Advancing Massachusetts' Technology, Talent and Economy Reporting System (MATTERS)

A Major Qualifying Project to be submitted to the faculty of Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Bachelor of Science.

By:

Alex Fortier

Long Hoang Nguyen Duc

Kevin Mee

Westley Russell

Date:

April 30, 2015

Submitted to:

_____

Prof. Elke A. Rundensteiner

Worcester Polytechnic Institute

Table of Contents

Table of Figures

Abstract

The goal of this project is to improve the data integration and administration center for the MATTERS Dashboard for the Massachusetts High Technology Council (MHTC), a pro-technology advocacy and lobbyist organization. Our system is comprised of two parts: the data integration pipeline manager and an administration center. Talend Open Studio is used as a core enabler for integrating data from a variety of online web sources. The Administration Center allows for future MHTC administrators to easily upload and view economy, talent, and ranking data, thus further impacting policies created in Massachusetts.

Acknowledgements

We would like to thank the following individuals, organizations, and institutions for their support and assistance throughout our project:

- **Massachusetts High Technology Council** for starting this project and partnering with Worcester Polytechnic Institute, thus providing us this great opportunity.
- **Jeff Manning** of MITRE for assisting us throughout the project and providing valuable feedback and resources.
- **Professor Elke Rundensteiner,** from WPI for her feedback and guidance throughout the project.
- **Caitlin Kuhlman, WPI,** for her support throughout the entirety of the project, specifically working with MHTC to provide direction and feedback for team.
- **Worcester Polytechnic Institute,** for providing us the opportunity to be able to work on this project and make this MQP possible.

Executive Summary

Big data comes in many different shapes and sizes, from many different sources all across the web. Analyzing and gathering this data is currently a complex and intriguing problem due to the variety of sources it comes from, each with a different way of representing it. Some of this data that is openly available on the web can be used to influence lawmakers and companies create policies within their states in order to improve and foster a period of economic and technological growth. With this in mind the Massachusetts High Technology Council (MHTC) set forth in creating MATTERS dashboard, the Massachusetts Technology, Talent, and Economic Reporting System. The MATTERS dashboard is a web-based tool to be used to explore a large collection of data from a variety of publicly available online sources. With it one could view trends in the economy of a given state; compare economic cost and talent competiveness across years, states and sectors; and evaluate the current potential of STEM graduates to meet the demands of the industry.

The goal of our project was to improve the data integration aspect and the administration center of the MATTERS website in order to further MHTC's goal to make Massachusetts one of the most competitive locations to operate a high tech business. In order to reach this goal we accomplished the following objectives:

- Integrate 20 new data sources into the MATTERS Dashboard
- Automate the process of updating data as much as possible
- Enable future users to easily add new data sources
- Develop the administration center into a working tool for non-technical users to maintain the system.
- Improve the look and feel of the Dashboard

The original pipeline architecture consisted of many custom components that needed to be recreated every time a new data source was added any could be very difficult for non-technical users to complete. In order to remedy this we investigated different Extract, Transformation, and Load (ETL) software. ETL software is excellent at manipulating, moving, and as the name suggests transforming data. We selected Talend Open Studio to be our software of choice out of all the others due the nice visual representation of the data flow and because it is open source. Talend allowed us to create pipelines to pull data from a variety of web sources; using both it's components and our own custom components. These pipelines were able to download new data from these sources and upload it to the database used to populate the information the MATTERS Dashboard. Since these pipeline require no coding experience it makes it easy for future MHTC administrators to be able to quickly and easily add new data sources.

While we were creating these pipelines we were also improving the MATTERS administration center. The administrated center we created is made up of a few different pages, each created to help future administrators maintain and improve the data on the MATTERS dashboard. The administration center consists of: a database explorer, for viewing the data without having to write code; a manual upload page, to upload files already in the appropriate format; a pipeline manager, for uploading the created Talend pipelines and executing them on the server; a scheduler tab for executing the pipelines at a future date for when new data becomes available; and a reports tab to view any errors that may occur in during the pipeline.

While we were creating these pipelines we experienced a lack of online documentation for Talend that would be essential for future administrators to easily create these pipelines. We developed a tutorial and documentation on how to create these pipelines. Finally, we also tested its utility out on a few users and we found that using it allowed the users to create a new pipeline easily and in an efficient manner.

With all of this in place MHTC will be able to continue to easily add new data sources and update existing sources in order to identify and advance pro-growth economic policies to help foster the growth of Massachusetts.

After completing the project we compiled a list of recommendations for future developers and for MHTC. The recommendations are as follows:

For future MATTERS developers:

- Develop a robust testing framework for future deployments of MATTERS. This is to ensure that for future deployments there is a set system in place to ensure the website will never crash during a deployment.
- Improve the Administrative Center by incorporating visual debugging tools. This will allow administrators to see more appropriately what is going wrong during the pipeline execution process. It would also allow them to see if there is any bad data in a much more visually appealing format than the current way of tracking errors.

For MHTC:

- Perform user studies on front end, Talend documentation, Administrative Center to ensure that all aspects of MATTERS are user friendly and easy to use. This will allow the system to grow and be properly maintained.

1 Introduction

Big data has become a difficult, yet intriguing problem in the world of computing. Data is readily available on the Internet for nearly anyone to use, both structured and unstructured. But how can this raw source of information be analyzed and used for the benefit of people? Interpreting data from across different sources and displaying it in a visually-appealing format may seem trivial, but the implementation of such a vast, expansive system can prove difficult to build.

The Massachusetts High Technology Council (MHTC) is an organization dedicated to making Massachusetts one of the "most competitive locations to operate a high tech business". Members include CEOs and senior executives representing the business and higher education and research sectors in Massachusetts is dedicated to making Massachusetts one of the "most competitive locations to operate a high tech business"("Mass High Technology Council Overview," 2014). As a part of this ongoing effort, MHTC brought together students and professors at WPI along with a number of council members to develop the Massachusetts' Technology, Talent, and Economy Report System, a web- based data analytics dashboard. Using this tool, data from across heterogeneous online sources could be accessed in one place, helping to "measure and to evaluate Massachusetts' current competitive position while providing policy makers with the information critical to developing public policy" ("Massachusetts' Technology, Talent and Economic Reporting System - MATTERS - Mass High Technology Council," 2014).

The MATTERS dashboard is defined by a process called Extract, Transform, and Load (ETL). Simply put, ETL extracts data from various sources, transforms the data to a format that is uniform for the system, and loads it into a data warehouse. Applying this process to MATTERS was necessary because the sources identified by MHTC varied from federal agencies to tech companies, all of which needed to be incorporated.

Since MATTERS was a fast-paced idea, a prototype was rapidly built order to provide a basis for the dashboard. Built upon the Spring MVC framework, the initial dashboard included publicly available data sets from six sources, user were able to perform simple side-by-side number comparison as well as more complex line and bar charts. The initial system required custom-built components to integrate the data sources. An administrative tool was created add new datasets by manually uploading Excel spreadsheets.

Very early on, the MQP team identified a number of steps to progress the system further. First, the existing system code needed to be reevaluated with object-oriented design principles in mind. Second, MHTC identified many other sources that would add value to the dashboard and allow for more insights into the economic competitiveness of the commonwealth. Third, an evaluation of existing open-source ETL tools must be made in order to determine if the development of this proprietary system continue. Fourth, the administrative tool must be completed to a state that will allow any end-user the ability to upload data sets from Excel spreadsheets without any unexpected errors or issues. Fourth, documentation and training materials were needed to allow a non-technical person to use the Administrative tool to add new sources.

2 Background and Related Work

The groundwork for the Massachusetts' Technology, Talent and Economy Reporting System (MATTERS) was initially developed with a group of students completing their Interactive Qualifying Project (IQP) in conjunction with other teams of WPI graduate students. The IQP team extensively researched the front-end visual aspect of the MATTERS dashboard, incorporating the use of colors, charts, and tables to represent data in an intuitive and quick analytics. The graduate students quickly developed the back-end of the system to allow for data to be extracted from a site, cleaned and parsed, then uploaded to a database.

2.1 System Architecture

The structure of this system, as it was currently received, can be found in Figure 1. This structure will be referred to as the data pipeline for the MATTERS dashboard. The data pipeline consists of 4 main components:

- Data Extractor/Wrapper
- Data Cleaner
- Data Transformer
- Database Loader

We will expand on each of these components as they are essentially to the function of the MATTERS dashboard.

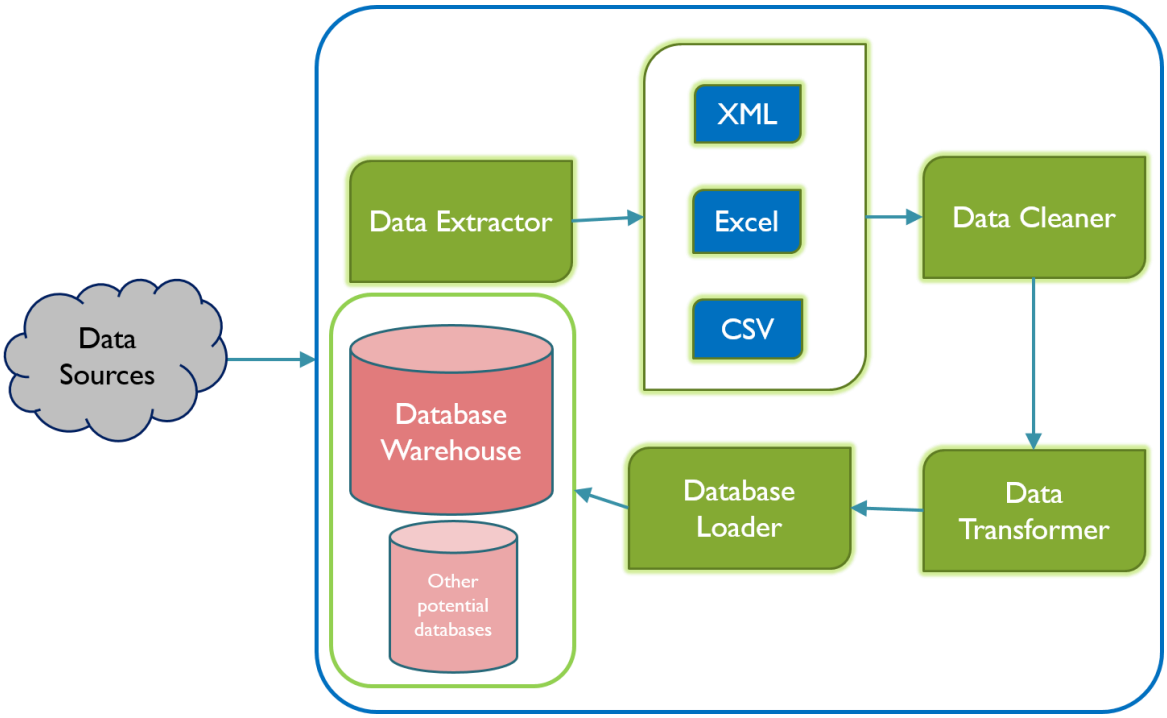


Figure 1 - Data Pipeline System Architecture for MATTERS Dashboard

2.1.1 Data Extractor/Wrapper

The data extractor component, or more commonly known as “wrappers,” is responsible for gathering the requested data from MHTC and entering it into the system. These can be as simple as accessing existing Application Programming Interfaces (APIs), or as complex as data scraping a website that contains the information. For example, the U.S. Energy Information Administration (EIA) has an available API to use. These wrappers are essential to the MATTERS dashboard as they provide a means to incorporate wanted outside data into the system.

These extracted sources are typically in the form of Excel files, but can also be CSV, XML, or JSON. As defined with the original development of the system, a base unit of data consists of a state, a year, and a metric value.

2.1.2 Data Cleaner

The data cleaners are responsible for taking the extracted data and ensuring its quality. If a state is misspelled due to human error, the data cleaner will find the error, use a spell checker and match the misspelled name to the closest actual match. If a state abbreviation contains extraneous characters, such as “M.A.” for MA (Massachusetts), the cleaner will also remove these unneeded characters.

2.1.3 Data Transformer

The data transformer, also known as a “parser,” is responsible for parsing the file into a structure that can then be used to upload the data to the database. Due to the guaranteed inconsistency between different files from different sources (it’s not reasonable to assume that every federal organization would format their data sheets the same), the parsers are highly specialized pieces of code that pertain to a specific source. Each source, in theory, will need its own parser.

On a general level, each parser identifies the necessary information in each file and forms tuples that consist of a year, a state and a metric value for the associated category. These currently need to be hand-written for each file, but there may be a potential solution to upload an Excel spreadsheet, select the desired rows or columns, and generate code.

2.1.4 Data Loader

The data loader is solely responsible for saving the many tuples created from a source file and storing them in the database, where they can later be used for visual analytics.

2.2 Requirement Elicitation

2.2.1 Administrative Tool

An important aspect of this project is the ability for end-users to complete maintenance on the MATTERS dashboard without the help of someone with a technical background. In order

to make the system intuitive enough for any user, it was decided that an administrative tool would be essential.

An administrative tool, in context of this project, would be a console that would be a visual curator integrated with admin capabilities. The following lists the ideal abilities the administrative tool would have. Admin capabilities can include, but are not limited to:

1. Manual upload of a file
    - As an end-user, I want the ability to upload a file, such as an Excel spreadsheet or a comma-separated values file, and have the data contained within the file be stored in the database and available for use on the MATTERS dashboard.
    - From there, an end-user can select the source (where it came from), the category of the file, as well as the metrics associated.
    - A visual representation of the file would appear, allowing for the end-user to manually select the cells that would correspond with the three data dimensions (state, year, metric) that are stored in the database, as well as specifying units for the metric(s).
    - The system would then be able to clean the data, generate an appropriate parser, and then transform the data.
    - Any errors would be reported to the user, and would give suggestions and recommendations to potentially fix any error.
    - If the file cannot be uploaded, the system would let the user know, and then prompt the user to re-upload the file after the user has done the proper cleaning and transforming.
    - During loading, if one tuple is found to be corrupt, the system would print out a message to the user and fix it on-the-fly.
2. Visual (or script) tool to build a data pipeline
    - An end-user would be able to point the administrative tool to an online source and be provided with options for how to extract the data (download file, select data displaying in an HTML table, etc.).
    - The file would then be passed through the same process that the manual upload goes through.
3. Visual inspection of the data warehouse
    - To ensure data integrity, a navigable visual representation of all sources, category hierarchy, and associated metrics currently stored in the database would be ideal.
    - After selecting a particular source file or category, an end-user should be able to:
        1. Get a data table of what is in the database
        2. See when the data was uploaded

3. See what file the data was extracted from
4. See where that file came from (which wrapper)

4. Visual tracking of data pipelines that are in place
    - A status menu would contain each pipeline that would show when the pipeline last ran, whether it was successful, and if not successful, an error report containing detail information as to why it was unsuccessful with possible resolutions.
    - An end-user may want to set up a scheduler for a certain pipeline, as an organization may update their data sets every month.

These are all ideal aspects that we'd like the administrative tool for the MATTERS dashboard to contain, but we recognize that many of these are much harder to implement than describe.

Talend Administration Center

Talend's paid-for products offer a built-in web-based administrative center for managing custom-built pipelines. Some services that are offered by Talend Administration Center are administrative settings, such as user and project management, a job conductor (similar to a scheduler), a dashboard for running real-time analysis of the current pipelines, as well as auditing capabilities.

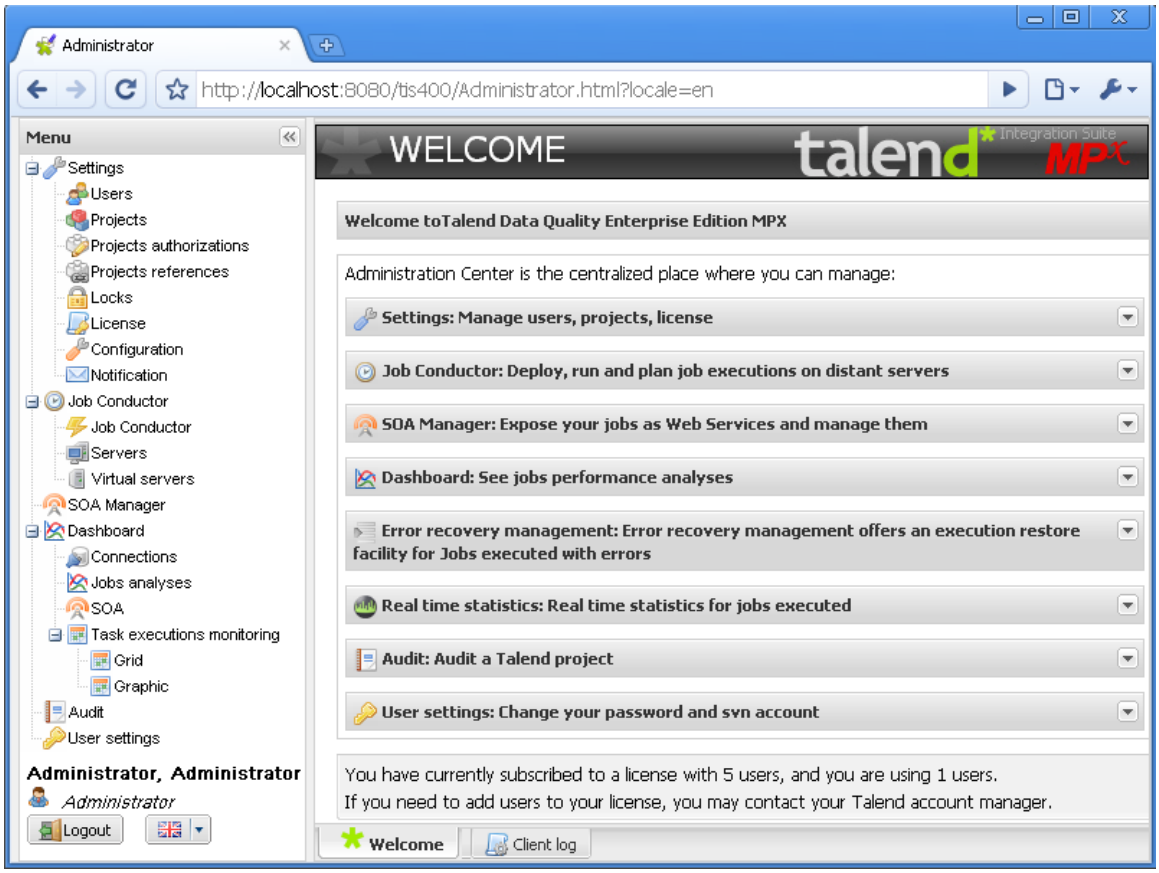


Figure 2 - Talend Administration Center Welcome Screen

User and Project Management

Many software engineering products use version control, which is easy to track changes between remote users. With this in mind, Talend created an easy interface for administrators to create a project (which may be the equivalent of a single pipeline, or a multitude of jobs) and grant access to certain users. For our simplified system, we ideally would only have a low-level implementation of this, where a user either has access to the entirety of the administration center, or no access at all. However, the ideas projected here are certainly important to keep in mind as this project progresses and moves forward.

Job Conductor

The Job Conductor module is responsible for scheduling certain tasks to run. A user (with proper access) would be able to generate a schedule, inputting the tasks required to run on the certain, and at what point in time.

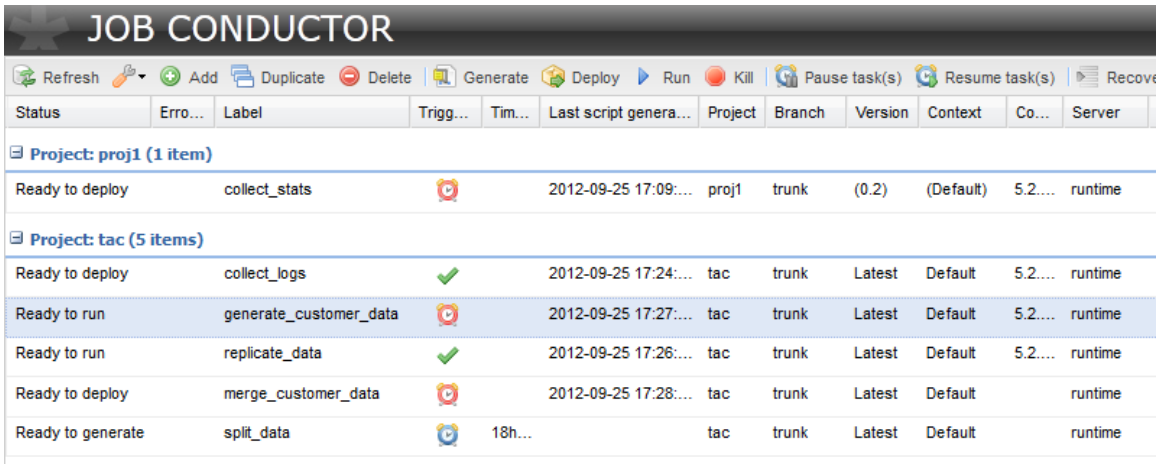


Figure 3 - Talend Administration Center Job Conductor View

This page gives some general statistics about the scheduled task, such as when it is scheduled to run, when it last ran, which branch the task may have been on, etc. This dashboard view provides an easy overview for any administrator to get a quick understanding of the status of the scheduling component of the pipelines. Our system will look to include something similar, which will give functionality to the user to create a schedule for certain uploaded pipelines to be run on the server and view the status of each pipeline.

Examples of Good Administrative Tools/Dashboards

Unlike products such as ETL tools, administrative dashboards have to be custom-built to tailor to the needs of the user. But, since our application is looking at a web-based dashboard, we can search for templates that are based off of HTML and other web technologies. A simple search for administrative dashboard will produce many templates that are open-source and can be used for nearly anything.

In terms of criteria, an admin dashboard should provide a level of management for any type of function the site performs. For instance, if we were running a community forum, we would want our admin dashboard to provide stats, the ability to manage users and their accounts, as well as moderation tools for the many different threads that the forum contains. We will analyze a few of the readily-available administrative dashboards to see the commonalities between them.

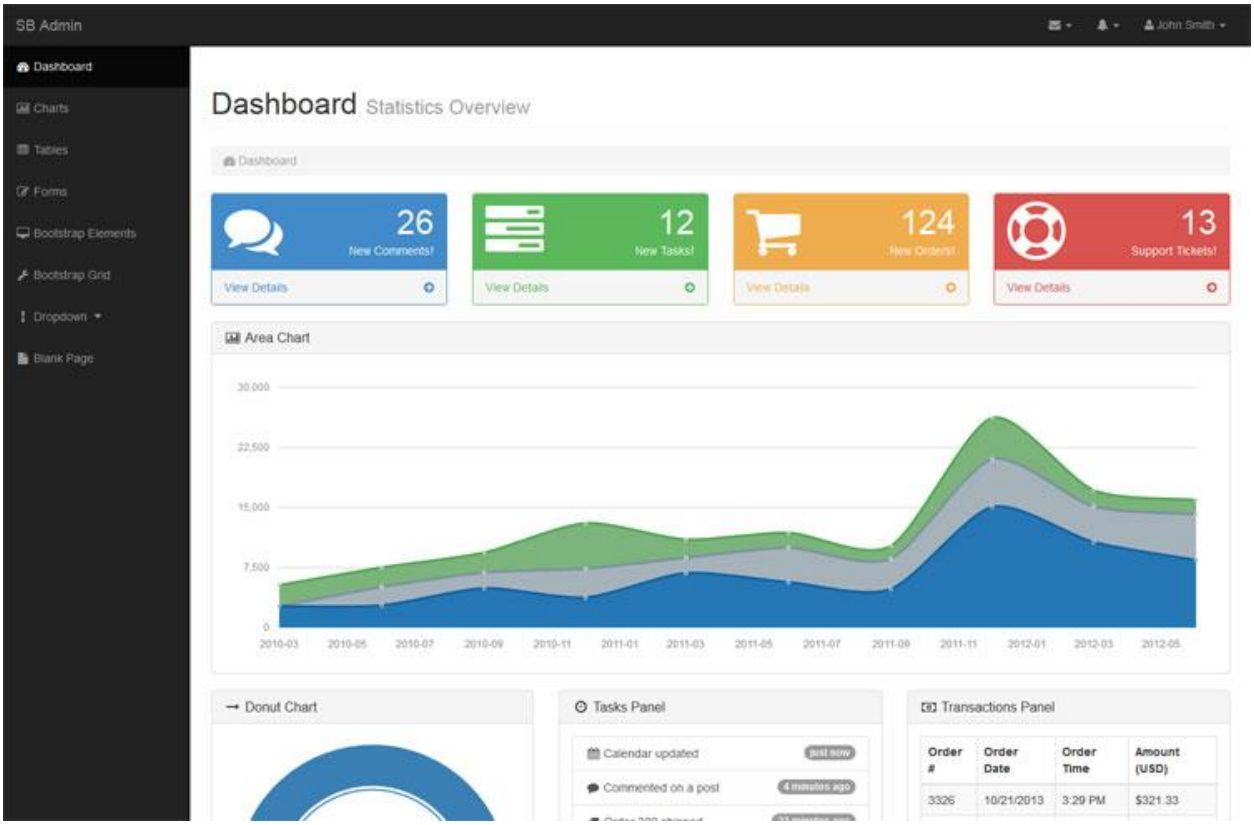


Figure 4 - SB Admin Dashboard Template (Bootstrap [25])

Figure 4 represents a well-developed and intuitive admin dashboard template, called SB Admin. Solely built upon HTML and Twitter's Bootstrap CSS framework, SB Admin also includes extra functionality, such as the ability to make charts, as seen above. To break down this dashboard, we see a common header at the top that contains the logged in user, as well as space for other icons to allow for quick access to other potential components, such as messages or reminders. Additionally, there is a navigation bar located on the left-hand side of the dashboard, to change different views and data of the dashboard.

The information in this template is laid out in a manner that any end-user who would be accessing the administrative dashboard could easily identify the different parts. SB Admin uses a minimalistic color approach, by only color-coding important categories (from the example, the categories would be "tasks", "comments", "orders", and "support tickets"). This simple approach reduces eye strain and information overload for the end-user.

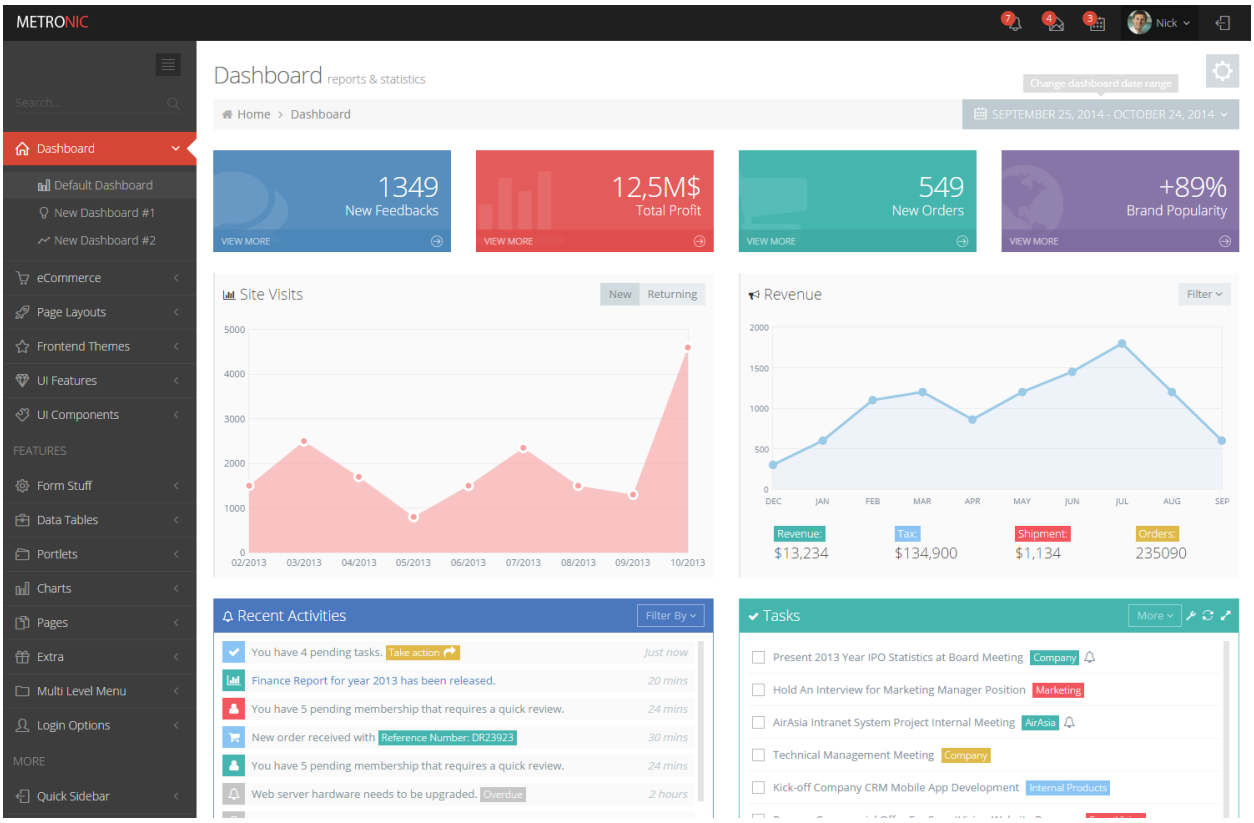


Figure 5 - Metronic Dashboard (Metronic [21])

Figure 5 is another example of an admin dashboard template, built upon the same frameworks as SB Admin. Doing a side-by-side comparison, it's easy to see the similarities between the two dashboards. Both contain a header that shows a branding, and the current user's potential inbox messages, reminders, as well as account settings. Each contain a navigation bar location on the left hand side of the page, as well as similarly designed color themes for categories.

Many administrative dashboard templates out there are built upon the same Bootstrap CSS framework, and boast the same functionality. At this point, it's a matter a personal preference, but nearly every administrative dashboard can accomplish the same task. In the end, these are merely visually, and do not provide that much functionality in terms of tools. Regardless, the visual aspect can greatly impact the efficiency of an end-user.

2.2.2 Data Pipelines

The other part of this project is for the end user to be able to continue to add and update data to ensure we are providing up to date information. However this needs to be done in a way that is simple and intuitive enough to be able to easily add data sources to the application. In order to do this there are a few capabilities required:

1. Small Overhead to Add a New Data Source

- Since not everyone will have a degree in computer science the software needs to be simple enough for the end user to be able to add a new data source to our database with having to do much coding or preferably any at all.
- It may also be necessary to add a new file every year for the same data source so the software will also need to be easily changed each year or be able to update itself in the best case.

2. Visual Representation of Data Flow
   - The end user should be able to view and create the pipeline from data source to database in a graphical interface as opposed to pure code.
3. Ability to Run Autonomously
   - The end user should be able to create a pipeline and then upload it to the admin panel where it will be able to run whenever scheduled to in order to get new data.

2.3 Extract, Transform, and Load (ETL)

Extract, Transform and Load (ETL) refers to a process in database/data warehouse usage that extracts data from data sources, transforms the data into a proper format for storing, or structure for querying and analysis, and loads it into the final target, typically a data warehouse. Typically, all three phases execute at the same time since data extraction takes time. While one piece of data is being loaded, the previous piece is being transformed so that there is always something ready to load into the database. Each step in the process is very important and an error in any step can cause the entire pipeline to fail. In today's market there are many tools that implement this process. Some free to use examples of these tools are *KNIME*, *Talend*, and *Clover ETL*. These tools implement the ETL architecture and allow users to develop their own pipelines in a graphical manner. This saves both time and money by eliminating the need for a software engineer when constructing or adding to a data warehouse. However, these tools must meet specific requirements in order to deliver optimal results to their users.

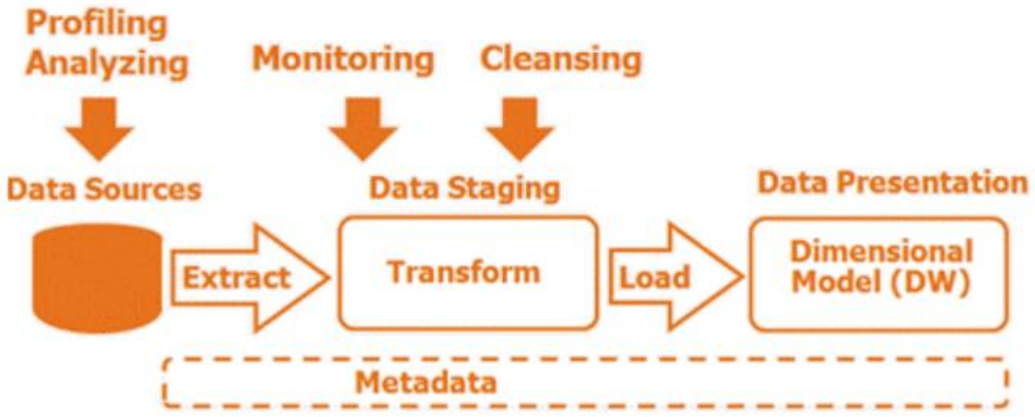


Figure 6 - Example of ETL Flow (Syncsort [11])

The first part of this process is the extraction. This is deemed one of the most challenging aspects of ETL since this sets the stage for the rest of the process. The goal of extraction is to convert the data into a single format that is ready for transformation. Once the data is extracted from the source the next step is transformation. During the transform phase a series of rules or business logic is applied to the data to derive the data to be loaded into the database. This brings us to the final phase: the load phase. During this phase the data is loaded into the end target, typically a data warehouse. For this phase the data needs to match the schema of the table it will be loaded into otherwise the operation will fail.

2.3.1 Open Source ETLs

We began to investigate a number of open source ETLs that could be incorporated within the MATTERS system. We looked into using Clover, Talend and Pentaho; all of which were recommended to us from some of the staff at MITRE. We spent numerous hours using each program, read documentation and also viewed help forums for each in order to decide which software would best fit our needs. After comparing all of the positives and negatives for each, we came to the conclusion that Talend would be the best option for our team in developing pipelines.

CloverETL

CloverETL was fairly easy to use and understand. However, the free software version could not achieve what we were looking for in a free to use software. CloverETL offers a number of versions of software including Community, Designer, Server, and Cluster. We looked into the Community version first because it didn't cost anything, but we also downloaded the Designer for a 45-day free trial to see if it was something that we wanted to find funds for.

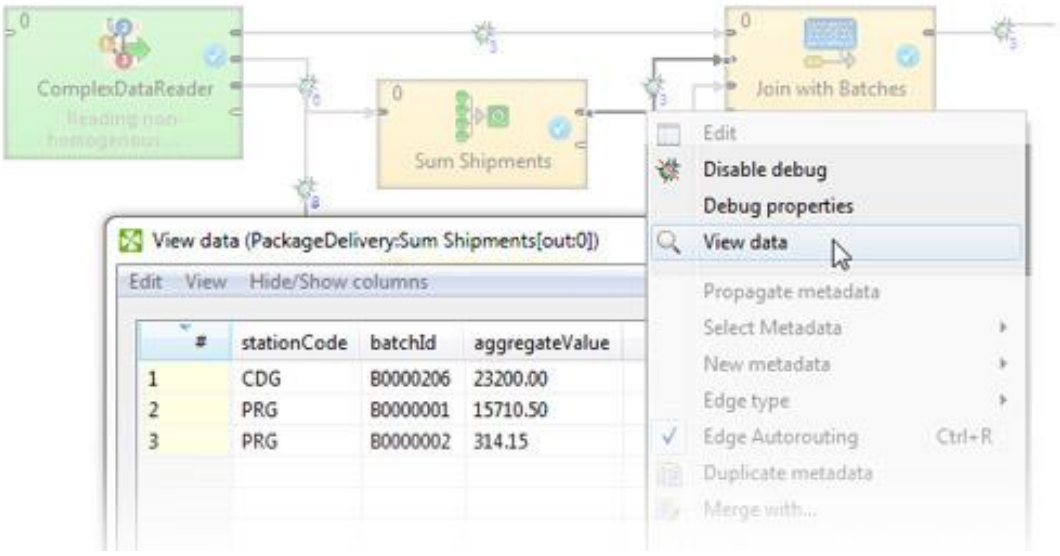


Figure 7 - Example of CloverETL (CloverETL [22])

Pentaho Data Integration

Pentaho Data Integration was found by a few searches for ETL's. Similar to CloverETL, Pentaho offered a free trial of their software, which we downloaded to experiment with. The software seemed to cover our basic needs but seeing how there was some other software that achieved the same results but were free to use, we decided to not look into Pentaho any further.

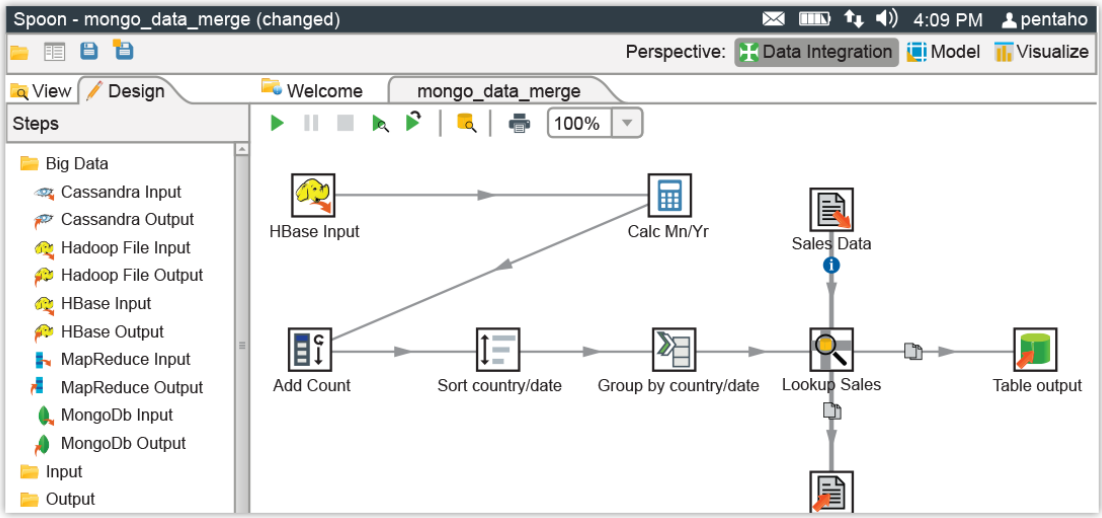


Figure 8 - Example of Pentaho Data Integration (Pentaho [23])

Talend

Talend is an open source software vendor (Manning [3]). They provide many products to users including those that focus on data integration, data management, enterprise application integration and big data. Their product Talend Open Studio for Big Data combines big data technologies into a unified open source environment simplifying the loading, extraction, transformation, and processing of large and diverse data sets. This software provides a graphical interface for users to set up what it calls jobs. These jobs are used to extract data from various sources, transform it and then it can load the transformed data into a variety of outputs. This software also generates java code for any job that you create.

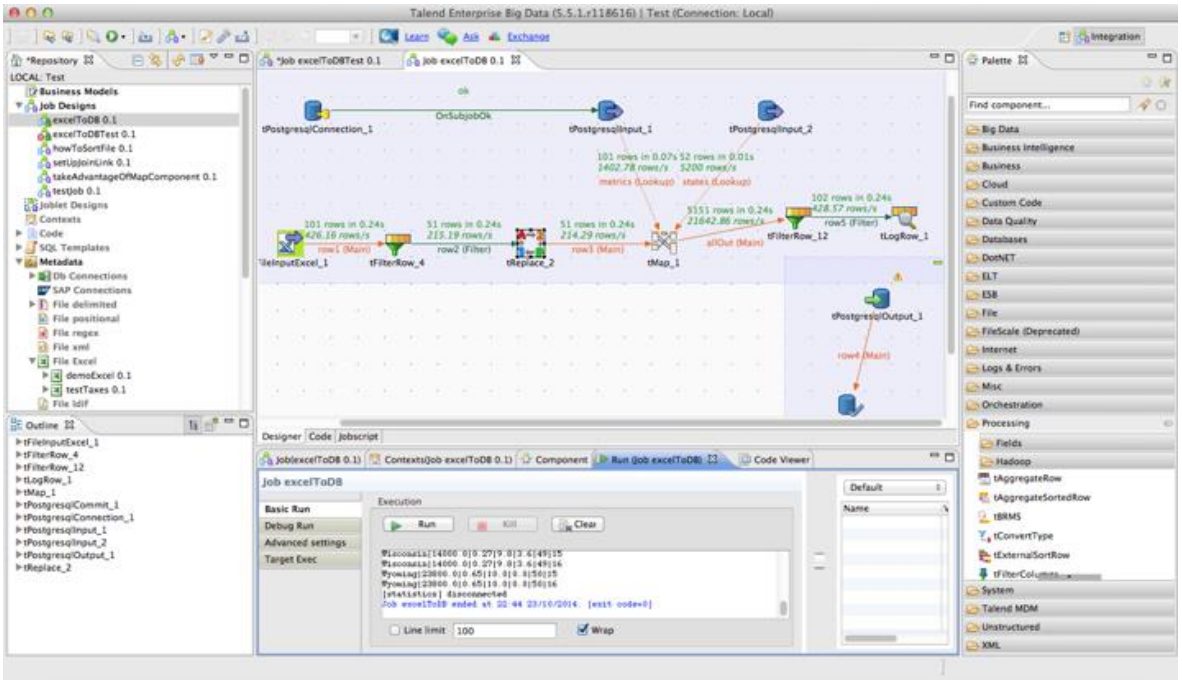


Figure 9 - Talend Open Studio (Talend [8])

KNIME

KNIME was another recommendation by MITRE (Manning [3]), as well as a few other people in various fields of work. At first glance, we liked that KNIME was open-source. We dove into testing out the features it presented and was overall happy with the software. We used this in comparison with Talend to make a decision on which software we wanted to move forward with.

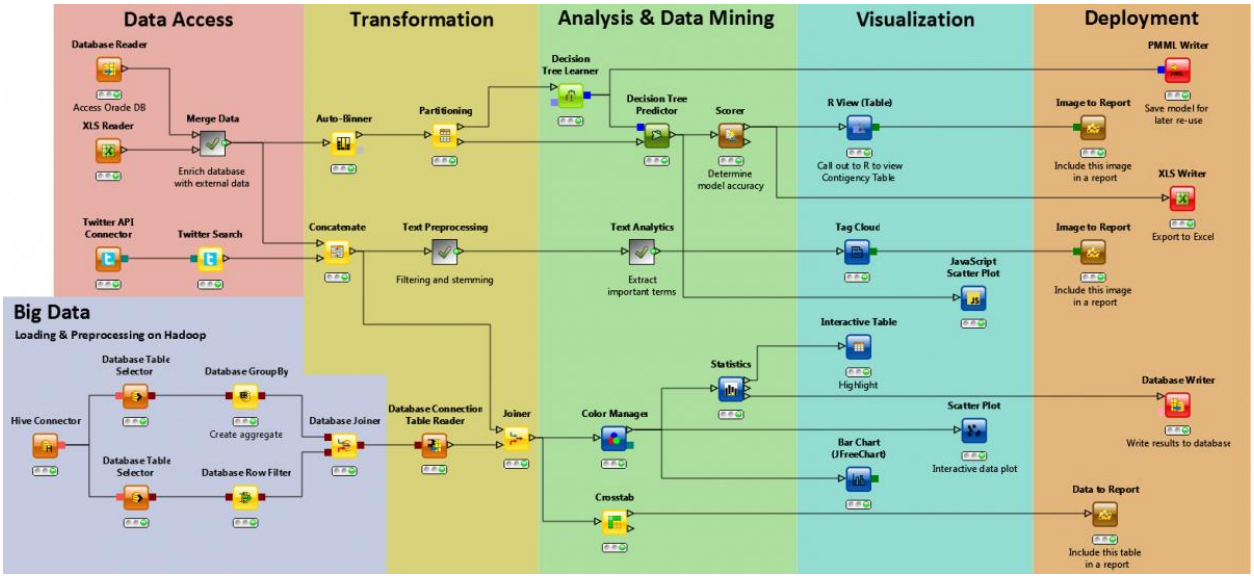


Figure 10 - Example of KNIME (KNIME [24])

3 Methodology

Our main goal for our system is to enable a future MHTC admin to be able add a new data source to their system easily and without any prior coding experience, as well as be able to visualize the data in the database. We will be accomplishing these tasks through the use of an Administration Center, which is part of the main MHTC MATTERS site, and Talend Open Studio.

Using Talend, a user can create pipelines in order to take a data source and transform the file they receive from said source so that it will be in an acceptable format for entry into the database. Talend Open Studio provides a graphical-user interface for creating these pipelines, which provides an easier approach than having to manually code these pipelines.

In order to gather data from certain sites, specific code functions were written to download these files, which are called data wrappers. Although Talend has functionality for downloading files directly, some sites use specific APIs that require a bit more complex code. Since these data wrappers were written in Java, we were able to incorporate them directly into the Talend pipelines as custom code snippets.

Once the pipeline is complete, the user can then login to the Administration Center on the MATTERS website and upload the pipeline on the Pipeline Manager tab. The user can then use the Scheduler to run the pipeline immediately or at a set date. Once the pipeline has finished, the admin can use the Reports tab to view any messages the pipeline sent and will be able to tell if it completed successfully. The admin can also upload data through the Manual Upload tab if the data in a file is already in the specified format. If this is a new metric to be added the admin can create new metrics in the database from the Administration Center instead of having to run a query, and they can also use the Database Explorer to check for data consistency or see if any data is already there that they may try to enter. The following sections will give an in depth overview of each component of our system starting with Talend Open Studio, moving into the data wrappers and ending with the MATTERS Administration Center.

3.1 Talend

Talend is used by a number of big businesses in order to represent their data in a way that is quick and effective (Talend.com). We will be using Talend to create pipelines that will be able to grab the data from various websites and push it through a cleaner and eventually into our database, where it can be accessed from our website in a number of ways including graphs, heat maps and of course in data tables.

Our ultimate goal is to incorporate Talend into the MATTERS Administration Center and not have the end user worry about it. However, the issue with this may be that the URL's we are pulling the data from might change in the future, the data could be in a different format, or a new data source might want to be added. Having these limitations makes us believe that Talend will have to be used concurrently with our system. With this in mind, clear and proper documentation will be provided as well as a number of template jobs that the end user can manipulate in order to gather information from a new web address.

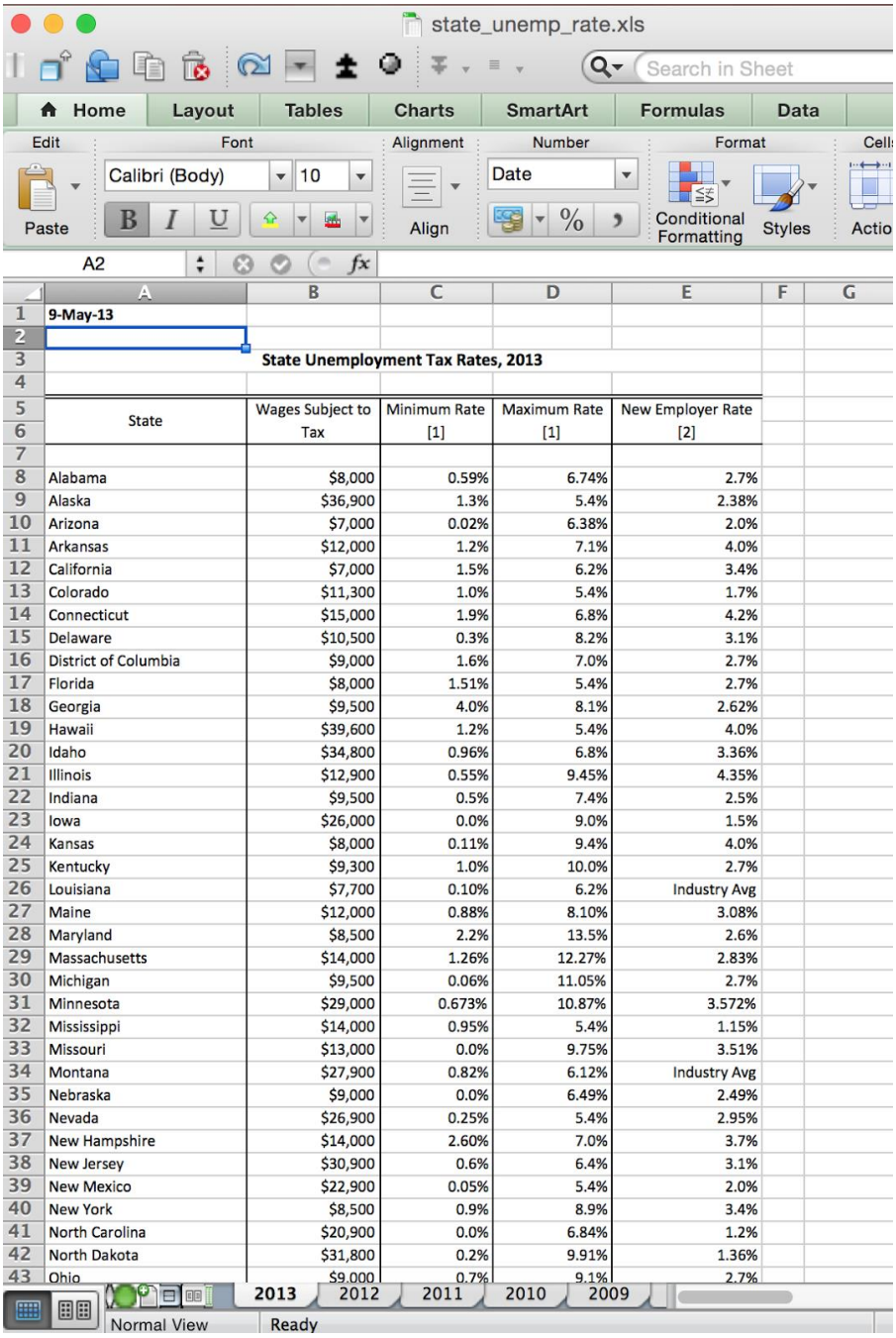
3.1.1 Selection of Talend

After reviewing all of the ETL software listed previously, we came to the conclusion that Talend would fit our needs best. We compared each of the ETL software to each other in order to determine which one would best suit our needs. During this time we found that Talend seemed to be used more widely amongst companies thus providing more documentation than the other ETLs could. From our own research, we also found that Talend seemed much easier to learn immediately than the others; however, we did notice that for more advanced features, there was somewhat of a steeper learning curve than others. Even with this factor, all of them seemed to be very similar to us during our initial research.

To help us reach a conclusion on a specific ETL to use, our contact at MITRE, Jeff Manning, was able to do a high level analysis of each of the software in question. Talend contains all the core capabilities we were searching for, supports a large number of data extract formats, and was the ETL cited most by small and medium enterprises (Manning). After reading his analysis of each existing ETL and rating Talend as the most viable ETL for MHTC and our team to use, we selected Talend as our ETL of choice. The full analysis can be found in Appendix B.

3.1.2 Learning of Talend

Once Talend had been chosen, we delved into it in order to start creating pipelines, a way to move the data from a file, clean and transform it, and insert it into the database. The first thing we did was to search for any tutorials on Talend that would help us get started. We came across a few, however, the documentation was not as complete as we originally thought and the tutorials were not as detailed as we had hoped. Due to this, we decided to switch gears towards jumping in and just attempting to get a pipeline up and running through trial and error. We began with what we thought would be a simple Excel file, but would cover most of the things we would need to do in future pipelines. It had multiple metrics, a header that would need to be removed and either blank spaces or missing information for certain metrics, as seen in Figure 11.



State	Wages Subject to Tax	Minimum Rate [1]	Maximum Rate [1]	New Employer Rate [2]
Alabama	$8,000	0.59%	6.74%	2.7%
Alaska	$36,900	1.3%	5.4%	2.38%
Arizona	$7,000	0.02%	6.38%	2.0%
Arkansas	$12,000	1.2%	7.1%	4.0%
California	$7,000	1.5%	6.2%	3.4%
Colorado	$11,300	1.0%	5.4%	1.7%
Connecticut	$15,000	1.9%	6.8%	4.2%
Delaware	$10,500	0.3%	8.2%	3.1%
District of Columbia	$9,000	1.6%	7.0%	2.7%
Florida	$8,000	1.51%	5.4%	2.7%
Georgia	$9,500	4.0%	8.1%	2.62%
Hawaii	$39,600	1.2%	5.4%	4.0%
Idaho	$34,800	0.96%	6.8%	3.36%
Illinois	$12,900	0.55%	9.45%	4.35%
Indiana	$9,500	0.5%	7.4%	2.5%
Iowa	$26,000	0.0%	9.0%	1.5%
Kansas	$8,000	0.11%	9.4%	4.0%
Kentucky	$9,300	1.0%	10.0%	2.7%
Louisiana	$7,700	0.10%	6.2%	Industry Avg
Maine	$12,000	0.88%	8.10%	3.08%
Maryland	$8,500	2.2%	13.5%	2.6%
Massachusetts	$14,000	1.26%	12.27%	2.83%
Michigan	$9,500	0.06%	11.05%	2.7%
Minnesota	$29,000	0.673%	10.87%	3.572%
Mississippi	$14,000	0.95%	5.4%	1.15%
Missouri	$13,000	0.0%	9.75%	3.51%
Montana	$27,900	0.82%	6.12%	Industry Avg
Nebraska	$9,000	0.0%	6.49%	2.49%
Nevada	$26,900	0.25%	5.4%	2.95%
New Hampshire	$14,000	2.60%	7.0%	3.7%
New Jersey	$30,900	0.6%	6.4%	3.1%
New Mexico	$22,900	0.05%	5.4%	2.0%
New York	$8,500	0.9%	8.9%	3.4%
North Carolina	$20,900	0.0%	6.84%	1.2%
North Dakota	$31,800	0.2%	9.91%	1.36%
Ohio	$9,000	0.7%	9.1%	2.7%

Figure 11 - Example of data in Excel format

Once we had the file, we began attempting to transform it in Talend. We initially kept the file stored locally for Talend to access due to issues that we encountered when attempting to download a file from the Internet using Talend. Talend's visual editor made learning through example much easier than trying to use the incomplete documentation online. Figure 12 shows the editor for Talend Open Studio along with the pipeline created from the file in Figure 11.

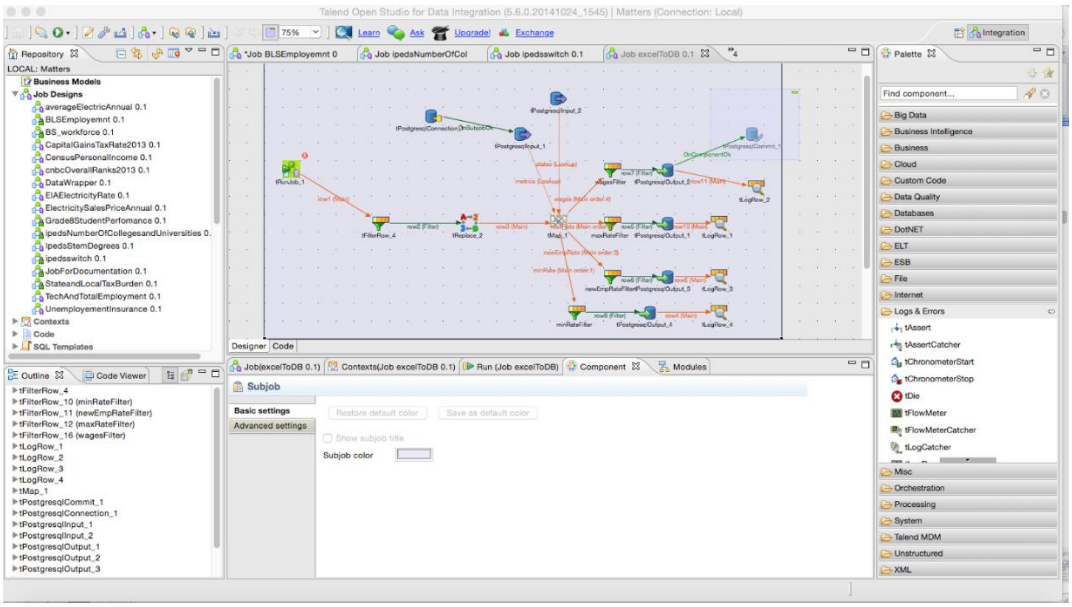


Figure 12 - Talend Open Studio pipeline (Talend [8])

Talend uses components, which can be added to the job designer in the center from the Palette window on the right. Each component has a different function. Some of the main features are getting data, processing the data, analyzing the data, and sending it to some output. Talend moves the data from one component to another by row. That is, each row is individually passed from component to component, for that component to perform its function upon the row. During the learning process we found that our main components for transforming the data would be the tReplace, tFilterRow, and tMap components.

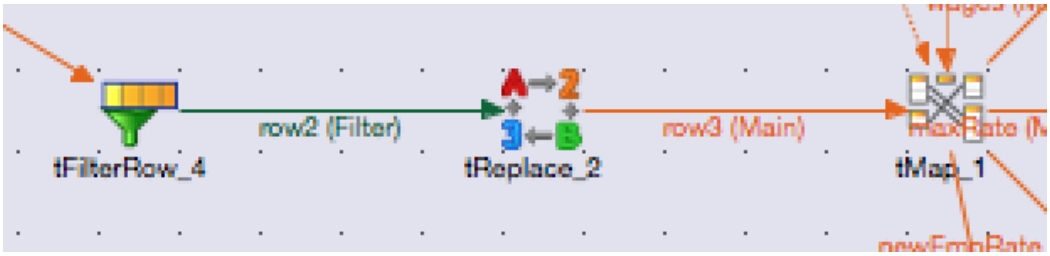


Figure 13 - Talend cleaning functions (Talend [8])

These three components were able to complete most of the transforming that was needed, with the tMap being the most useful component. The tFilterRow component was used to filter out any rows that were not necessary to be inserted into the database. The tReplace was then used to remove any bad data such as having words in columns for numbers or removing any percent signs or other punctuation. The tMap allows you to make different connections by moving the variables from the input and any variables created inside the tMap over to different output tables that can then be sent to various components. Figure 4 shows the editor for the tMap with the input tables on the left, variables in the center and the output tables on the right.

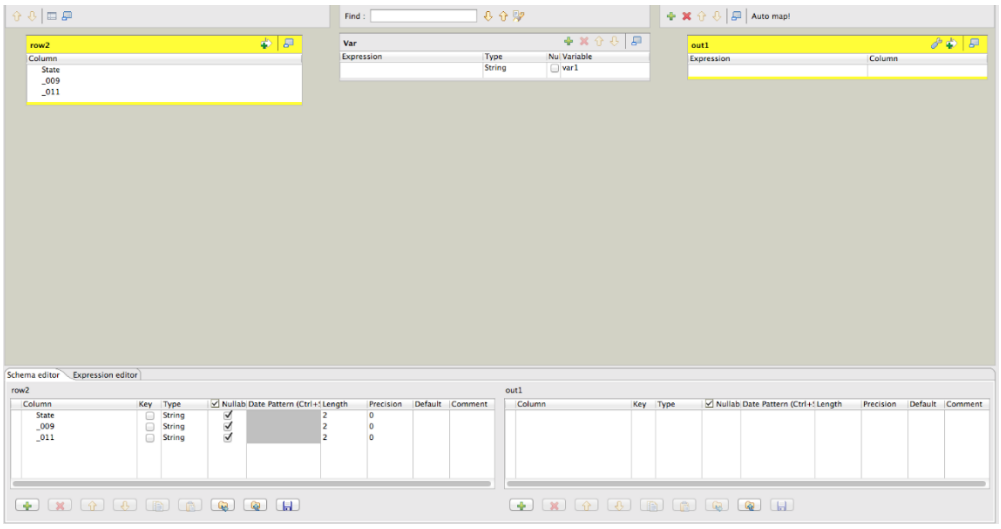


Figure 14 - Talend tMap component (Talend [8])

By using the tMap, we were able to take multiple metrics from one file and separate them into different outputs to be inserted into the database. We were also able to manipulate the data in order to get each row into our format of a state ID, a metric ID, the metric value, and the year. From here, we used Talend's database components in order to insert each row into our database.

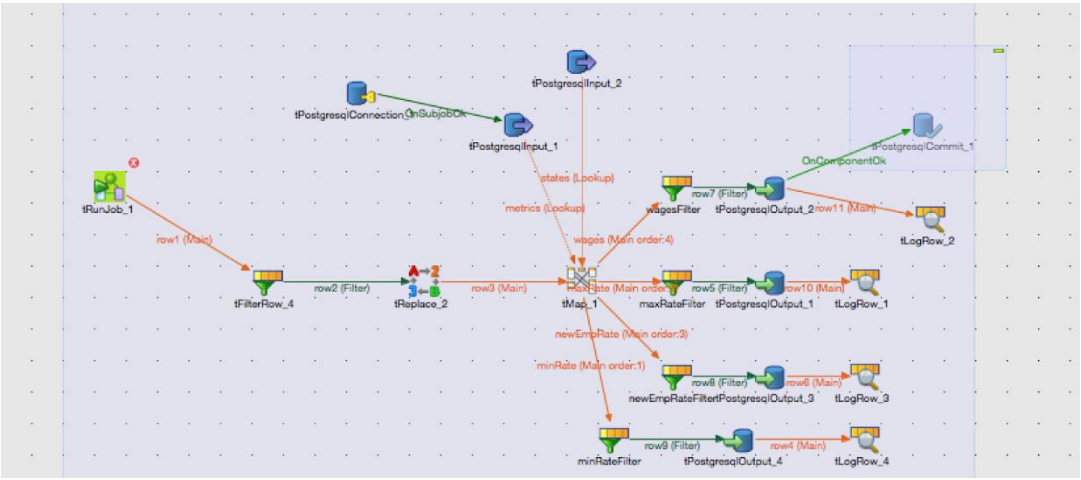


Figure 15 - Complete Talend pipeline job (Talend [8])

Once we had finished creating this pipeline, we felt as though we had a good understanding of how Talend worked and could move onto creating pipelines for the sources that MHTC had requested.

At the same time, we also had a few more goals in order to improve the pipelines. The first was to figure out how to run the pipeline on a server. Next, we wanted to add the ability to log and view any errors or status updates that may occur during the pipeline. We also needed a way to incorporate the download of a file from a website through Talend. Finally, we wanted to

create documentation in order to help future users learn the process in a much easier and more efficient manner.

The first step we took was to add logging features to all of our pipelines. Talend has its own components that are used to create log messages, as well as a component that catches any log messages created. The two components we use to create log messages are the tWarn component and the tDie component. The tWarn simply creates a message when it is triggered. The tDie component however ends the pipeline when it is triggered in addition to creating a log message. These components are triggered when a certain component is either successful or fails.

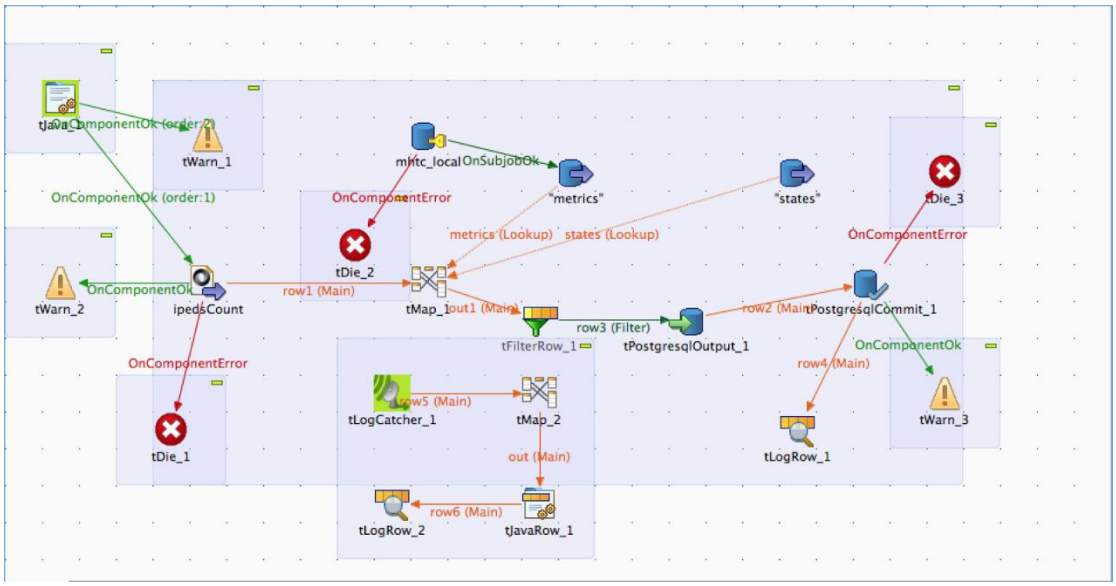


Figure 16 - Talend pipeline with logging features (Talend [8])

Once all the tWarn and tDie components were in place and all our log messages were set, we then needed a way to send them to the Administration Center for viewing by an MHTC admin. To implement this, we created a subjob in our pipeline that would catch all the logs and then send them to the Administration Center. Talend's tLogCatcher component listens for any log messages created for the duration of the pipeline. This will get all the information for each log and then we used a tMap in order to filter out any extraneous information. From there, the data is sent to a tJavaRow component, which allows the user to run custom Java code on individual rows. The code we are running is a simple HTTP POST method.

This is done inside of a routine, which is a way to create a Java class inside of Talend and be able to call functions using the custom code components. This routine takes in the URL to post to, and all the information from the tMap. We originally tried to use the HTTP component inside of Talend, however, during this time, the current release of Talend had a bug that affected the HTTP component. More information on the logging process can be found in Appendix B.

```java
public class SendLogs {

    /**
     * sendLogsFromFile
     * @param webAddress the url you are sending the log messages to
     * @param fileLocation the location of the file the logs were written to
     * @throws IOException
     *
     * This function takes in an url and a file location and then reads the file, takes the data and puts it into a
     * http post message that is then sent to the specified url.
     */


    public static void sendLogs(String address, String message, Date moment, int priority, String job, String origin, int code)
        URL url = new URL(address);
        Map<String,Object> params = new LinkedHashMap<String, Object>();
        params.put("moment", moment.toString());
        params.put("message", message);
        params.put("priority", priority);
        params.put("job", job);
        params.put("origin", origin);
        params.put("code", code);

        StringBuilder postData = new StringBuilder();
        for (Map.Entry<String,Object> param : params.entrySet()) {
            if (postData.length() != 0){
                postData.append('&');
            }
            postData.append(URLEncoder.encode(param.getKey(), "UTF-8"));
            postData.append('=');
            postData.append(URLEncoder.encode(String.valueOf(param.getValue()), "UTF-8"));
        }
        byte[] postDataBytes = postData.toString().getBytes("UTF-8");

        HttpURLConnection conn = (HttpURLConnection)url.openConnection();
        conn.setRequestMethod("POST");
        conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
        conn.setRequestProperty("Content-Length", String.valueOf(postDataBytes.length));
        conn.setDoOutput(true);
        conn.getOutputStream().write(postDataBytes);

        Reader in = new BufferedReader(new InputStreamReader(conn.getInputStream(), "UTF-8"));
        for (int c; (c = in.read()) >= 0; System.out.print((char)c));
    }
```

Figure 17 - Example custom code for HTTP POST within Talend

After we had finished with add logging capabilities to pipelines, we then moved onto creating a way to download the file from an external website rather than getting the data from a local file. When we originally tried to download a file from the Internet, we encountered various encoding errors with the files, however with the knowledge of routines gained from creating the logs we now had a new way to access the files.

Using routines, we were able to add the data wrappers that were originally written in Java to grab the sources, as well as new wrappers to get new sources and some generic downloaders for scrapping data from an HTML table. Using these wrappers, we were able to grab files from the web, save them locally where the job is stored, and then access them to run the pipeline to transform the data. More information on data wrappers in Talend can be found in Appendix B.

Once we no longer had to worry about having the files with the pipeline to run the pipeline, we moved onto the major task of how to run these pipelines. This task became very simple with the addition of data wrappers as routines in the pipelines. Talend allows you to build the job as an executable, which you can run through the command line or any scripting language. With this we were able to upload pipelines to the server through the Administration Center, which could then later be accessed by other parts of the Administration Center, such as the Scheduler.

Finally, we created documentation for the use of Talend. This was to ensure that future admins of MHTC would have an easy time adding new sources through the use of Talend. The documentation is mainly a tutorial on how to build a pipeline, detailing each of its parts. It also

contains more information on logging, routines, and information on each specific pipeline that was created. This documentation can be found in Appendix B.

3.2 Data Wrappers

3.2.1 IPEDS

IPEDS Datacenter is a web service that provides extraction of data from the National Center for Education Statistics. A user must navigate through the website, picking Institutions and interested variables. After a few steps, the user will be able to reach the download page. The process is time-consuming, and the interface is not user-friendly.

After each query, the IPEDS Datacenter webpage will send a POST request to a query builder. The query builder is intelligent enough to prevent non-legitimate access with cookie and request validation. However, this can be circumvented with some patterns in the POST requests.

We used a Firefox plugin called Firebug (the Net function – explained in the appendix section) and Tamper Data to monitor the flow of data between the client and server. Next, we will simulate the web browser by sending a correct POST request to the appropriate webpage location. Here are some key properties of the request we must consider:

- For the POST request
  - POST requests for every page change (the browser goes to another page from pressing Continue/Submit button)
  - POST requests have the *__VIEWSTATE* parameter (a must)
  - POST requests have the *__VIEWSTATEGENERATOR* parameter (there are some cases that the request doesn't contain this)
  - POST requests must contain data from our selection.
- Request headers
  - The headers must have the referrer parameter. A correct parameter value for each request is necessary.
  - The headers should contain standard web browser headers (Host, User-Agent, Connection, etc.)
  - Set of cookie data accumulated from the querying steps. This is very important.

An engineer maintaining this wrapper should pull those request parameters by manually navigating through the IPEDS Datacenter website. Every time the browser changes the page, the engineer should examine the POST request send at the moment of button press or page change. The engineer will need to extract EVERY request header. Cookies can be temporarily ignored thanks to a Java library, JSoup (JSoup [10]).

At the end, IPEDS Datacenter will render a "Download" page which will give you the option to download a CSV file. Please note that, it is zipped.

Wrapper Design and Implementation

Here are the steps to download the data manually:

1. Select “DOWNLOAD CUSTOM DATA FILES”
2. Use final release data and press Continue
3. Select Institutions By Variables and then select Browse/Search Variables

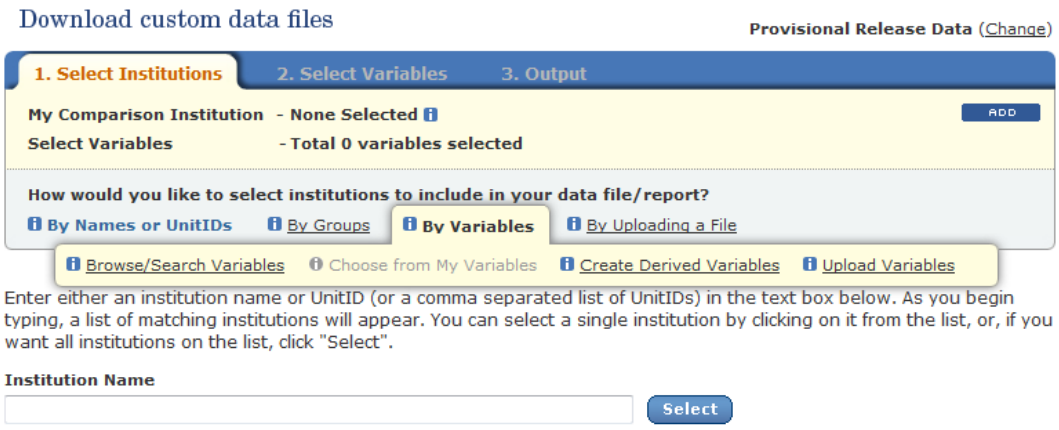


Figure 18 - IPEDS Custom Data File download

4. Select “Frequently used/Derived variables”
5. Select Institutions
6. Select Year and in List of variables select “State Abbreviation” and “Degree granting status”
7. Press Continue
8. Press Continue
9. Click State abbreviation to select the state. (Can select multiple states but in end would get multiple files)
10. Click Save
11. Click “Degree granting status” and select “Degree granting”
12. Click Save
13. Press Submit
14. Click Continue
15. In Select variables, select Year and then select Completions
16. In Select Qualifying Variables
    a. Select CIP Code- 2010 Classification: select CIP two digit codes (11,14,15, 27, 40)
    b. In Award Level Code select all from Associate degree to “Doctor’s degree”, Save.
17. In Select from List of Variables select “Grand Total”
18. Click Continue
19. Save the file

unitid	institution name	year	C2012_A.First or Second Major	C2012_A.C	CipTitle	C2012_A.Award Lev	C2012_A.Grand total	IDX_C
164447	American International	2012	First major	'11'	Computer	Bachelor's degree	0	-2
164447	American International	2012	First major	'26'	Biological	Bachelor's degree	16	-2
164447	American International	2012	First major	'27'	Mathema	Bachelor's degree	0	-2
164447	American International	2012	First major	'40'	Physical S	Bachelor's degree	1	-2
164447	American International	2012	Second major	'26'	Biological	Bachelor's degree	0	-2
164447	American International	2012	Second major	'27'	Mathema	Bachelor's degree	0	-2
164465	Amherst College	2012	First major	'11'	Computer	Bachelor's degree	11	-2
164465	Amherst College	2012	First major	'26'	Biological	Bachelor's degree	53	-2
164465	Amherst College	2012	First major	'27'	Mathema	Bachelor's degree	10	-2
164465	Amherst College	2012	First major	'40'	Physical S	Bachelor's degree	21	-2
164465	Amherst College	2012	Second major	'11'	Computer	Bachelor's degree	2	-2
164465	Amherst College	2012	Second major	'26'	Biological	Bachelor's degree	7	-2
164465	Amherst College	2012	Second major	'27'	Mathema	Bachelor's degree	12	-2
164465	Amherst College	2012	Second major	'40'	Physical S	Bachelor's degree	12	-2

Figure 19 - Data Format for IPEDS wrapper

From line 32 to 51, we will do step 1-2 above – Logging in the IPEDS Datacenter system. On line 31 to 48, we have customized the POST request with headers and data parameters to send to http://nces.ed.gov/ipeds/datacenter/login.aspx. As a result, IPEDS Datacenter will return some session cookie data that we need to save (line 51.) Please note that every POST data and header parameters must be completed in order to simulate a real browser request. Otherwise, IPEDS Datacenter will reject your request and send back a starting HTML page – which means you had done something wrong. A tip: Write the HTML content into a separate file and track those files step-by-step to make sure that the process goes smoothly.

In the POST request, parameters such as *__VIEWSTATE* and *__EVENTVALIDATION* are unchanged, while *ibtnLoginLevelOne.x* is not. The engineer should investigate multiple times to determine the constants and focus on making them right. The non-constants do not seem to affect the overall result. The easiest way to do this is to imitate the request exactly the same as those of the browser.

This wrapper covers 6 crucial steps – with the last step involve downloading the zipped file. For each step, the engineer needs to simulate the browser's request just like explained above. However, be extremely careful about COOKIES.

IPEDS Datacenter website tells the browser to save cookie data differently on each query page, so we must build the cookie data step-by-step. On line 72, there is a cookieMerge operation that merges the response cookie assignment from Step 2 to those of Step 1. The same principle is applied to each step. The engineers should also track the cookies for each step, validate to see if the cookie returned by the JSoup library is sufficient and correct (JSoup.Response.cookies() method.)

Finally, step 6, IPEDS Datacenter will render a page that contains the link to download a zipped CSV file, JSoup also supports jQuery-style element. The engineer should investigate the source of the download page to pick out the most suitable CSS selector, and grab the correct download link. After that, use JSoup to download the file, *don't forget the cookies and the request headers*!

.ignoreContentType(true) – line 174 is also necessary for JSoup to download the zip file. By default, JSoup will only parse HTML/XML content. That method will tells JSoup to ignore and save the content as bytes. Finally, use FileOutputStream to write the JSoup's response body in bytes from the last step. After that, we can extract the downloaded zip file with the UnZip class.

3.2.2 HTML Table Scraper

This is a universal wrapper for HTML-based tabular data. Each HTML table in a web page will have a unique element identifier, which differentiate itself from other elements in the DOM tree. The HTML Table Scraper utilizes this property to extract the data in those tables.

Wrapper Design and Implementation

This wrapper makes use of a Java library called JSoup. JSoup grants us the ability to perform a CSS-selector query to access the HTML elements and its contents. The wrapper will treat the first <tr> / <th> as a table header, and subsequent rows as data. Because JSoup queries every <tr> element in the table, the wrapper can then extract the text content from each <td> elements belong to the current <tr> element. This information will be written to a Microsoft Excel file and saved to a temporary folder of the web server.

Example:

We have this page: http://www.cnbc.com/id/100824779, which contains a table tag:

```
<table class="csvData data table-sorter" cellspacing="0">

          ...

</table>
```

Since the CSS selector of this table is .csvData.data.table-sorter, we will need to call the wrapper with the URL, the CSS-selector, the Excel file path we want to save to, and a list of column index we want to ignore (start from zero).

```
WebTableWrapper.download("http://www.cnbc.com/id/100824779",
".csvData.data.table-sorter", "tmp/cnbc-13-overall-ranks.xls", Arrays.asList(-
1));
```

3.2.3 Data.gov API Wrapper

Data.gov provides a wide variety of data for users to extract information from. The website supports SQL-query to extract customized data and provides original document or spreadsheet files available to download. The wrapper itself is very simple. It gives the developer an ability to download a JSON file generated from a SQL query that was sent to the data.gov website. The wrapper also offers a "smart" download function that intuitively looks for a Download button, extracts the URL from it and use a HTTP Downloader to retrieve the file.

Wrapper Design and Implementation

The wrapper offers two functions to extract data from data.gov. Generally, they are an extended class of HTTP Downloader class. For the query, it's important to understand the table structure from the webpage in order to write a precise query to retrieve the data in JSON format.

```
// Download using smart downloader, the wrapper looks for a download button,
grab the link and retrieve the file

data_gov_downloader.smartDownload("https://inventory.data.gov/dataset/032e19b4
-5a90-41dc-83ff-6e4cd234f565/resource/38625c3d-5388-4c16-a30f-d105432553a4",
"tmp/ipeds.csv");
```

```
// Download using query: Select data from database 38625c3d-5388-4c16-a30f-
d105432553a4 (IPEDS number of college and universities)

data_gov_downloader.queryDownload("SELECT \"STABBR\", COUNT(*) from
\"38625c3d-5388-4c16-a30f-d105432553a4\" GROUP BY \"STABBR\"",
"tmp/ipeds_count.json");
```

3.3 Administrative Center

The Administrative Center was created to allow an MHTC employee the ease of doing multiple tasks without needing the knowledge of programming. Within the Administrative center, the admin can explore data that is contained within the database, upload data files, manage pipelines, schedule events and view reports. One of our goals is to make the administrative center easy to use and mobile-friendly. We have used many third-party libraries in order to achieve this. Some technologies used for this section were Bootstrap, jQuery, DataTables library, (DataTables [9]) Quartz Scheduler library (Quartz [16]), and the jQuery Lightbox extension (JQuery [26]).

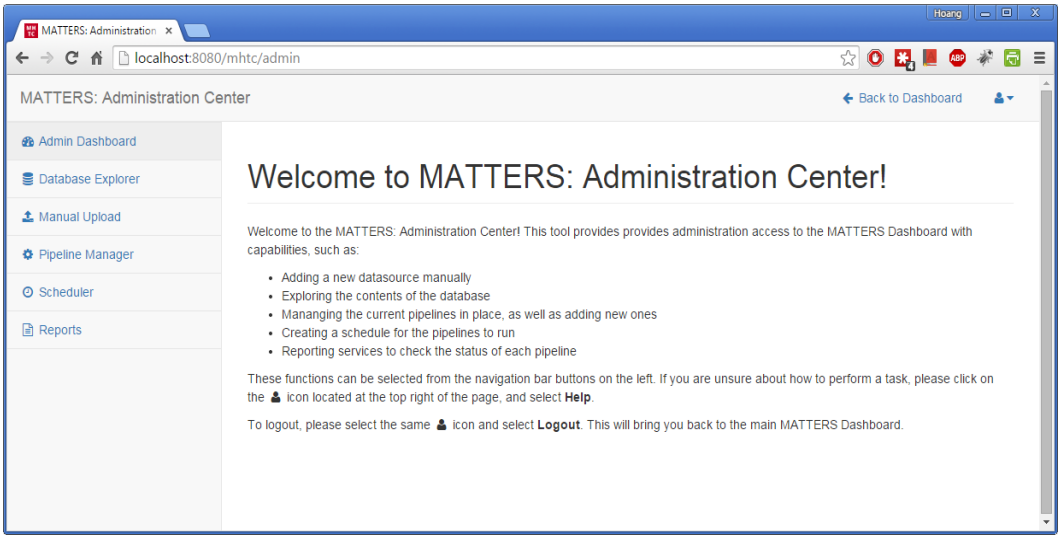


Figure 20 - MATTERS Administration Center Welcome Page

3.3.1 Database Explorer

The Database Explorer allows the user to view the current data in the database. These data are pulled from the "statistics" table in the PostgreSQL (Postgres [17]) database. They also have the option of selecting metrics to filter the data. This process allows the user to make SQL queries without actually needing to know SQL. After executing a pipeline or manually uploading a unified format data file, MHTC administrators can use this functionality to verify that everything is imported into the database properly.

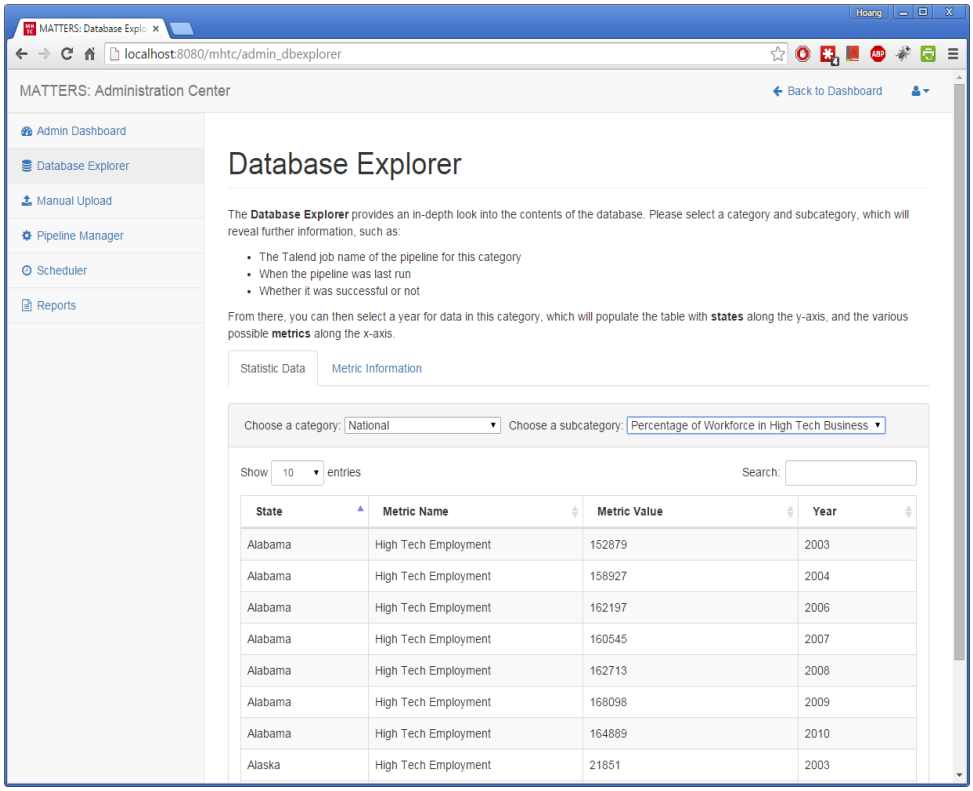


Figure 21 - MATTERS Administration Center Database Explorer

3.3.2 Manual Upload

Users can also upload a data source that cannot be automatically downloaded and imported by the system. The system will parse the uploaded Excel file, consistent with the Unified Format model, and import them into the database, available to be viewed in the *Database Explorer* page. We created this function in order to help parsing irregular data sources where data wrapper and Talend pipelines cannot be built.

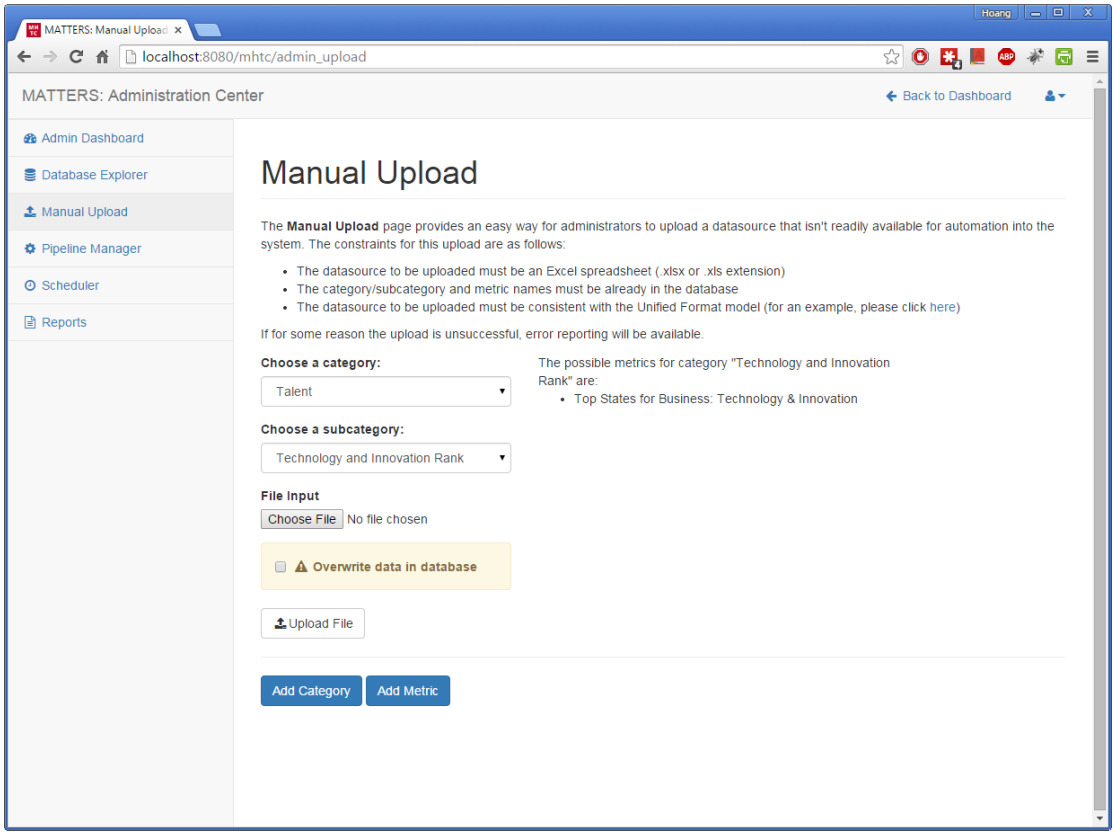


Figure 22 - MATTERS Administration Center Manual Upload

3.3.3 Pipeline Manager

This page allows administrators to upload compressed Talend pipelines to the system. The system will automatically detect the Operating System it is running on, and determine which executable file to run. For example, Windows Tomcat server can only run .bat script while UNIX-based systems should run shell .sh scripts. These pipelines can be executed to automatically retrieve external resources, parse and import them into the database. The pipelines are listed in a table and can be configured to run manually. After these pipelines are executed, they will communicate with the web server to log their actions. We created this functionality to help MHTC administrators easily manage pipeline in a scientific manners. They will also have a total control of the Talend pipelines loaded into the system, and decide when to execute, update or remove the pipeline at their own will.

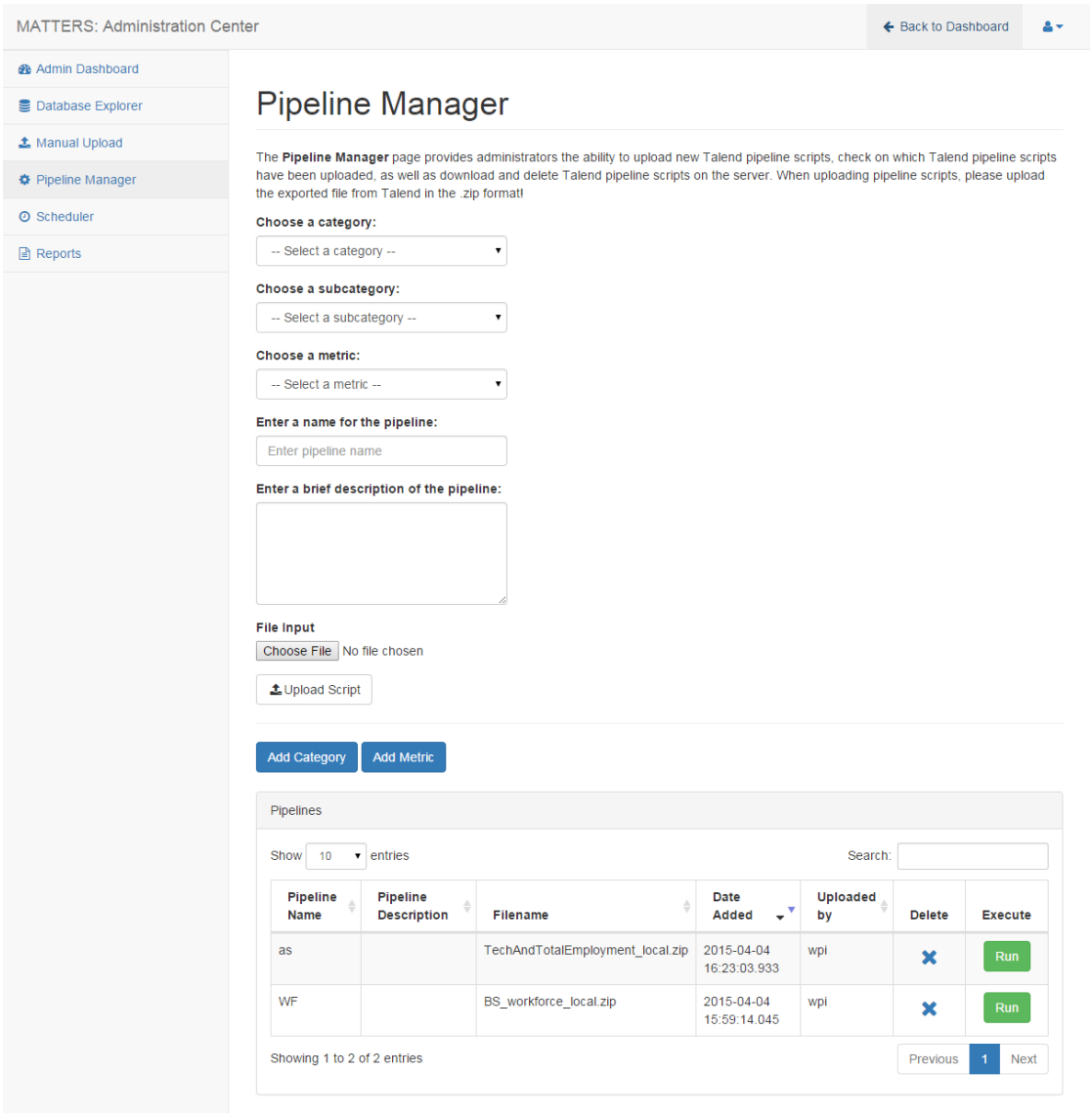


Figure 23 - MATTERS Administration Center Pipeline Manager

3.3.4 Scheduler

The Scheduler allows the administrators to schedule a Talend pipeline to run in a future date. The functionality allows one-time schedule or cron-job schedule, which runs repetitively based on the administrator's choice. The Scheduler makes use of Java Quartz Scheduler 2.2.1 library (Quartz [16]). We created this function in order to allow MHTC administrators to effectively organize the flow of data sources in the system. With this tool, they have a complete control of the process. Specifically, they can make the pipelines to execute periodically without human intervention to grab latest data from source sites.

The Quartz Scheduler is multi-threaded. Therefore, it has the ability to execute multiple pipeline jobs at a same time. The Scheduler also boots up along with the Tomcat Web server, which allows non-interrupted job scheduling and making.

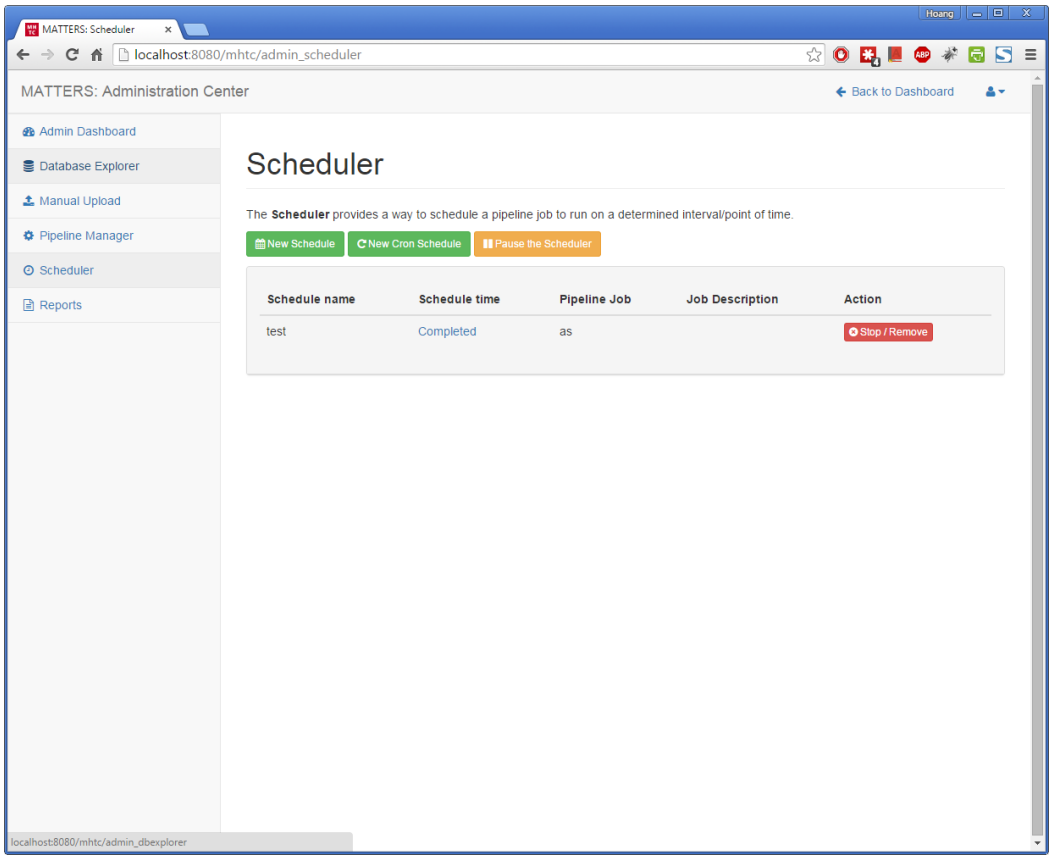


Figure 24 - MATTERS Administration Center Scheduler

3.3.5 Reports

Administrators can view the recent logging information sent by executed Talend pipelines. It offers various logging capabilities, such as error and notice logging. This is very helpful for the administrator to know if the pipelines are running successfully or not, and to know which components encounter critical error that needs immediate attention. Not many administrators have technical knowledge to connect to the Tomcat Web server using SSH and view the log file written by Tomcat itself. We created this interface so that Talend pipeline can redirect its output and debugging flow into our PostgreSQL database instead of Tomcat Standard Output Stream.

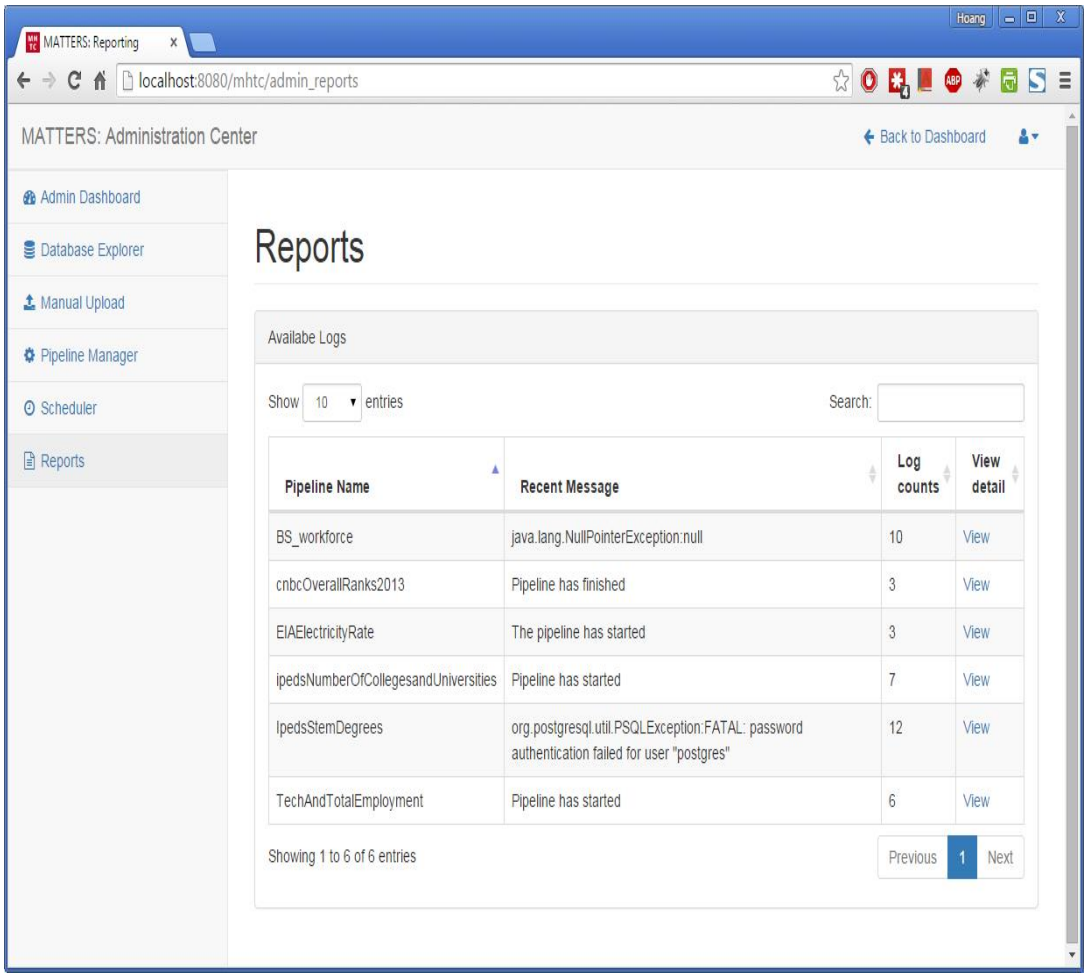


Figure 25 - MATTERS Administration Center Reports

The log is also very detailed for experienced system administrator who knows a little bit of coding to identify the cause of the problem and propose a solution to fix that. Some error codes are in Java Exception format. They will help the developers who wrote the wrapper integrated in the pipeline find the error in the integration process:

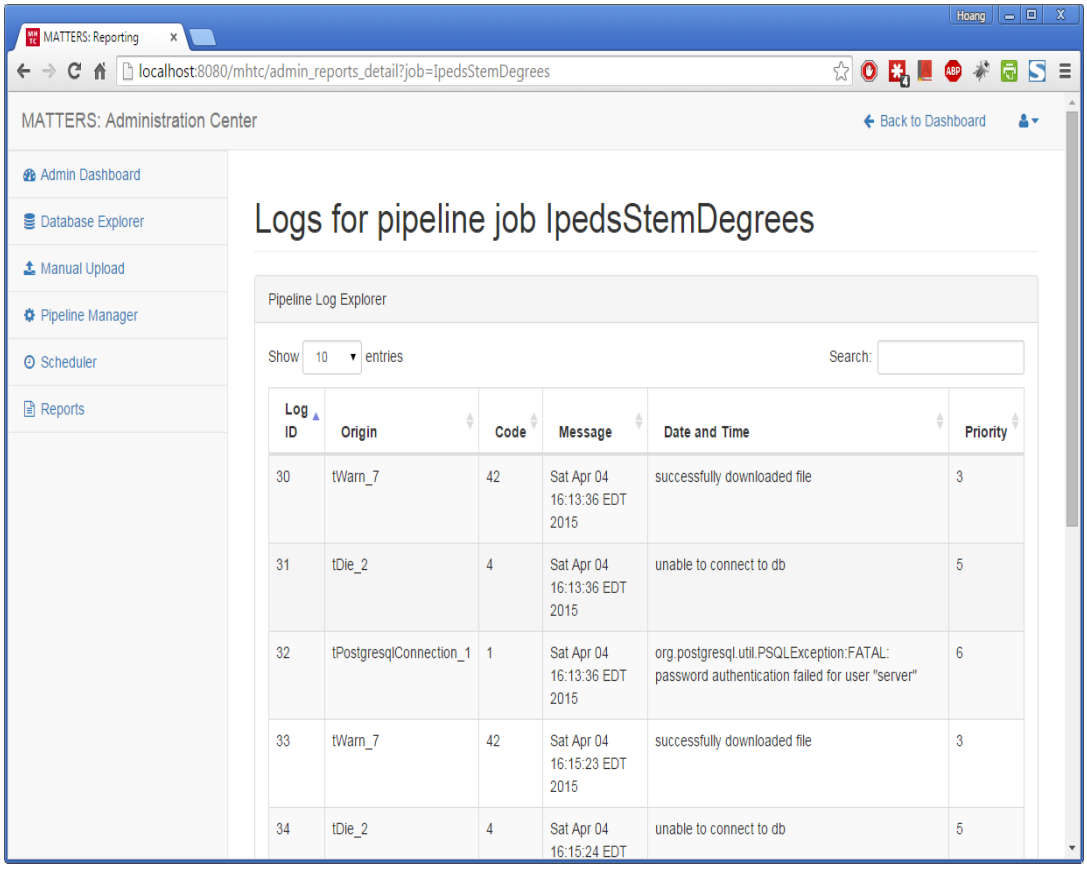


Figure 26 - Detailed logs for a specific pipeline

4 Testing, Results, and Evaluation

In order to evaluate our newly designed system, we were able to have our code base benchmarked by MITRE, which included an in-depth report on areas such as robustness, performance, security, transferability, and changeability. Following this feedback received, we entered an iterative process to refactor our code. In conjunction with the report from MITRE, CodePro Analytix (CodePro [6]), a static code analysis, was used to find further violations per standard coding rule sets. Following refactoring, we were able to resubmit our code to MITRE to provide a comparison from the previous report. Due to time constraints, only manual testing could be conducted on the code base following a process of deploying the system to a development server before pushing it into production.

4.1 Software Quality Assurance Evaluation Code Wash Report – MITRE

Following completion of all new major functionality to the project, our code was submitted to MITRE to perform a performance evaluation. Based upon Common Vulnerabilities and Exposures (CVE) and Common Weakness Enumeration (CWE) currently maintained by MITRE, the evaluation provided a wide range of coverage for our code base.

The CAST Application Intelligence Platform (AIP) Analysis is an off-the-shelf static analysis tool used by MITRE to evaluate software systems before they are put into production. Initial analysis by this tool, before any refactoring had been completed, characterized the project as “small” with a Total Quality Indicator (TQI) of 2.59 on a scale of 4. For commercial applications, a high score would be above 3.5. Due to the scale of the project and its academic background, the focus of the report was not on the various different indicators used, but rather the learning experience for identifying common code vulnerabilities and violations to help us, as software engineers, improve our quality of work.

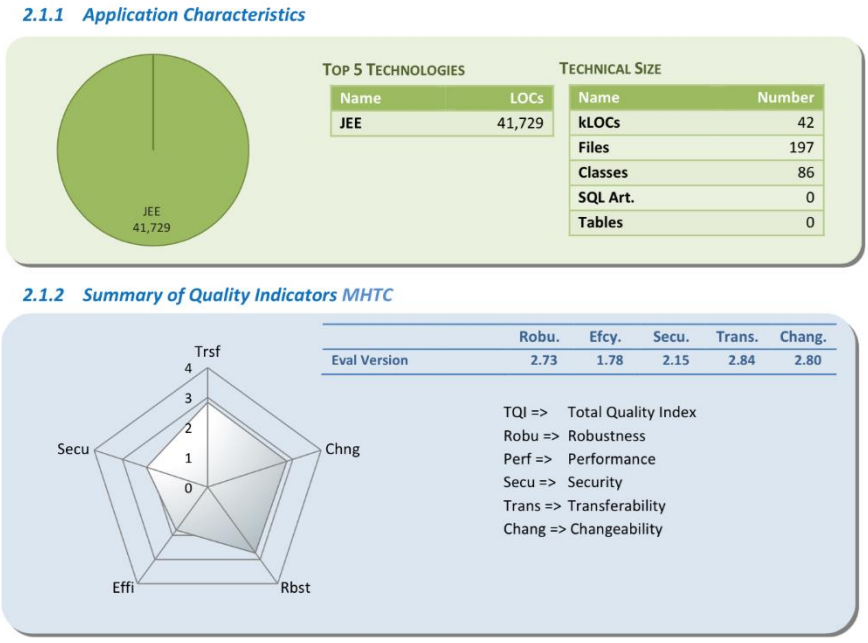


Figure 27 - High-Level Analysis of MHTC project

Figure 27 supplied additional indicators regarding our system, breaking it down into robustness, performance, security, transferability, and changeability. Our system contained nearly 42,000 lines of code, with roughly 2.35 critical violations per 1,000 lines of code. Other risk drivers included expensive calls within loops, which negatively impacted efficiency scores, and lack of multi-layered data access. 31 CWEs were identified within the application.

To provide a bit more detail regarding risk mitigation, there were no critical violations for all 48 of our high-complexity functions, with object level dependencies and OS and platform independence well maintained.

In a presentation of the results by MITRE, some violations were marked as high-impact areas:

- Efficiency
- Use of Style sheets
- Documentation
- Automated documentation
- Security
- Error and Exception handling

For more information regarding specific details of the report, please check Appendix C.

4.2 CodePro Analytix

In order to supplement the code review from MITRE with further analysis tools, we chose to use CodePro Analytix auditing with a custom-built rule set from a previous class taken at WPI (CS 4233, Object Oriented Analysis and Design). On initial run, without doing any refactoring to the system, we received 39 High violations, 730 Medium violations, and 974 Low violations.

These violations are purely static, and are able to point to the exact line of offending code. Usually, fixes are suggested by the actual violation, so turnaround time is extremely low. Many of these violations were completed after refactoring.

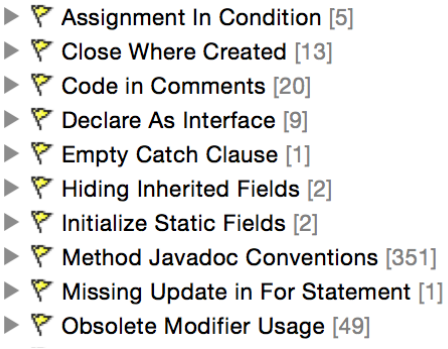


Figure 28 - Example CodePro violations found within MHTC project

4.3 Refactoring

After receiving initial feedback on the quality and robustness of the system, we planned to address the most critical violations (those that impacted the score highly) as well as those that would be a quick turnaround. In addition to addressing code violations, there was also a fundamental design flaw with the system, specifically with data persistence.

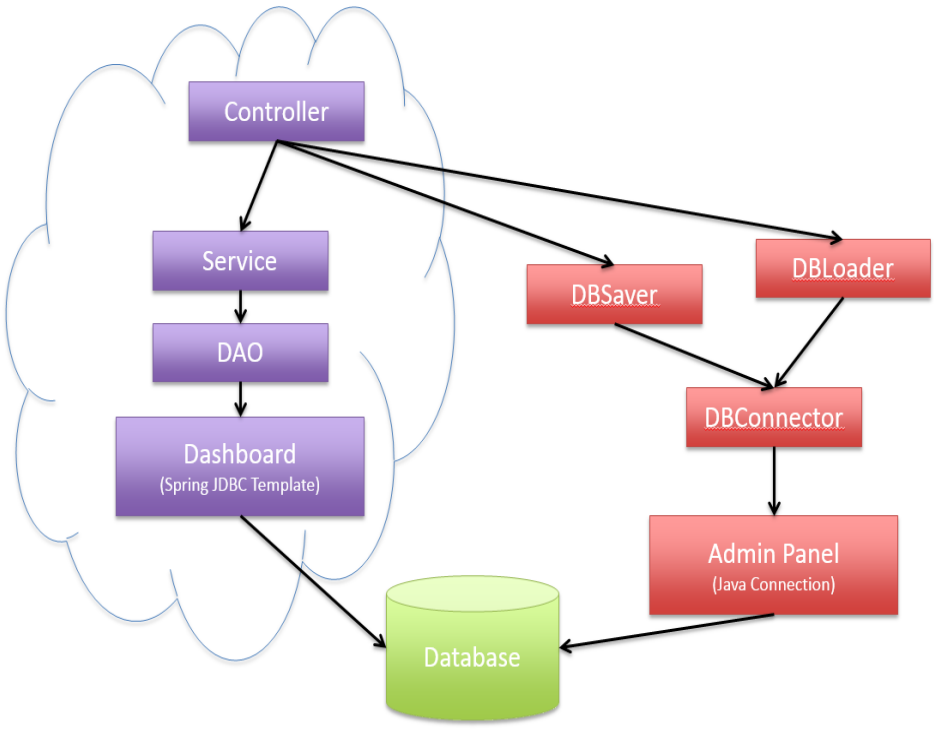


Figure 29 - Database connectivity before refactoring

The project is based upon the Spring MVC framework, with a PostgreSQL database. Spring contains functionality and base classes for database interaction through Java database connectivity (JDBC). These classes were in use by the front-end, specifically used for grabbing the statistic data from the database.

When the Administration Center was first developed, the previous version used standard Java Connection objects to connect with the database, followed by other JDBC standards. Although this was not initially a problem, it was determined that it would make sense for one framework to be used to access the database, as to mitigate security issues and other bugs. Figure 29 shows a rough outline of the database connectivity before refactoring.

We decided to use Spring's implementation of JDBC, as it provided more abstractions and management rather than using standard JDBC. Following guidelines from Spring's documentation, we were able to convert all instances of database connectivity not using Spring's built-in framework. This was able to eliminate some of the violations found by both CodePro Analytix and MITRE's evaluation.

For a little more detail on refactoring, we were able to develop certain layers of abstraction for the project. Layers included:

- Models
- Data Access Objects (DAO)
- Services

4.3.1 Models

Models represent the most basic object in the Spring MVC platform. They are essentially the containers of information for particular parts of the entire project. In the case of refactoring, many classes have fields directly corresponding to columns in a particular table in the database.

4.3.2 Data Access Objects (DAO)

Data Access Objects (DAO) contains the actual methods responsible for creating and making queries on the database. Most DAOs contain methods such as save(), get(), and delete(). These are all standard functions that someone may want for updating a table in the database, or for retrieving information from a database.

A DAO is created for each particular model. For instance, a Metric model would have a MetricDAO, which would be responsible for populating a list of Metrics, or deleting a certain Metric from the database.

4.3.3 Services

Services contain business logic responsible before a database query is run. Although in our case, many of the methods found in a Service simple call the corresponding DAO method, Services are useful for doing certain checks (i.e. make sure a Metric with a certain ID doesn't already exist, or make sure a Metric object has an ID of 0 if it is going to be saved to the database).

Once this structure was in place, each of the Service implementations had to be initialized by a Controller. Overall, this structure was able to reduce the amount of required code within the methods in a Controller and centralized all database code to one particular package.

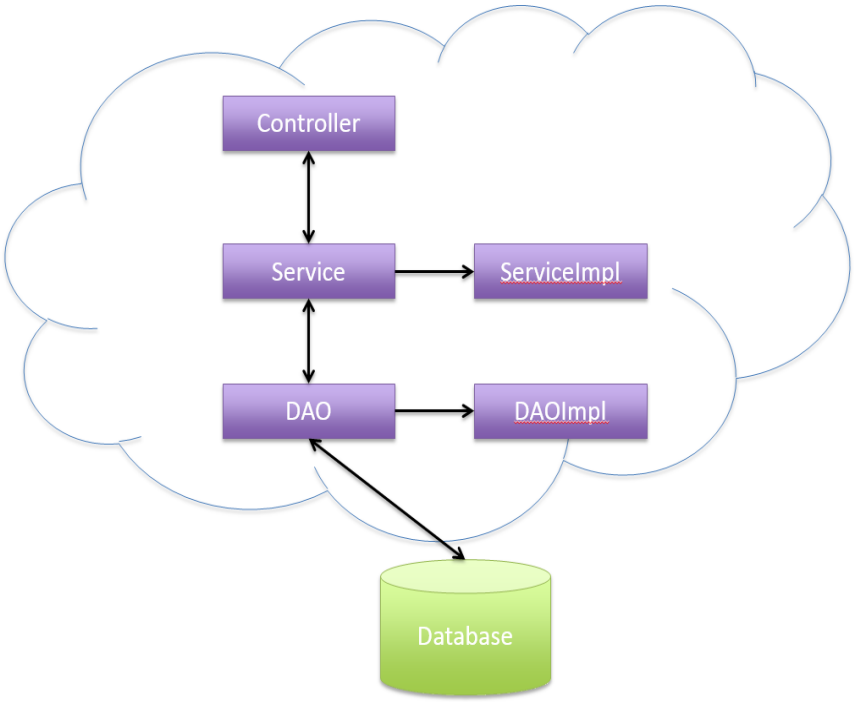


Figure 30 - Database connectivity after refactoring

4.4 Post-refactoring Evaluation by MITRE

After completing refactoring and addressing as many violation outlined by MITRE's initial report, we were able to resubmit the code base to MITRE to get a post-evaluation of our efforts. Refactoring took roughly 2 months.

Our Total Quality Indicator (TQI) was increased from 2.59 to 2.83. Each quality area measured received an increase, with the larger increases occurring in the robustness, efficiency/performance, and security. All of these were areas we decided to focus on after receiving the initial feedback from MITRE. Even though we reduced many redundant functionality and classes when refactoring occurred, our code base increased by roughly 4,000 lines of code. However, it was noted that our technical debt was increased, due to an increase in functions with high complexity, most likely due to the increase of Spring-managed classes.

Quality Area	Percent Increase
Total Quality Index	6.8%
Robustness	14.9%
Efficiency/Performance	13.5%
Security	13.6%

Transferability	8.4%
Changeability	2.8%

Table 1 - Second Evaluation Results from MITRE

4.5 Talend Testing

Once all the pipelines had been created and the documentation was finished, we wanted to test the usability and robustness of our newly created Talend pipelines. Since the system will be used and maintained by MHTC administrators, we wanted to gauge the quality of the Talend software for someone who many not be a computer scientist, as well as the quality of our supporting documentation, so that the same administrator can create a new pipeline on their own.

4.5.1 Documentation User Study

To determine usability, we gathered participants who would be willing to test the software. Each participant was given the task to generate a pipeline for the metric "IPEDS Number of Colleges and Universities". This metric was taken from a CSV file that contained the number of colleges for each state. The participant would have to take this file and transform it in order to map the state to a state ID, get a metric ID, the year, and the value to be inserted into the database using Talend. The participants were randomly selected and split into two groups. The first group was given only the Talend Help Center website as a way to guide them and the second group was given the documentation created by our team. They were also given a file explaining the task assigned to them and the file they needed to transform.

The specific instructions can be found in Appendix B.

Each participant was timed to see how long it would take someone to complete a pipeline when first encountering Talend. Due to the lack of documentation online, everyone in the first group gave up before completing the pipeline with the longest attempt being 4 hours and 3 minutes. However, all participants in the group with documentation were all able to finish a pipeline after reading the documentation in under 3 hours. This time did not include the time it took for them to read through the documentation. The timer was only started once they began attempting to make the pipeline. Each participant was alone when attempting this task except for a member of our team, who was only there as an observer and could not help or answer any questions about the task. Below you can see a table showing each participant's time and whether or not they had documentation:

Participant	Time (Hours, Minutes)	Given Documentation (Yes/No)	Completed (Yes/No)
1	4h 3m	No	No
2	2h 27m	Yes	Yes
3	1h 46m	Yes	Yes
4	2h 39m	No	No
5	2h 12m	Yes	Yes
6	43m	Yes	Yes
7	12m	No	No
8	28m	No	No

Table 2 - Talend Documentation User Study Results

Since participants were so successful when given the documentation, we were able to conclude that with the documentation our team has created, future MHTC admins will be able to learn Talend in a quick and effective manner. This will decrease the learning curve and allow them to be able to start adding new sources much faster.

4.5.2 Pipeline Robustness Testing

We also needed to test the usability of the entire process of using Talend to pull down data (via a data wrapper), extract and transform it, then load it into a database. To do this, we decided to analyze each pipeline created that included a data wrapper and uploaded them via the Administration Center to a development environment (similar set up to that of the production environment) to see if they were able to run successfully without any issues.

Of the 20 pipelines created, 7 were able to run successfully on a server-like environment, and 1 contained issues at runtime. The other 12 pipelines were not tested as they did not contain a data wrapper; however, we'd like to note that all data uploaded into the MATTERS Dashboard over the period of this project was done so either through the Talend pipelines or through the Manual Upload functionality of the Administration Center.

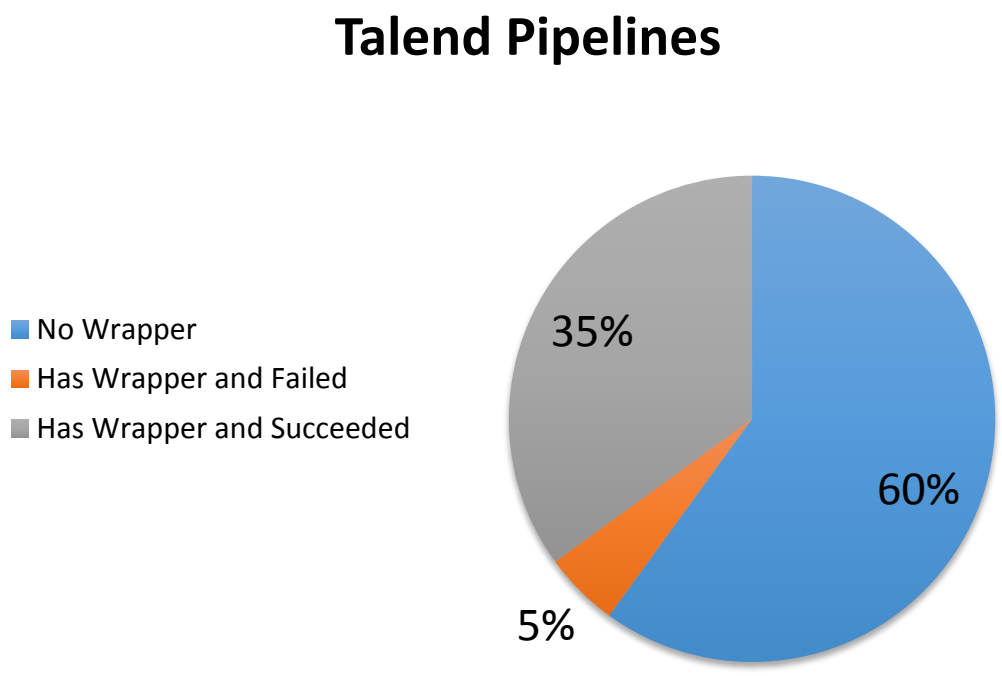


Figure 31 - Breakdown of Talend Pipelines

5 Conclusion

After 21 weeks of work spent into this project we not only gained knowledge about a number of technologies, but also created a system whose purpose is to help keep technology companies in Massachusetts, which coincidentally helps everyone in the team when they eventually search for a job.

The use of this tool will help give visual representation of a wide range of data ranging from tax rates to performance in sciences by the 8th grade class. This information will be used to compare data between states to help policy makers determine incentives for technology companies to keep them in the state of Massachusetts.

Throughout this project we had some great experiences. In particular, we got to see first-hand the process of improving a system from a very early alpha version into a large, fully-function application. We were able to work on and utilize so many different technologies, which was a great experience even though it required us to spend hours researching and learning new things. Another opportunity we received was research in the context of a real-world application. We spent weeks going through different technologies that we were interested in implementing into our system and had to do an analysis to determine if we thought it was worth putting in extra time to bring in another technology.

As with many systems, there is always a need for future work. The majority of our work was on the backend, which means the front end could use some additional work to implement new features. There is always the option of creating more pipelines for additional data sources.

Spending three terms on a project was pretty intense, but also provides a sense of pride because we know this system will be used and is not just some assignment for one of our classes. We hope that this does get used and improved upon because we spent a lot of time and effort on it. The project was a great experience and we are happy with how everything came together to present a working product for MHTC.

References

1. "Connecting TheData-Driven Enterprise." *Integration Software Leaders*. Web. 26 Apr. 2015. <http://www.talend.com/about-us>
2. "Talend by Example." -. Web. 26 Apr. 2015. <http://www.talendbyexample.com/>
3. Manning, Jeff. "ETL Tool Analysis." E-mail interview. 1 Oct. 2014.
4. "TalendForge Tutorials." *TalendForge Tutorials*. Web. 26 Apr. 2015. https://www.talendforge.org/tutorials/menu.php
5. "If You're Looking for Help, You're in the Right Place!" *Welcome*. Web. 27 Apr. 2015. https://help.talend.com/display/HOME/Welcome
6. CodePro - https://developers.google.com/java-dev-tools/codepro/doc/
7. Talend - https://help.talend.com/display/TalendAdministrationCenterUserGuide521EN/4.3+Managing+Job+Conductor
8. Talendforge - https://www.talendforge.org/tutorials/tutorial.php?idTuto=54&nbrFields=10
9. DataTables - https://datatables.net
10. JSoup - http://jsoup.org/
11. "New ETL World Record: 5.4 TB Loaded in Under 1 Hour - Syncsort"
12. "Mass High Technology Council Overview," 2014
13. "Massachusetts' Technology, Talent and Economic Reporting System - MATTERS - Mass High Technology Council," 2014
14. Firebug - http://getfirebug.com/whatisfirebug
15. Tamper Data - https://addons.mozilla.org/en-us/firefox/addon/tamper-data/
16. Quartz - http://quartz-scheduler.org/
17. Postgres - http://www.postgresql.org/
18. Spring - http://spring.io/
19. Maurizio Lenzerini (2002). "Data Integration: A Theoretical Perspective" (PDF). *PODS 2002*. pp. 233–246.
20. Kimball, The Data Warehouse Lifecycle Toolkit, p 332

21. Metronic - http://themeforest.net/item/metronic-responsive-admin-dashboard-template/4021469
22. CloverETL - http://www.cloveretl.com/
23. Pentaho - http://www.pentaho.com/product/product-overview
24. KNIME - https://www.knime.org/
25. Bootstrap - http://getbootstrap.com/
26. JQuery - https://jquery.com/

Appendix A: Analysis of ETL Software by Jeff Manning

Introduction

Organizations grapple with the extract, transform, and load (ETL) quandary every day. As organizations' needs grow, their data needs grow. Unfortunately, so do their respective ETL projects grow increasingly more complex (and unwieldy). Organizations face a painful choice: continue in-house development of custom solutions or leverage a data integration frameworks. The focus of this report is to analyze and rank the current (F/OSS) ETL market for possible inclusion in the MATTERs project. Here F/OSS refers to Free and Open Source Software.

ETL systems "offload" tedious, time consuming, one-off custom data integration coding to a streamlined framework (both engines and graphical user interfaces). In short, ETL traditionally refers to the following core capabilities:

1. Extract. The process of extracting data from external (to the system) sources. These can include, but are not limited to, files (csv, Excel, reports, tar, zip), URLs, Web services, and other databases (e.g. SQL query).
2. Transform. The application of "transforming" the data, including cleaning, rules, mappings, and functions required to transform disparate sources into operational form.
3. Load. Persistently store (insert, update) information (relational, data warehouse, or data mart).

To motivate the analysis, a story board was generated.

A new data source is released somewhere on the Internet. Staff from Massachusetts High Technology Council sees this source and wants to use it in the MATTERs application. They will either (1) submit a request for an admin to add the source or (2), if they have admin privileges, they will go to add the source. The admin will navigate to the admin page and be able to add the source to the application. By adding the source, the data will be scraped, cleaned, and loaded into the data warehouse.

The story board is very general. New sources are squirreled away in local files. The local files are parsed and eventually feed to the data warehouse. The above story board was analyzed and generated the following additional levied (or derived) capabilities:

1. Allow nonprogrammers/software engineers to bring online new data sources.
2. Code Generation. Ability to export code to seamlessly integrate into the MATTERs data pipeline. The MATTERs data pipeline is a custom, non-threaded ETL process for importing data into the MATTERs system.[1] See Section 2 for details.

[1] It is unknown at this writing whether the existing MATTERs pipeline will require additional modification to support code execution. It is assumed the option does exist to modify/generalize the pipeline to incorporate additional third (This seems incomplete)

3. Advanced Visualization (nice to have). Applied more advanced visualization techniques including scatter plots and 3D capabilities. Visualizations for both the inbound data set (external source), as well as operational databases.
4. Analytics (nice to have). Various data mining (DM) techniques including both supervised and unsupervised methods. Analytic(s) apply to both the inbound data set (external source), as well as operational databases.

The story board was generated under the following assumptions:

1. Source addition (by privileged user) is assumed to update the production tables.
2. The IDEs (see capability 4) have some capability to test workflow before load stage (debug).
3. Rollback/restart a new source is required (inserts can fail). Performance/Integrity is important.
4. It is beyond the scope of this ETL treatise to "remove" previously added metric sources.

In addition to the above story, several software packages were identified for analysis (in no particular order)[2]:

- *Talend Open Studio.*
- *CloverETL*
- *KNIME*
- *North Concepts Data Pipeline,*

It is beyond the scope of this analysis to integrate and demonstrate any specified product with MATTERs. It is the focus of this report to collect market information, ascertain subject matter expertise, and analyze product literature for matching capabilities. Once collected, analysis will rank the products.

Challenges:

Just because a pipeline has started, does not mean it successfully completed. How will you recover? In looking at the current implementation (quickly), I did not see any transaction support. There is no rollback. I could be wrong, looked like a straight insert. Assuming no transaction support, several questions are raised:

- What happens if the super-user (or admin) resubmitted the file for pipeline process?
- Does the pipeline know how to restart the job?
- My assumption is this is a production database (no staging). What is the state of the database if an import fails? What 10,000 rows were inserted?

[2] This product list is not considered exhaustive or static. As new frameworks are identified, thy will be quickly added and analyzed.

- Can the conclusions be trusted? Do we even allow "analysis" to occur in this condition?

Perhaps the pipeline has a GUI that gives intermediate results (I did not look into the models to WEB-INF). If not, it might be nice to provide the user some feedback (progress)).

The document presents the analysis in the following sections. Section 2 describes a high-level architecture of the MATTERs data pipeline. It captures the salient ETL capabilities used by MATTERs. Section 3 reviews in greater detail the aforementioned four ETL frameworks. The section builds a product ranking. Section 4 recommends a way forward/next steps.

MATTERs Data Pipeline Architecture

The following figure represents the high-level MATTERs data pipeline architecture (with respect to core ETL capabilities).

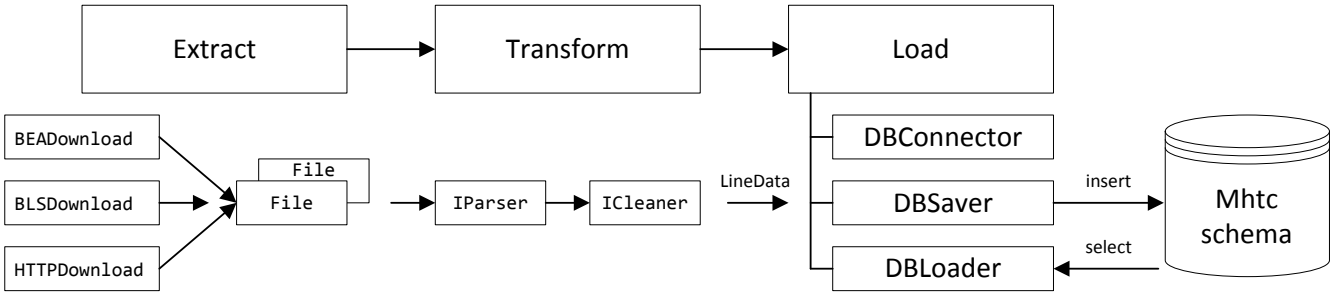


Figure 32 - Existing MATTERs Pipeline Mapped to Logical ETL

The implementation leverages concrete classes to implement specific functions required to achieve operational data integrity. There are some exceptions. The specifics of the file parsers (IParser) and the types of ICleaners are left to run-time construction. The ICleaner run time construction means that a class factory is used to construct the IParser for parsing. It is not known until runtime which IParser (concrete implementation) is constructed (based on the file passed to the class factory). The DBSaver inserts the collected metrics (represented in LineData) into the Statistics table (Mhtc_Schema). Extract capabilities download data and store local files (used by IParser).

Limitations:

- There is limited flexibility in the ETL flow. The process is a one directional process from source to sink. Only parsers are tuned to the concrete file types. ICleaner is really a subprocess within the IParser.parseALL().
- New sources will likely require new parsers (or the extraction process writes to csv file formats). What if the new source is a web service? Will the existing tools work here? What about an API (Wsdl)? Who wraps the protocol to get the data? It can be argued that these are in the purview of the data access (extract block). However, this still needs to be written. If the extractor (downloads) writes out the file in the intermediate form that the parser can handle, then no new parser will have to be written. However, the

problem has just shifted to the extract capability to map the new format to the existing (and all that it entails). Either way it is work.

- Extract creates local files for reading/parsing/inserting. IParser detects the correct parser (factory pattern) based on the file.
- ICleaner transforms are simple mappings (MASS to MA). Data values are cleaned with regular expressions (remove spaces, commas, etc.). Current data sources require simple transforms (normalization/standardization).
- Difficult for users to get a feel for ETL progress. Failures through terminating exceptions. How do you restart a previously failed data source (from file)?
- Scale. The ETL flow is not multithreaded (not sure this was a consideration).

A logical model can be made available upon request. For the purposes of this report, the central schema element is the STATISTICS table. The analysis focuses solely in the inserting of data into this operational table (STATE, METRIC, and CATEGORY are already present). The primary keys (StateId, MetricId, and Year) are sufficient to maintain integrity (this was not validated). The data element stored is VALUE (and the date the value was entered).

ETL Framework Analysis

Although the set of products was given for analysis, no set of discriminants (metrics or grading criteria) were specified to evaluate the product list. Based on limited conversations (email) with team members, the following metrics are used for evaluation:

1. All core capabilities
2. At a minimum, the first two extended capabilities
3. Simple. Both GUI (drag/drop) and ETL engine framework with Java code generation capabilities. Cannot have an arduous learning curve.
4. Scalable/performance. Multithreaded—extract, transform, and load are all individually managed in a workflow pipeline.
5. Market adoption. Ideally, the framework has a vibrant community and a market presence.
6. Subject matter experts (SME) were consulted for professional experience with specified tools.

Any product not meeting both 1 and 2 was removed from consideration (there were none).

Some metrics were removed from a previously more exhaustive list. Most products considered had an abundance of data access methods. The supported access methods not only covered existing hand- coded techniques but all supported likely future data sources. Additional metrics such as low impact to MATTERs current, custom ETL code base were likewise removed. The MATTERs pipeline is likely to see wholesale changes (or two separate, orthogonal, complementary pipelines will be supported).

The proposed discriminates are qualitative measures (subjective). In the event the grading criteria is altered, the recommendation could possibly change.

Talend Open Studio

Talend is a comprehensive Open Source ETL.

- The off-the-shelf product supports all core capabilities (basic needs).
- Supports the following extended features:
    - Does have a simple intuitive IDE to design "procedures." Procedures can be tested in IDE! Supports a large number of components.
    - Eclipse-based tool; generated Java code (single threaded); encompasses the Java ecosystem; no black box engine.
    - Support a large number of data extract formats (future proof).
- Cannot determine whether nice-to-have features are supported.
- Cited most often from the SME pool but considered a "steeper" learning curve.

CloverETL

CloverETL is a lightweight, Java-based data integration framework. CloverETL can be used either standalone or as and embedded framework. Analysis reveals several issues/concerns:

- The community version (F/OSS) does support the basic needs of the MATTERs project (core requirements).
    - Does have a simple interface
    - Limited scaling (six readers, five writers, nine transformers).
- None of the "nice-to-have" capabilities are supported in the community (F/OSS) product.
- Necessary to get the same feature set as in other the community (F/OSS) version must be upgraded to the designer product line (commercial product).
- Unsure as to market adoption; no SME queried previously used the product.

KNIME

KNIME is an advanced framework meeting all seven capabilities. KNIME provides ETL capabilities. However, it is an advanced analytics package (i.e., similar to R, Orange, SAS). KNIME advanced capabilities including but not limited to exploratory data analysis (EDA), powerful analysis tools (regression, logistics, GLM, etc.), predictive analytic tools, visualization, and reporting.

Analysis reveals:

- Powerful framework

- Depending on analysis needs, KNIME can augment existing analysis with additional capability (beyond the scope of the immediate ETL needs).
- No SME recommendation. Acknowledged: the focus of the SME was ETL, not predictive analytics.

North Concepts Data Pipeline

To be completed.

Recommendation

The research done to date largely involved reading documentation, consulting SMEs, and reading various product reviews. Due to the time commitments and the number of solutions to investigate, the products were ranked.

1. Talend Open Studio
2. KNIME
3. CloverETL

Next Steps

These frameworks present a compelling case for adoption. The simple, intuitive IDEs and code generation can greatly reduce the impact of incorporating new data sources, thus free engineers to work other critical capabilities.

The next steps are straightforward:

1. Download the trial software to validate the top-ranked product claims.
2. If successful, take a sample source and re-create the data pipeline, using the framework (output to file not STATISTICS table).
3. Examine the Java code, performance, and scalability.
4. Investigate the incorporation of the Java code into the MATTERs pipeline.

Regardless of the selected tool, MATTERs pipeline is likely to see significant changes. Alternatively, a second "pipeline" is developed, orthogonal and complementary to the existing approaches. The goal would be to add new sources support, then back fill existing sources (remove original ETL implementation).

As noted, minimal time was allocated to download trail-ware versions and validate claims. The number of products and the time allocated to the effort could not support the full regression testing. This effort is a necessary step. It validated the market and capabilities for the F/OSS ETL market. Secondly, it honed the set of products to meet the current (and quite possibly) future needs of the MATTERs project.

Appendix B: Talend Documentation

Talend Documentation

Table of Contents

Table of Figures

Creating a Talend Job

1.1 Setting up Talend

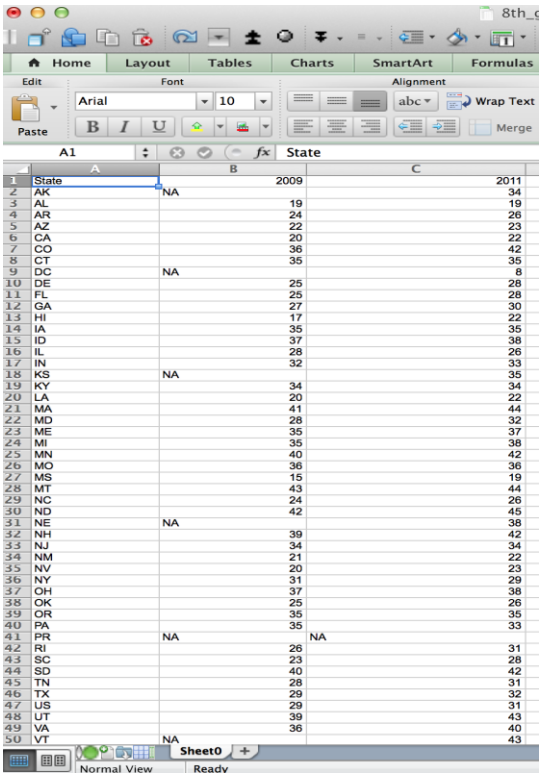
First you will need to have Talend installed on your machine. Talend can be found here: https://www.talend.com/

Follow the site's simple download and setup instructions to get Talend up and running on your machine. Once you have Talend installed you can start creating jobs. A job is a workflow that you create for Talend to execute. This can be as simple as reading a file or can get more complicated to clean data from a file, transform it and insert into either a new file or a database. In this example we will be taking an excel file and inserting it into our database. Along the way we will also be cleaning the data so it fits our database schema.

1.2 Getting Your Data

First before you can begin you will need to have your excel file that you plan to load somewhere you can access it whether it be on your machine, some shared file system or being downloaded by one of our wrappers. More information on wrappers can be found in section 3.2.2. In this example we will be using the 8th grade student performance in Math and Science file.

Below you can see what the file looks like:



State	2009	2011
AK	NA	34
AL	19	19
AR	24	26
AZ	22	23
CA	20	22
CO	36	42
CT	35	35
DC	NA	8
DE	25	28
FL	25	28
GA	27	30
HI	17	22
IA	35	35
ID	37	38
IL	28	26
IN	32	33
KS	NA	35
KY	34	34
LA	20	22
MA	41	44
MD	28	32
ME	35	37
MI	35	38
MN	40	42
MO	36	36
MS	15	19
MT	43	44
NC	24	26
ND	42	45
NE	NA	38
NH	39	42
NJ	34	34
NM	21	22
NV	20	23
NY	31	29
OH	37	38
OK	25	26
OR	35	35
PA	35	33
PR	NA	NA
RI	26	31
SC	23	28
SD	40	42
TN	28	31
TX	29	32
US	29	31
UT	39	43
VA	36	40
VT	NA	43

Figure 33: Example Excel File for Talend to Use

As you can see this file is not in a proper format to be entered into our database and has some data missing so we will need to fix that using Talend.

1.3 Creating Your First Job

Now that we have Talend installed and we have our excel file we can move onto creating a Job to put this data into our database. First from the Talend screen you are going to want to right click on job designs and click create new job:

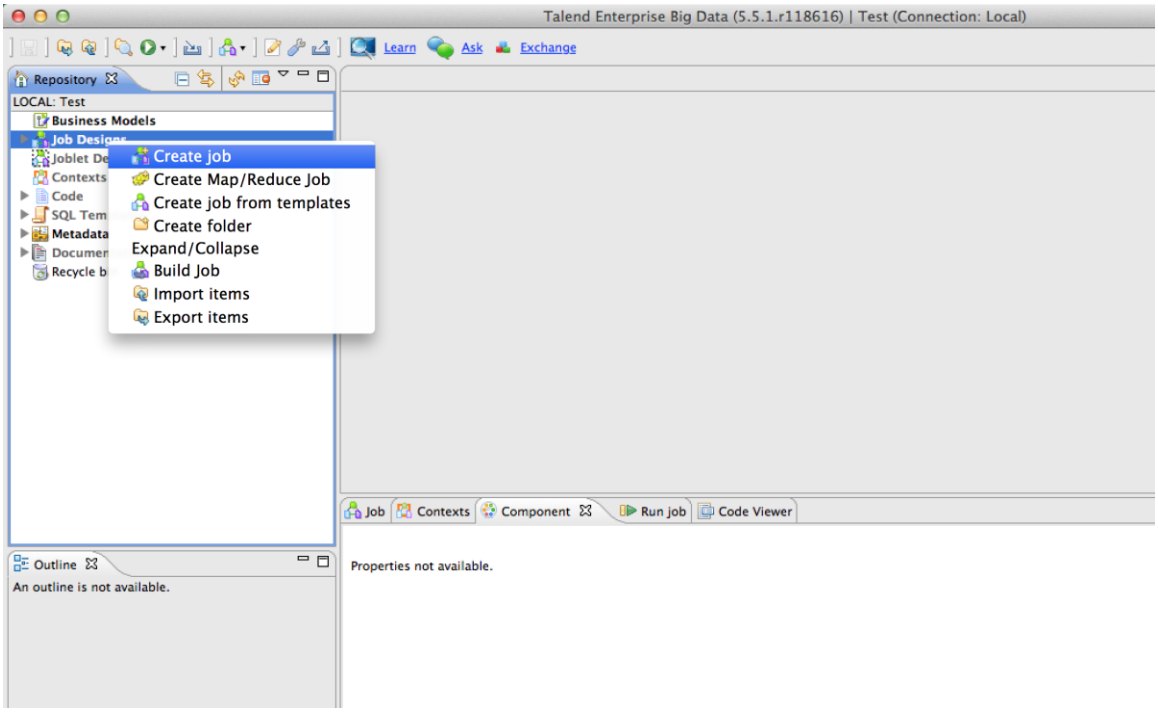


Figure 34: Displays the Create a Job Button in Talend

You will then be prompted with a **New Job Wizard** seen in the image below. The only required field is the name field however writing in a purpose and description will help future users understand the pipeline better.

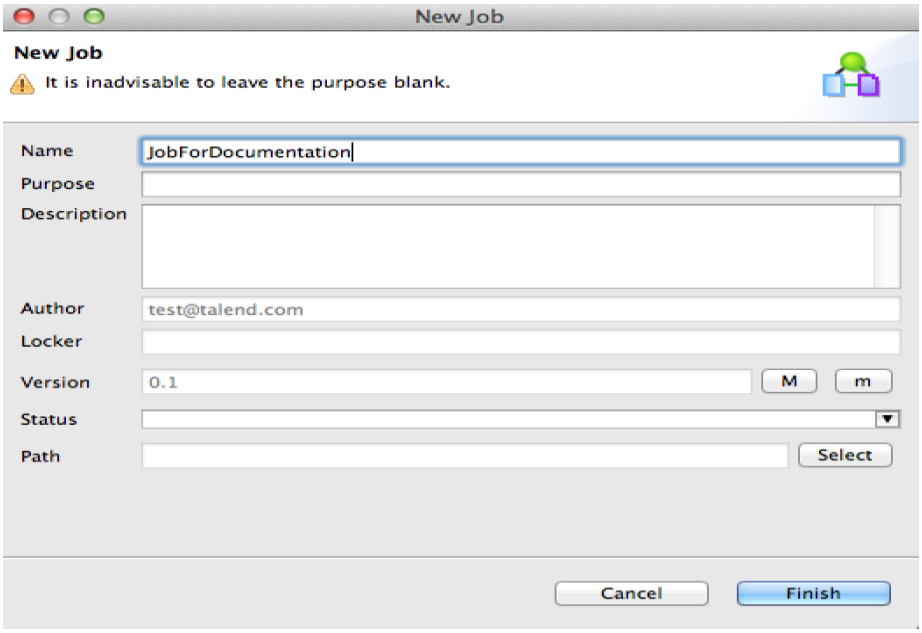


Figure 35: New Job Wizard

The first thing you may notice after creating a job is that the **Palette pane** has populated, this is where you can find all the components Talend has to offer and is how we will build our job.

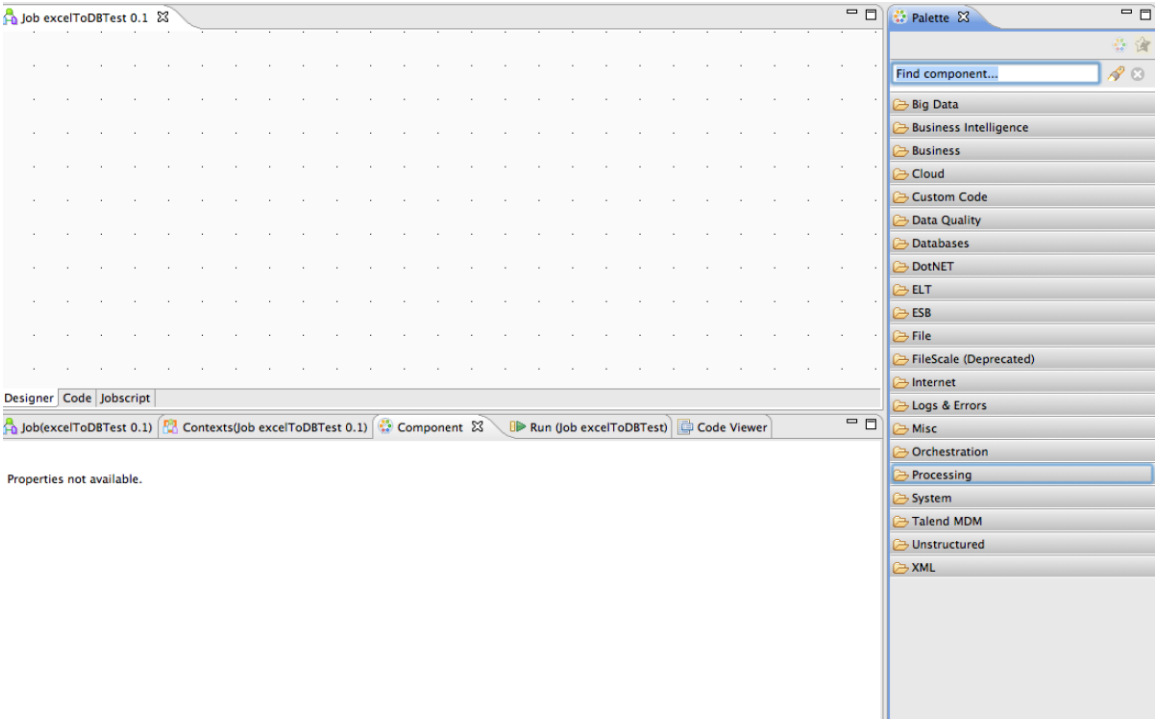


Figure 36: The Palette Pane is Populated

1.4 Adding a File

Now that we have our Job the first thing we will need to do is to pull in the data from our excel file. However in order to ensure that this file can be used in other jobs as well we are going to create a template for it in Talend. To do this: expand the **Metadata** tab in the **Repository pane**. Then right click **File Excel** and click **Create file Excel.**

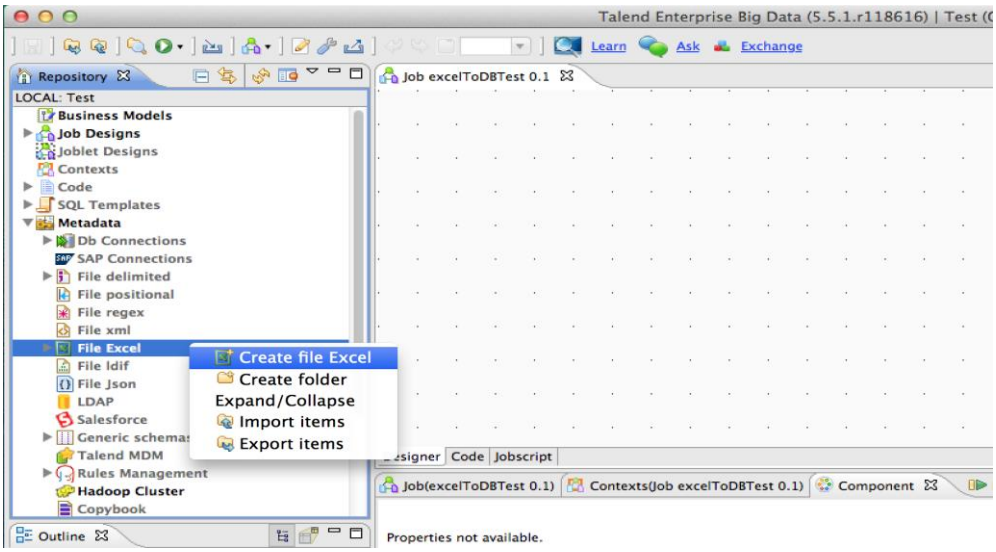


Figure 37: Creating an Excel File in Metadata

This opens up a wizard for creating the excel file. The first screen is self-explanatory, just enter a name and optionally a purpose and description. Then on the next page you will need to browse to the location of the excel file you wish to use and select the sheets for Talend to use. Some times a file will have multiple sheets and you will need to select which sheet to base the schema off of for this example there is only one sheet so just select that one.

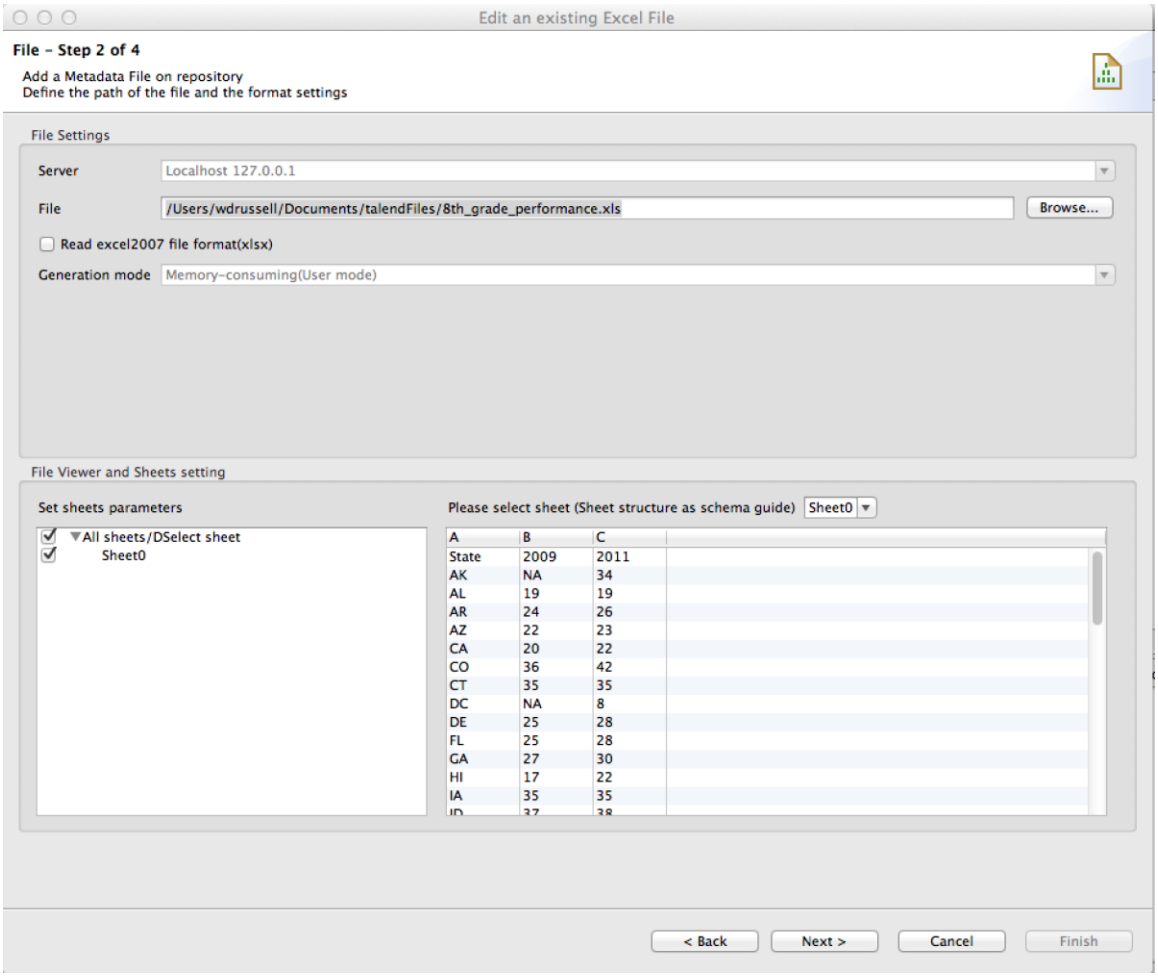


Figure 38: Excel File Wizard Step 2

Then on the next page you can define the column names of the excel file. You can set limits on where the sheet starts and ends both for columns and rows. It also provides a preview of what it will look like. You can also adjust where you want the file to start. For this example we just need to set the heading row as column names which takes the first row in the excel file and makes that cell the column name. Below you can see what this looks like:

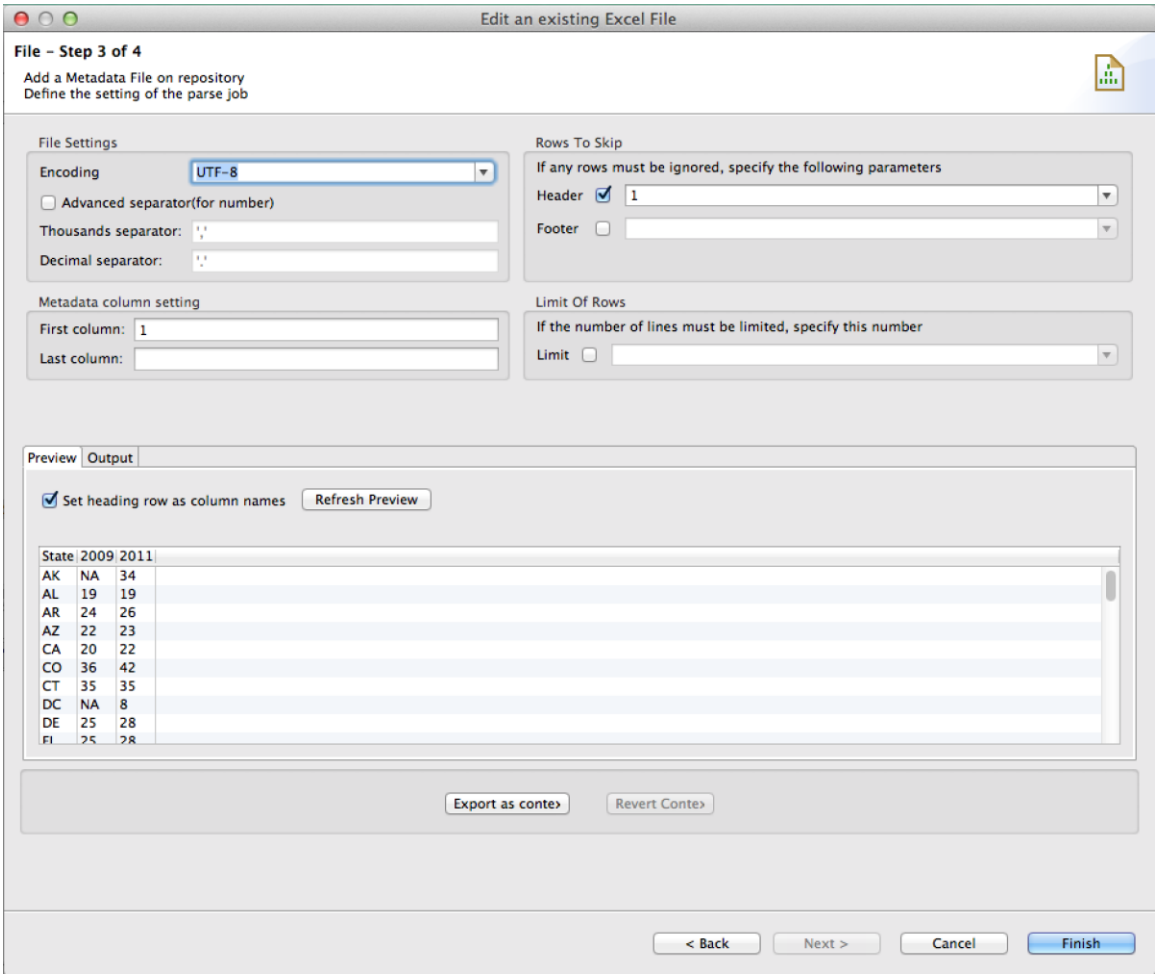


Figure 39: Excel File Wizard Step 3

As you can see the preview now displays the appropriate column names in the header. Then in the final panel you can update the schema. To keep things simple we will be leaving this alone. Then just hit finish and you should now see the excel file you created under the Metadata tab in the file excel section.

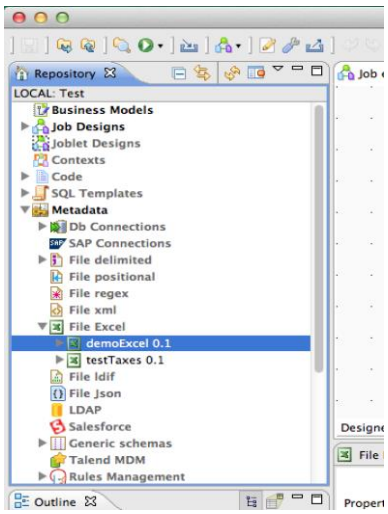


Figure 40: Metadata Tab With New Excel

Now you can drag the excel file you just created and drag it from the Repository Pane into your main Job window and make sure to select tFileInputExcel.

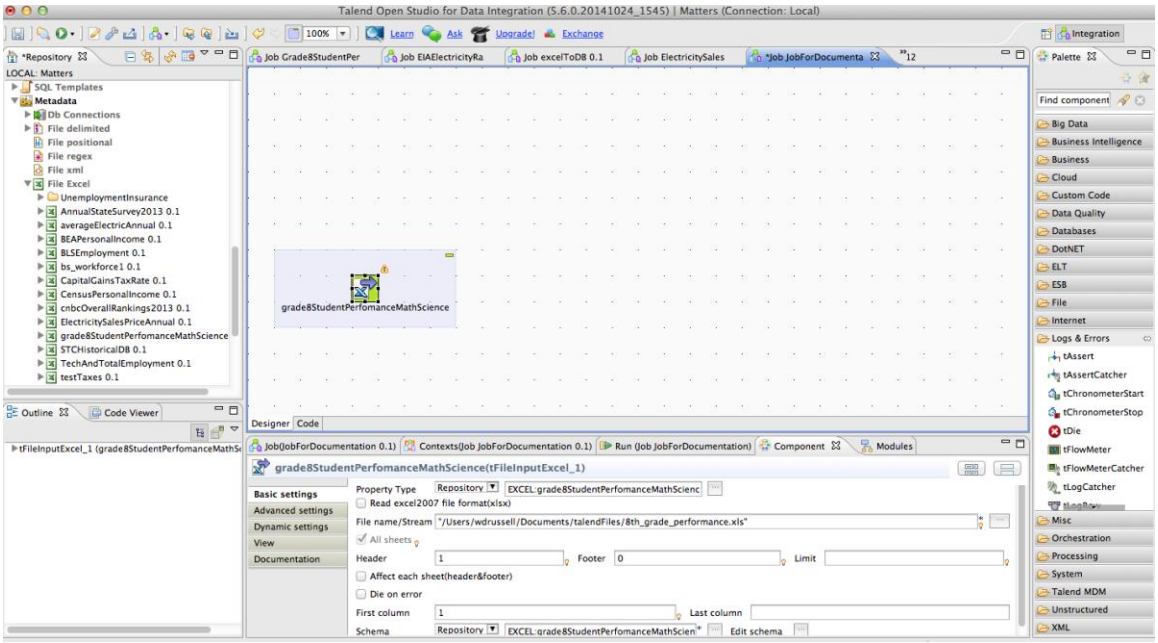


Figure 41: Adding Input to the Job

Now your job should look something like this. However we are still not done. We have a data wrapper for this file, which we can use to download it for the job so it doesn't have to point to somewhere locally. In order to set up the data wrapper we want to add a tJava component, which can be found under Custom Code in the Palette Pane. Below you can see the job with the added component.

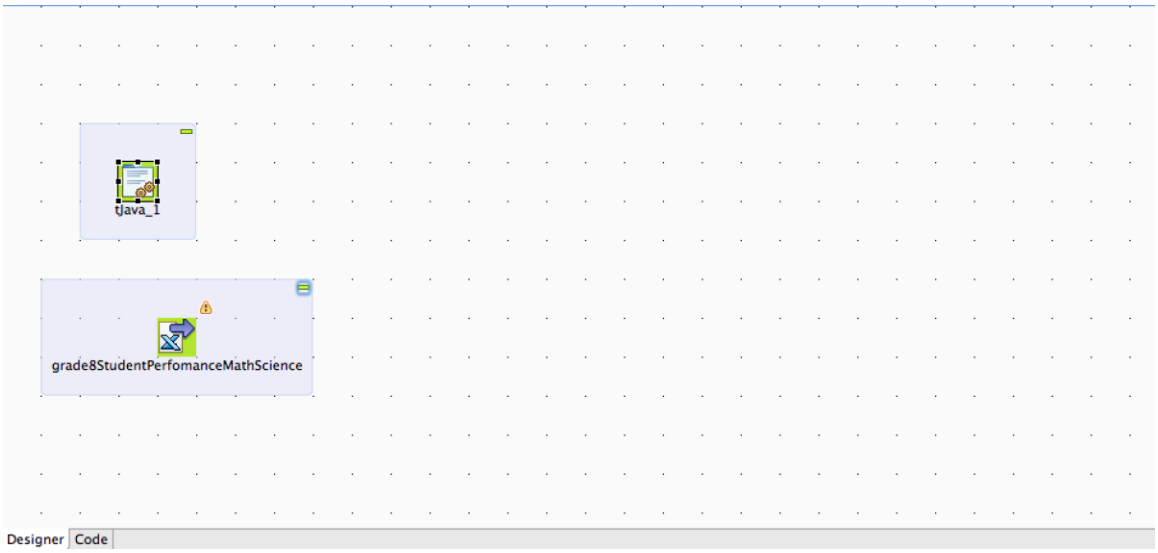


Figure 42: Adding a Wrapper to the Job

For the tJava component to do anything we need to edit its component tab to use our data wrapper. (For more information on data wrappers visit the routine documentation in section 3.2) In the component pane you are going to want to enter the following code in order to call the routine to download the file:

```
DataWrapperMain.createDirectory();

DataWrapperMain.EightGradePerformanceDownload();
```

Then your component body should look like this:

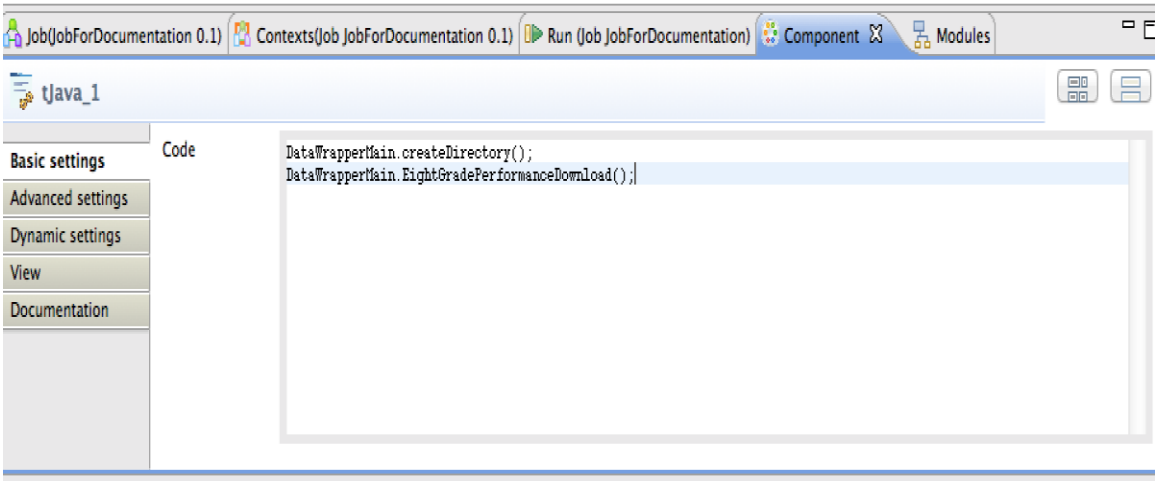


Figure 43: Component Body of tJava Component

Next you need to edit in the tInputExcel the file path. Select the component and then click inside the box. You will be prompted to either view the repository setting or change to built in connection. Click on the change to built in property radial and then continue

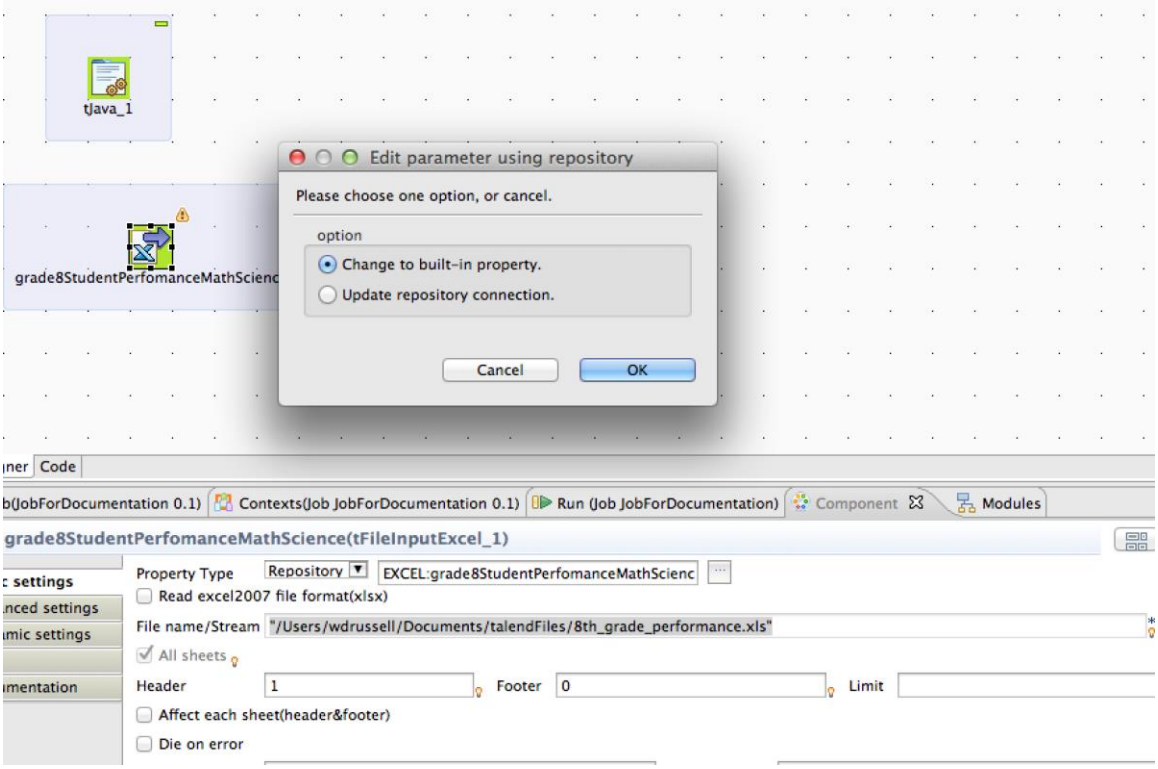


Figure 44: Editing the Excel Input Location

Then you are going to want to change the file name field to be wrapperDownloadLocation/<filename> so for this example it would look like it does in the image below:

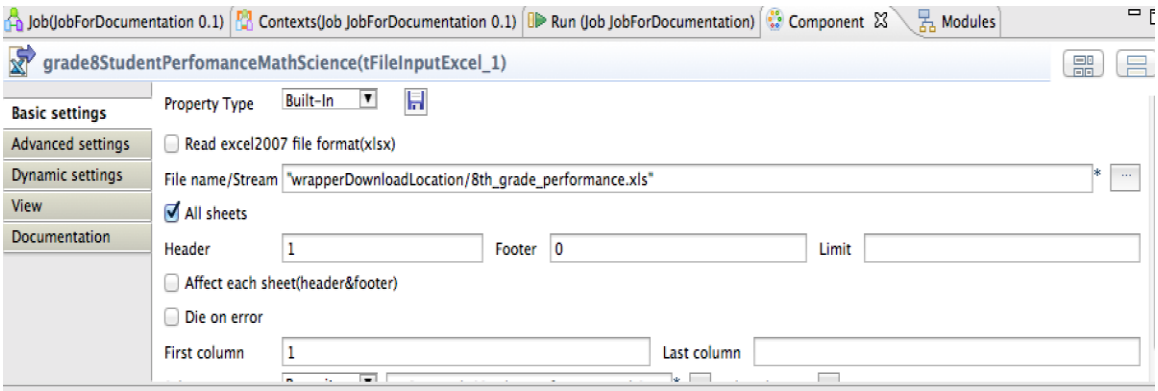


Figure 45: Setting the Input to Come From the Wrapper

Finally you need to right click on the tJava component and go to Trigger and select on component ok and attach that to the excel file so the pipeline wont start until the file has been properly downloaded. Now are pipeline should look like this:

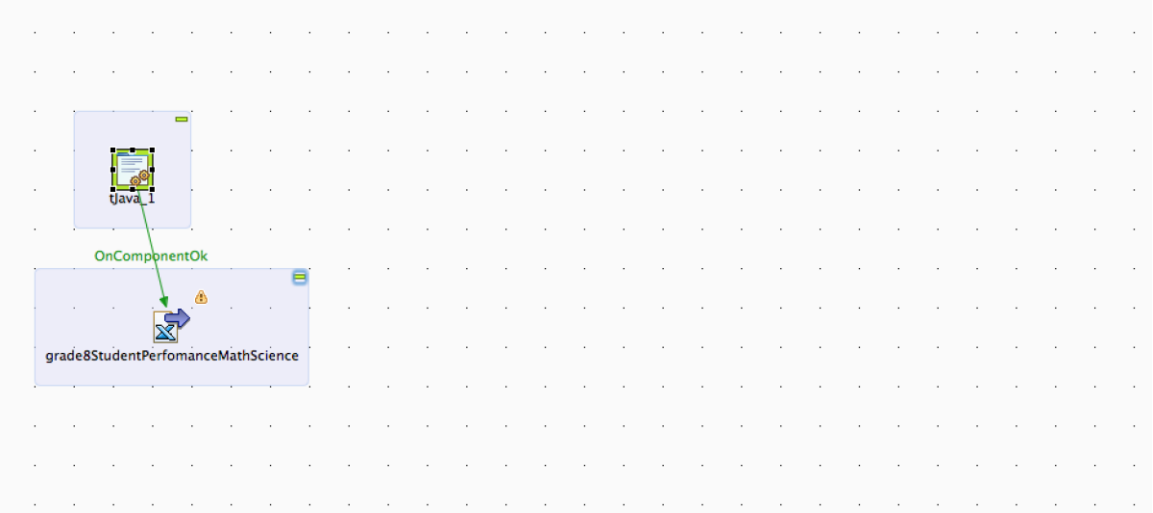


Figure 46: Connecting the two Components

1.5 Transforming the Data

First we are going to start by going over to the Palette tab and expanding the processing bar. From here click on the tReplace and add one into the job. Connect the main output of the file to the tReplace in order to feed the data long the pipeline from one component to the next.

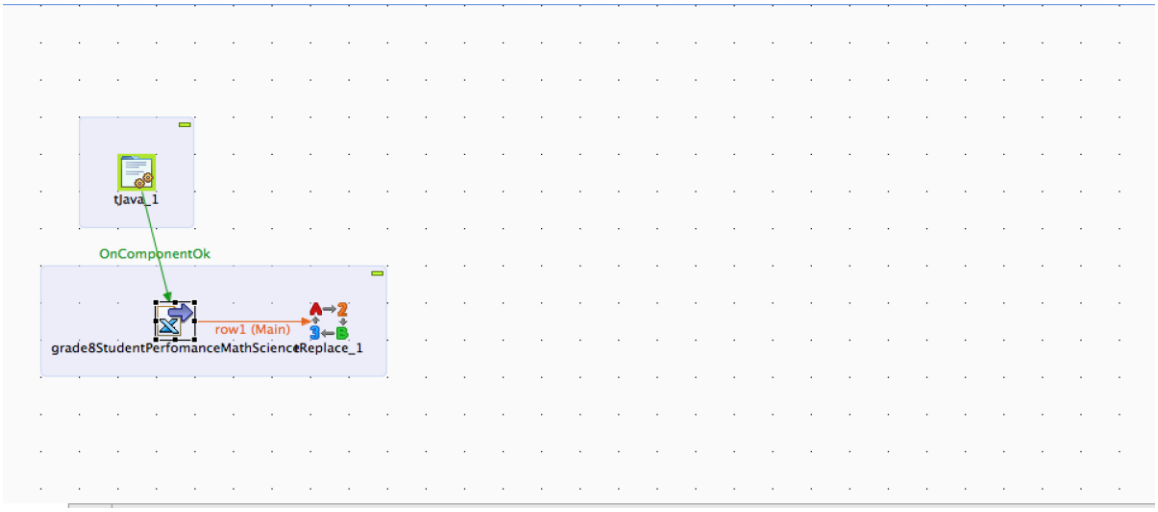


Figure 47: Adding tReplace to the Job

Now click on the tReplace component and got to its component tab in the bottom pane. For this file there are a few 'NA' in it instead of a number. We are going to need to replace these in order to transform the data later on so you will want to click the green plus sign in the tReplace to add a new rule to the component. (For this example you need to add two rules, one for each column) In the rule you are going to want to set it to the appropriate column and then have the search for be "NA" and in this example we will be replacing it with "0". Below you can see what the component should look like once you are done.

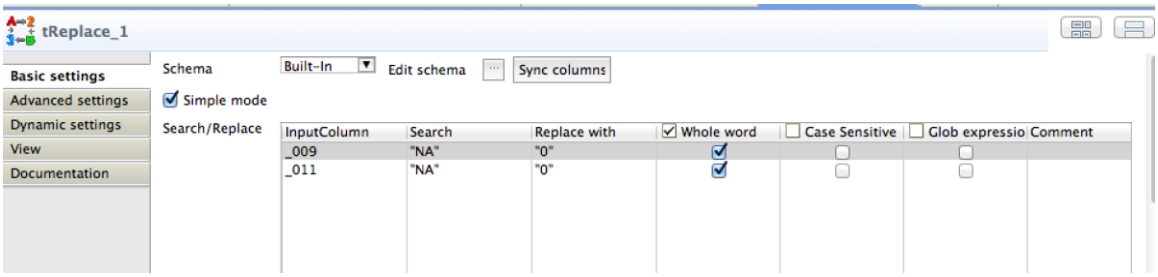


Figure 48: Component Tab of the tReplace

Sometimes there are some more things that may need to be removed or replaced such as commas or percent signs. Below is an example from another job so you can see some other rules you can set to help you clean your data files.

1.5.1 Extra Replace Example

Check the advanced mode check box so we can use regex to remove certain symbols. In here add four new rules. We want to replace any non-number characters in the wages column and percent signs from any of the percentage columns. Here are the patterns to do so:

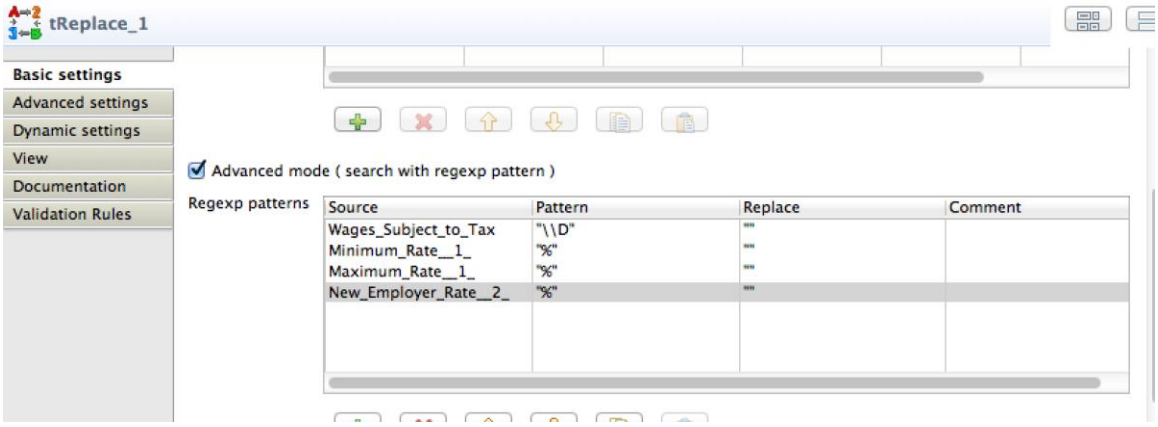


Figure 49:Patterns for Removing Non-Numeric Characters

The \\D just removes any non-numeric character and in the replace column we swap it for nothing. The other three rules are just searching to remove percent signs from the cells.

1.6 Displaying the Output

At any point during the process of creating your job you can stop and check what the current output is. To do this in the Palette Pane go to Logs and Errors and add a tLogRow to your job and connect it to your pipeline. This will allow you to see the current output. To run your job, in the bottom pane click on the run tab and then click the run button. After doing so your job should look something like this:

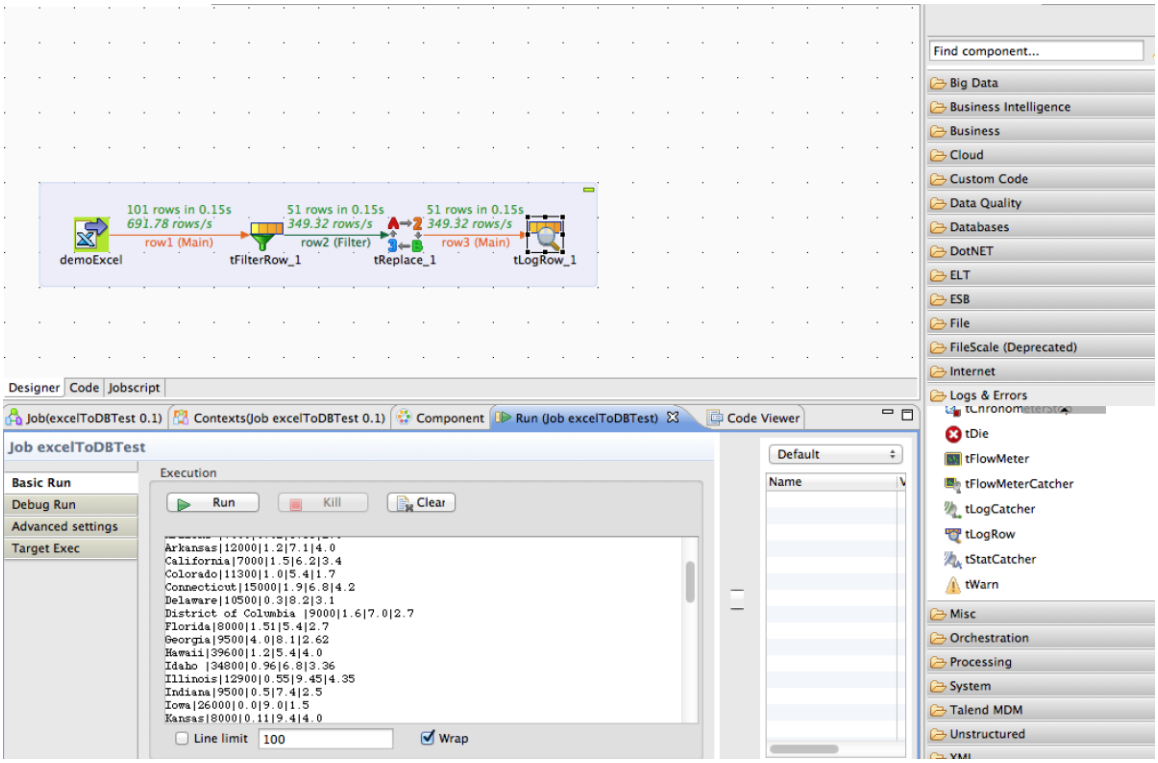


Figure 50: Adding a tLogRow to Display Output

1.7 The tMap

Now that you have checked your output at this point you can remove the tLogRow and we can continue on with the pipeline. Now we want to go back to the processing section in the Palette Pane and select the tMap component and add one to our job. Then connect the tReplace component to the tMap.

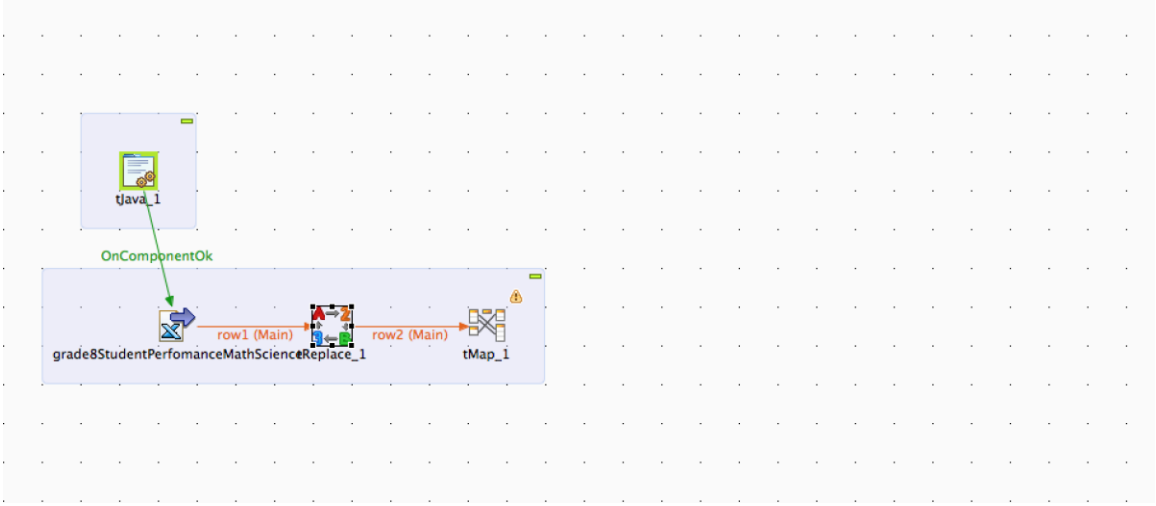


Figure 51: Adding the tMap to the Job

With the tMap component we can extract the information we want and select what we want to go into the database. Double click on the tMap component and you should see a screen like this:

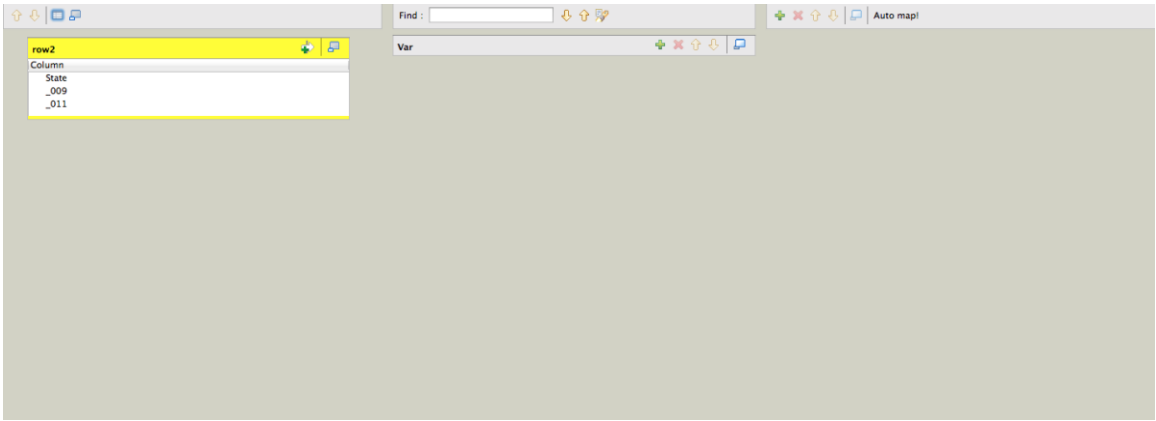


Figure 52: Inside the tMap

Here you can see three columns. The column all the way to the left is the input column. These are all the components feeding into our tMap. The middle column is for variables. This column lets you manipulate your input fields. Then on the right is the output column. Here is where you select what you want to be outputted from the tMap. To start lets first click the plus sign in the output column to create a new output and name it out1.

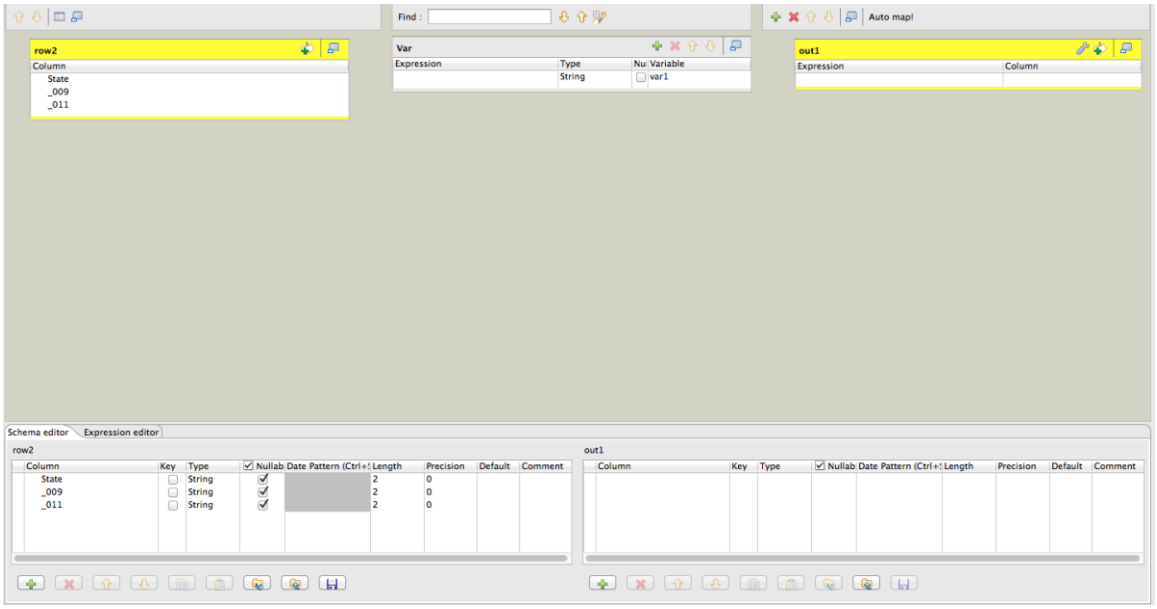


Figure 53: Adding an Output to the tMap

The tMap allows you to make different connections by moving the variables from input and variables over to different output tables so you can get just the data you want. For this we are going to need the value of minimum rate however currently it is set as a string and the database wants it as a floating-point number. To make this change lets first click the plus sign on the variable table to add a new variable and then drag _009 into that variable. Make sure to set the variable as a float. Once you drag it in you can see that a yellow line appears to show you where the input is going.

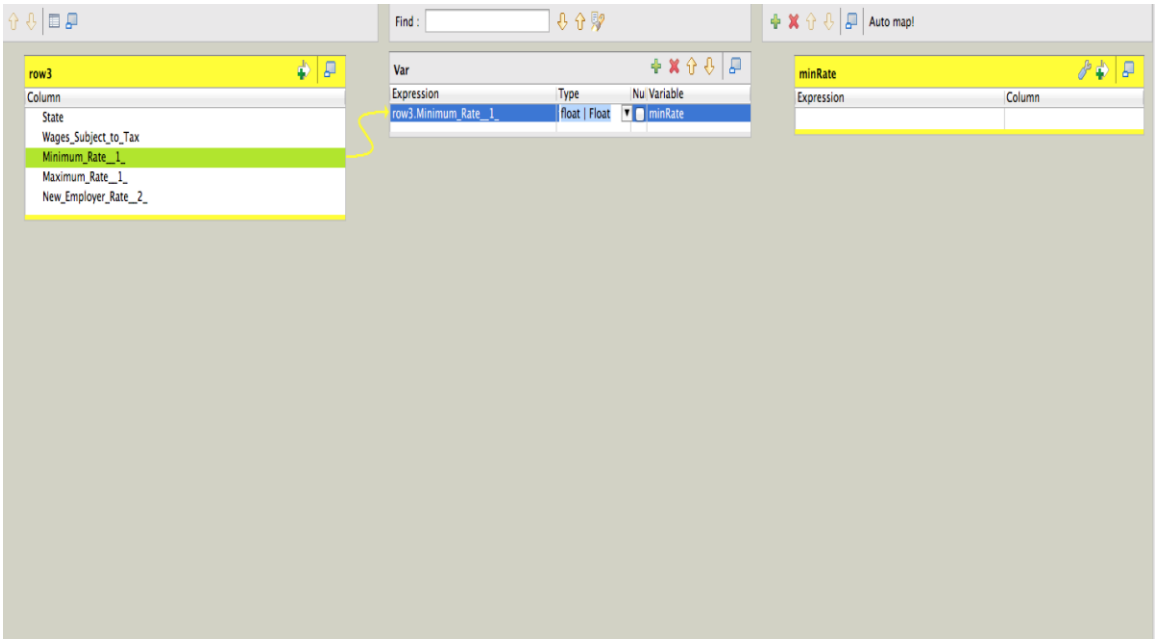


Figure 54: Adding Variables to the tMap

However this does not automatically change the input from a string to a float. First we need to write a little expression. Click on the variable and then in the lower left click on the Expression Editor tab. In here you can use various java functions to manipulate your data. For our purposes we want to make sure that there is data there and that we can turn it into a float to do so paste:

```
Float.parseFloat(row2._009)/100
```

into the expression editor. The parseFloat call turns the string into a float and we divide by 100 because these are percentages that we want to enter into the database as decimals. Once you have done this drag the variable into the output table we created earlier. Another yellow line should appear to show the connection.

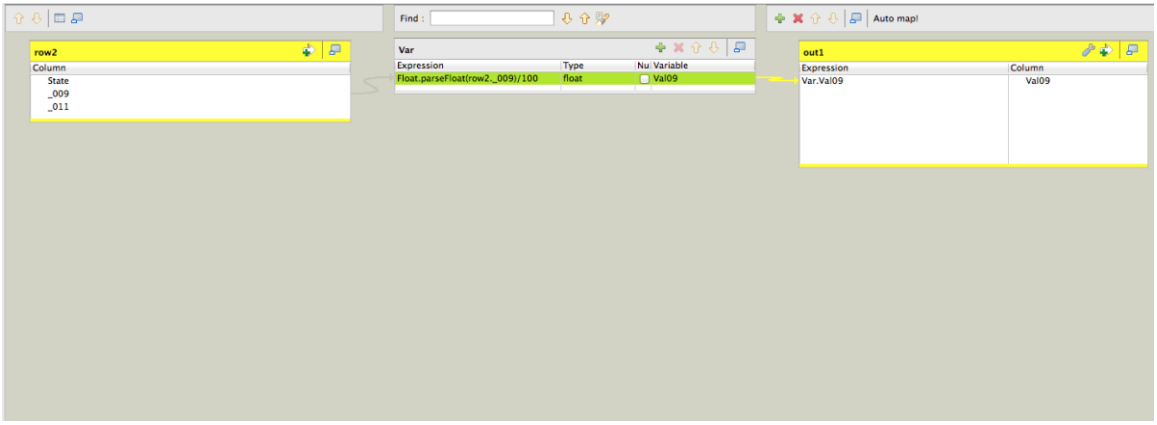


Figure 55: Moving the Data From Input to Output

Now we only need three more items in output: the StateId, metricId, and year. Lets first get the year since it's the easiest. Add another Variable to the variable column and then for the name call it year and for the expression type 2009 and make sure the variable is set to an int. Finally drag the variable to the output.

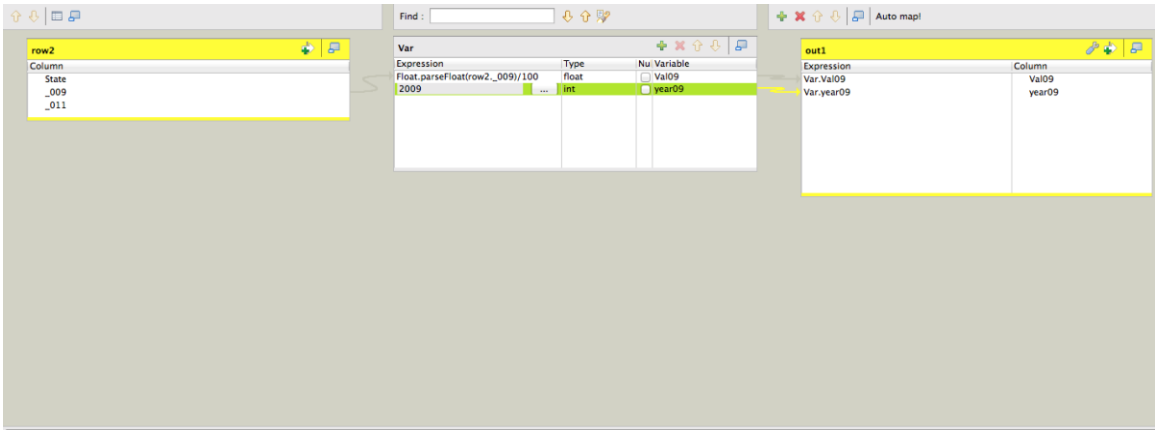


Figure 56: Adding the Year to the Output

1.8 Database Lookup

Now to deal with the StateId and MetricId, for these we will need to pull in some data from our database. To do so first we need to create a database connection. Under Metadata on the Repository, right-click DB Connections and select Create Connection, this will open up a wizard to create your connection. First just fill out the name of the connection.

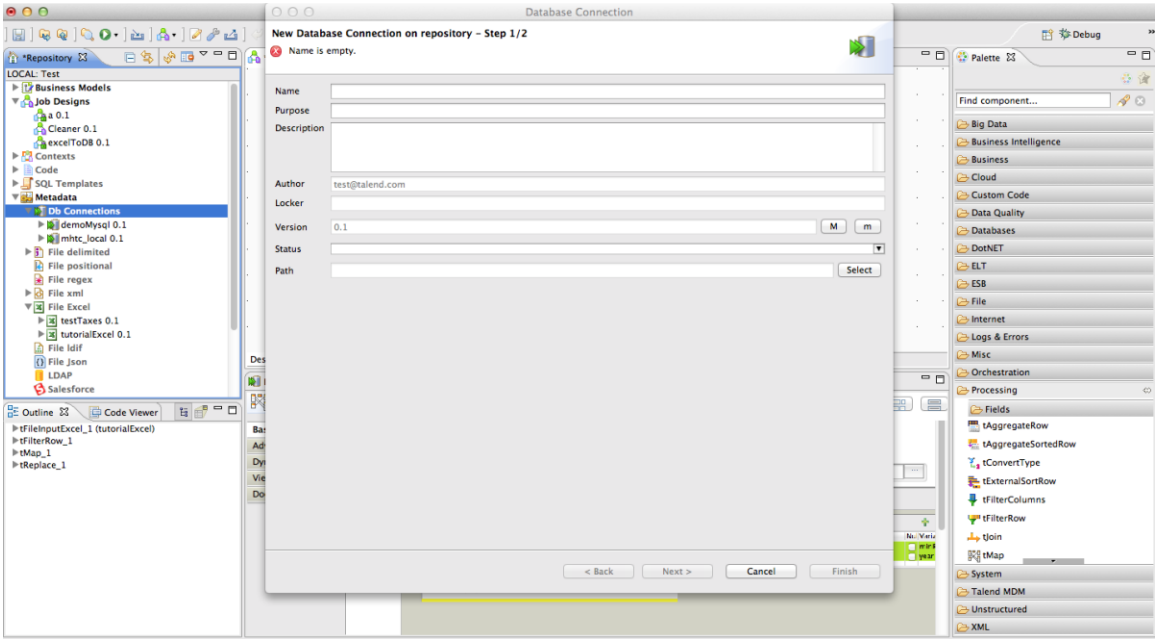


Figure 57: Step 1 of the Database Wizard

Then on the next page you will fill out all of the connection details. For this I will be using my local database.

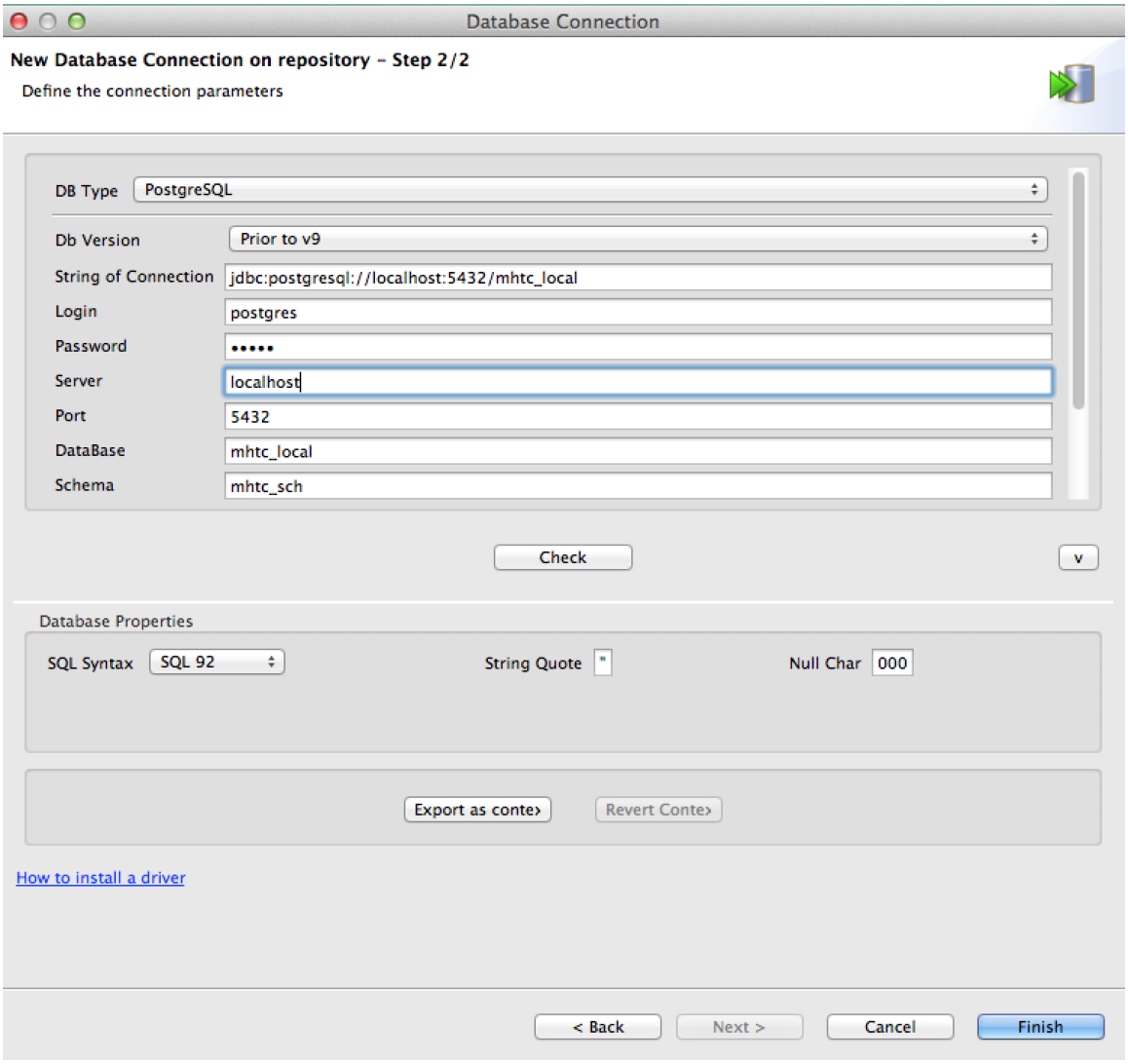


Figure 58: Database Connection Details

Once you have entered your information click the check button to make sure everything is right and you can establish a connection. Once done select Finish and then select your new connection from your DB Connections, drag it into the job window and select tPostgresqlConnection. This component will establish an initial connection to your database for use by other database components.

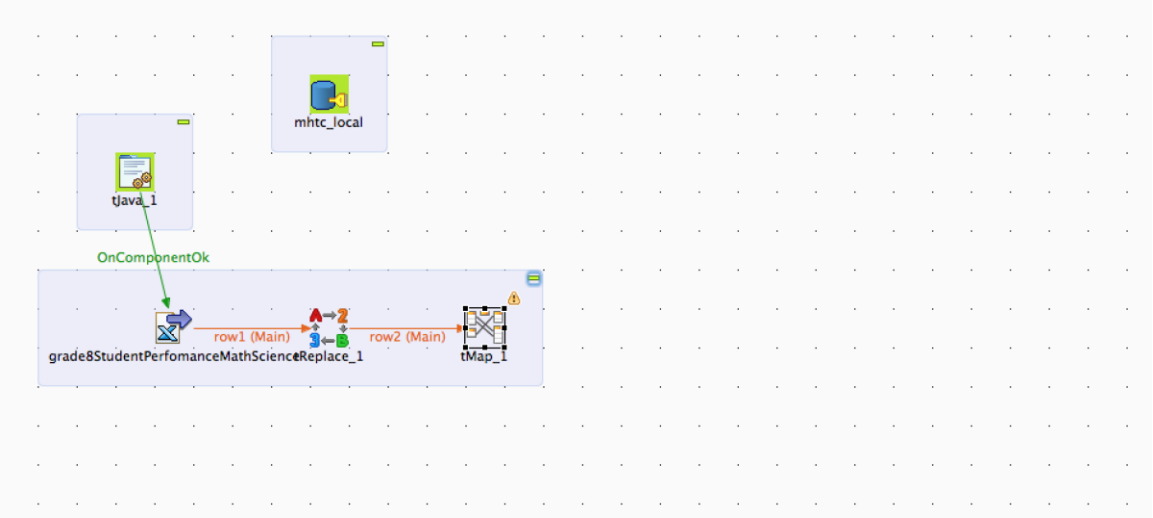


Figure 59: Adding the Database Connection

Next to make our lives a bit easier we are going to add some more items to our database connection. Right click your new DB Connection and click Retrieve Schema. This will let Talend know everything it needs to about the schema of all the tables in the database. It will open up a new wizard. On the First window just click next.

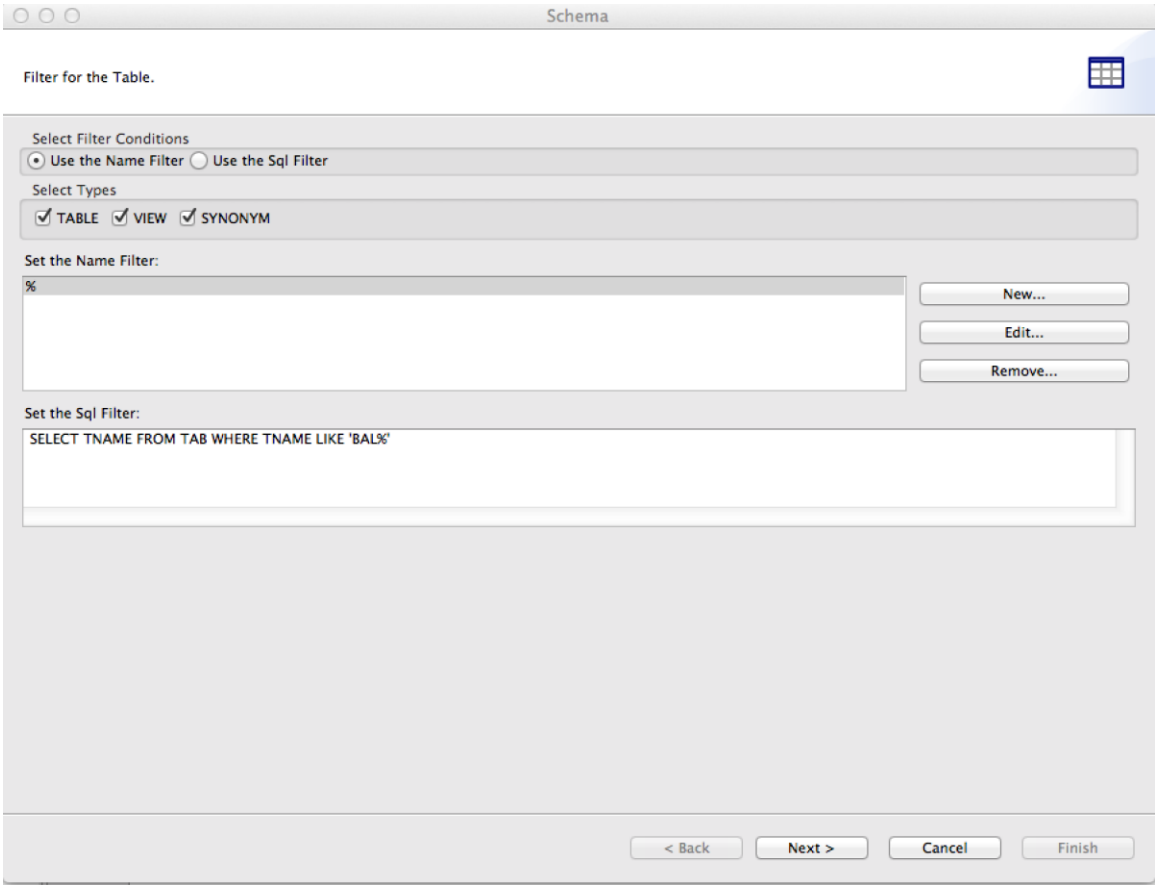


Figure 60: Retrieving the Database Schema

Then on the next page we want to select all the tables to import so click the check box at the highest level to select all and click Next.

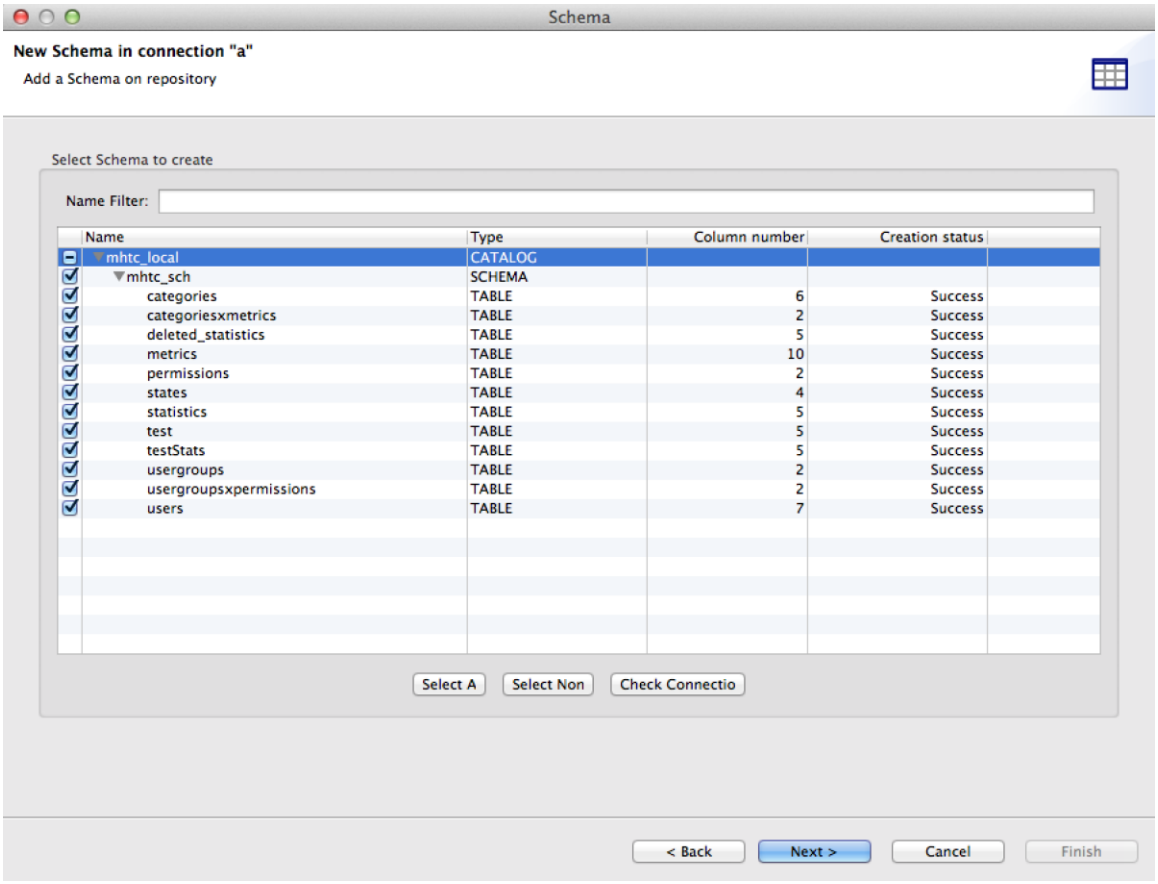


Figure 61: Selecting the Tables to Import

Then on the last pane just select finish unless you want to edit any of the existing schemas, which we do not in this tutorial. Now that we have the tables in Talend you want to drag in both the States table and the Metrics Table into our Job and set them as tPostgresqlInput and on both check their component property "Use an existing Connection"

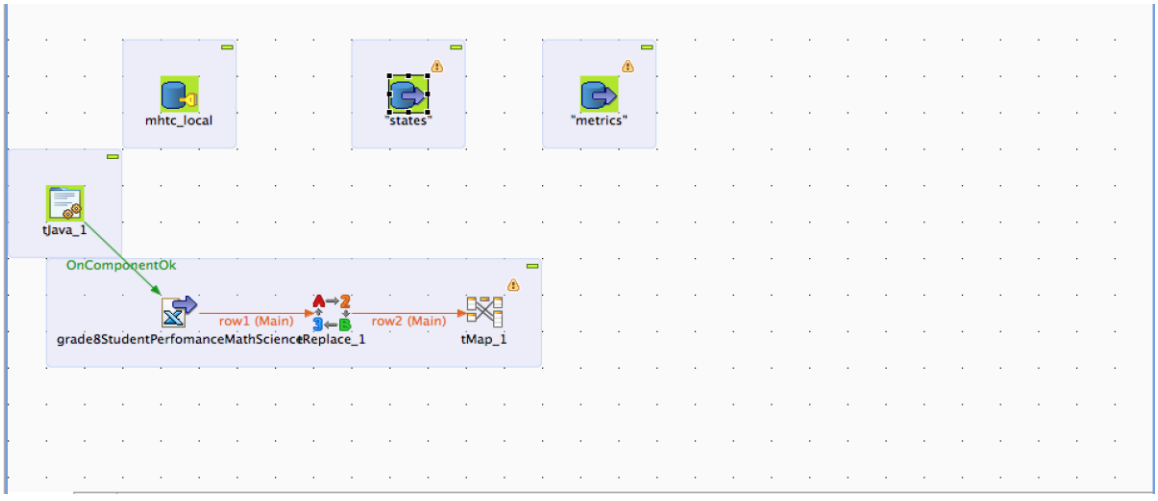


Figure 62: Adding Database Inputs

These inputs select all items in the table. Next right click each and connect them to the tMap to be used as lookups. We also want to ensure that these don't run before the database connection has been made so right click the tPostgresSqlConnection hover over trigger and select onSubJobOk and connect that to the states input.

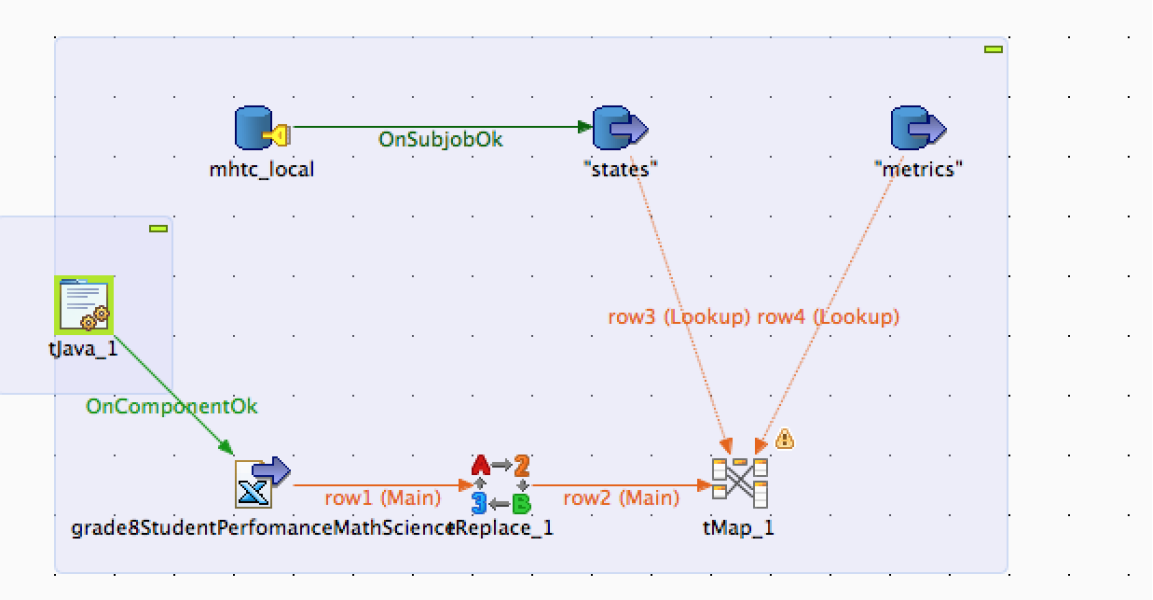


Figure 63: Connecting the Database Components

Next open up the tMap again so we can add our final two elements to the Output. The first thing we need to do is associate the state id with the State. Click on the state and drag it to match the Abbreviation field in your states input. This creates a link between The State input from the excel sheet and the state names in the database. Next drag the ID from the state input to the output.

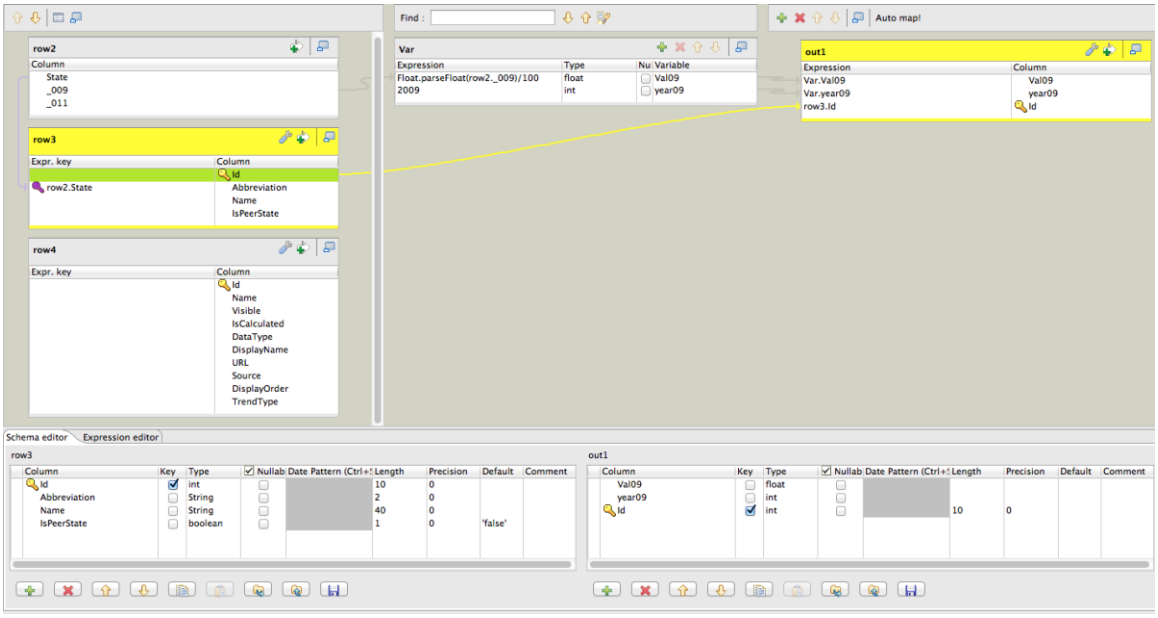


Figure 64: Mapping the State Names to IDs

Now we only need the metricId to get this we need to know what our metric is. For this you can go to the admin panel and navigate the database to find what metric is associated with the source you using. The metric should match the column name you are using. Once we have the metric name we can then create a new variable to add it. For this Variable we are going to make it an int and use the expression:

```
Row4.Name.equals("Proficiency in 8th Grade Science ") ? row4.Id :  0
```

This expression checks to see if the name equals Proficiency in 8th Grade Science and lets that id pass through otherwise it sets the id to 0. Once you have this drag the new variable into our output. Now our output table is complete and is almost ready to be inserted into the database.

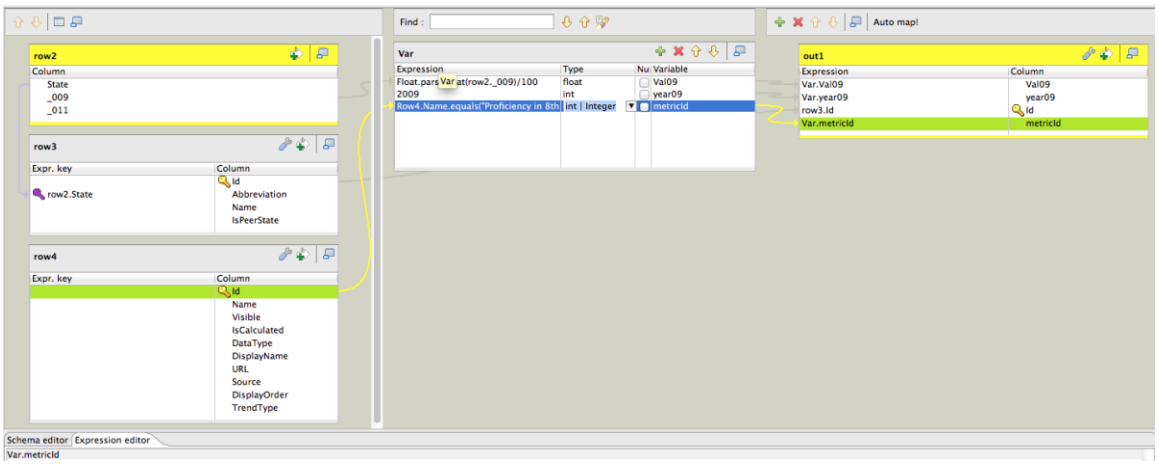


Figure 65: Getting the Metric ID

Now that we have our mapping done we just need to remove a bit more. When we mapped the metric id we brought in every metric id except anyone that isn't minimum rate had their id set to 0 so we need to remove all of these before inserting into the database. To do so we are going to add another tFilterRow component. Once you add it to the job you will then want to connect the tMap row minRate (the output we created) to the new tFilterRow.

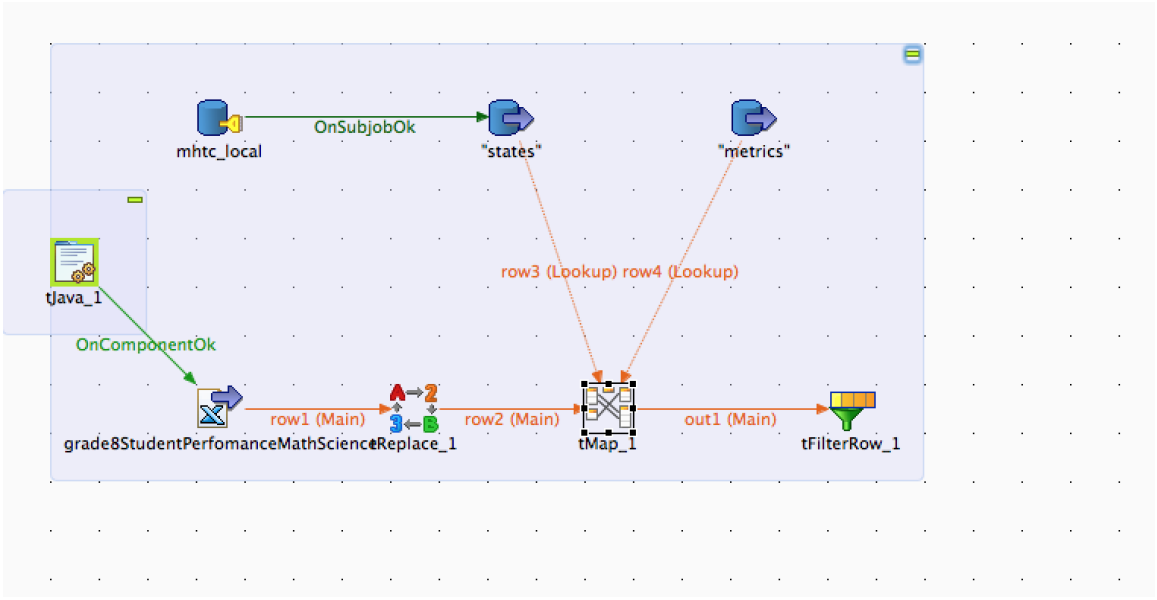


Figure 66: Adding the tFilterRow

On the filter you want to add a new condition and set the properties of it to those seen in the picture below.

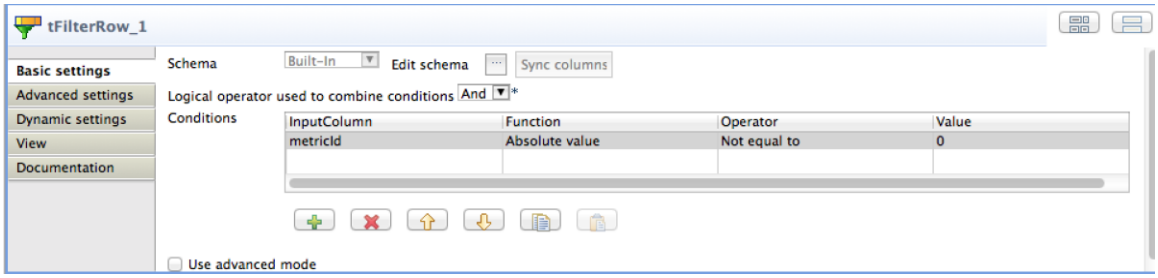


Figure 67: Component Pane of the tFilterRow

The tFilterRow reads in each row and checks it against a set of rules that are defined in the component pane. In this example we remove anything with a metricId of 0. It then filters out anything that does not meet the criteria giving us our final data ready to be inserted into the database.

1.9 Inserting to the Database

Now we are ready to add our data to the database. First we are going to need to add the component tPostgresqlOutput to our job, which can be found in the Palette under databases in the PostgreSql section. Once the component is in the job connect the filter row to the tPostgresqlOutput and on the tPostgresqlOutput select use an existing connection on the component tab.

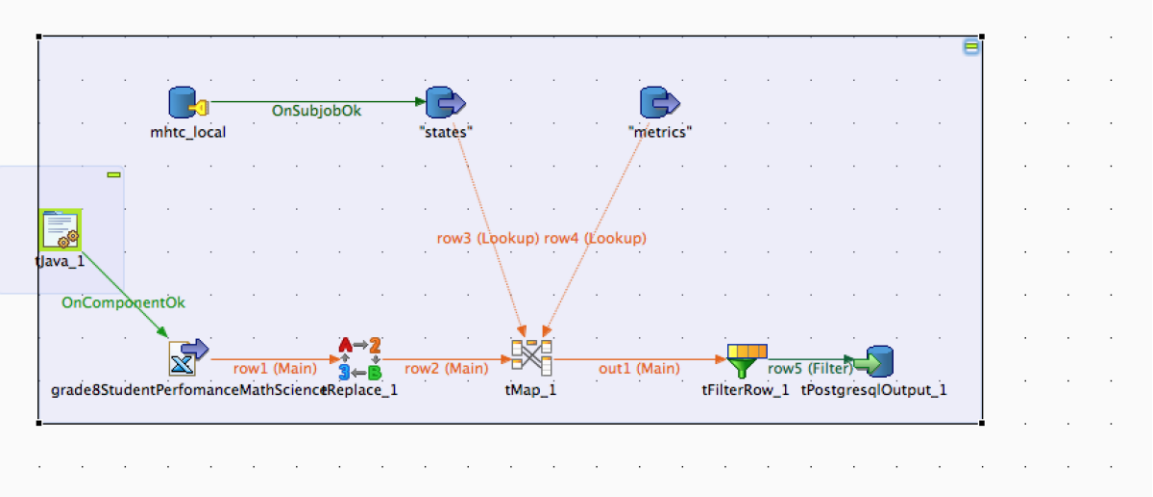


Figure 68: Adding the Database Output

Next on the component tab we need to set the table property to be the statistics table. Click on the button next to the table field and browse to the statistics table. Also make sure to set the Action on data field to "Insert or update" Then we need to edit the schema to make sure it aligns with the database. Click on edit schema and make sure that you edit yours to match the db columns such as in the picture below.

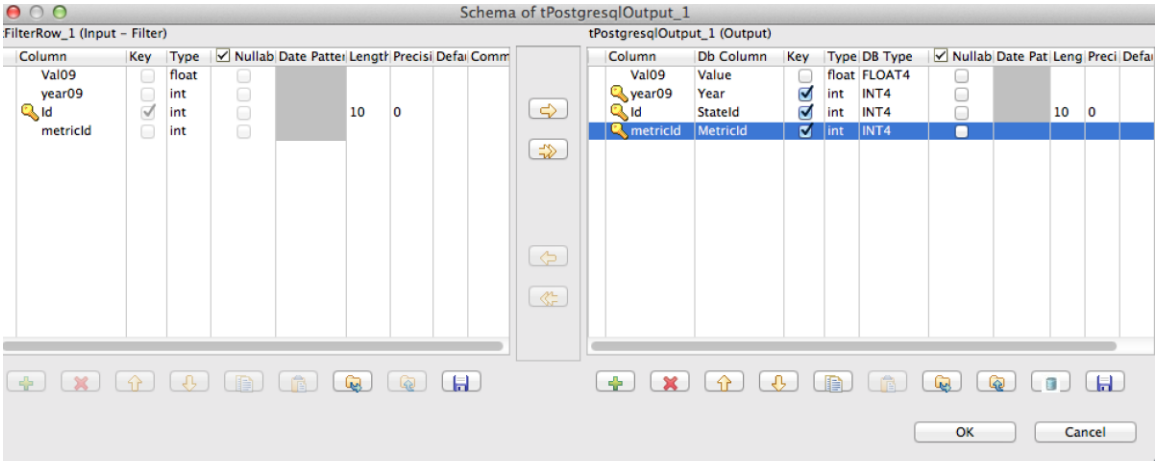


Figure 69: Schema to match the Output to the Database

Finally add the component tPostgresqlCommit to commit these changes to the database. On the component make sure close connection is unchecked. Connect the output to the commit component by using the OnComponentOk trigger. In a pipeline you will only need one commit component and you should attach it to the last output you have in order of execution. So if you had two outputs branch it off the second one and both will be committed. In the picture below we also added a tLogRow to the end in order to see what the output is that we are inserting into the database. At this point if you hit run you should be able to insert your data into the database.

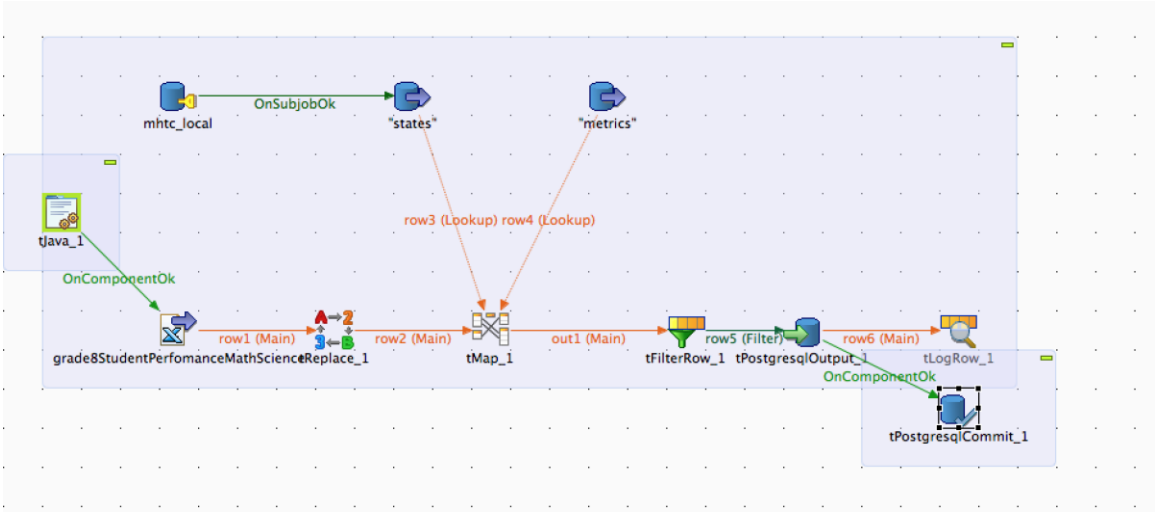


Figure 70: Committing to the Database

In order to ensure that all of the data gets entered into the database for the job and not just some of it you will need to add one last component. To ensure this add a tPostgresqlRollback component to the pipeline and attach all tPostgresqlOutputs to it with the OnComponentError Trigger. This will make sure that if one of the outputs fails to load the data none of the data from the current transaction will be entered.

1.10 Exporting

At this point you have done everything you need to take data from a file and put it into the database. The next steps after this are to add logging to the pipeline (view the logging

documentation found in section 4.0) and to add the other output to the pipeline. Once done with that it should look something like the image below:

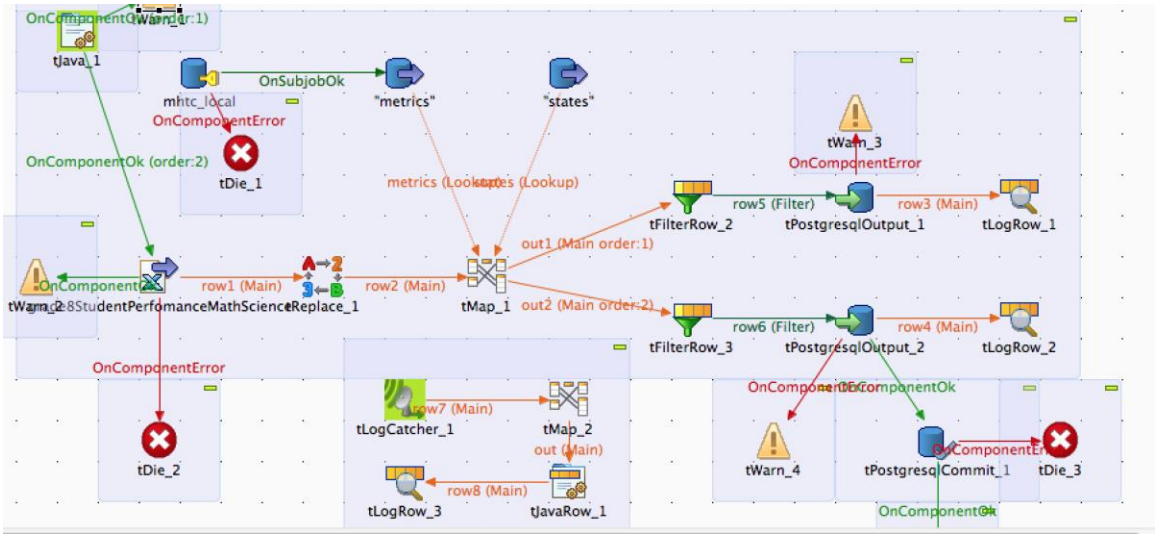


Figure 71: Finished Pipeline

Next once you job is all set and done you will need to export it in order for it to be uploaded to the admin tool to do so navigate to the repository pane and right click on the job and select build job:

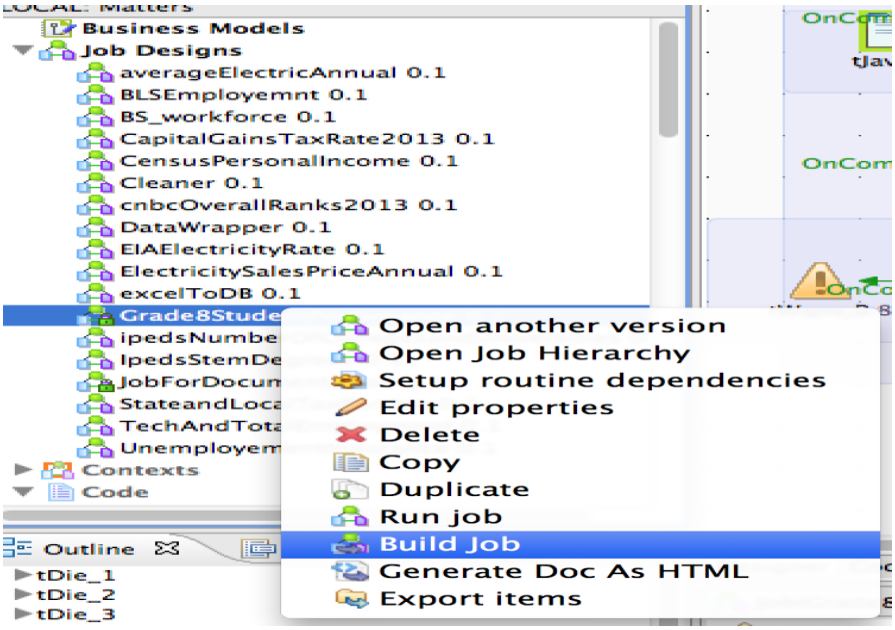


Figure 72: Building the Job to Export

On the next screen you want to make sure that it is set to standalone job and then save it to the location of your preference. The default format is a zip file however you may select extract the zip file, which will extract it for you. Then hit finish and it will create an executable

to run your job, which you can do so by typing ./<jobname> where job name is the name of the executable.

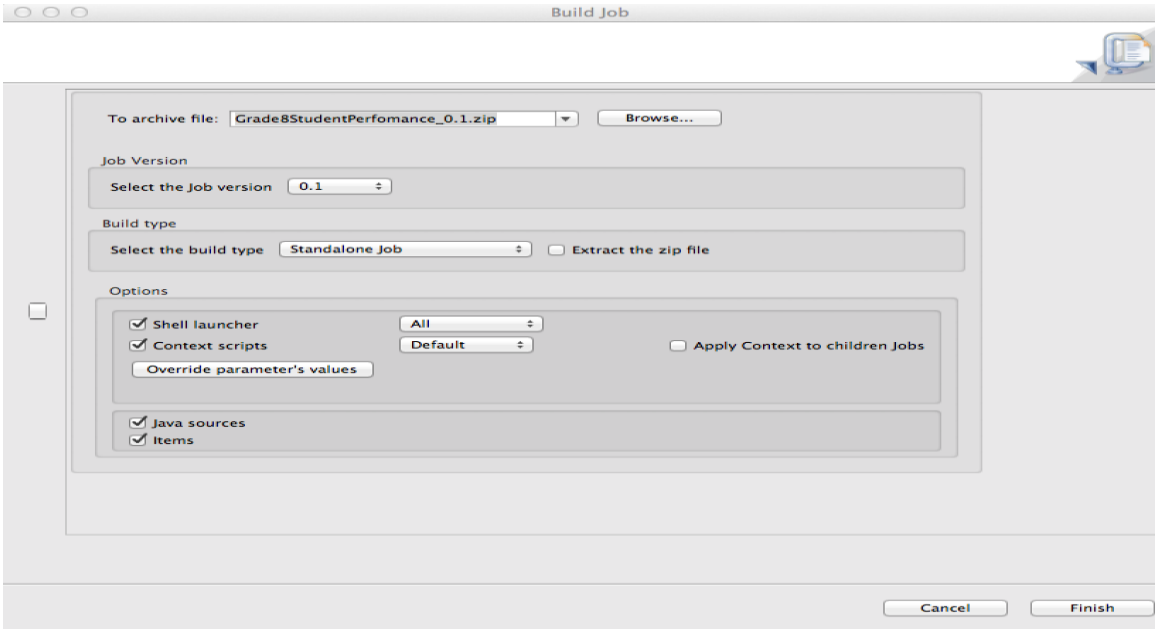


Figure 73: Build Job Wizard

1.10.1 Running with context variables.

Sometimes in a job you will have a context variable, this is a variable that is passed in when you try to run the job either as an input box from talend or as a command line argument. The way to run a job with passing in context is to run the command like so:

context_demo_run.bat **--context_param** ***<param-name>*=*<param-value>***

where context_demo_run.bat is an example script name to run the job param-name is the name of the parameter and param-value is the value you pass into that parameter

2.0 Moving From Development to Production:

1. **The File Source (Only if a wrapper is not used)**

   On most pipelines the current location of the file is set to a local path on a users machine. This will need to change to where the file will be uploaded to on the server or if possible a download URL. Below you can see an input excel component where the file name will need to be changed.

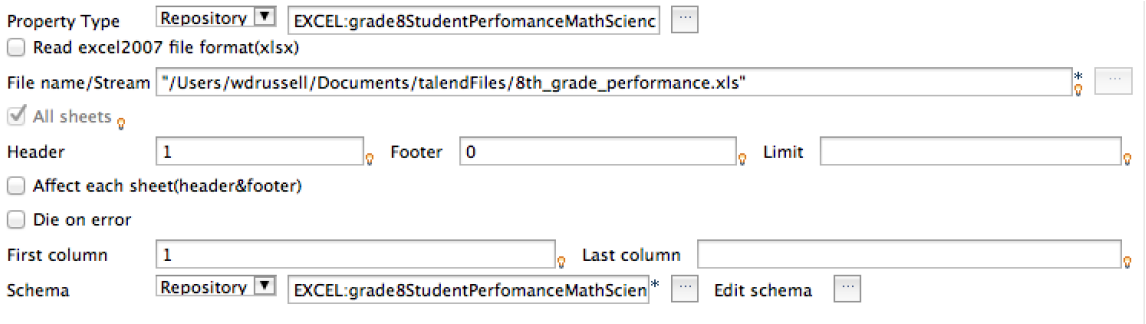


Figure 74: File Input Component Pane

2. **The database connection and anything that pulls from the database**

   Currently we are using local test databases for the jobs. These will need to be changed in each job to the appropriate production database connection. Below is the component for the tpostgreSQL_Connection component that needs to be changed to hold the appropriate information.

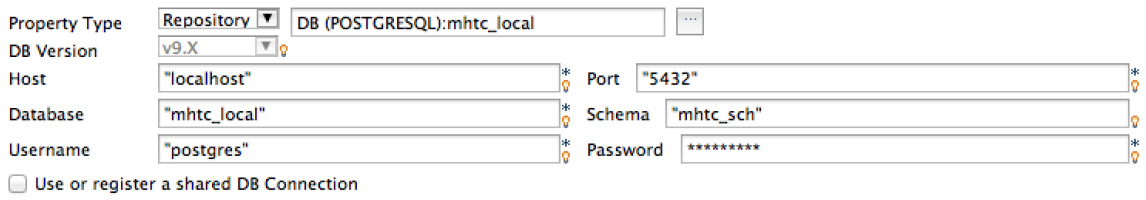


Figure 75: Database Connection Component Pane

This next image is of the component for a tPostgreSQLInput component. This holds a query to the database that will need to be changed to match the production databases name.

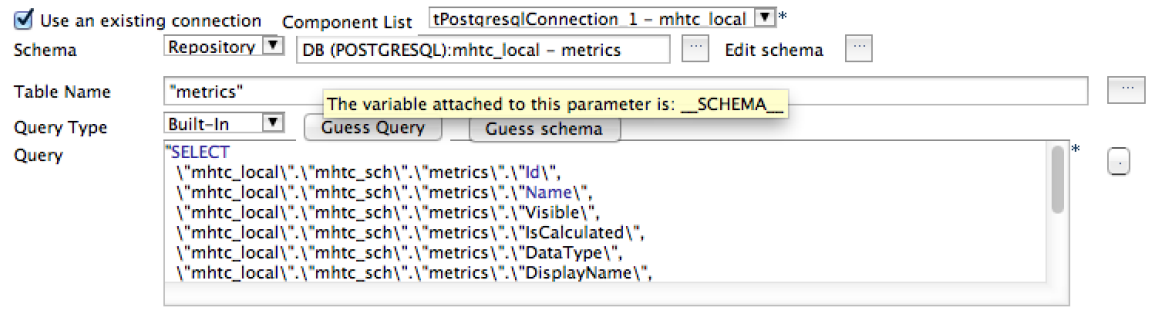


Figure 76: Example of Database Input Component Pane

You also need to ensure that all database components have their Use an existing connection checkbox checked as seen above and that that connection is for the production database. You can double check this by looking in the query pane seen above and make sure that the query is using your database.

3. The output and input file for Logging.

During Logging we use a tJavaRow component. This component calls a routine we created to send data in an http message (more information on routines can be found in the routine documentation). For the tJavaRow you will need to change the URL to that of the route for logging on the mhtc website. Below you can see an image of the component. The first String is the URL and the following variables (which don't need to be changed) are the information that is sent in the message to the website.

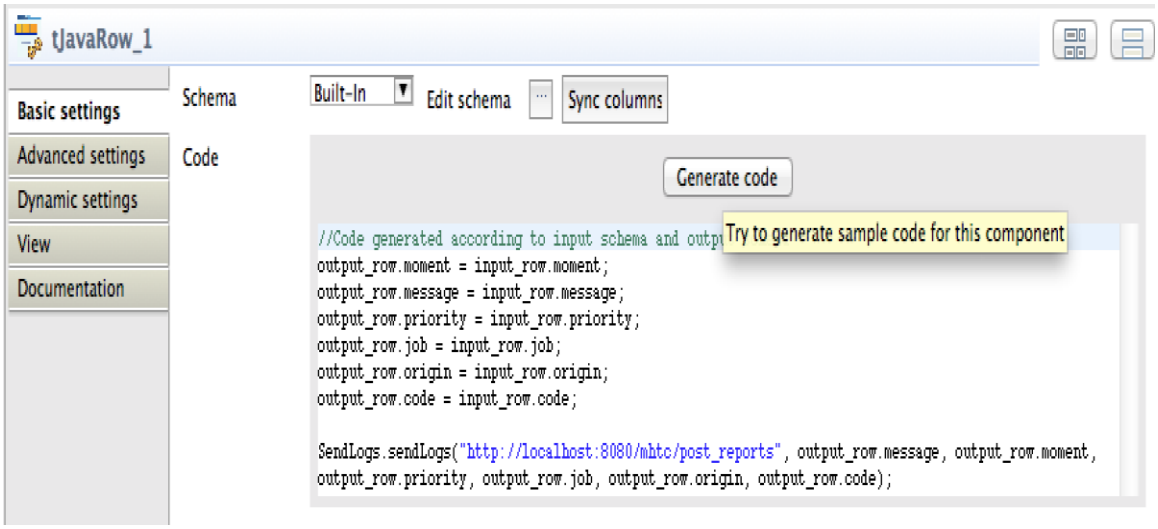


Figure 77: tJavaRow Component Pane

4. The metric names in t_map components.

When testing on your local database your metric names may not match up to that of those in the production server all the time. If this is the case then you will need to ensure that everywhere you get a metric id in the tmap to make sure that it is the proper name of the metric in the production database. Below you can see an image of the t_map trying to match the metric id based upon the name.

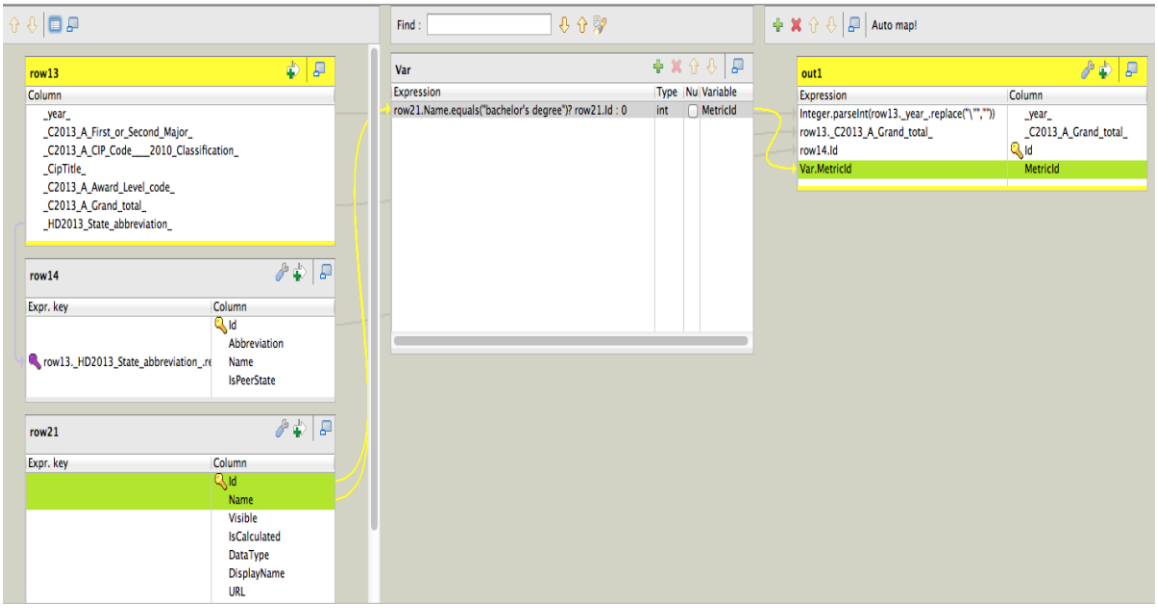


Figure 78: Example of Inside of a tMap

This is the inside of a t_map. In order to get the metric id we check to see if the metric has the name we want in the variable column and if it is the appropriate metric then we send its id to the output table. Below is a closer look at the variable table to see the expression. The part that needs to be changed is the string in parenthesis after the equals. Make sure that your metric name is surrounded by quotes in the expression.

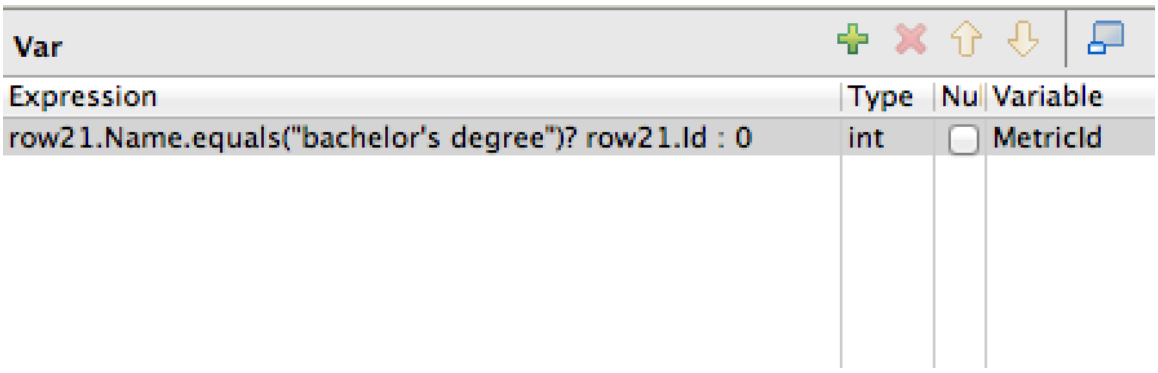


Figure 79: Close up of Variable Part of tMap

3.0 Talend Routines

3.1 Overview

Talend Routines are a way to write java code once, which can then be called many times from within the same job or different jobs. In our pipelines we use routines to send the post messages with the logs and routines are used for the data wrappers in order to download the files. This document will give you an overview of our routines for more information on routines visit: http://www.talendbyexample.com/talend-code-routines-reference.html

Routines can be found in the repository pane under code as seen below

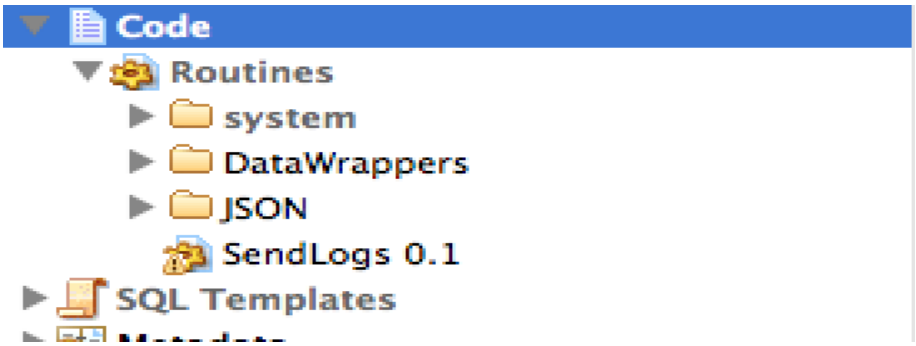


Figure 80: Close up of Code Section of Repository Pane

In order to create a new routine you can right click on routines and select create new routine and from there you can edit the file to create your functions to be called in your job.

3.2 Routines

3.2.1 Logging

Currently we have two purposes for routines. First is for logging with the SendLogs routine. This routine is used to take in all the information for a log message as well as a website and send it to the website for the logs to be displayed there. Below you can see the routine as an example of what it looks like. Routines are called in the job by using any of the custom components. From inside the code window you call the routine by the class name then the function name. So for the send logs routine you would type:

```
SendLogs.sendLogs("http://localhost:8080/mhtc/post_reports", output_row.message,
output_row.moment, output_row.priority, output_row.job, output_row.origin,
output_row.code);
```

This calls the routine sendLogs from the SendLogs class with the appropriate variables. More information on Logging can be found in section 4.0.

```
24 public class SendLogs {
25
26     /**
27      * sendLogsFromFile
28      * @param webAddress the url you are sending the log messages to
29      * @param fileLocation the location of the file the logs were written to
30      * @throws IOException
31      *
32      * This function takes in an url and a file location and then reads the file, takes the data and puts it into a
33      * http post message that is then sent to the specified url.
34      */
35
36
37     public static void sendLogs(String address, String message, Date moment, int priority, String job, String origin, int code)
38         URL url = new URL(address);
39         Map<String,Object> params = new LinkedHashMap<String, Object>();
40         params.put("moment", moment.toString());
41         params.put("message", message);
42         params.put("priority", priority);
43         params.put("job", job);
44         params.put("origin", origin);
45         params.put("code", code);
46
47         StringBuilder postData = new StringBuilder();
48         for (Map.Entry<String,Object> param : params.entrySet()) {
49             if (postData.length() != 0){
50                 postData.append('&');
51             }
52             postData.append(URLEncoder.encode(param.getKey(), "UTF-8"));
53             postData.append('=');
54             postData.append(URLEncoder.encode(String.valueOf(param.getValue()), "UTF-8"));
55         }
56         byte[] postDataBytes = postData.toString().getBytes("UTF-8");
57
58         HttpURLConnection conn = (HttpURLConnection)url.openConnection();
59         conn.setRequestMethod("POST");
60         conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
61         conn.setRequestProperty("Content-Length", String.valueOf(postDataBytes.length));
62         conn.setDoOutput(true);
63         conn.getOutputStream().write(postDataBytes);
64
65         Reader in = new BufferedReader(new InputStreamReader(conn.getInputStream(), "UTF-8"));
66         for (int c; (c = in.read()) >= 0; System.out.print((char)c));
67     }
```

Figure 81: SendLogs Routine

3.2.2 Data Wrappers

The folder labeled datawrappers contains all the routines required to download the files from various sites in order to run these pipelines. If a new wrapper is added to the code base it should also be added in talend to ensure that all wrappers are there for the pipelines. The main Class is DataWrapperMain. This has all the functions to run the various wrappers. These wrappers do make use of external jars that talend does not have. These jars need to be added to your version of Talend by opening the module view and clicking the add external jars button. The jars required can be found in the repository under documentation in the folder jarfiles. Once these are added to your Talend application your routines will work fine.

3.2.3 Adding to Pipeline

In order to get your routines to work in a pipeline however there is one extra step you must take. In the repository pane you need to right click the pipeline and select edit routine dependencies as seen below

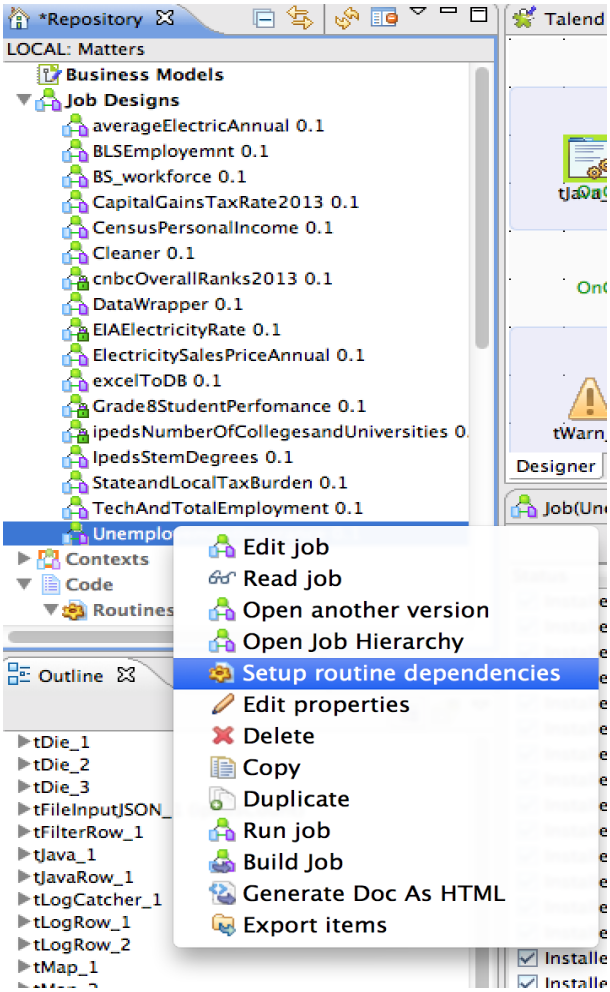


Figure 82: How to Add Routines to Pipeline

Once here you are going to want to click the green plus sign and add all the routines you will use in the pipeline. For the data wrapper you need to include main and routine that main uses. If you just want to be safe you can add them all. Then click ok and you are now ready to run your pipeline with routines.

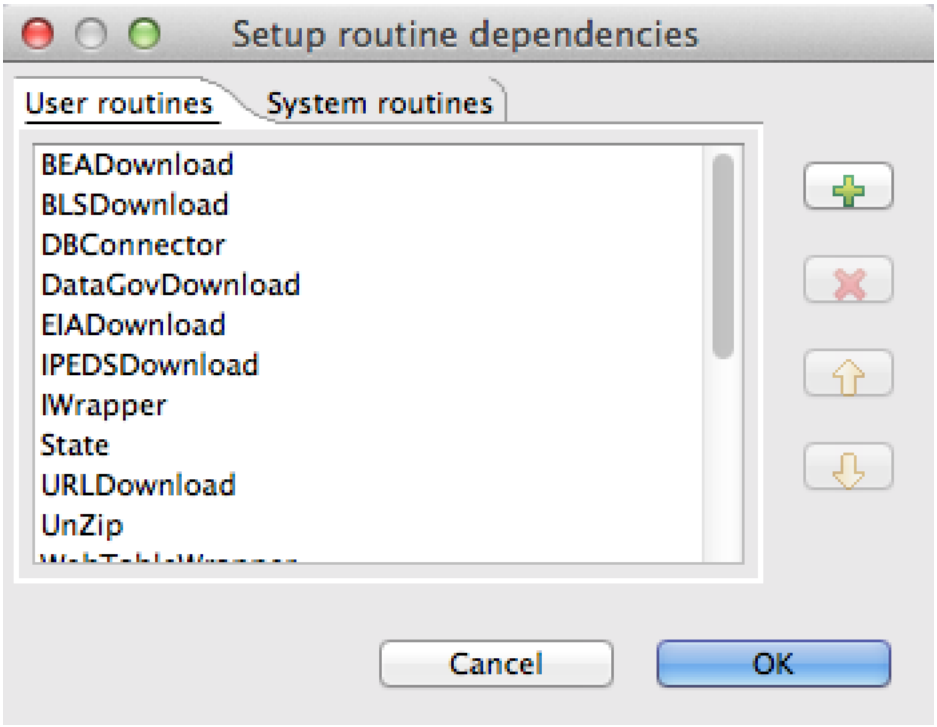


Figure 83: Routines to Be Added

4.0 Talend Logging

4.1 Overview:

For our logging purposes we use talend to create and send our own log messages tailored to our needs. To do so we add components such as tWarn and tDie to create messages whenever they are reached. Below you can see an example of a job with logging in place. (Each new component used for logging will be explained in a later section)

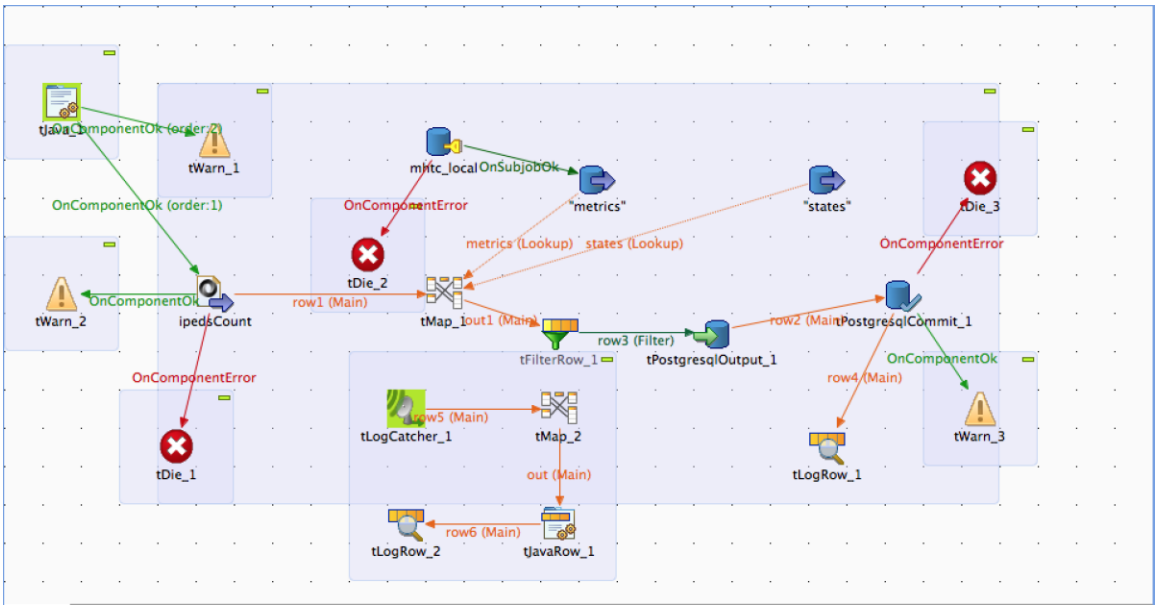


Figure 84: Example of a Pipeline with Logging

4.2 Creating the Messages

Here as you can see there are tWarn and tDie components coming off of certain other components. For the purpose of our logging we have one tWarn on the input file for saying the file was downloaded and one tWarn on the tJava component for saying the pipeline has started and one. The onComponentOk trigger triggers these. We also have tDie coming off that component where if the file can't be found the pipeline will die and send the message that the file could not be downloaded. The next tDie is for the database connection; this ends the pipeline if it cannot connect to the database. Finally there is the tDie on the commit which if the pipeline is unable to commit to the database it ends. There is also a tWarn where upon completion of the pipeline will create the message that the pipeline has finished. You can add as many tWarns and tDies as you wish depending on how much logging you want done on the pipeline but these are the recommended ones to be used. In the tWarn and tDie components there is a place where you can set the code in the component. Below you can see a table that displays what each code we have defined corresponds to.

Code	Message
1	Pipeline has Started
2	Pipeline has Finished

3	Successfully Downloaded File
4	Unable to Connect to Database
5	Unable to Commit to Database
6	Unable to Download File
7	Error in Database Output Component
8	Error in tMap Component

The codes 1, 2, 3, and 7 are used in tWarns and codes 4, 5, 6, and 8 are used in tDie components due to the severity of the message.

4.3 Catching and Sending the Messages

Next after you have all your tWarn and tDie components in place you need a way to catch these messages. This is done using the sub job at the bottom of the picture above. Below you can see a zoomed in version of that part of the job.

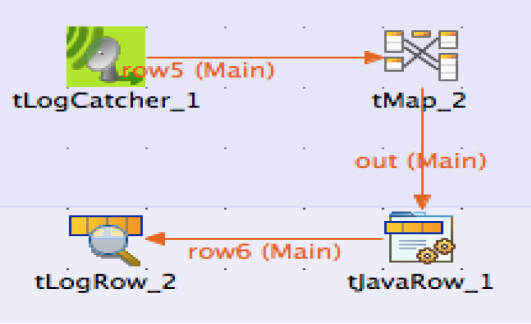


Figure 85: The Catcher and Sender Subjob for Logging

This part of the job is what catches all the log messages compiles them and then sends it to a server. The tLogCatcher sits and listens to any messages sent by tWarn or tDie components and then sends them into the tMap. The tMap structure can be seen below.

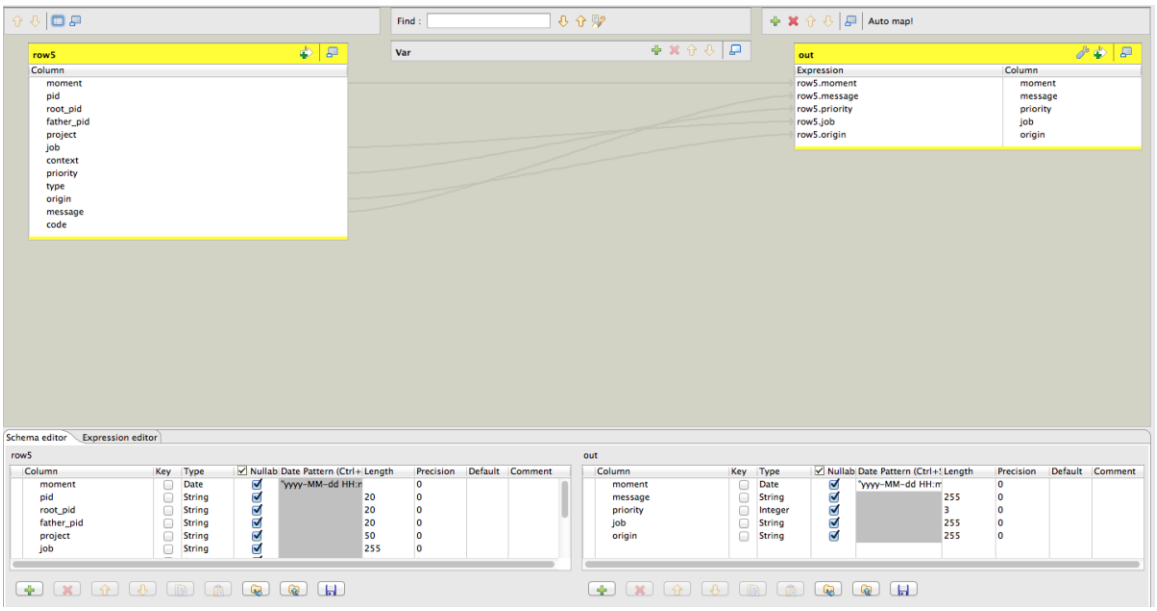


Figure 86: tMap for Logging

This map gives us control over the data we want from the log. For our purposes we only need the moment (the time the message was created), the message, the priority (IMPORTANT: This is the type of message being sent. 1 = Trace, 2= Debug, 3 = Info, 4 = Warning, 5 = Error, 6 = Fatal), the job it came from, the origin, which is what tWarn or tDie sent it and the code associated with the message which is declared in the tWarn or tDie component. This information is then sent into the tJavaRow component. This component calls a java routine that we have created to make an http post request to a specified URL and sends data from the tMap (More information on routines can be found in section 3.2). Below you can see the component information for the tJavaRow.

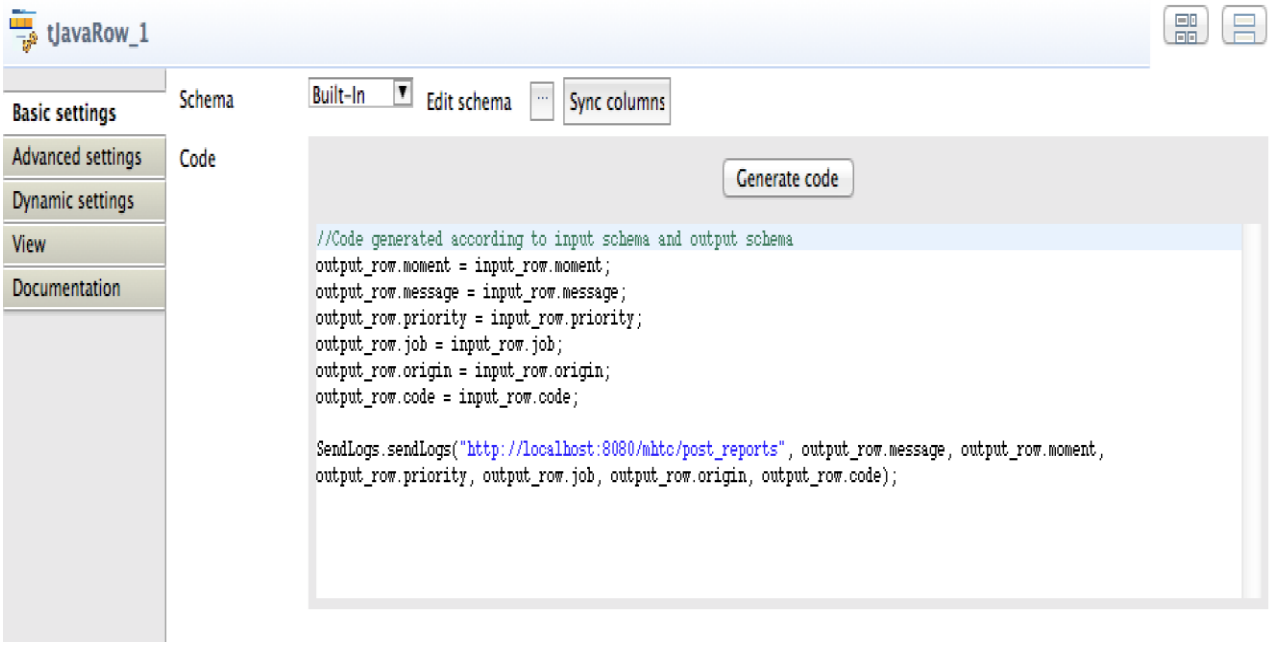


Figure 87: tJavaRow Component Pane

This calls the method sendLogsFromFile from the class SendLogs. The parameters are used for calling this are the location of the file and then each variable that was output from the t_map. The top half is the mapping of the variables from input to output and then the lower half is the call the sendLogs function. The order for the variables is URL, message, moment, priority, job, origin, and code. This sub job of logging (the tLogCatcher, tMap, tJavaRow, and tLogRow) can be copied and pasted into each new job because none of the information is job specific.

5.0 Talend Jobs

5.1 Overview

This section details each pipeline that has been created. Each section is a different pipeline. Each section will follow the pattern of first showing the tMap (more information on tMaps can be found in section 1.7) of the pipeline and then an image of the entire pipeline and will include any extra information that may be necessary to understand the pipeline. However the pipelines are very similar and really only differ in their cleaning of the file or if they use multiple files.

5.2 8th Grade Proficiency in Science

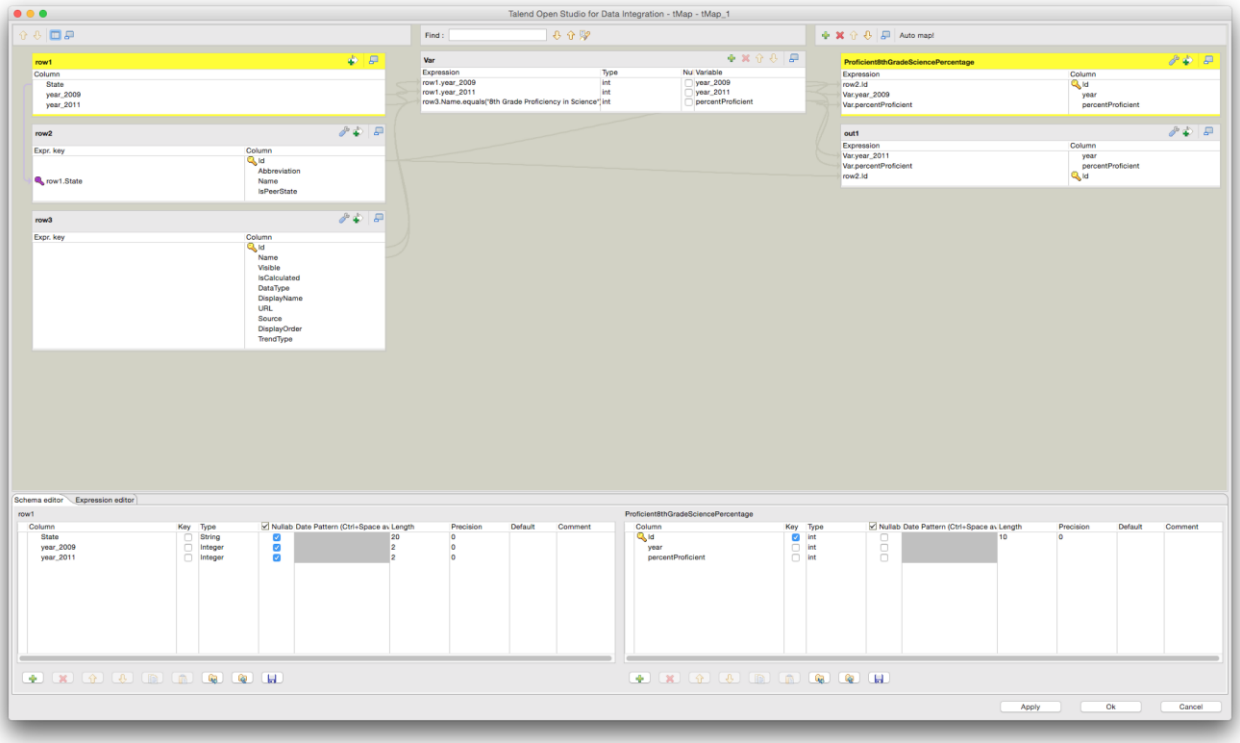


Figure 88 8th Grade Proficiency in Science tMap

Name: 8th Grade Proficiency in Science

Source: NSF

Columns: State, Year, 8th Grade Proficiency

row1 --

- excel spreadsheet being read in
- State being used as a foreign key for the State table from the database (row2)
- metric is being used in the output (Proficiency in 8th grade science)

row2 --

- state id is being used in the output

row3 --

- metric name is being looked at to see if it exists in database. If yes, then passing the metricID.

The above figure shows the tMap of the pipeline that maps the data from the excel spreadsheet to the database after looking up some information that is contained in the database. This lookup is required to make sure that the data being inserted is both correct and in the right format.

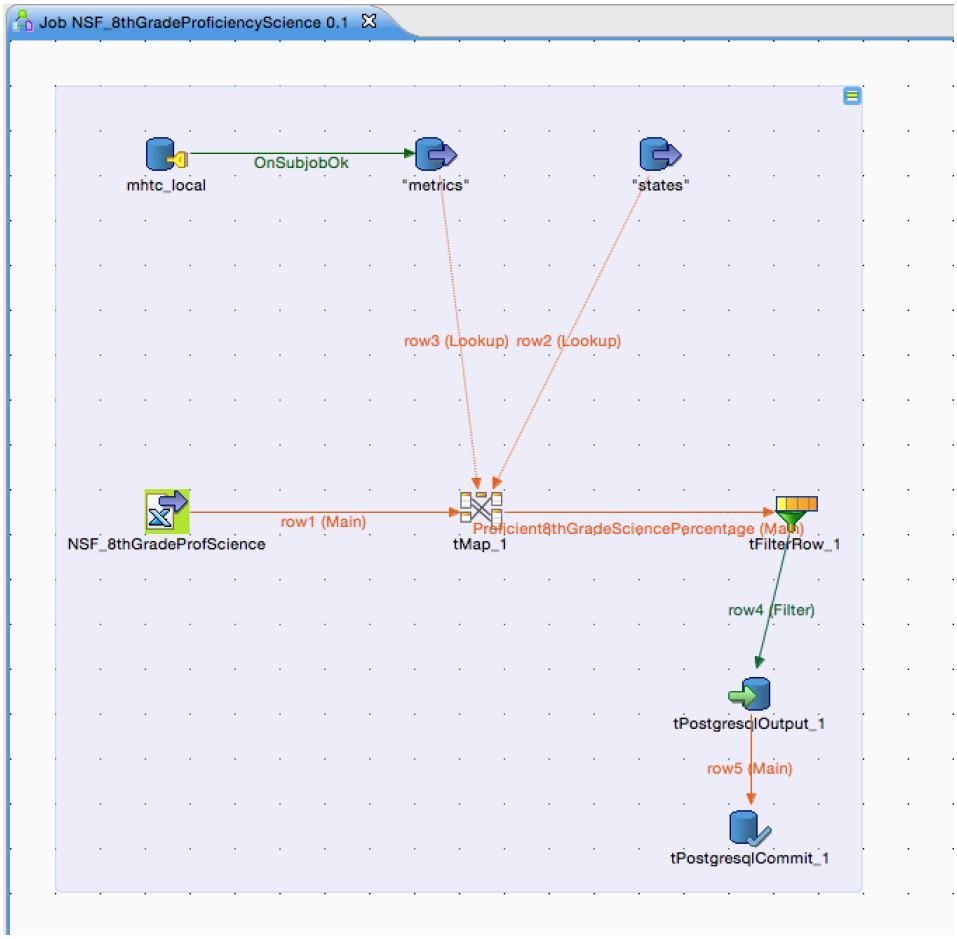


Figure 89: 8th Grade Proficiency In Science Pipeline

The above figure shows an idea of what the pipeline looks like. There is a database connection, which is used to do lookups on the states and metrics contained within the database. This is important to validate the data that is being added to the database. The tMap contains the physical mapping between the file and the database. tFilterRow is used to filter out some unnecessary information before inserting into the database.

5.3 Annual State Survey

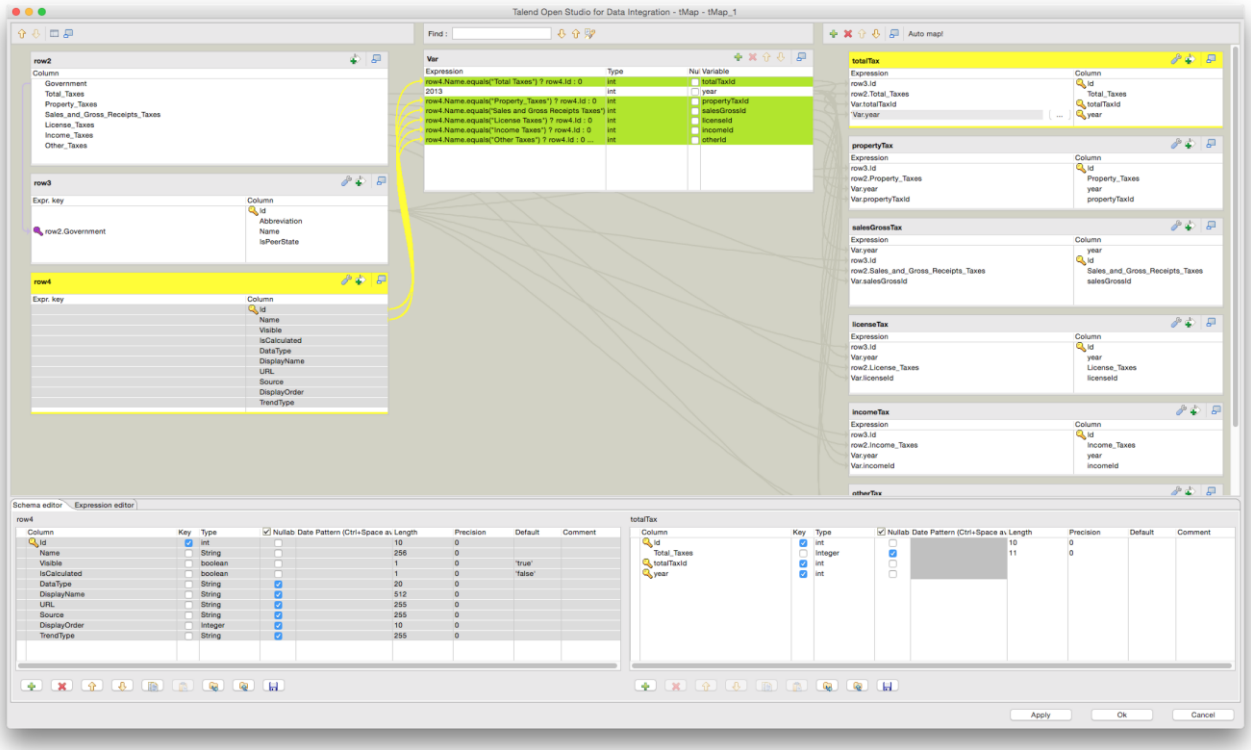


Figure 90: Annual State Survey tMap

The above Figure shows the tMap of the pipeline that maps the data from the excel spreadsheet to the database after looking up some information that is contained in the database. This lookup is required to make sure that the data being inserted is both correct and in the right format.

row2 --

- excel spreadsheet being read in
- State being used as a foreign key for the State table from the database (row2)
- metric is being used in the output (Total tax, Property tax, Gross Sales tax, License tax, income tax, and other tax)

row3 --

- state id is being used in the output

row4 --

- metric name is being looked at to see if it exists in database. If yes, then passing the metricID.

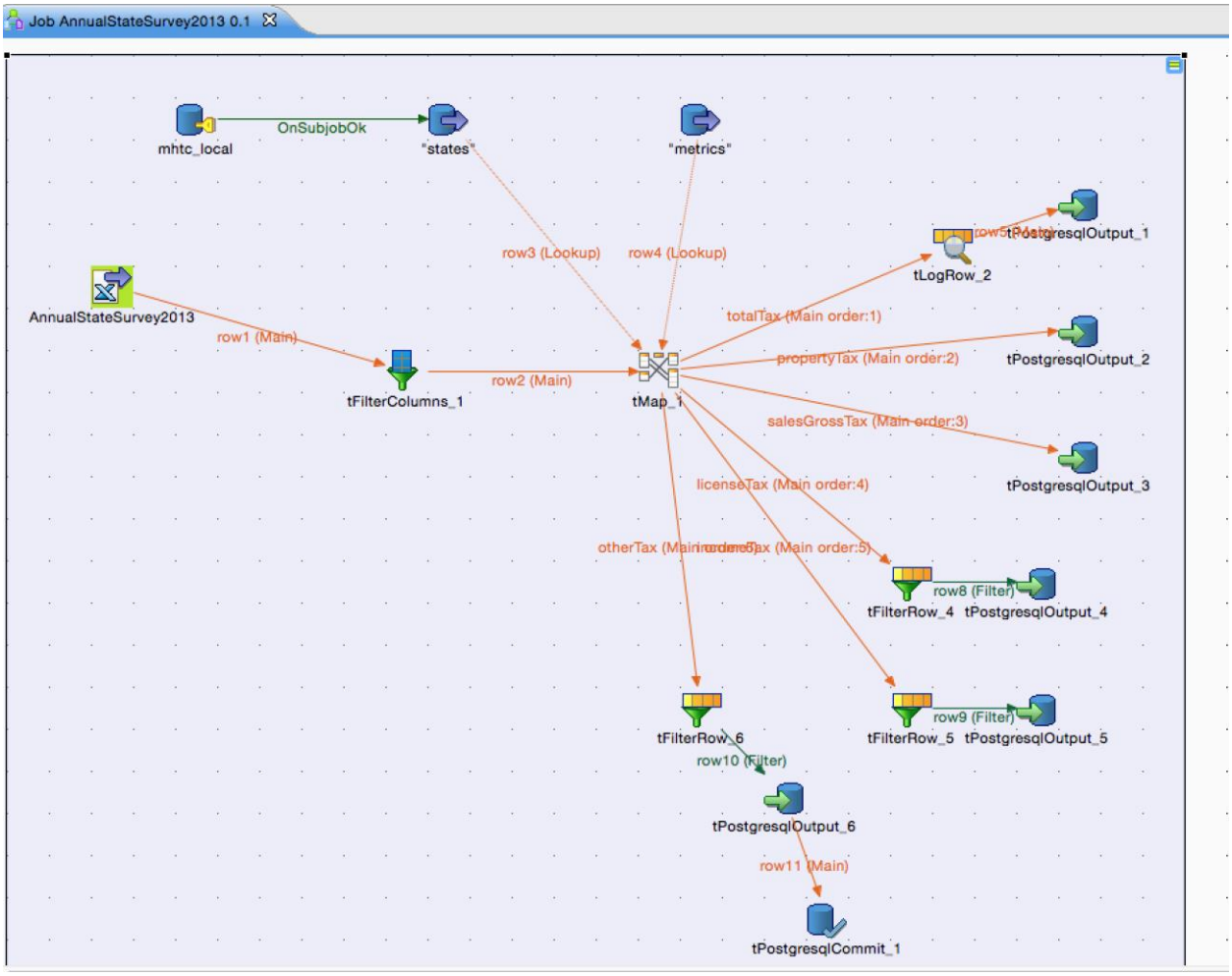


Figure 91: Annual State Survey Pipeline

The figure above shows an idea of what the pipeline looks like. There is a database connection, which is used to do lookups on the states and metrics contained within the database. This is important to validate the data that is being added to the database. The tMap contains the physical mapping between the file and the database. tFilterRow is used to filter out some unnecessary information before inserting into the database.

5.4 Average Electric Annual

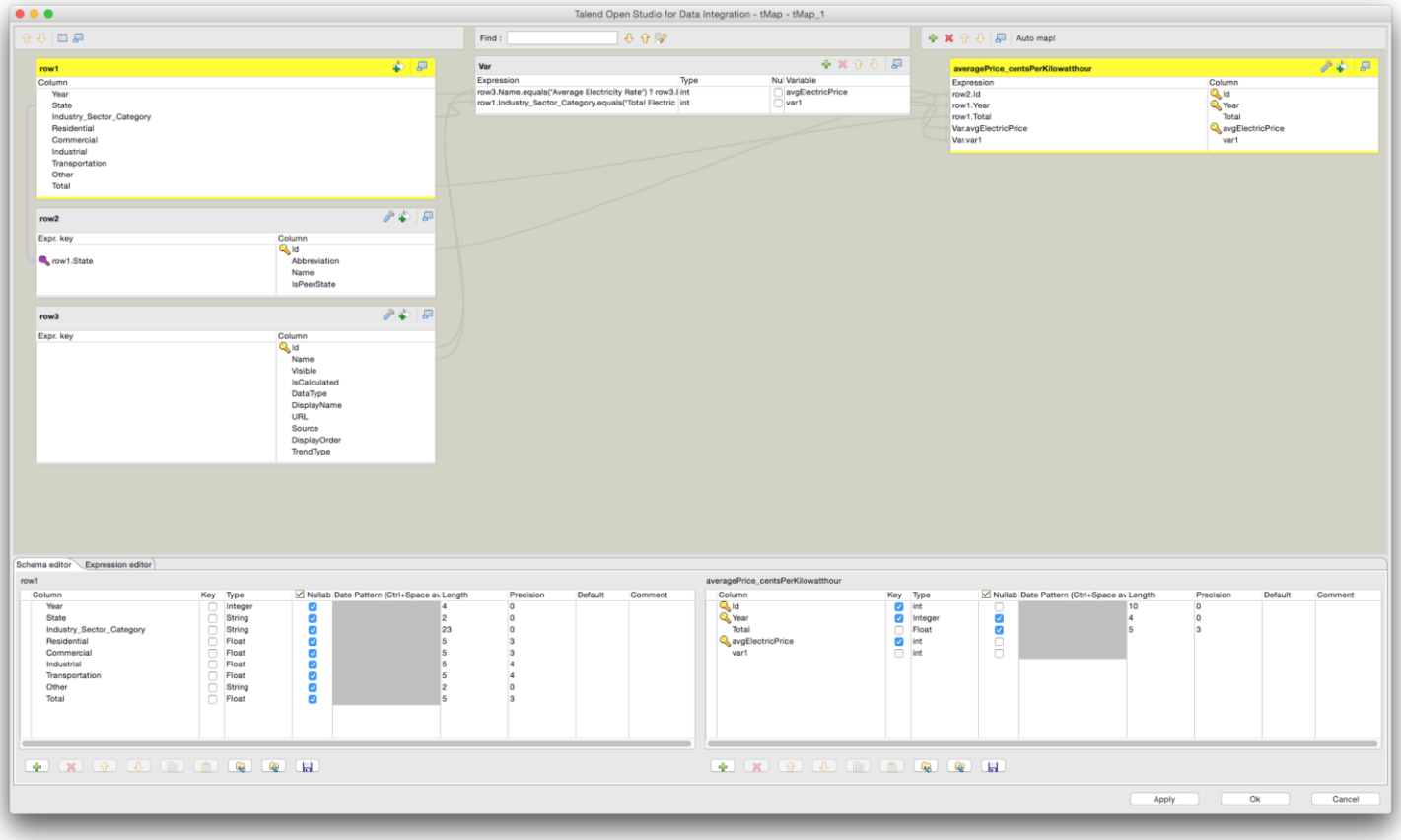


Figure 92: Average Electric Annual tMap

Name: Average Electric Annual

Source location: http://www.eia.gov/electricity/data/state/avgprice_annual.xls

Columns: State, Year, Total rate

row1 --

- excel spreadsheet being read in
- State being used as a foreign key for the State table from the database (row2)
- metric is being used in the output (Total Electric Sales average)

row2 --

- state id is being used in the output

row3 --

- metric name is being looked at to see if it exists in database. If yes, then passing the metricID.

The above figure shows the tMap of the pipeline that maps the data from the excel spreadsheet to the database after looking up some information that is contained in the database. This lookup is required to make sure that the data being inserted is both correct and in the right format.

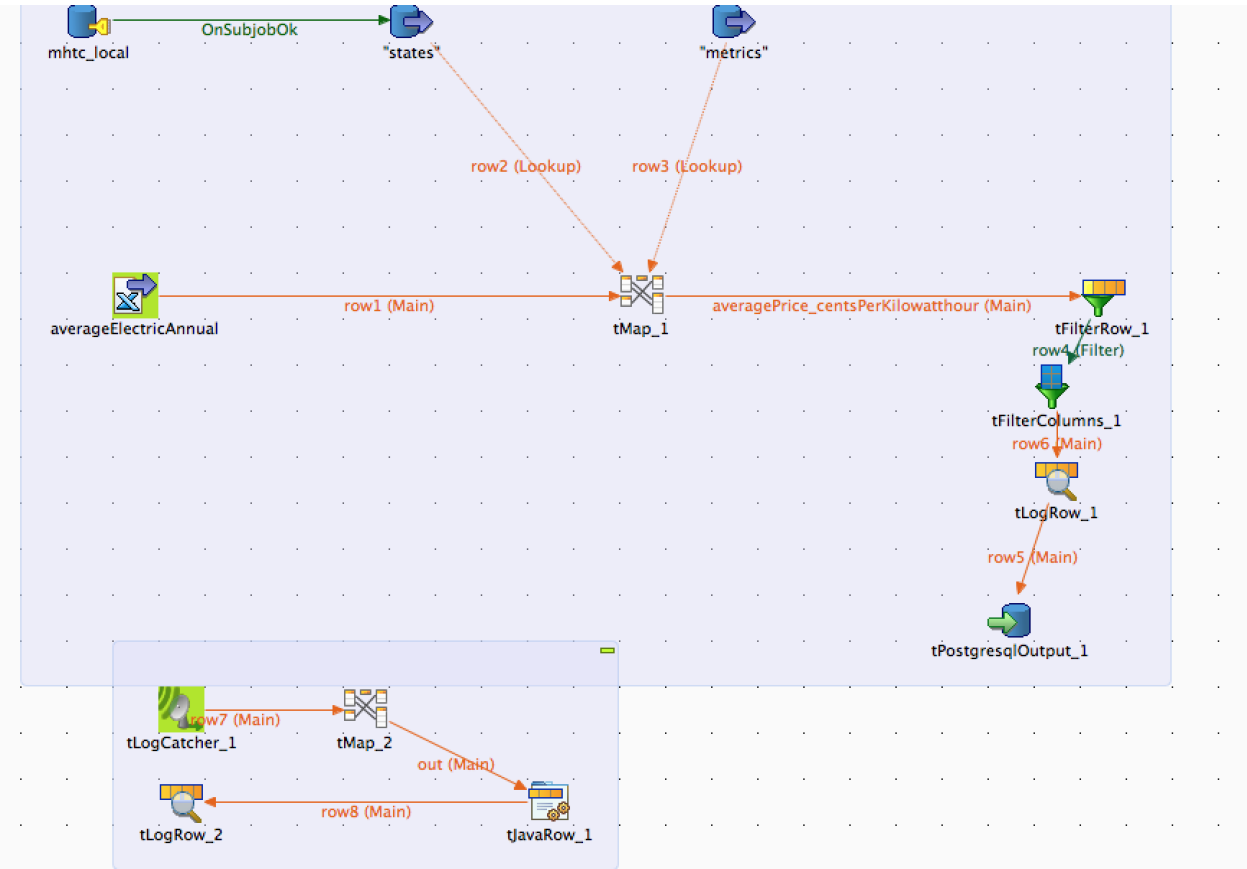


Figure 93: Average Electric Annual Pipeline

The above figure shows an idea of what the pipeline looks like. There is a database connection, which is used to do lookups on the states and metrics contained within the database. This is important to validate the data that is being added to the database. The tMap contains the physical mapping between the file and the database. tFilterRow is used to filter out some unnecessary information before inserting into the database.

5.5 BS Workforce

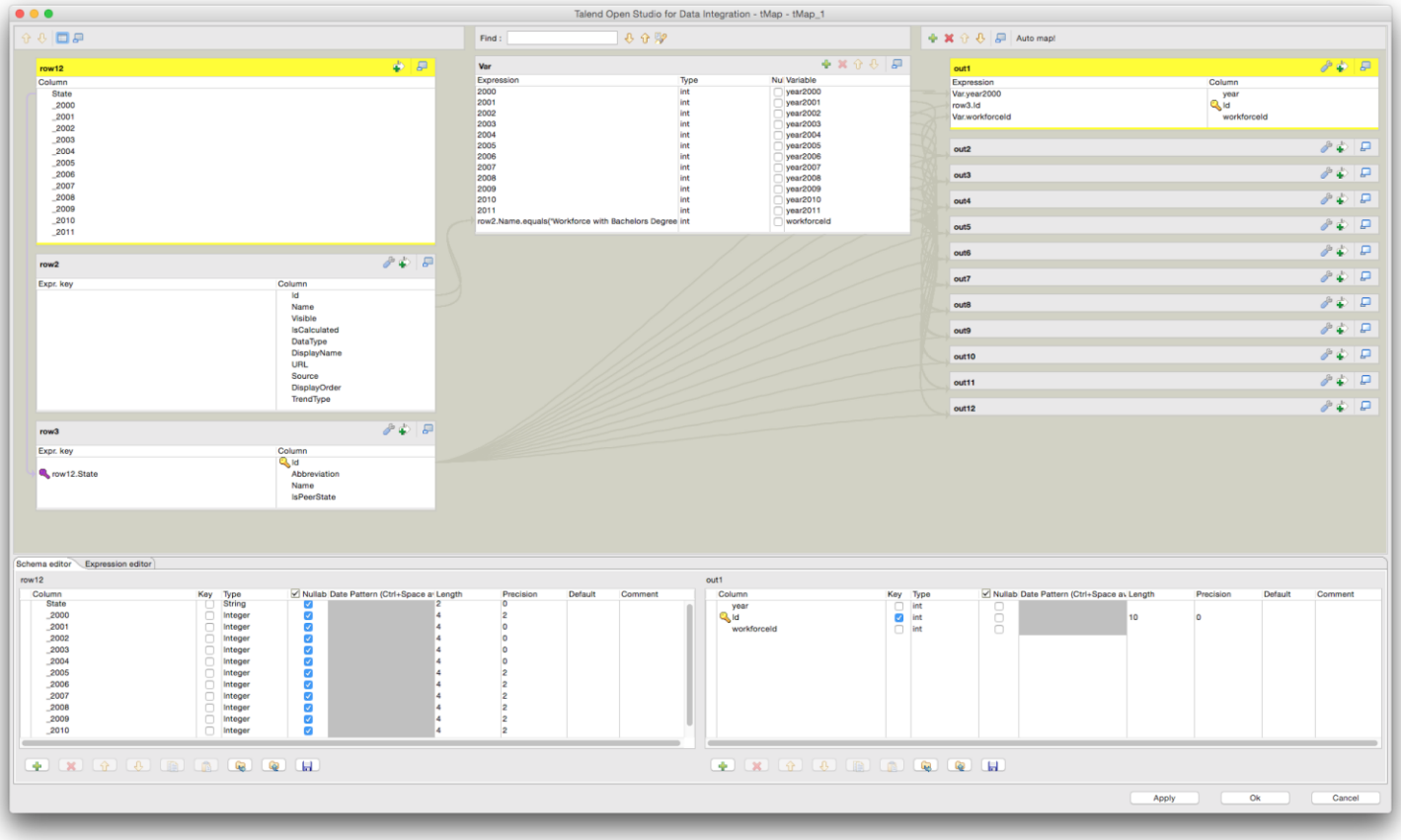


Figure 94: BS Workforce tMap

Name: B.S degrees in the workforce (1990-2011)

Source location: http://www.nsf.gov/statistics/seind14/index.cfm/state-data/download.htm

Columns needed: State, Year, BS in workforce

row1 --

- excel spreadsheet being read in
- State being used as a foreign key for the State table from the database (row2)
- metric is being used in the output (Workforce with Bachelors Degree)

row2 --

- state id is being used in the output

row3 --

- metric name is being looked at to see if it exists in database. If yes, then passing the metricID.

The above figure shows the tMap of the pipeline that maps the data from the excel spreadsheet to the database after looking up some information that is contained in the database. This lookup is required to make sure that the data being inserted is both correct and in the right format.

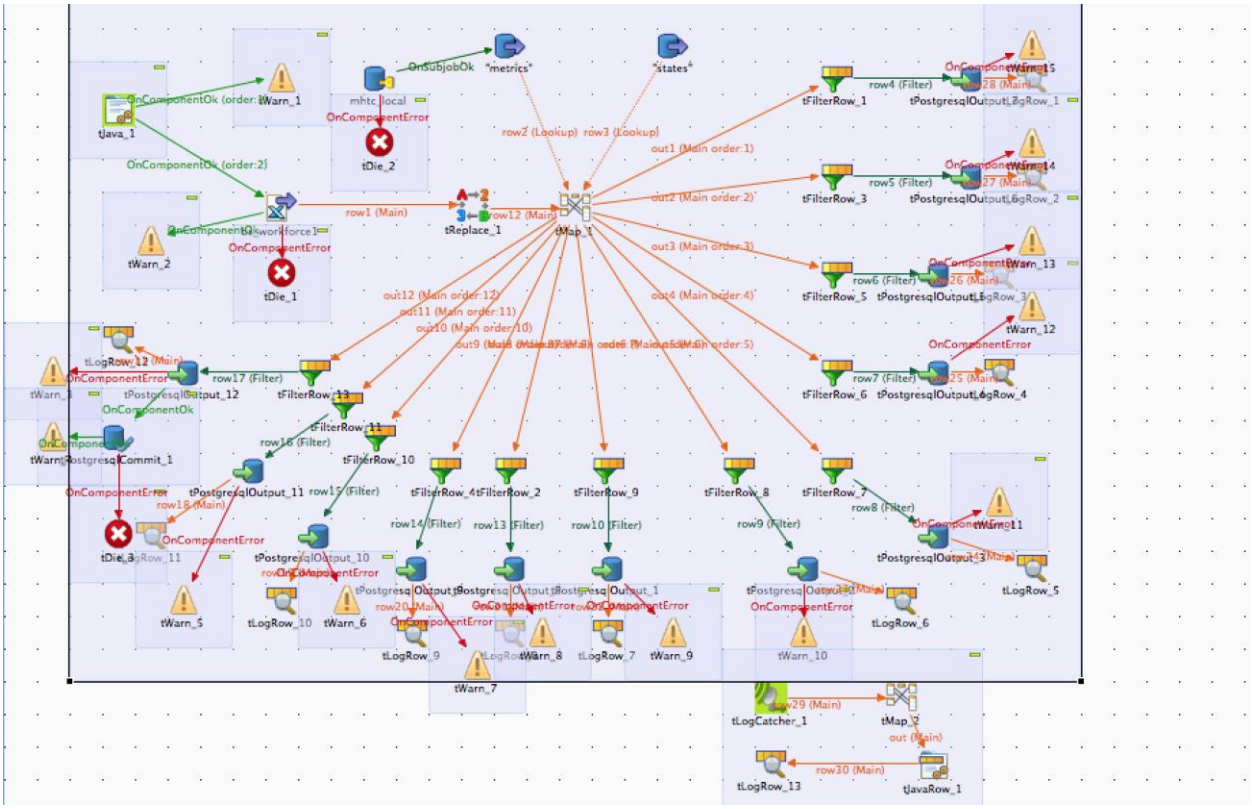


Figure 95: BS Workforce Pipeline

The above figure shows an idea of what the pipeline looks like. There is a database connection, which is used to do lookups on the states and metrics contained within the database. This is important to validate the data that is being added to the database. The tMap contains the physical mapping between the file and the database. tFilterRow is used to filter out some unnecessary information before inserting into the database.

5.6 Electricity Sales Price Annual

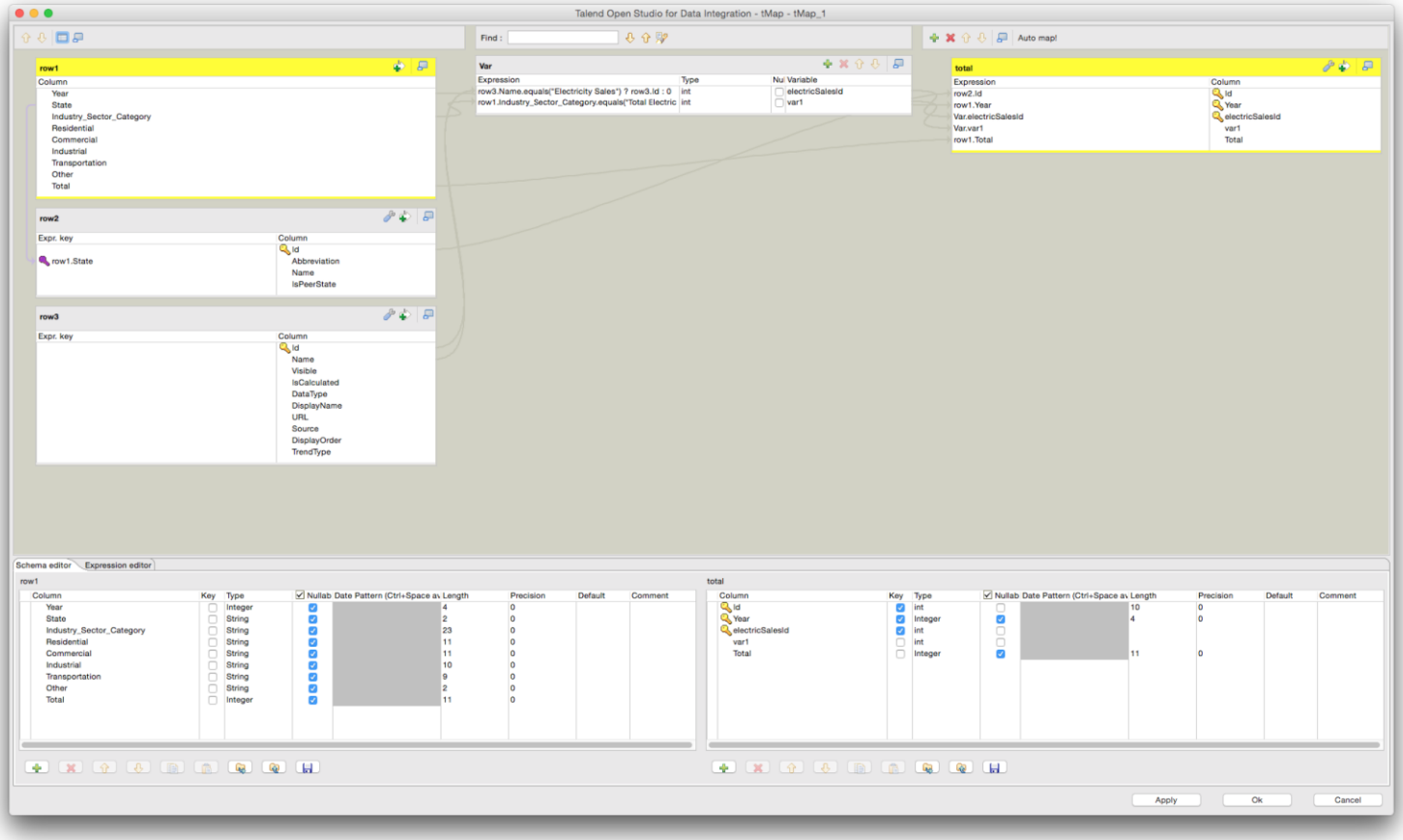


Figure 96: Electricity Sales Price Annual tMap

row1 --

- excel spreadsheet being read in
- State being used as a foreign key for the State table from the database (row2)
- metric is being used in the output (Total)

row2 --

- state id is being used in the output

row3 --

- metric name is being looked at to see if it exists in database. If yes, then passing the metricID.

The above figure shows the tMap of the pipeline that maps the data from the excel spreadsheet to the database after looking up some information that is contained in the database. This lookup is required to make sure that the data being inserted is both correct and in the right format.

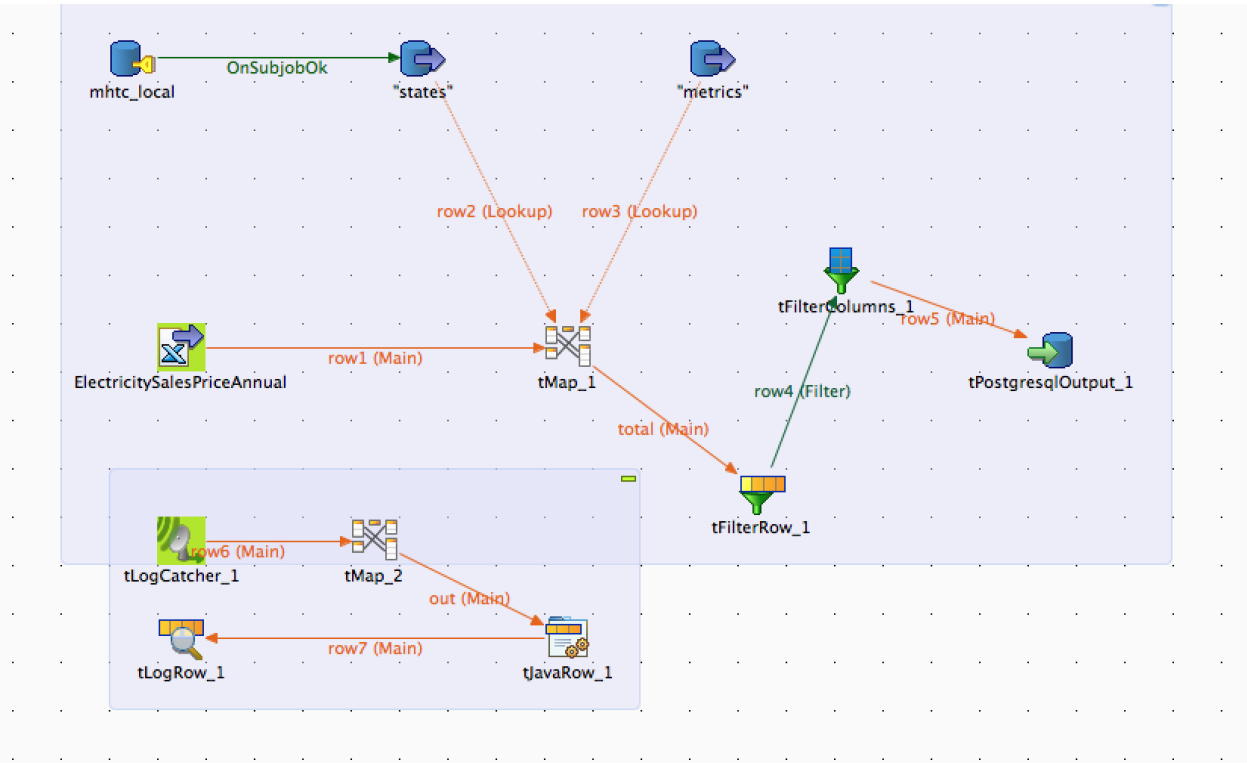


Figure 97: Electricity Sales Price Annual Pipeline

The above figure shows an idea of what the pipeline looks like. There is a database connection, which is used to do lookups on the states and metrics contained within the database. This is important to validate the data that is being added to the database. The tMap contains the physical mapping between the file and the database. tFilterRow is used to filter out some unnecessary information before inserting into the database.

5.7 NSF Talent Supply

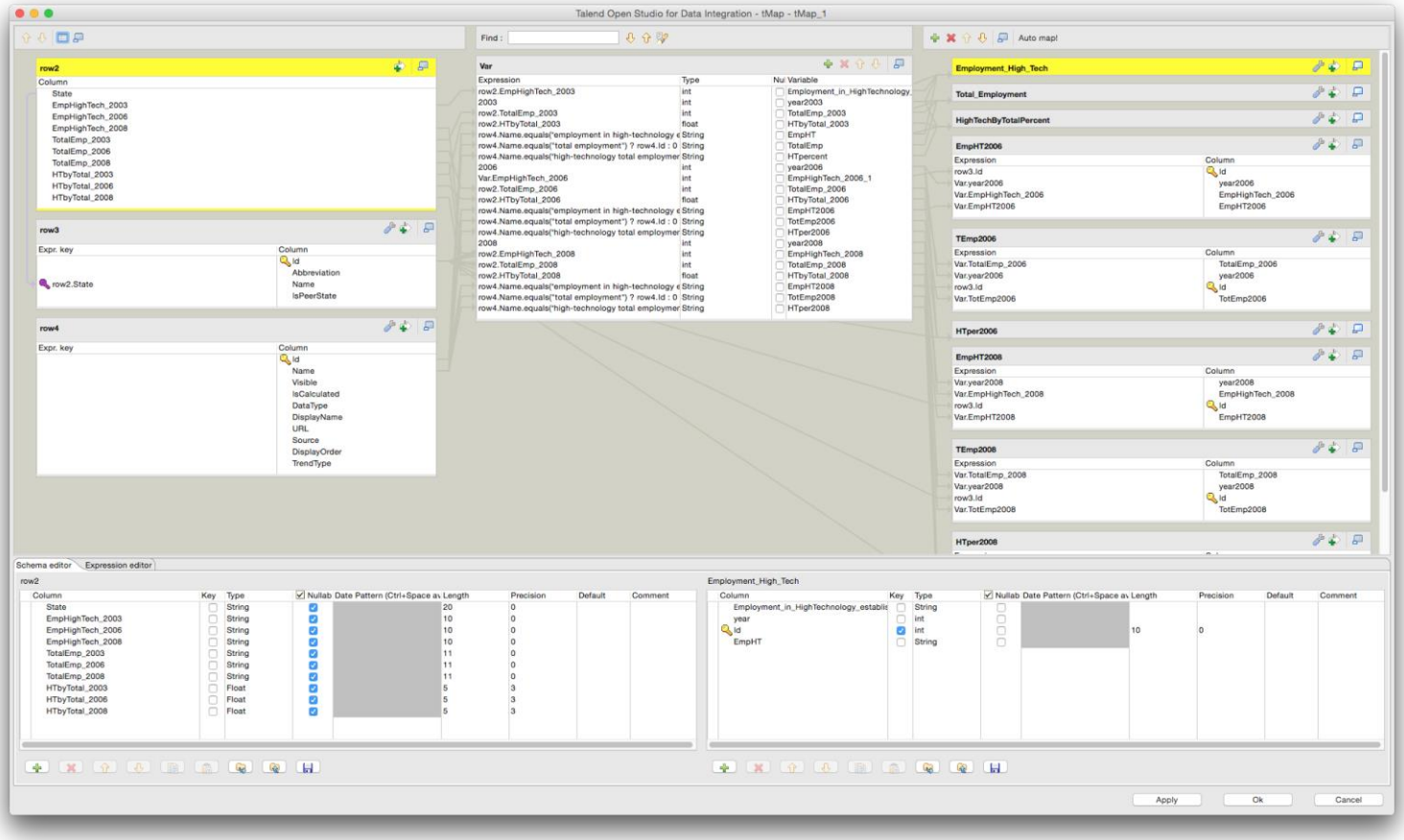


Figure 98: NSF Talent Supply tMap

row2 --

- excel spreadsheet being read in
- State being used as a foreign key for the State table from the database (row3)
- metrics is being used in the output (Employment in HighTech, Total Employment, HighTech per Total)

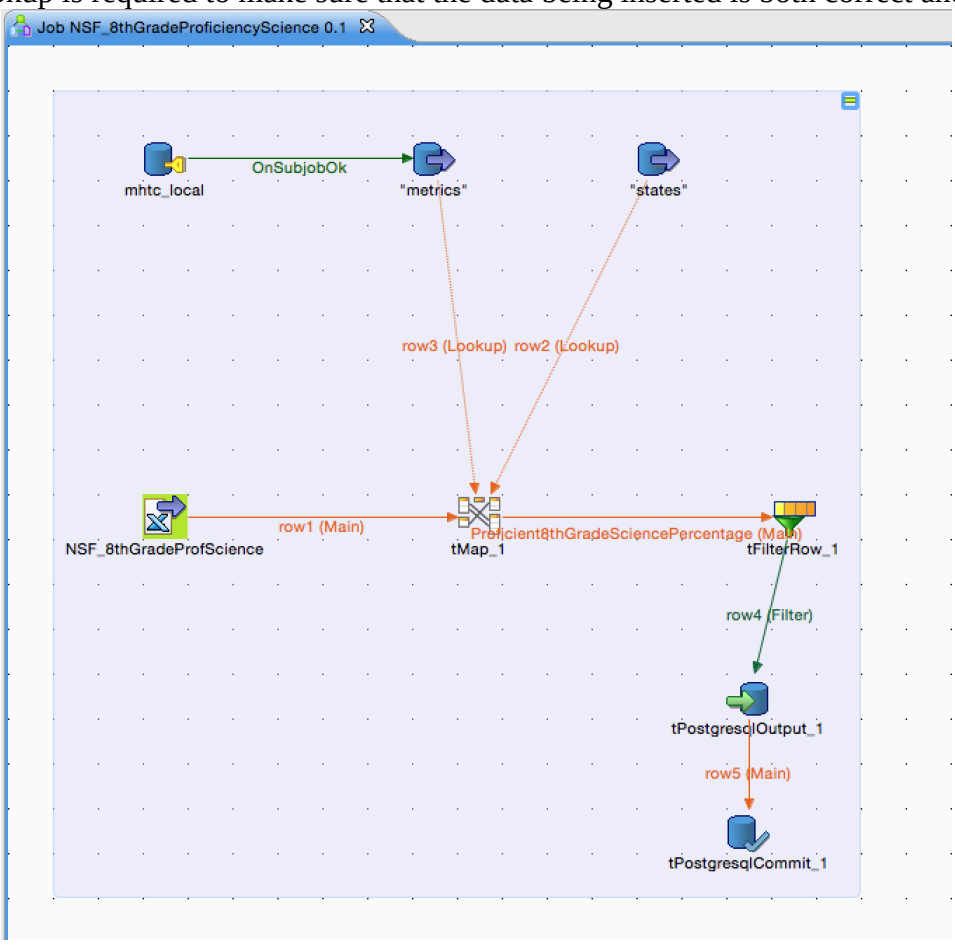
row3 --

- state id is being used in the output

row4 --

- metric name is being looked at to see if it exists in database. If yes, then passing the metricID.

The above figure shows the tMap of the pipeline that maps the data from the excel spreadsheet to the database after looking up some information that is contained in the database. This lookup is required to make sure that the data being inserted is both correct and in the right format.



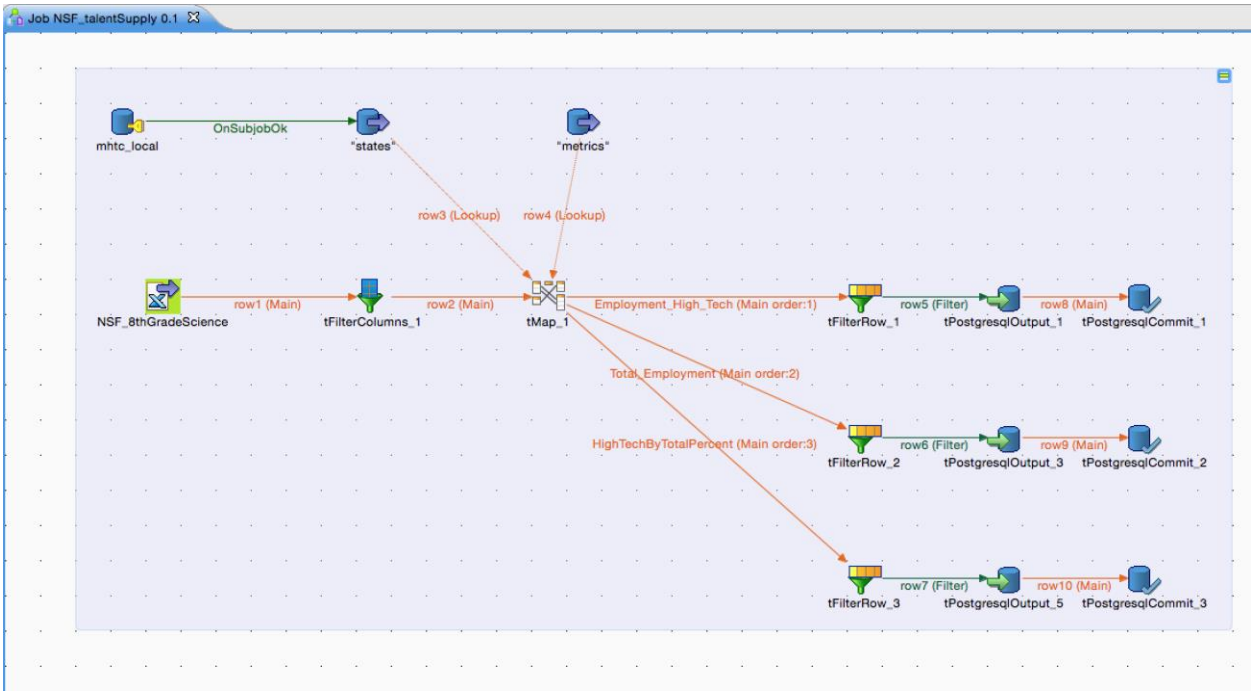


Figure 99: NSF Talent Supply Pipeline

The above figure shows an idea of what the pipeline looks like. There is a database connection, which is used to do lookups on the states and metrics contained within the database. This is important to validate the data that is being added to the database. The tMap contains the physical mapping between the file and the database. tFilterRow is used to filter out some unnecessary information before inserting into the database.

5.8 BLS Employment

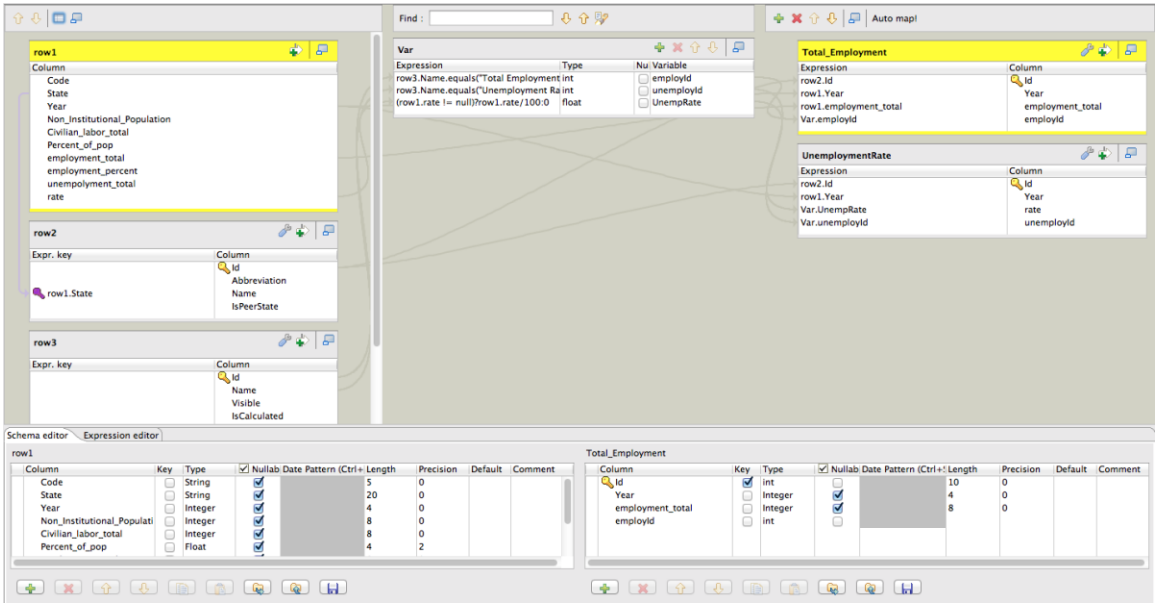


Figure 100: BLS Employment tMap

Name: BLS Employment

Source: BLS – http://data.bls.gov/oes/

Columns: State, Year, Total Employment, Unemployment Rate

row1 --

- excel spreadsheet being read in
- State being used as a foreign key for the State table from the database (row2)
- metric is being used in the output (Total Employment, Unemployment Rate)

row2 --

- state id is being used in the output

row3 --

- metric name is being looked at to see if it exists in database. If yes, then passing the metricID.

Total_Employment –

- StateID (row2)
- Year (input file)
- Employment Total (input file)

- employID (row3, database lookup)

UnemploymentRate

- StateID (row2)
- Year (input file)
- Unemployment rate (computed from input file)
- unemployID (row3, database lookup)

The above figure shows the tMap of the pipeline that maps the data from the excel spreadsheet to the database after looking up some information that is contained in the database. This lookup is required to make sure that the data being inserted is both correct and in the right format.

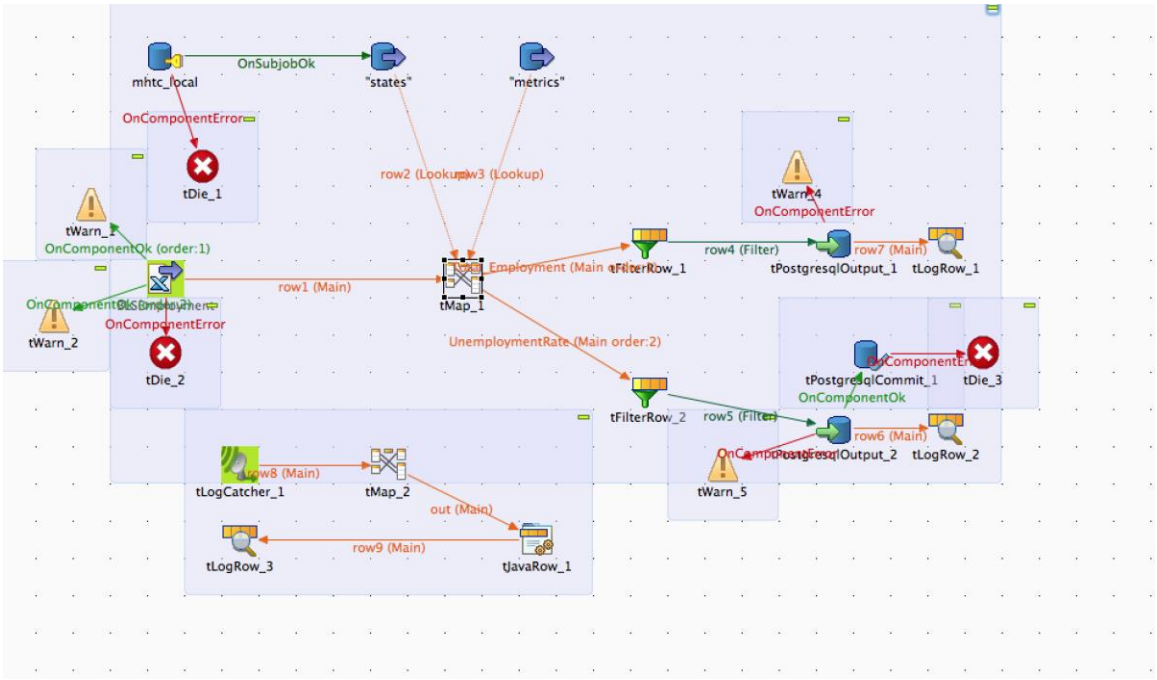


Figure 101: BLS Employment Pipeline

The above figure shows an idea of what the pipeline looks like. There is a database connection, which is used to do lookups on the states and metrics contained within the database. This is important to validate the data that is being added to the database. The tMap contains the

physical mapping between the file and the database. tFilterRow is used to filter out some unnecessary information before inserting into the database.

5.9 Capital Gains Tax Rate

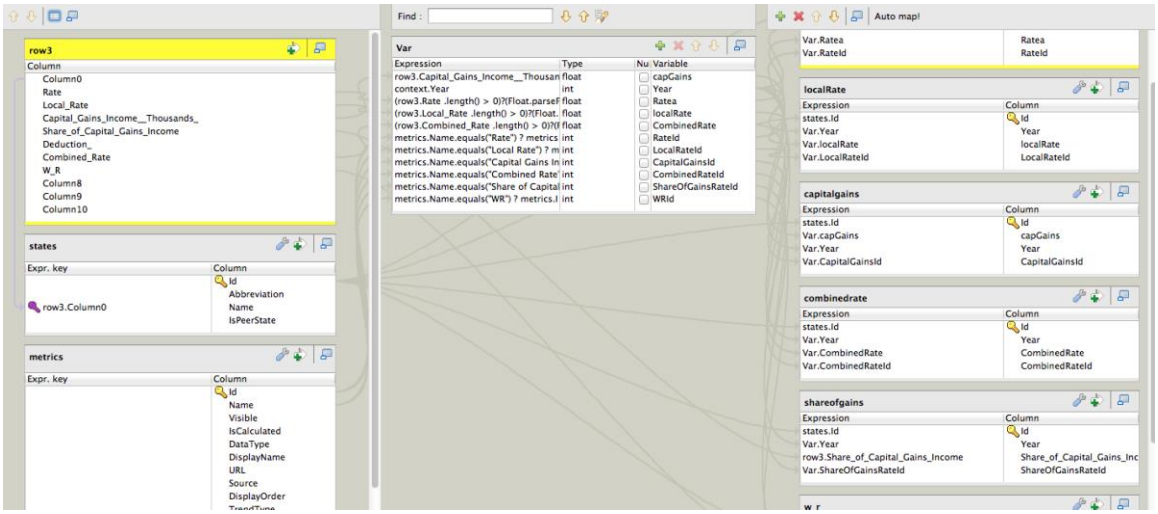


Figure 102: Capital Gains Tax Rate tMap

Name: Capital Gains Tax Rate

Source: Tax Foundation - http://taxfoundation.org/article/high-burden-state-and-federal-capital-gains-tax-rates

Columns: State, Year, Rate, Local Rate, Capital Rate, Combined Rate, Share of Capital Gains, WR

row3 --

- excel spreadsheet being read in
- State being used as a foreign key for the State table from the database (states)
- metric is being used in the output (Local Rate, Capital Rate, Combined Rate, Share of Capital Gains, WR)

states --

- state id is being used in the output

metrics --

- metric name is being looked at to see if it exists in database. If yes, then passing the metricID.

localRate –

- StateID (states)
- Year (context of input file)
- Local rate (input file)
- Local rate ID (metrics, database lookup)

CapitalRate –

- StateID (states)
- Year (context of input file)
- Capital rate (input file)
- Capital rate ID (metrics, database lookup)

CombinedRate –

- StateID (states)
- Year (context of input file)
- Combined rate (input file)
- Combined rate ID (metrics, database lookup)

Share of Capital Gains –

- StatesID (states)
- Year (context of input file)
- Share of Capital Gains (input file)
- Share of capital gains ID (metrics, database lookup)

WR –

- StatesID (states)
- Year (context of input file)
- WR (input file)
- WRID (metrics, database lookup)

The above figure shows the tMap of the pipeline that maps the data from the excel spreadsheet to the database after looking up some information that is contained in the database. This lookup is required to make sure that the data being inserted is both correct and in the right format.

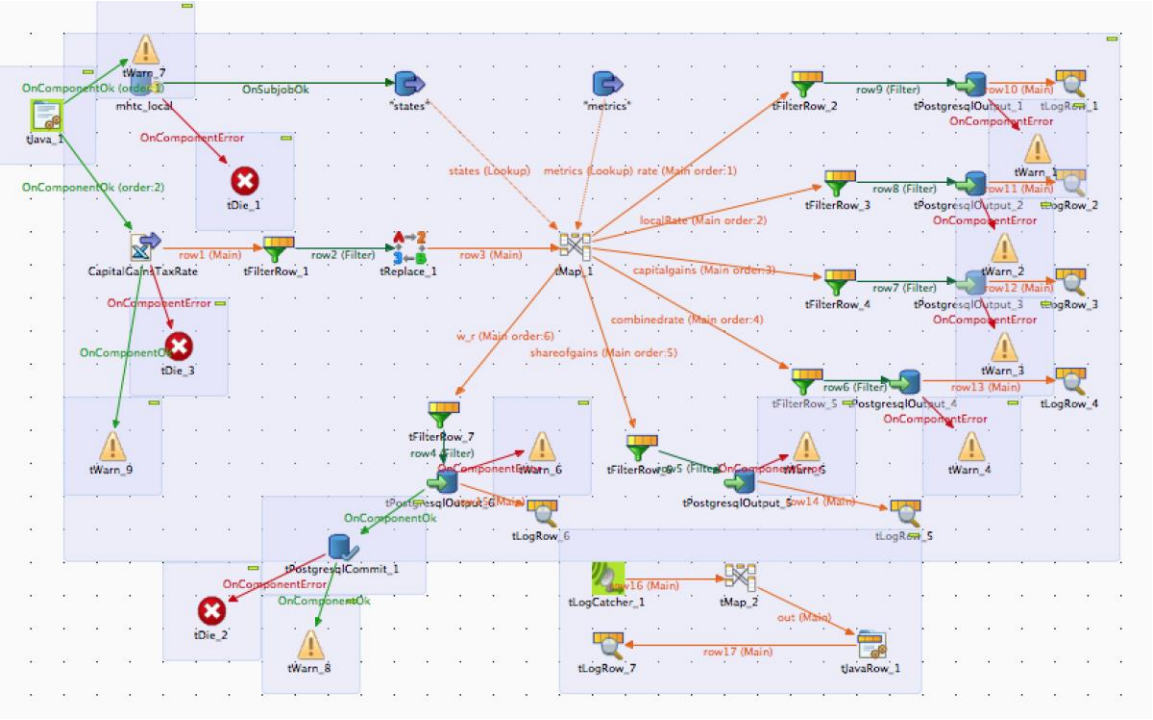


Figure 103: Captial Gains Tax Rate Pipeline

The above figure shows an idea of what the pipeline looks like. There is a database connection, which is used to do lookups on the states and metrics contained within the database. This is important to validate the data that is being added to the database. The tMap contains the physical mapping between the file and the database. tFilterRow is used to filter out some unnecessary information before inserting into the database.

5.10 Census Personal Income

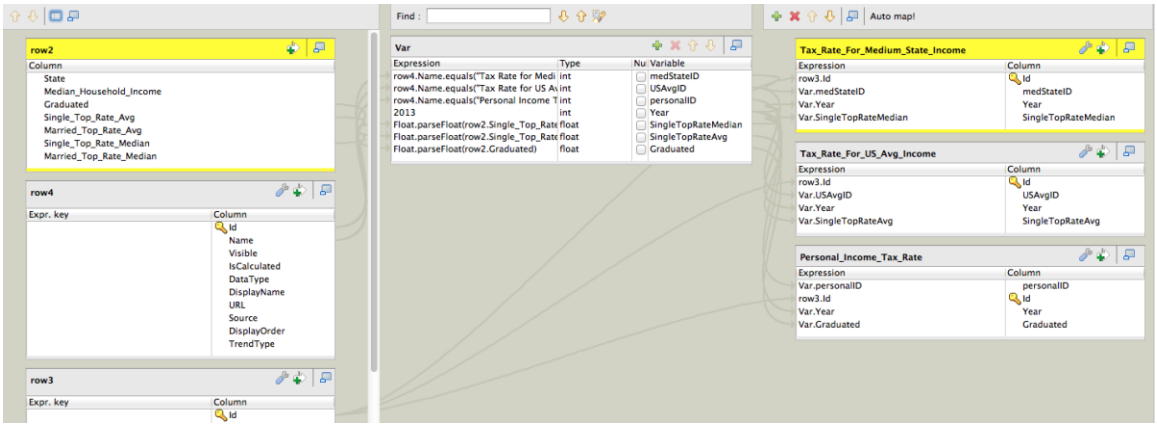


Figure 104: Census Personal Income tMap

Name: Census Personal Income

Source: Tax Foundation - http://taxfoundation.org/sites/taxfoundation.org/files/docs/State%20Individual%20Income%20Tax%20Rates%2C%202000-2014.xlsx

Columns: State, Year, Medium State Income, US Average Income, Personal Income Tax Rate

Row2 --

- excel spreadsheet being read in
- State being used as a foreign key for the State table from the database (row3)
- metric is being used in the output

row3 --

- state id is being used in the output

row4 --

- metric name is being looked at to see if it exists in database. If yes, then passing the metricID.

Tax_Rate_For_Medium_State_Income –

- StateID (row3)
- medStateID (row4, database lookup)
- year (variable)
- single top rate medium (input file)

Tax_Rate_For_US_Avg_Income

- StateID (row3)
- USAvgID (row4, database lookup)
- Year (variable)
- SingleTopRateAvg (input file)

Personal_Income_Tax_Rate

- StateID (row3)
- personalID (row4, database lookup)
- year (variable)
- graduated (input file)

The above figure shows the tMap of the pipeline that maps the data from the excel spreadsheet to the database after looking up some information that is contained in the database. This lookup is required to make sure that the data being inserted is both correct and in the right format.

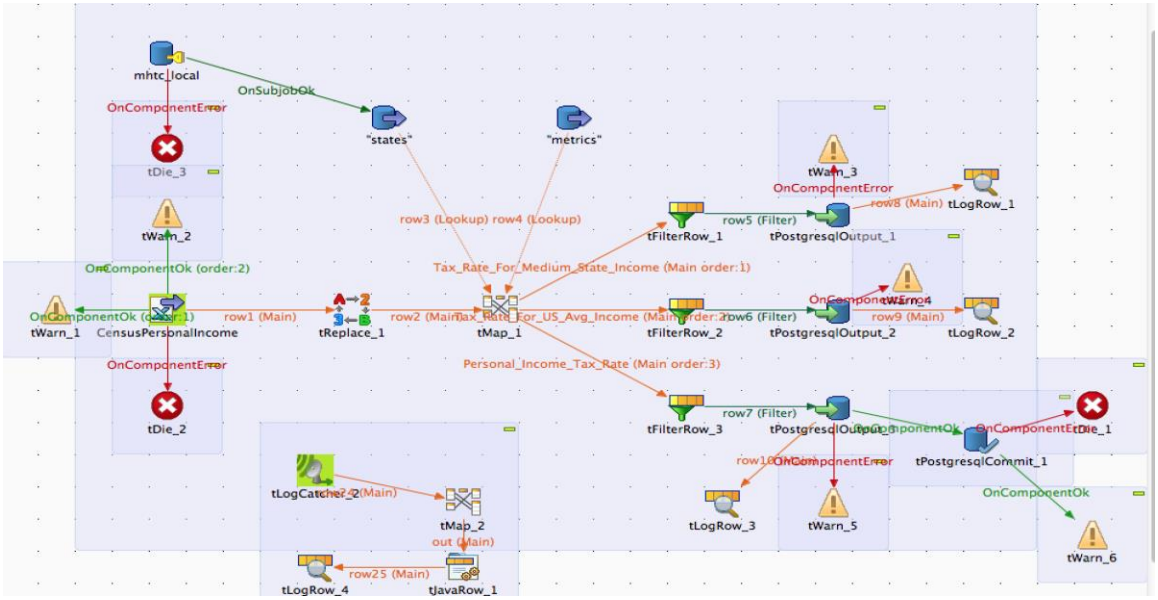


Figure 105: Census Personal Income Pipeline

The above figure shows an idea of what the pipeline looks like. There is a database connection, which is used to do lookups on the states and metrics contained within the database. This is important to validate the data that is being added to the database. The tMap contains the

physical mapping between the file and the database. tFilterRow is used to filter out some unnecessary information before inserting into the database.

5.11 CNBC Overall Ranks

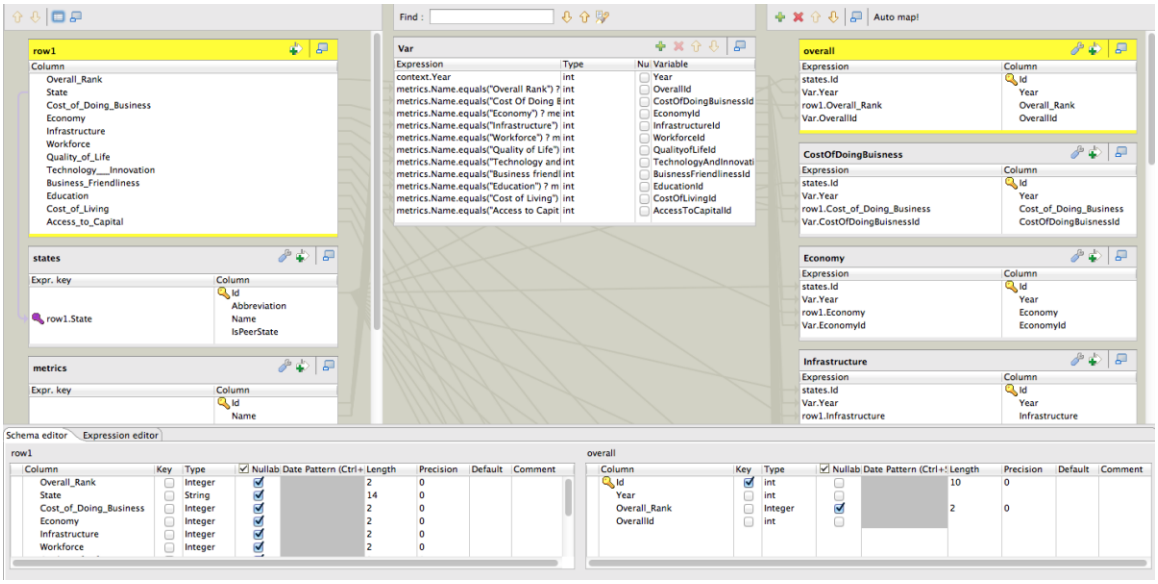


Figure 106: CNBC Overall Ranks tMap

Name: CNBC Overall Ranks

Source: http://www.cnbc.com/id/100824779

Columns: State, Year, Overall Rank, Cost of Doing Business, Economy, Infrastructure, Workforce, Quality of Life, Technology, Business, Education, Cost of Living, Access to Capital

row1 --

- excel spreadsheet being read in
- State being used as a foreign key for the State table from the database (states)
- metric is being used in the output

states --

- state id is being used in the output

metrics --

- metric name is being looked at to see if it exists in database. If yes, then passing the metricID.

Overall –

- StatesID (states)
- Year (context of input file)

- Overall_Rank (input file)
- OverallID (metrics, database lookup)

CostOfDoingBusiness

- StatesID (states)
- Year (context of input file)
- Cost_of_doing_business (input file)
- CostOfDoingBusinessID (metrics, database lookup)

Economy

- StatesID (states)
- Year (context of input file)
- Economy (input file)
- EconomyID (metrics, database lookup)

Infrastructure

- StatesID (states)
- Year (context of input file)
- infrastructure (input file)
- infrastructureID (metrics, database lookup)

Workforce

- StatesID (states)
- Year (context of input file)
- workforce (input file)
- workforceID (metrics, database lookup)

Quality of Life

- StatesID (states)
- Year (context of input file)
- Quality_of_life (input file)
- QualityOfLifeID (metrics, database lookup)

Technology

- StatesID (states)
- Year (context of input file)
- Technology (input file)
- TechnologyID (metrics, database lookup)

Business

- StatesID (states)
- Year (context of input file)
- Business (input file)
- BusinessID (metrics, database lookup)

Education

- StatesID (states)
- Year (context of input file)
- Education (input file)
- EducationID (metrics, database lookup)

Cost of Living

- StatesID (states)
- Year (context of input file)
- Cost_of_living (input file)
- CostOfLivingID (metrics, database lookup)

Access to capital

- StatesID (states)
- Year (context of input file)
- Access_to_capital (input file)
- AccessToCapitalID (metrics, database lookup)

The above figure shows the tMap of the pipeline that maps the data from the excel spreadsheet to the database after looking up some information that is contained in the database.

This lookup is required to make sure that the data being inserted is both correct and in the right format.

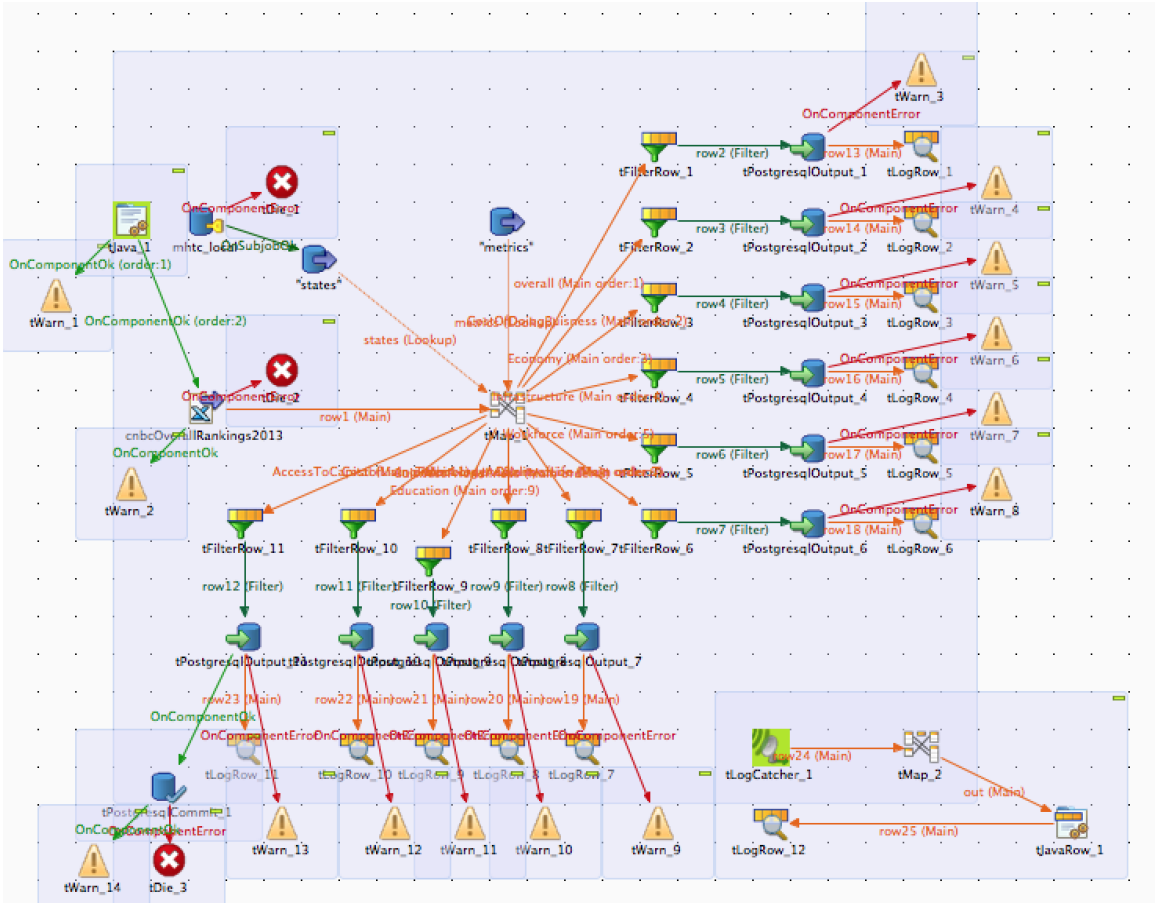


Figure 107: CNBC Overall Ranks Pipeline

The above figure shows an idea of what the pipeline looks like. It may seem overwhelming however it is similar to any other pipeline except that there is more data to move from the file to the database in this one. There is a database connection, which is used to do lookups on the states and metrics contained within the database. This is important to validate the data that is being added to the database. The tMap contains the physical mapping between the file and the database. tFilterRow is used to filter out some unnecessary information before inserting into the database.

5.12 EIA Electricity Access

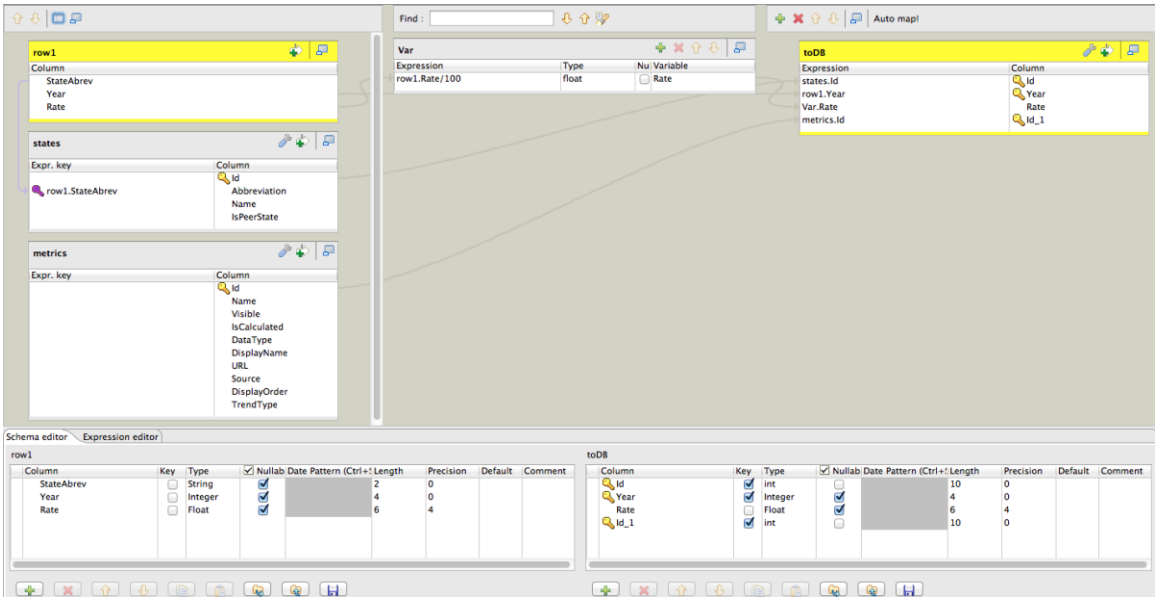


Figure 108: EIA Electricity Access tMap

Name: EIA Electricity Access

Source: eia - http://www.eia.gov/electricity/data/state/sales_annual.xls

Columns: State, Year, Rate

row1 --

- excel spreadsheet being read in
- State being used as a foreign key for the State table from the database (states)
- metric is being used in the output

states --

- state id is being used in the output

metrics --

- metric name is being looked at to see if it exists in database. If yes, then passing the metricID.

toDB –

- StatesID (states)
- Year (input file)
- Rate (input file, converted by /100)
- MetricID (metrics, database lookup)

The above figure shows the tMap of the pipeline that maps the data from the excel spreadsheet to the database after looking up some information that is contained in the database. This lookup is required to make sure that the data being inserted is both correct and in the right format.

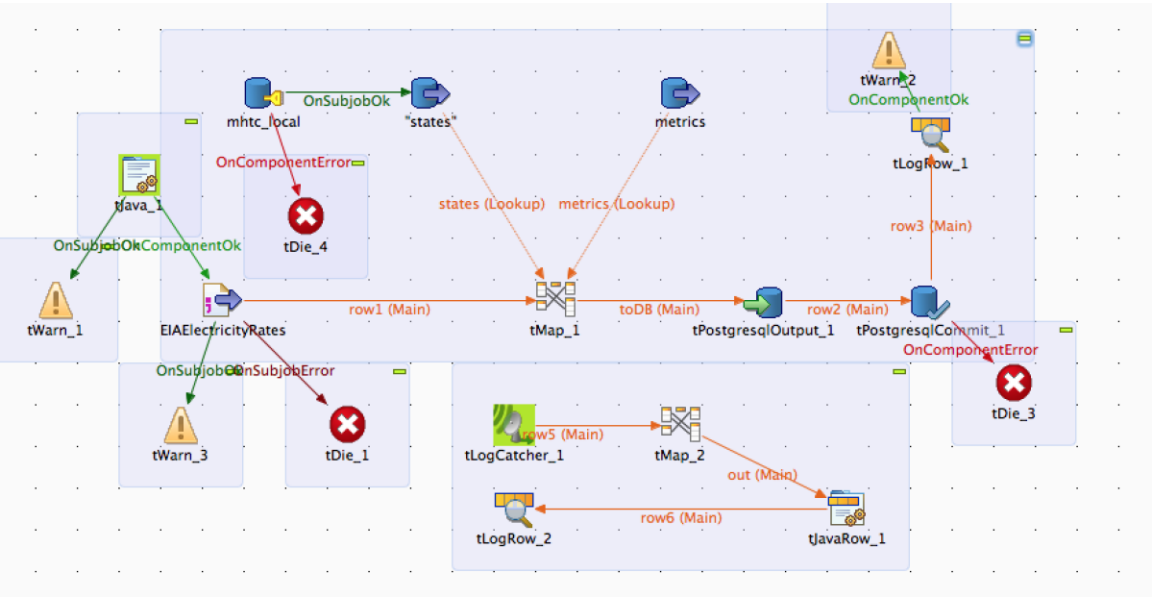


Figure 109: EIA Electricity Access Pipeline

The above figure shows an idea of what the pipeline looks like. There is a database connection, which is used to do lookups on the states and metrics contained within the database. This is important to validate the data that is being added to the database. The tMap contains the physical mapping between the file and the database. tFilterRow is used to filter out some unnecessary information before inserting into the database.

5.13 Grade 8 Student Performance

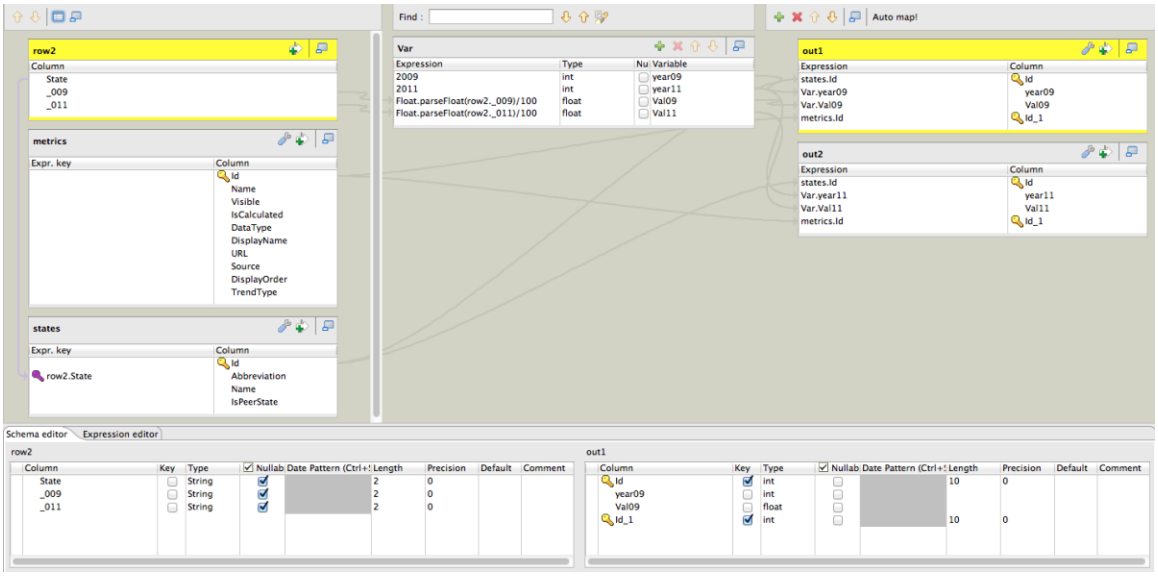


Figure 110: Grade 8 Student Performance tMap

Row2 --

- excel spreadsheet being read in
- State being used as a foreign key for the State table from the database (states)
- metric is being used in the output

states --

- state id is being used in the output

metrics --

- metric name is being looked at to see if it exists in database. If yes, then passing the metricID.

Out1 –

- StatesID (states)
- Year (input file)
- Val09 (input file)
- metricID (metrics, database lookup)

Out2 –

- StatesID (states)
- Year (input file)
- Val11 (input file)
- metricID (metrics, database lookup)

The above figure shows the tMap of the pipeline that maps the data from the excel spreadsheet to the database after looking up some information that is contained in the database. This lookup is required to make sure that the data being inserted is both correct and in the right format.

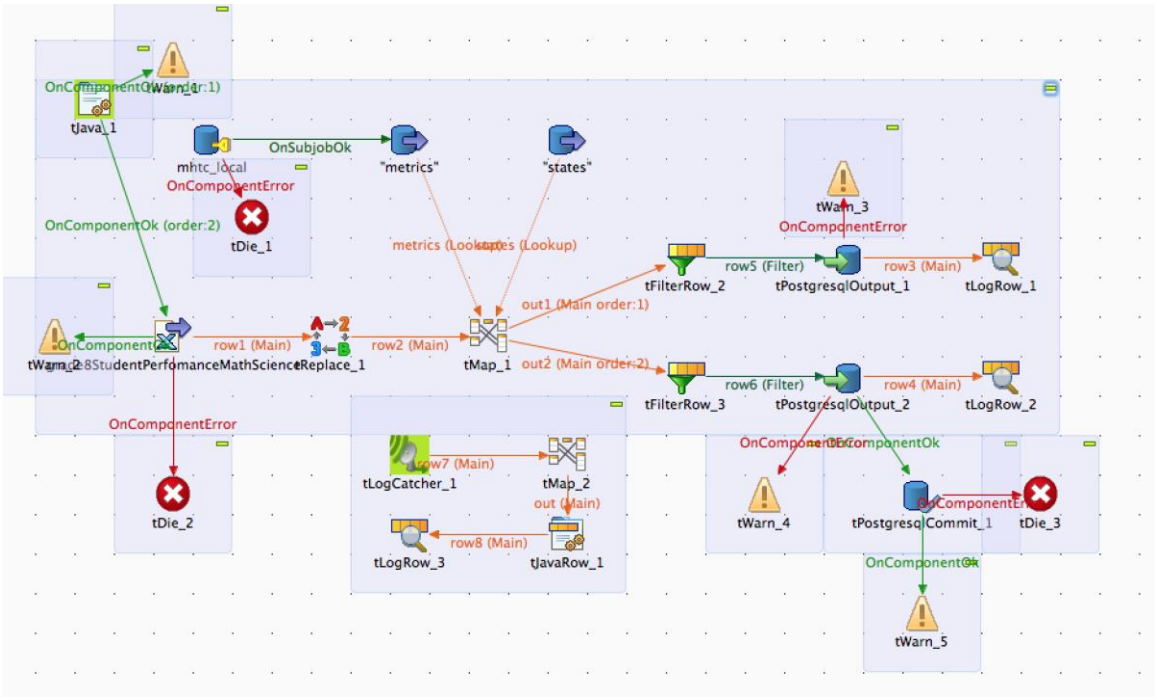


Figure 111: Grade 8 Student Performance Pipeline

The above figure shows an idea of what the pipeline looks like. There is a database connection, which is used to do lookups on the states and metrics contained within the database. This is important to validate the data that is being added to the database. The tMap contains the physical mapping between the file and the database. tFilterRow is used to filter out some unnecessary information before inserting into the database.

5.14 IPEDS Number of Colleges and Universities

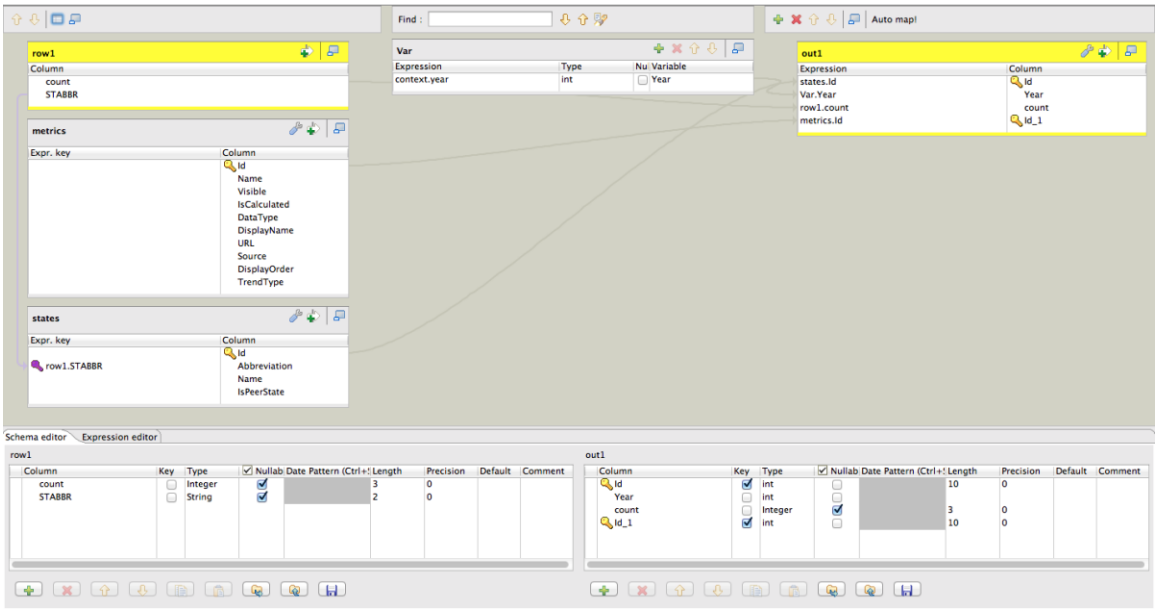


Figure 112: IPEDS Number of Colleges and Universities tMap

Name: IPEDS

Source: Data.gov -- https://inventory.data.gov/dataset/032e19b4-5a90-41dc-83ff-6e4cd234f565/resource/38625c3d-5388-4c16-a30f-d105432553a4

Columns: State, Year, Count

row1 --

- excel spreadsheet being read in
- State being used as a foreign key for the State table from the database (states)
- metric is being used in the output

states --

- state id is being used in the output

metrics --

- metric name is being looked at to see if it exists in database. If yes, then passing the metricID.

Out1 –

- StatesID (states)
- Year (context of input file)
- count (input file)
- metricID (metrics, database lookup)

The above figure shows the tMap of the pipeline that maps the data from the excel spreadsheet to the database after looking up some information that is contained in the database. This lookup is required to make sure that the data being inserted is both correct and in the right format.

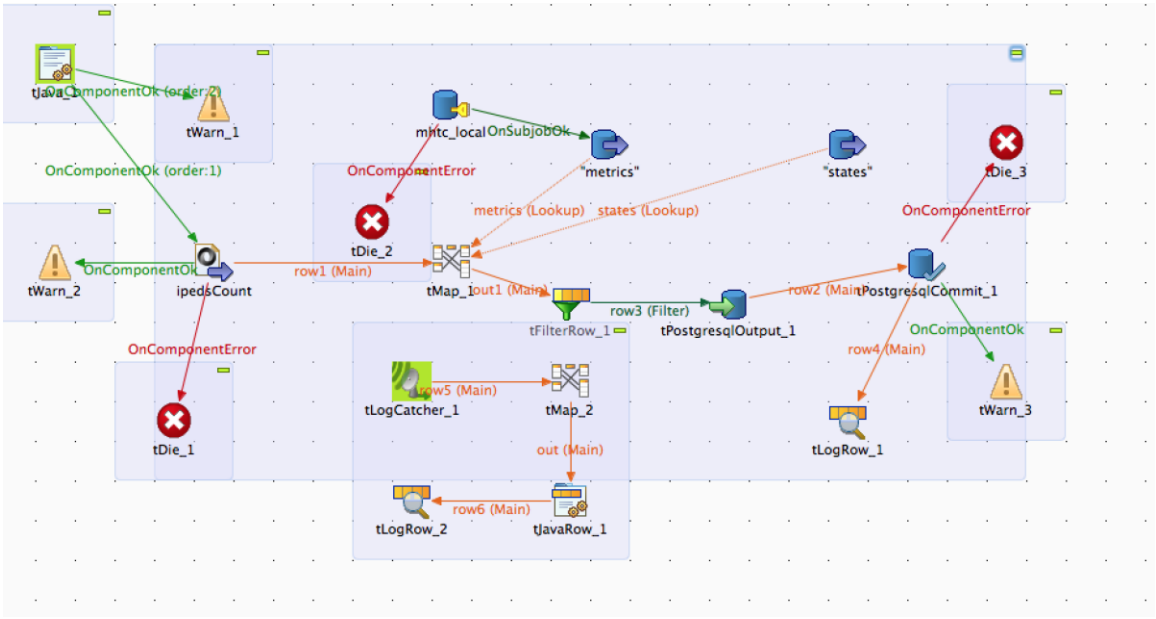


Figure 113: IPEDS Number of Colleges and Universities Pipeline

The above figure shows an idea of what the pipeline looks like. This pipeline is a bit different in that it reads from a JSON file however it does not complicate it. There is a wizard to create a JSON Metadata file just as there is for excel and csv files. There is a database

connection, which is used to do lookups on the states and metrics contained within the database. This is important to validate the data that is being added to the database. The tMap contains the physical mapping between the file and the database. tFilterRow is used to filter out some unnecessary information before inserting into the database.

5.15 IPEDS Stem Degrees

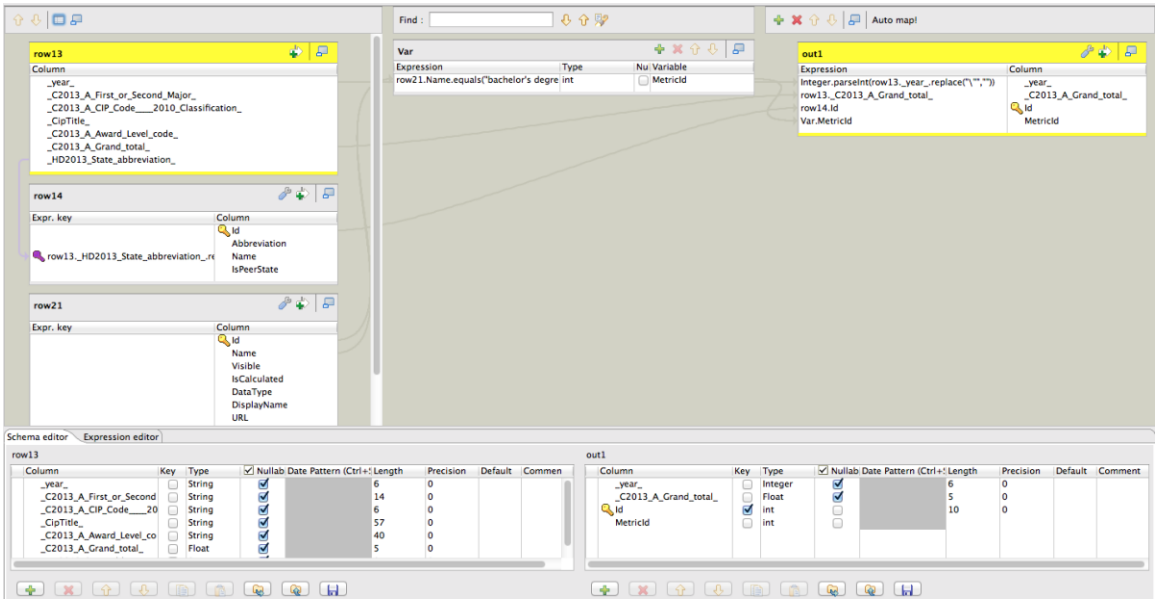


Figure 114: IPEDS Stem Degrees tMap for Output

Name: IPEDS Stem Degrees

Source: data.gov

Columns: State, Year, Bachelor's degree in STEM, Grand Total

row13 --

- excel spreadsheet being read in
- State being used as a foreign key for the State table from the database (row14)
- metric is being used in the output

row14 --

- state id is being used in the output

row21 --

- metric name is being looked at to see if it exists in database. If yes, then passing the metricID.

Out1 –

- StatesID (states)
- Year (input file)
- Grand_total (input file)

- metricID (metrics, database lookup)

The above figure shows the tMap of the pipeline that maps the data from the excel spreadsheet, after the data was joined based on the state and year to sum the total number of each degree type, with the database after looking up some information that is contained in the database. This lookup is required to make sure that the data being inserted is both correct and in the right format. For this pipeline there are four of these similar to each other, one for each degree type.

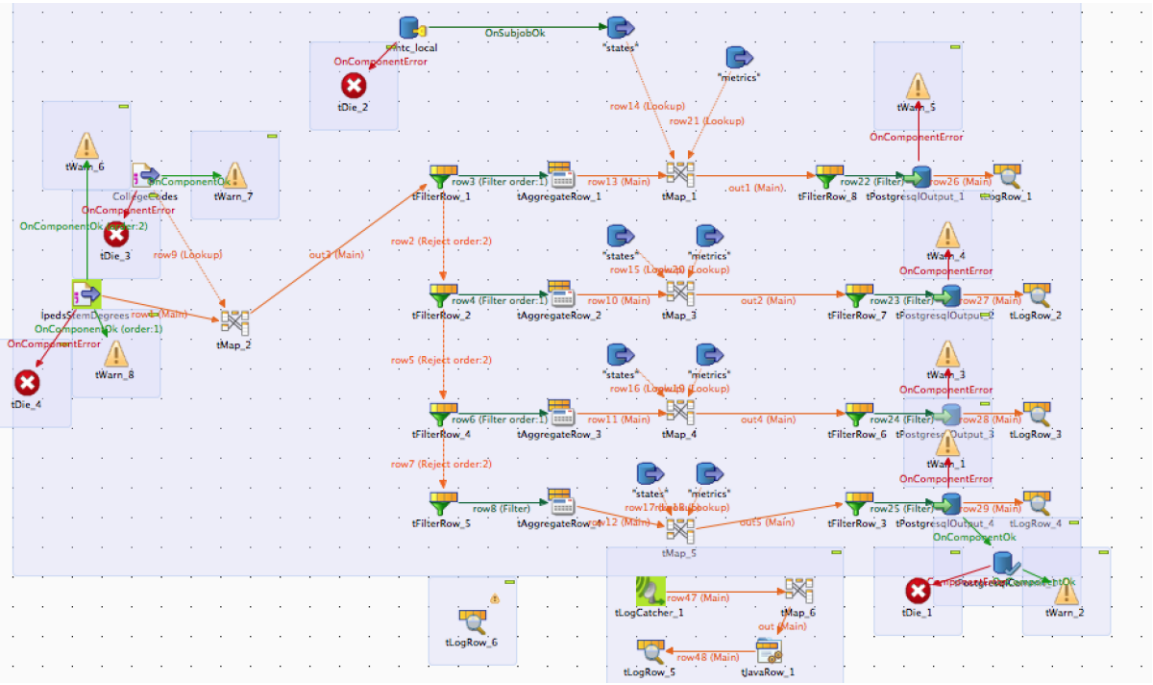


Figure 115: IPEDS Stem Degrees Pipeline

The above figure shows an idea of what the pipeline looks like. There is a database connection, which is used to do lookups on the states and metrics contained within the database. This is important to validate the data that is being added to the database. The tMap contains the physical mapping between the file and the database. tFilterRow is used to filter out some unnecessary information before inserting into the database. This also uses two files; below you can see the tMap, which joins the files together to get the state name for the college. It also uses the tAggregateRow component which is just used to sum all of the degrees of a certain type from each college to give each college a total of all bachelors, masters, doctorates and associates degrees separately to be entered into the database.

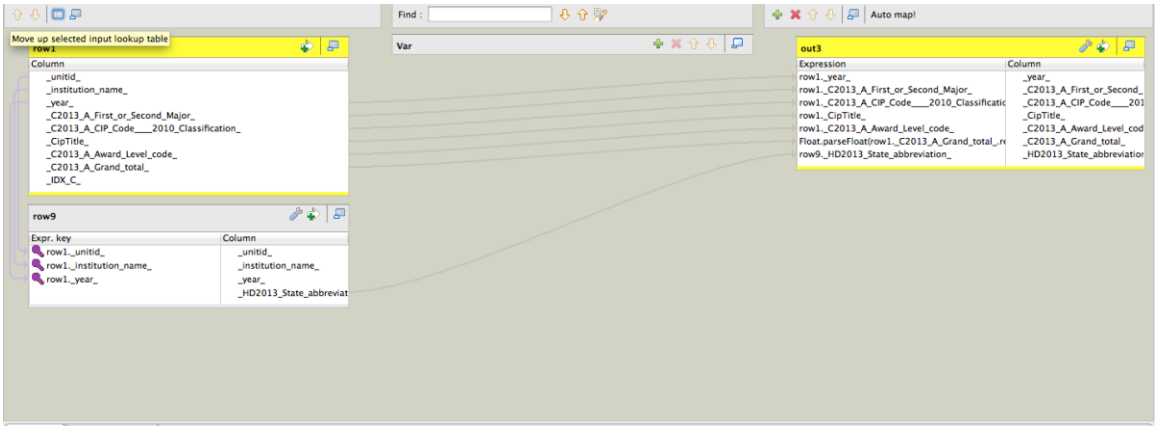


Figure 116: IPEDS Stem Degrees tMap for Input

5.16 State and Local Tax Burden

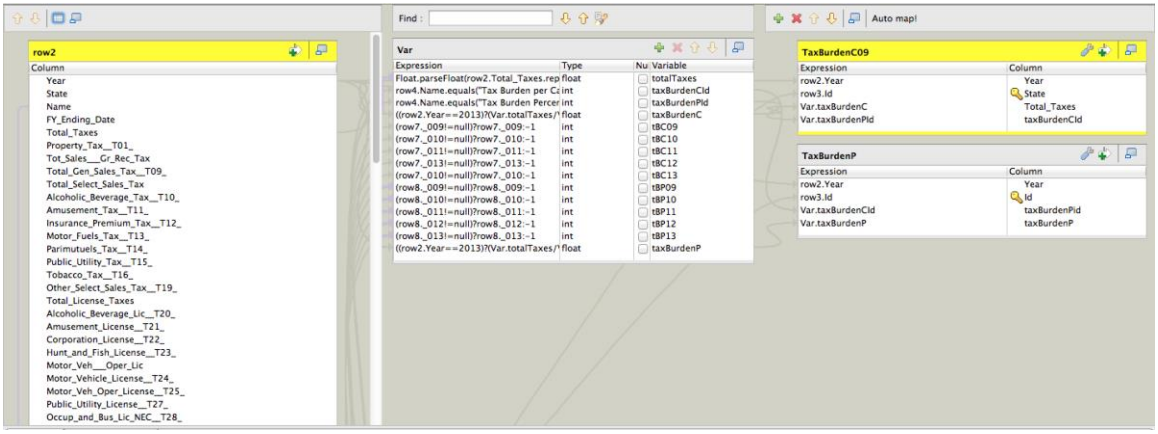


Figure 117: State and Local Tax Burden tMap

Name: State and Local Tax Burden

Source: Tax Foundation -- http://taxfoundation.org/article/state-and-local-property-tax-collections-capita-state-2006-2010

Columns: State, Year, Tax Burden per Capita, Tax Burden Percent

Row2 --

- excel spreadsheet being read in
- State being used as a foreign key for the State table from the database (row3)
- metric is being used in the output

row3 --

- state id is being used in the output

row4 --

- metric name is being looked at to see if it exists in database. If yes, then passing the metricID.

TaxBurdenC09 –

- StatesID (row3)
- Year (input file)
- Total_taxes (input file)
- totalTaxesID (metrics, database lookup)

TaxBurdenP

- StatesID (row3)

- Year (input file)
- taxBurdenP (input file)
- taxBurdenID (metrics, database lookup)

The above figure shows the tMap of the pipeline that maps the data from the excel spreadsheet to the database after looking up some information that is contained in the database. This lookup is required to make sure that the data being inserted is both correct and in the right format.

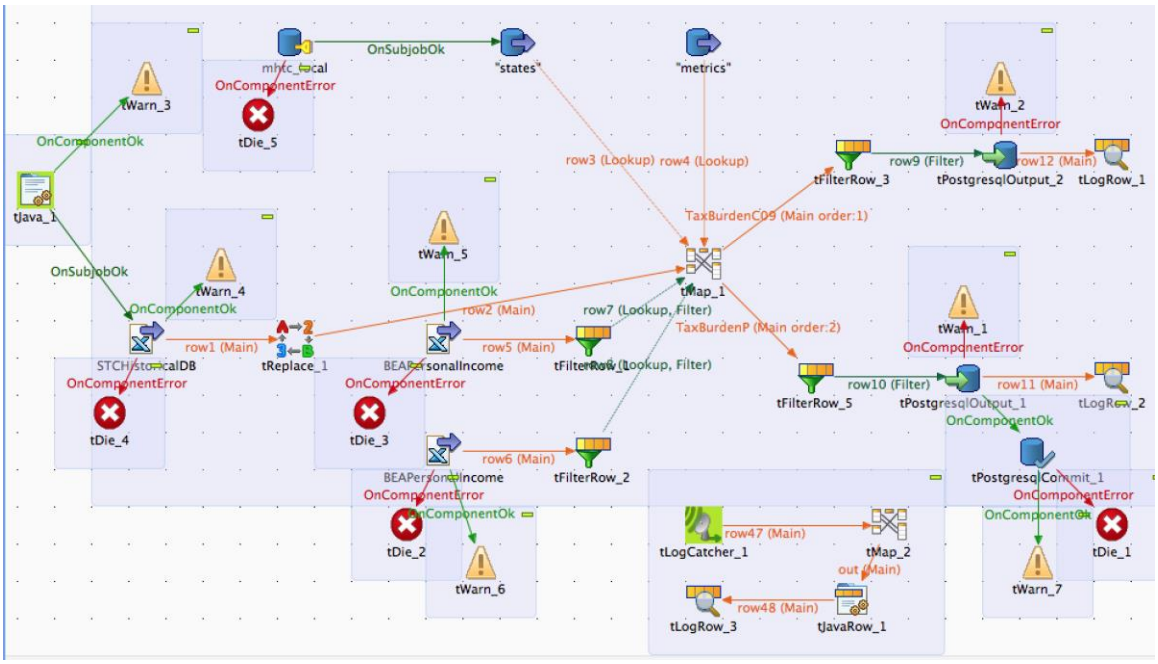


Figure 118: State and Local Tax Burden Pipeline

The above figure shows an idea of what the pipeline looks like. There is a database connection, which is used to do lookups on the states and metrics contained within the database. This is important to validate the data that is being added to the database. The tMap contains the physical mapping between the file and the database. tFilterRow is used to filter out some unnecessary information before inserting into the database. This also uses two files but only to do some division that is done in the variables part of the tMap.

5.17 Tech and Total Employment

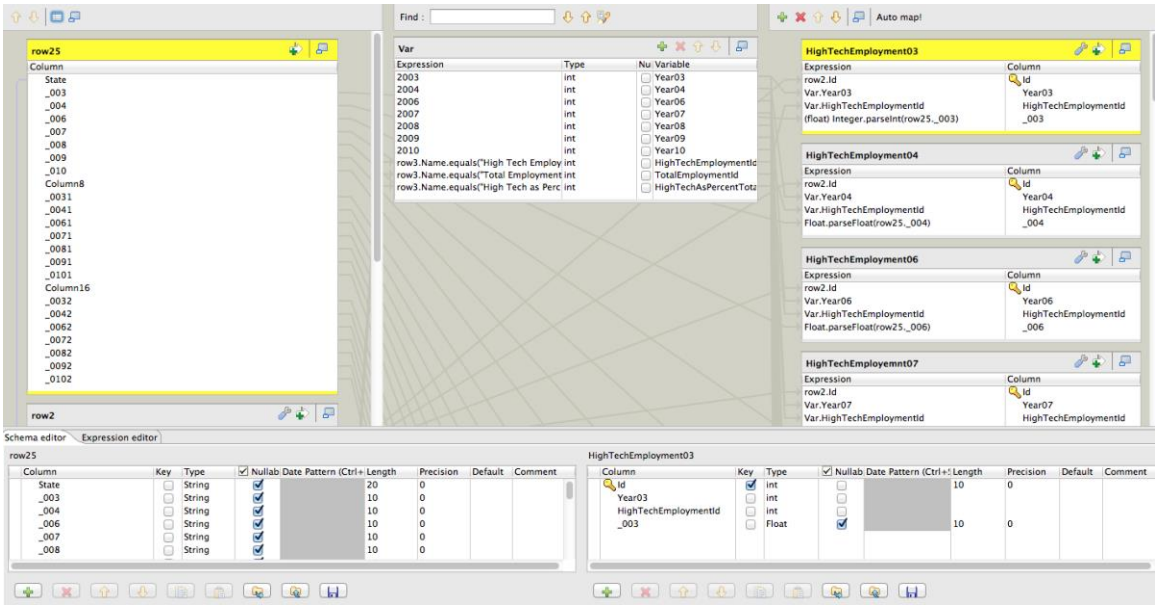


Figure 119: Tech and Total Employment tMap

Name: Tech and Total Employment

Source: http://www.nsf.gov/statistics/seind12/c8/tt08-54.xls

Columns: State, Year, High Tech Employment, Total Employment, High Tech as Percent

Row25 --

- excel spreadsheet being read in
- State being used as a foreign key for the State table from the database (row2)
- metric is being used in the output

row2 --

- state id is being used in the output

row3 --

- metric name is being looked at to see if it exists in database. If yes, then passing the metricID.

HighTechEmployment03 –

- StatesID (row2)
- Year (input file)
- _003 (input file)

- HighTechEmploymentID (metrics, database lookup)

HighTechEmployment04 –

- StatesID (row2)
- Year (input file)
- _004 (input file)
- HighTechEmploymentID (metrics, database lookup)

HighTechEmployment06 –

- StatesID (row2)
- Year (input file)
- _006 (input file)
- HighTechEmploymentID (metrics, database lookup)

HighTechEmployment07 –

- StatesID (row2)
- Year (input file)
- _007 (input file)
- HighTechEmploymentID (metrics, database lookup)

HighTechEmployment08 –

- StatesID (row2)
- Year (input file)
- _008 (input file)
- HighTechEmploymentID (metrics, database lookup)

HighTechEmployment09 –

- StatesID (row2)
- Year (input file)
- _009 (input file)
- HighTechEmploymentID (metrics, database lookup)

HighTechEmployment10 –

- StatesID (row2)

- Year (input file)
- _0010 (input file)
- HighTechEmploymentID (metrics, database lookup)

The above figure shows the tMap of the pipeline that maps the data from the excel spreadsheet to the database after looking up some information that is contained in the database. This lookup is required to make sure that the data being inserted is both correct and in the right format.

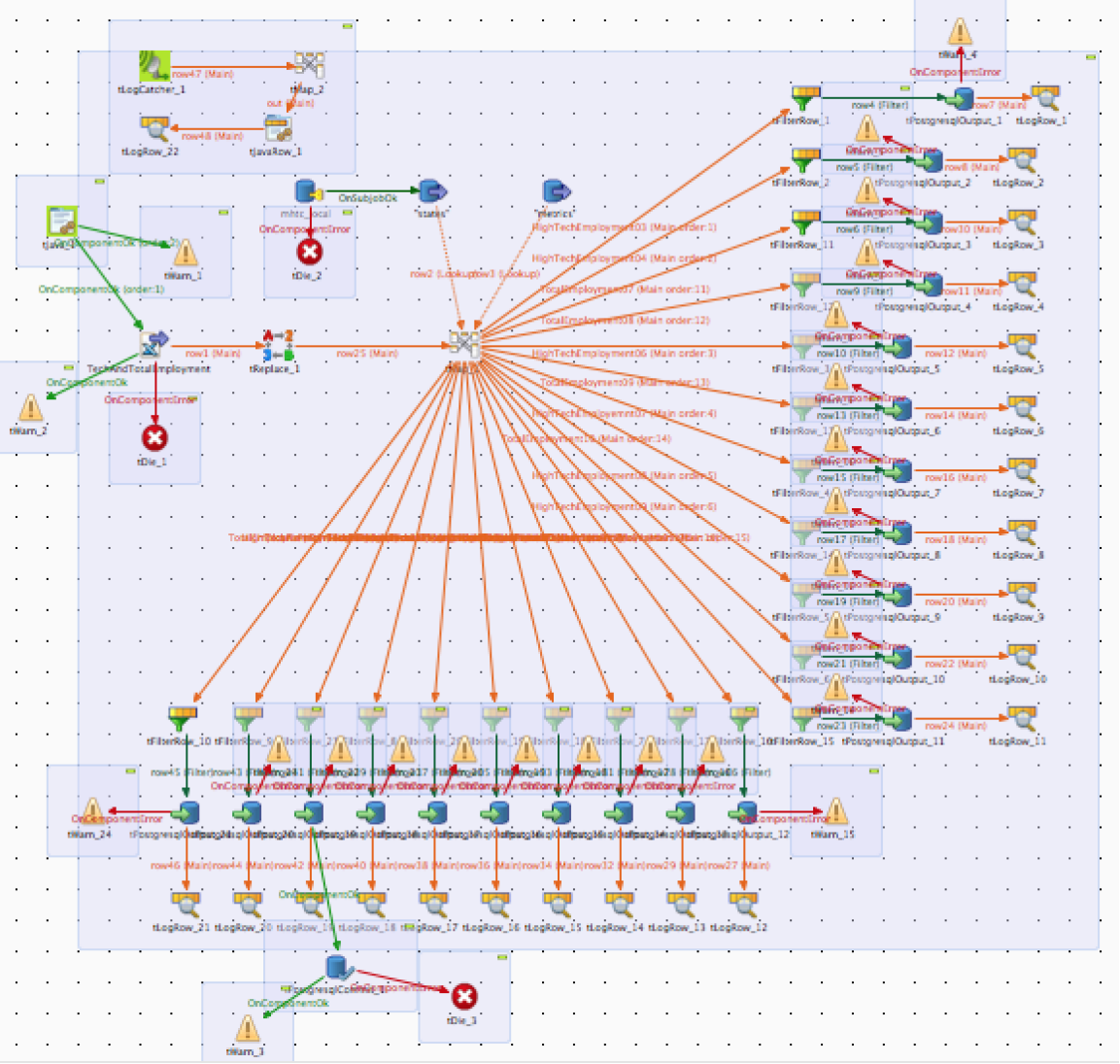


Figure 120: Tech and Total Employment Pipeline

The above figure shows an idea of what the pipeline looks like. There is a database connection, which is used to do lookups on the states and metrics contained within the database. This is important to validate the data that is being added to the database. The tMap contains the physical mapping between the file and the database. tFilterRow is used to filter out some unnecessary information before inserting into the database. This pipeline may look overwhelming but it is as simple as all the others there is just a lot of data to be inserted from this file.

Appendix C: MITRE Code Evaluation Report - January 30, 2015

Appendix D: MITRE Code Evaluation Report – April 21, 2015

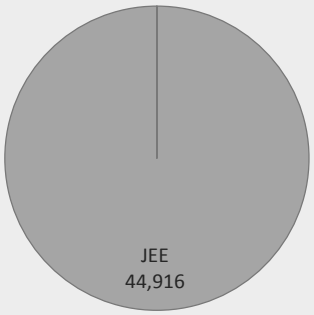
CAST Application Intelligence Platform (AIP) Analysis

Summary

The Application Assessment evaluates the overall quality of the MHTC421 application.

MHTC421 is a small application and has a medium quality with a ***Total Quality Indicator (TQI) of 2.83 on a scale of 1-4.*** Each of the additional health metrics and their scores are identified below.

Application Characteristics



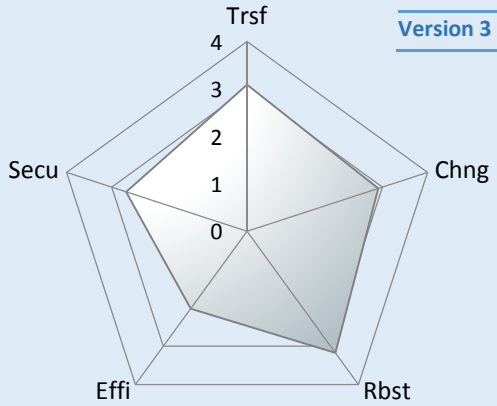
TOP 5 TECHNOLOGIES

Name	LOCs
JEE	44,916

TECHNICAL SIZE

Name	Number
kLOCs	45
Files	218
Classes	83
SQL Art.	0
Tables	0

Summary of Quality Indicators



	TQI	Robu.	Efcy.	Secu.	Trans.	Chang.
Version 3	2.83	3.17	2.02	2.68	3.09	2.91

TQI => Total Quality Index
Robu => Robustness/Reliability
Perf => Performance/Efficiency
Secu => Security
Trans => Transferability
Chang => Changeability

Assessment Highlights

STATISTICS ON VIOLATIONS

Name	Number
Critical Violations	59
per File	0.27
per kLOC	1.31
Complex Objects	42

Rule Names	Count

Complexity Distribution

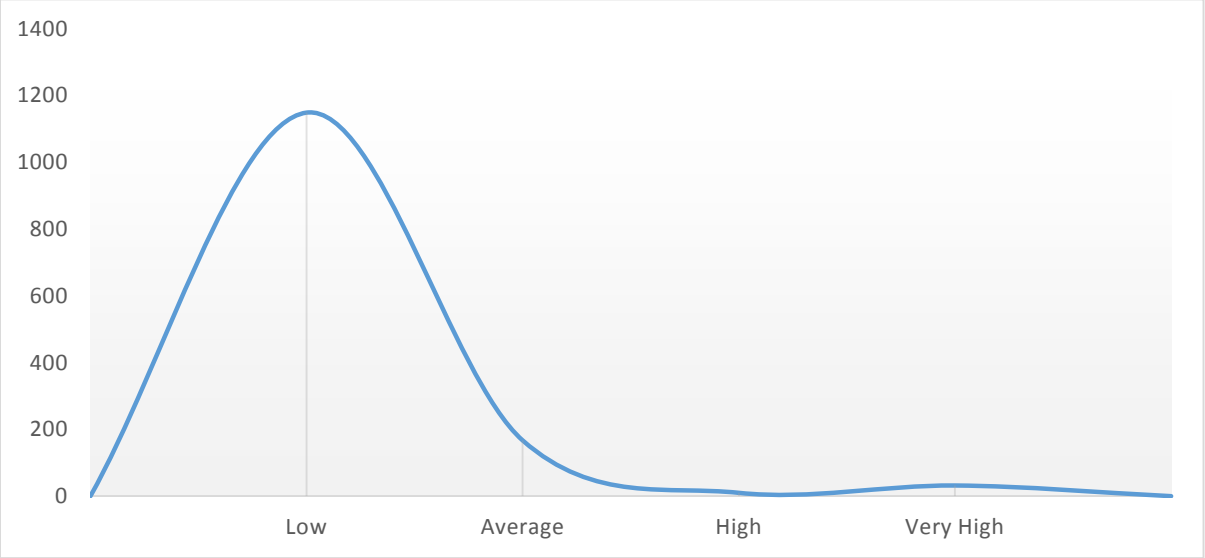


Figure 121: Cyclomatic Complexity Distribution

Cyclomatic Complexity Distribution	Current total	Previous total	% on total elements
Low Complexity	1,003	819	87.9 %
Average Complexity	96	97	8.41 %
High Complexity	10	16	0.88 %
Very High Complexity	32	32	2.80 %

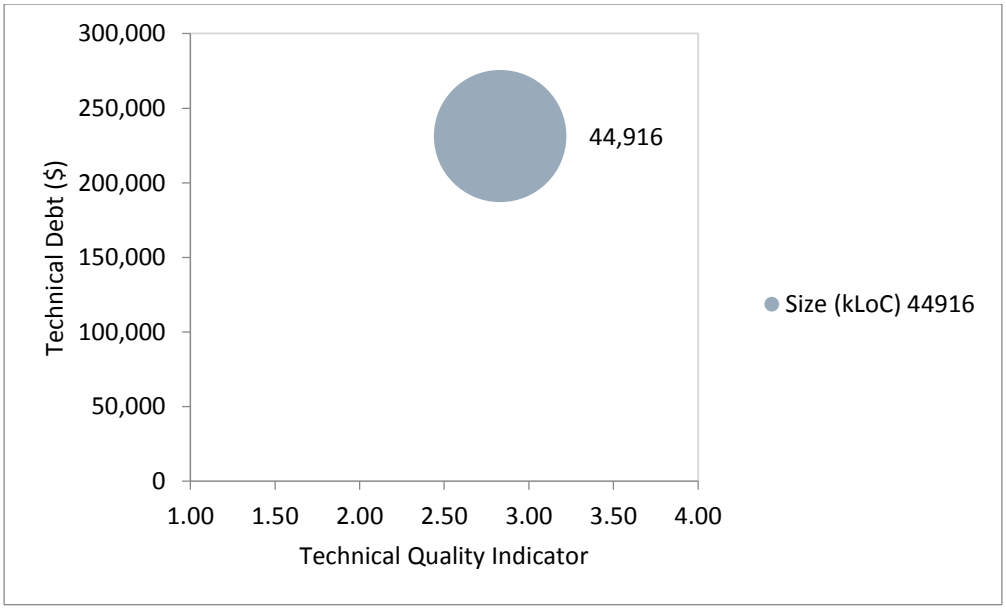
Technical Debt

The complexity of MHTC421 application has been converted into Technical Debt – the cost of fixing the structural quality violations that cause serious business disruption. The data on technical debt provides an objective, empirical frame of reference for the developer community. They also provide a platform for characterizing the management tradeoffs between expending resources on correcting weaknesses in the source code versus risking the problems these flaws may cause such as outages or security breaches.

If necessary, this value can be benchmarked with The CAST Appmarq repository. The CAST Appmarq repository provides a unique opportunity to calculate the Technical Debt across different technologies, based on the number of engineering flaws and violations of good architectural and coding practices in the source code. A parameterized formula for calculating the Technical Debt of each application in Appmarq is based on the percent of violations to be remediated at each level of severity, the time required to fix a violation, and the burdened hourly rate of a developer. Please see the Appendix for details on how Technical Debt is calculated at CAST.

CAST AIP categorizes violations into low, medium and high severity. The technical Debt calculation assumes that only 50% of high-severity violations, 25% of medium-severity violations, and 10% of low severity violations require fixing to prevent business disruption. With this in mind, the formula for technical debt becomes:

Technical Debt of MHTC421 = **231,338 $**



How Can Technology Address Application Quality Challenges?

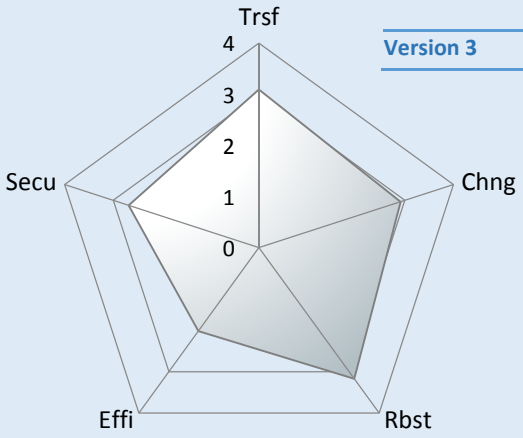
The quality attributes of an application can be characterized by the quality attributes of its component parts no more than the attributes of a molecule can be characterized by the attributes of its constituent atoms. Since high quality components do not equate to a high quality system in any field of engineering, code quality, although necessary, is not sufficient to ensure high quality applications. Organizations need the help of application quality diagnostic tools which can discover inter-component issues and measure the internal quality of the application across its tiers.

There are numerous commercial, freeware, and open source tools available that measure code quality specific to a programming language and are often integrated into Integrated Development Environments (IDEs). These tools are becoming standard components of every developer's toolset since they provide quick feedback during the coding and unit test process. However, these tools are not sufficient to address application quality since they cannot evaluate interactions across the various languages, technologies, and tiers of an application.

Technology that measures application quality analyzes the integrated software produced by a build once the code is checked into a central repository by all the developers. In addition to analyzing each component, application quality technology analyzes their interactions for the types of problems described in earlier sections. Moreover, application quality trends can be compared across builds or releases to monitor the progress against application quality objectives and evaluate the risks posed by the application.

Application quality measurement tools provide several benefits for both the development team and management:

- ***Visibility across application(s):*** Consistent and continuous analysis of all core business applications provides executives with the metrics and information needed to better manage their portfolio of applications and projects.
- ***Analysis of the internal quality of an application***: Reviewing the integrated software system for quality in order to detect architectural and structural problems that hide in interactions between tiers, provides application or project managers with continual status about application quality and risk.
- ***Team performance:*** Since a detailed knowledge of the whole system is usually beyond any individual developer's capabilities, analyzing application quality helps improves developer skills, the team's breadth of application knowledge, and the efficiency of team performance.



	TQI	Robu.	Efcy.	Secu.	Trans.	Chang.
Version 3	2.83	3.17	2.02	2.68	3.09	2.91

STATISTICS ON VIOLATIONS

Name	Number
Critical Violations	59
per File	0.27
per kLOC	1.31
Complex Objects	42
with violations	0

A dynamic business environment, new technology, and multiple sourcing options, amplify the complexity of business application software. Since even the most talented developers can no longer know all the nuances of all the different languages, technologies, and tiers in an application, their capability needs to be augmented by automated tools to evaluate the entire application. Without such assistance, defects hidden in the interactions between

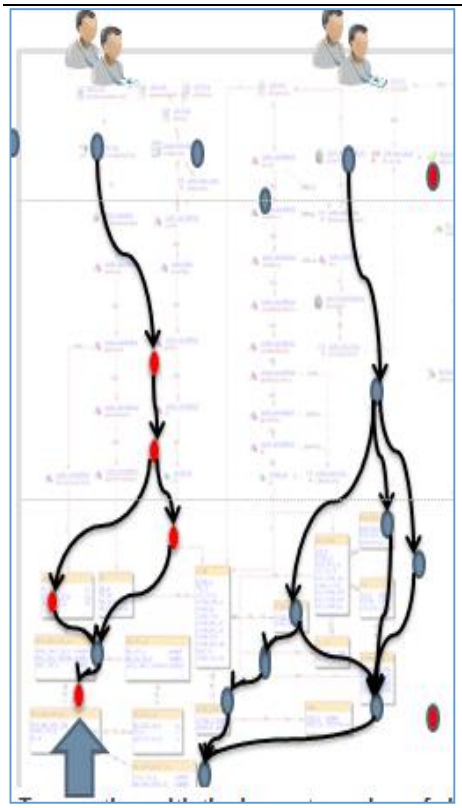
application tiers will place the business at risk for the outages, degraded service, security breaches, and corrupted data that are caused by poor quality applications.

Potential Points of Failures: Critical rules

The CAST AIP quality model assess automatically the application and raise the main issue of the application through a weighted aggregation of more than +1000 rules across the different technology. The below list represent the different rules which contain some violation on some component which can create some abnormal behavior during the execusion of the application.

Potential Points of Failures: Transaction wide Risk Index

Transaction Risk Index (TRI) enables easy identification of the riskiset transactions within the application

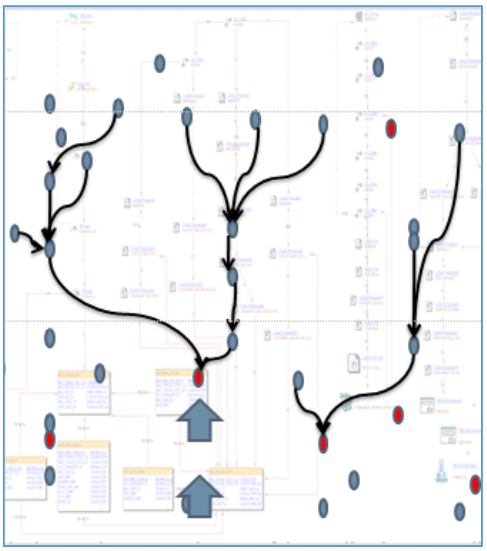


Transaction Wide Risk Index (TwRI) is an indicator of the riskiest transactions of the application. The TwRI number reflects the cumulative risk of the transaction based on the risk in the individual objects contributing to the transaction; in the below list the focus is on the efficiency of the application. The TwRI is calculated as a function of the rules violated, their weight/criticality, and the frequency of the violation across all objects in the path of the transaction. TwRI is a powerful metric to identify, prioritize and ultimately remediate riskiest transactions and their objects.

Transaction Entry Point	TRI

Potential Point of Failures: Propagated Risk Index

Propagated Risk Index (PRI) enables easy identification of the riskiset objects/artifacts within the application



Propagated Risk Index (PRI) is a measurement of the riskiest artifacts or objects of the application along the Health Factors of Robustness, Performance and Security.

PRI takes into account the intrinsic risk of the component coupled with the level of use of the given object in the transaction. It systematically helps aggregate risk of the application in a relative manner allowing for identification, prioritization, and ultimately remediation of the riskiest objects.

The PRI number reflects the cummulative risk of the object based on its relationships and interdependencies. The PRI is calculated as a function of the rules violated, their weight/criticality, and the frequency of the violation.

The Top 14 objects with the highest PRI are:

Artefact Name	PRI
edu.wpi.mhtc.util.persistence.PSqlStringMappedJdbcCall.buildQuery	3,200
edu.wpi.mhtc.util.helpers.UnZip.unZipIt	1,160
edu.wpi.mhtc.util.pipeline.wrappers.URLDownload.HTTPDownload	960
edu.wpi.mhtc.util.helpers.MD5.getMD5	800
edu.wpi.mhtc.util.pipeline.wrappers.IPEDSDownload.download	700
edu.wpi.mhtc.util.pipeline.wrappers.BLSDownload.getBLS	540
edu.wpi.mhtc.util.pipeline.scheduler.TalendJob.runPipeline	480
edu.wpi.mhtc.util.pipeline.wrappers.WebTableWrapper.downloadHtmlUnit	440
[E:\CASTMS\MHTC421\Deploy\MHTC421\Package 3\src\main\webapp\WEB-INF\views\unifiedHeader.jsp]	360

edu.wpi.mhtc.dao.admin.ScheduleDAOImpl.getSchedStatus	320
edu.wpi.mhtc.dao.dashboard.StatsServiceJDBC.getDataForState	320
edu.wpi.mhtc.util.pipeline.wrappers.URLDownload.getText	320
edu.wpi.mhtc.util.pipeline.wrappers.WebTableWrapper.download	320
edu.wpi.mhtc.util.pipeline.parser.TextParser.getHeaderColumnNames	300

Strenghts and Weaknesses

Technical Criteria Name	Grade
Architecture - Multi-Layers and Data Access	1.00
Architecture - Object-level Dependencies	3.37
Architecture - OS and Platform Independence	4.00
Architecture - Reuse	1.97
Complexity - Algorithmic and Control Structure Complexity	3.86

Measures of Robustness

The root causes of poor reliability are found in a combination of non- compliance with good architectural and coding practices. This non-compliance can be detected by measuring the static quality attributes of an application. Assessing the static attributes underlying an application's reliability provides an estimate of the level of business risk and the likelihood of potential application failures and defects the application will experience when placed in operation.

Technical Criteria Name	Grade
Architecture - Multi-Layers and Data Access	1.00
Architecture - Object-level Dependencies	3.37
Architecture - OS and Platform Independence	4.00
Architecture - Reuse	1.97
Complexity - Algorithmic and Control Structure Complexity	3.86

Measures: Assessing reliability requires checks of at least the following software engineering best practices and technical attributes:

- Application Architecture Practices
  - Multi-layer design compliance
  - Coupling ratio
  - Component or pattern re-use ratio
- Coding Practices
  - Error & Exception handling (for all layers GUI, Logic & Data)
  - Compliance with Object-Oriented and Structured Programming best practices (when applicable)
- Complexity
  - Transaction complexity level
  - Complexity of algorithms
  - Complexity of programming practices
  - Dirty programming

Depending on the application architecture and the third-party components used (such as external libraries or frameworks), custom checks should be defined along the lines drawn by the above list of best practices to ensure a better assessment of the reliability of the delivered software.

Rule Names	Count

Measures of Performance & Efficiency

As with Reliability, the causes of performance inefficiency are often found in violations of good architectural and coding practice which can be detected by measuring the static quality attributes of an application. These static attributes predict potential operational performance bottlenecks and future scalability problems, especially for applications requiring high execution speed for handling complex algorithms or huge volumes of data.

Technical Criteria Name	Grade
Complexity - Dynamic Instantiation	4.00
Efficiency - Expensive Calls in Loops	1.47

Efficiency - Memory, Network and Disk Space Management	1.17

Measures: Assessing performance efficiency requires checking at least the following software engineering best practices and technical attributes:

- Application Architecture Practices
  - Appropriate interactions with expensive and/or remote resources
  - Data access performance and data management
  - Memory, network and disk space management
- Coding Practices
  - Compliance with Object-Oriented and Structured Programming best practices
  - Compliance with SQL best practices

Measures of Security

Most security vulnerabilities result from poor coding and architectural practices such as SQL injection or cross-site scripting. These are well documented in lists maintained by CWE http://cwe.mitre.org/, and CERT.

Technical Criteria Name	Grade
Architecture - Multi-Layers and Data Access	1.00
Architecture - OS and Platform Independence	4.00
Efficiency - Memory, Network and Disk Space Management	1.17
Programming Practices - Error and Exception Handling	1.00
Programming Practices - Unexpected Behavior	4.00

Measures of Maintainability

Maintainability includes concepts of modularity, clarity, changeability, testability, reusability, and transferability from one development team to another. These do not take the form of critical issues at the code level. Rather, poor maintainability is typically the result of thousands of minor violations with best practices around documentation, complexity avoidance strategy, and basic programming practices that make the difference between clean and easy to read code vs. ugly and difficult to read code.

Transferability

Technical Criteria Name	Grade
Architecture - Object-level Dependencies	3.37
Complexity - Algorithmic and Control Structure Complexity	3.86
Complexity - Dynamic Instantiation	4.00
Complexity - OO Inheritance and Polymorphism	4.00
Dead code (static)	2.15

Changeability

Technical Criteria Name	Grade
Architecture - Multi-Layers and Data Access	1.00
Architecture - Object-level Dependencies	3.37
Architecture - OS and Platform Independence	4.00
Architecture - Reuse	1.97
Complexity - Algorithmic and Control Structure Complexity	3.86

Measures: Assessing maintainability requires checking the following software engineering best practices and technical attributes:

- Application Architecture Practices
  - Multi-layer design compliance
  - Coupling ratio
  - Component or pattern re-use ratio
- Programming Practices (code level)
  - Compliance with Object-Oriented and Structured Programming best practices (when applicable)
- Complexity
  - Complexity level of transactions
  - Complexity of algorithms
  - Complexity of programming practices
  - Dirty programming
- Documentation
  - Code readability
  - Architecture, Programs and Code documentation embedded in source code
  - Source code file organization cleanliness
- Portability
  - Hardware, OS, middleware, software components and database independence

Security Assessment Overview

Here we list down all the security vulnerabilities identified by CAST AIP.

TOP TECHNICAL CRITERIA THAT MOST IMPACT THE SECURITY **2.68**

Technical criterion name	Total violations	Total check	Grade
Programming Practices - Error and Exception Handling	133	10,216	1.00
Efficiency - Memory, Network and Disk Space Management	127	8,078	1.17
Secure Coding - Encapsulation	44	1,486	3.16
Architecture - OS and Platform Independence	25	6,770	4.00
Architecture - Multi-Layers and Data Access	2	277	1.00

*Formula – Sum(rule weight x technical criterion weight) * (4 – technical criterion grade)*

Rules list below display the most impacting rules for the current snapshot. Rules are sorted according to the grade and the weight of the rule. In other terms, on top of the list, you will see the rules that have a big impact (low grade * big weight) and the rules that are difficult to correct (lots of violations to be correct).

Top rules that most impact the SECURITY

Rule Name	Current Violations	Grade
Avoid declaring throwing an exception and not throwing it	58	1.93
Declare as static all Methods not using Instance Fields	54	1.97
Pages should use error handling page	38	1.00

Use of style sheets	37	1.00
Avoid declaring Instance Variables without defined access type	28	1.83
Avoid using 'java.io.File'	21	3.86
Avoid using 'Throwable.printStackTrace()' within a try catch block	12	3.60
Avoid using 'System.err' and 'System.out' outside a try catch block	9	3.76
Close the outermost stream ASAP	8	1.17
Avoid large number of String concatenation	8	3.81

*Formula – (quality rule weight x technical criterion weight) * (4 – rule grade)*

Software Security Standards

CAST AIP is aligned to capture the Security requirements listed by CWE (http://cwe.mitre.org/), and CERT as top Security weaknesses.

Summary of Security Violations

HIGH LEVEL AREA	NUMBER OF VIOLATIONS
Insecure Interaction Between Component	0
Risky Resource Management	0
Porous Defences	0
Security vulnerabilities identified by OWASP & CWE (not part of top-25)	0

Mapping CAST Rules to CWE Most Dangerous Software Errors

The rules categorized into four high-level areas listed below:

- Insecure Interaction Between Components
- Risky Resource Management
- Porous Defenses
- Security vulnerabilities identified by OWASP & CWE (not part of top-25)

Insecure Interaction Between Components

These weaknesses are related to insecure ways in which data is sent and received between separate components, modules, programs, processes, threads, or systems.

Rank	CWE ID	Name	Recommendation/Mitigation/Comments	Corresponding CAST Rule
[1]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	Checking for SQL Injection	Avoid SQL injection vulnerabilities
[2]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	Checking for OS Command Injection	Avoid OS command injection vulnerabilities
[4]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	Checking for Cross-site scripting	Avoid cross-site scripting vulnerabilities
[9]	CWE-434	Unrestricted Upload of File with Dangerous Type	Input Validation	Extend existing rules - Avoid non standard file extensions, Avoid file path manipulation vulnerabilities, Avoid XPath injection vulnerabilities
[12]	CWE-352	Cross-Site Request Forgery (CSRF)	Ensure that application is free of cross-site scripting issues (CWE-79), because most CSRF defenses can be bypassed using attacker-controlled script.	Avoid cross-site scripting vulnerabilities
[22]	CWE-601	URL Redirection to Untrusted Site ('Open Redirect')	Checking for Cross-site scripting	Avoid cross-site scripting vulnerabilities

Risky Resource Management

The weaknesses in this category are related to ways in which software does not properly manage the creation, usage, transfer, or destruction of important system resources.

Rank	CWE ID	Name	Recommendation/Mitigation/Comments	Corresponding CAST Rule
[3]	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	Perform input validation on any numeric input by ensuring that it is within the expected range. Enforce that the input meets both the minimum and maximum requirements for the expected range.	Extend existing rules - Avoid using getopt() function, Never use sprintf() function or vsprintf() function, Never perform C cast between incompatible class pointers, Avoid using static_cast on class/struct pointers
[13]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	Checking for file path manipulation	Avoid file path manipulation vulnerabilities

[14]	CWE-494	Download of Code Without Integrity Check	Check Download Code Integrity	Extend existing rules - Avoid non standard file extensions, Avoid file path manipulation vulnerabilities, Avoid XPath injection vulnerabilities
[16]	CWE-829	Inclusion of Functionality from Untrusted Control Sphere	When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.	Extend existing rules - Avoid XPath injection vulnerabilities
[18]	CWE-676	Use of Potentially Dangerous Function	Checking for programming best practices	Avoid using snprintf() function, Avoid using realpath() function, Avoid using the scanf() function, etc
[20]	CWE-131	Incorrect Calculation of Buffer Size	Perform input validation on any numeric input by ensuring that it is within the expected range. Enforce that the input meets both the minimum and maximum requirements for the expected range.	Extend existing rules - Never use sprintf() function or vsprintf() function, Never perform C cast between incompatible class pointers, Avoid using static_cast on class/struct pointers
[23]	CWE-134	Uncontrolled Format String	Whenever possible, use functions that do not support the %n operator in format strings.	Extend existing rules - Never use sprintf() function or vsprintf() function, Avoid using the scanf() function, etc
[24]	CWE-190	Integer Overflow or Wraparound	Perform input validation on any numeric input by ensuring that it is within the expected range. Enforce that the input meets both the minimum and maximum requirements for the expected range.	Extend existing rules - Avoid using getopt() function

Porous Defenses

The weaknesses in this category are related to defensive techniques that are often misused, abused, or just plain ignored.

Rank	CWE ID	Name	Recommendation/Mitigation/Comments	Corresponding CAST Rule

[5]	CWE-306	Missing Authentication for Critical Function	Avoid implementing custom authentication routines and consider using authentication capabilities as provided by the surrounding framework, operating system, or environment.	Extend existing rules - Avoid cross-site scripting vulnerabilities, Avoid LDAP injection vulnerabilities
[6]	CWE-862	Missing Authorization	Users should not be able to access any unauthorized functionality or information by simply requesting direct access to that page.	Extend existing rules - Avoid cross-site scripting vulnerabilities, Avoid LDAP injection vulnerabilities
[7]	CWE-798	Use of Hard-coded Credentials	Store passwords, keys, and other credentials outside of the code in a strongly-protected, encrypted configuration file or database that is protected from access by all outsiders, including other local users on the same system.	Extend existing rules - Avoid LDAP injection vulnerabilities
[8]	CWE-311	Missing Encryption of Sensitive Data	Periodically ensure that you aren't using obsolete cryptography. Avoid using old encryption techniques using MD4, MD5, SHA1, DES, and other algorithms that were once regarded as strong.	Extend existing rules - Avoid cross-site scripting vulnerabilities, Avoid LDAP injection vulnerabilities
[10]	CWE-807	Reliance on Untrusted Inputs in a Security Decision	consider getcookies as unsafe	Avoid cross-site scripting vulnerabilities
[11]	CWE-250	Execution with Unnecessary Privileges	Checking for privileges being appropriately implemented based on the scenario/usecase. Perform extensive input validation and canonicalization to minimize the chances of introducing a separate vulnerability.	Extend existing rules - Avoid cross-site scripting vulnerabilities, Avoid LDAP injection vulnerabilities
[15]	CWE-863	Incorrect Authorization	consider getcookies as unsafe	Avoid cross-site scripting vulnerabilities
[17]	CWE-732	Incorrect Permission Assignment for Critical Resource	Path manipulation	Avoid file path manipulation vulnerabilities
[21]	CWE-307	Improper Restriction of Excessive Authentication Attempts	Check login implementation	Extend existing rules - Avoid direct access to database Procedures/Functions, User Interface elements must not use directly the database
[25]	CWE-759	Use of a One-Way Hash without a Salt	Checking for programming best practices	Extend rules - Avoid using Hashtable, Avoid classes overriding only equals() or only hashCode()

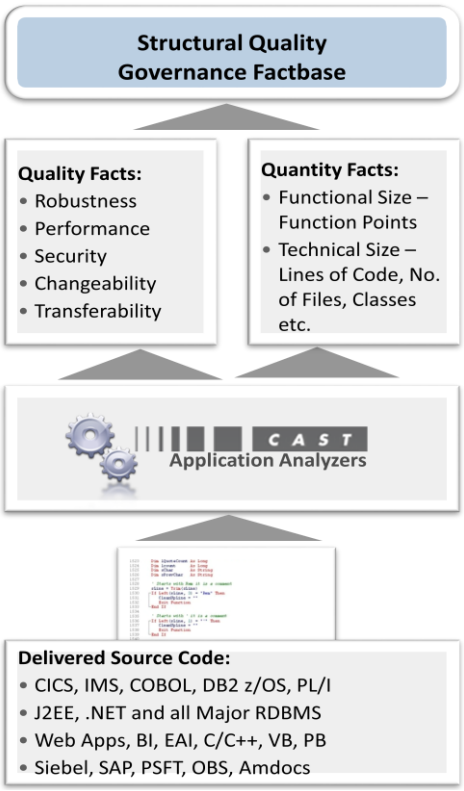
Security vulnerabilities identified by OWASP & CWE (not part of top-25)

CWE ID	Name	Recommendation/Mitigation/Comments	Corresponding CAST Rule

CWE-20:	Improper Input Validation	Checking for best programming practices	Avoid SQL injection vulnerabilities, Avoid XPath injection vulnerabilities, Avoid cross-site scripting vulnerabilities
CWE-116:	Improper Encoding or Escaping of Output	Checking for best programming practices	Avoid SQL injection vulnerabilities, Avoid OS command injection vulnerabilities, Avoid cross-site scripting vulnerabilities
CWE-90	LDAP Injection	Checking for LDAP injection	Avoid LDAP injection vulnerabilities
CWE-91	XPATH Injection	Checking for XPATH injection	Avoid XPath injection vulnerabilities
CWE-73:	External Control of File Name or Path	Checking for file path manipulation	Avoid file path manipulation vulnerabilities
CWE-99:	Improper Control of Resource Identifiers ('Resource Injection')	Checking for best programming practices	Avoid file path manipulation vulnerabilities
CWE-117:	Improper Output Neutralization for Logs	Checking for log forging	Avoid Log forging vulnerabilities

Security Weaknesses Spotted

CAST AIP Assessment Approach Overview



This assessment is an effort to determine the overall quality of the application MHTC421 and identify any risks that may be inherent in the application. The assessment looks at the implementation of MHTC421 to determine whether the application is constructed according to industry best practices, follows best practices for software engineering, and is maintainable.

This assessment is focused solely on the Source code and Database structure with no view to functionality provided by backend services.

The CAST AIP is the industry leading automated code analysis platform, with coverage of all major development tools and languages. CAST AIP automatically scans and analyzes all of the source code and database elements that are part of an Enterprise system. CAST AIP applies over 1000+ metrics based on standards and measurements developed by the Software Engineering Institute (SEI), International Standards Organization (ISO), Consortium for IT Software Quality (CISQ), and Institute of Electrical and Electronics Engineers (IEEE). These metrics objectively measure software for the quality and quantity of work.

CAST AIP provides Application Analysts the ability to examine and drill down on critical application characteristics and attributes. The primary Application Health Factors that are addressed are:

Health Factor	Description	Example business benefits

Robustness	Attributes that affect the stability of the application and the likelihood of introducing defects when modifying it	• Improves availability of the business function or service • Reduces risk of loss due to operational malfunction • Reduces cost of application ownership by reducing rework
Performance	Attributes that affect the performance of an application	• Reduces risk of losing customers from poor service or response • Improves productivity of those who use the application • Increases speed of making decisions and providing information • Improves ability to scale application to support business growth
Security	Attributes that affect an application's ability to prevent unauthorized intrusions	• Improves protection of competitive information-based assets • Reduces risk of loss in customer confidence or financial damages • Improves compliance with security-related standards and mandates
Transferability	Attributes that allow new teams or members to quickly understand and work with an application	• Reduces inefficiency in transferring application work between teams • Reduces learning curves • Reduces lock-in to suppliers
Changeability	Attributes that make an application easier and quicker to modify	• Improves business agility in responding to markets or customers • Reduces cost of ownership by reducing modification effort

Understanding Quality Indicators, Quality Rules

CAST AIP has 1000+ quality rules and each rule produces a Grade. Depending on the impact the grades are aggregated into high level Indicators: **Quality indicators** and **Best practices indicators**.

Each aggregation is a weighted average of the contributing metrics grades where certain metric grades are flagged critical, i.e. it is nearly a defect. We talk about **Critical Violations**.

Quality Indicators

The structure, classification and terminology are from the ISO 9126- 3 and the subsequent ISO 25000:2005 quality model. The main focus is on internal structural quality. Subcategories have been created to handle specific areas like business application architecture and technical characteristics such as data access and manipulation or the notion of transactions. The dependence tree between software quality characteristics and their measurable attributes is represented in the following diagram, where each of the 5 characteristics that matter for the user or owner of the business system depends on measurable attributes: Application Architecture Practices, Coding Practices, Application Complexity, Documentation, Portability, and Technical & Functional Volume.

Quality Indicator	Description
Performance / Efficiency	The source code and software architecture attributes are the elements that ensure high performance once the application is in run-time mode. Efficiency is especially important for applications in high execution speed environments such as algorithmic or transactional processing where performance and scalability are paramount. An analysis of source code efficiency and scalability provides a clear picture of the latent business risks and the harm they can cause to customer satisfaction due to response-time degradation.
Robustness / Reliability	An attribute of resiliency and structural solidity. Reliability measures the level of risk and the likelihood of potential application failures. It also measures the defects injected due to modifications made to the software (its "stability" as termed by ISO). The goal for checking and monitoring Reliability is to reduce and prevent application downtime, application outages and errors that directly affect users, and enhance the image of IT and its impact on a company's business performance.
Security	A measure of the likelihood of potential security breaches due to poor coding and architectural practices. This quantifies the risk of encountering critical vulnerabilities that damage the business and provides a list of prevention measures.
Transferability	The effort necessary to diagnose the cause of a failure or section of code to be modified. It establishes the level of dependency on specific developers
Changeability	The effort necessary to modify the source code. It establishes the level of responsiveness to business-driven change requests
TQI	Total Quality Index (TQI) is computed on all the measures made by the CAST AIP

Best practices Indicators

Health Factor	Description
Programming Practices	Measures the level of compliance of the application to coding best practices. Compliance to best practices reduces risks of failures in production and improves productivity through increased readability and reduced debugging.
Architectural Design	Measures the level of compliance of the application to software architecture and design rules. Compliance to architecture rules improves productivity through better use of existing frameworks and code and reduced debugging.
Documentation	Measures the level of compliance of the application to code documentation best practices. Compliance to documentation best practices improves productivity through increased readability and faster understanding of source code.

The risk level of a grade shall be assessed according to the below scale

Scale	Risk Level
4	Low Risk
3	Moderate Risk
2	High Risk
1	Very High Risk

Importance of measuring all layers of an application

Measuring the technical quality of business software applications is evolving from an art to a science with the availability of software tools that automate the process of code analysis. However, it is critical to understand that there are two categories of software quality with very different implications for operational performance. The first category is Code Quality which measures individual or small collections of coded components written in a single language and occupying a single tier (e.g., user interface, logic, or data) in an application. The second category, Application Quality, analyzes the software across all of the application's languages, tiers, and technologies to measure how well all an application's components come together to create its operational performance and overall maintainability.

Although the code quality of individual components is important, by itself it will not ensure the overall quality of the application. Quality is not an intrinsic property of code: the exact same piece of code can be excellent in quality or highly dangerous depending on the context in which it operates. Ignoring the larger context in which the code operates – the multitude of connections with other code, databases, middleware, and APIs – will often generate a large number of false positives.

Today's business applications are complex, built in multiple languages on multiple technologies. Even more challenging, these applications usually interact with other applications built on different technologies. Analyzing the quality of modern applications is monstrously complex and can only be accomplished with automated software that analyzes the inner structure of all components and evaluates their interactions in the context of the entire business application.

Typical application quality problems are listed below to clarify the distinction between application and code quality. Performance testing alone is not sufficient to detect these application quality problems.

Bypassing the Architecture.

Components in one tier of a multi-tier application are typically designed to access components in another tier only through an intermediate "traffic management" component. Bypassing this traffic management component will usually result in a cascade of problems.

Failure to Control Processing Volumes.

Applications can behave erratically when they fail to control the amount of data or processing they allow. This problem is often caused by a failure to incorporate controls in each of several different architectural tiers.

Application Resource Imbalances.

When database resources in a connection pool are mismatched with the number of request threads from an application, resource contention will block the threads until a resource becomes available, tying up CPU resources with the waiting threads and slowing application response times to a crawl.

Security Weaknesses.

Applications are vulnerable to security attacks when they lack appropriate sanitization checks on user inputs in all relevant tiers of the application.

Lack of Defensive Mechanisms.

Since the developers implementing one tier cannot anticipate every situation, they must implement defensive code that sustains the application's performance in the face of stresses or failures affecting other tiers. Tiers that lack these defensive structures are fragile because they fail to protect themselves from problems in their interaction with other tiers. Each of these application quality problems will result in unpredictable application performance, business disruption, data corruption, and make it difficult to alter the application in response to pressing business needs. Reliably detecting these problems requires an analysis of each application component in the context of the entire application as a whole – an evaluation of application rather than code quality.

Technical Debt Calculation in the CAST AIP

Purpose

Purpose of this specification is to add new indicators to the CAST AIP dashboard.

1) **Total Technical Debt per Application**
2) **Total Technical Debt per Module**
3) **Technical Debt Added in Current Release of the Application**
4) **Technical Debt Removed in Current Release of the Application**

Note: These should be calculated at module and application level and can be summed up to the system level in the portal.

Calculation of Technical Debt per Module and Application

1) **Total Technical Debt per Module and Application =**

{ (% of low severity violations to be fixed X # of low severity violations in Application and Module) X (Weighted time, in hours, for fixing low severity violations) +

(% of medium severity violations to be fixed X # of medium severity violations in Application and Module) X (Weighted time, in hours, for fixing medium severity violations) +

(% of high severity violations to be fixed X # of high severity violations in Application and Module) X (Weighted time, in hours, for fixing high severity violations) **}** X

Cost per staff hour to fix violations

2) **Technical Debt Added in Current Release per Application =**

{ (% of low severity violations to be fixed X # of low severity violations added in current release of Application) X (Weighted time, in hours, for fixing low severity violations) +

(% of medium severity violations to be fixed X # of medium severity violations added in current release of Application) X (Weighted time, in hours, for fixing medium severity violations) +

(% of high severity violations to be fixed X # of high severity violations added in current release of Application) X (Weighted time, in hours, for fixing high severity violations) **}** X

Cost per staff hour to fix violations

3) **Technical Debt Removed in Current Release per Application =**

{ (% of low severity violations to be fixed X # of low severity violations removed in current release of Application) X (Weighted time, in hours, for fixing low severity violations) +

(% of medium severity violations to be fixed X # of medium severity violations removed in current release of Application) X (Weighted time, in hours, for fixing medium severity violations) +

(% of high severity violations to be fixed X # of high severity violations removed in current release of Application) X (Weighted time, in hours, for fixing high severity violations) **}** X

Cost per staff hour to fix violations

Definition of Variables

Variable Name	Description	Configurable	Default Value
% of low severity violations to be fixed	Only a portion of the low severity violations will be fixed	Yes	0%
# of low severity violations	Actual # of low severity (level 1,2,3) violations across all health factors	No (comes directly from analysis)	Not Applicable
# of low severity violations added in current release	Actual # of low severity (level 1,2,3) violations across all health factors added in current release	No (comes directly from analysis)	Not Applicable
# of low severity violations removed in current release	Actual # of low severity (level 1,2,3) violations across all health factors removed in current release	No (comes directly from analysis)	Not Applicable

Variable Name	Description	Configurable	Default Value
% of medium severity violations	Only a portion of the medium severity violations will be fixed	Yes	50%
# of medium severity violations	Actual # of medium severity (level 4,5,6) violations across all health factors	No (comes directly from analysis)	Not Applicable
# of medium severity violations added in current release	Actual # of medium severity (level 4,5,6) violations across all health factors added in current release	No (comes directly from analysis)	Not Applicable
# of medium severity violations removed in current release	Actual # of medium severity (level 4,5,6) violations across all health factors added in current release	No (comes directly from analysis)	Not Applicable
% of high severity violations	Only a portion of the high severity violations will be fixed	Yes	100%
# of high severity violations	Actual # of high severity (level 7,8,9) violations across all health factors	No (comes directly from analysis)	Not Applicable
# of high severity violations added in current release	Actual # of high severity (level 7,8,9) violations across all health factors added in current release	No (comes directly from analysis)	Not Applicable
# of high severity violations removed in current release	Actual # of high severity (level 7,8,9) violations across all health factors added in current release	No (comes directly from analysis)	Not Applicable
Weighted time, in hours, for fixing **LOW** severity violation	Not all violations will need the same amount of time, hence we take the weighted time to fix the violations. Weighted based on the distribution of level of difficulty to fix violations. Violations will be categorized as follows: 1) Easy 2) Hard 3) Very Hard Wt. time to fix low severity violations=		

Variable Name	Description	Configurable	Default Value
	(Low_%Easy **X** Low_Time_Easy) **+** (Low_%Hard **X** Low_Time_Hard) **+** **(**Low_%Very_Hard **X** Low_Time_Very_Hard)		
	Low_%Easy = % of violations which are "Easy"	Yes	90%
	Low_Time_Easy = Time take for fixing "Easy" violations	Yes	0.5 hour
	Low_%Hard = % of violations which are "Hard"	Yes	9%
	Low_Time_Hard = Time take for fixing "Hard" violations	Yes	1 hour
	Low_%Very_Hard = % of violations which are "Very_Hard"	Yes	1%
	Low_Time_Very_Hard = Time take for fixing "Very_Hard" violations	Yes	8 hours
Weighted time, in hours, for fixing **MEDIUM** severity violation	Not all violations will need the same amount of time, hence we take the weighted time to fix the violations. Weighted based on the distribution of level of difficulty to fix violations. Violations will be categorized as follows: 4) Easy 5) Hard 6) Very Hard Wt. time to fix low severity violations= (Medium_%Easy **X** Medium_Time_Easy) **+** (Medium_%Hard **X** Medium_Time_Hard) **+** **(**Medium_%Very_Hard **X** Medium_Time_Very_Hard)		
	Medium_%Easy = % of violations which are "Easy"	Yes	90%
	Medium_Time_Easy = Time take for fixing "Easy" violations	Yes	0.5 hour
	Medium _%Hard = % of violations which are "Hard"	Yes	9%

Variable Name	Description	Configurable	Default Value
	Medium _Time_Hard = Time take for fixing "Hard" violations	Yes	4 hour
	Medium _%Very_Hard = % of violations which are "Very_Hard"	Yes	1%
	Medium _Time_Very_Hard = Time take for fixing "Very_Hard" violations	Yes	16 hours
Weighted time, in hours, for fixing **HIGH** severity violation	Not all violations will need the same amount of time, hence we take the weighted time to fix the violations. Weighted based on the distribution of level of difficulty to fix violations. Violations will be categorized as follows: 7) Easy 8) Hard 9) Very Hard Wt. time to fix low severity violations= (High_%Easy **X** High _Time_Easy) **+** (High _%Hard **X** High _Time_Hard) **+** **(**High _%Very_Hard **X** High _Time_Very_Hard)		
	High _%Easy = % of violations which are "Easy"	Yes	80%
	High _Time_Easy = Time take for fixing "Easy" violations	Yes	1 hour
	High _%Hard = % of violations which are "Hard"	Yes	19%
	High _Time_Hard = Time take for fixing "Hard" violations	Yes	8 hours
	High _%Very_Hard = % of violations which are "Very_Hard"	Yes	1%
	High _Time_Very_Hard = Time take for fixing "Very_Hard" violations	Yes	24 hours
Cost per hour of developer time	Blended rate of different people who may work on a violation (architect,	Yes	$75/hr

Variable Name	Description	Configurable	Default Value
	lead, developer,QA resource etc.)		