types of data. Traditionally, data are structured and fit relational databases. With big data, data comes in new unstructured or semi-structured data types, that requires additional pre processing and support metadata.

The digitalization phenomenon encourages the rise of Big Data, and consequently the growing need to leverage its potential. Not surprisingly, a lot of companies investing in digital technologies find theirselves with big quantities of operational data to manage and from which hopefully obtain information. The process that includes the collection, the analysis and the elaboration of big amounts of data with the objective of extracting information is called *Big Data Analytics*. In this context, the Knowledge Discovery in Database (KDD) explains the stages that usually drive data analysis and data elaboration. The Figure 1.1 shows the KDD procedure:

1. **Selection**: Data of interest are selected from the Database
2. **Processing**: Data are cleaned and integrated
3. **Transformation**: Data are organized in a useful form for analysis
4. **Data Mining**: Chosen algorithms analyse transformed data to find patterns
5. **Interpretation**: Analysis allow domain-experts to do evaluations and to extract significant knowledge

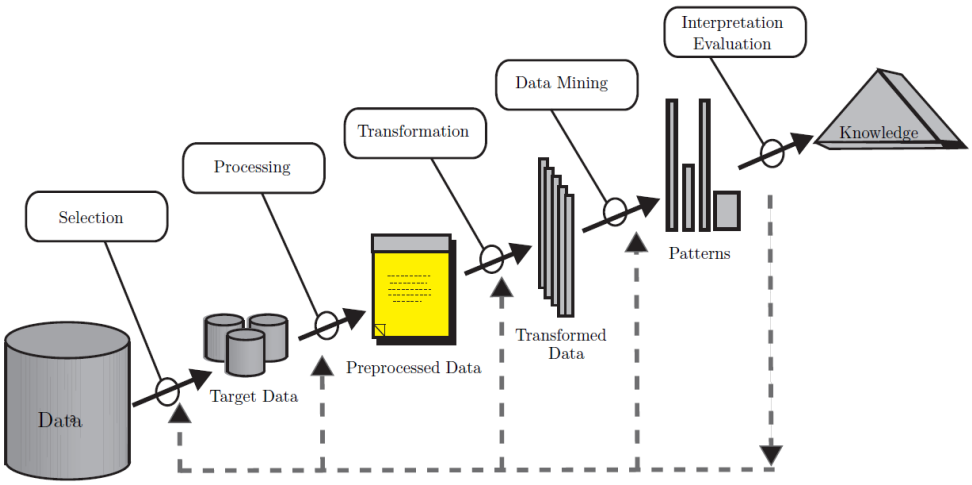


Figure 1.1: Knowledge Discovery Process [11]

Big Data Analytics employ advanced and complex methodologies, such as inferential statistics and non-linear systems to discover patterns, correlations and

dependencies in data, to predict outcomes and behaviours. Furthermore, in order to analyse the company status and to make faster and better decisions, it is crucial to use Decision Support Systems (DSSs), like a Data Dashboard, which is an information management tool that visually tracks, analyses and displays Key Performance Indicators (KPIs) and other significant metrics. A KPI is a performance measurement evaluating the success of a organization or a particular activity [12] and DSSs are most commonly employed in forecasting and planning, identification of critical area, financial transparency, definition and implementation of successful strategies.

Some of the most impactful benefits that a company can achieve by adopting Big Data Analytics solutions are [13]:

- **Transactional benefits**: Improvements in efficiency and productivity
- **Transformational benefits**: Reduction of operating-costs and enhancements of the returns on financial assets, enabling new opportunities
- **Strategic benefits**: Possibility to align IT with business strategy
- **Informational benefits**: At the same time, there are also benefits obtainable in terms of data accuracy, data access and data management

Of course, business fields, data types and data storage are different from a company to another. So, there is no unique-and-right way to deal with information available, also because the strategy is driven by the goals of the considered organization.

1.3 A new solution for Big Data

To satisfy the necessity of exploring data and extracting information, the solution widely adopted so far is Data warehouse (DWH). A DWH can be defined as a system used for reporting and data analysis. Data are extracted from the source, transformed in a pre-defined structure, and finally loaded in the Data warehouse (ETL process, see Figure 1.2). This approach is useful to provide a single model for all data (regardless of the source), maintain data history, quality, security and governance.

If we look at the business field rather than the academic one, a new emerging technology worth introducing is Data Lake. The term "Data Lake" first appeared in 2010, when James Dixon (founder and CTO at Pentaho) described it in this way:

> "The contents of the Data Lake stream-in from a source to fill the lake, and various users of the lake can come to examine, dive in, or take samples [15]."

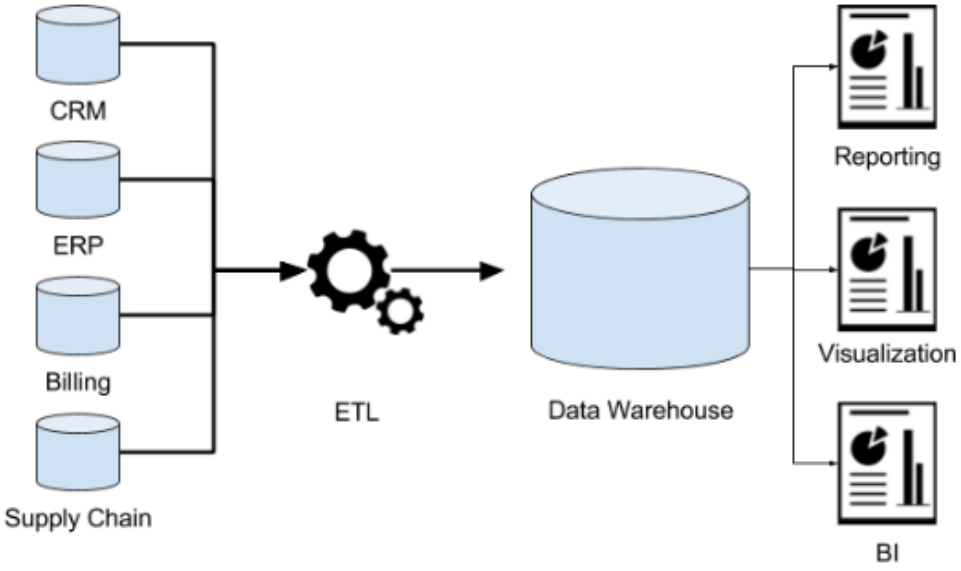


Figure 1.2: DWH Schema [14]

This metaphoric description gives only a generic idea of what a Data Lake is. More in details, a Data Lake is a repository that can contain raw, unstructured and multi-structured data, potentially coming from different sources, and once there, data are available for analysis by everyone in the organization. Moreover, transformation step is performed only when required, avoiding expensive data pre-processing activities. As shown in Figure 1.3:

- Data source can be of different types: relational DBs, json files, text files, etc.
- Transformation phase only occurs after the data are loaded in the Data Lake

Possibility to have raw data, to store unstructured data and to perform transformation only at query-time, makes Data Lake a competitive and flexible technology to be adopted in various situations. Queries usually are less complex than in DWH, but more custom-oriented. All of this permits the exploration of data that would not be engaged if using DWH technology, due to the structured and processed requirement on data and to the tendency

1.4 Data Lake to support Repairs process

The aim of this work is to design and implement an application Data Lake to monitor Repair Process in Baker Hughes (BH) Company.

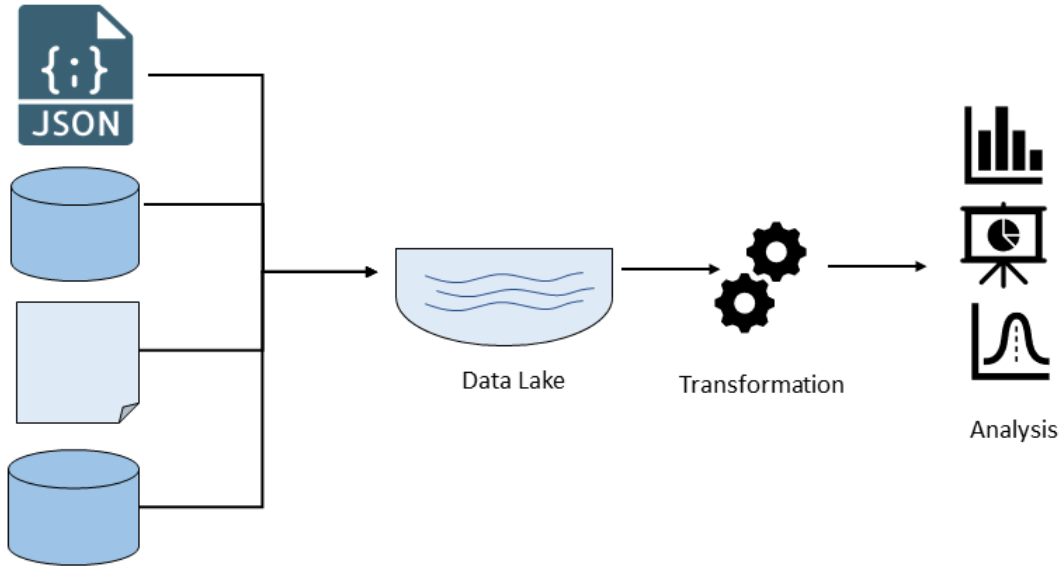


Figure 1.3: DL Schema

Baker Hughes is an American International Company in energy industry, and is driving its digital transformation, investing in new technologies and improving the existing ones. BH Repair Team tracks Repair Inquiry-To-Remittance (ITR) Process leveraging on an internal web-application, called Repair One Portal. The application is divided in four big modules, representing the different macro-areas of the repair-cycle. Consequently, application data are stored in multiple DBs: the storage is not only based on the module they refers to, but also on the data type and data structure. Different data granularities make it necessary for the consumption an intermediate step, where the data are modelled to be retrieved at the same *logical* level.

This thesis project answers the need of the business to consolidate data in a single point for analytical purposes, in order to make specific reports on activities status and evaluate major KPIs without affecting Database performances. Implementing a Data Lake in this way, enables a decoupling layer between application DBs and repository for data analysis, making the analysis flexible, expandable, dynamic, and more custom-oriented.

1.4.1 Data Lake Realization

The core of this thesis is to design and implement from scratch a suitable Data Lake solution for this case study, integrating existing technologies, such as Relational Database System (RDS) on Amazon Cloud, considering the data dimension to deal

with, improving the previous algorithms applied on data aggregations in order to calculate required KPIs. Auxiliary initiatives started from this work, to answer the challenge of data access for the users and data integration with other domains.

1.4.2 Thesis organization

To start, this thesis explains Data Lake technology, comparing it with Data Warehouse solution. After that, a description of the context follows, from Baker Hughes Company, to the Repair Services and Process, contextualizing the business needs driving this project.
The design and implementation of the Data Lake is the central focus of this study. For this reason, an expensive space is given to analyse in details the solution architecture and to explain the implementation method.
Some final considerations highlights the obtained results, opening the view to a wider horizon for this work.

Chapter 2

Data Analytics

2.1 Data warehouse

There are two types of operation in data processing: transactional and analytical [16].

- **On-Line Transaction Processing** (OLTP) refers to systems that manage transaction-oriented applications. Usually, *transaction* means *database transaction*, i.e. atomic change of state, like CRUD (Create, Read, Update, Delete) operations. Classical examples of OLTP systems are financial transaction systems, retail sales, order entry.
- **On-Line Analytical Processing** (OLAP) is associated to complex queries, for the purpose of business intelligence or reporting, like multidimensional DBs and aggregations.

Typically, an organization has Relational Data Base Systems (RDBSs), with structured operational SQL data and ACID characteristics, to store transactional records. In this scenario, Data Warehouse emerged to perform analytical operations.

2.1.1 Data warehouse concepts

Data warehouse is a decision support Database kept separated from operational Databases of the company. DWH maintains data separately for multiple reasons:

- **Performances**: Complex data search affects the quality of operational transactions, for this reason OLAP has different access method with respect to OLTP.
- **Data management**: With DWH it is possible to have historical data, to avoid data inconsistency and consolidate data from different sources.

The Data Warehouse technology centralizes and consolidates big amount of data from multiple sources; then, its analytical capabilities enable a company to make valuable business insights from their data, in order to make better decisions. Moreover, DWH builds a historical record over time that data scientists and business analysts can use [17]. It's important to keep in mind that Data warehouse focuses on reading data: data cannot be modified and is usually accessible in a read-only way.

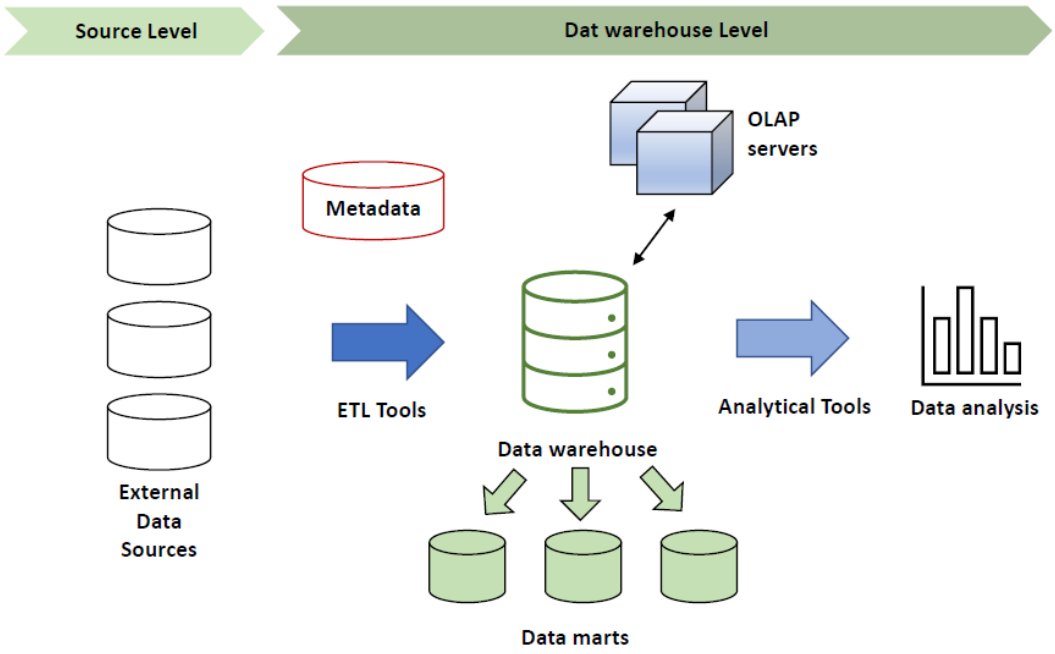


Figure 2.1: DWH Architecture

As shown in Figure 2.1, Data Warehouse architecture includes:

- **External Data Sources**: Transactional data from OLTP relational DBs, texts, excel files
- **ETL Tools**: Extraction, Transformation and Loading solution to prepare data to entry in DWH [17]
  - **Extraction** is the data acquisition from the sources
  - **Transformation** is the data cleaning and data conversion to the operational format of the Data warehouse
  - **Loading** is when data is stored in the Data warehouse

- **Metadata**: Information necessary for the DWH to work, describing how data is organized and monitoring data behaviour
- **Data warehouse**: Where all the transformed OLTP data are loaded
- **Data marts**: Small Data warehouses focusing on a specific area or company department (it can be fed from the central DWH or directly from the sources)
- **OLAP Servers**: Servers where the aggregations on data are performed
- **Analytical Tools**: Statistical analysis, reporting, and data mining capabilities [17]
- **Customer Tools**: For visualization and presentation of data to business users [17]

Data warehouse leverages on OLAP Servers, a Database technology optimized for complex query execution. A OLAP Server aggregates OLTP data in structures that allow sophisticated analysis: usually, OLAP data is stored in cube format (multidimensional array of data that provide rapid access to data for analysis), rather than tabular format [18].

2.1.2 Data warehouse challenges

Nowadays, Data warehouse is the most prevalent solution for analytical data. Even if really powerful, Data warehouse introduces some challenges.

- **Data Types**
  Data warehouse stores structured organizational data, excluding all data coming from other sources such as sensors, logs, social media data [19].
- **Processing**
  Data warehouse has data clearly defined and organized. Metadata are applied before data is written and stored: this is called *schema-on-write* process and the ETL Tool is in charge of this [19].
- **Storage**
  Data scientists have to carefully analyze data before loading them in the Data warehouse: data has to concretely answer to a business need, because big-volumes storage can be expensive [19]. Data warehouse is optimized for query execution, not for large-volumes storage.
- **Flexibility**
  Data introduced into DWH must have a predefined structure. This is useful for correctly answering specific business questions, but it is insufficient for

rapidly-evolving needs of a corporation. The effort to adapt a Data warehouse and the associated ETL tool to new business questions is significantly high [19].

What if not structured data carry significant information? And, given not structured data, it would be risky to apply metadata before loading phase, in terms of data management and comprehension? Or maybe, could it be a good idea to keep the metadata stored as well in the OLAP technology? What if a company needs to engage large volumes and different types of data in the analysis? And what if the type of investigation on data changes fast over time?

2.2 Data Lake

In the Big Data era, NoSQL Databases appeared as a solution for data organized in any other way than SQL tabular format of Relational Databases. NoSQL approach provides simplicity of design: usually data are saved in a key-value pair format [20]. Also, wide-columns[1], graphs or documents data formats are allowed [21] (see Figure 2.2).

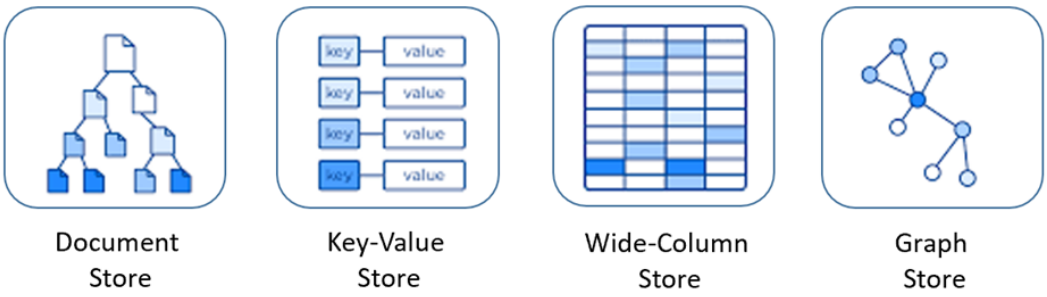


Figure 2.2: Data models of NoSQL Databases [22]

The rapid spread of NoSQL and other unstructured data sources, with the high-volumes and high-velocity characteristics of Big Data, all together introduce new challenges for data analysis. Data Lake concept emerged from a desire to address these challenges: extracting information from all the data of an organization, with no data types or data sources being excluded, and always ensuring elasticity to adjust data analysis in tandem with business demands.

[1]having a table with rows and columns, names and format of the columns can vary from row to row in the same table

2.2.1 Data Lake architecture

Data Lake can be imagined as a huge repository where data - structured, unstructured, semi structured - flows from its sources in their original format. Understanding data nature is delegated to the data consumer at the moment of the retrieval, in order to perform transformations, according to the business needs [16] (see Figure 2.3). Data are not classified when stored, enabling the investigation even on not predictable topics.

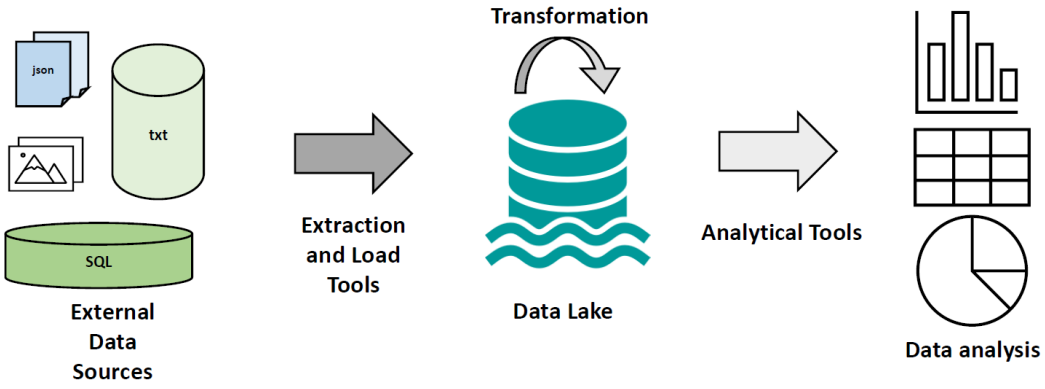


Figure 2.3: Data Lake schema

Data Lake uses a flat architecture: each element has its unique identifier and metadata associated. It's worth saying that when data are loaded in the DL, also metadata are added to the DL, enabling a *schema-on-read* metadata mechanism: Data Lake maintains both data and metadata, and applies metadata only at read-time. Since there is not a predefined schema, metadata management is a very critical aspect of a DL [23].

Data Lake architecture can be either based on "data structure" or on "data lifetime". The data structure option divides the Data Lake in three tiers: intake data tier, management tier -where all the transformations and analytics are done-, and consumption tier -where data are ready for the final user-. On the other hand, dividing data based on their lifetime, introduces a classification in three categories: data that has lifetime less than 6 months (or another arbitrary short period of time), older but still active data, archived data [24].

The Data Lake architecture based on data structure is the most used. There is no a unique model to follow to design a Data Lake, this thesis generally relies on the model based on what Pradeep Pasupuleti and Beulah Salome Purra claim in their book *Data Lake Development with Big Data* [26]. Other descriptions of Data Lake architecture are similar: maybe, calling with different names the same

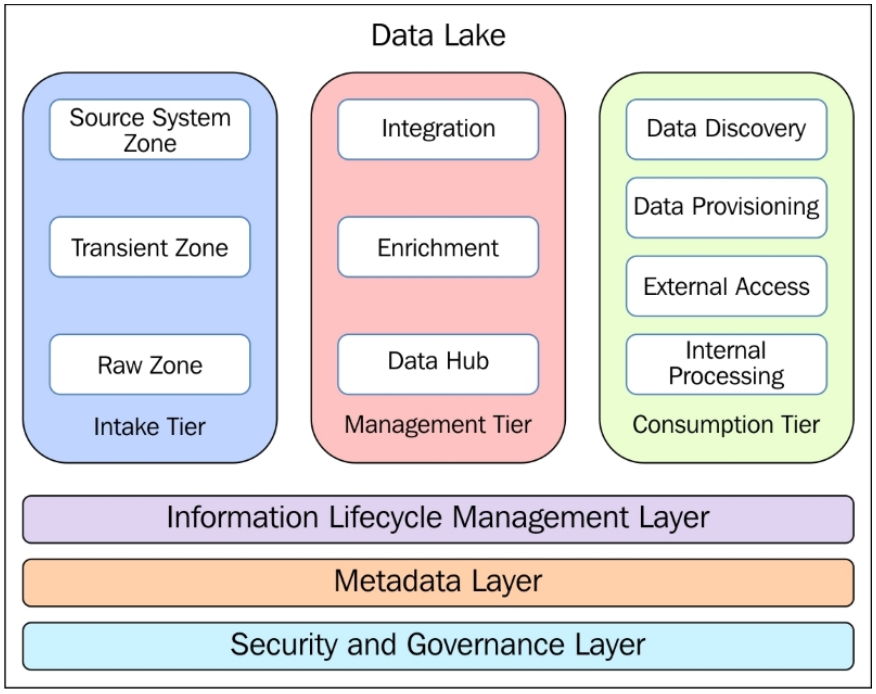


Figure 2.4: Data Lake architecture [25]

element, or using the same name to indicate an element, with small differences in characteristics. Figure 2.4 shows the internal architecture of a Data Lake, based on the authors' opinion. It can be immediately noticed that there are three horizontal layers running over every DL tier:

- **Metadata Layer**
  This is the most important layer in the Data Lake. Metadata indexes information, so that users can explore metadata before accessing data contents. Furthermore, metadata provides important information about the significance of the data. The Metadata layer defines the structure for raw data and dictates their organization, it tracks the schema evolution of a file/record, allowing associations between entities and facilitating browsing and searching. [25].

- **Security and Governance Layer**
  It is in charge of governing the privileges on data access, data definition and modification. It ensures the appropriate access control, it detects and prevents cyber attack, block unauthorized access, takes care of sensitive data [25].

- **Information Life-cycle Management Layer (ILM)**
  ILM layer rules what can or cannot be stored in the Data Lake, defines the strategy and policy on data value and on data permanence in the Data Lake,

since older data can lose in value over long period of time [25].

For what it concerns Data Lake vertical tiers, as previously said, data is passed from a tier to another in the following order:

1. **Intake Tier**
   This tier deals with connectivity and acquisition of data from the external sources (*source system zone*). Using a file-based storage, it checks the file size and the record counts, it performs validity check and quality check, all in the *transient zone*. Finally, it stores raw data with metadata associated in the *raw zone*, and data governance is applied as well on raw data, preparing the migration to the next tiers of the DL [25]. Not surprisingly, Intake Tier is commonly called Raw Data Tier, for simplicity.

2. **Management Tier**
   Data flows in Management tier, through three steps:

   - **Integration**: Common transformations are applied on raw data, in order to standardize and clean data for the consumers. Here, different processes perform data validation, quality and integrity checks, and track logs about these checks.
   - **Enrichment**: Automated business rules for enhancement, augmentation, classification and standardization can be applied to the data.
   - **Data Hub Zone**: The final storage for clean and processed data is done with a push in relational Databases or NoSQL Databases (see Figure 2.5).

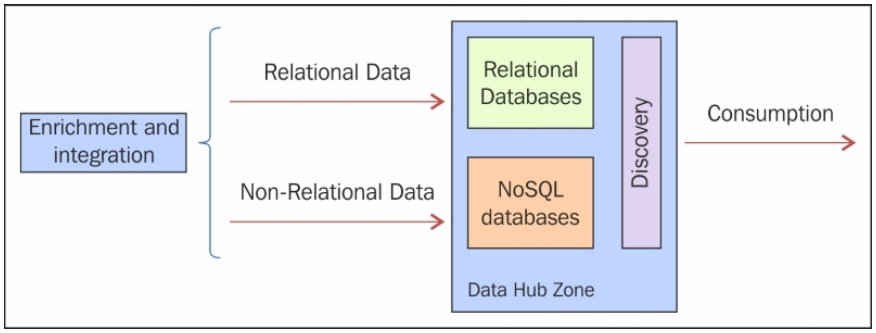


Figure 2.5: Data Hub Zone in Management Tier [25]

Integration and Enrichment are performed in a file-based Hadoop Distributed File System (HDFS), more advantageous since data are most of the time schema-less: it loads data faster, leaving the structure flexible and modifiable. In Management tier, metadata are attached to the related object, tracking the progresses and movements from a step to another [25].

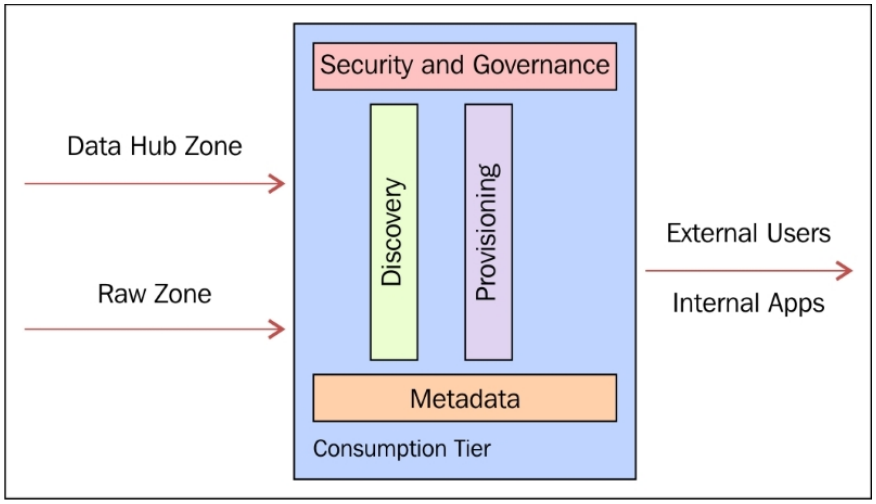


Figure 2.6: Consumption Tier [25]

3. **Consumption Tier**
   Data is made ready and available through this tier, for external consumption for visualization, analysis and other scopes. There is a *data discovery* mechanism that leverages the metadata and make it possible a flexible, self-driven exploration for the users with an interface. Consumers can then retrieve data in the *data provisioning zone* (see Figure 2.6) [25].

2.2.2 Data Lake advantages

After understanding Data Lake context and its architecture, it's important to highlight its has numerous benefits.

- **Not expensive storage**
  Data Lake can scale horizontally to gain more space for data to be saved, remaining not expensive even if the space required increases. Using HDFS-based storage, makes it easy to add a new cluster when necessary [25].

- **Different data sources**
  As stated multiple times, a powerful feature of the Data Lake is admitting different data source types, without the structured data constraint: sensor data, logs, binary, photos, XML, and so on. This aspect enables quick integration of datasets, analysis of otherwise unexplored data, and, with the help of some tools (), Data Lake supports high-speed streams of data: it is able to acquire and to integrate high-velocity data in large-volumes [25].

- **Add a structure on top of all the data**
  Data Lake can be a single point where to apply a structure on multiple and different datasets, allowing the data to be handled in advanced analytics scenarios [25]. Moreover, the possibility to have raw data on DL, allows to manipulate it in disparate ways. Hence, Data Lake supports more easily customizable queries: transforming data only when required provides flexibility for data scientists that are familiar with the domain to perform custom-oriented type of analysis.

2.2.3 Data Lake challenges

Even if Data Lake answers to the DWH challenges providing great and not expensive solutions, it still has something to be improved. Data Lake lacks in security, metadata management and performances.

- Since DL is still maturing as a technology, for this reason its security robustness is as well an open research area [27].
- Usually, Data Lakes do not provide a specific metadata management mechanism: for this reason, many users don't know how to deal with metadata [16].
- Performances have not been tested: this comes from the fact that Data Lake main focus is the storage of large and various data, rather than how or why data is used, managed, defined, or secured [16].

After all, one of the major risk of Data Lake is the Data Swamp. Even Data Lake supporters agree with the fact that there are no procedures to avoid wrong, repeated or incorrect data to enter in the DL. If a control mechanism is missing, there's the possibility of realizing that some data is damaged when it's too late. This is something on which paying attention on, because Data Lakes can really be data-pollution-prone and the possibility of becoming a Data Swamp is forthcoming [16].

2.3 Comparative analysis: DWH vs DL

Data Lake has emerged as a new way of structuring the analysis process. In order to better understand differences, advantages, disadvantages between DWH and DL, a comparative analysis is following.

- **Data**
  DL allows different types of data (structured, semi-structured, unstructured, unprocessed), while DWH only admit structured and processed data.

- **Processing**
  In DWH architecture, data are Extracted, Transformed and Loaded, following the **ETL** schema: usually, transformation operations are performed before data is stored in the Data warehouse. Instead, in Data Lake architecture, data is firstly extracted and stored in the raw zone. Only after this, transformations are performed, and the transformation phase occurs in the Management tier, meaning in a Data Lake zone. Therefore, DL follows an **ELT** schema, as previously said in Section 1.3. Directly storing data in their raw format avoid the preprocessing and the transformation costs of Data warehouse systems [16].

- **Metadata**
  The processing on the data follows two different approaches with respect to metadata. DWH uses a *schema-on-write* approach, leveraging metadata when data are stored in the Data warehouse (at writing time). On the other hand, DL uses a *schema-on-read* approach: metadata are stored in the DL as well as data contents, and are employed in each tier only when reading data of interest [16].

- **Separation between OLTP and OLAP**
  DWH schema keeps transactional data in operational databases while complex queries run on the Data warehouse, dividing OLTP and OLAP operations. Instead, DL schema, usually having stored a copy of raw data (or data flowing directly from the sources) and running queries for analytical purposes on top of the Data Lake, combines OLTP and OLAP in the same solution.

- **Cost**
  DL can store big amount of data in a not expensive way, due to the file-based HDFS, and is often implemented on open-source frameworks. Instead, beyond the fact that DWH has storage constraints that make it onerous to maintain large volumes of data, DWH usually has high-licensing fees [16].

- **Flexibility**
  Changing Data warehouse design is possible, but it requires big effort and it is very time consuming; moreover, changes may affect in the long run maintenance costs. This is due to the fact that DWH has a highly structured definition and data management. At the same time, even if Data Lake doesn't have the formal structure of a Data warehouse, it allows developers and data scientist to quickly and simply configure models, queries, and apps [16].

- **Security**
  Being Data Lake a quite new technology, its security performances are still

monitored and studied. Data warehouse, on the other side, is older and is protected by a well-defined security.

- **Users**
  Data Lake is a suitable technology for data analyst and data scientists: to deal with Data Lake, it's necessary to be domain-experts and know how to make robust analysis on raw data. Instead, Data warehouse provides a more easily understandable data layer with a clearly defined structure, answering to highly-specific questions, making it more approachable to a wide range of company's roles, such as business analyst.

Comparison	**Data warehouse**	**Data Lake**
Data	Structured and processed data	Structured, semi-structured, unstructured, unprocessed data
Processing	ETL	ELT
Metadata	Schema-on-write	Schema-on-read
OLTP and OLAP	Separated	Together
Cost	More expensive (Licences)	Cheaper (Open-source frameworks)
Flexibility	Fixed configuration	Dynamic configuration allowed
Security	Matured	Maturing
Users	Business professional	Data Scientists (especially those familiar with domain)

Table 2.1: Comparison DWH vs DL

Chapter 3

Case Study: Repair One Portal

3.1 Baker Hughes Company

Baker Hughes is an energy technology company, with a long time business in the petroleum service industry. It was founded in 1907 and during its history, Baker Hughes has acquired and assimilated numerous oilfield pioneers. Today, Baker Hughes is again an independent company, after the fusion in 2016 and then the separation in 2019, with General Electric [28].
As a company, it has four operating segments to provide different products and services: Oilfield Services, Oilfield Equipment, Turbo-machinery & Process Solutions, Digital Solutions [29]. Turbo-machinery & Process Solutions (TPS) segment provides equipment and related services for mechanical-drive, compression and power-generation applications across the oil and gas industry and energy industry. TPS is mainly focused on designing, manufacturing, maintaining and upgrading rotating equipment across the entire oil and gas value chain [29]. An important area in TPS segment is the so-called Advanced Repairs, that provides repair and upgrade services [30] for Gas Turbines. In the Digital Technology corporate function, The Repair Digital Technology Team develops digital solutions for managing the Inquiry To Remittance (ITR) repair process, as well as supporting repair operations at Baker Hughes workshops.

3.2 Repair Process

Repair Process in Baker Hughes have multiple steps and actors. Broadly speaking, to start, Figure 3.1 shows the main phases of an entire Repair Life-cycle.

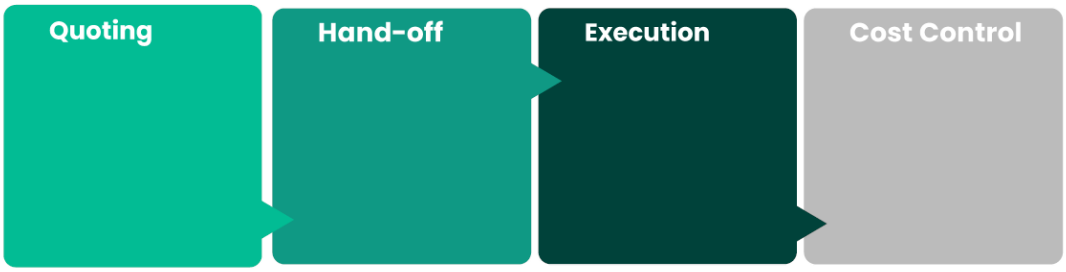


Figure 3.1: Repair Phases

3.2.1 Quoting

Every Baker Hughes customer has a Commercial Operator, called *ComOp*, in charge of taking care about the commercial relation with the Company. When a customer needs a repair performance, the ComOp prepares a Quotation: this process is called Request for Quotation (RFQ). The ComOp collects all the necessary information: customer data, machine model and items to be repaired, the workshop where to send the parts. The customer can express the preference that the repair is done in a specific workshop, which for the sake of simplicity is always referenced to as Shop. The choice must be motivated and successively approved. At that point, a Technical User verifies the configuration (repair activities, feasibility, spare parts); then, the ComOp makes a Quotation based on the configuration, providing a first idea of the repair costs, which - at later time - will have to be confirmed, after inspecting the real machine status. The ComOp sends the proposal to the Customer and asks the formal Customer approval, in order to finally close the RFQ (see Figure 3.2).

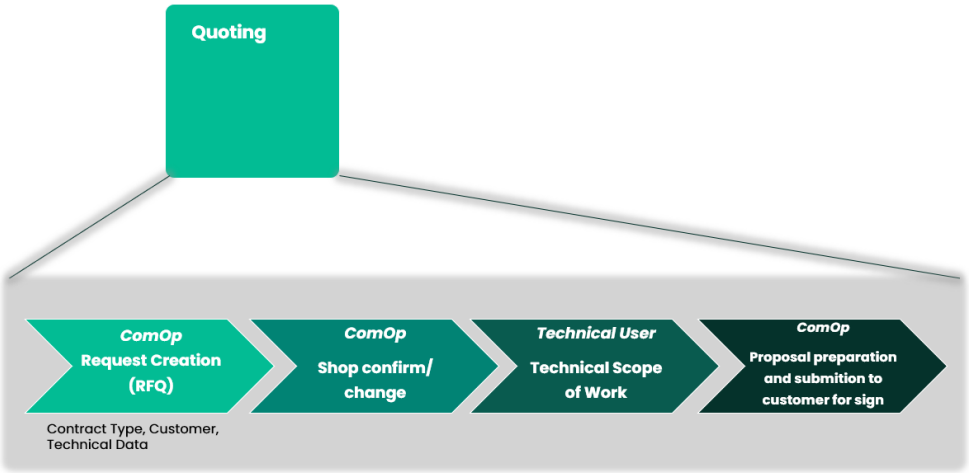


Figure 3.2: Quoting Phase

3.2.2 Hand-off

After having closed the Request for Quotation, the Request for Induction (RFI) process stars. RFI has multiple phases: Creation, Pre-Inspection, Post-Inspection and at Completion.
First, in the RFI Creation Phase, the Shop where to perform the Repair activities must be approved by a member of the Forecast and Planning (FnP) team. The Shop is approved following a Routing strategy: the best one is proposed based on Shop capability (the possibility to perform the repair activities needed) and Shop capacity (the possibility to complete the repair in the required time). Then, before items arrive on site, the Request proposal is detailed also with the Expected Arrival Date (when the machine items will arrive at the shop), the Due Date for Inspection (when the inspection on the machine item will be performed), with the costs: this is included into the Pre-Inspection RFI.
The machine items arrive at the Shop, then they are inspected. The inspection can reveal additional activities to be performed on the machine. If so, a Post-Inspection proposal is opened in order to reflect the actual situation and costs to be incurred (see Figure 3.3).

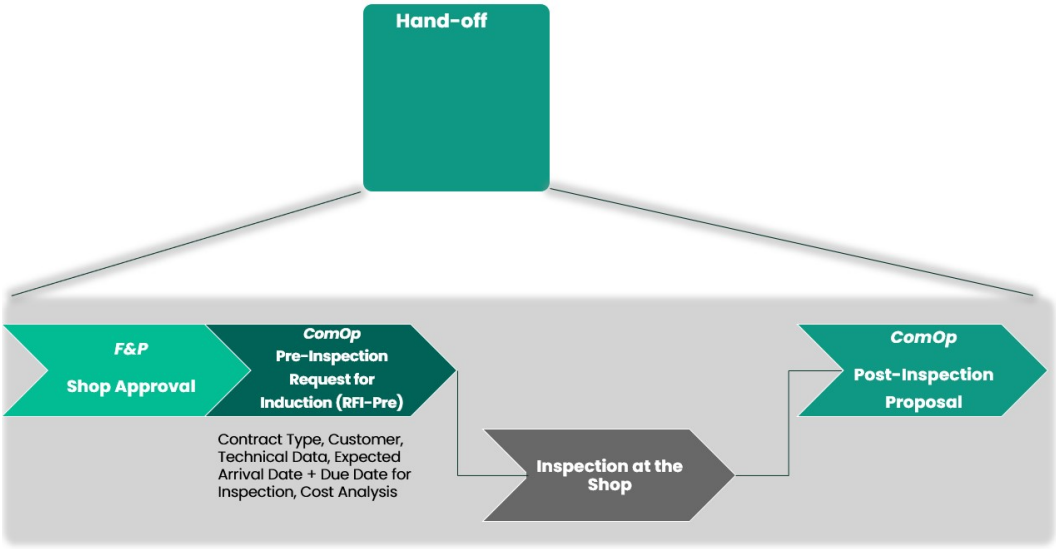


Figure 3.3: Hand-off Phase

Inspection

When the machine items physically arrive on site, all items are checked and inspected by Shop Operators. A Technical User creates the Inspection Work-Order and creates a document for the Inspection, called Check Form (CF), with the

general details about the RFI. As the inspection goes by, the Operators fill the Check Form with all the information of the check operations performed on the items. At the end of the inspection, a Foreman approves the Check Form document. At this point, the Technical user prepares a preliminary report on the real status of the items, prepares the Repair Work-Order and the Material Work-Order. The Planner team makes a cost analysis, a lead time analysis, completes the Work-Order and notifies the ComOp that the Inspection has been done (see Figure 3.4).

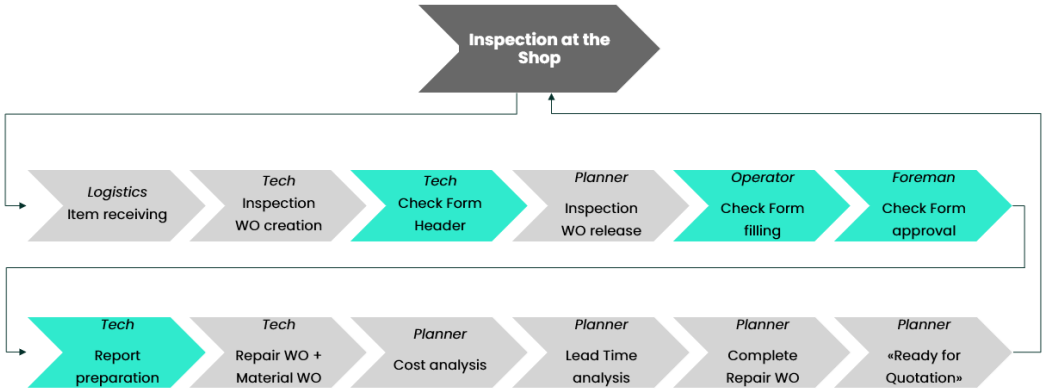


Figure 3.4: Inspection at the Shop

3.2.3 Execution

When the details about the Repair Request are updated, the ComOp takes care of the Post-Inspection proposal. A Post-Inspection RFI is created, linked to the Pre-Inspection one, with the changed costs and the required date of completion (also called Customer Want Date CWD) of the Repair. The Hand-off is signed between the ComOp and the Project Manager (PM) in charge of the Repair activities for that specific customer. This is called Post-Inspection RFI. With the "Ok to proceed", the Repair activities can be finally performed in the Shop (see Figure 3.5).

Repair

When the "Ok to proceed" is agreed, the core Repair procedure stars. The Planner Team updates Actual Start Date, checks the capitals availability, assigns the Shop resources and opens a Repair Work-Order. The Operators who perform the repairs fill the Check Form with the details of the done repair activities. Then, the Planner Team closes the Work-Order and the Quality Team does final tests. A Technical

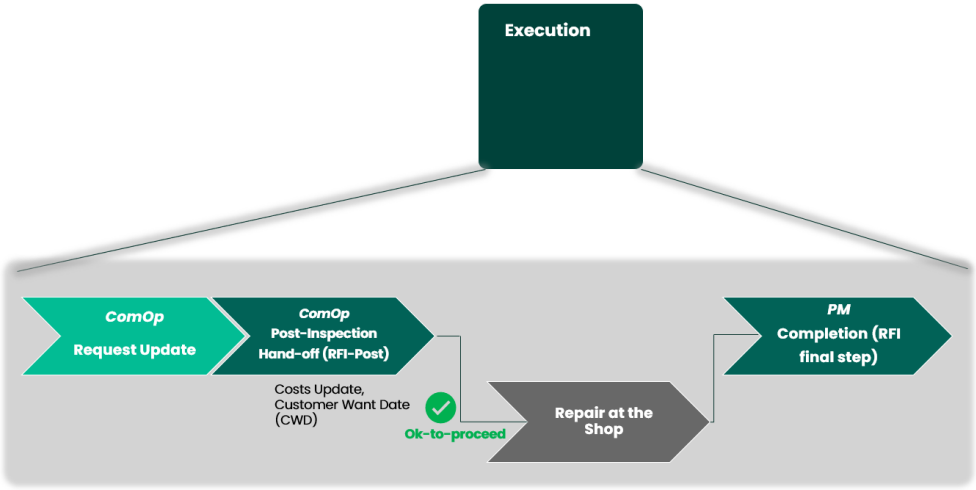


Figure 3.5: Execution Phase

operator prepares a final report of the entire Repair: what has been done, what has changed during the process, and so on 3.6.

To better understand the Repair process, it's important to highlight that the main products that this company segment takes care of are Gas Turbines (see Figure **??**). Gas Turbines can be standard or Engineered To Order (ETO). ETO Gas Turbines are usually new customer-specific products, high fuel consumer: they are called informally **Heavy-Duty Gas Turbines**. On the other hand, there are Standard Gas Turbines, such as the **Aeroderivative Gas Turbines** (also called LM, that stand for Land Marine, because they can be used both for land or for marine applications), created for airplane engines, but applied also in a energy domain.
The firsts are part of Baker Hughes's original equipment manufacturing, whereas the seconds usually are in Baker Hughes business thanks to Joint Venture agreements with other companies.
For aeroderivative Gas Turbine, since the Repair activities are performed in a Joint Venture's Shop, an Intra-Company process has to be managed and a Scope of Work with technical and commercial details must be done: this, increases the complexity of the information to be handled.

The repair operations can be on different. Some Shops are dedicated on Heavy Duty items, such as Components, Rotors or Modules. On the other hand, other Shops are dedicated to Aeroderivative items, taking care of Major Overhaul, Hot sections, Modules and Tests. That's why, at the beginning of the Repair process, the Shop must be selected also with respect to the type of machine and the repair activities to be performed.

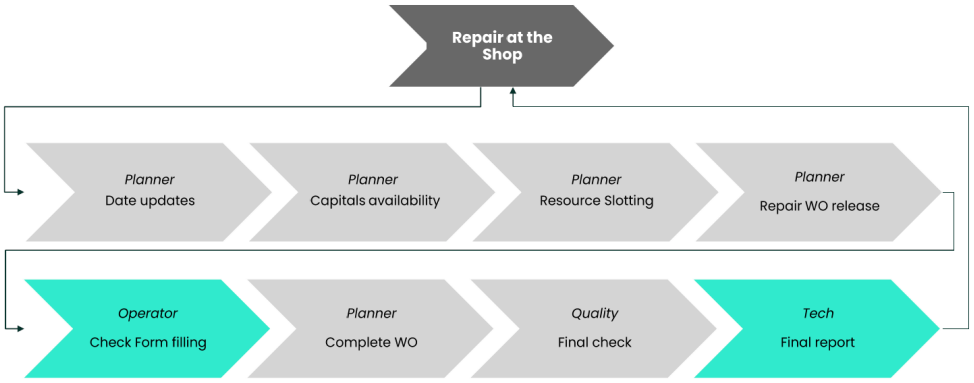


Figure 3.6: Repair at the Shop

Gate process

When the items arrive physically on site, there is a procedure that tracks all the life-cycle at the Shop: it goes on during all the Inspection and Repair Phase, until the end. It is called *Gate process*. Gate Process is divided in four main stages (see Figure 3.7):

- **G0**: Technical Scoping
  Information of incoming Job are received in advance and technical preparation of material for input into Inspection is done. For Aeroderivative items, Repair and Inspection Scope of Work are created at Gate 0.
- **G1**: Items Inspection
  Items disassembly, cleaning and crack detection are performed at this Gate. A report issuing the deviation versus the budget costs is done and, if Aeroderivative, an updated Repair Scope of Work must be confirmed.
- **G2**: Items Repair
  When the Customer approval arrives, the internal Repair are performed and new parts are procured, if necessary.
- **G3**: Items Assembly
  When all the materials required arrive on site, items are assembled again, the technical Report is completed and the final checks on the machine close the last Gate.

For every type of Repair, the Gate process can include different activities at each Gate. Moreover, the Turn Around Time (TAT) from G0 to G3 can vary: Tests usually are expected to last 9 days, while Modules about 90 days.

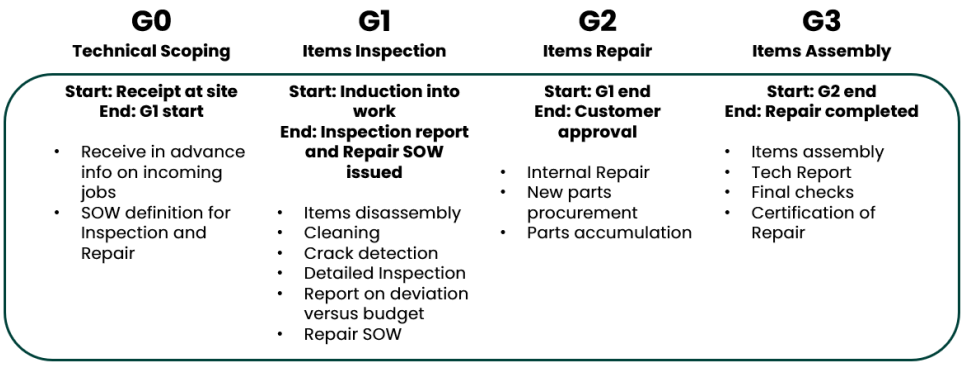


Figure 3.7: Gate Process

3.2.4 Cost Control

The Cost Control Phase has two important steps, performed by the Project Manager of the current Repair.

- **Changes Management**
  It is the management step of all the changes following the creation of a new contract: it includes the management of change orders (the customer can ask for something more or less or different to do), or changes that may be required during the execution, such as rescheduling or Shop change.

- **Shipping & Billing**
  It is the final step: the bill is closed and the machine items are sent back to the Customer.

3.3 Repair One Portal Application

The entire Repair Process leverages on a web application for internal use, called Repair One Portal (ROP), running on AWS. ROP allows the different actors involved in the process to exchange information in various stages. The application has been divided in four modules:

- Repair One Portal (**ROP**): The main module that rules the end-to-end ITR process across different Shops and commercial regions

- Service Shop Cockpit (**SSC**): The cockpit module monitors Shops' activities progress

- Scope Of Work Configurator (**SOW**): The module to manage technical and commercial proposals for aeroderivative gas turbine repairs

- Check Forms (**CF**): The module that tracks all the tests done on the machines at the Shop

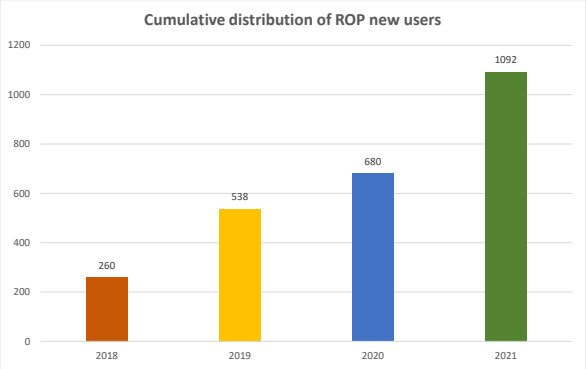


Figure 3.8: New Users of ROP application from 2017 to September 2021

The application started from ROP module and during the years it was integrated with other tools, such as SOW for technical and cost details about Repair of aeroderivative Gas Turbines, or SSC to monitor the performances of the Shops. Consequently, ROP application in 2021 reached more than 1000 users. The growing number of users is shown in Figure 3.8. ROP is distributed over 43 Repositories and has approximately 20K Lines of Code. ROP development team has about 15 people working on it and gained a total of 120 Sprints from 2017 to September 2021 (~30 Sprints every year).

With ROP adoption, it can be also noticed that the number of RFQ and RFI created are growing during the years. This is good in term of management and monitor of Repair life-cycle, and Figure 3.9 and Figure 3.10 shows the number of new Requests and new SOW every year: ~34 Requests per week and ~20 new commercial configurations every week. These dimensions clearly indicate how critical it is to have an analytical layer in this context.

3.3.1 Repair Process on ROP

Figure 3.11 shows how the multiple stages of the Repair procedure leverage on the four ROP modules.

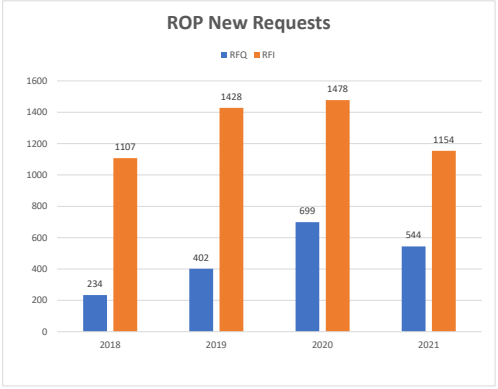


Figure 3.9: New RFQ and RFI in ROP application from 2018 to September 2021

Figure 3.10: New SOW in ROP application from 2018 to September 2021

- Quoting phase (RFQ) is entirely managed on ROP main module. Only with the aeroderivate items it is necessary to configure the Scope of Work on SOW module, with all the activities to be done and the Pre-Inspection cost breakdown analysis.
- At this point, RFI Creation is done on ROP module, filling it with contract type, customer and machine information.
- The Routing Phase leverages on the information handled in SSC module: the better Shop is selected having all the information updated about current activities across different sites.
- The Pre-Inspection RFI is done again via ROP module, managing the

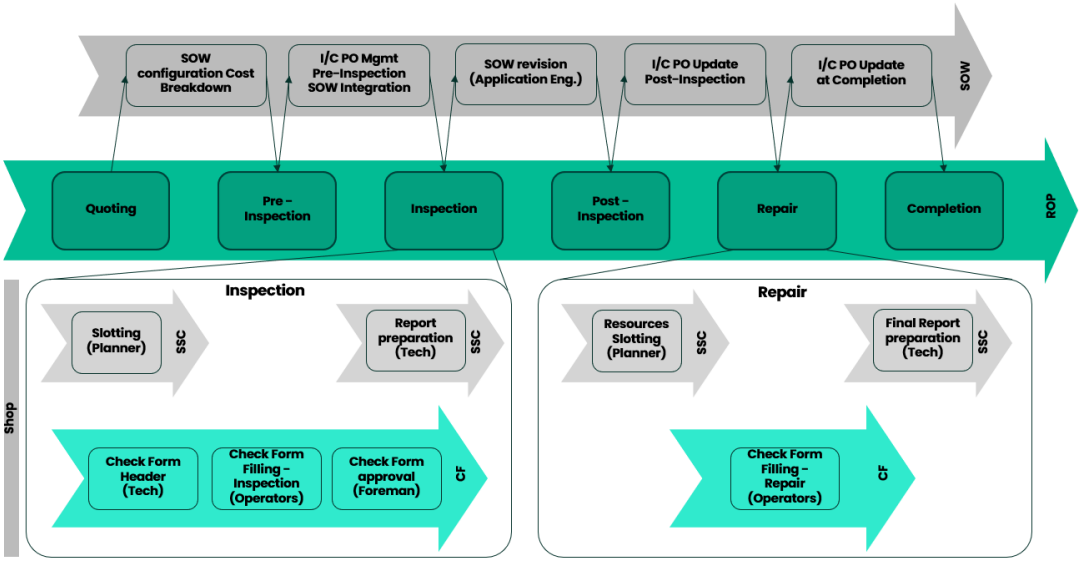


Figure 3.11: Repair Process

aeroderivative-case on SOW module, in order to integrate details about Intra-Company process (the aeroderivative items are sent to the JV Company, while Baker Hughes acts as the Primary Contractor with the Customer) and associated costs and percentages.

- During the Inspection Phase, the slotting of the resources of the Shop is tracked on the SSC module, while all the checks performed on the Gas Turbine's items leverage on the CF module.

- After the Inspection, the RFI is updated on ROP module (RFI Post-Inspection). If in aeroderivative case, again, the Scope of Work need to be changed, on SOW module, together with Post-Inspection costs. Then, the Hand-off is signed.

- After the Customer approval, the Repair is performed on Shop site. Again, SSC module is used to track the resources engaged and the CF module is used to write details about all the Repair activities.

- At Repair Completion, the RFI is closed. Of course, in aeroderivative case, eventually the updates are done also in SOW module.

As said, the procedure is really complex: to allow monitoring, Business Function wanted a way to collect, analyse, aggregate repair data in a report shape.

3.3.2 Repair data

Repair data are stored based on the application's module they come from and on their data type. The modules on which this project focuses are: ROP, SOW Configurator and CF. Almost all data passing across the modules are stored on a RDS AWS (Relational Database System of Amazon Web Service), based on a PostgreSQL Database Engine. A portion of semi-structured data, instead, are saved in a AWS MongoDB. These Databases contain operational data of Repair processes.

Structured Data

ROP and SOW Configurator data are entirely stored in the same RDS AWS, based on a PostrgreSQL Database Engine. ROP and SOW Configurator module contain data about new requests, customer details, machine models, machine components, shop details, quoting based on the type of the repair, and so on. The Repair process follows predefined steps. Since it is unnecessary to have a flexible schema, data can be represented and stored into tables.
At this point, it's appropriate to briefly describe PostgreSQL.

- **PostgreSQL**
  PostgreSQL is a free and open-source Relational Database Management System (RDBMS), using SQL language and storing data in tabular format. Data stored in tables are related to each other with *external keys*. Usually, SQL is not suitable to implement complex logics. This may be a limit for a Database system, but PostgreSQL allows programmers to build **logics**, in different ways: with PL/pgSQL, wrappers for scripting languages (like Perl and Python), complex functions in C/C++ or interfaces for R language [31]. Due to the fact that it is highly-programmable, this pushes PostgreSQL ahead of the other Database engine competitors, providing some advantages:

  – Better performances, thanks to the fact that some logics are directly applied on the database, reducing the workload in a eventual client-server information exchange [31].

  – Reliability growth, due to the code centralization on server machine: logics don't have to be synchronized between different clients [31].

  – Thinner client, thanks to the fact that logics are moved on the server [31].

  A PostgreSQL Database cluster internal organization relies on **schemas**. Schemas are like name-spaces, allowing objects with the same name to co-exist in the same Database [31]. Schemas handle all the data except users and groups, that are shared across the so-called PostgreSQL cluster: potentially, if

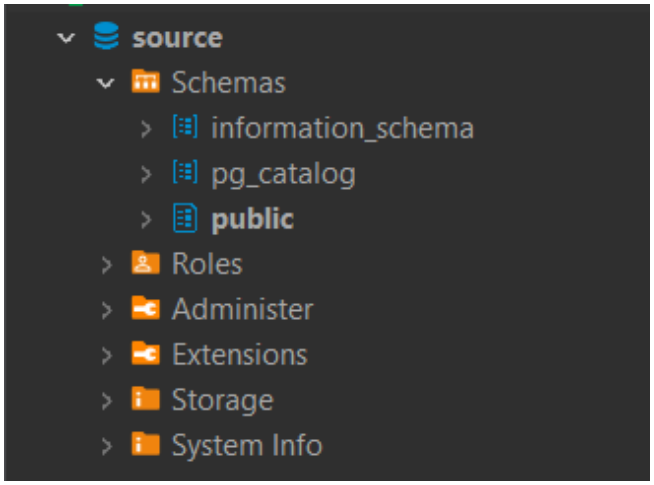


Figure 3.12: PostgreSQL default schemas

a user has the privileges, he is able to access to any schema [32].
By default, new tables and data are put in *public* schema, but it is possible to create new schemas according to the needs. In addition to these schemas, *pg_catalog* schema contains system tables and all the built-in data types, functions and operators [32]. The *information_schema* mainly contains views of all the objects defined in the current database [33]. If *information_schema* can be portable on other DBs, *pg_catalog* is specific to PostgreSQL.

ROP and SOW Configurator data are stored in a database schema called *prodorp* (actually, there are three environments *dev/qa/prod*[1], each corresponding to a deployment stage; for simplicity, for the moment we will merely mention *prodorp*[2], always keeping in mind that what it is said for *prodord* applies also for the other two environments). There are an overall of 152 tables, weighting approximately 170 MB. Every year, about 1.500 new RFI are opened (see Figure 3.9). Beyond theses numbers, what really matters in this context is that the variety, quantity and granularity of different information to be handled for a single repair life-cycle are quite large: every repair request have to be reach in detail, passing from general information about the customer, to the single cost-drivers in the quotation of the repair service or to the technical specifics of the machine items to be repaired. Directly reading these type of information is not so easy.

[1]DEV = development, QA = quality assurance, PROD = production. The deployment of the project follows three steps, with three different work environments, in order to not affect the stable solution while developing and in order to be able to perform multiple checks on data, before the changes become permanent. We will describe it better in Chapter 3.4.

[2]*orp* stands for "One Repair Portal", how this application module was previously called.

Semi-structured data

Check Forms module main objective is to track everything that happens physically on the items at the Shop, either during the Inspection and during the Repair. Since every repair performance is different from each other, due to the machine types, machine models or the current problems, this part of the application must be flexible enough to allow the different users to dynamically create a form for each repair case.

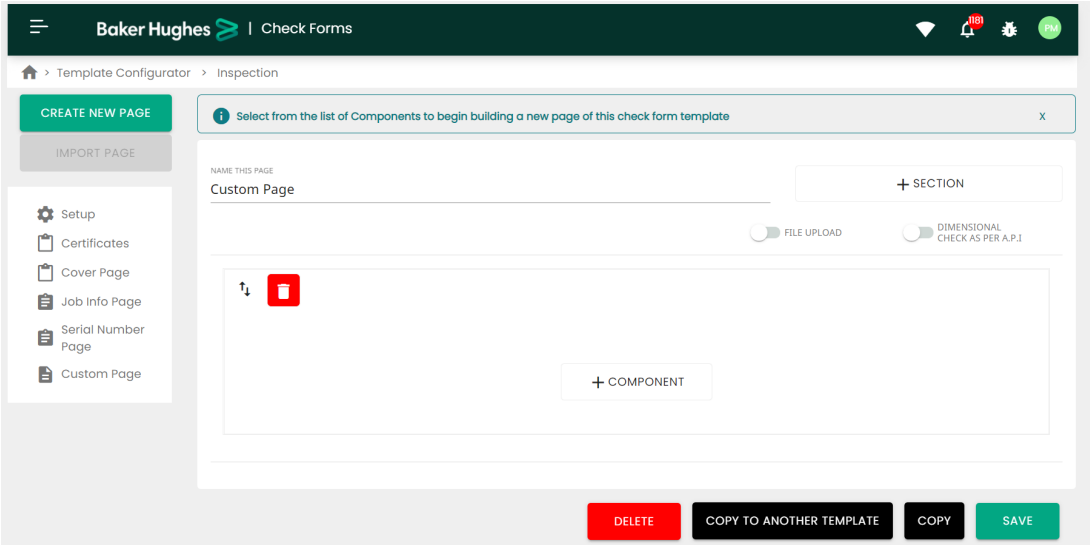


Figure 3.13: Check Form new Template for Inspection

Every Check Form is divided in multiple Pages. There is a fixed section with the repair request details, which includes Setup Page, Cover Page, Job Info Page and Serial Number Page (see Figure 3.13). This part is stored in a dedicated schema on PostgreSQL RDS, called *prodorp_ifs*[3]: it involves only 16 tables, approximately 9MB. This structured part of the data module contains mainly general information of the request and template information to relate it with to the core part of this module: the real check forms, implemented in a semi-structured way.

Check Forms can be dynamically extended thanks to a template configurator: the user can create one or more new *Custom Pages*. Custom Pages can be renamed, can be divided in Sections, can be populated with multiple Components to choose. Components can be different in types: text, image, table, check box, and so on

[3] *ifs* stands for "Inspection Forms", how this application module was previously called. Recently the module was redesigned and renamed with "Check Forms", but the back-end Database structure renovation is still in progress.

(see Figure 3.14). Once chosen the Components needed, each one can be obviously filled with current information of the checks performed.

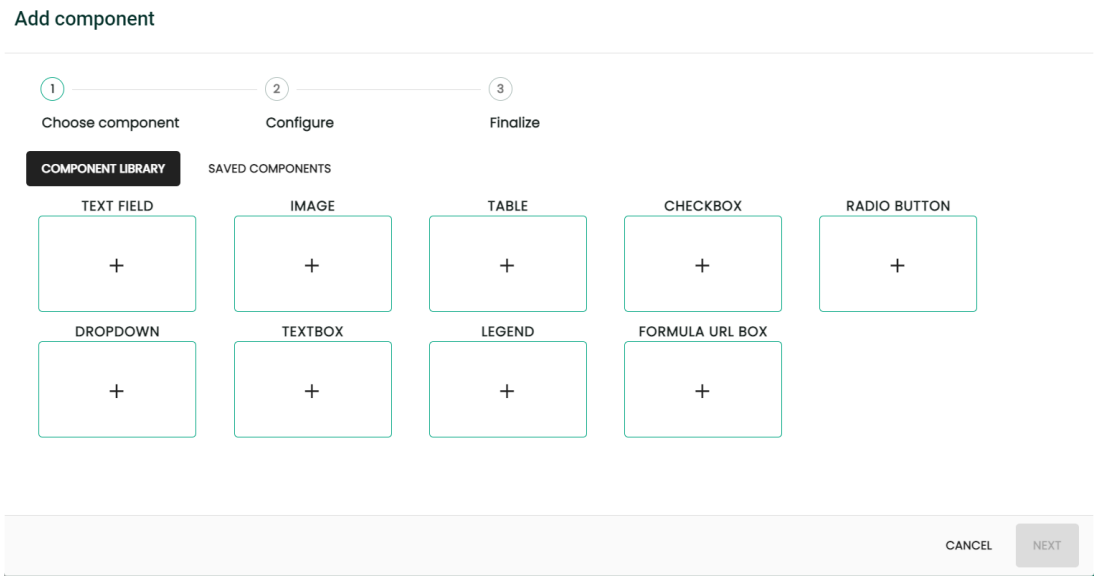


Figure 3.14: Check Form new Custom Page options

This custom part is stored on a MongoDB AWS.

- **MongoDB**
  MongoDB is an open source NoSQL database that provides support for JSON-styled, document-oriented storage systems. MongoDB has some interesting features.
  - MongoDB supports ad-hoc queries, such as fields, range, regular expression searches. The output can be specific fields of documents, and can include user-defined JavaScript functions [34].
  - Any field in a MongoDB document can be indexed, also with primary, secondary, geospatial or other indices [34].
  - MongoDB emerged as a Cloud based technology. For this reason, it is reliable in terms of data data loss. Leveraging Replica Set technolgy, MongoDB ensures there is always a copy of data.
  - For the same reason above, MongoDB can scale horizontally and run over multiple servers: it has a load balancer to keep the system up and running also in case of high workload [34].
  - MongoDB can also be used as a file system, taking advantage of the replication and balancing features across multiple servers to store even large files.

MongoDB is the most popular NoSQL Database, also used by some famous companies like eBay, Google, Adobe or Cisco. It is a really advanced and sophisticated solution to store unstructured data.

For each Check Form, every Custom Page instance represents a Page, with its Sections (at most 3), which can have its multiple Components. To model a Custom Page as shown in Figure 3.15, JSON document structure is used.

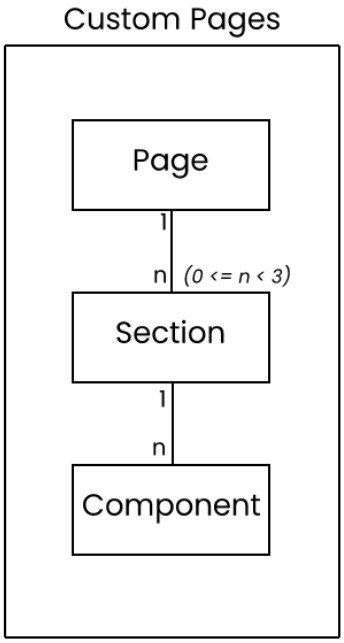


Figure 3.15: Check Form, Custom Pages Schema on MongoDB

An instance of a Custom Page has:

- **templateId** that refers to the current Check Form Template instance
- **pageId** that refers to the current Page
- **sections** representing the array of at most three different Sections of the Page

While, every Section has:

- **index** that can assume value equals to 0, 1 or 2
- **components** representing the array of Components of the current Section

Finally, every Component has:

- **componentName** associated with the name of the component
- **componentDetails** providing details about component type and contents

Custom Pages can be very different one from each other, so the idea of storing these information on a MongoDB was the best option to choose. Every Custom Page correspond to a JSON file, with the same high-level structure, but being different on the inside.

3.4 Work method

This thesis work followed several project steps.

Solution Design and Requirements investigation

First of all, the study of the technologies available for the implementation of the solution was made. When the design reached a fairly stable solution, the examination of the first functional requirements with the reports owners started, in order to determine if the solution's design was appropriate for the type of requests.

Having made the necessary checks on the feasibility of the solution, eventual adjustments or updates on the technologies to be used, a naming convention was designed to be used in the implementation phase.

Furthermore, the analysis of the first requirements of the reports continued thanks to weekly meetings in which the required data, their origin, their meaning, the calculations to be made on them, etc. were discussed. All reports followed these same phases, more or less time-consuming depending on the effort required for each report and its priority.

Solution Deployment and Reports Realization

The implementation is divided in three stages: Development, Quality Assurance, Production. To support this deployment method, three working environments are used:

- DEV environment
  It is where all the changes are developed [35]; unit tests on the changes are performed before moving to QA environment.
- QA environment
  It is where data checks and failure tests on the changed code are performed. If something is wrong, additional changes are applied from DEV environment;

otherwise, changes becomes permanent and are moved to PROD environment [35].

- PROD-environment
  It is the environment that users directly interact with. Moving to PROD is the most sensitive step and is important that tests and checks on the previous environments succeed [35].

Chapter 4

Data Lake Solution: Design

4.1 Study of the requirements

Monitoring important KPIs and performing specific analysis of Repair area is crucial in terms of performance improvements. Some basic examples of what can be achieved using data analytics in this company area can be long-term forecasts of repair requests, failures or outages prevention, checks that deadlines are met. It is quite difficult without a digital support, considering the complexity of the process and the number of new Repair Requests and Commercial Configurations increasing over the time.

ROP Database is continuously updated with transactional data (through INSERT, UPDATE and DELETE statements), reflecting the changes that users make on the web application. So, performing analysis directly on the Database would give inconsistency errors or locks on the data, sharply reducing performances. Having a **dedicated platform** where to perform analytics is clearly necessary in this context.

Furthermore, ROP web application handles the complex Repair process. It manages its multiple sub-processes, each one composed by multiple sub-steps. ROP is a highly-operational tool: the needs it fulfills are reflected in its database, whose structure is designed to facilitate its operations and to support the process-flow. So, it is not easy to analyze data as they are in the ROP Database: thanks to its structure and multiple data granularities, a layer where to **model** data and bring it to the same level becomes quite fundamental.

BH business function needs to create **custom reports**, adopting a user-oriented approach: as said in Chapter 3.4, Business people or teams, who want a report on Repair data, should provide a detailed description of the structure and the fields needed in it. Then, a lot of important KPIs do not have a one-to-one correspondence with a DB field, but they must be computed on top of some of these fields: simple

examples can be the number of days between the start date and the end date of a repair activity, or the overall of some costs, etc. These types of information can be included in a report, and the owner usually provides details about their computational logics.

Another important aspect of this context is that Repair reports are the first stage of a multi-phase strategy of the BH Digital Technology function, with the objective to **integrate data from various sources** (such as Enterprise Resource Planning (ERP) data, other teams data or other Data Lakes and Data warehouses, etc.) as much as possible, in order to improve cross-sectional analysis. From this point of view, on one hand the chosen solution should provide direct access to Reapair ITR process information, and on the other hand it should be designed with the intention of allowing it to communicate with other data sources of the company.

4.2 Solution architecture

Given the requirements, Data Lake is the best option to be adopted for many reasons. First of all, it collects all the data of the different ROP modules. Also, it allows a decoupling layer between ROP operational Database and repository where to perform analysis. Then, it provides the flexibility to explore all data available from ROP and to build custom reports, based on the specific requirements of the data analysis. Moreover, thanks to the raw data copy in the DL, data scientists can model and apply transformations on data and to optimize logics. Finally, Data Lake can become a single source for retrieving and integrating Repair information with data sources from other areas of Baker Hughes Company.

This project is actually the start point of the ROP Data Lake initiative. As seen in Section 3.3.2, ROP web application stores its data both on RDS and on MongoDB AWS technologies, but the main work of this thesis focuses on ROP and SOW modules' data, as the most crucial for the business needs. As said, ROP application runs on AWS, and these two modules rely on a AWS RDS with a PostgreSQL engine, just like the majority of all the application data. The Company runs plenty of different softwares on AWS Cloud, so the choice to implement the Data Lake leveraging the same technologies was the most immediate, less expensive and complicated in terms of set-up, costs and maintenance.

The architecture of the solution can be seen in Figure 4.1. Data Lake is structured in three tiers:

- **Raw Data Tier**
  ROP and SOW data is copied from the Relational Database system AWS in the Raw Data Tier of the Data Lake with a Logical Replication mechanism, which allows the near-live synchronization of the updates from the source to

the destination.

- **Modeled Data Tier**
  Once Raw Data is copied in the Data Lake, several transformations are applied in order to provide a layer of Modeled Data.

- **Consumption Data Tier**
  After having all the data logically organized in the Modeled Data Tier, data can be analyzed in different ways. The output of the analysis is packaged and provided in the Consumption Data Tier, where information is available to be retrieved from outside. Different methods can be used to access data from the Consumption Data Tier:

  – Data scientists can directly extract data from the Data Lake in a Excel document

  – For some external Teams, Custom APIs are implemented to directly feed their applications with ROP data

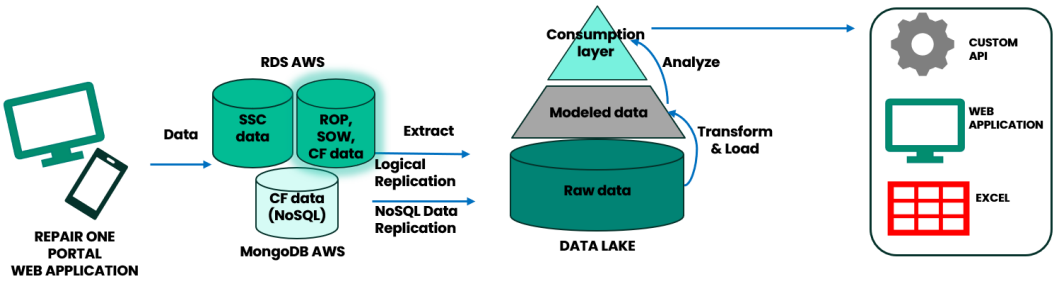


Figure 4.1: ROP Data Lake Architecture

A detailed description of the solution design follows.

4.2.1 Data Replication

. In order to have Raw Data in the first Tier of the Data Lake, it is necessary to transport it from ROP Database to ROP Data Lake.

PostgreSQL version 10.4 introduces Logical Replication to its features. Since Data Lake is implemented on a PostgreSQL RDS, such as the Database on which ROP application stores its data, Logical Replication technology is used to feed the **Raw Data Tier** of the Data Lake.

Logical Replication replicates data objects and their changes, based on a Replication Identity (usually, a *primary key*) [36]. A data object is usually a table or a view, identified by its schema name and its name (*schema_name.table_name*

or *schema_name.view_name*), while Replication Identity is a value that uniquely identifies a record.
Logical Replication uses a **publisher-subscriber** model: one or more subscribers subscribe to a publication instance on the publisher. More precisely, subscribers pull data from the publication they subscribe to [36]. Logical Replication has two phases:

1. **Initialization**: Where a snapshot of the data on the publisher is copied on the subscriber
2. **Synchronization**: where all the changes on the publisher are replicated to the subscriber in near-real time

The publisher is the source where data actually resides, while the subscriber is the destination where data is going to be replicated. In this case, the publisher is the application Database and the subscriber is the Data Lake instance.

There are some prerequisites:

- On the subscriber, objects' structures are not replicated, so they must be manually created in order to start the replication: when, for example, a table structure is changed on the publisher and replicated data starts arriving at the subscriber but does not fit into the structure of the destination's tables, replication will fail until the table structure is updated [36].
- Data types of the columns between the publisher's and subscriber's objects don't need to match, as long as you can convert the text representation of the data at the publisher to the target data type at the subscriber [37].
- A subscriber table can have additional columns with respect to the publisher.

The steps of the Initialization process are the following (see Figure 4.2):

- A Publication is created on the Publisher instance: the publication contains the set of objects of the source that are going to be replicated. A snapshot of the publication has to be copied on the destination.
- A Subscription is created on the Subscriber instance. A successfully created Subscription, having all the connection information, triggers a TCP connection to the Publisher instance [37].
- The incoming connection on the publisher triggers the creation of a Temporary Logical Replication Slot. PostgreSQL keeps track of its transactions in special files called Write-Ahead Logs (WAL); the Replication Slot handles the standby situation and prevent from deletion of WAL during standby. Replication Slot is used with the support of a decoding plugin (*pgoutput*, in this case), in charge of transforming transactions from WAL to Logical Replication Protocol [37].

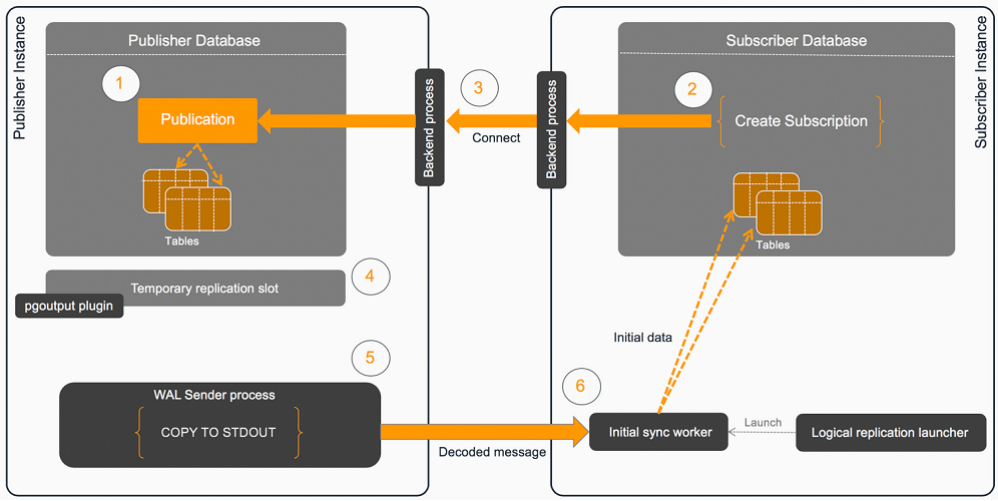


Figure 4.2: AWS Logical Replication Initialization [37]

- The initial data is transferred to the subscriber, via COPY command. The subscriber receives the snapshot and the *initial sync worker* handles the application of the required operations on the subscriber instance [37].

Following the initialization phase, the operational situation is in charge of keeping source and destination synchronized. From now on, any change or update on the publication instance must be replicated on the subscriber. This is how Synchronization works (Figure 4.3):

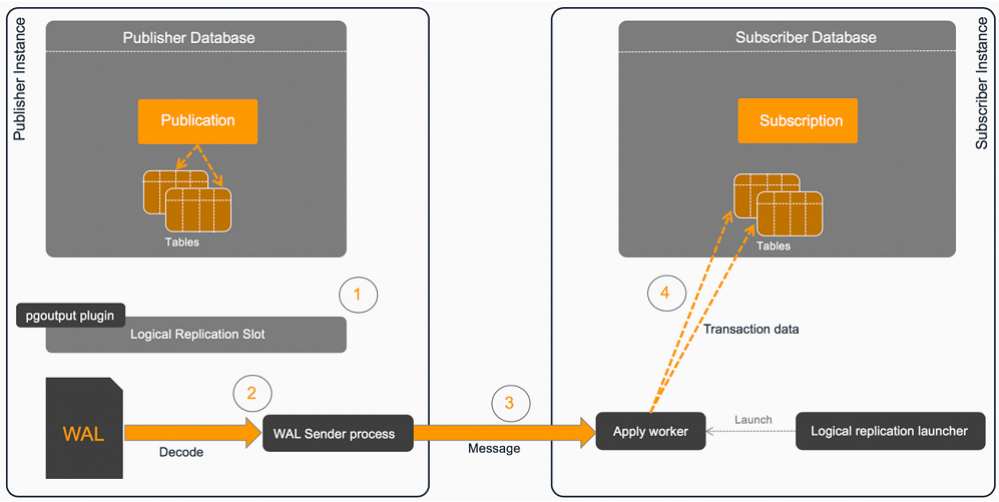


Figure 4.3: AWS Logical Replication Synchronization [37]

- When the initialization is completed, a permanent Logical Replication Slot is created thanks to the logical decoding plugin (*pgoutput*). The Slot continues existing as long as the related subscription [37].

- On the publisher a WAL sender process extracts all the persistent changes of the WALs. Then, the plugin decodes the changes in the Logical Replication Protocol, filtering the data according to the publication specifics [37].

- Data is transferred to the *apply worker*, which applies the changes on the subscriber instance [37].

4.2.2 Scheduling

On the Data Lake some repetitive actions need to be performed. In order to prevent Data Scientists from performing these actions manually every time, the idea to have a Custom Scheduling Micro Service is considered. Automatize the routine activities would reduce maintenance effort and avoid error-prone manual tasks. In general, the Custom Scheduling Micro Service is meant to periodically call functions on the Data Lake through dedicated APIs, with the objective of scheduling some activities to be performed.

The Micro Service is designed to choose and schedule different activities to be performed on the DL, with the possibility to also choose the different levels in which to perform the selected activities: developing environment (Dev/Qa/Prod) level, tier level (Modeled Data or Consumption Data Tier) or single object level.

The design of Modeled Data and Consumption Data Tier includes the possibility to inquiry data both live and periodically-saved. Views provide data in real-time, while Materialized Views provide a snapshot of the data at a certain time. It is important to remark that each View is related with the corresponding Materialized one. The mechanism of periodically consolidating data in Materialized Views is possible thanks to the Refreshing command.

Since the Data Lake purpose is to have a platform where to implement reports, having stable versions of the data to inquiry is a crucial requirement. An important objective in this sense is to guarantee that no updates or changes are applied on data while inquiring it. To do this, the idea is to enable the control of exposure of the data that needs to be analyzed, providing the possibility of ruling the Materialization of the Views, so that data can be refreshed with an arbitrary frequency. This is done thanks to the Custom Scheduling, where the Micro-Service periodically triggers the Materialization refresh function, through an API. Of course, Materialization can be always refreshed manually from a Data Scientist that has adequate privileges to access the Data Lake.

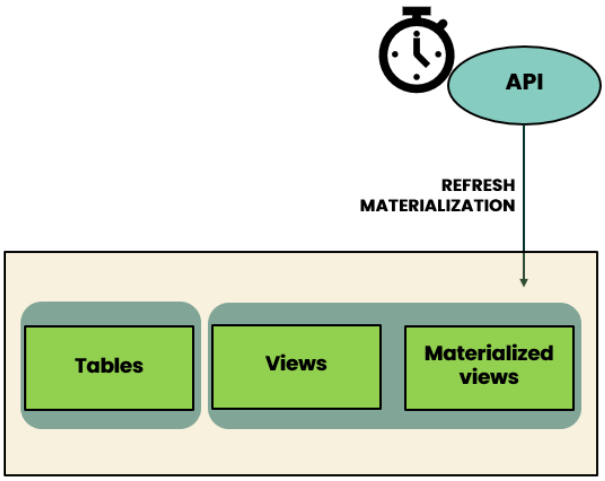


Figure 4.4: Materialization Refresh mechanism

4.2.3 Data Modeling

In this use case, the raw data structure contains several levels of detail, also known as levels of granularity. Thus, the analysis to be performed directly on Raw Data Tier can be impacted. Even if more granularity means more information availability, it also means that the need in terms of storage and computing resources increases, and, most of all, that the information retrieval is more complex. In this situation, the analysis requires to aggregate the underlying detail into an higher level of granularity [38].

A way to face this challenge is a virtualization and materialization approach. In terms of **Virtualization**, the main objective is to collect data from the various sources (the possible sources are physically stored in a single point, the Raw Data Tier of the Data Lake), implementing a layer where to bridge data and logically organize them. Data virtualization enables to access, manage and aggregate data in real-time. In this case, SQL Views are used. A View is the result of a stored query; it is a virtual table computed or aggregated dynamically when access to that certain view is requested [39]. Changes made to data in a relevant underlying table are mirrored in data displayed in corresponding views.
On the other hand, **Materialization** is a form of caching the result of a query, to optimize performances of accessing data. Materialized Views implement this concept in any Database with a relational model, like the one on which the data analysis relies for this work case.

As said, Data in Raw Data Tier has different levels of granularity. To provide an easy-path access to Raw Data, information is modelled in a Entity-Relationship

architecture, which constitutes the **Modeled Data Tier**. An Entity is an abstraction from the complexities of a domain, it can be a thing that exists either physically or logically, while a Relationship captures how Entities are related to one another [40]. Each View represents an Entity, related with other Views thanks to external keys.

The Modeled Data Tier becomes the centralized point where the logics are implemented.

4.2.4 Data preparation for Consumption Data

Making consistent report on Baker Hughes Repair area is the main reason of this thesis work. A report usually includes selected KPIs, which are significant values reflecting the success of the Company on particular activities.

KPIs are not immediately visible only from data as it is. The Modeled Data Tier strongly contributes to provide a layer where to have data modeled in a simple and logical way and where to retrieve it easily, but the Modeled Data Tier by itself is clearly not enough. It is necessary to add an additional layer, where information belonging to different logical Entities is aggregated with the aim of being included into a single Data Provider. A Data Provider represents a group of reports that the Business needs, following the objective of creating multiple Data Providers, one for each new group of reports request. The reason for referring to a group of reports rather than a single report is that when a report creation request is made, the report is often expected to be flexible, such as by creating multiple versions of the same report that shows different portions of the same data or the same data pivoted differently. Respect to this, the Data Provider becomes a basis on which several different extractions can be done: this allows the generation more than one report starting from the same Data Provider.

The **Consumption Data Tier**, just like the Modeled Data Tier, is composed of Views and Materialized Views. In this context, a View represents a Data Provider, a sort of basis where data is aggregated, pivoted, calculated, highlighted in a user-focused way. Also in this context, each View is related to its own Materialized View. Materialized Views inherit the same data frequency consolidation feature of the Modeled Data Tier: the data of the View is periodically cached by a Materialized View, in order to guarantee data stability for analysis.

Data Provider's main function is to answer as closely and accurately as possible to business requests. For this reason, implementing a specific View for each report/group of reports allows to analyse data with maximum flexibility and to structure a targeted aggregation of data. The consolidation can be performed again both manually and automatically with the Scheduling mechanism.

4.2.5 Data Access

In theory, data to be exposed is the Data Providers' one: the Views and Materialized Views of the Consumption Data Tier. The requirements for data access are different according to the different end users.

Part of data within the Consumption Data Tier is used by teams outside the Repair, for their business analysis. The fastest solution found is to provide them with the data directly through some Custom APIs. These Custom APIs give the opportunity to directly connect Data Lake with teams' applications, so that they are fed with required data. An API simplifies programming by abstracting the underlying implementation and only exposing objects or actions the developer needs. In this case, the objective is to retrieve objects in the Consumption Data Tier, transform them into JSON format and finally provide them through the target end point.
Given the low-frequency access to this data and its medium-sized order of magnitude, the idea of creating a Micro Service with Custom APIs gives various advantages. Creating Custom APIs gives the opportunity to flexibly build ad-hoc data inquiries; moreover, JSON document is a common data format useful in data interchange, meaning that this solutions simplifies portability and readability.

The previous solution is specific for some types of data requests, but it is not suitable as general method for reports retrieval by Repair team. Given that, Data scientists can always directly perform extractions from the Data Lake and save them in Excel documents.

4.3 On-going activities

After the beginning of ROP Data Lake implementation, other related initiatives started.

First of all, the reports' creation continues over time. New report requests continuously arrive, and their realization includes investigating their requests with functional users, importing new necessary data into the Raw Data Tier, enriching the Modeled Data Tier, creating new Data Providers in the Consumption Data Tier.

Then, it was crucial to provide a solution for Data Lake access to the web application users. The Team leverages an existing embedded report tool in ROP web application, to allow users to access their reports autonomously.

Furthermore, as mentioned in the Chapter 4.1, ROP Data Lake is the first of a wider initiatives of data integration. After the completion of the first Data Lake release, the integration of ROP Data Lake data into the Enterprise Data Warehouse of BH TPS segment took place.

4.3.1 My Reports 2.0

Excel extraction is the temporary solution adopted to provide users with report data access. The purpose is to allow users to have ready-to-use reports, as soon as they need. To do this, the idea is to leverage a ROP web application embedded tool, called *My Reports*. *My Reports* is a self-service reporting tool that allows users to dynamically create Excel extractions, choosing the fields to retrieve and the filters to apply on data. It is a simple tool, but application users find it very easy-to-navigate and useful.

My Reports 2.0 initiative, has the purpose to improve My Reports tool, in order to allow the users to download Data Lake reports through it: users have their reports at any time, the access to the Data Lake is controlled, and finally Data Lake has a single platform where people, who are not Data Scientists, can retrieve Repair analytical data.

Enabling My Reports to provide Data Lake's data is still an ongoing initiative, divided into two phases:

1. My Reports Micro Service re-pointing: MS points to ROP Database, so it is necessary to make it point to the Data Lake instance.

2. Model a layer that enables choosing fields from a Data Provider and apply filters, in a suitable way for the newly designed Consumption Data Tier.

4.3.2 Enterprise DWH Integration

To build cross-sectional analysis, a set of data is selected to be integrated also in the Enterprise Data Warehouse of the TPS segment, called ARGO. Migration data has been studied and also transformed in order to meet the structural requirements of the Data Warehouse, such as tabular format, primary keys, data types.

Given an ETL tool, the data migration from ROP Data Lake to ARGO DWH follows a multi-steps procedure. To transfer ROP data, some selected tables are first loaded in incremental or full mode, passing through the staging area: Load tables and Key tables are created. Then, to consolidate tables in the DWH, Warehouse tables are created.

4.4 Solution Design observations

The Data Lake solution has been designed by taking into account the Company's existing available technologies first, then the Data Lake's priority needs, and finally the solution's future use and maintenance perspectives.

Considering alternative Cloud Providers or other technologies would have been too expensive in terms of data privacy and management, costs or integration of

different technologies. So, the decision to leverage the Data Lake on an AWS RDS with PostgreSQL was the straightforward one, allowing everything to be kept on AWS: ROP application, ROP Database and ROP Data Lake. This, however, delayed the investigation on how to migrate NoSQL data into the Relational Database on which ROP DL relies. At first, the usage of MongoDB Foreign Data Wrapper, an MIT license technology, a solution designed exclusively for moving NoSQL data to a PostgreSQL database, was investigated. But, due to some restrictions on the RDS on which the Data Lake is implemented, in addition to the fact that CF data (the one stored in a MongoDB) is not crucial for the moment, this option has been shelved, postponing the study of a new solution.

In this context, Logical Replication is the most cost-effective solution, balancing the automation in synchronization between the source and destination instances with the implementation and maintenance effort. The amount of tables to keep aligned is not feasible manually. Logical Replication is a PostgreSQL native solution and it avoids writing a custom program, highly error-prone, while responding effectively to the situation needs.
With respect to Physical Replication, Logical Replication is more flexible and it guarantees the integrity of the transactions. Indeed, Physical Replication is typically used for backups, and not for live synchronization of two DB instances.
The synchronization of replica data goes on with the lifespan of the related subscription, so it is important to keep the source and destination structures aligned: it would be appropriate to build a source structure changes management process, in order to avoid a large transactions' accumulation to be replicated on the destination, before the synchronization is restored.

Scheduling is a powerful solution that could work for several different types of automation on the Data Lake, but it is only used for materialization refresh, for the time being. This gives data scientists a lot of leeway, but also plenty of effort for other common tasks, including manually aligning the structures to keep the logical replication alive, mentioned above. Certainly, the latter is a well-considered option for maintaining a certain amount of control over data entering the Data Lake, at least partially limiting data pollution.

Data modeling and Data Preparation seem the most trivial part of this project. The reality is that, on one side, Modeling requires a long time for studying the data domain and for the investigation on data meaning. Moreover, some scenarios make it necessary to deal with database errors and exceptions.
On the other side, the functional analysis of the requirements for the Data Preparation needs many meetings with the owners of the reports; solution and data presentation proposals must be discussed with them, together with further enhancements, extensions, data quality testing to be performed.
These two are the longest and most crucial parts of this thesis work.

Finally, data access options currently available are sufficiently accurate for

external teams that require data on their platforms (thanks to the Custom APIs), but they fall short of meeting the majority of report demands coming from ROP application users. For this reason, *My Reports 2.0* initiative started as soon as possible, to ensure that business users were autonomous in retrieving ready-to-use reports when they need them, without having to rely on Data Scientists for Data Lake manual extractions.
Additionally, starting the data integration with the corporate DWH is the first step in allowing other business users to perform cross-sectional analysis and leverage other existing powerful report tools, such as Tableau[1] or OBIEE[2].

[1]Tableau is a visual analytics platform, able to generate graph-type data visualizations [41] [42].

[2]Oracle Business Intelligence Enterprise Edition - OBIEE is a unique platform that enables customers to uncover new insights and make faster, more informed business decisions by offering agile visual analytics and self-service discovery [43].

Chapter 5

Data Lake Solution: Implementation

5.1 Logical Replication

Logical Replication technology is a powerful solution to replicate data from a source to a destination and have an automatic synchronization of the update.

5.1.1 Implementation

Environment set-up

AWS supports Logical Replication from PostgreSQL version 10.4. So, before starting, both publisher and subscriber instances must be upgraded if they still are on some previous versions.

After several sessions necessary to define the requirements of the reports with the first requests owners, a set of 90 Tables from *prodorp* schema in ROP Database has been selected to be imported in the Raw Data Tier of the DL.

Having selected a set of tables to be replicated on the subscriber, to speed up the process of creating on the DL the same tables' structure of the publisher, DDL scripts has been generated. Data Definition Language (DDL) is part of the SQL language: it is the syntax used for creating and modifying database objects, such as tables, indices, and users [44].

To replicate the UPDATE and DELETE operations, Logical Replication must use a replica identity, a value identifying the rows involved in the current changes. All tables have a replica identity, usually its primary key: these tables follow a DEFAULT replica mode. But, since it is not strictly mandatory, on the Database some tables may lack it. These tables, without a defined primary key, are replicated with a FULL mode replica, meaning that all columns in a row act as replica identity.

Out of 90 tables, 5 tables of ROP Database are replicated in FULL mode, because they do not have a primary key. The remaining tables can be replicated in DEFAULT mode.

DEFAULT replicated tables have been created on the subscriber with two additional timestamp columns:

- *replica_ins*: the timestamp of the row insert
- *replica_upd*: the timestamp of the last row update

These fields are useful to check that replica is working correctly and to immediately know if the row considered is up to date. Together with tables creation, also two other types of DDL are generated and executed on the subscriber: GRANT and REPLICA TRIGGER. These additional scripts are used to control tables' access and to manage tables' triggers.

Since replicated objects in this RDS are identified by *schema_name.table_name*, the *prodorp* schema is manually created also on the Subscriber instance, and all the replica tables DDLs are executed in there.

RDS Publication

On ROP Database instance, the following scripts must be executed, in order to set up the Publication.

Listing 5.1: Publication Creation

```
1 CREATE PUBLICATION [publication_name];
```

Listing 5.2: Publication Adding Tables

```
1 ALTER PUBLICATION [publication_name]
2 ADD TABLE [table_name_1], [table_name_2],... [table_name_n];
```

Listing 5.3: Slot Creation

```
1 SELECT pg_create_logical_replication_slot([slot_name], 'pgoutput');
```

First, the Publication is created (see Code Listing 5.1). Then, it is modified to add the list of tables to be replicated (see Code Listing 5.2). At the end, the replication slot is created; the decoding plugin (*pgoutput*) must be specified (see Code Listing 5.3). It can be noticed that all the names for publication and replica slot are arbitrary and can be chosen based on the necessities. These first script must be executed by a user with *rds_superuser* privileges: in our case, the DBA is in charge of this kind of operations.

RDS Subscription

On the subscriber instance, the following scripts must be executed by a DBA, with *rds_superuser* privileges

Listing 5.4: Subscription Creation

```
CREATE SUBSCRIPTION [subscription_name] CONNECTION 'dbname=[db_name]
    host=[host] port=[port] user2=[user_name] password=[password]'
    PUBLICATION [publication_name] WITH (slot_name=[slot_name],
    create_slot=false);
```

Creating the RDS subscription triggers a TCP connection with the publisher instance [37]. The incoming connection from the Subscriber enables the following steps of the initialization phase to proceed, as described in Chapter 4.2.1.
Then, when the initial snapshot is copied on the Subscriber, the synchronization starts, and it remains alive, as described in Chapter 4.2.1, until some error occurs (for example: network errors, changes on the source structure).

Data checks

Logical replication automatically provides some metadata on publisher and subscriber *pg_catalog* schema. On the publisher, the most relevant objects are:

- Tables:
  - **pg_publication**: Contains all publications created [45].
  - **pg_publication_rel**: Contains the mapping between relations and publications [46].
- Views:
  - **pg_publication_tables**: Provides information about the mapping between publications and the tables they contain [47].
  - **pg_stat_replication**: Provides one row per each WAL sender process, showing statistics about replication to that sender's connected standby server [48].

Instead, on the subscriber:

- Tables:
  - **pg_subscription**: Contains all existing logical replication subscriptions [49].
  - **pg_subscription_rel**: Contains the state for each replicated relation in each subscription [50].

- Views:
  - **pg_replication_slots**: provides a listing of all replication slots that currently exist on that database cluster, along with their current state [51]
  - **pg_stat_subscription**: provides at least one row per subscription, showing information about the subscription workers [48]

To provide a better metadata management, on the Data Lake (the current subscriber) a new metadata schema is created, called *pgc_catalog*[1]. Within this schema, a new table is created, called *pgc_replica_tables*:

	id	table_owner	table_name	replica_identity	replica_type	creation_date	last_update_date
1	1	prodorp	ong_orp_add_serial_num		FULL	2021-07-08 17:23:51	2021-07-08 17:23:51
2	2	prodorp	ong_orp_component_sow		FULL	2021-07-08 17:23:51	2021-07-08 17:23:51
3	3	prodorp	ong_orp_costing_project_details	ong_orp_costing_project_details_pkey	DEFAULT	2021-07-08 17:23:51	2021-07-08 17:23:51
4	4	prodorp	ong_orp_costing_project_lines	ong_orp_costing_project_lines_pkey	DEFAULT	2021-07-08 17:23:51	2021-07-08 17:23:51
5	5	prodorp	ong_orp_por_details		FULL	2021-07-08 17:23:51	2021-07-08 17:23:51
6	6	prodorp	ong_orp_por_release_history	ong_orp_por_release_history_pkey	DEFAULT	2021-07-08 17:23:51	2021-07-08 17:23:51

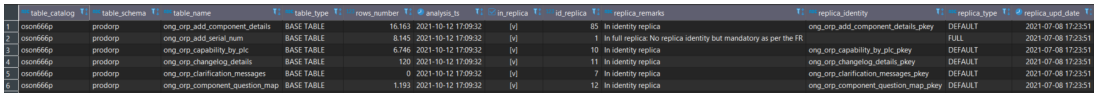
Figure 5.1: Customized metadata table on the Subscriber with replica tables details: pgc_replica_tables

As shown in Figure 5.1, *pgc_replica_tables* keeps track of the replica tables on the subscriber. Every table has an *id*, a *table_owner* representing the schema to which the table belongs, a *table_name*, the specification of its *replica_identity*, the *replica_type* (FULL or DEFAULT), and the creation and last update timestamp (*creation_date* and *last_update_date* fields).

Besides this custom-defined table, four important views are created to perform data checks on *pgc_replica_tables* table and query also the *information_schema.tables* view of the Data Lake RDS:

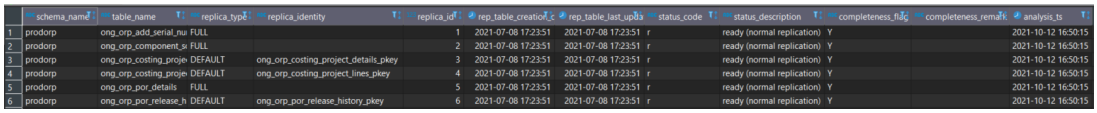
- **pgc_tables_analysis_v**: to perform record counts on each RDS table
- **pgc_tables_hashes_v**: to perform MD5 hash check on each RDS table details
- **pgc_replica_tables_analysis_v**: to monitor record counts on replica tables (see Figure 5.2)
- **pgc_replica_tables_status_v**: to monitor replica status of each table during the initialization and synchronization phase; *status_code* can be i="initialize", d="data is being copied", s="synchronized" or r="ready (normal replication)" (see Figure 5.3).

[1]where the additional **c** stands for customer, meaning this metadata schema is done by the customer of the Relational Database



	table_catalog	table_schema	table_name	table_type	rows_number	analysis_ts	in_replica	id_replica	replica_remarks	replica_identity	replica_type	replica_upd_date
1	oson666p	prodorp	ong_orp_add_component_details	BASE TABLE	16.163	2021-10-12 17:09:32	[v]	85	In identity replica	ong_orp_add_component_details_pkey	DEFAULT	2021-07-08 17:23:51
2	oson666p	prodorp	ong_orp_add_serial_num	BASE TABLE	8.145	2021-10-12 17:09:32	[v]	1	In full replica: No replica identity but mandatory as per the FR		FULL	2021-07-08 17:23:51
3	oson666p	prodorp	ong_orp_capability_by_plc	BASE TABLE	6.746	2021-10-12 17:09:32	[v]	10	In identity replica	ong_orp_capability_by_plc_pkey	DEFAULT	2021-07-08 17:23:51
4	oson666p	prodorp	ong_orp_changelog_details	BASE TABLE	120	2021-10-12 17:09:32	[v]	11	In identity replica	ong_orp_changelog_details_pkey	DEFAULT	2021-07-08 17:23:51
5	oson666p	prodorp	ong_orp_clarification_messages	BASE TABLE	0	2021-10-12 17:09:32	[v]	7	In identity replica	ong_orp_clarification_messages_pkey	DEFAULT	2021-07-08 17:23:51
6	oson666p	prodorp	ong_orp_component_question_map	BASE TABLE	1.193	2021-10-12 17:09:32	[v]	12	In identity replica	ong_orp_component_question_map_pkey	DEFAULT	2021-07-08 17:23:51

Figure 5.2: Customized metadata view on the Subscriber with replica tables analysis: pgc_replica_tables_analysis_v



	schema_name	table_name	replica_type	replica_identity	replica_id	rep_table_creation_ts	rep_table_last_update	status_code	status_description	completeness_flag	completeness_remark	analysis_ts
1	prodorp	ong_orp_add_serial_nu	FULL		1	2021-07-08 17:23:51	2021-07-08 17:23:51	r	ready (normal replication)	Y		2021-10-12 16:50:15
2	prodorp	ong_orp_component_s	FULL		2	2021-07-08 17:23:51	2021-07-08 17:23:51	r	ready (normal replication)	Y		2021-10-12 16:50:15
3	prodorp	ong_orp_costing_proje	DEFAULT	ong_orp_costing_project_details_pkey	3	2021-07-08 17:23:51	2021-07-08 17:23:51	r	ready (normal replication)	Y		2021-10-12 16:50:15
4	prodorp	ong_orp_costing_proje	DEFAULT	ong_orp_costing_project_lines_pkey	4	2021-07-08 17:23:51	2021-07-08 17:23:51	r	ready (normal replication)	Y		2021-10-12 16:50:15
5	prodorp	ong_orp_por_details	FULL		5	2021-07-08 17:23:51	2021-07-08 17:23:51	r	ready (normal replication)	Y		2021-10-12 16:50:15
6	prodorp	ong_orp_por_release_h	DEFAULT	ong_orp_por_release_history_pkey	6	2021-07-08 17:23:51	2021-07-08 17:23:51	r	ready (normal replication)	Y		2021-10-12 16:50:15

Figure 5.3: Customized metadata view on the Subscriber with replica tables status: pgc_replica_tables_status_v

Customized metadata structure is quite intuitive, but at the same it needs to be continuously monitored. It acts as the *transient zone* described in Chapter 2.2.1, where checks are performed to store data in the *raw zone*. Metadata helps to find out easily if logical replication fails and to check tables status.

5.1.2 Observations

Logical Replication implementation is not so difficult, since there are few scripts to be executed for the environment set-up phase and DDLs can be generated automatically. In this case, it's worth saying that Data Lake developers are not completely autonomous, since only DBAs have *rds_superuser* privileges required to execute some scripts. This is the situation of the environment set-up phase, publication and subscription creation, but also when some changes are applied on the publisher structures. To align the two instances, Code Listing 5.5 must be executed, and this latter command requires to be executed only with *rds_superuser* privileges.
Of course, this limit is due to company policies, and it can be dropped if this technology is implemented in other situations, not depending on such policies. However, it is not a big limitation, as the scripts that DBAs have to run are few in number.

Listing 5.5: Alter Subscription Refresh Publication

```
1 ALTER SUBSCRIPTION [subscription_name] REFRESH PUBLICATION;
```

Moreover, as stated in Chapter 4.4, the alignment of the publisher and subscriber structures is provided manually by PostgreSQL, allowing control on data entering the DL, but leaving this task to Data Scientists, which can be time consuming especially in environments where changes are frequent, such as DEV or QA.

Logical Replication is a native PostgreSQL solution (and not an AWS feature: AWS supports it from version 10.4); thanks to this, there is no need to write a custom program, and this aspect allows to avoid human-based errors. The only custom part in this Data Lake implementation with respect to Logical Replication is the additional custom metadata deployed on the Subscriber. The metadata infrastructure, both PostgreSQL one and customized one, is the most important part of this technology. Metadata generally allows Logical Replication to work correctly and to be fixed if the synchronization needs to be restored. Though, custom metadata becomes strategic for the data management on Data Lake side, to perform checks and monitor the status. Nonetheless, still it is a custom mechanism: there is risk of error or not enough documentation for someone new using it.

Another minor detail to be mentioned is that FULL replica tables don't have the additional *replica_ins* and *replica_upd* fields. It is not a relevant lack in this case, because there few tables in FULL mode, but it is better to keep this point in mind in case the number of replica tables without a primary key increases.

In conclusion, Logical Replication has considerable strengths. First of all, the synchronization is automatic and the replication is near-live: this is an important point in terms of performance. With respect to data quantity to be handled, data transfer through custom APIs, for example, takes much longer. Then, it guarantees flexibility on objects and transactions' integrity. Furthermore, it can be implemented also with other PostgreSQL Database engines, on condition that the decoding plugin is additionally managed.

5.2 Scheduling

Scheduling routine activities to be automatically performed on the Data Lake is a powerful additional features that has been implement in this project.

5.2.1 Implementation

A Micro Service Custom Scheduling is used to periodically call from outside some functions defined on the Data Lake. These functions are defined in the Data Lake custom metadata schema *pgc_catalog*.

Before talking about the functionalities automatized thanks to the Scheduling, it's worth clarifying how this mechanism is implemented on Data Lake side.

Three important parametric-objects are defined:

- **Operations**: Operations represent the set of different types of operation that can be executed on the DL
- **Stages**: Stages represent the set of different DL Environments (Logs, Modeled Data Tier, Consumption Data Tier) where to perform the chosen operations

- **Tasks**: Tasks represent the set of single DL objects (Log Tables, Views, Materialized Views) where to perform the chosen operations

With respect to these objects, in *pgc_catalog*, related tables and related functions are defined.
In particular (see Figure 5.4):

- **pgc_operations**:
  It is the table with all the possible types of operation that can be performed

- **pgc_stages**:
  It is the table with all the possible stages where to perform each type of operation

- **pgc_tasks**:
  It is the table with all the possible tasks for each stage and each type of operation; thanks to the details in this table, the SQL commands are generated to be then executed.

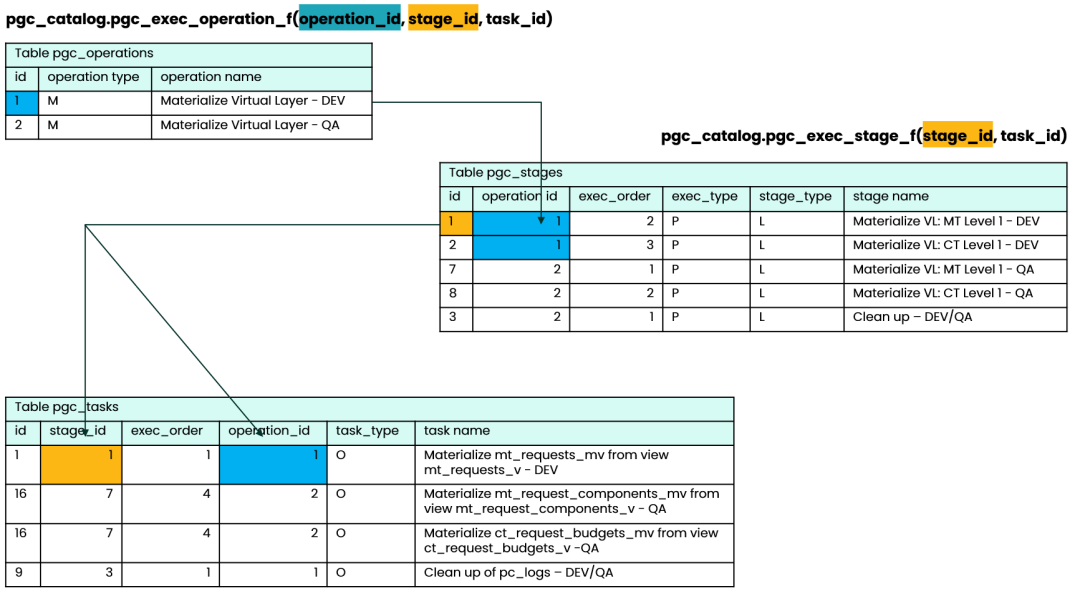


Figure 5.4: Scheduling tables

And also:

- **pgc_exec_operation_f(operation_id, stage_id DEFAULT "-1", tasks_id DEFAULT "-1")**
  It is the function that calls the execution of the selected operation. When *pgc_exec_operation_f()* is called, the type of operation must be specified

in input, while the stage and the single task are optional. This function calls *pgc_exec_stage_f()* for the selected stages (if any stage specification is provided), otherwise on all the possible stages for that operation.

- **pgc_exec_stage_f(stage_id, task_id DEFAULT "-1")**
  It is the function in charge of executing the selected operation on the specified stage; it divides the job in multiple tasks, one for each object of the stage. When *pgc_exec_stage_f()* is called, the stage must be specified, while the task is optional. This function calls the task execution on the selected tasks (if any task specification is provided), otherwise on all the tasks of the stage.

These functions are implemented to allow the parameters specification. In this way, simply calling the *pgc_exec_operation_f()* function with different parameters enables the control of the the operation execution: on all the stages, on a single stage or on a single object of a stage.

Figure 5.4 shows how functions and tables are related to each other. The content of the tables will be more clear after Data Modeling and Data Preparation implementation descriptions (Chapter 5.3 and 5.4).

In this project, the Scheduling mechanism is used to automatically perform the refresh of the Materialized Views of the Modeled Data Tier and the Consumption Data Tier. In this way, the Materialized View refresh frequency can be decided and set to be performed automatically, without doing it manually.

As shown in Figure 5.5, a collateral log metadata mechanism has been implemented, to track the different executions, also with timestamp and parameter details. Besides, a clean up function for log tables (such as Operation Execution, Stage Execution, Task Execution and Log) has been defined to avoid memory saturation and remove no longer useful log data.

5.2.2 Observations

This Scheduling mechanism is implemented in a modular way, in order to always allow the addition and management of new automatic actions to be performed on the Data Lake.

So far, its use is dedicated to the materialization of Views. Considering the context of the report, materialization is an important task, as data consistency and consolidation are mandatory features when talking about report realization. However, the mechanism becomes really useful as the number of reports increases: the Materialized Views to be consolidated with different frequencies grow in number over time, and performing it manually is not feasible, also considering the different time zones in which the reports must be available. Obviously, the same evaluation can be applied when adding any other repetitive commands to execute.

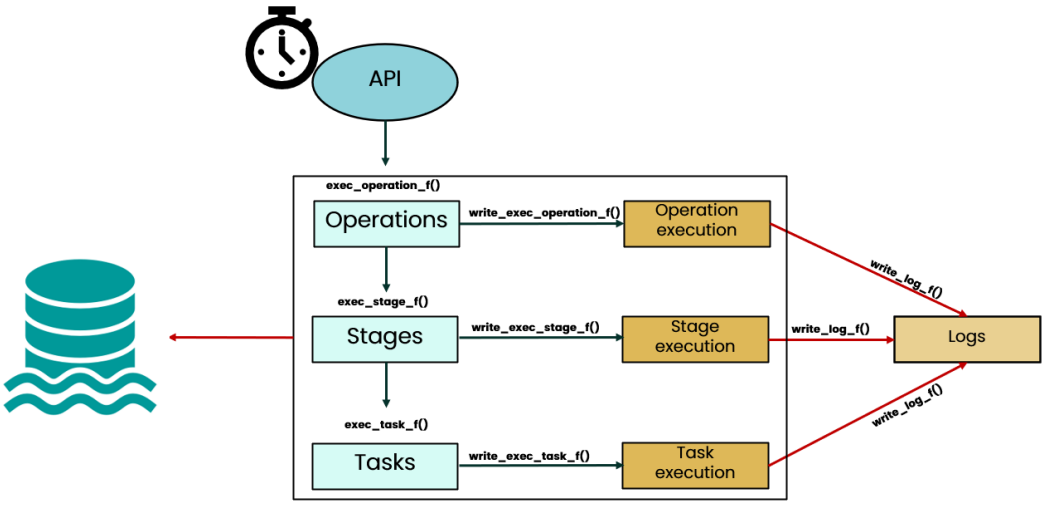


Figure 5.5: Scheduling mechanism

5.3 Data Modeling

Starting from a highly-operational DB, there is need to model data in order to provide a single layer where logics are applied, the different granularities are smoothed out and the data retrieval is not complicated.

5.3.1 Implementation

Modeled Data Tier leverages on a new schema of the Data Lake RDS: *prodorp_mt*. As stated in Chapter 4.2.3, the objective of this layer is to provide a Entity-Relationship architecture to the Raw Data. In there, five Entities are created so far, in order to bring the data to the same level.

- **Projects**
  A Project is the set of inspection or repair activities to be performed on specified machine components. It represents a project (project id, component id, project type, project manager) with all its information in the three different phases - Pre-Inspection, Post-Inspection and At-Completion (such as inspection date, OK-to-proceed date, shop where the activity is performed, cost variations, etc.).

- **Requests**
  A Request is the set of customer request information, like customer name, customer address, request creation date, contract type, the reference to other actors of that current request (like the ComOp), change order generated, etc.

- **Request Components**
  If Request is associated with the general information of a new Request, the Component Request is the related Request described at item-level. Request Component includes the information in detail based on the items: Pre and Post Inspection budget, actual costs, arrival dates, shop where the activities are performed, and so on.

- **Shop cockpit**
  Shop cockpit analyzes Repair Gate Process, for each shop, including business type, contract type, all the expected g-steps dates, the actual g-steps dates, project manager name, component category, the current status of the repair activity and other information that can help to monitor a Shop status.

- **Shop capability**
  Shop Capability models the routing strategy adopted to suggest the best Shop for a new Request. It is realized as a Table Function, where a Data Matrix is exposed and the I/O values are columns of the Table. The Data Matrix has all the possible combinations of the Routing Strategy.

These Entities are realized in View format, based on data of the Raw Data Tier. The Views' code is written following a sub-query factoring approach, using the WITH-Clause. WITH-Clause allows to name a sub-query, creating a temporary table and using it multiple times in the main query. It improves code readability, it simplifies complex queries and it decreases computation effort.

As stated multiple times, in Modeled Data Tier, for each View (see Figure 5.6) that allows real-time data access, there is the related Materialized View, which actually provides periodically-refreshed data (e.g. daily, weekly, etc). A Materialized View is built on top of its associated View, simply selecting all the fields of the latter one (see Figure 5.7).

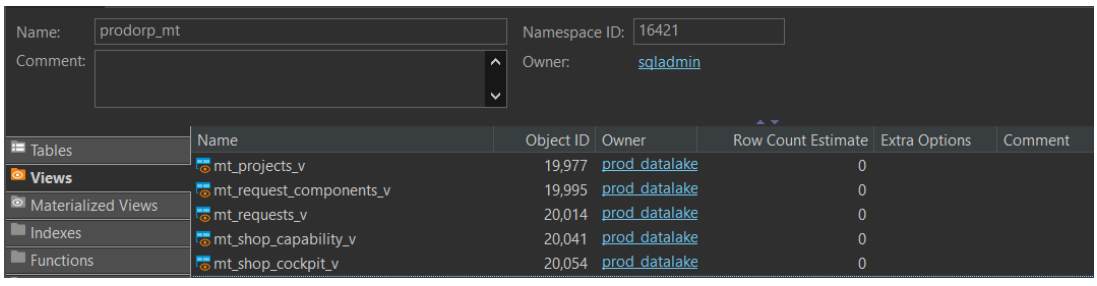


Figure 5.6: ROP Data Lake - Modeled Data Tier Views

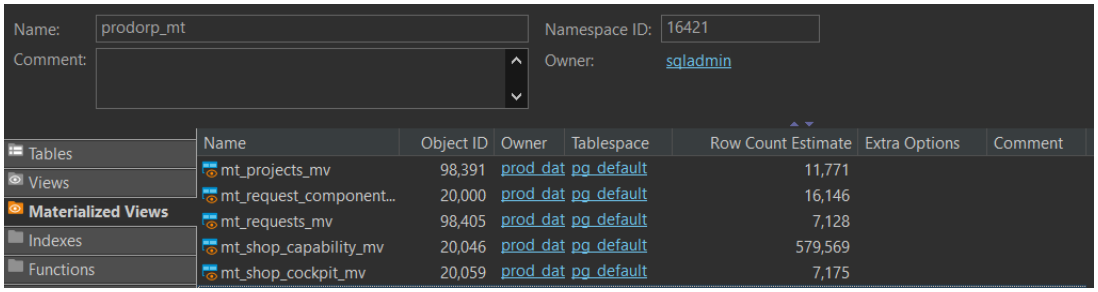


Figure 5.7: ROP Data Lake - Modeled Data Tier Materialized Views

5.3.2 Observations

Modeled Tier is the core Tier of all the Data Lake implementation, because analysis on data becomes really complex if data relationships are not highlighted or the meaning of some fields is not clear. Although ER architecture frequently misses to accurately represent all the aspects of the application, the main objective of this Tier is to highlight the key information surrounding a process or concept (for example: the Project, the Request process or the Shop Routing strategy), rather than to describe everything.
Moreover, in Modeled Data Tier, logics can be optimized, such as the Routing strategy, and some filters can be applied in order to exclude dirty data from the analysis.
In terms of implementation time, this is the longest part, as it is highly probable to find inconsistencies on the data, several errors and fields with ambiguous or variable meaning.

The Scheduling technology rules the materialization frequency of Views but, of course, it can be always performed manually. New fields can be added from the Raw Data Tier to each View of the Modeled Data Tier, in order to better detail the Entity they describe. To represent the View's additional fields in the Materialized View, the latter is implemented by listing the fields of the linked View one by one (rather than using a SELECT *), in order to maintain a tighter control over the data and decide from time to time, whether or whether not the update should be propagated to the consolidated data. This is advantageous during the development.

Modeled Data Tier is really strategic because it allows easy-path access to Raw Data. Also, it is a centralized source point where the logics are implemented, avoiding redundancy and simplifying future enhancements and maintenance.

5.4 Data preparation for Consumption

5.4.1 Implementation

The Consumption Data Tier requires some preliminary activities before the actual deployment. This tier contains the group of reports requested by the users. The Report Owner is the one who submits the formal report creation request, who clarifies the data content meaning and is in charge of report approval.

Therefore, the Owner must provide a detailed description of the report: its purpose, its usage and how often the report is needed. Furthermore, the Owner must write the list of the fields that the report should include, together with filters, logics, computations, groupings to be applied on the data. Usually, it is not immediately clear if it is possible to meet the requirements, so the developer must check if the data is correct, that additional calculations can be performed, that there are no exceptions on data (and, if there are, discuss how to handle them). This phase is very time-consuming, but is crucial: not clarifying how a logic is calculated or the meaning of a field can lead to incorrect reports.

After describing and discussing in detail what the report should look like, the implementation starts. Some reports can be a simple list of fields, with the aim of highlighting them in relation to others, and on which filters or basic calculations can eventually be applied. Some reports may includes data to be partially shown, depending on the use-case. Other reports can be structured in multiple pages, and the same data can be pivoted differently to highlight some KPIs.

As already anticipated, every group of report is implemented through a so-called Data Provider. Data Providers are available in a new schema *prodorp_ct*, in View format, with real-time data, and in Materialized View format, with periodically consolidated data. Again, View and Materialized View scripts are built with a sub-query factoring approach, using the WITH-clause.
The creation of a group of reports mainly consists in aggregating fields from the different Entities of the Modeled Data Tier, based on their relationships, following the required logics, in order to provide the data in the desired format.

5.4.2 Observations

Consumption Data Tier is the one exposed to the outside and on which the extractions are made for the users. Consumption Tier answers in a user-focused way to each requirement, so it finally allows the users to have their custom reports, one of the main strength features of the Data Lake, with respect to DWH.
Fully-relying on the Modeled Data Layer, it inherits all its advantages.

Data recovery through Consumption Data Tier reduces errors and analysis execution time, and, combined with the entire Data Lake structure, allows for a

more sustainable, accessible, and extendable reporting approach.

5.5 Data Access

5.5.1 Implementation

There are two options for data access:

- Custom APIs to directly provide Consumption Data Tier to other team's platforms
  Custom APIs are implemented to retrieve only few of the Data Providers. A Custom API collects data from each of the selected objects, identified by the schema name (*prodorp_ct*) and their View and Materialized View name. Then, it transforms all the columns in string/integer/timezone type creating the corresponding JSON format of the object. JSON format allows high portability of the solution.

- Excel extractions
  For the remaining objects, the default solution is through an Excel file, created by a Data Scientist, with at least basic read-only privileges to the Data Lake, who adequately inquires the Consumption Data Tier. Usually, data required covers only a period of time (e.g., month, quarter, half-year, or year): of course, this condition is something to be manually applied while writing the script querying the Data Lake.

5.5.2 Observations

For intermittent data access, custom APIs are a good option. If Custom API mechanism is unable to keep the data continuously synchronized due to their volume and velocity, for a reporting system, queried once a month, a week or even daily, it responds precisely and directly to the need for an external party to obtain data from the Data Lake.

Extracting Excel files from Data Lake, on the other hand, is a solution that makes customers depending on Data Scientists. This is only a temporary alternative, as mentioned above, while the My Reports 2.0 project is nearing completion.

In general, using Excel extractions or (in future) My Reports tool, enables querying a single Data Provider and creating different versions of the same report, with respect to the filters, grouping, ordering by that can be applied to it. This is a powerful feature that allows Data Lake to be extremely adaptable and responsive to user needs. Certain actions that were previously left to the user are now automated, reducing the amount of users' effort before the report is ready and completed.

Chapter 6

Conclusions

The Data Lake built for this thesis project has been implemented with some notable features. First and foremost, it is a low-cost (in comparison to a DWH) and flexible system that allows ROP data analysis. It's a custom-oriented solution that enables direct access to ITR Repair data. Furthermore, its three-tiers design allows good report quality and fast data retrieval by modeling, transforming, and aggregating data.

This solution still doesn't address how to incorporate unstructured data from MongoDB. Certainly, when compared with one of the main reasons why Data Lakes became popular (i.e., the integration of unstructured data), it can be considered an important aspect that is missing; however, the main objective of this project, in terms of the company's needs, was to develop a flexible solution capable of managing data from a highly operational database in a cost-effective way. In any case, Data Lake was chosen precisely because it also benefits unstructured data, and even though it is not a priority in this use-case and most of the data to be analyzed is stored on a RDS, NoSQL data integration has been planned.

Some examples of future types of analysis that can be performed thanks to this technology are:

- Data clustering to find behavioral patterns (e.g., on customers or machine failures)
- Repetitive activities
- Commercial analysis on sales
- Condition-based maintenance
- Statistical analysis
- Advanced analysis with ML algorithms

ROP Data Lake initiative is not entirely completed. Some mentioned following initiatives are already started and in progress, while others can be planned as next steps.

First of all, the transfer of MongoDB data would complete the ingestion design of the whole ROP data into Data Lake's Raw Data Tier.
Then, among the on-going initiatives, certainly the most critical is *My Reports 2.0*, which would allow business people and ROP application users to download reports autonomously, in a self-service way.

Furthermore, the development of a reporting system for Repair data is the starting point for future integration with data from other business areas in Baker Hughes. This would allow more comprehensive and cross-functional monitoring of performances. In this sense, the integration with ARGO DWH, the Enterprise DWH in BH TPS segment, where data from different teams' platforms flow, could be used to create detailed and in-depth analyses, also leveraging the very powerful tools available, such as Tableau and OBIEE.

In addition to this, another possible future step could be the integration of the data coming from ROP Data Lake with external systems, such as the one of the JV Companies. This type of activity would consolidate the entire end-to-end Repair process reporting system, with the possibly have a complete historical data.

Arrivata a questo punto, mi sembra tutto incredibile. Non solo perché la fine di questo percorso è anche il traguardo che ho sempre avuto paura di non riuscire a raggiungere, ma anche perché nel frattempo ho avuto la fortuna di vivere tutte quelle emozioni che hanno reso il tragitto fino a qui uno dei più intensi e importanti della mia vita. In questi anni sono cresciuta, mi sono conosciuta di più, ho incontrato persone nuove e ho coltivato gli affetti di una vita, ho imparato e ho scoperto con una fame che non pensavo di possedere.

I miei ringraziamenti più sinceri vanno ai miei genitori, Antonella e Stefano, che per primi hanno sempre lottato per aprire davanti a me la porta delle possibilità, permettendomi di scegliere cosa studiare, come e dove studiarlo, sostenendomi anche nei momenti più duri, dandomi la forza di non arrendermi di fronte alle difficoltà. Insieme a loro, ringrazio Irene, mia sorella, la mia compagna di vita e la persona più importante per me.

Ringrazio Francesco Tamberi e Marco Perrone, i colleghi con cui ho svolto questo progetto di Tesi. Mi hanno dato fiducia, guidandomi e insegnandomi più di quello che pensavo fosse possibile. Con loro ho affrontato tutta la realizzazione di questo lavoro, con i suoi innumerevoli imprevisti, ma soprattutto ho condiviso con loro

la gioia di aver portato a termine questo progetto. Estendo il mio ringraziamento anche a Baker Hughes e alle persone che ho conosciuto con quest'esperienza, per avermi dato modo di mettermi in gioco e contribuire nel mio piccolo in un'iniziativa cruciale.

Inoltre, ringrazio la mia Relatrice Elena Baralis, per avermi fatto conoscere e appassionare alla sua materia nel miglior modo possibile e avermi saputo dare consigli indispensabili nella realizzazione di questa Tesi.

Ci sono poi un'infinità di persone che mi sento di dover ringraziare di cuore, per aver fatto parte della mia vita durante questi anni di studio.
La prima persona, quella che mi ha dato il coraggio di intraprendere questo tortuoso percorso, che sento mi abbia dato quella spinta finale per convincere me stessa che dovevo buttarmi, inseguire ciò che mi piaceva e cambiare corso di studi, è stato Rodolfo: anche se le nostre strade si sono separate da molto tempo, non posso non serbargli il grazie più significativo.

Ringrazio le mie amiche di sempre, Camilla e Flavia, per volermi bene come se avessimo lo stesso sangue, per essere i miei fari ovunque io sia, la mia casa e il posto in cui mi diverto di più, per avermi vista arrivare fino a qui, attraverso ogni fase della mia vita.

Ringrazio Gabriele, il mio primo compagno di studi a Ingegneria Informatica, per non aver mai lasciato la mia mano in questi anni, accompagnandomi nei periodi più bui e più difficili che abbia mai passato, assicurandosi sempre che non fossi lasciata indietro. Con lui ho condiviso esami, progetti universitari, paure, gioie, e non potevo trovare un amico migliore con cui condividerle.

Ringrazio Bibi, la persona che Torino mi ha regalato come compagna di avventure, l'amica insostituibile che ho avuto la fortuna di aver sempre accanto, la persona che non ha mai esitato un attimo a farmi sentire a casa.

Ringrazio Matteo, con lui ho vissuto gran parte di questo percorso, lo ha reso speciale e senza di lui non sarei la ragazza che sono adesso.

Ringrazio poi tutti i miei amici più cari: Claudia, Valerio, Simona, Elena, Silvio, Letizia. Non posso escludere da questi ringraziamenti anche i miei compagni di corso, che hanno vissuto insieme a me lezioni, progetti, esami: Riccardo, Francesco, Matteo, Vito. Con loro, allargo questo pensiero a tutte le altre persone che ho avuto occasione di incontrare al Politecnico.

Ringrazio tutta la mia famiglia e specialmente le mie nonne, Anna e Francesca, per avermi sempre profuso l'affetto più puro e sincero che conosco. In particolare, dedico questo momento alla mia Nonna *Checca*, la luce che mi manca ogni giorno, che se ne è andata in silenzio, lasciandomi il ricordo dell'allegria che ha sempre emanato.

Infine, ringrazio il Politecnico di Torino e tutti i professori che ho conosciuto in questi anni, per aver sempre cercato di trasmettere, attraverso l'insegnamento, la passione di cui ogni sfida ha bisogno, e con cui, anche io, ho imparato ad affrontare

le mie.

Appendix A

Insights

A.1 HDFS

The Hadoop distributed file system (HDFS) is a distributed, scalable, and portable file system written in Java for the Hadoop framework[52]. HDFS stores large files (typically in the range of gigabytes to terabytes) across multiple machines. It achieves reliability by replicating the data across multiple hosts, and hence theoretically does not require redundant array of independent disks (RAID) storage on hosts (but to increase input-output (I/O) performance some RAID configurations are still useful) [52].

A.2 PL/SQL and PL/pgSQL

PL/SQL is a procedural extension for SQL and the Oracle relational database. It includes procedural language elements such as conditions and loops. It allows declaration of constants and variables, procedures and functions, types and variables of those types, and triggers. It can handle exceptions (run-time errors) [53].
PL/pgSQL (Procedural Language/PostgreSQL) is a procedural programming language supported by the PostgreSQL Object-Relational RDBMS. It closely resembles Oracle's PL/SQL language [54].

A.3 Replica Set

A replica set consists of two or more copies of the data. Each replica-set member may act in the role of primary or secondary replica at any time [34]. All writes and reads are done on the primary replica by default. Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary

replica fails, the replica set automatically conducts an election process to determine which secondary should become the primary. Secondaries can optionally serve read operations, but that data is only eventually consistent by default [34].

Bibliography

[1] Gartner Glossary. URL: https://www.gartner.com/en/information-technology/glossary/digitalization (cit. on p. 1).

[2] Lorin M. Hitt Erik Brynjolfsson. *Computing Productivity: Firm-Level Evidence*. Nov. 2002 (cit. on p. 1).

[3] Maximilian Röglinger Nils Urbach. *Introduction to Digitalization Cases: How Organizations Rethink Their Business for the Digital Age*. Jan. 2019. DOI: 10.1007/978-3-319-95273-4_1 (cit. on p. 1).

[4] Adam Uzialko. «Industry 4.0: How Technology is Revolutionizing the Manufacturing Industry». In: *Business News Daily* (Nov. 2017). URL: https://www.businessnewsdaily.com/10156-industry-manufacturing-iot.html (cit. on p. 1).

[5] Albert Boswijk. *Transforming Business Value through Digitalized Networks: A Case Study on the Value Drivers of Airbnb*. May 2017. DOI: 10.1177/2394964317697736 (cit. on p. 2).

[6] Gartner Glossary. URL: https://www.gartner.com/en/information-technology/glossary/digitization (cit. on p. 2).

[7] Brookings Institute. URL: https://www.sbrookings.edu/research/digitalization-and-the-american-workforce/ (cit. on p. 2).

[8] Forbes. URL: https://www.forbes.com/sites/jasonbloomberg/2018/04/29/digitization-digitalization-and-digital-transformation-confuse-them-at-your-peril/?sh=21e661852f2c (cit. on p. 2).

[9] Gartner Glossary. URL: https://www.gartner.com/en/information-technology/glossary/big-data (cit. on p. 2).

[10] Oracle. URL: https://www.oracle.com/big-data/what-is-big-data/ (cit. on p. 2).

[11] Mebarki Nasser Atif Shahzad. *Discovering dispatching rules for job shop scheduling problem trough data mining*. July 2010 (cit. on p. 3).

[12] Wikipedia. URL: https://en.wikipedia.org/wiki/Performance_indicator (cit. on p. 4).

[13] Elisabetta Raguseo. «Big data technologies: An empirical investigation on their adoption, benefits and risks for companies». In: *International Journal of Information Management* (Feb. 2018). DOI: 10.1016/j.ijinfomgt.2017.07.008 (cit. on p. 4).

[14] Lorenzo Govoni. URL: https://www.lorenzogovoni.com/data-warehouse/ (cit. on p. 5).

[15] James Dixon. URL: https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/ (cit. on p. 4).

[16] Zhao Shun Wang Pwint Phyu Khine. «Data lake: a new ideology in big data era». In: *ITM Web of Conferences* (Jan. 2018). DOI: 10.1051/itmconf/20181703025 (cit. on pp. 8, 12, 16, 17).

[17] Oracle. URL: https://www.oracle.com/database/what-is-a-data-warehouse/ (cit. on pp. 9, 10).

[18] Microsoft. URL: https://support.microsoft.com/en-us/office/overview-of-online-analytical-processing-olap-15d2cdde-f70b-4277-b009-ed732b75fdd6 (cit. on p. 10).

[19] Panoply.io. URL: https://panoply.io/data-warehouse-guide/data-warehouse-vs-data-lake/ (cit. on pp. 10, 11).

[20] Wikipedia. URL: https://en.wikipedia.org/wiki/NoSQL (cit. on p. 11).

[21] Wikipedia. URL: https://en.wikipedia.org/wiki/Wide-column_store (cit. on p. 11).

[22] Microsoft. URL: https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/relational-vs-nosql-data (cit. on p. 11).

[23] Coral Walker Hassan Alrehamy. «Personal Data Lake With Data Gravity Pull». In: *IEEE Fifth International Conference on Big Data and Cloud Computing 2015* (Aug. 2015). DOI: 10.13140/RG.2.1.2817.8641 (cit. on p. 12).

[24] Alexander Tolstoy Natalia G. Miloslavskaya. «Application of Big Data, Fast Data and Data Lake Concepts to Information Security Issues». In: *2016 4th International Conference on Future Internet of Things and Cloud Workshops. The 3rd International Symposium on Big Data Research and Innovation* (Aug. 2016). DOI: 10.1109/W-FiCloud.2016.41 (cit. on p. 12).

[25] Packt. URL: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781785888083/1/ch01lvl1sec14/data-lake-architecture (cit. on pp. 13–16).

[26] Pradeep Pasupuleti and Beulah Salome Purra. *Data Lake Development with Big Data*. Packt Publishing, 2015 (cit. on p. 12).

[27] Solution Reviews. URL: https://solutionsreview.com/data-management/data-warehouse-vs-data-lake-whats-the-difference/ (cit. on p. 16).

[28] Wikipedia. URL: https://en.wikipedia.org/wiki/Baker_Hughes (cit. on p. 19).

[29] Baker Hughes. URL: https://www.bakerhughes.com/company/about-us (cit. on p. 19).

[30] Baker Hughes. URL: https://www.bakerhughes.com/turbomachinery-services/maintenance-services/advanced-repairs (cit. on p. 19).

[31] Wikipedia. URL: https://en.wikipedia.org/wiki/PostgreSQL (cit. on p. 29).

[32] PostgreSQL Documentation. URL: https://www.postgresql.org/docs/9.1/ddl-schemas.html (cit. on p. 30).

[33] PostgreSQL Documentation. URL: https://www.postgresql.org/docs/9.1/information-schema.html (cit. on p. 30).

[34] Wikipedia. URL: https://en.wikipedia.org/wiki/MongoDB (cit. on pp. 32, 65, 66).

[35] Wikipedia. URL: https://en.wikipedia.org/wiki/Deployment_environment (cit. on pp. 34, 35).

[36] PostgreSQL Documentation. URL: https://www.postgresql.org/docs/10/logical-replication.html (cit. on pp. 38, 39).

[37] AWS Documentation. URL: https://aws.amazon.com/it/blogs/database/using-logical-replication-to-replicate-managed-amazon-rds-for-postgresql-and-amazon-aurora-to-self-managed-postgresql/ (cit. on pp. 39–41, 50).

[38] C3.ai. URL: https://c3.ai/glossary/features/data-granularity/ (cit. on p. 42).

[39] Wikipedia. URL: https://en.wikipedia.org/wiki/View_(SQL) (cit. on p. 42).

[40] Wikipedia. URL: https://en.wikipedia.org/wiki/Entity-relationship_model (cit. on p. 43).

[41] Wikipedia. URL: https://en.wikipedia.org/wiki/Tableau_Software (cit. on p. 47).

[42] Tableau.com. URL: https://www.tableau.com//why-tableau/what-is-tableau (cit. on p. 47).

[43] Oracle. URL: https://www.oracle.com/it/business-analytics/business-intelligence/technologies/bi-enterprise-edition.html (cit. on p. 47).

[44] Wikipedia. URL: https://en.wikipedia.org/wiki/Data_definition_language (cit. on p. 48).

[45] PostgreSQL Documentation. URL: https://www.postgresql.org/docs/10/catalog-pg-publication.html (cit. on p. 50).

[46] PostgreSQL Documentation. URL: https://www.postgresql.org/docs/10/catalog-pg-publication-rel.html (cit. on p. 50).

[47] PostgreSQL Documentation. URL: https://www.postgresql.org/docs/10/view-pg-publication-tables.html (cit. on p. 50).

[48] PostgreSQL Documentation. URL: https://www.postgresql.org/docs/10/monitoring-stats.html (cit. on pp. 50, 51).

[49] PostgreSQL Documentation. URL: https://www.postgresql.org/docs/10/catalog-pg-subscription.html (cit. on p. 50).

[50] PostgreSQL Documentation. URL: https://www.postgresql.org/docs/10/catalog-pg-subscription-rel.html (cit. on p. 50).

[51] PostgreSQL Documentation. URL: https://www.postgresql.org/docs/9.4/catalog-pg-replication-slots.html (cit. on p. 51).

[52] Wikipedia. URL: https://en.wikipedia.org/wiki/Apache_Hadoop#File_systems (cit. on p. 65).

[53] Wikipedia. URL: https://en.wikipedia.org/wiki/PL/SQL (cit. on p. 65).

[54] Wikipedia. URL: https://en.wikipedia.org/wiki/PL/pgSQL (cit. on p. 65).