**Summary**

This essay explains about the technical learning journey I have had in my time at ICBC. It focuses on technical aspects and processes involved for getting the job done. As a data engineer, one should be able to understand these processes deeply and be flexible for learning new technologies everyday as newer software become available and integrated to the processes. It is best attempted to avoid technical jargon and explain the material in the simplest manner. There is a background needed for understanding the process of creating data pipelines, and that background knowledge is given in the introduction section. Then in discussion section, the breakdown of process is described. This essay is written in a way that after reading it, the reader is going to be most likely have a good understanding of big picture of how things related to data engineering are done in ICBC.

Contents

[Section 1 : Introduction 3](#_Toc74576362)

[Section 2 : Discussion 6](#_Toc74576363)

[Section 2.1: Moving Source Files to Big Data (Allocated Weight: 5%): 8](#_Toc74576364)

[Section 2.2 : Creating Mapping Document(Allocated Weight: 10%): 8](#_Toc74576365)

[Section 2.3 : Reading Source Tables in Spark and Creating Staging Tables(Allocated Weight: 20%): 9](#_Toc74576366)

[Section 2.3 : Applying the SQL Queries on Staging Tables by Changing those Queries as Suited to Spark Environment (Allocated Weight: 15%): 10](#_Toc74576367)

[Section 2.4 : Creating Enterprise Tables Based on Transformations and Joins after the Query was Applied (Allocated Weight: 25%): 11](#_Toc74576368)

[Section 2.5: Integrating the Entire Code base to Hadoop Cluster (Allocated Weight: 10%): 11](#_Toc74576369)

[Section 2.6: Data Validation (Allocated Weight: 10%): 12](#_Toc74576370)

[Section 2.7: Setting up Fisheye Code Review (Allocated Weight: 5%): 12](#_Toc74576371)

[Section 3: References 13](#_Toc74576372)

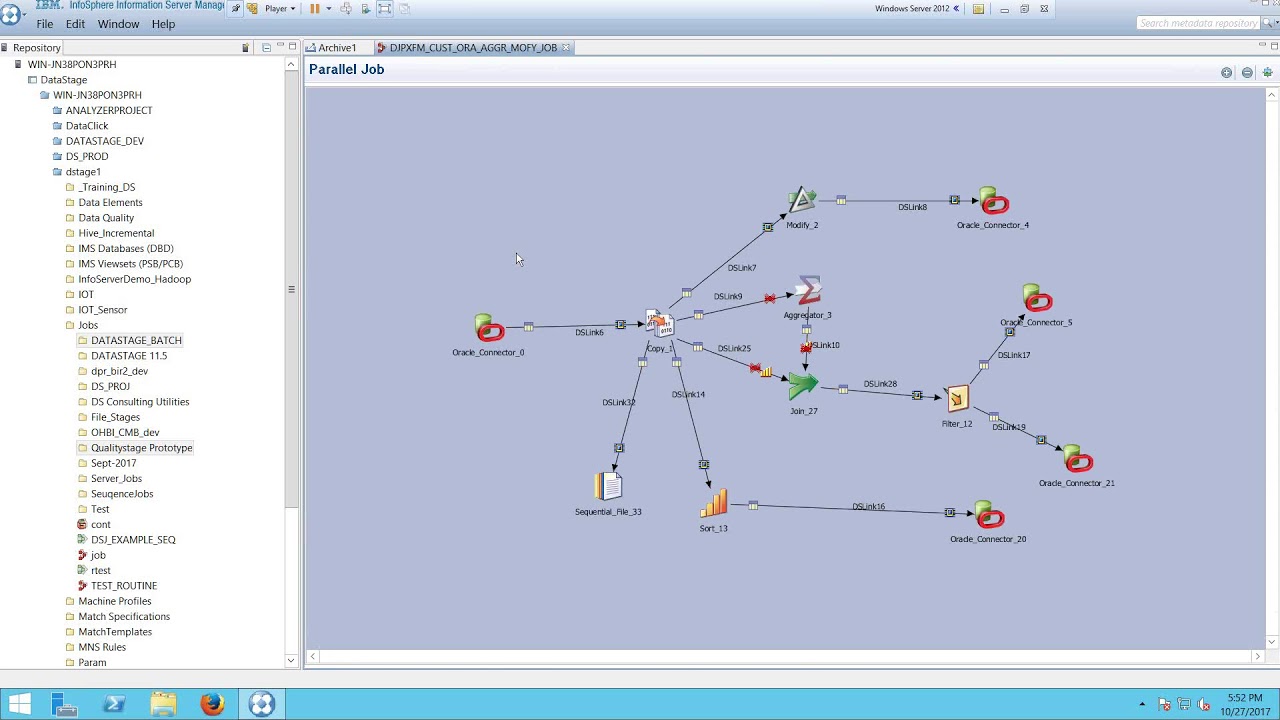
# Section 1 : Introduction

To give a broad perspective, I will start by giving some information about what the Controllable Cost report is and for what reason it is being used. The Controllable Costs Cube was developed in 2004, to provide an efficient means to analyze controllable costs for the development of financial requirements for preparation of Rate Applications to the British Columbia Utilities Commission and to allow Finance analysts to respond quickly and accurately to questions posed by BCUC about rate applications. The cube is also used by others in the Finance area, where analysis of Controllable Costs is required for non-BCUC related purposes. The Controllable Cost Cube allows the user to drill down on a given expense type, via a variety of business unit views and cost centre hierarchies, and to apply appropriate Basic/Optional and Non-Insurance percentages, as required. Multiple versions of cost data are maintained on the cube, allowing users to inquire about specific instances of cost data (such as Actuals, Plan, etc). Therefore, two important factors while maintaining this report are the accuracy of the data, and the speed at which the tables get updated.

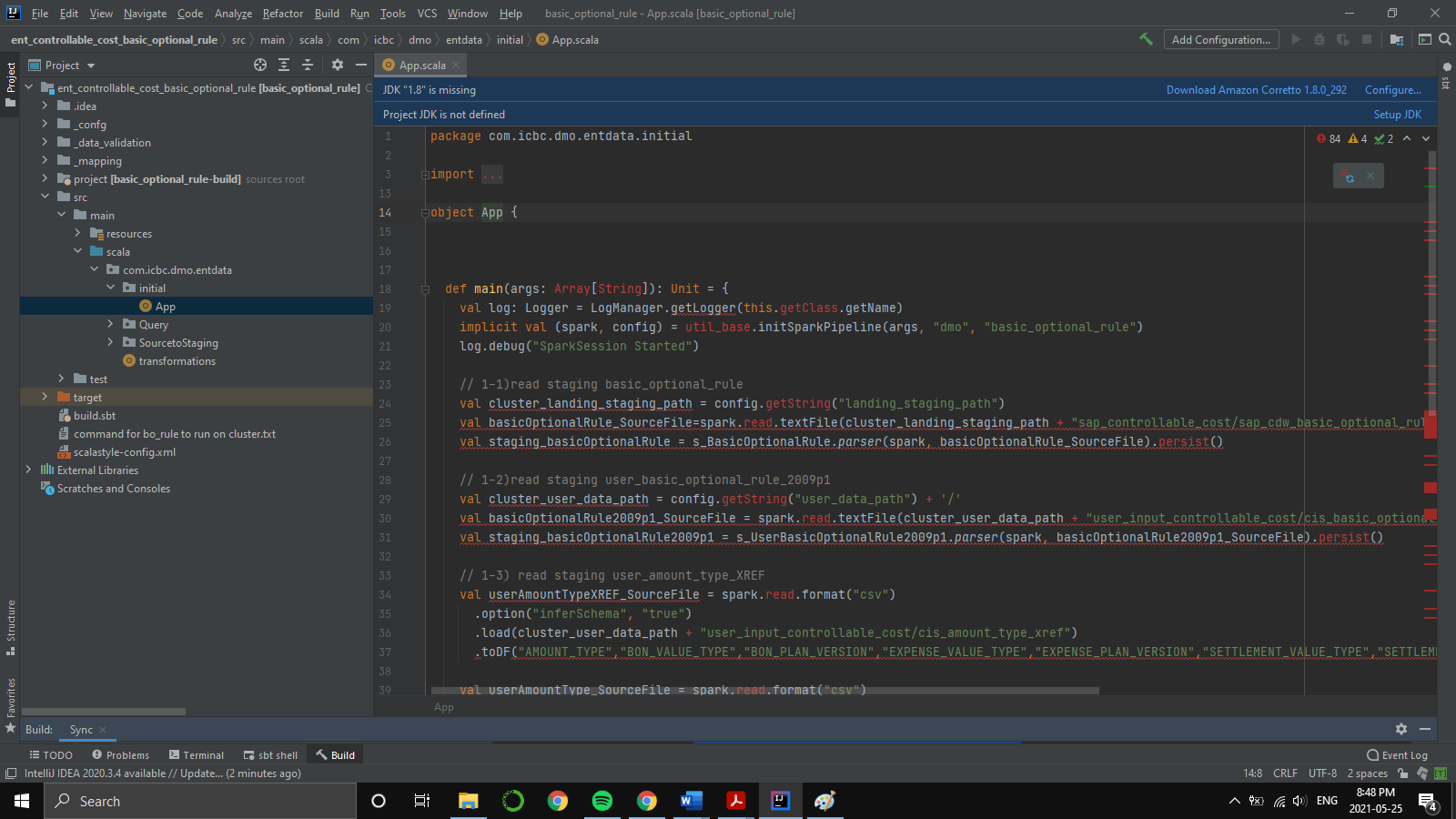
Until few months ago, ICBC utilized Enterprise Data Warehouse (EDW) to create the final report from the base data that was generated by one of its business systems used internally and named SAP. With the new big data technologies coming to the market, the Information Management made a decision to decommission EDW and utilize the fast and in memory Spark engine. To create a roadmap of transformations from the source data to final data we need a data pipeline. A data pipeline is a series of logical transformations that take the source data and make it useful in its format that is desired by business users of that report. Since ICBC decided to move towards creating every pipeline in EDW system to Big Data and Hadoop cluster, so the Data Engineers had to move the pipelines responsible for creating this particular report to Big Data. This was the job that was initially assigned to me. For Controllable Cost report and pipeline, there are a total of 7 smaller pipelines that needed to be created and tested. Each pipeline needed to produce accurate results individually, but the most important aspect was that all the pipelines produce correct results when collaborating with other pipelines.

In its essence, a pipeline uses a mixture of SQL queries and logical transformations to group together, change the format, get a sum, average, or manipulate and mix the data in its own way. Therefore, the nature of the job for each pipeline remains the same between EDW and Big Data platform, however, the implementation will for each system is going to be its own unique way as EDW uses IBM’s DataStage to create and manage the pipelines, whereas Big Data platform uses the Spark Engine and a variety of integrated Hadoop platforms to make sure the pipelines are running smoothly. As mentioned before, the speed at which the Big Data platform loads each pipeline is significantly faster (100x times) than EDW and of the main reasons for this is that Spark engine runs on memory basis whereas EDW system uses MapReduce which runs on disk and when a program runs on memory, it reduces the time necessary for Input/Output operations significantly. Another advantage of spark is that the programmers can write the code in a variety of different ways and using their own style of coding, however, there is a graph called directed acyclic graph (DAG) that is responsible for optimizing and simplifying the code that are understood by the machine. Therefore, it can be said that spark uses lazy evaluation meaning some parts of the code are not executed until absolutely needed, and this is also one of the reasons that make spark extremely faster in comparison. (Williams, n.d.)

There is a fundamental difference between how a pipeline is created in EDW versus Big Data platform. In EDW, a program by IBM called DataStage was used to create and run the pipelines on weekly basis. This program used a mixture of SQL statements and visuals to guide the Data Engineer create the pipeline, and then a Database Administrator would decide how often the pipelines should be run to refresh the tables. On the other hand, in Big Data, creating a pipeline was more involved with programming languages such as Spark and Scala, and one needed to have the programming skills required to design a pipeline. Therefore, it was my job as a junior Data Engineer to understand the grammar of DataStage, and to apply the same logic within the programming framework of Spark and Scala. Next, you will be able to see the difference between pipeline frameworks DataStage versus Scala.



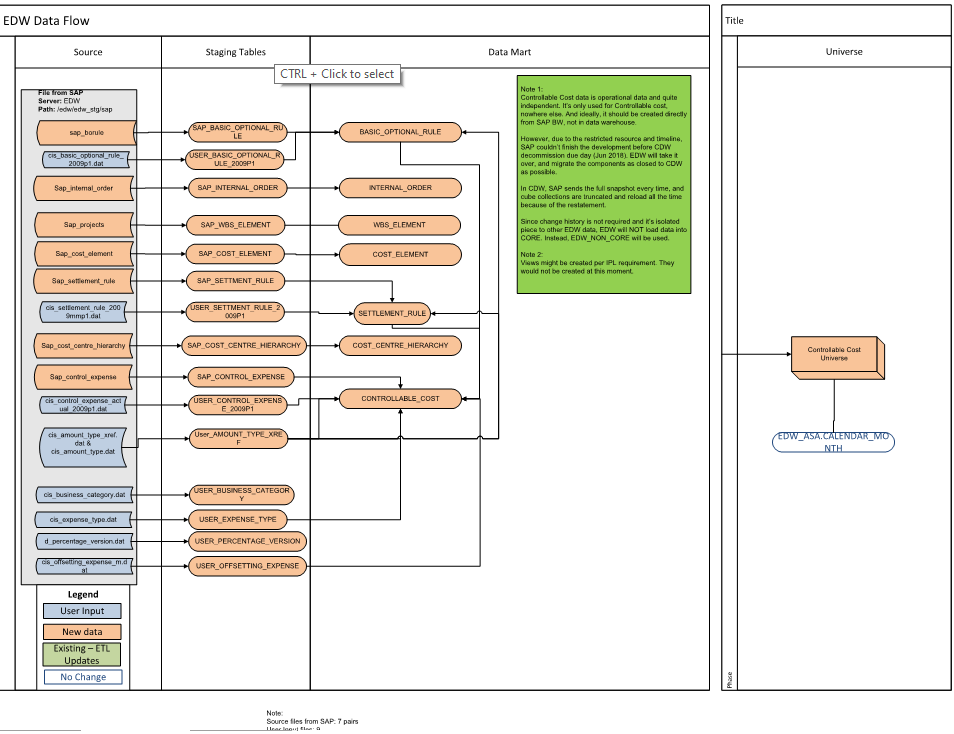
DataStage Environment Retrieved from : https://www.youtube.com/watch?v=S49mLFjh3EM

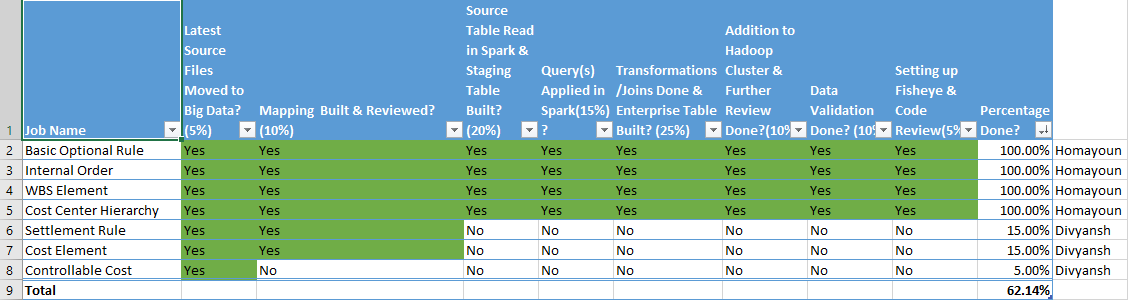


Example of one of the pipelines I created in Scala and by using Spark

# Section 2 : Discussion

Now, let us discuss in more details how I created my first pipeline. It is all about how the data is generated overall from the start to the end. Source data is received from a business system called SAP. This is a system that most internal and external users of ICBC work with and generates a basic dataset every week and sends that dataset to the transformation engine (past EDW, now Big Data). The Controllable Cost pipeline was unique because the format of source data that was received from SAP was quite different than other types of pipelines. Therefore, I could not exactly follow the same pattern as other data engineers and I needed to create a process bar to keep track of each pipeline. This process bar was also a good map for my own understanding of what needs to be done, and a decent representation of my work to my manager and other business users, without having to engage too much in the details. The process bar evolved as I learnt more about how to create each pipeline, but it was absolutely necessary to have it even in the earliest time of getting started because when I did not have it, I could not keep track of what exactly it is that I was doing, and it was really hard to explain things to my manager because he usually had a limited 15 minutes of time per week to speak with me. Here we will be going into more details about each part of the progress bar and explain why it is important.

This is how different smaller pipeline interacted with each other to create the results.

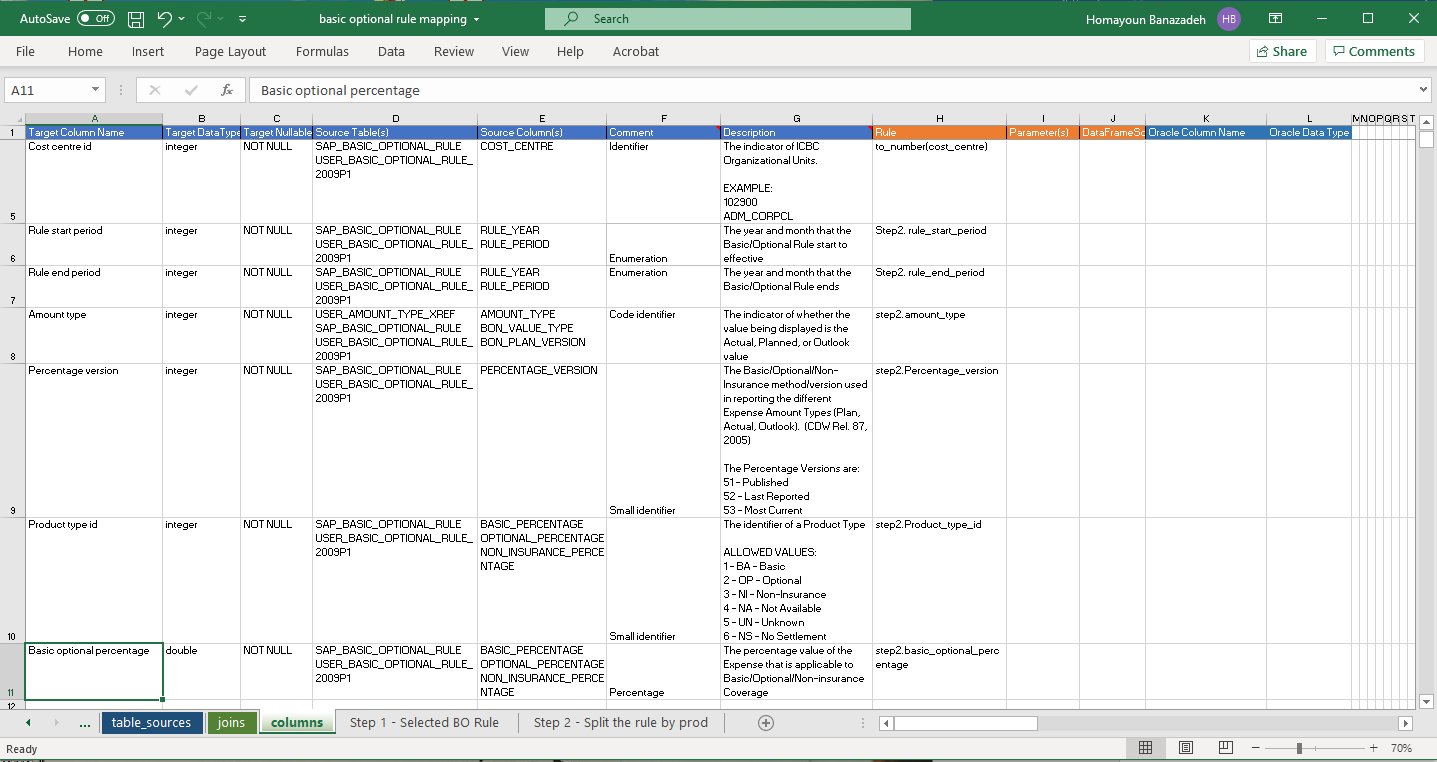


This is the progress bar that was created for this project specifically.

# Section 2.1: Moving Source Files to Big Data (Allocated Weight: 5%):

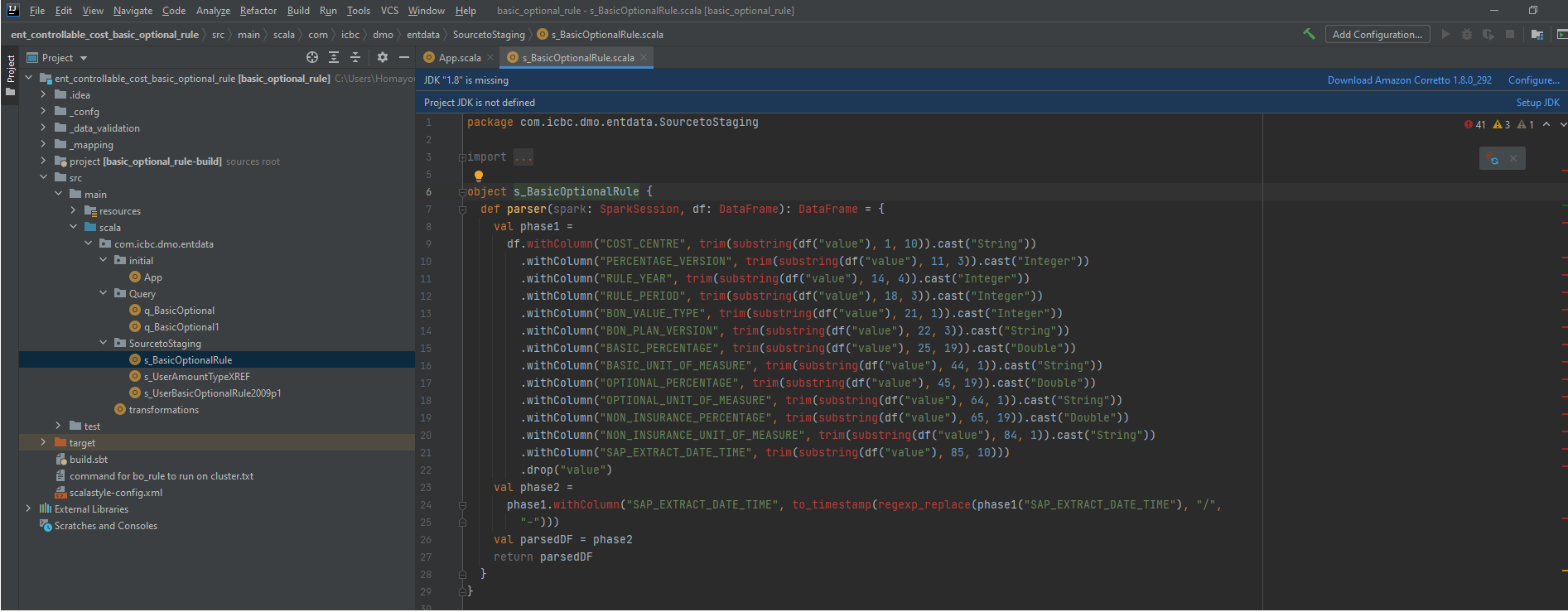
This step involved moving the source data files to Hadoop platform. The hard part about moving the source data files was that they no one knew where they were located. Since the design of Controllable Cost in EDW, lots had happened and the people who designed it had already left the organization. Therefore, this step involved constant communication with the senior Data Engineer and from him to the business owner of Controllable Costs to find the location of source files. Since these files are kept in a highly protected environment, I initially intended to learn how to access them myself, and it was an interesting task because there were multiple layers of security, and one needed to learn how to access those files in Shell environment by using specific shell commands, however, due to high volume of security, I did not receive the eligibility.

# Section 2.2 : Creating Mapping Document(Allocated Weight: 10%):

A mapping document is a high-level representation of the work that needs to be done. This includes information about what the source tables and columns are, and what is going to be the target table. Most importantly, it includes what the steps are that are involved in going from the source data to reaching the target data. Therefore, if there is a join, or a small query, or a change in name of the columns, they should be mentioned there. 

Example of mapping document.

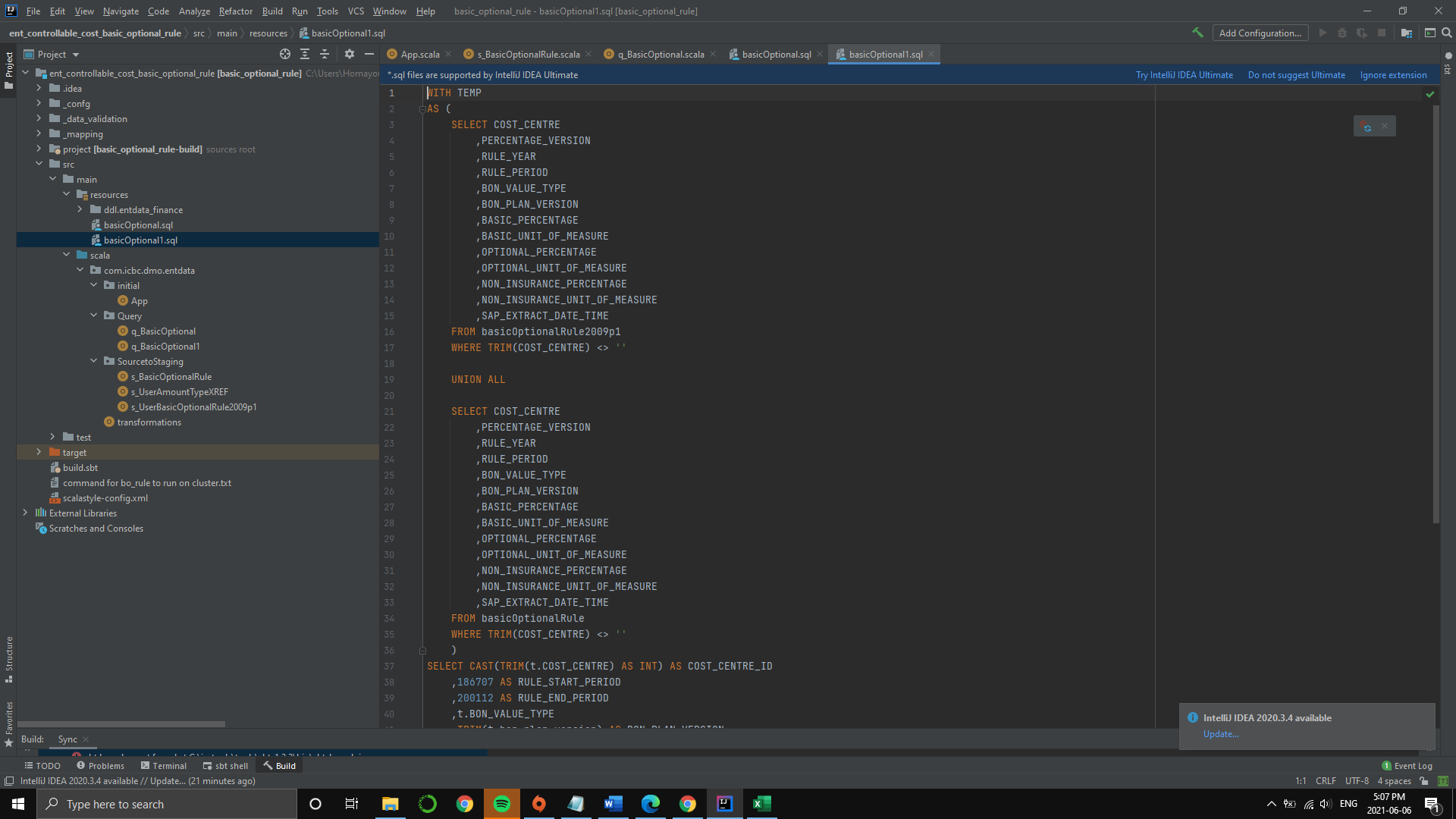
# Section 2.3 : Reading Source Tables in Spark and Creating Staging Tables(Allocated Weight: 20%):

The main problem here was that the columns in source tables were not delimited by any character (such as comma) and if they were, I would be able to use the ICBC libraries dedicated for Data Engineers. However, each column was arbitrarily separated by starting from a character in the line up to another character, and these numbers were specified in DataStage, so that way I was able to find them out and do the same task in Spark. For this task and many others, as all pipeline followed a similar format, I made use of separating my code into different parts so that each part will be of use in a later time for another pipeline. 

Example of reading the source file of one of the pipelines.

# Section 2.3 : Applying the SQL Queries on Staging Tables by Changing those Queries as Suited to Spark Environment (Allocated Weight: 15%):

In one of the stages of data transformation in DataStage, a long query needed to be applied on the result of previous step to receive a new table. I did not know why these types of queries were applied or what it was that they were doing. They were extremely large, and I limited myself to just making things work. However, there was one problem. The queries were written by Oracle SQL and I needed to use Spark SQL for them. Sometimes the syntax was completely different or in a more intense time one of the functions that was required to perform recursive operation was not even implemented in Spark SQL still. So even though it took 2 weeks to come up with an implementation myself, this was an interesting data engineering challenge. To understand the queries and debug them, it was a very hard challenge to figure out what part of this query is not working in Spark, and I did a research to find PoorSQL for better query demonstration.



Example of applying the query on base table

# Section 2.4 : Creating Enterprise Tables Based on Transformations and Joins after the Query was Applied (Allocated Weight: 25%):

This was usually the main meaningful step in a pipeline that included figuring out the logic from DataStage and old mapping documents and doing the same in Spark environment. Over time, I slowly came to understand what different concepts mean in DataStage and I became faster to understand what needs to be done.

# Section 2.5: Integrating the Entire Code base to Hadoop Cluster (Allocated Weight: 10%):

This step included integrating the entire code base with ICBC Hadoop clusters. This was an essential and time taking process because there are some configuration files that needed to be created and learning their settings and how they connect to ICBC code base libraries was not trivial. On the other hand, debugging a certain problem here was also an important skill to learn because one needed to debug the entire program on Hadoop cluster. This meant debugging both the program itself and also the configurations including the DDL(data definition language and schema for final tables), the source and final table locations on the cluster, and many other settings. Because of that, it would be important for everything in the code base to be tested and made sure it is working properly before moving to the cluster.

# Section 2.6: Data Validation (Allocated Weight: 10%):

After the code was working properly on the cluster, it was time for data validation. To ensure the data matched, two criteria were examined and compared against EDW tables. First point was the distinct counts of every column in the target tables. These quantities should have matched the ones in EDW tables. Second point, a query was made to get some specific records from the EDW table, the same query was run on big data to evaluate whether the records are indeed similar. The results of queries were put on Excel and compared against each other. If these two criteria were met, one could claim the tables are more or less the same and move on to the next pipeline.

# Section 2.7: Setting up Fisheye Code Review (Allocated Weight: 5%):

After the data validation step was done, the result of all the code base was uploaded on Fisheye for code optimization and recommendations at that point were made by a senior data engineer to enhance the code and make improvements. After this point, the code was finalized.

# Section 3: References

Williams, M. (n.d.). *Apache Spark vs. MapReduce*. Retrieved from Dzone: https://dzone.com/articles/apache-spark-introduction-and-its-comparison-to-ma