

A NEW APPROACH TO PIPING ENGINEERING DATA CONTROL AND MANAGEMENT IN THE EPC COMPANY DURING THE ENGINEERING PHASE

Master's Thesis for Data Science



SAPIENZA
UNIVERSITÀ DI ROMA



TechnipFMC

Author: Rana Muhammad Ahmad

Muhammad.ahmad.rana@gmail.com

Matricola: 1734354

Acknowledgments

I would like to express my sincere thanks to my university supervisor Ioannis Chatzigiannakis for guiding me with experience and competence in the development of this work. I also thank my external supervisor Andrea Fiorellino, who gave me a chance to work on this project and helped me with his deep knowledge and expertise in the related field. He was very polite and helpful throughout my work. I am also thankful to Francesco Lippo, Daria Pasqualucci, Simone Martucci, Giuseppina Musumeci, Matilde, and Raffaele. All of them provided me with full support and complete guidance while solving major problems I faced during my work.

I would like to thank the Sapienza University of Rome and all the teachers who played their role in finishing my educational journey here. I am also thankful to TechnipFMC and its entire team which gave me a chance to improve my practical skills while working with them. Lastly, I would thank my friends for making this journey very beautiful to me.

Table of Contents

1	Introduction	4
1.1	Organization's Principles	7
1.1.1	Engineering Manufacturing and Supply Chain organization (EMS)	7
1.1.2	Business Units	7
1.1.3	Six Regional Business Units	8
1.1.4	Functions.....	8
1.1.5	Main RBUs.....	9
1.2	EPC	12
1.2.1	General Workflow	12
1.2.2	Detailed Explanation.....	13
1.2.3	Activities in Time Frame	15
1.2.4	Application Mapping of Data Work Flow.....	16
1.3	Piping.....	17
1.4	Software Applications	18
1.4.1	Intergraph Smart® Materials.....	18
1.4.2	SmartPlant® P&ID.....	19
1.4.3	Intergraph Smart® 3D	19
1.5	Data	20
1.5.1	Data Centers.....	20
1.5.2	Data accessibility	20
1.6	The problem statement:.....	20
1.6.1	Issues	22
1.6.2	Data misalignment	23
1.6.3	Data Availability	24
2	Conversion from SQL to NoSQL.....	26
2.1	Smart Intergraph Materials Database.....	26
2.1.1	cx_Oracle	26
2.1.2	Database overview	26
2.1.3	Data Retrieval.....	27
2.1.4	Conversion to MongoDB.....	27
2.2	Intergraph Smart® 3D.....	29

2.2.1	Pyodbc	29
2.2.2	Database Overview.....	29
2.2.3	Data retrieval and conversion to MongoDB	31
3	Data Availability.....	32
3.1	Comparison between SQL and NoSQL	32
3.1.1	SQL.....	32
3.1.2	NoSQL	35
3.2	Optimizing Query	37
3.2.1	Scope	37
3.2.2	What is it about?	37
3.2.3	Contents of '3DCAT_COMMODITY'	37
3.2.4	Flow and description	38
3.2.5	Stream and query optimization	51
4	Data Misalignment	53
4.1	Certified and Uncertified Pipelines	53
4.2	SQL vs NoSQL	53
4.2.1	SQL.....	53
4.2.2	NoSQL	54
4.3	Restructuring NoSQL Database.....	55
5	Appendix	57
6	Appendix B Code	61

1 Introduction

TechnipFMC is a public limited company incorporated and organized under the laws of England that provides complete project life cycle services (conception, feasibility study, front end engineering, detailed engineering, procurement, construction, commissioning, test runs, maintenance & decommissioning) for the energy industry (Oil refineries, Chemical Plants). TechnipFMC was ranked 23rd among the world's Top 225 International Design Firms in the year 2017 by Engineering News-Record. It was formed by the merger of FMC Technologies of the United States and Technip of France that was announced in 2016 and completed in 2017. It is headquartered in London and has major operations in Houston and Paris where its predecessor companies were headquartered. It has 37,000+ employees from 126 nationalities and operates in 48 countries.

Unique worldwide footprint

TechnipFMC has a presence in 48 countries around the world

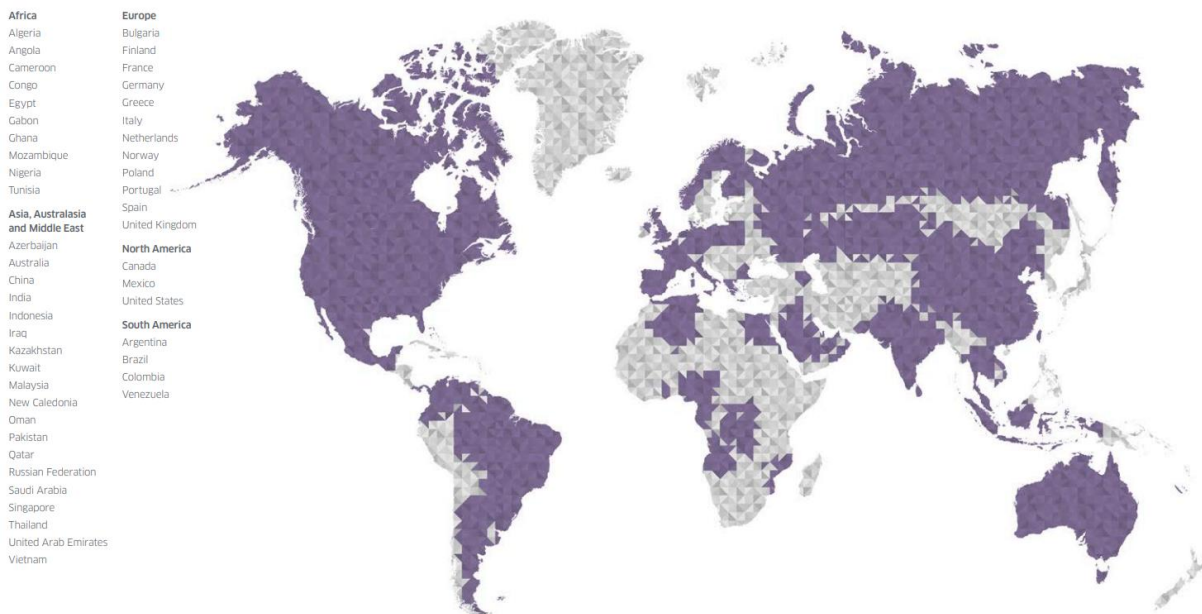


Figure 1: Unique Worldwide Footprint

TechnipFMC acts in three distinct segments: Subsea, Offshore/Onshore and surface projects. These projects include offshore oil and gas exploration and extraction platforms, rigs, crude oil refinery, petrochemical plants such as Ethylene, Hydrogen, Syngas plants, Naphtha, Benzene, etc. plastics & rubber industry, fertilizer plant, onshore as well as floating LNG plants. It is uniquely positioned to deliver greater efficiency across project lifecycles from concept to project delivery and beyond. Through innovative technologies and improved efficiencies, its offering unlocks new possibilities for the clients in developing their oil and gas

resources. Each of its more than 37,000 employees is driven by a steady commitment to clients and a culture of purposeful innovation, challenging industry conventions, and rethinking how the best results are achieved. It seeks to achieve one vision: 'enhance the performance of the world's energy industry'.

The diagram below shows a graphical view of three distinct segments the company works in i.e. subsea, offshore/onshore and surface.

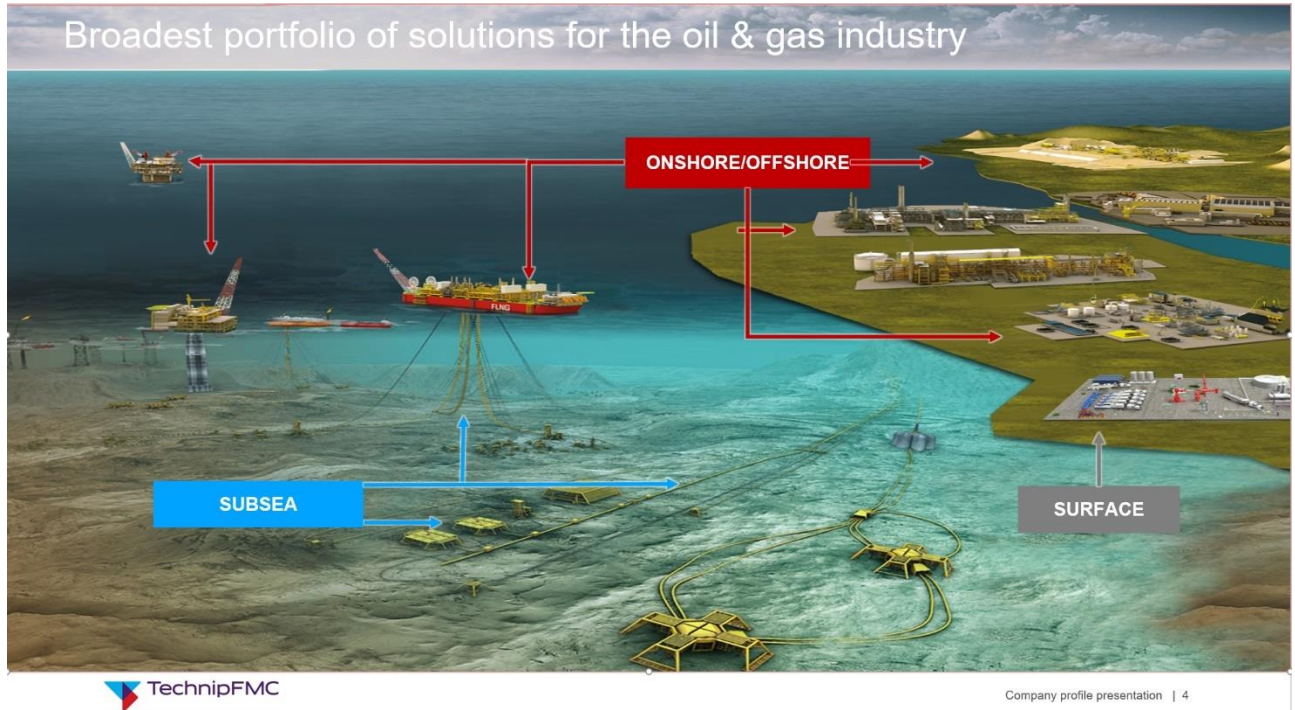


Figure 2: Portfolio of Oil and Gas Industry Solutions

The diagram below shows pictures of some offshore/onshore projects completed by TechnipFMC.

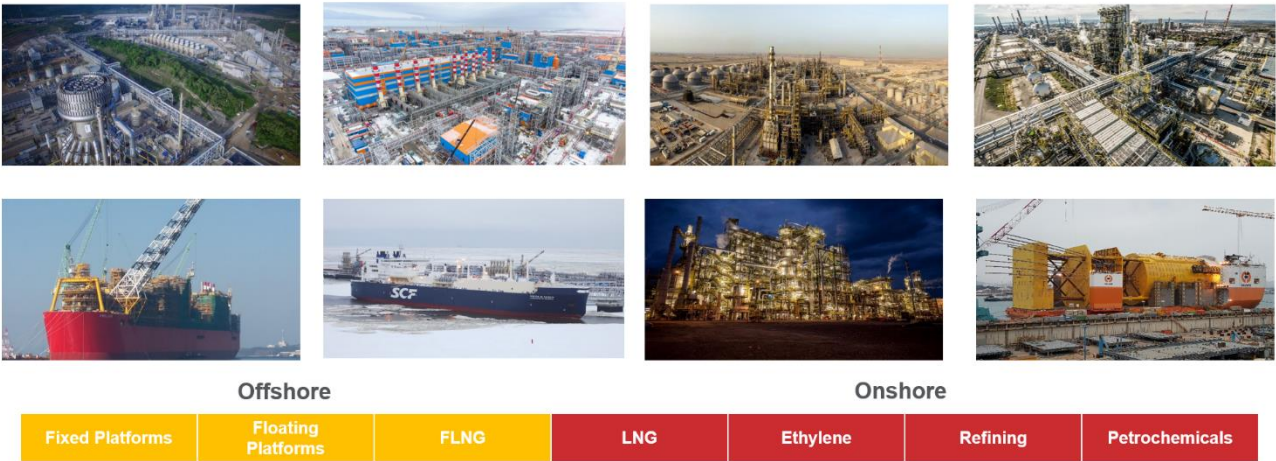


Figure 3: Complete Projects

TechnipFMC – Executive Leadership Team

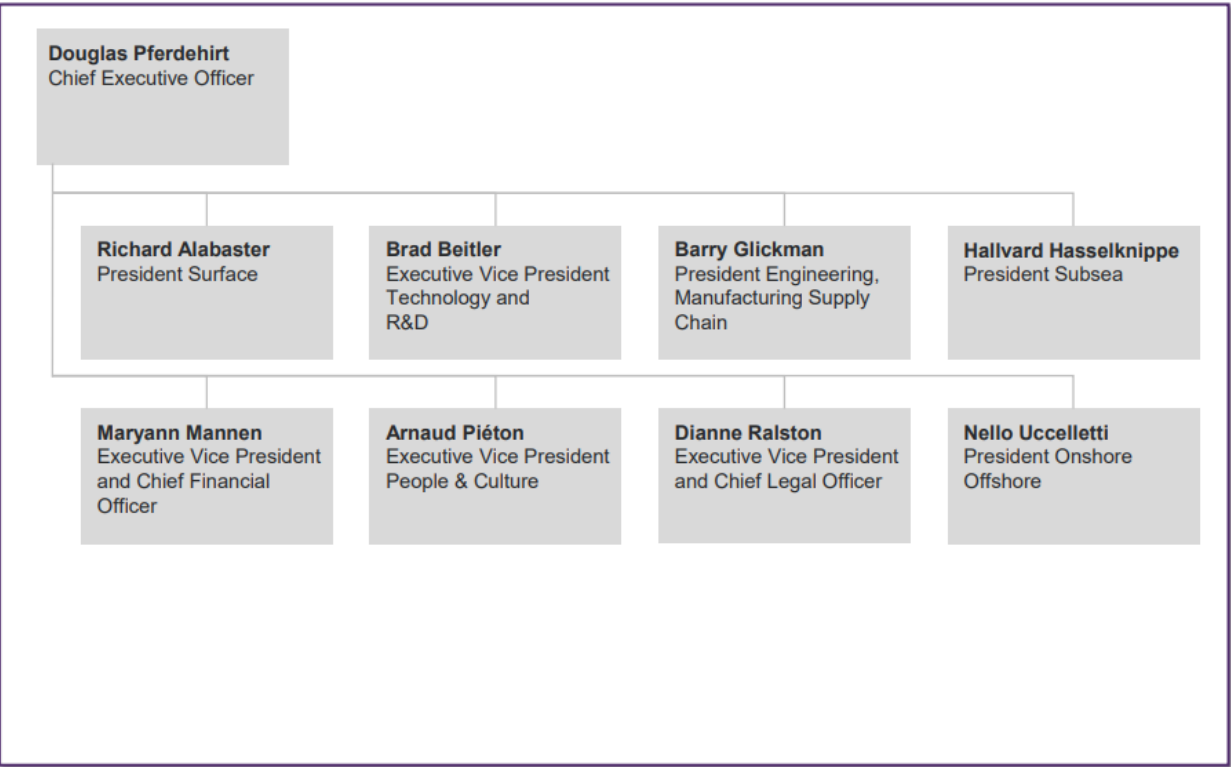


Figure 4: Executive Leadership Team's Organizational Chart

1.1 Organization's Principles

TechnipFMC's businesses are organized into three distinct Global Business Units (GBUs) that align with the way the clients operate and make decisions. Each GBU will set and direct global strategy within its segment and be responsible for building a market leading position. The GBUs are:

- Subsea
- Onshore/Offshore
- Surface

1.1.1 Engineering Manufacturing and Supply Chain organization (EMS)

In addition to the three GBUs, the EMS organization was created to help to ensure TechnipFMC's continued leadership position in the energy industry. As the name implies, EMS brings together management of selected engineering activities, Surface and Subsea manufacturing and Assembly, Surface and Subsea Product Lines (Including Product Delivery Teams and associated Engineering Centers) and Global Sourcing and Procurement. In addition, this organization will manage at the corporate level the Global Quality, health, Safety, Environment and Security (QHSES) and Integrated Marketing and Branding organizations.

All engineering activities relevant to Subsea FEEDs, systems, rigid pipes and installation will remain in the Subsea GBU. All engineering activities relevant to Offshore projects will remain as part of the Onshore-offshore GBU. Selected engineering activities associated in Research and Development will remain in the TR&D group.

1.1.2 Business Units

The GBUs are split into Business Units (BUs). The BUs are geographically focused, translating the GBU strategy to deliver customer success in any area where they operate. Each BU will be responsible for delivering separate business that will make up the GBU results. The GBUs will ensure global alignment of business goals and collaboration with BUs on activities such as commercial and tender reviews, project execution, key talent management, and client focus. They will set priorities and allocate resources (people and assets) between BUs and act as the final arbiter if conflicts and priorities between BUs.

1.1.3 Six Regional Business Units

One Business Unit in each geographic area has been designated to be the Regional Business Unit (RBU).

Table 1: Regional Business units designations

Region	BU Designated as RBU
Asia Pacific (APAC)	Subsea Asia Pacific RBU
Europe	Subsea Europe RBU
Africa	Subsea Africa RBU
North America (NAM)	Subsea North America RBU
South America (SAM)	Subsea South America RBU
Europe, Middle East, India & Africa	Onshore/Offshore Europe, Middle East, India & Africa RBU

1.1.4 Functions

The headquarters (HQ) and globally directed functions will direct strategy worldwide to be applied by BUs in accordance with overall company strategy and in alignment with the GBUs. The functions will set standards for process, procedures and cost allocation mechanisms to be applied on a company-wide basis.

1.1.4.1 Headquarter (HQ) Functions

- People and Culture
- Finance
- Legal
- Corporate Development and Digital
- Corporate Communications

1.1.4.2 Globally Directed Functions

- Information Technology (IT)
- Technology / R&D

1.1.5 Main RBUs

RBU EMIA (European, Middle East, India & Africa) is further divided into three main operating centers, which are controlling other local operating centers:

1.1.5.1 POC (*Paris Operating Center*)

- Paris
- Lyon
- Barcelona

1.1.5.2 IOC (*Indian Operating Center*)

- Chennai
- Mumbai
- Delhi

1.1.5.3 ROC (*Rome Operating Center*)

- Abu Dhabi
- Colombia
- Venezuela
- TPIDL
- RUS Technip
- Rome

1.1.5.3.1 The organizational structure of ROC

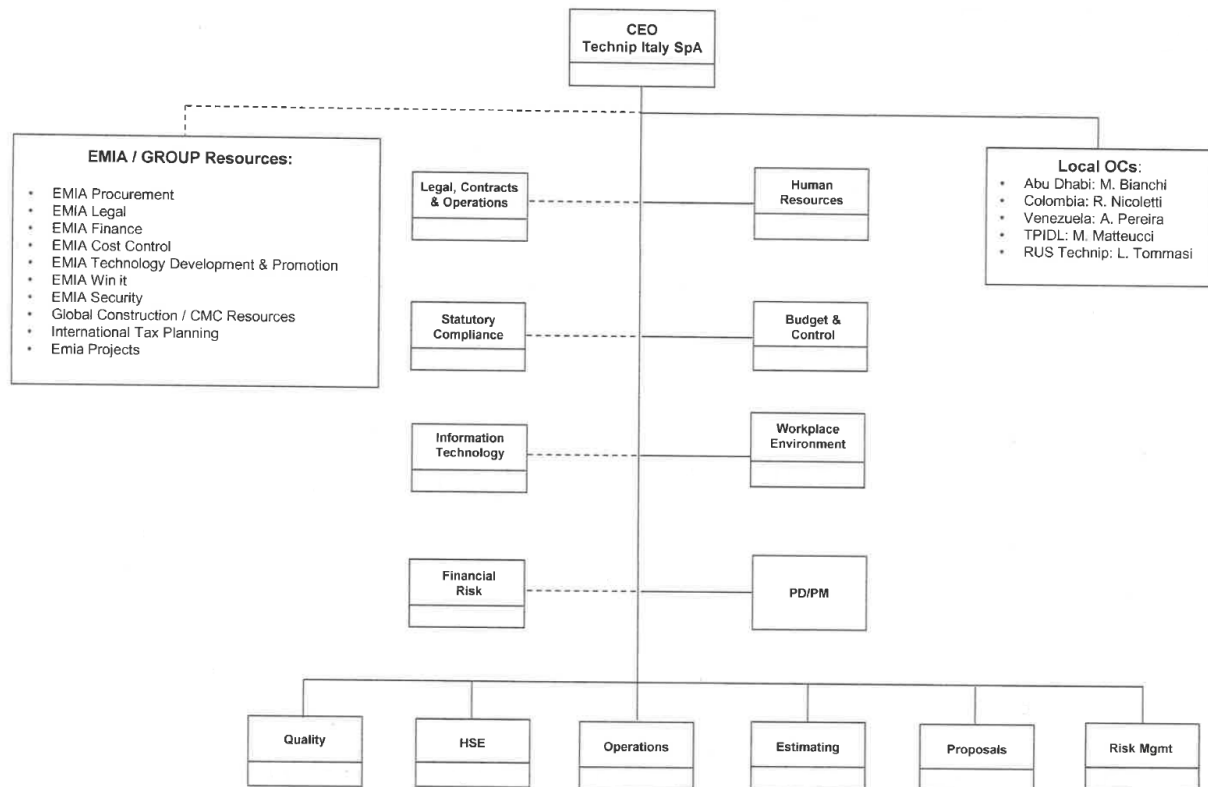


Figure 5: Technip Italy Organizational chart (October 2018)

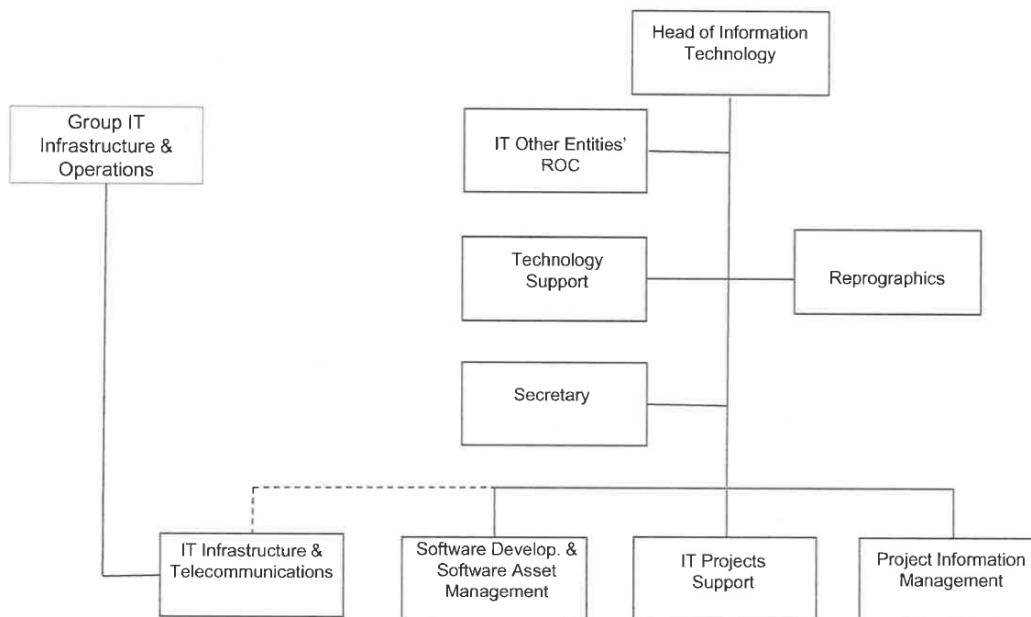


Figure 6: Technip Italy, IT Department, Organizational Chart (October 2018)

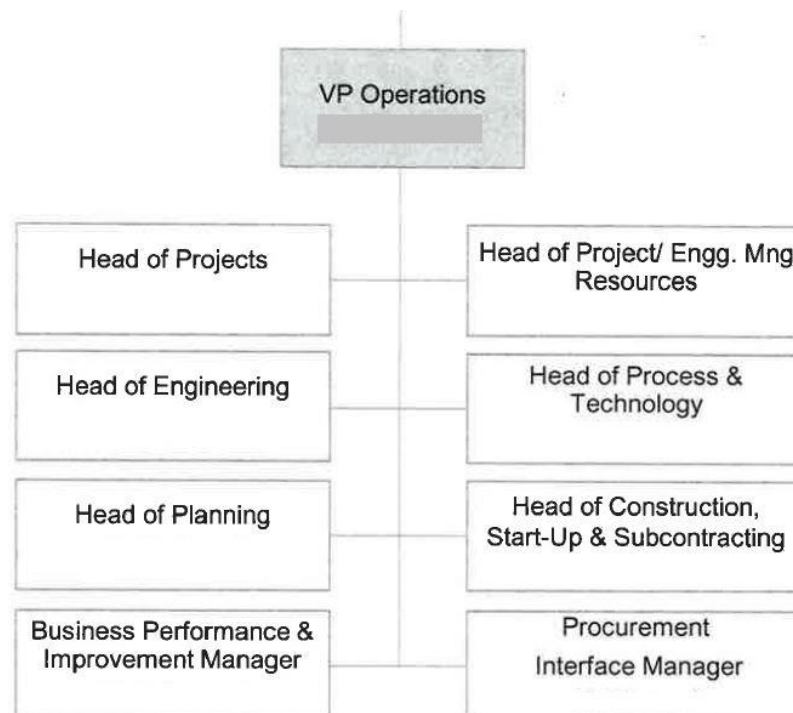


Figure 7: Technip Italy, Operations Department, Organizational Chart (October 2018)

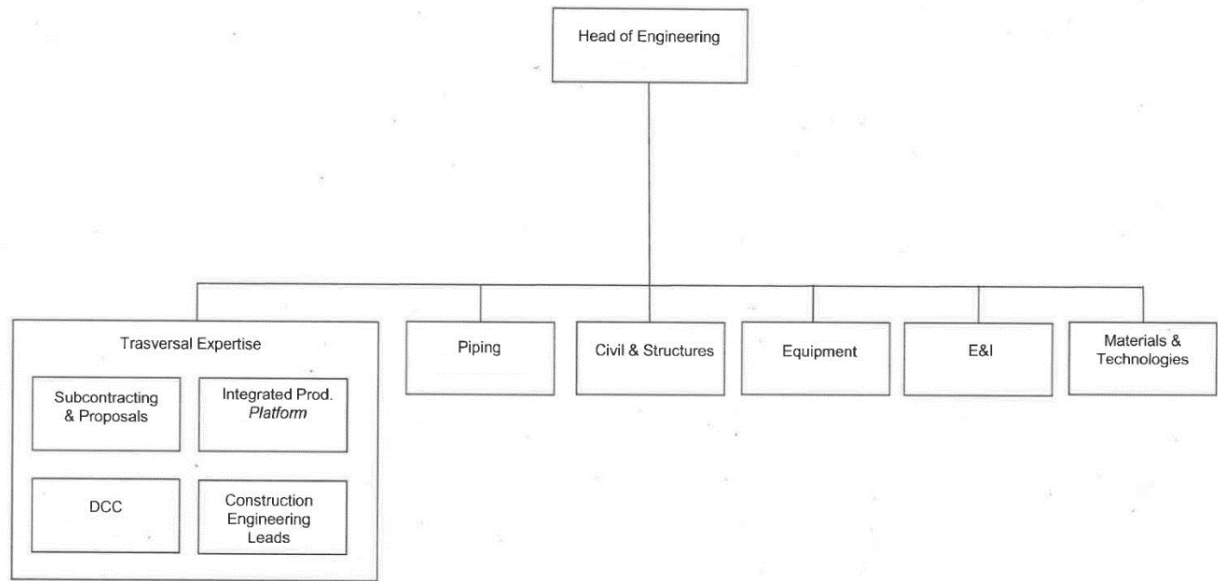


Figure 8: Technip Italy, Engineering Department, Organizational Chart (October 2018)

1.2 EPC

EPC stands for Engineering, Procurement, Construction and is a prominent form of contracting agreement in the construction industry. The engineering and construction contractor will carry out the detailed engineering design of the project, procure all the equipment and materials necessary, and then construct to deliver a functioning facility or asset to their clients. Companies that deliver EPC Projects are commonly referred to as EPC Contractors.

1.2.1 General Workflow

An engineering, procurement and construction company works in the typical flow given in the diagram below.

Typical project activities on an EPC company

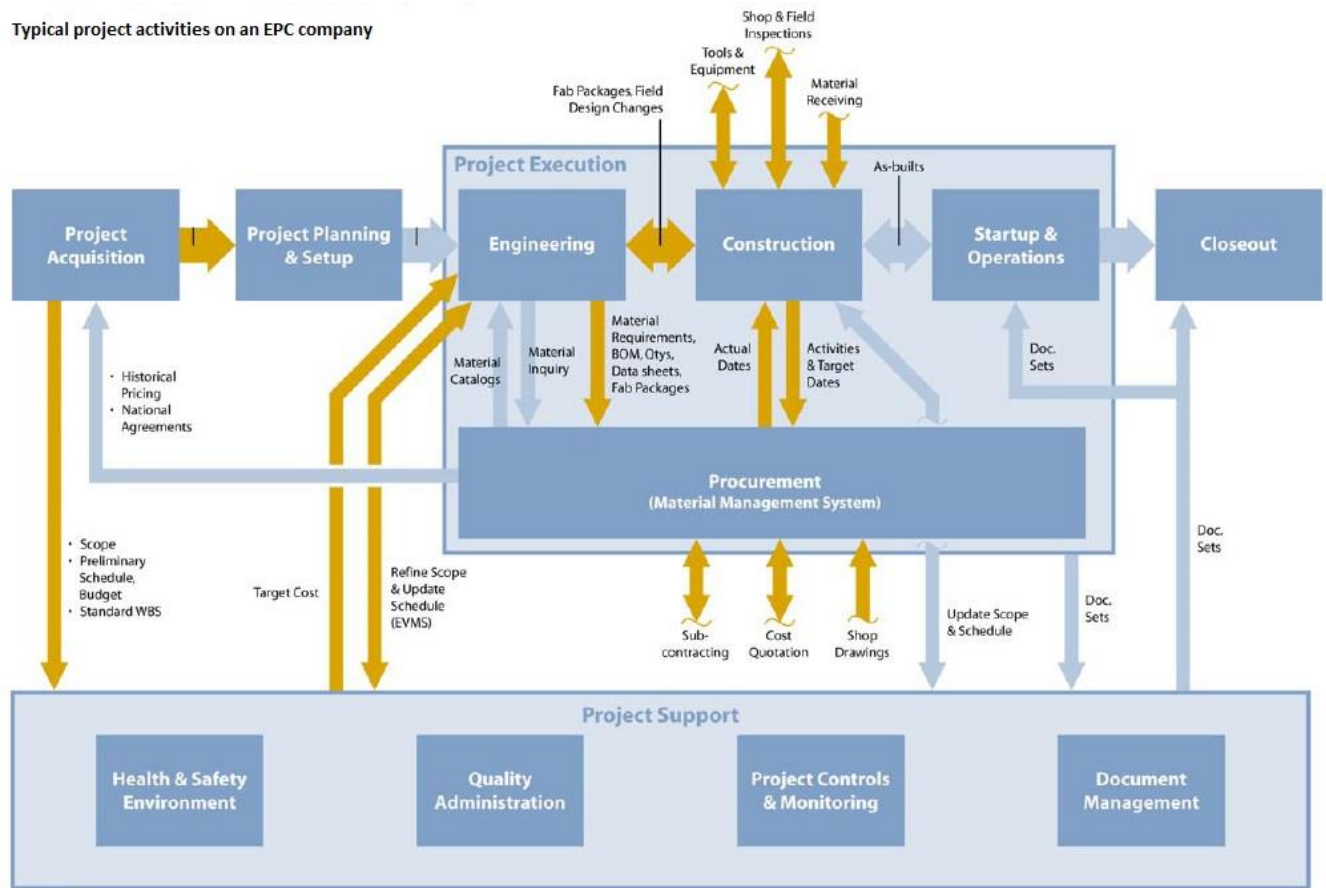


Figure 9: Typical Project activities on an EPC Company

After acquiring the project, the first phase is the planning phase. Where steps like materials selection and making initial 2D diagrams are done. This 2D diagram of the project is then converted to a 3D structure. After the Engineering department, it enters to Construction department. And finally, to Startup & Operations.

1.2.2 Detailed Explanation

It all starts with the functional requirements from the business. Process engineers perform process simulation and develop process schemes. Upon completion of these simulations and finding suitable process scheme, they issue the process flow diagram (PFD) and heat & Mass Balance (HMB). These documents form the basis for plant functional description. The heat & mass balance allows to identify and specify the duty of each equipment. Process department then performs the sizing of process equipment based on the flow of fluids that are being handled in such equipment i.e. either it's a gas flow or liquid flow. The process equipment sizing yields equipment dimensions together with the process flow diagram. Which shows connectivity between equipment. It allows to establish the plant layout. Heat & mass balance

allows to identify all types of fluids and their conditions in various pieces of equipment and pipes of the plant. This in turn allows to select the proper material of construction. Once the material of equipment of construction has been specified, the equipment can be purchased from the vendors and their mechanical design can be performed. This will result in equipment drawings, showing precisely all the information about equipment such as dimensions, the position of support, weights and load on foundation, the position of piping connections, etc. Such information will allow confronting the plant layout. The finalization of plant layout and information from equipment vendors such as equipment dimensions and weight allow performing the design of the process structure. The equipment bearing structures can be made of concrete or steel. The information from vendors also allows for designing the equipment foundation.

P&IDs (Piping & Instrumentation Diagrams) is an essential document in plant design as they depict all the operational features such as valves and lines and instruments that will be available for the operator to control to monitor the process. And for the operator to perform maintenance on equipment. Once P&IDs are established, pipes can be designed. The piping design entails the definition of a pipe routing that will meet many requirements including accessibility for the operator to operate the valves and to have access to instrumentation on the pipes. It also involves calculations of lines subject to thermal expansions. Such calculation determines the requirements for expansion loops and such expansion loops give rise to additional steel structures. Once piping calculations are completed and required expansion loops are identified, the corresponding loads and geometry are given by piping to civil.

Once the calculations have been completed and a significant number of pipes have been routed in the 3D model, the piping materials takeoff (a list of materials) can be extracted from the model. This will allow the first set of piping materials to be purchased. Before piping construction design, isometric drawings can be issued. Information is needed from equipment vendors about the precise position of equipment nozzles. The P&ID diagram defines precisely the instrumentation and control. Therefore, they are also the basis for the instrument and control engineer to specify and purchase the field instrumentation as well as the control system. The electrical design starts with the inventory of all the plant parallel consumers. This allows to size the power generation. The plant layout is used to define the architecture of the electrical distributions and the number of substations. Power cables are sized, and their routes are defined. Since such cables are underground, such information is provided to civil engineer which performs the underground network layout drawing. Quite a few networks are located on the ground and the civil engineer coordinates their respective positions. The network includes the electrical and instrumentation cables, the underground piping services, such as the drains, process drains, pip support foundations, the fire water network designed by the safety engineer. As well as the plant rainwater drainage network.

The above explanation can be visualized in the diagram below.

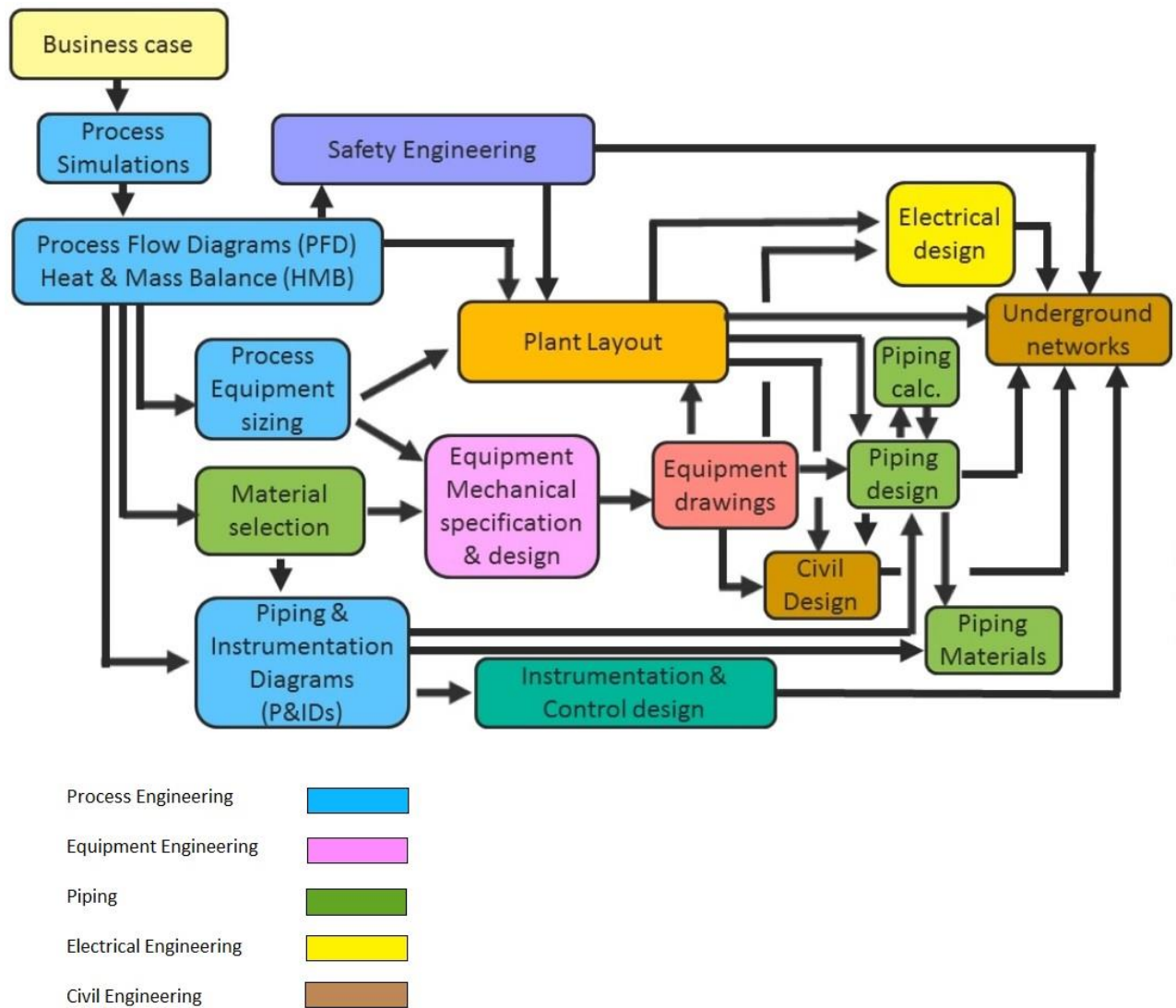


Figure 10: General Workflow of an EPC company

1.2.3 Activities in Time Frame

The Diagram below shows the detailed workflow of an EPC company in different time frames. Different colors in this workflow represent different departments.

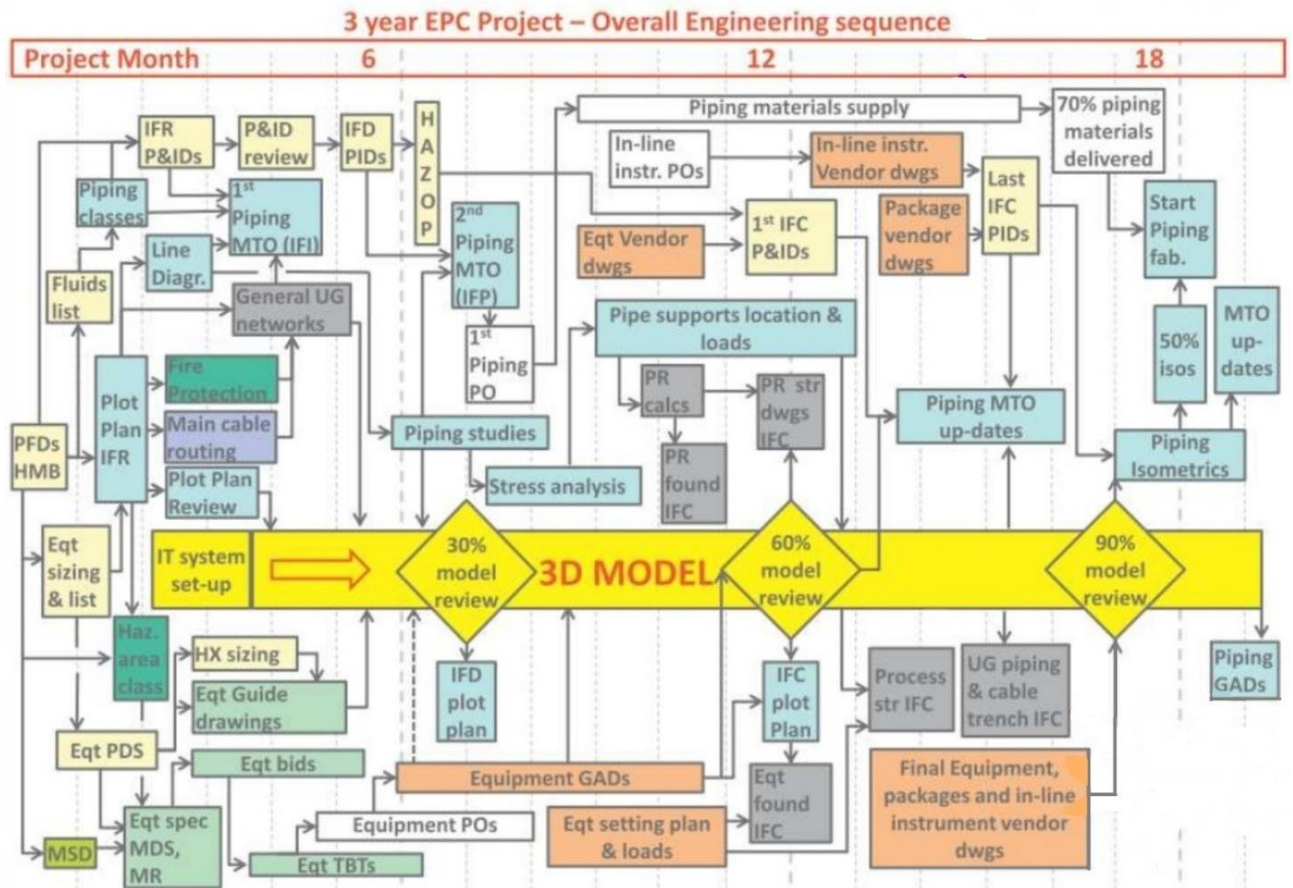


Figure 11: Activities in Time Frame of an EPC Project

1.2.4 Application Mapping of Data Work Flow

This diagram shows the most important documents participating in the PIPING WORKFLOW (stress analysis is not included) highlighting software application used for their management.

In this thesis I will work on main data involved in the PIPING WORKFLOW only from three main applications: Intergraph Smart Materials (SPMAT), Smart Plant Piping and Instrumentation Diagram (SPP&ID) and Intergraph Smart 3D(S3D).

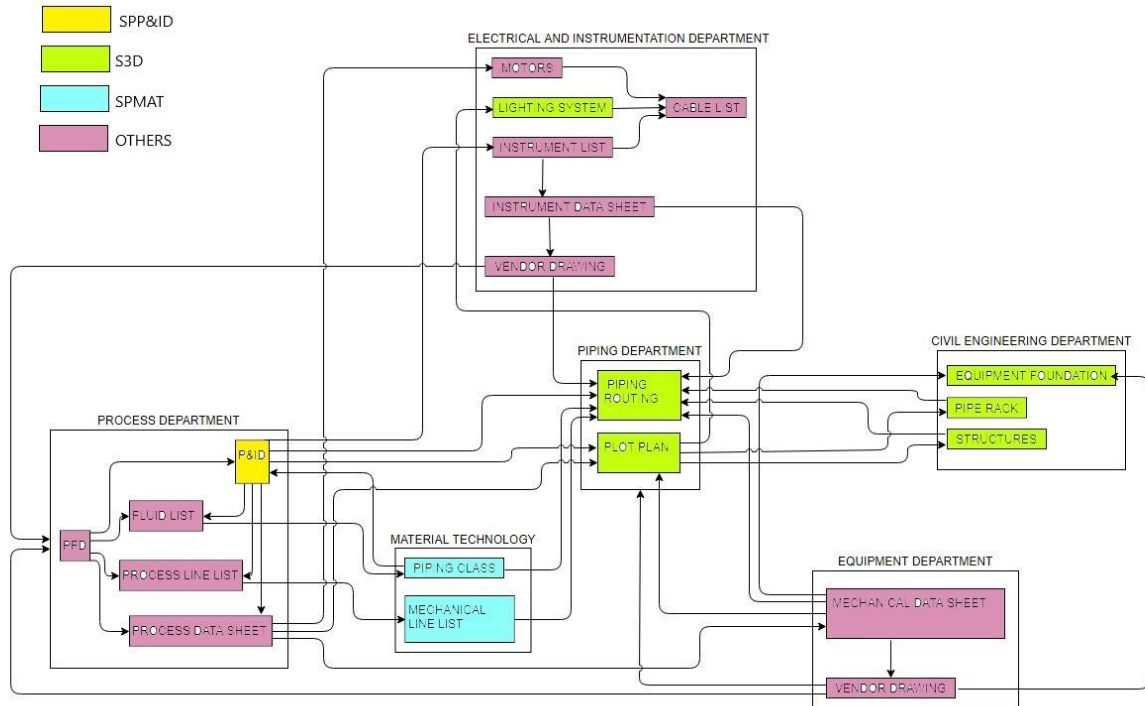


Figure 12: Application Mapping of Data Workflow

1.3 Piping

The major portion of any project performed by TechnipFMC involves ‘piping’. **Piping** is a system of pipes used to convey fluids (liquids and gases) from one location to another. The engineering discipline of piping design studies the efficient transport of fluid. Industrial process piping (and accompanying in-line components) can be manufactured from wood, fiberglass, glass, steel, aluminum, plastic, copper and concrete. The in-line components, known as fittings, valves, and other devices, typically sense and control the pressure, flow rate and temperature of the transmitted fluid, and usually are included in the field of piping design (or piping engineering). Piping systems are documented in the piping and instrumentation diagram (P&IDs). If necessary, pipes can be cleaned by the tube cleaning process.

Piping systems in a chemical plant (refineries, petrochemical plant, fertilizer plant, gas plant, etc.) are comparable to the arteries and veins through which fluid, vapors, slurries, solids, etc. flow under various conditions as imposed by the process design of the plant. The piping network is subjected to almost all the severest conditions of the plant such as high temperature, pressure, flow and combination of these. In addition to the above, corrosion, erosion, toxic conditions and radioactivity add to more problems and difficulties in piping design. With the advancement of process design and technological development, a

continuous effort is required to be carried on coping up with the demands of the process. This makes the job of a piping engineer more complex and responsible.

In almost all chemical industries, the installed capital cost of piping is a major factor in plant investment. As far as material procurement is concerned, excluding equipment costs, piping is the largest plant cost component. It is also observed that piping exceeds all other field costs by a substantial amount. Also, design engineering utilizes approximately 45% of engineering man-hours and 50% of these man-hours are used in piping design.

1.4 Software Applications

Piping sometimes refers to piping design, the detailed specification of the physical piping layout within a process plant or commercial building. In earlier days, this was sometimes called drafting, technical drawing, engineering drawing, and design, but is today commonly performed by designers that have learned to use an automated computer-aided drawing or computer-aided design (CAD) software. However, TechnipFMC uses typical software that is made for specific purposes. The main softwares used by TechnipFMC are provided by HexagonPPM.

HexagonPPM is the world's leading provider of enterprise engineering software and project control solutions. The main softwares used by TechnipFMC are the products of HexagonPPM. TechnipFMC is using the whole suite of softwares released by HexagonPPM. However, the three main tools involved in the “data management of Piping workflow” are **Intergraph Smart® Materials**, **SmartPlant® P&ID**, **Intergraph Smart® 3D**.

1.4.1 Intergraph Smart® Materials

Intergraph Smart® Materials is an integrated solution for life cycle material, supply chain, and subcontract management. It provides a common collaboration platform and project workbench for all partners in any engineering, procurement, and construction (EPC) project supply chain.

Intergraph Smart Materials helps to lower project costs, compress schedules, improve risk management, and enable companies to act globally to maintain an advantage in a highly complex, international, and competitive market. From initial cost estimation through the supply chain to on-site management, Smart Materials handles Piping Standardization, Piping Classes, Bill of Materials (Material Take Off), Mechanical Line List (and Calculation Module).

1.4.2 SmartPlant® P&ID

SmartPlant® P&ID helps you develop and manage your piping and instrumentation diagrams with a focus on the plant asset. The piping and instrumentation diagram, or P&ID, is the "process roadmap" of the plant and is developed, accessed, shared, and modified throughout the plant life cycle. Therefore, it is critical that the P&ID is kept up-to-date, accurately reflecting the as-built plant because other disciplines base their design decisions on the data from P&IDs.

SmartPlant® P&ID helps you develop and manage your P&IDs with a focus on the plant asset rather than the document representation. It exercises rules and connectivity checks to speed the entire engineering process, helping you save money without compromising design quality or integrity. SmartPlant P&ID also facilitates faster project startup because several engineering standards are included with the software.

In addition to the engineering phase, the P&ID plays a key role in operational tasks such as safety, inspections, turn-around planning and much more.

1.4.3 Intergraph Smart® 3D

Intergraph Smart® 3D – a next generation, data-centric, and rule-driven solution – is specifically designed to deliver mission-critical requirements. Breaking through barriers imposed by traditional technologies to enable a truly iterative design environment, Smart 3D provides a competitive edge to EPCs and owner-operators alike.

A fundamental component of SmartPlant® Enterprise, Intergraph Smart 3D provides all the capabilities needed to design plant, marine, and materials handling facilities, and then maintain their 3D “as-built” representations. Take advantage of data-centric technology, a strong rule- and relationship-based architecture, customized automation capabilities, and an integrated reuse approach to execute even the largest and most complex projects with centralized visibility and control.

All these three softwares are licensed softwares. Software customization is standard practice and is generally applied to all tools. Software customizations are always propriety of TechnipFMC even if those customizations are performed by the third party.

1.5 Data

1.5.1 Data Centers

There are three main shared data centers of TechnipFMC worldwide. These data centers provide data globally to all the graphical users, those accessing data from different softwares provided. These three data centers are:

- EDC (European Data Center) in Paris
- ADC (American Data Center) in Houston
- MDC (Malaysian Data Center) in Kuala Lumpur

In addition to main shared data centers, each main Operating Centre has its own data center.

1.5.2 Data accessibility

The databases at the backend to these applications are provided by the software house to TechnipFMC for direct querying purposes. For Smart3d and P&ID, a new database in the local data center is generated for every new project. For Smart Materials database instead, there exists a single instance for the whole company and all projects are stored in EDC. Every user accessing Smart Materials all around the world will access the same single instance of the database.

1.6 The problem statement:

The company has several data generating sources. Different applications are used during different phases of a project. These applications generate data that is different but somehow connected. There is vertical and horizontal data transfer between different applications. Retrieval and exchange of information at certain levels during the execution of a project is the area that particularly needs improvements. There can be several possibilities to observe these improvements. The company is trying to figure out the best possible solution that can be implemented to improve this data management. Costs and efficiency are the key points in deciding the best possible solution. Two possible solutions to be considered are:

Data Lake: A data lake is a centralized repository that allows you to store all your structured and unstructured data at any scale. You can store your data as-is, without having to first structure the data, and run different types of analytics, from dashboards and visualizations to big data processing, real-time analytics, and machine learning to guide better decisions.

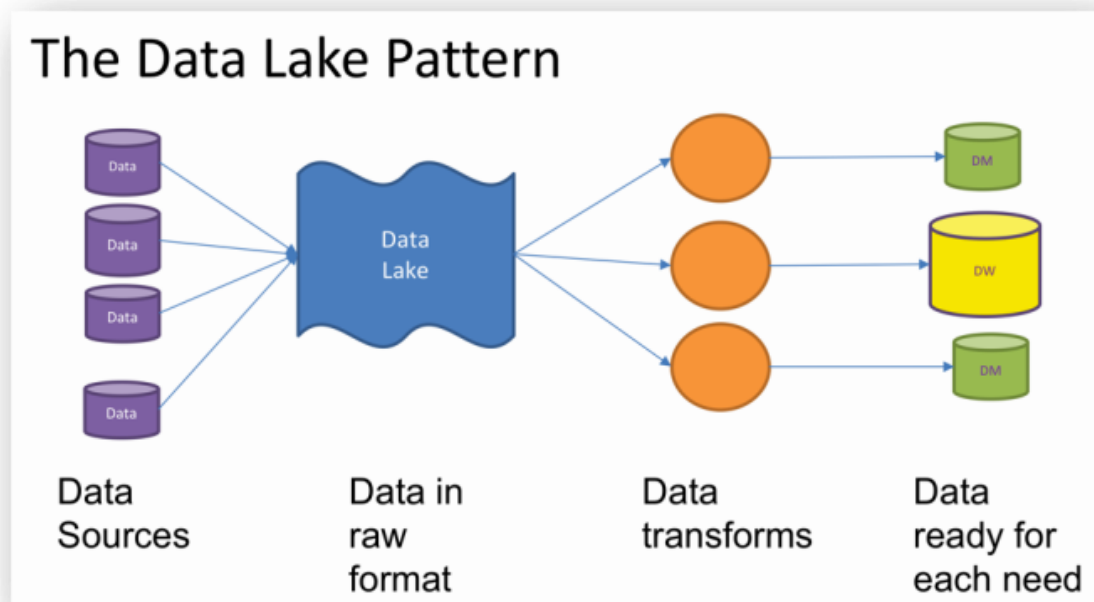


Figure 13: A Data Lake

Data Warehouse: A data warehouse is a system that pulls together data from many different sources within an organization for reporting and analysis. The reports created from complex queries within a data warehouse are used to make business decisions. In more comprehensive terms, a data warehouse is a consolidated view of either a physical or logical data repository collected from various systems. The primary focus of a data warehouse is to provide a correlation between data from existing systems.

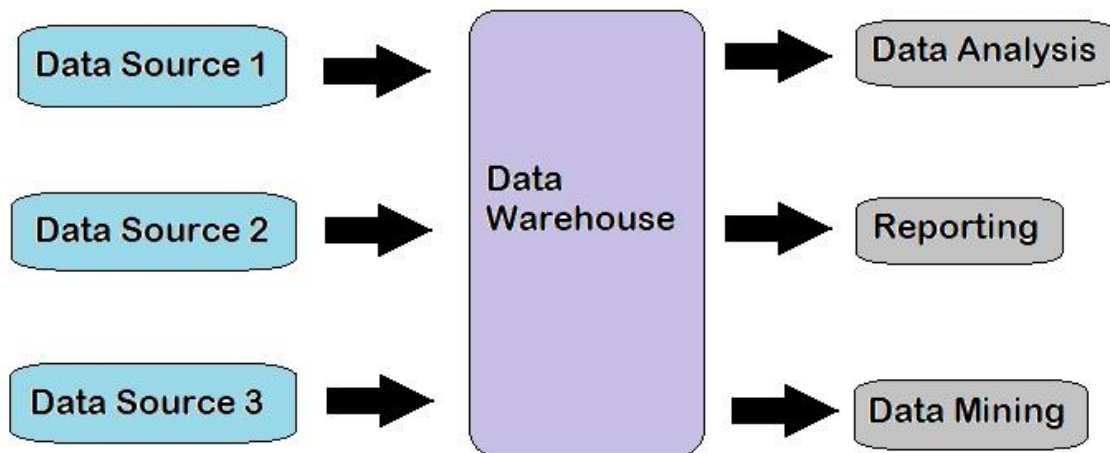


Figure 14: A Data Warehouse

While working with all databases, we realized that some databases are very complex and huge. And they are growing very rapidly. The data they contain include much extra information, which is not even required by us for report generation and further data analysis purposes. So, it was not necessary to fetch all data from all sources as 'Data Lake'. Hence, we decided to work with '**Data Warehouse**' methodology i.e. fetching only selective data, that will be useful in further report generation and analysis transform it and store it.

1.6.1 Issues

Data misalignment: The major issue is with misalignment and incompleteness of data. There is no automated connection of data flow between main applications. There are only manual ways to check if data in different sources are aligned or not. This problem reduces the pace of work in the work field. Different teams must wait for certain checks to be performed by some other teams before they can start working. There is a need to automate this data quality check to increase the pace of work.

Data Availability: The second issue is data related to information retrieval. The company has a very complex and huge amount of data stored in different databases. The report generation can involve several databases meantime. The report generation using GUI's can take many hours to have the results. Which is a huge waste of time while constructing big and complex projects. The data is being generated at an enormous rate. So, there is a need to tackle this information retrieval problem in an efficient way.

1.6.2 Data misalignment

The data we are interested in is related to pipelines. So, in this thesis, our focus will remain on data related to pipelines. The evolution of line list is from process line list to mechanical line list to construction line list. It takes 2 weeks for the Process department to prepare a process line list after P&ID completion. This document is not used by designers. It is used by the coordinators of the activity to understand all the information at a higher level. All the information like name of the line, name of connected equipment, diameter, piping class, etc. required by the designers for line list is present in P&ID. So, they can fetch this information directly from there.

Process Line List is used as input from Piping Material Specialists to apply the mechanical calculation to produce Construction Line List

Once the construction line list is ready, it is exported as an excel file and then imported to S3D. After that, the isometrics can be extracted by designers from S3D through an automated process. The isometric extraction process reads the CLL data that are present in S3D. The data in these isometrics and CLL is not aligned sometimes. These misalignments between CLL and isometrics from S3D can create problems and slow down the pace of work.

There are problems with materials also. Materials are more complex because you must consider the correctness of ident. Ident is a unique code for each component assigned automatically. Materials are checked in terms of their description. Their description is checked not to be incomplete or wrong.

1.6.2.1 Current Practices

The way we control these checks depends on the phase. If it is just some intermediate phase, the visual check is enough. Isometrics are not only the graphical part, but they also contain detailed information in them. This information comes from SP3D. Our data is organized in pipe runs. We take some of the entire data related to each pipe run from the Construction Line List and export it to SP3D. The data is not stored in the database if that pipe run doesn't already exist in 3D. The simple visual check can be performed by checking if data is present or not, or if data is complete or not. But if these isometrics are issued for the construction or some critical phase, we need to be sure of its quality and for that purpose, there is a dedicated team to perform manual checks which can take a lot more time. It verifies all elements in P&ID are present and aligned in Isometrics.

To be sure of material alignment we perform a simulation of data loading in smart plant materials. to check the assistance in that condition of the ident (unique code) that whether

the chain of data is good. Several errors can be caught during this simulation. Many activities of control must be performed to arrive on the conclusion.

1.6.2.2 Problems raised by data misalignment

This data misalignment raises the following problems to be solved.

- 1) Detection of inconsistent lines ID between the 3D model and CLL
- 2) Detection of lines with inconsistent Piping Class between the 3D model and CLL
- 3) Detection of existing Diameters of lines not defined on CLL
- 4) Detection of lines containing data which will be printed on isometrics which are not aligned with CLL.
- 5) Detection of components in the 3D model which needs an update

1.6.2.3 Solutions

- 1) Verify the alignment between the 3D model and CLL by using the tuple Project-Unit-Area-Line
- 2) Verify the alignment between the 3D model and CLL by using the tuple Project-Unit-Area-Line-Piping Class
- 3) Verify the alignment between the 3D model and CLL by using the tuple Project-Unit-Area-Line-Piping Class-Diameter
- 4) Verify consistency of data used in the isometric layout between CLL and in the 3D model (existence and congruence)
- 5) Check "update date" of existing components in the catalog DB with "update date" of components existing in the 3D model DB. Highlighting components (and their related Area\Lines) on the 3D model DB where "update date" is older than the relevant component in the Catalogue DB.

1.6.3 Data Availability

The data in three main databases are generated from the graphical interfaces of 3 main applications i.e. SPMAT, SP3D, SPPID. As these applications belong to a third party, the complete structure of databases is not provided. Anyhow the information needed to generate reports by querying from these databases is known. Therefore, it is hard to understand and restructure the databases completely. So, there must be some alternative way to reduce this data retrieval time.

1.6.3.1 Solution

The company took help of some external software houses that are specialists in dealing with these applications to build efficient queries that can provide the same information as done by reports generated from the graphical interface. They wrote queries for both Oracle and SQL Server databases and that reduced the time to a few minutes.

We did further enhancement by automating these queries and running them parallelly to save more time. This task was performed by writing a piece of code in Python language. which reduced the time further from 15 minutes to 8 minutes.

Another hypothetical solution is using newer database solutions. We decided to convert our databases to MongoDB to compare data retrieval times. Another potential advantage for this solution will be reduced costs of software licenses, since using MongoDB is far cheaper than using SQL.

2 Conversion from SQL to NoSQL

2.1 Smart Intergraph Materials Database

This database is an Oracle instance. This is the first database we chose to work on. As mentioned earlier, this database has a single copy to be used all around the world. However, a weekly copy of its data is generated and copied in the European data center for test/development access. We were given a read-only access to this copy of the original database. Due to the company's policies, it is not possible to have a personal copy of this instance of the database. So, we must fetch all the records from that database.

2.1.1 cx_Oracle

cx_Oracle is a Python extension module which enables access to Oracle Database. It conforms to the Python database API 2.0 specification with a considerable number of additions and a couple of exclusions. I used this module for connection between Python and Oracle.

2.1.2 Database overview

Without knowing any prior information about this database like a number of tables, users, names of tables, their columns, etc. We start retrieving basic information like a number of tables, columns in every table and the total number of records in all those tables. Knowing this information, we make categories of tables. Depending on how many records every table has. I haven't worked before with an Oracle database, so it took a little longer for us to understand the things and find the possible solutions to the problems we were facing. The number of non-system tables in this database is 2897. The smallest table has as low as 10's of records and the largest table contains more than 200 million records. The database in total has more than a billion records and increasing rapidly. Talking about the number of columns in every table, it can vary from less than 10 columns to around 350 columns in a single table. There are more than 10 different data types used in this database and 247 users.

There is a total of 5 different categories based on a number of records in each table. The outer bounds of these categories and the number of tables in each category are given below:

Table 2: SPMAT Table categories

Category Name	Number of Rows	No. of tables
Category0	0 to 10,000 (zero to ten thousand)	1062
Category1	10,000 to 100,000 (ten thousand to hundred thousand)	131
Category2	100,000 to 1,000,000 (hundred thousand to one million)	82
Category3	1,000,000 to 10,000,000 (one million to 10 million)	43
Category4	10,000,000 + (more than 10 million)	17

Large Object data types are also used in some tables, which can store data up to 4GBs.

It took almost one month to download and convert the entire database to MongoDB.

Note: These statistics were taken 24/09/2018.

2.1.3 Data Retrieval

Working with this database was most hectic and time taking process. While fetching records, the main thing we had to keep in mind was the amount of ram we needed to store and convert those records to MongoDB. Tables having a large number of records couldn't be fetched entirely at once. So, we had to fetch a batch of records at once, that our ram can easily store and convert to MongoDB. Tables with lesser records were very fast to be fetched however tables with a large number of records took very long to retrieve data from. Fetching all records from a table having 500,000 records was quicker than fetching 500,000 records from a table containing millions of records.

2.1.4 Conversion to MongoDB

Conversion of data to MongoDB was more complex than just retrieving it. It took us long to understand that the tables having a large number of columns and large objects take a very long time to be processed. After recalculation of our estimated time several times, we realized that we need more machines to increase the pace of data retrieval and conversion to MongoDB. So, we were provided with a second PC and later two more remote PCs to work on. It took almost one month to process the single table with more than 300 columns in category4. This task could be accomplished in less time if the code was run on a cloud with a

large number of powerful processors and rams. But due to the inaccessibility of data outside the company's private network, it was impossible to do that.

2.1.4.1 Changing tables name

MongoDB doesn't accept '\$' sign in table names, however, in Oracle database, there are some tables that include '\$' sign in their names. So, we changed this '\$' sign to 'dollar' in MongoDB. For example, the name "table\$name" was changed to "tabledollarname" in MongoDB.

2.1.4.2 Conversion of Data Types

To make things simpler we have converted most of the datatypes to string from Oracle database to MongoDB. But there were some data types that contain large objects up to 4GBs. Such data types can be converted to a string, neither in any other datatype. Since MongoDB just stores objects in BSON format and is limited to 16MB. To store such large objects in MongoDB we used GridFS.

GridFS is a specification for storing and retrieving files that exceed the BSON-document size limit of 16 MB. Instead of storing a file in a single document, GridFS divides the file into parts, or chunks, and stores each chunk as a separate document. By default, GridFS uses a default chunk size of 255 kB; that is, GridFS divides a file into chunks of 255 kB except for the last chunk. The last chunk is only as large as necessary. Similarly, files that are no larger than the chunk size only have a final chunk, using only as much space as needed plus some additional metadata.

GridFS uses two collections to store files. One collection stores the file chunks, and the other stores file metadata. The section GridFS Collections describes each collection in detail. When you query GridFS for a file, the driver will reassemble the chunks as needed. You can perform range queries on files stored through GridFS. You can also access information from arbitrary sections of files, such as to "skip" to the middle of a video or audio file. GridFS is useful not only for storing files that exceed 16 MB but also for storing any files for which you want to access without having to load the entire file into memory. See also When to Use GridFS.

Large objects found in this database are:

BLOB, CLOB -> 4GB

LONG, LONGRAW -> 2GB

2.1.4.3 Encoding Errors

There are some miss-entered special characters in the database. Since data updated to this database is from different countries, so these special characters belong to different languages. Theoretically, we should be able to encode/decode these characters with “utf-8” encoding. But practically some of them are not encoded/decoded by “utf-8”. And “utf8” solves the problem for those not read by “utf-8”. In short, “utf-8” and “utf8” encode/decode different special characters. we spent many days in trying to understand and find a mature solution to this problem. According to all the information we got, “utf-8” and “utf8” are different names for the same encoding. But practically in our case, it's different. There isn't any method provided by the library “cx_Oracle” so far that can “replace” or “ignore” such characters.

In this database almost, all the tables were encoded/decoded by either “utf-8” or “utf8”. But there were two such tables that have mixed data of both encodings. To retrieve data from these tables we wrote a different function. That fetches single record at a time and tries if it is encoded/decoded by “utf-8”, else it uses “utf8”.

2.2 Intergraph Smart® 3D

This database is a Microsoft SQL Server instance. This is the second database I chose to work on. An instance of this database was placed in Rome's data center for me to have direct access to its data. The database contains data related to one project only.

2.2.1 Pyodbc

Pyodbc is an open source Python module that makes accessing ODBC (open database connectivity) databases simple. It implements the DB API 2.0 specification. Using Pyodbc, you can easily connect Python applications to data sources with an ODBC driver. As with other application stacks connecting through the ODBC API, the application, in this case, your Python code along with the Pyodbc module will use an ODBC driver manager and ODBC driver. The ODBC driver manager is platform-specific, while the ODBC driver is database-specific. The ODBC driver manager and driver will connect, typically over a network, to the database server. We used this module to connect and work with Microsoft SQL Server database.

2.2.2 Database Overview

Six schemas of this database were made available to us.

"S3D16TEST_SITE"
"S3D16TEST_SITE_SCHEMA"
"TENOC16_CAT"
"TENOC16_CAT_SCHEMA"
"TENOC16_MDB"
"TENOC16_RDB"
"TENOC16_RDB_SCHEMA"

Without knowing any prior information about this database like a number of tables, users, names of tables, their columns, etc. We start retrieving basic information like a number of tables, columns in every table and the total number of records in all those tables. Knowing this information, we make categories of tables.

There are total of 5 different categories based on a number of records in each table. The outer bounds of these categories are given below:

Table 3: S3D Table Categories

Category Name	Number of Rows	No. of tables
Category0	0 to 10,000 (zero to ten thousand)	199
Category1	10,000 to 100,000 (ten thousand to hundred thousand)	51
Category2	100,000 to 1,000,000 (hundred thousand to one million)	42
Category3	1,000,000 to 10,000,000 (one million to 10 million)	13
Category4	10,000,000 + (more than 10 million)	2

The data required in our study was contained in "TENOC16_MDB".

2.2.3 Data retrieval and conversion to MongoDB

Data retrieval and conversion to MongoDB was not that difficult as the previous database. It took less than 2 days to complete the entire process.

3 Data Availability

One of the major problems with data is its availability. Usually, the information is acquired in the form of reports. These reports are generated with the help of graphical interfaces provided by the respective applications. But the problem we are facing is that the report generation process can take execution time from hours to days. Specifically, a single isometric emission report related to a single unit can take almost one day. And if reports for 15 units are to be generated, it can take almost 15 days to complete. Which is a huge waste of time?

3.1 Comparison between SQL and NoSQL

Now that we have data in SQL as well as in NoSQL format. We can compare their efficiency by applying the same queries to them.

3.1.1 SQL

We wrote specific queries to have direct access to the data we require. These queries were engineered in collaboration with specialists from the company. The results were further filtered to have the same results as in the reports generated by the application interface. More specifically reports related to isometric emission. These carefully engineered queries reduced this time of acquiring information from days to a few minutes. So, now it takes just 25 minutes to have the desired information instead of many days.

3.1.1.1 Parallel Processing

To reduce the information retrieval time further. We wrote a code in python that runs all the queries parallelly using processes. By performing this step, I was able to reduce the information retrieval time further by two third. Running these queries manually took 15 minutes almost. Which was reduced to 8 minutes using parallel processing using python? Running them through the python interface, not only made this process automated. But also saved a few more minutes.

Code (Python):

```
if __name__ == '__main__':  
    fin = Value('i', 0)  
  
    p1 = Process(target=CAT_COMMODITYQ, args=(fin,))
```

```

p2 = Process(target=MOD_AREASQ, args=(fin,))
p3 = Process(target=MOD_OBJECTSQ, args=(fin,))
p4 = Process(target=MOD_PIPERUNDATA_CLLQ, args=(fin,))
p5 = Process(target=MOD_PIPERUNSQ, args=(fin,))
p6 = Process(target=TNP_IT_DV_BOM_DETAIL, args=(fin,))
p7 = Process(target=TNP_IT_DV_FR_LINES, args=(fin,))
p8 = Process(target=TNP_IT_DV_PIPING_BOM, args=(fin,))
p9 = Process(target=TNP_IT_DV_PLL, args=(fin,))
p10 = Process(target=REF_CLL, args=(fin,))

p1.start()
p2.start()
p3.start()
p4.start()
p5.start()
p6.start()
p7.start()
p8.start()
p9.start()
p10.start()

p1.join()
p2.join()
p3.join()
p4.join()
p5.join()
p6.join()
p7.join()
p8.join()
p9.join()
p10.join()

```

The detailed processing time is given below:

Table 4: Detailed Processing Time of each query

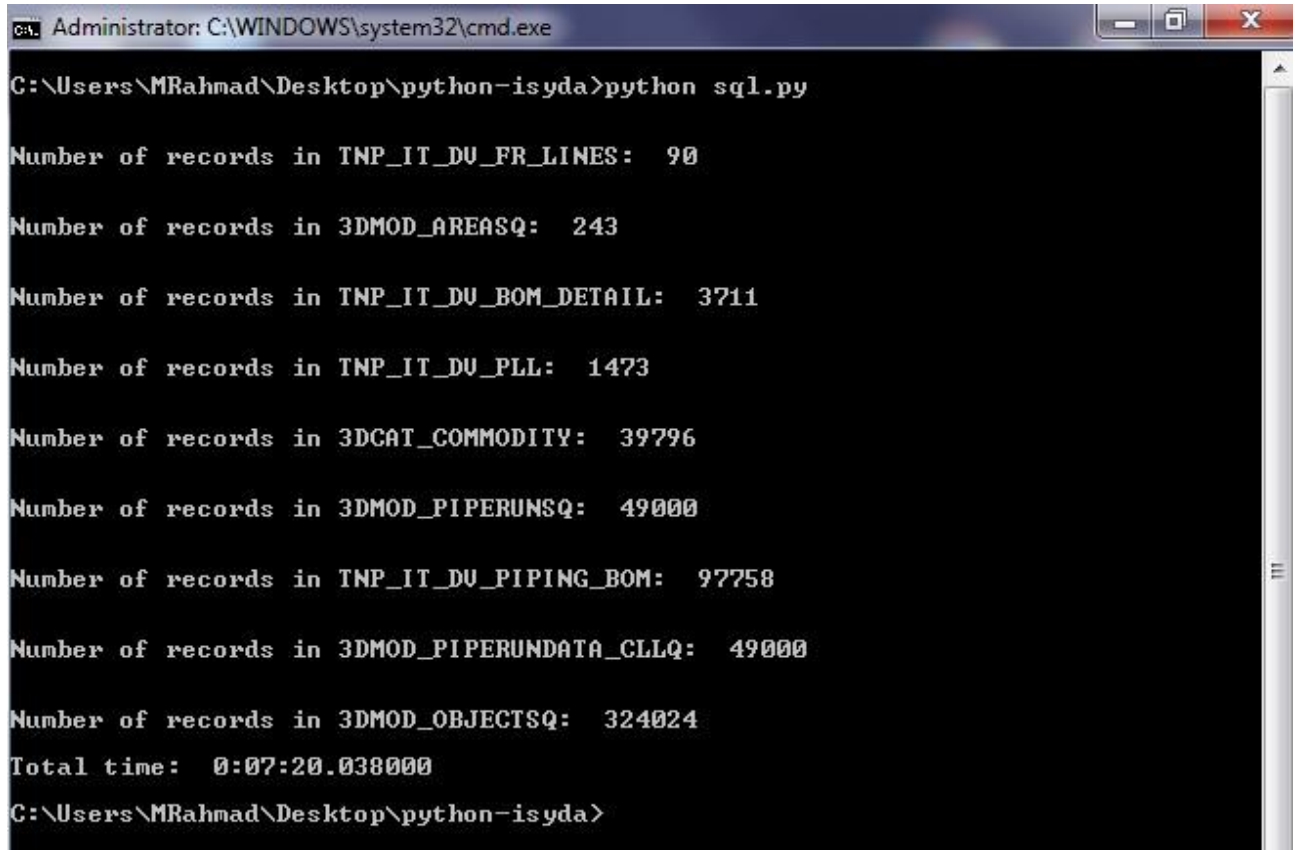
Database	Query Name	Execution Time(minutes:seconds)
S3D	3DCAT_COMMODITY	03:19
S3D	3DMOD_AREASQ	00:05
S3D	3DMOD_OBJECTSQ	07:36
S3D	3DMOD_PIPERUNDATA_CLLQ	02:25
S3D	3DMOD_PIPERUNSQ	01:40
SPMAT	TNP_IT_DV_BOM_DETAIL	00:29
SPMAT	TNP_IT_DV_FR_LINES	00:05
SPMAT	TNP_IT_DV_PIPING_BOM	00:51
SPMAT	TNP_IT_DV_PLL	00:36

Total time taken by running individual queries: 14 minutes: 10 seconds

Total time taken by parallel querying: 07 minutes: 20 seconds

Time saved: 06:50 (48.23 % decrease)

The results of running queries parallelly are given below:



```
Administrator: C:\WINDOWS\system32\cmd.exe
C:\Users\MRahmad\Desktop\python-isysda>python sql.py

Number of records in TNP_IT_DU_FR_LINES:  90

Number of records in 3DMOD_AREASQ:  243

Number of records in TNP_IT_DU_BOM_DETAIL:  3711

Number of records in TNP_IT_DU_PLL:  1473

Number of records in 3DCAT_COMMODITY:  39796

Number of records in 3DMOD_PIPERUNSQ:  49000

Number of records in TNP_IT_DU_PIPING_BOM:  97758

Number of records in 3DMOD_PIPERUNDATA_CLLQ:  49000

Number of records in 3DMOD_OBJECTSQ:  324024

Total time:  0:07:20.038000

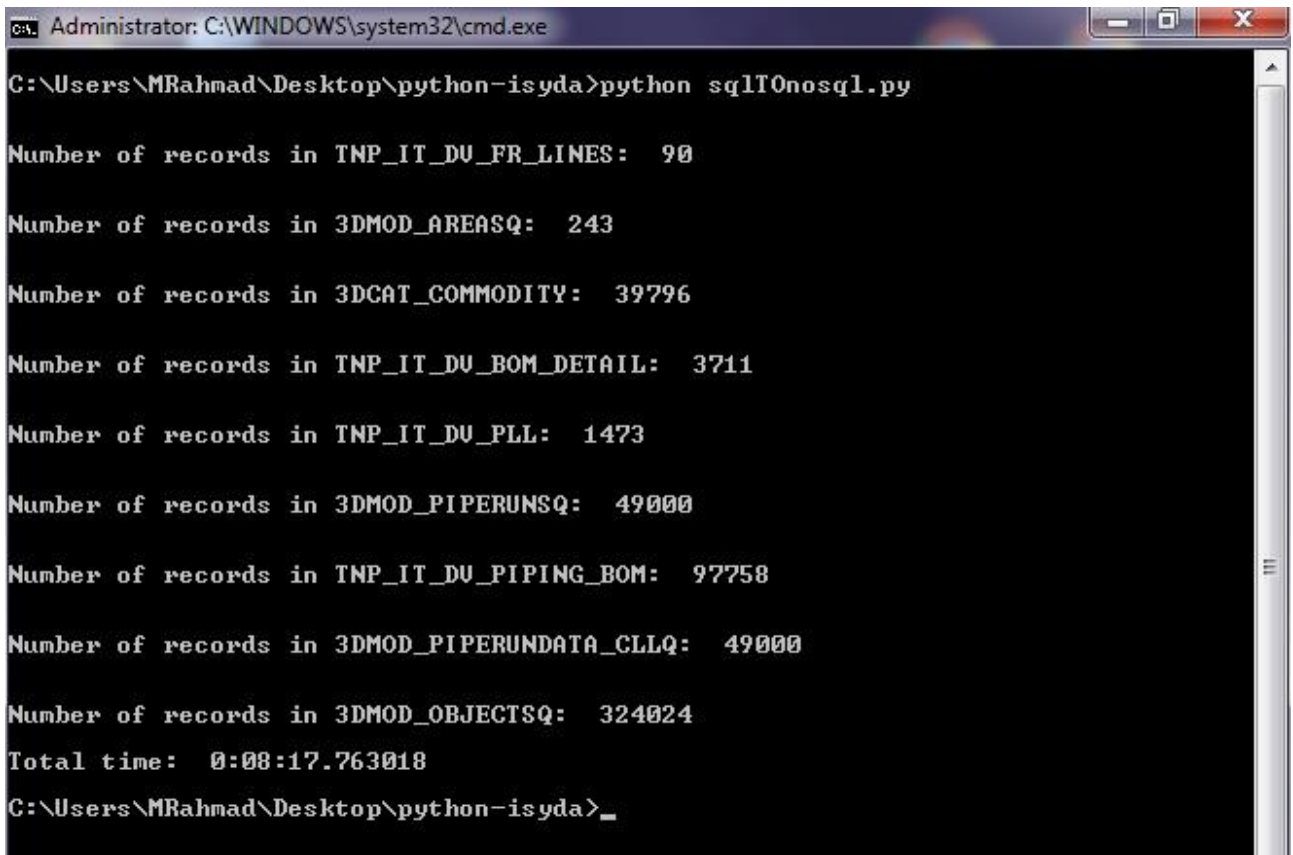
C:\Users\MRahmad\Desktop\python-isysda>
```

Figure 15: Parallel Processing Results

However, to run the queries parallelly and convert the results to MongoDB takes a little extra time, and the results are shown in the next section.

3.1.1.2 Output to MongoDB

To get the desired information, the results of these queries need to be stored for further filtration on them. This further filtration will be helpful in dealing with the 'Data Misalignment' problem. We used python to convert these results into MongoDB tables. With this step, it became possible to make comparisons between SQL and NoSQL on the same data with the same queries.



```
Administrator: C:\WINDOWS\system32\cmd.exe
C:\Users\MRahmad\Desktop\python-isysda>python sqlToNoSQL.py

Number of records in TNP_IT_DU_FR_LINES: 90
Number of records in 3DMOD_AREASQ: 243
Number of records in 3DCAT_COMMODITY: 39796
Number of records in TNP_IT_DU_BOM_DETAIL: 3711
Number of records in TNP_IT_DU_PLL: 1473
Number of records in 3DMOD_PIPERUNSQ: 49000
Number of records in TNP_IT_DU_PIPING_BOM: 97758
Number of records in 3DMOD_PIPERUNDATA_CLLQ: 49000
Number of records in 3DMOD_OBJECTSQ: 324024
Total time: 0:08:17.763018
C:\Users\MRahmad\Desktop\python-isysda>_
```

Figure 16: Parallel Processing Results – Conversion from SQL to NoSQL

3.1.2 NoSQL

Now it's time to write equivalent queries in MongoDB to compare the information retrieval timings from both database systems. For comparison purposes, I chose a single query. To compare the performance of SQL with NoSQL databases, I wrote the same query in MongoDB. As the simplest query included joins in it. And MongoDB doesn't support joins naturally. But in the latest versions, we can find some functions that can perform joins for us. So, I used '\$lookup' function for this purpose. This function can perform a left outer join for us. To convert the results of this left outer join to inner join, I further used '\$match' to filter records.

The results were not satisfactory for us. The simplest query which took less than 5 seconds to run on SQL, took more than 10 minutes to perform the same task in MongoDB. So, this solution didn't work for us. Somehow, I was able to perform automation of conversion from SQL to NoSQL. But we need to find a workable solution for NoSQL as well.

Results from SQL and NoSQL are shown below:

```
1 beginTime = datetime.datetime.now()
2
3 cursor.execute("""
4 Select
5 ran.xmin-----AS 'LowX',
6 ran.Ymin-----AS 'LowY',
7 ran.xMax-----AS 'HighX',
8 ran.YMax-----AS 'HighY',
9 cm.cm_DesignArea-----AS 'Unità',
10 cm.cm_SubArea-----AS 'Tipo Area',
11 cm.cm_ModuleNo-----AS 'Design Area'
12 from JDArea v
13 join JNamedItem VolumeName on VolumeName.Oid = v.oid
14 join UConstModule cm on v.Oid = cm.oid
15 join JRange ran on ran.oid = v.oid;
16 """)
17 row = cursor.fetchall()
18
19 finalTime = datetime.datetime.now() - beginTime
20 print(finalTime)
```

0:00:04.903771

Figure 17: Query processing result time in SQL

```
1 beginTime = datetime.datetime.now()
2
3
4 data = []
5 pipeline = [
6     { '$lookup':
7         {
8             'from': 'TENOC16_RDB_JNamedItem',
9             'localField': 'Oid',
10            'foreignField': 'Oid',
11            'as': 'NewOid1',
12        }
13    },
14    { '$match': { 'NewOid1': { '$exists': 'true', '$ne': [] } } },
15    { '$unwind': '$NewOid1' },
16
17    { '$lookup':
18        {
19            'from': 'TENOC16_RDB_UConstModule',
20            'localField': 'Oid',
21            'foreignField': 'Oid',
22            'as': 'NewOid2',
23        }
24    },
25    { '$match': { 'NewOid2': { '$exists': 'true', '$ne': [] } } },
26    { '$unwind': '$NewOid2' },
27
28    { '$lookup':
29        {
30            'from': 'TENOC16_RDB_JRange',
31            'localField': 'Oid',
32            'foreignField': 'Oid',
33            'as': 'NewOid3',
34        }
35    },
36    { '$match': { 'NewOid3': { '$exists': 'true', '$ne': [] } } },
37    { '$unwind': '$NewOid3' },
38
39    { '$project': { 'NewOid3.xmin' : 1, 'NewOid3.ymin' : 1, 'NewOid3.xmax' : 1, 'NewOid3.ymax' : 1,
40                    'NewOid2.cm_DesignArea' : 1, 'NewOid2.cm_SubArea' : 1, 'NewOid2.cm_ModuleNo' : 1 } }
41
42 ]
43
44 res = mydb.TENOC16_RDB_JDArea.aggregate(pipeline)
45 l = []
46 for i in res:
47     l.append(i)
48
49 finalTime = datetime.datetime.now() - beginTime
50 print(finalTime)
```

0:10:13.932409

Figure 18: Query processing result time in MongoDB

3.2 Optimizing Query

To assure minimum data retrieval time, we further decided to optimize the queries. For the sake of this document, we decided to optimize only 1 query. So, we chose '3DCAT_COMMODITY' for this purpose.

3.2.1 Scope

Analysis of the query '3DCAT_COMMODITY' (extraction of nominal diameters for each component of the catalog) in "TENOC16_MDB".

3.2.2 What is it about?

This query extracts the component of specific Pipeline and its corresponding diameters from Intergraph Smart 3D database directly. While analysis of this query we were able to decrease the response time from more than 3 minutes to 15 seconds only.

3.2.3 Contents of '3DCAT_COMMODITY'

```
SET NOCOUNT ON;
-----
declare @temp table (lvl int, sp uniqueidentifier, CF uniqueidentifier,
Npd1 float, NpdUnitType1 varchar(256), Npd2 float, NpdUnitType2
varchar(256));
insert @temp (lvl, sp,cf)
select 0, ps.oid, toCF.OidDestination
from JDPipeSpec ps
JOIN XPipeSpecContainsPartSpecs toCF on toCF.OidOrigin = ps.oid
-----
insert @temp (lvl, sp,cf, Npd1,NpdUnitType1)
select 1,source.sp, source.cf, possibleSize.NPD, possibleSize.NpdUnitType
from
@temp source
join XSpecDefinesAllowableNpd rel on rel.OidOrigin = source.sp
    join JDPipeNominalDiameters PossibleSize on PossibleSize.oid =
rel.OidDestination
where source.lvl = 0;
delete @temp where lvl =0;
delete @temp from
JDPipePartSpec pd where pd.Oid = cf and (npd1 <pd.FirstSizeFrom or
npd1>pd.FirstSizeTo or NpdUnitType1 <> pd.FirstSizeUnits)
--<> stands for 'not equal'
-----
```

```
update @temp set lvl = 2 from JDPipePartSpec pd where pd.oid = cf and lvl =
1 and pd.SecondSizeFrom is null;

insert @temp (lvl, sp,cf, Npd1,NpdUnitType1, Npd2,NpdUnitType2)
select 2, source.sp, source.cf, source.npd1, source.NpdUnitType1,
PossibleSize.NPD, possibleSize.NpdUnitType
from
@temp source
    join XSpecDefinesAllowableNpd rel on rel.OidOrigin = source.sp
    join JDPipeNominalDiameters PossibleSize on PossibleSize.oid =
rel.OidDestination
where source.lvl = 1;
delete @temp where lvl =1;
delete @temp from JDPipePartSpec pd
where pd.Oid = cf and pd.SecondSizeFrom is not null and (npd2
<pd.SecondSizeFrom or npd2>pd.SecondSizeTo or NpdUnitType2 <>
pd.SecondSizeUnits)
--<> stands for 'not equal'
-----
select
PS.SPECNAME AS 'CAT_SPEC',
cf.ShortCode AS 'CAT_SHORTCODE',
cf.OptionCode AS 'CAT_OPTIONCODE',
cf.OptionCode_LongValue AS 'CAT_OPTIONCODE_DESCRIPTION',
--cf.MultiSizeOption as 'CAT_MULTISIZE_OPTION',
explodeSizes.Npd1 AS 'CAT_SIZE1',
explodeSizes.Npd2 AS 'CAT_SIZE2',
--fss.FirstSizeSchedule_ShortValue as 'First_Size_Sch',
--sss.SecondSizeSchedule_ShortValue as 'Second_Size_Sch',
cf.CommodityCode as 'CAT_COMMODITY_CODE',
NULL as 'CAT_PART_TAG',
cfObj.DateLastModified 'CAT_DATE_LAST_MODIFIED'
-----
from JDPipeSpec ps
JOIN XPipeSpecContainsPartSpecs toCF on toCF.OidOrigin = ps.oid
JOIN JDPipePartSpec_cl cf on cf.oid = toCF.OidDestination
join @temp explodeSizes on explodeSizes.CF = cf.Oid
-----
JOIN JDObj cfObj ON cfObj.oid = cf.oid
JOIN JFirstSizeSchedule_CL fss on fss.oid = cf.oid
JOIN JSecondSizeSchedule_CL sss on sss.oid = cf.oid
order by ps.SpecName asc, cf.ShortCode asc, explodeSizes.Npd1,
explodeSizes.Npd2;
```

3.2.4 Flow and description

This query is composed of 9 steps, used for the preparation of a table variable @temp and a final step containing data extraction query. The final @temp table shows the diameters of each object. The instructions for inserting, modifying and deleting @temp are performed exclusively in memory. Each entry/update is assigned a level code that identifies the status of the record in the table.

level 0 – an association of Pipe Specification codes and CF specification without defining diameters

level 1 - definition of the first diameter for codes extracted at level 0

level 2 - I and II diameter consolidation, where required

The final level is, therefore, level 2, while levels 0 and 1 are intermediate levels useful for the composition of the table variable @temp.

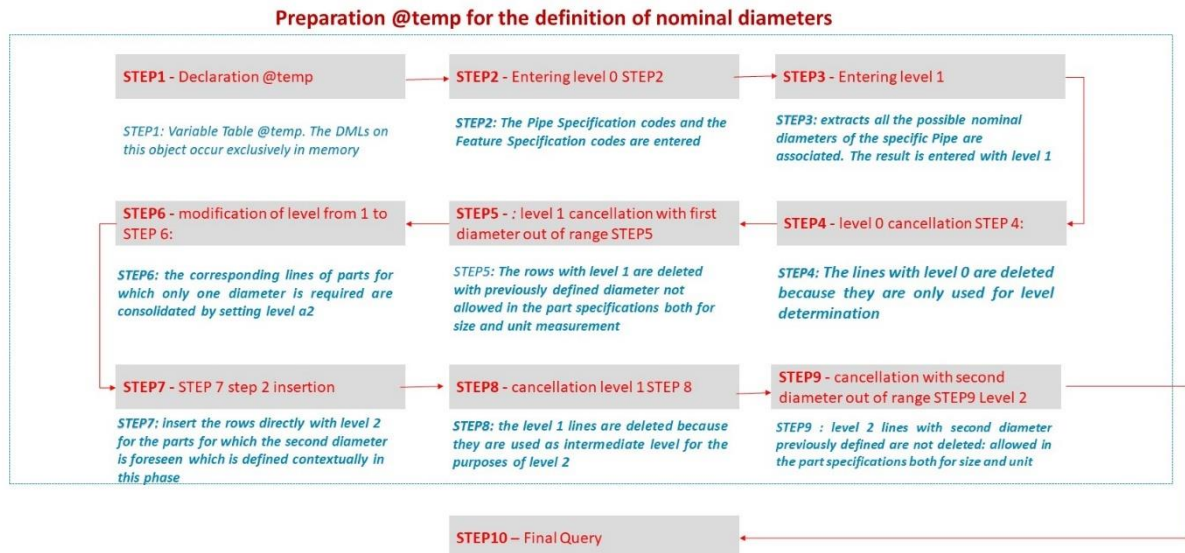


Figure 19: Preparation of @temp table

3.2.4.1 Step 1 – Declaration of @temp

The table variable @temp is declared. The manipulations of this object take place exclusively in memory. During this step, @temp is empty.

The following is the instruction:

```

declare @temp table
(lvl int
, sp uniqueiden_fier
, CF uniqueiden_fier
, Npd1 float
, NpdUnitType1 varchar (256)
, Npd2 float
, NpdUnitType2 varchar (256)) ;
  
```

Table 5: Fields information

lvl	Row level: 0 = first insert 1 = second insert 2 = third insert
sp	Id of PipeSpec
CF	Component id
NPD1	Diameter 1

NpdUnitType1	Diameter unit type 1
Npd2	Diameter 2
NpdUnitType2	Unit type of Diameter 2

3.2.4.2 Step 2 – level 0 entry

Insertion of Pipe Specification IDs and PartSpecifications with level 0 association.

A join is performed between the PIPE specifications (JDPipeSpec) and all the specifications of the Part (or features) contained in them.

```
insert @temp (lvl, sp, cf)
select 0 , ps.oid, toCF.OidDestination
from JDPipeSpec ps
JOIN XPipeSpecContainsPartSpecs toCF on toCF.OidOrigin = ps.oid
```

In this phase, @temp contains all records with level 0.

Below is the relationship between the objects of the first entry of the variable @temp. The first three fields are inserted. The first field is the integer numeric constant 0 (zero).

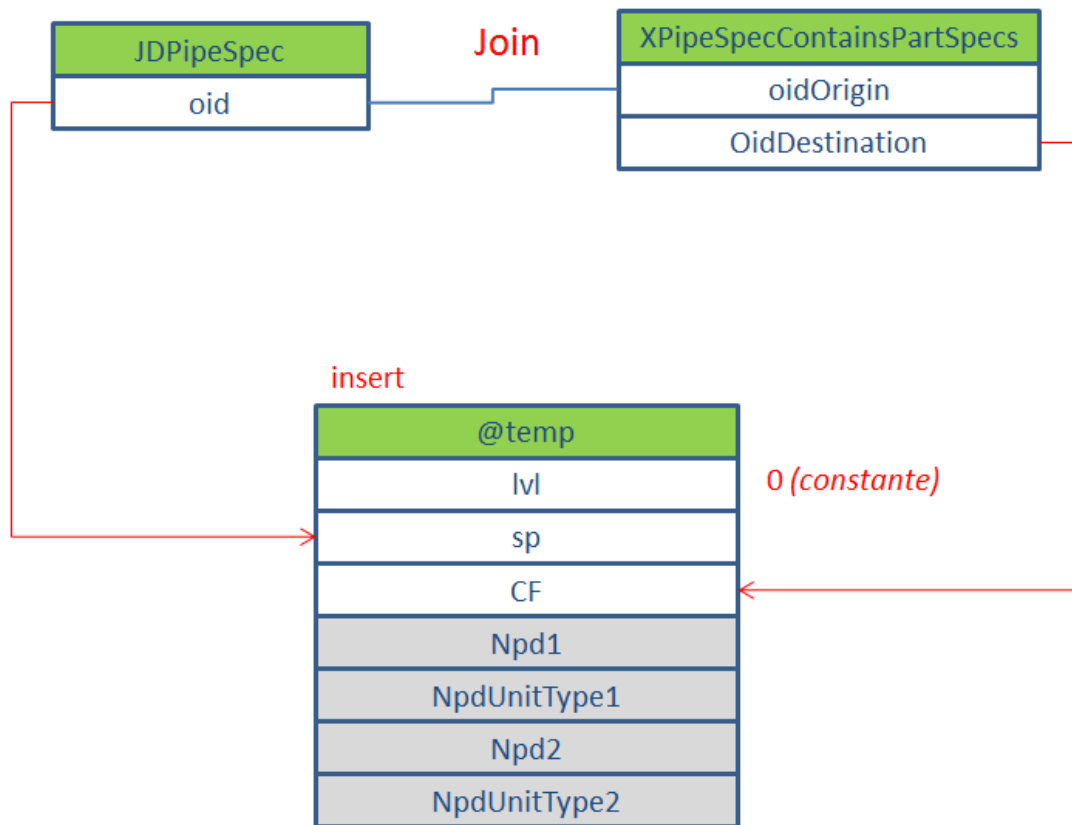


Figure 20: Relationship between objects

Below an example of this which is inserted with this step

0	0000ea72-0000-0000-0000-1d9ae257c490	0000ea73-0000-0000-0000-0c5bc1583354
0	0000ea72-0000-0000-0000-1d9ae257c490	0000ea73-0000-0000-0000-0c5bc1583f52
0	0000ea72-0000-0000-0000-1d9ae257c490	0000ea73-0000-0000-0000-0d5bc1581b58
0	0000ea72-0000-0000-0000-1d9ae257c490	0000ea73-0000-0000-0000-0d5bc1582756
0	0000ea72-0000-0000-0000-1d9ae257c490	0000ea73-0000-0000-0000-0f5bc158df61
0	0000ea72-0000-0000-0000-1d9ae257c490	0000ea73-0000-0000-2900-1e4bca596704
0	0000ea72-0000-0000-0000-1d9ae257c490	0000ea73-0000-0000-0000-22c508588d1f
0	0000ea72-0000-0000-0000-1d9ae257c490	0000ea73-0000-0000-0000-23c508587523
0	0000ea72-0000-0000-0000-1d9ae257c490	0000ea73-0000-0000-0000-23c508588121
0	0000ea72-0000-0000-0000-1d9ae257c490	0000ea73-0000-0000-0000-24c508585d27

Figure 21: Results

3.2.4.3 Step 3 – Entry level 1

For all parts, level 0 lines are associated with all the first nominal diameters permitted in Pipe specifications. The rows obtained are inserted in level 1.

```
insert @temp (lvl, sp, cf, Npd1, NpdUnitType1)
select 1 , source.sp, source.cf, possibleSize.NPD, possibleSize.NpdUnitType
from
@temp source
join XSpecDefinesAllowableNpd rel on rel.OidOrigin = source.sp
join JDPipeNominalDiameters PossibleSize on PossibleSize.oid =
rel.OidDes_na_on
where source.lvl = 0 ;
```

In this phase, @temp contains both the newly entered level 1 records and the level 0 records entered with step 2.

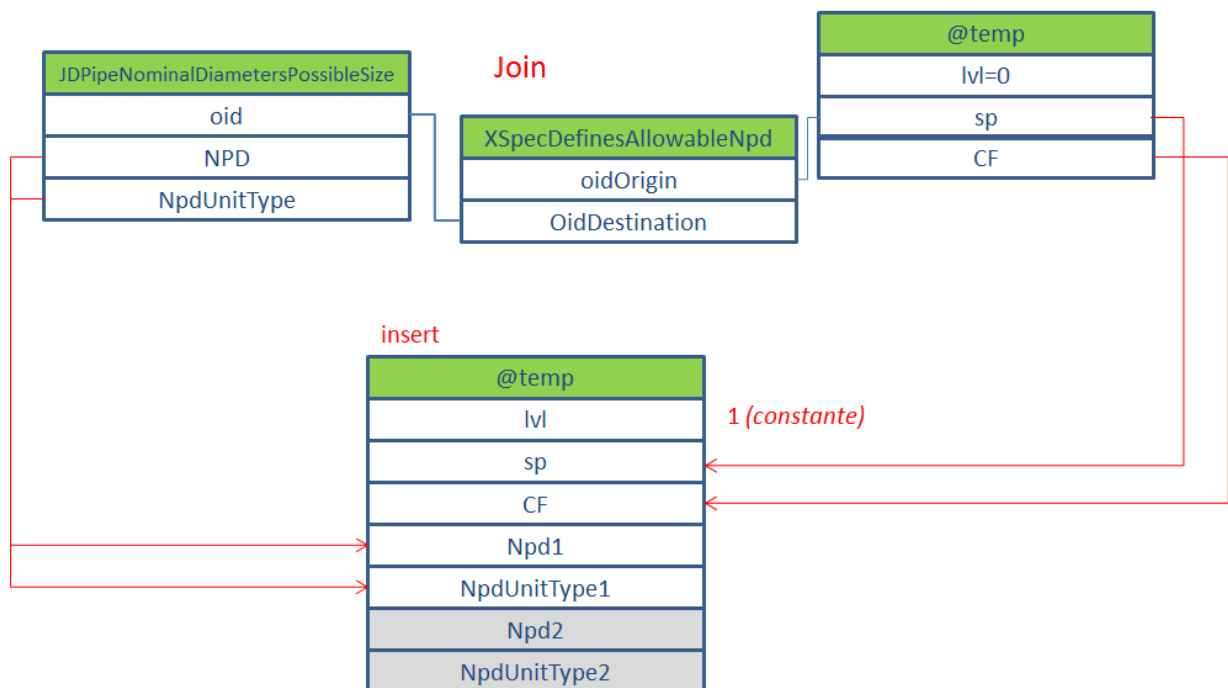


Figure 22: Relationship between objects

Below is an extract of @ temp after this step (in red the added fields):

[No name 1]	sp	cf	NPD	NpdUnitType
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-903002594950	8	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-903002594950	6	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-90300259554e	10	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-90300259554e	14	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-90300259554e	3	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-90300259554e	8	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-90300259554e	6	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-913002593154	10	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-913002593154	14	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-913002593154	3	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-913002593154	8	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-913002593154	6	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-913002593d52	10	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-913002593d52	14	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-913002593d52	3	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-913002593d52	8	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-913002593d52	6	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-923002592556	10	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-923002592556	14	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-923002592556	3	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-923002592556	8	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-923002592556	6	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-93300259015c	10	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-93300259015c	14	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-93300259015c	3	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-93300259015c	8	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-93300259015c	6	in
1	0000ea72-0000-0000-0000-59fbec57fbda	0000ea73-0000-0000-0000-94300259dd61	10	in

Figure 23: Results

3.2.4.4 Step 4 - level 0 cancellation

Level 0 was necessary for the determination of the treated codes in level 1 i.e. to what to associate the first diameter. At this point level 0 is no longer needed and is deleted.

```
delete @temp where lvl = 0 ;
```

In this phase, @temp contains only level 1 records or all parts of the Pipe Specs with the first diameter set.

DELETE

@temp
lvl=0
sp
CF
Npd1
NpdUnitType1
Npd2
NpdUnitType2

Figure 24: Objects

3.2.4.5 Step 5 - cancellation level 1 with a first invalid diameter

The first diameter has been determined for all the Parts, but it is not said that the established diameter (for the PipeSpecs) is valid for the Part specification (PipePartSpecs).

The invalid ones must be deleted.

The step deletes the lines of @temp with diameter included in the range present in JDPipePartSpec or Unit that is not allowed.

```
delete @temp from JDPipePartSpec pd
where pd.Oid = cf and (npd1 < pd.FirstSizeFrom or npd1 > pd.FirstSizeTo or
NpdUnitType1 <> pd.FirstSizeUnits)
--<> stands for 'not equal'
```

In this phase, @temp contains level 1 with valid nominal diameters.

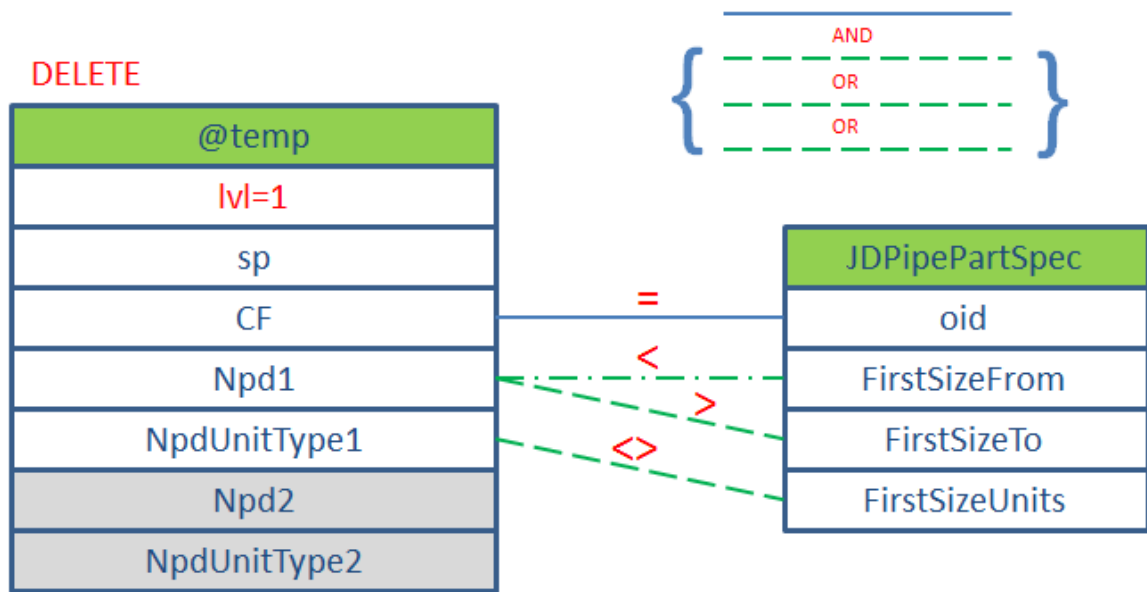


Figure 25: Relationship Between Objects

3.2.4.6 Step 6 - modification level 1 to 2

Here we consolidate the parts for which a second diameter is not provided, setting precisely the level 2 which is the level of consolidation with a specific update.

Parts for which a second diameter is provided are left with level 1.

The JDPipePartSpec view establishes, whether one or two diameters are provided for the treated component.

```
update @temp
set lvl = 2
from JDPipePartSpec pd
where pd.oid = cf
and lvl = 1
and pd.SecondSizeFrom is null;
```

In this phase, @temp contains level 2 for all parts of diameter and level 1 for parts for which the second diameter must be recovered.

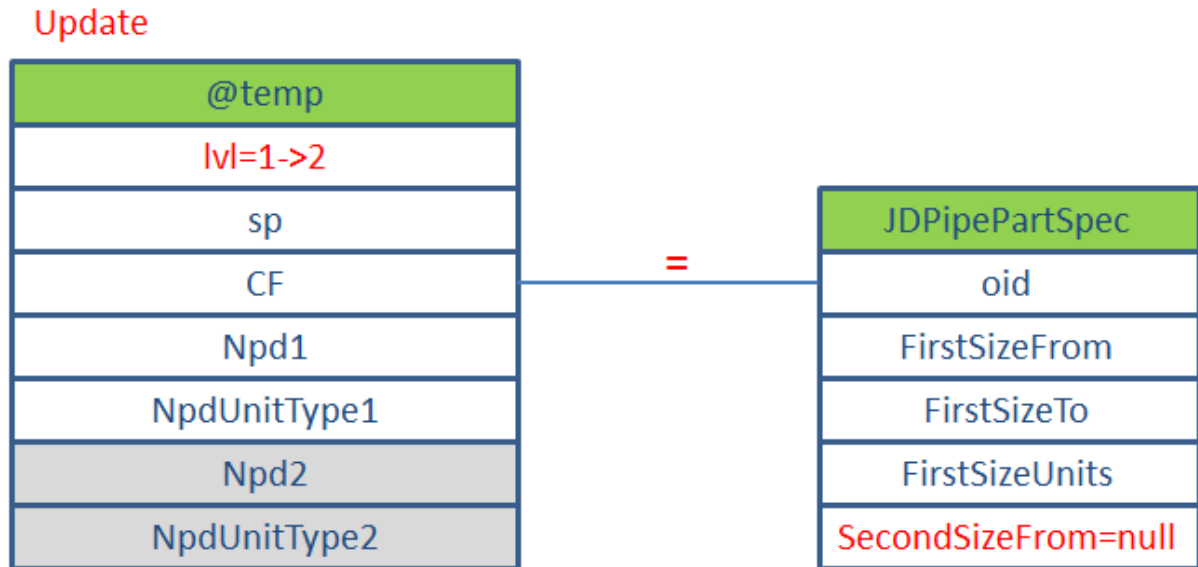


Figure 26: Relationship Between Objects

3.2.4.7 Step 7 - insertion level 2

Here the second diameter is determined for the level 1 parts left in the previous step. For each row for which the first diameter has been established and the second diameter is provided, many rows are inserted with the possible combinations of the second diameter. Level 2 is inserted starting from level 1 and integrating into level 2 with the second diameter.

```
insert @temp (lvl, sp, cf, Npd1, NpdUnitType1, Npd2, NpdUnitType2)
select 2 , source.sp, source.cf, source.npd1, source.NpdUnitType1,
PossibleSize.NPD, possibleSize.NpdUnitType e
from @temp source
join XSpecDefinesAllowableNpd rel on rel.OidOrigin = source.sp
join JDPipeNominalDiameters PossibleSize on PossibleSize.oid =
rel.OidDes_na_on
where source.lvl = 1 ;
```

In this phase, @temp contains the level 2 for all the parts with the first consolidated diameter and the parts for which the second diameter is required, which must be consolidated, or controlled that is in the range specified in the specifications of the parts. It also contains the level1 used for insertion in this step.

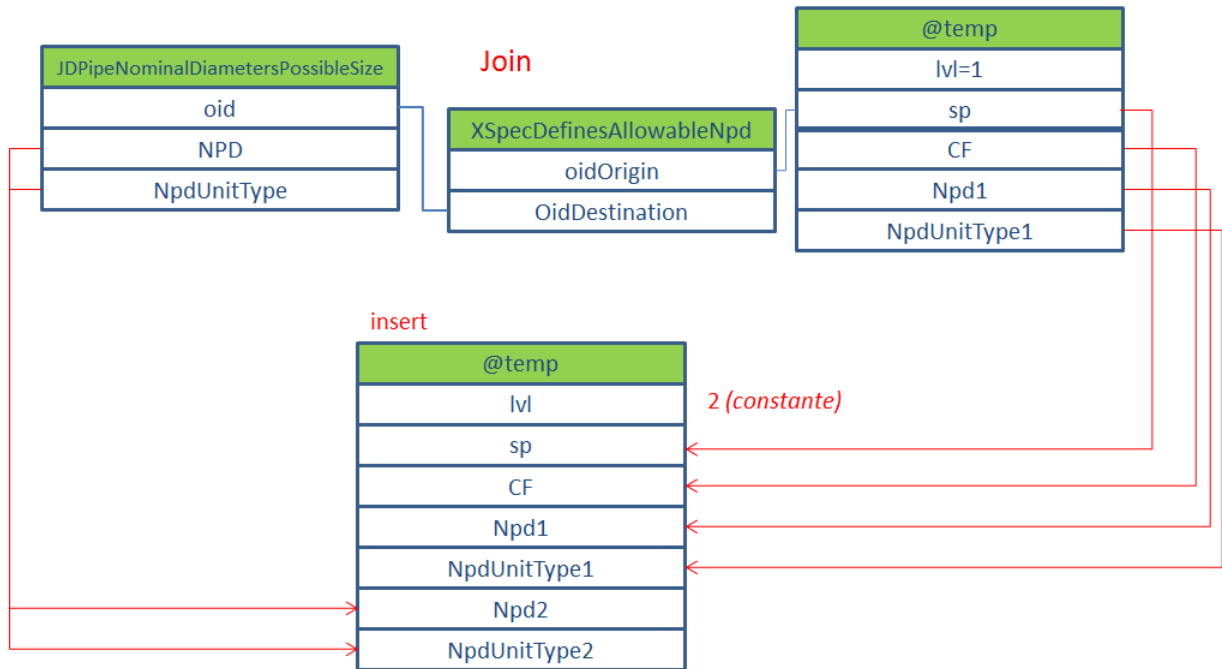


Figure 27: Relationship Between Objects

Here is an excerpt of the result

lvl	sp	CF	Npd1	NpdUnitType1	Npd2	NpdUnitType2
2	0000ea72-0000-0000-0000-80d9e057040a	0000ea73-0000-0000-0000-6b270f588ff4	18 in		3 in	
2	0000ea72-0000-0000-0000-80d9e057040a	0000ea73-0000-0000-0000-6c270f5883f6	18 in		18 in	
2	0000ea72-0000-0000-0000-80d9e057040a	0000ea73-0000-0000-0000-6c270f5883f6	18 in		14 in	
2	0000ea72-0000-0000-0000-80d9e057040a	0000ea73-0000-0000-0000-6c270f5883f6	18 in		4 in	
2	0000ea72-0000-0000-0000-1d125c58a1bf	0000ea73-0000-0000-0000-7a135c58bd67	2 in		{null}	{null}
2	0000ea72-0000-0000-0000-1d125c58a1bf	0000ea73-0000-0000-0000-7a135c58bd67	6 in		{null}	{null}
2	0000ea72-0000-0000-0000-80d9e057040a	0000ea73-0000-0000-0000-6c270f5883f6	18 in		3 in	
2	0000ea72-0000-0000-0000-80d9e057040a	0000ea73-0000-0000-0000-6d270f5877f8	18 in		18 in	
2	0000ea72-0000-0000-0000-80d9e057040a	0000ea73-0000-0000-0000-6d270f5877f8	18 in		14 in	
2	0000ea72-0000-0000-0000-1d125c58a1bf	0000ea73-0000-0000-0000-7a135c58d563	2 in		{null}	{null}
2	0000ea72-0000-0000-0000-80d9e057040a	0000ea73-0000-0000-0000-6d270f5877f8	18 in		4 in	
2	0000ea72-0000-0000-0000-80d9e057040a	0000ea73-0000-0000-0000-6d270f5877f8	18 in		3 in	
2	0000ea72-0000-0000-0000-80d9e057040a	0000ea73-0000-0000-0000-6f270f5853fa	4 in		18 in	

Figure 28: Results

3.2.4.8 Step 8 - cancellation level 1

The step cancels the level 1 records that were used in Step7 for setting the second diameter.

```
delete @temp where lvl = 1 ;
```

In this phase, @temp contains sol or level 2 record.

DELETE

@temp
lvl=1
sp
CF
Npd1
NpdUnitType1
Npd2
NpdUnitType2

Figure 29: Objects

3.2.4.9 Step 9 - cancellation level 2 with the second diameter not valid

The second diameter has been determined for all the parts for which it is intended, but it is not certain that the second diameter established (for the PipeSpecs) is valid for the specification of the part (PipePartSpecs).

The invalid ones must be deleted.

This step deletes the lines of @temp with second diameter not included in the range present in JDPipePartSpec or whose Unit measurement is not allowed

```
delete @temp
from
JDPipePartSpec pd
where pd.Oid = cf
and pd.SecondSizeFrom is not null
and (npd2 < pd.SecondSizeFrom or npd2 > pd.SecondSizeTo or NpdUnitType2 <>
pd.SecondSizeUnits)
--<> stands for 'not equal'
```

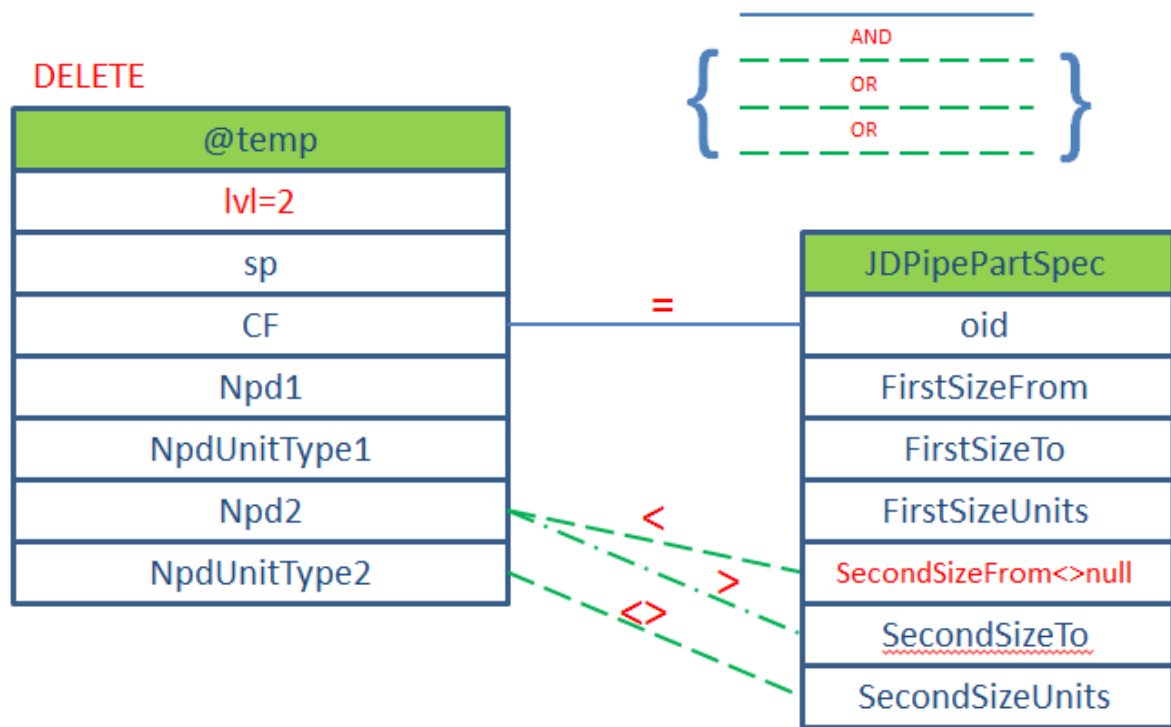


Figure 30: Relationship Between Objects

3.2.4.10 Step 10 - Final query

Extraction of all parts of the catalog and corresponding diameters

```
select
PS.SPECNAME AS 'CAT_SPEC' ,
cf.ShortCode AS 'CAT_SHORTCODE' ,
cf.Op_onCode AS 'CAT_OPTIONCODE' ,
--cf.Mul_SizeOp_on as 'CAT_MULTISIZE_OPTION',
explodeSizes.Npd1 AS 'CAT_SIZE1' ,
explodeSizes.Npd2 AS 'CAT_SIZE2' ,
--fss.FirstSizeSchedule_ShortValue as 'First_Size_Sch',
--sss.SecondSizeSchedule_ShortValue as 'Second_Size_Sch',
cf.CommodityCode as 'CAT_COMMODITY_CODE' ,
NULL as 'CAT_PART_TAG' ,
cfObj.DateLastModified 'CAT_DATE_LAST_MODIFIED'
-----
from JDPipeSpec ps
JOIN XPipeSpecContainsPartSpecs toCF on toCF.OidOrigin = ps.oid
JOIN JDPipePartSpec_cl cf on cf.oid = toCF.OidDes_na_on
join @temp explodeSizes on explodeSizes.CF = cf.Oid
-----
JOIN JDObject cfObj ON cfObj.oid = cf.oid
JOIN JFirstSizeSchedule_CL fss on fss.oid = cf.oid
JOIN JSecondSizeSchedule_CL sss on sss.oid = cf.oid
```


3.2.5 Stream and query optimization

By executing the query described above, the result was the following:

```
39
40 select
41 ----->
42 PS.SPECNAME AS 'CAT_SPEC',
43 cf.ShortCode AS 'CAT_SHORTCODE',
44 cf.OptionCode AS 'CAT_OPTIONCODE',
45 cf.OptionCode_LongValue AS 'CAT_OPTIONCODE_DESCRIPTION',
46 --cf.MultiSizeOption as 'CAT_MULTISIZE_OPTION',
47 explodeSizes.Npd1 AS 'CAT_SIZE1',
48 explodeSizes.Npd2 AS 'CAT_SIZE2',
49 --fss.FirstSizeSchedule_ShortValue as 'First_Size_Sch',
50 --sss.SecondSizeSchedule_ShortValue as 'Second_Size_Sch',
51 cf.CommodityCode as 'CAT_COMMODITY_CODE',
52 NULL as 'CAT_PART_TAG',
53 cfObj.DateLastModified 'CAT_DATE_LAST_MODIFIED'
54 ----->
55 from JDPipeSpec ps
56
57 JOIN XPipeSpecContainsPartSpecs toCF on toCF.OidOrigin = ps.oid
58 JOIN JDPipePartSpec_cl cf on cf.oid = toCF.OidDestination
59 join @temp explodeSizes on explodeSizes.CF = cf.Oid
60 ----->
61 JOIN JDObj cfObj ON cfObj.oid = cf.oid
62 JOIN JFirstSizeSchedule_CL fss on fss.oid = cf.oid
63 JOIN JSecondSizeSchedule_CL sss on sss.oid = cf.oid
64 order by ps.SpecName asc, cf.ShortCode asc, explodeSizes.Npd1, explodeSizes.Npd2
```

CAT_SPEC	CAT_SHORTCODE	CAT_OPTIONCODE	CAT_OPTIONCODE_DESCRIPTION	CAT_SIZE1	CAT_SIZE2	CAT_COMMODITY_CODE	CAT_PART_TAG	CAT_DATE_LAST_MODIFIED
A2	<45 Degree Direction Change	1	Default	2	{null}	DBE420C5C830010000	{null}	12/20/2016 1:22:34 PM
A2	<45 Degree Direction Change	1	Default	3	{null}	DBE420C5C830010000	{null}	12/20/2016 1:22:34 PM
A2	<45 Degree Direction Change	1	Default	4	{null}	DBE420C5C830010000	{null}	12/20/2016 1:22:34 PM
A2	<45 Degree Direction Change	1	Default	6	{null}	DBE420C5C830010000	{null}	12/20/2016 1:22:34 PM
A2	<45 Degree Direction Change	1	Default	8	{null}	DBE420C5C830010000	{null}	9/21/2016 2:33:47 PM
A2	<45 Degree Direction Change	1	Default	10	{null}	DBE420C5C830010000	{null}	9/21/2016 2:33:47 PM
A2	<45 Degree Direction Change	1	Default	12	{null}	DBE420C5C830010000	{null}	9/21/2016 2:33:47 PM
A2	<45 Degree Direction Change	1	Default	14	{null}	DBE420C5C830010000	{null}	4/27/2017 5:25:57 PM
A2	<45 Degree Direction Change	1	Default	16	{null}	DBE420C5C830010000	{null}	4/27/2017 5:25:58 PM

Duration: 0:03:19.640 Executed: 01/25/2019 13:49:32 Record 1 of 39796

Figure 33: Results before optimization

After modification, the results are:

```

55 PS.SPECNAME AS 'CAT_SPEC',
56 cf.ShortCode AS 'CAT_SHORTCODE',
57 cf.OptionCode AS 'CAT_OPTIONCODE',
58 cf.OptionCode_LongValue AS 'CAT_OPTIONCODE_DESCRIPTION',
59 --cf.MultisizeOption as 'CAT_MULTISIZE_OPTION',
60 explodeSizes.Npd1 AS 'CAT_SIZE1',
61 explodeSizes.Npd2 AS 'CAT_SIZE2',
62 --fss.FirstSizeSchedule_ShortValue as 'First_Size_Sch',
63 --sss.SecondSizeSchedule_ShortValue as 'Second_Size_Sch',
64 cf.CommodityCode as 'CAT_COMMODITY_CODE',
65 NULL as 'CAT_PART_TAG',
66 cfObj.DateLastModified 'CAT_DATE_LAST_MODIFIED'
67 ----->
68 from JDPipeSpec ps
69
70 JOIN XPipeSpecContainsPartSpecs toCF on toCF.OidOrigin = ps.oid
71 JOIN JDPipePartSpec_cl cf on cf.oid = toCF.OidDestination
72 join @temp explodeSizes on explodeSizes.CF = cf.Oid
73 -----
74 JOIN JDBObject cfObj ON cfObj.oid = cf.oid
75 --JOIN JFirstSizeSchedule_CL fss on fss.oid = cf.oid
76 --JOIN JSecondSizeSchedule_CL sss on sss.oid = cf.oid
77 order by ps.SpecName asc, cf.ShortCode asc, explodeSizes.Npd1, explodeSizes.Npd2

```

Results

Result Sets Messages Execution Plan

Set 1

CAT_SPEC	CAT_SHORTCODE	CAT_OPTIONCODE	CAT_OPTIONCODE_DESCRIPTION	CAT_SIZE1	CAT_SIZE2	CAT_COMMODITY_CODE	CAT_PART_TAG	CAT_DATE_LAST_MODIFIED
A2	<45 Degree Direction Change	1	Default	2	(null)	D8E420C5C830010000	(null)	12/20/2016 1:22:34 PM
A2	<45 Degree Direction Change	1	Default	3	(null)	D8E420C5C830010000	(null)	12/20/2016 1:22:34 PM
A2	<45 Degree Direction Change	1	Default	4	(null)	D8E420C5C830010000	(null)	12/20/2016 1:22:34 PM
A2	<45 Degree Direction Change	1	Default	6	(null)	D8E420C5C830010000	(null)	12/20/2016 1:22:34 PM
A2	<45 Degree Direction Change	1	Default	8	(null)	D8E420C5C830010000	(null)	9/21/2016 2:33:47 PM
A2	<45 Degree Direction Change	1	Default	10	(null)	D8E420C5C830010000	(null)	9/21/2016 2:33:47 PM
A2	<45 Degree Direction Change	1	Default	12	(null)	D8E420C5C830010000	(null)	9/21/2016 2:33:47 PM
A2	<45 Degree Direction Change	1	Default	14	(null)	D8E420C5C830010000	(null)	4/27/2017 5:25:58 PM
A2	<45 Degree Direction Change	1	Default	16	(null)	D8E420C5C830010000	(null)	4/27/2017 5:25:58 PM
A2	<45 Degree Direction Change	1	Default	18	(null)	D8E420C5C830010000	(null)	4/27/2017 5:25:58 PM
A2	<45 Degree Direction Change	1	Default	20	(null)	D8E420C5C830010000	(null)	4/27/2017 5:25:58 PM
A2	<45 Degree Direction Change	1	Default	24	(null)	D8E420C5C830010000	(null)	4/27/2017 5:25:58 PM
A2	<45 Degree Direction Change	1	Default	26	(null)	D8E420C5C830030000	(null)	9/21/2016 2:33:47 PM
A2	<45 Degree Direction Change	1	Default	28	(null)	D8E420C5C830030000	(null)	9/21/2016 2:33:47 PM
A2	<45 Degree Direction Change	1	Default	30	(null)	D8E420C5C830030000	(null)	9/21/2016 2:33:47 PM

Duration: 0:00:15.364 Executed: 01/25/2019 13:57:23 Record 1 of 39796

Figure 34: Results after optimization

4 Data Misalignment

The other issue is with misalignment and incompleteness of data. There is no automated connection of data flow between main applications. There are only manual ways to check if data in different sources are aligned or not. This problem reduces the pace of work in the work field. Different teams must wait for certain checks to be performed by some other teams before they can start working.

4.1 Certified and Uncertified Pipelines

The certified pipelines are the pipelines which:

- Are aligned between the 3D model and CLL by using:
 1. The tuple Project-Unit-Area-Line
 2. The tuple Project-Unit-Area-Line-Piping Class
 3. The tuple Project-Unit-Area-Line-Piping Class-Diameter
- Have consistent data used in the isometric layout between CLL and 3D model
- Have lower 'update date' of components in the 3D model than in Catalogue DB.

Basically, this is about alignment between data related to pipelines stored at different sources. If the data is aligned, the pipelines are certified, otherwise not. Current practices require a lot of effort and time to accomplish this task. Which is not efficient. Hence there is a strong need to make this process efficient and error-free. So, I automate this task by querying the data directly from the S3D database using a Python interface and comparing it with the other data source, that is Construction Line List (CLL). This provides us the list of certified and Uncertified lines at any specific time during the project execution.

4.2 SQL vs NoSQL

The identification of certified and uncertified lines was performed using two different approaches, SQL and NoSQL.

4.2.1 SQL

The results obtained from the queries we used in 'Data Availability' phase was stored in SQL. There was a simple query used further to obtain the list of certified lines. In the case of SQL,

this query took a few seconds to provide the results. The query that gives certified lines is given below:

```
select 3D_SIZE, 3D_WBS_UNIT, 3D_WBS_LINE, 3D_SPEC
from 3DMOD_PIPERUNS JOIN REF_CLL
where 3DMOD_PIPERUNS.3D_SIZE = REF_CLL.SIZE
and 3DMOD_PIPERUNS.3D_WBS_UNIT = REF_CLL.pll Units
and 3DMOD_PIPERUNS.3D_WBS_LINE = REF_CLL.LINE
and 3DMOD_PIPERUNS.3D_SPEC = REF_CLL.Spec Header
```

```
beginTime = datetime.datetime.now()

cnxn3 = pyodbc.connect('DRIVER='+driver+';SERVER='+server+';DATABASE='+database+';UID='+user+';PWD='+password';Trusted_connection
cursor3 = cnxn3.cursor()

quer3 = """
select PIPERUNS.SIZE, WBS_UNIT, WBS_LINE, SPEC
from PIPERUNS JOIN CLL ON PIPERUNS.SIZE = CLL.Size
and PIPERUNS.WBS_UNIT = CLL.pllUnits
and PIPERUNS.WBS_LINE = CLL.Line
and PIPERUNS.SPEC = CLL.SpecHeader
"""

cursor3.execute(quer3)
#cursor.execute("*****")

row3 = cursor3.fetchall()

cursor3.close()
del cursor3
cnxn3.close()

print(len(row3))
if (len(row3) > 0):
    print(row3[0])

finalTime = datetime.datetime.now() - beginTime
print("\nTotal time: ", finalTime)

1849
(0.5, '02', '02-TW601', 'UXC14')

Total time: 0:00:00.064000
```

Figure 35: SQL query for obtaining certified lines

4.2.2 NoSQL

While working with 'Data Availability' problem. Results obtained from both SQL and NoSQL were stored in MongoDB for further filtration. As the query in SQL has where clause on multiple fields. Since MongoDB's join feature is immature yet and it doesn't support multiple 'localField' and multiple 'foreignField'. Just for the sake of comparison, I ran this query with a single 'localField' and 'foreignField' (i.e. single field). The results are given below:

```
1 beginTime = datetime.datetime.now()
2
3 data = []
4 pipeline = [
5     { '$lookup':
6         {
7             'from' : 'REF_CLL',
8             'localField' : '3D_SIZE',
9             'foreignField' : 'SIZE',
10            'as' : 'New'
11        }
12    },
13    { '$match' : { 'New' : { '$exists' : 'true', '$ne' : [] } } },
14    { '$unwind' : '$New' }
15 ]
16 res = mydb.MOD_PIPERUNS.aggregate(pipeline)
17 l = []
18 for i in res:
19     l.append(i)
20
21 finalTime = datetime.datetime.now() - beginTime
22 print(finalTime)
```

0:05:19.878985

Figure 36: NoSQL Query for obtaining certified lines

4.3 Restructuring NoSQL Database

In the previous results, we have observed that using a join in MongoDB is quite expensive. To tackle this problem, we decided to restructure the way the data is stored in MongoDB collections. The idea behind this restructuring was to create such tables, that doesn't require using joins in them. Tables were generated depending specifically on the queries used to generate reports.

With this solution, we draw this conclusion that it is possible to use MongoDB at least at the final layer of our system. The layer used to present the final information. However, the original databases connected to our main applications are still using SQL. The purpose behind replacing SQL with NoSQL to the maximum is its cheap costs and versatility to manage data from different sources efficiently.

The code to retrieve data from the table is given below:

```
myclient = MongoClient()
mydb = myclient["name"]
dic = dict()
listcollec = []
tabData = []

for k in row:
    tabData.append(k)

cols = ['size', 'WBS_UNIT', 'WBS_LINE', 'SPEC']

name = "CERTIFIED"
mycollec = mydb[name]
listcollec = []
```

```
for l in range (len(tabData)):  
    dic.clear()  
    for m in range (len(cols)):  
        colName = cols[m]  
        dic[str(colName)] = tabData[l][m]  
    listcollec.append(dic.copy())  
mycollec.insert_many(listcollec)
```

5 Appendix

Cable List: Entire list of all the cables that are going to be used in the project.

Equipment Foundation: Is the structure engineered by civil engineers. Keeping in mind the various aspects like the weight of the equipment to be placed, vibration, etc.

Fluid list: is the complete list of fluids with their full characteristics, that are going to be involved in the project. It contains details like temperature, pressure, piping class etc.

Heat and mass balance (HMB): are a document produced by process design engineers while designing a process plant. Sometimes heat and mass balance is not a separate document but appears alongside the Process Flow Diagram (PFD). A heat and mass balance sheet represent every process stream on the corresponding PFD in terms of the process conditions.

Instrument Data Sheet: Instrument Data Sheet is a document containing specification and information of an instrument device. It specifies general information of instrument such as tag number identification, service description, location (line number/equipment number), P&ID number or drawing number reference, process data (if applicable), calibrated range (if applicable), material, performance details (such as accuracy, linearity – if applicable), hazardous certification (for electrical device), accessories required, etc. The details of information in data sheet may differ among each type of an instrument such as a transmitter, switch, gauge, control valves.

Instrument List: Entire list of instruments going to be used in the project as provided by P&ID.

Isometric Drawings: It is a standard practice for engineering companies to create isometric drawings of piping systems to present all details in –depth. An isometric drawing is nothing but a detailed orthographic drawing that represents the details of the 3D structure of the piping system in a 2D format. With the help of sophisticated computer-aided design (CAD) tools, piping engineers and designers can produce isometric drawings from 3D models with ease.

Line list: A line list is a document (deliverable generated by Smart P&ID) created to communicate between the process and mechanical engineering teams, that use it as a reference when designing piping in a plant or process unit.

Line list is produced by Process Engineers. It specifies the process parameters such as the design and operating pressures & temperatures, flowing medium, piping code, Line ID, reference P&ID numbers, etc. This process line list is “extended” by Mechanical Engineers in a Mechanical Line List which includes fixed and certified mechanical conditions, with the addition of some other parameters (insulation materials and thickness).

Mechanical data Sheet: A Mechanical Data Sheet is developed by the mechanical engineer after a process engineer initiated and provided the equipment process data sheets with other disciplines’ information. The mechanical engineer uses the process data sheet and further develops mechanical details including equipment sizes and associated detail information for each equipment item, and detailed sized equipment list for other engineering disciplines to complete their detailed engineering and design (e.g. piping engineers used the equipment list to create the plot plan and electrical engineers used it to design the electrical systems, etc.) The Mechanical Data Sheet is a basis of equipment purchasing and fabrication by vendor or manufacturer.

P&ID: A piping and instrumentation diagram (P&ID) is a detailed diagram in the process industry which shows the piping and vessels in the process flow, together with the instrumentation and control devices. Superordinate to the piping and instrumentation diagram is the process flow diagram (PFD) which indicates the more general flow of plant processes and equipment and the relationship between major equipment of a plant facility.

PFD: A process flow diagram (PFD) is a diagram commonly used in chemical and process engineering to indicate the general flow of plant processes and equipment. The PFD displays the relationship between major equipment of a plant facility and does not show minor details such as piping details and designations. Another commonly used term for a PFD is a flowsheet.

PipeLine: Series of straight pipe welded together for a long distance. It can be underground, aboveground and underwater such as a subsea pipeline.

Pipe Rack: is a structure, built from steel or concrete. It is used for holding pipes, wires and some equipment that require to be a certain height.

Piping Routing: Are the routes taken by the pipelines.

Piping Class: Piping class is a document that specifies the type of the components such as a type of pipe, schedule, material, flange ratings, branch types, valve types and valve trim material, gasket, and all the other components specific requirements to be used for different fluids under different operating conditions in a plant. Pipe class is developed considering Operating Pressure, temperature, and corrosive environment. Different material specifications are segregated in separate “Piping Class”. Pipe class is part of the Piping specification.

Piping Workflow: A piping workflow is built by connecting nodes to each other. Every interface can be treated as a pipeline node having defined inputs and outputs. Creating a workflow then is a matter of connecting appropriate outputs to inputs. Currently, workflows are limited to being directional and cannot have any loops, thereby creating an ordering to the data flow.

Plot Plan: Most of the Process Plants require a plot plan. Equipment spacing requirements will vary with the type of plant and location. Plot plans are considered key documents to projects and are normally initiated in the pre-contract, conceptual and development stages of a proposal. After the contract is awarded for engineering, plot plans are developed at a rather rapid pace with very limited information. This early stage plot plan usually is very limited in detail, containing only enough dimensional data to define the outer limits of the available property selected for plant development. Located within the boundaries of the available property, rough equipment sizes and shapes are pictorially positioned, along with anticipated pipe rack configurations, structure shape, and rough sizes. The plot plan at this level of detail is then used for constructability evaluation and is normally submitted to the client for approval.

Process data Sheet: A Process Data Sheet (PSD) is generally related to a single item of equipment and contains the essential process data for initiating the detail design of an item. A Process Equipment Data Sheet describes the fundamental data necessary to start discipline-wide specific engineering in the mechanical, structural, piping, control systems, and electrical areas. It includes the overall size, number, approximate geometry and identification of the

connections, material of construction and the full range of operating conditions. The Process Data sheet generally includes a simple diagram.

Procurement: is the process of finding and agreeing to terms, and acquiring goods, services, or works from an external source, often via a tendering or competitive bidding process. Procurement is used to ensure the buyer receives goods, services, or works at the best possible price when aspects such as quality, quantity, time, and location are compared. Corporations and public bodies often define processes intended to promote fair and open competition for their business while minimizing risks such as exposure to fraud and collusion.

Almost all purchasing decisions include factors such as delivery and handling, marginal benefit, and price fluctuations. Procurement generally involves making buying decisions under conditions of scarcity. If sound data is available, it is good practice to make use of economic analysis methods such as cost-benefit analysis or cost-utility analysis.

Safety Engineer: It is an engineering discipline which assures that engineered systems provide acceptable levels of safety. It is strongly related to industrial engineering/systems engineering, and the subset safety engineering. Safety engineering assures that a life-critical system behaves as needed, even when components fail.

Stress Analysis: is an engineering discipline that uses many methods to determine the stresses and strains in materials and structures subjected to forces. In continuum mechanics, stress is a physical quantity that expresses the internal forces that neighboring particles of a continuous material exert on each other, while a strain is the measure of the deformation of the material.

Technical Drawing: Technical drawing, drafting or drawing, is the act and discipline of composing drawings that visually communicate how something functions or is constructed.

The technical drawing is essential for communicating ideas in industry and engineering. To make the drawings easier to understand, people use familiar symbols, perspectives, units of measurement, notation systems, visual styles, and page layout. Together, such conventions constitute a visual language and help to ensure that the drawing is unambiguous and relatively easy to understand. Many of the symbols and principles of technical drawing are codified in an international standard called ISO 128.

Vendor Drawing: is the detailed drawing of objects you are going to buy from any vendor. It explains all the physical characteristics of an object.

6 Appendix B Code

Code for running queries parallelly and conversion from SQL to NoSQL

For simplicity, details of queries are not mentioned here. Since queries are very large. Also, other codes like conversion of SPMAT and S3D from SQL to NoSQL are not mentioned here. This code touches those domains as well, so I preferred to write this one only in this document.

```
import pyodbc

import cx_Oracle
import pymongo
from pymongo import MongoClient
import gridfs
import time
import datetime
from datetime import timedelta
import sys
from sys import getsizeof
import gc
import pandas
import numpy as np
from multiprocessing import Process, Value, Lock
import csv
import xlwt
import xlrd

beginTime = datetime.datetime.now()

databases = ["S3D16TEST_SITE",
             "S3D16TEST_SITE_SCHEMA",
             "TENOC16_CAT",
             "TENOC16_CAT_SCHEMA",
             "TENOC16_MDB",
             "TENOC16_RDB",
             "TENOC16_RDB_SCHEMA"]

#Working with first schema "S3D16TEST_SITE"
driver = "{SQL SERVER}"
server = "unknown"
database = databases[5]
user = "unknown"
passwd = "unknown"
trusted = "yes"

# Generating connection to SPMAT Oracle database
host = "unknown"
port = unknown
serviceName = "unknown"
dsn_tns = cx_Oracle.makedsn(host, port, service_name=serviceName)
```

```
username = "unknown"
password = "unknown"
```

```
def CAT_COMMODITYQ(fin):
```

```
    cnxn1 =
pyodbc.connect('DRIVER='+driver+';SERVER='+server+';DATABASE='+database+';U
ID='+user+';PWD='+passwd+';Trusted_connection='+trusted+')
    cursor1 = cnxn1.cursor()
```

```
    querY1 = """
Query
"""
```

```
    cursor1.execute(querY1)
    row1 = cursor1.fetchall()
    cursor1.close()
del cursor1
cnxn1.close()
```

```
print("\n")
print("Number of records in 3DCAT_COMMODITY: ",len(row1))
```

```
myclient1 = MongoClient()
mydb1 = myclient1["isyda"]
dic1 = dict()
listcollec1 = []
tabData1 = []
```

```
tabData1 = []
for k in row1:
    tabData1.append(k)
```

```
    cols1 = ['CAT_SPEC', 'CAT_SHORTCODE', 'CAT_OPTIONCODE',
'CAT_OPTIONCODE_DESCRIPTION', 'CAT_SIZE1',
            'CAT_SIZE2', 'CAT_COMMODITY_CODE', 'CAT_PART_TAG',
'CAT_DATE_LAST_MODIFIED']
```

```
    name1 = "3DCAT_COMMODITY"
    mycollec1 = mydb1[name1]
    listcollec1 = []
```

```
    for l in range (len(tabData1)):
        dic1.clear()
        for m in range (len(cols1)):
            colName1 = cols1[m]
            dic1[str(colName1)] = tabData1[l][m]
        listcollec1.append(dic1.copy())
    mycollec1.insert_many(listcollec1)
```

```
    fin.value = fin.value + 1
```

```
def MOD_AREASQ(fin):
```

```
    cnxn2 =
pyodbc.connect('DRIVER='+driver+';SERVER='+server+';DATABASE='+database+';U
ID='+user+';PWD='+passwd+';Trusted_connection='+trusted+')
    cursor2 = cnxn2.cursor()
```

```
querY2 = """
Query
"""

cursor2.execute(querY2)
row2 = cursor2.fetchall()
cursor2.close()
del cursor2
cnxn2.close()

print("\n")
print("Number of records in 3DMOD_AREASQ: ",len(row2))

myclient2 = MongoClient()
mydb2 = myclient2["isyda"]
dic2 = dict()
listcollec2 = []
tabData2 = []

tabData2 = []
for k in row2:
    tabData2.append(k)

cols2 = ['LowX', 'LowY', 'HighX', 'HighY', 'Unità', 'Tipo Area',
'Design Area']

name2 = "3DMOD_AREAS"
mycollec2 = mydb2[name2]
listcollec2 = []

for l in range (len(tabData2)):
    dic2.clear()
    for m in range (len(cols2)):
        colName2 = cols2[m]
        dic2[str(colName2)] = tabData2[l][m]
        listcollec2.append(dic2.copy())
    mycollec2.insert_many(listcollec2)

fin.value = fin.value + 1

def MOD_OBJECTSQ(fin):

    cnxn3 =
pyodbc.connect('DRIVER='+driver+';SERVER='+server+';DATABASE='+database+';U
ID='+user+';PWD='+passwd+';Trusted_connection='+trusted+')
    cursor3 = cnxn3.cursor()

    querY3 = """
Query
"""

    cursor3.execute(querY3)
    row3 = cursor3.fetchall()
    cursor3.close()
    del cursor3
    cnxn3.close()

    print("\n")
    print("Number of records in 3DMOD_OBJECTSQ: ",len(row3))
```

```
myclient3 = MongoClient()
mydb3 = myclient3["isyda"]
dic3 = dict()
listcollec3 = []
tabData3 = []

tabData3 = []
for k in row3:
    tabData3.append(k)

cols3 = ['3D_WBS_PROJECT', '3D_WBS_UNIT', '3D_WBS_AREA', '3D_WBS_LINE',
'3D_PIPELINE', 'SPEC',
        'SHORTCODE', 'OPTIONCODE', 'OPTIONCODE_DESCRIPTION', 'SIZE1',
'SIZE2', 'BOLT_LENGTH',
        'COMMODITY CODE', 'PART(tag)', 'Date Last Modified']

name3 = "3DMOD_OBJECTS"
mycollec3 = mydb3[name3]
listcollec3 = []

for l in range (len(tabData3)):
    dic3.clear()
    for m in range (len(cols3)):
        colName3 = cols3[m]
        dic3[str(colName3)] = tabData3[l][m]
    listcollec3.append(dic3.copy())
mycollec3.insert_many(listcollec3)

fin.value = fin.value + 1

def MOD_PIPERUNDATA_CLLQ(fin):

    cnxn4 =
pyodbc.connect('DRIVER='+driver+';SERVER='+server+';DATABASE='+database+';U
ID='+user+';PWD='+passwd+';Trusted_connection='+trusted+')
    cursor4 = cnxn4.cursor()

    querY4 = """
Query
"""

    cursor4.execute(querY4)
    row4 = cursor4.fetchall()
    cursor4.close()
    del cursor4
    cnxn4.close()

    print("\n")
    print("Number of records in 3DMOD_PIPERUNDATA_CLLQ: ",len(row4))

    myclient4 = MongoClient()
    mydb4 = myclient4["isyda"]
    dic4 = dict()
    listcollec4 = []
    tabData4 = []

    tabData4 = []
    for k in row4:
        tabData4.append(k)
```

```
cols4 = ['3D_WBS_PROJECT', '3D_WBS_UNIT', '3D_WBS_AREA', '3D_WBS_LINE',
'3D_PIPELINE', 'Run Name',
'Fluid', 'Run Piping Class', 'Run NPD', 'Run NPD Unit', 'Pid', 'Rx
Test', 'Heat Treat',
'FINISH CD', 'FINISH CD_DESCRIPTION', 'Ins Thk1', 'P Spec Mat',
'RATING', 'Norm Des Press',
'Norm Des Temp', 'Paint Sys', 'Norm Op Press', 'Norm Op Temp', 'Reg
Tst P', 'TRACING CD',
'Corr All', 'Main Diam']

name4 = "3DMOD_PIPERUN_CLL"
mycollec4 = mydb4[name4]
listcollec4 = []

for l in range (len(tabData4)):
    dic4.clear()
    for m in range (len(cols4)):
        colName4 = cols4[m]
        dic4[str(colName4)] = tabData4[l][m]
    listcollec4.append(dic4.copy())
mycollec4.insert_many(listcollec4)

fin.value = fin.value + 1

def MOD_PIPERUNSQ(fin):

    cnxn5 =
pyodbc.connect('DRIVER='+driver+';SERVER='+server+';DATABASE='+database+';U
ID='+user+';PWD='+passwd+';Trusted_connection='+trusted+')
    cursor5 = cnxn5.cursor()

    queryY5 = """
Query
"""

    cursor5.execute(queryY5)
    row5 = cursor5.fetchall()
    cursor5.close()
    del cursor5
    cnxn5.close()

    print("\n")
    print("Number of records in 3DMOD_PIPERUNSQ: ",len(row5))

    myclient5 = MongoClient()
    mydb5 = myclient5["isyda"]
    dic5 = dict()
    listcollec5 = []
    tabData5 = []

    tabData5 = []
    for k in row5:
        tabData5.append(k)

    cols5 = ['3D_WBS_PROJECT', '3D_WBS_UNIT', '3D_WBS_AREA', '3D_WBS_LINE',
'3D_SPEC', '3D_SIZE']

    name5 = "3DMOD_PIPERUNS"
    mycollec5 = mydb5[name5]
    listcollec5 = []
```

```
for l in range (len(tabData5)):
    dic5.clear()
    for m in range (len(cols5)):
        colName5 = cols5[m]
        dic5[str(colName5)] = tabData5[l][m]
    listcollec5.append(dic5.copy())
mycollec5.insert_many(listcollec5)

fin.value = fin.value + 1

def TNP_IT_DV_BOM_DETAIL(fin):
    db6 = cx_Oracle.connect(username, password, dsn_tns, encoding = "UTF-
8", nencoding = "UTF-8")
    cur6 = db6.cursor()

    cur6.execute("""
begin
mpck_login.select_login(107023);
end;
""")

    cur6.execute("alter session set nls_numeric_characters = '.,' ")
    cur6.execute("alter session set nls_date_language = 'AMERICAN' ")
    cur6.execute("alter session set nls_date_format = 'DD-MON-RRRR' ")

    statement = """
Query
"""

    cur6.execute(statement)
    row6 = cur6.fetchall()

    print("\n")
    print("Number of records in TNP_IT_DV_BOM_DETAIL: ",len(row6))

    myclient6 = MongoClient()
    mydb6 = myclient6["isyda"]
    dic6 = dict()
    listcollec6 = []
    tabData6 = []

    tabData6 = []
    for k in row6:
        tabData6.append(k)

    cols6 = ['DISC', 'UNIT', 'AREA', 'LINE', 'SHEET', 'BEST_ISSUE_STATUS',
'ISSUE_STATUS', 'LIST_STATUS', 'TRANS',
'SPEC_CODE', 'SHORT_CODE', 'OPTION_CODE', 'IDENT_CODE',
'COMMODITY_CODE', 'COMMODITY_DESCRIPTION', 'DIA1',
'DIA2', 'THK1', 'THK2', 'DIA1_C', 'DIA2_C', 'DN_UNIT', 'OBJ',
'CG_GROUP_CODE', 'PART_CODE', 'LP_QTY',
'ISSUE_QTY', 'RESV_QTY', 'QTY_UNIT', 'TAG_NUMBER', 'FABCAT',
'PURC_DEST', 'UNIT_WEIGHT', 'total_weight',
'UNIT_WEIGHT_VENDOR', 'ITY_CODE', 'ASSEMBLY', 'PID_MTO', 'MTO_REV',
'ISSUE_DATE_1', 'ISSUE_DATE_2',
'ISSUE_DATE_3', 'ISSUE_DATE_4', 'ISSUE_DATE_5', 'ISSUE_DATE',
'CRITICITY', 'BOM_NODE_WEIGHT', 'INS_MAT',
```

```

        'INS_THK1', 'INS_THK2', 'PAINT_SYS', 'TRACING_CD', 'FINISH_CD',
'INPUT_1', 'INPUT_2', 'INPUT_3', 'INPUT_4',
        'ID_INPUT_1', 'ID_INPUT_2', 'ID_INPUT_3', 'ID_INPUT_4', 'LP_POS',
'LP_SUB_POS', 'NLS_ID', 'LN_ID', 'LP_ID',
        'PARENT_LP_ID', 'STAT_ID', 'LST_ID', 'SPEC_HEADER_ID', 'IDENT',
'COMMODITY_ID', 'GROUP_ID', 'UNIT_ID', 'ITY_ID']

```

```

name6 = "TNP_IT_DV_BOM_DETAIL"
mycollec6 = mydb6[name6]
listcollec6 = []

```

```

for l in range (len(tabData6)):
    dic6.clear()
    for m in range (len(cols6)):
        colName6 = cols6[m]
        dic6[str(colName6)] = tabData6[l][m]
    listcollec6.append(dic6.copy())
mycollec6.insert_many(listcollec6)

```

```

fin.value = fin.value + 1

```

```

def TNP_IT_DV_FR_LINES(fin):
    db7 = cx_Oracle.connect(username, password, dsn_tns, encoding = "UTF-
8", nencoding = "UTF-8")
    cur7 = db7.cursor()

```

```

    cur7.execute("""
begin
mpck_login.select_login(107023);
end;
""")

```

```

    cur7.execute("alter session set nls_numeric_characters = '.,' ")
    cur7.execute("alter session set nls_date_language = 'AMERICAN' ")
    cur7.execute("alter session set nls_date_format = 'DD-MON-RRRR' ")

```

```

statement = """
Query
"""

```

```

    cur7.execute(statement)
    row7 = cur7.fetchall()

```

```

    print("\n")
    print("Number of records in TNP_IT_DV_FR_LINES: ",len(row7))

```

```

myclient7 = MongoClient()
mydb7 = myclient7["isyda"]
dic7 = dict()
listcollec7 = []
tabData7 = []

```

```

tabData7 = []
for k in row7:
    tabData7.append(k)

```

```

    cols7 = ['FAH_ID', 'FAH_CODE', 'RUN_NUMBER', 'FAH_TYPE', 'LN_ID',
'BOM_PATH', 'LP_POS', 'LP_ID', 'WH_CODE',
        'PRED_ON_SITE_DATE', 'IDENT_CODE', 'TAG_NUMBER',
'CG_GROUP_CODE', 'PART_CODE', 'COMMODITY_CODE', 'IDENT_DESC',

```

```
'DIA1', 'DIA2', 'UNIT_DIA', 'WALL1', 'WALL2', 'UNIT_WALL',
'IDENT_WEIGHT', 'UNIT_DIA_INCH', 'LST_CODE',
'STATUS', 'STAT_ID', 'FABCAT', 'PURC_DEST', 'SPEC_CODE',
'LP_QTY', 'LP_ISSUE_QTY', 'LP_RESV_QTY',
'ACTUAL_RESV_QTY', 'TOTAL_RESV_QTY', 'Shortage_Qty',
'global_Qty', 'ON_HAND_QTY']

name7 = "TNP_IT_DV_FR_LINES"
mycollec7 = mydb7[name7]
listcollec7 = []

for l in range (len(tabData7)):
    dic7.clear()
    for m in range (len(cols7)):
        colName7 = cols7[m]
        dic7[str(colName7)] = tabData7[l][m]
    listcollec7.append(dic7.copy())
mycollec7.insert_many(listcollec7)

fin.value = fin.value + 1

def TNP_IT_DV_PIPING_BOM(fin):
    db8 = cx_Oracle.connect(username, password, dsn_tns, encoding = "UTF-
8", nencoding = "UTF-8")
    cur8 = db8.cursor()

    cur8.execute("""
begin
mpck_login.select_login(107023);
end;
""")

    cur8.execute("alter session set nls_numeric_characters = '.,' ")
    cur8.execute("alter session set nls_date_language = 'AMERICAN' ")
    cur8.execute("alter session set nls_date_format = 'DD-MON-RRRR' ")

    statement = """
Query
"""

    cur8.execute(statement)
    row8 = cur8.fetchall()

    print("\n")
    print("Number of records in TNP_IT_DV_PIPING_BOM: ",len(row8))

    myclient8 = MongoClient()
    mydb8 = myclient8["isyda"]
    dic8 = dict()
    listcollec8 = []
    tabData8 = []

    tabData8 = []
    for k in row8:
        tabData8.append(k)

    cols8 = ['DISC', 'UNIT', 'AREA', 'LINE', 'SHEET', 'LN_ID']

    name8 = "TNP_IT_DV_PIPING_BOM"
    mycollec8 = mydb8[name8]
```



```
listcollec8 = []

for l in range (len(tabData8)):
    dic8.clear()
    for m in range (len(cols8)):
        colName8 = cols8[m]
        dic8[str(colName8)] = tabData8[l][m]
    listcollec8.append(dic8.copy())
mycollec8.insert_many(listcollec8)

fin.value = fin.value + 1

def TNP_IT_DV_PLL(fin):
    db9 = cx_Oracle.connect(username, password, dsn_tns, encoding = "UTF-8", nencoding = "UTF-8")
    cur9 = db9.cursor()

    cur9.execute("""
begin
mpck_login.select_login(107023);
end;
""")

    cur9.execute("alter session set nls_numeric_characters = '.,' ")
    cur9.execute("alter session set nls_date_language = 'AMERICAN' ")
    cur9.execute("alter session set nls_date_format = 'DD-MON-RRRR' ")

    statement = """
select * from (
Query
"""

    cur9.execute(statement)
    row9 = cur9.fetchall()

    print("\n")
    print("Number of records in TNP_IT_DV_PLL: ",len(row9))

    myclient9 = MongoClient()
    mydb9 = myclient9["isyda"]
    dic9 = dict()
    listcollec9 = []
    tabData9 = []

    tabData9 = []
    for k in row9:
        tabData9.append(k)

    cols9 = ['PLL_UNITS', 'UNIT_SECT', 'FLUID_CODE', 'DESCRIPTION2',
'LINE', 'DN', 'STATUS', 'CANCELLED', 'SPEC_HEADER',
'P_SPEC_MAT', 'DESCRIPTION', 'CORR_ALL', 'FACING', 'FROM_LOCATION',
'TO_LOCATION', 'FINISH_CD', 'FL_PH', 'JACKET_LINE',
'PID', 'CHEM_TREAT', 'HEAT_TREAT', 'MAINT_T', 'NORM_DESCR',
'NORM_DES_PRESS', 'NORM_OP_PRESS', 'PRESS_UN',
'NORM_DES_TEMP', 'NORM_OP_TEMP', 'TEMP_UN', 'NORM_FG_CAL',
'NORM_FG_TST', 'NORM_PED_GR', 'NORM_PED_PH', 'EQUIP_CAT',
'DELIVERY_PCK', 'M_CRITIC_RATING_FCT', 'ALT1_DESCR', 'ALT1_DES_P',
'ALT1_OPE_P', 'ALT1_DES_T', 'ALT1_OPE_T',
'ALT1_FG_CAL', 'ALT1_FG_TST', 'ALT1_PED_GR', 'ALT1_PED_PH',
'ALT2_DESCR', 'ALT2_DES_P', 'ALT2_OPE_P', 'ALT2_DES_T',
```

```
'ALT2_OPE_T', 'ALT2_FG_CAL', 'ALT2_FG_TST', 'ALT2_PED_GR',
'ALT2_PED_PH', 'ALT3_DESCR', 'ALT3_DES_P', 'ALT3_OPE_P',
'ALT3_DES_T', 'ALT3_OPE_T', 'ALT3_FG_CAL', 'ALT3_FG_TST',
'ALT3_PED_GR', 'ALT3_PED_PH', 'ALT4_DESCR', 'ALT4_DES_P',
'ALT4_OPE_P', 'ALT4_DES_T', 'ALT4_OPE_T', 'ALT4_FG_CAL',
'ALT4_FG_TST', 'ALT4_PED_GR', 'ALT4_PED_PH', 'ALT_FG_INS',
'TST_P_TYP', 'ALT_FG_PAINT', 'REG_TST_P', 'MAX_TST_P', 'PLL_NOTE1',
'PLL_NOTE2', 'PLL_NOTE3', 'PLL_NOTE4',
'PLL_NOTE5', 'PLL_REV', 'PLL_DATE_REV', 'DIAMETER', 'DIAMETER_UN',
'PIPE_THK', 'PIPE_THK_UM', 'WT_FORCED',
'INS_LMT', 'INS_JACKET', 'INS_MAT', 'INS_THK1', 'INS_THK2',
'TRACING_CD', 'PAINT_CODE', 'PAINT_SYS', 'PAINT_SYS2',
'INSUL_MAN', 'TRACING_MAN', 'PAINT_SYS_MAN', 'WTMECC_MIN',
'PSxDN_NORM', 'PSxDN_ALT1', 'PSxDN_ALT2', 'PSxDN_ALT3',
'PSxDN_ALT4', 'NORM_VAP_PRESS', 'MIN_EFF_WT', 'EQUIP_CAT_NORM',
'EQUIP_CAT_ALT1', 'EQUIP_CAT_ALT2', 'EQUIP_CAT_ALT3',
'EQUIP_CAT_ALT4', 'DN_COMMENT', 'FL_DENS', 'DIAMETER_MM',
'UNIT_MM', 'SYSTEM_CD', 'SUB_SYSTEM', 'TEMP_PAINTING_SET',
'PAINTING_METHOD', 'P_SPEC_RATING', 'P_SPEC_LOCATION',
'line_max_dia', 'TEMP_INS_THK', 'TEMP_PAINT_SYS', 'TRAC_DIAM',
'TRAC_NUMBER', 'PWHT', 'RX_TEST', 'MAX_HARDNESS', 'MP_TP',
'MAIN_TEMPERATURE', 'SEQ_NUMBER', 'CANCELLED_PAR']
```

```
name9 = "TNP_IT_DV_PLL"
mycollec9 = mydb9[name9]
listcollec9 = []

for l in range (len(tabData9)):
    dic9.clear()
    for m in range (len(cols9)):
        colName9 = cols9[m]
        dic9[str(colName9)] = tabData9[l][m]
    listcollec9.append(dic9.copy())
mycollec9.insert_many(listcollec9)

fin.value = fin.value + 1
```

```
def REF_CLL(fin):
```

```
    df = pandas.read_excel('C:\\Users\\MRahmad\\Desktop\\python-
isyda\\REF_CLL.xlsx')
    cols10 = df.columns.tolist()
    tabData10 = df.values.tolist()

    myclient10 = MongoClient()
    mydb10 = myclient10["isyda"]
    dic10 = dict()

    name10 = "REF_CLL"
    mycollec10 = mydb10[name10]
    listcollec10 = []

    for l in range (len(tabData10)):
        dic10.clear()
        for m in range (len(cols10)):
            if cols10[m] == 'Pll Units':
                colName10 = cols10[m]
                dic10[str(colName10)] = str(tabData10[l][m]).zfill(2)
            else:
```

```
        colName10 = cols10[m]
        dic10[str(colName10)] = tabData10[l][m]
    listcollec10.append(dic10.copy())
mycollec10.insert_many(listcollec10)

print("\n")
print("Number of records in REF_CLL: ",len(tabData10))

fin.value = fin.value + 1

if __name__ == '__main__':

    fin = Value('i', 0)

    p1 = Process(target=CAT_COMMODITYQ, args=(fin,))
    p2 = Process(target=MOD_AREASQ, args=(fin,))
    p3 = Process(target=MOD_OBJECTSQ, args=(fin,))
    p4 = Process(target=MOD_PIPERUNDATA_CLLQ, args=(fin,))
    p5 = Process(target=MOD_PIPERUNSQ, args=(fin,))

    p6 = Process(target=TNP_IT_DV_BOM_DETAIL, args=(fin,))
    p7 = Process(target=TNP_IT_DV_FR_LINES, args=(fin,))
    p8 = Process(target=TNP_IT_DV_PIPING_BOM, args=(fin,))
    p9 = Process(target=TNP_IT_DV_PLL, args=(fin,))

    p10 = Process(target=REF_CLL, args=(fin,))

    p1.start()
    p2.start()
    p3.start()
    p4.start()
    p5.start()

    p6.start()
    p7.start()
    p8.start()
    p9.start()

    p10.start()

    p1.join()
    p2.join()
    p3.join()
    p4.join()
    p5.join()

    p6.join()
    p7.join()
    p8.join()
    p9.join()

    p10.join()

    if fin.value == 10:
        finalTime = datetime.datetime.now() - beginTime
        print("\nTotal time: ", finalTime)
```

