**Project Abstract, What Mic Should I Use?**

**NOTE**: What follows is a journal-style document I wrote as I developed this project. The repository contains different folders marking periodic progress. I’ve assembled a directory here of the most relevant files for review if you’d like to ‘cut to the chase’:

* Final ERD can be found in: 3.11.2023
* Final data warehouse schema script can be found in: 3.11.2023
* A materialized view and cube script used for analytics can be found in: 3.11.2023
* Screenshots of Apache Airflow runs and data visualizations with IBM Cognos can be found in:

3.11.2023 🡪 Screenshots

* Final versions of all Python programs can be found in: 3.11.2023 🡪 Python
* The machine learning application can be found in: 3.16.2023

Hello and thanks for checking in! Having recently completed IBM’s Data Engineering Professional course, I’m eager to apply what I’ve learned to a personal project involving a critical tool of my past trade as an audio engineer: microphones! I’ve only just begun, so there’s not much to see yet, but I’ll be adding to this repository regularly and encourage you to check in occasionally for updates. Because I’ll be working in an environment that is local to my machine, I’ll be using screenshots and pdf’s of reports generated with Power BI to document progress.

My goal is to develop programs, design and develop a database/data warehouse, and automate pipelines to generate source data for an analytics dashboard and Apache Spark machine learning algorithm to answer the question, “What mic should I use?” The intended specific context is the amplification/reinforcement of live music. Because the means is more important to me than the end (at least for now), I make no promises that the results will actually provide even remotely scientific evidence of what mic should be used for what instrument in any given situation! Most of the data will be at least partially randomly generated for lack of an extant source of data available as fuel for my idea. However, I hope to involve the community of audio engineers I’ve gotten to know over the years to ultimately provide a resource of value to the audio community.

To begin, I’ll outline my plan for the basic components of the project:

1. An ERD created with Postgres utilizing a star schema (possibly snowflake schema?), along with a corresponding .sql script.
2. A MySQL staging database/data warehouse with the above Postgres .sql script as schema source.
3. A Python program to randomly generate data for analysis on a daily basis. This program will run via Apache Airflow, i.e. a ‘DAG’, or ‘directed acyclic graph’, and update the staging database with the new records.
4. A DB2 production database/data warehouse. This database will update daily from the staging database through a Python script via Apace Airflow (i.e., a separate, 2nd DAG).
5. An analytics dashboard connected to the production database. I haven’t chosen the platform just yet: IBM’s Cognos or Microsoft’s Power BI.
6. An Apache Spark job that will utilize machine learning to make a classification prediction regarding what mic should be used for what sound source in what situational context. For the moment, I plan to code this with Python in a Jupyter Notebook running on IBM’s Watson Studio

* David

**Progress Update, 2.28.2023**

Good progress thus far! The repo currently contains the following items:

1. A screenshot of an ERD created in pgAdmin (postgres) outlining the schema for the data warehouse. The structure is that of a snowflake schema composed of the primary table ‘factResults’; ‘dimSource’, ‘dimMics’, ‘dimBand’, and ‘dimVenue’ as the dimensions; and ‘flakeMicUsed’ and ‘flakeMembers’ as sub-dimensions of ‘dimSource’, ‘dimMics’, and ‘dimBand’.
2. ‘what-mic-schema.sql’ — this will be used to build the schema of the staging data warehouse (MySQL) and production data warehouse (DB2).
3. ‘records\_generator.py’ — this is the Python program I’ve developed to populate the staging database on day one. It creates all the data necessary to populate the tables as outlined in the ERD as .csv files. It uses a minimum of libraries: ‘random’ and ‘pandas’ (for DataFrame generation).

Other notes: I’m using an Ubuntu-based virtual machine (22.04.3) for development along with two conda environments: one for python development using Spyder as the IDE and one for Apache Airflow (which won’t play nice with Spyder!) to automate pipelines. I’ve also tested connectivity with the staging data warehouse and production data warehouse by moving some dummy data around with the Python libraries ‘mysql-connector-python’ and ‘ibm\_db’. All is working as expected thus far!

**Progress Update, 3.04.2023**

Lots of progress this week! All new files are located in the folder marked with the progress update date (03.04.2023). Here’s what’s included:

* generator\_functions.py: this acts as a module for import purposes
* records\_generator\_alt.py: this uses generator\_functions.py and creates the first run of records for all relevant tables to be loaded into the data warehouses (staging and production)
* init\_data\_warehouses.py: this loads all first-run records created with records\_generator\_alt.py into both the staging data warehouse and the production data warehouse. Both of these programs are run ‘manually’, i.e. not triggered by an automated process
* new\_records\_generator\_alt.py: this is used to create all records beyond the first run. Only records for 4 tables are generated (dimFactResults, dimBand, flakeMembers, and flakeMicUsed) as the other tables do not change. This program is run through an automated pipeline.
* update\_staging\_dw.py: this is used to load newly created records from new-records\_generator\_alt.py into the staging data warehouse (MySQL). This program is run through an automated pipeline.
* update\_production\_dw.py: this is used to load all new records from the staging data warehouse into the production data warehouse (DB2). This program is run manually. Concept: the data engineer checks the staging data warehouse each day to ensure all is well. Once verified, the production data warehouse is updated.
* create\_and\_load.py: this program is written specifically for Apache Airflow. It is the program that runs new\_records\_generator\_alt.py (‘create’) and update\_staging\_dw.py (‘load’). The program is run once per day without intervention from the engineer.
* what-mic-schema.sql: this is an update to the previous what-mic-schema.sql with changes specific to MySQL syntax. It is used as the source schema for the staging data warehouse. It is also used (with slight modification) as the source schema for the production data warehouse.

All in all, this is what I’ve got thus far in pipeline format:

generate records -> insert records into the staging and production data warehouses -> generate new records (daily) and load them into the staging data warehouse (automatic process through Airflow) -> manually check the staging data warehouse (daily) -> manually load any new records from the staging data warehouse into the production data warehouse

Up next: connecting IBM Cognos or Power BI to the production data warehouse for analytics

**Progress Update, 3.11.2023**

Both a staging data warehouse and production data warehouse are now extant with identical data. There are between 6,000 and 7,000 fact records. Here is the generation process I used:

1. The schema was instantiated on both data warehouses. I did this from the command line in mysql (source ~/Documents/what-mic-schema.sql) and through the DB2 UI on the production data warehouse.
2. An initial round of data was created using records\_generator\_alt.py. This is a simulation of ‘day 1’ data. Then, ‘init\_data\_warehouses.py’ was run to populate both the staging and production data warehouses with this data simultaneously.
3. I simulated daily pipeline runs over a period of 2 hours using an Airflow DAG, ‘create\_extract\_load.py’. The DAG ran every 5 minutes for a total of 23 runs, creating, extracting, and loading data into the staging data warehouse with each run. As mentioned above, this came to a total of 6,000 some fact entries. I have provided several screenshots of the Airflow UI showing some basic statistics on these runs.
4. I manually ran some basic SQL statements in MySQL to see that entries were loaded properly.
5. Once I was reasonably certain the MySQL data warehouse was in order, I manually ran ‘update\_production\_dw.py’ to load all new entries created through the DAG runs from the staging data warehouse to the production data warehouse.
6. Once this was complete (it took several minutes since the records were updated to a cloud-based production data warehouse over TCP/IP), I manually ran some basic SQL statements in the DB2 UI and checked them against the results of the staging data warehouse to ensure all was in order.

With both data warehouses populated, I created a materialized view on the production data warehouse with data that could provide readable statistics on mic usage. Though the ‘factResults’ table contains useful data, all fields show id’s, rather than readable names (for example, ‘mic\_id’ rather than ‘Sennheiser 935e’. Of most interest is:

* Average results for mics based on the fields source (instrument), genre (musical style), number of band members, and venue size.

In creating the view, I made two changes to the original data tables:

1. One column was added to ‘factResults’… a numerical representation of the categorical column ‘result’, called ‘result\_numeric’. This allows for categorical machine learning in Spark and the averaging of results for graphical purposes. (Not until I began playing with the data in Cognos did the obvious fact that categories cannot be averaged occur to me!)
2. The reverberation column in ‘dimVenue’ was changed from a string (e.g., “1.67 seconds”) to a float, and the column was renamed ‘reverberation\_seconds’. Being able to view this data in a numerical format is far more useful than that in string format.

Finally, I created a Dashboard in Cognos and created some sample visualizations, largely based on the materialized view, titled ‘mqt\_mic\_results’. I have included screenshots of all tabs of the dashboard. Finally, we have some insight to the original question, ‘What mic should I use?’.

Summary of new and updated file information:

In the folder marked ‘3.11.2023’, one can find updated and new files as follows…

* The postgres-created ERD has been updated to reflect the minor changes to the tables.
* Minor changes were made to all files to accommodate the changes mentioned above in the tables, and to generate a larger number of records per run. This allowed me to create more records with fewer DAG runs. The latest versions of all files are included.
* ‘create\_extract\_load.py’ is a new version of the previously named ‘create\_and\_load.py’ file. It is the exact version used to make the DAG runs detailed above.
* ‘mqt\_mic\_results.sql’ has been included. This was used to create the materialized view on the production data warehouse.
* Multiple screenshots… as mentioned there are screenshots detailing the DAG runs and showing the sample dashboard tabs I’ve created in Cognos.

Up next: I have accomplished most of what I had wanted. What remains is the machine learning algorithm with Spark. I am not a data scientist, so this will not be exhaustive. I plan to use a random forest classification model to make predictions answering questions such as “What result can I expect when using a Shure SM58 for the lead vocalist of an 8-piece rock band performing in a mid-sized venue?”

**Progress Update, 3.16.2023**

I’ve applied a machine learning algorithm using random forest classification in pyspark to the mic data to attempt to predict results of using microphones in given situations. The file is available in the folder ‘3.16.2023’ in notebook format. The materialized view table on the production data warehouse is queried and stored; first as a pandas dataframe, then as a Spark dataframe. The data is broken into training and testing groups; an rfc model is fit to the training data and applied to the test data. Because microphone usage results are randomly generated in my programs — no fields in my data are directly related to results — the algorithm is not effective for making predictions, but it does ‘work’, i.e. the machine makes predictions based on the fields ‘manufacturer’, ‘model’, ‘source’ (instrument), ‘style’ (musical genre), ‘num\_members’ (in the band), and ‘size’ (of the venue). As I stated when setting out on this project, the process is more important to me than the results. Tying results to specific fields would require significantly more complicated data-generation algorithms, and because this is not really the domain of data engineering, I feel it’s not worth the time.

At this point, I’ve accomplished what I had planned to when I devised the project. Next steps: I’d like to see if I can combine the randomly generated results with product reviews pulled from the web. Because I can’t find a free, relevant API to do so, I’m hoping to simply scrape it from individual websites. Because there aren’t too many mics used in my data, this may not be so bad… we’ll see!

* David