

## MusicBase

### ***Proposal***

The music industry is a vast one. When considering for specifics, it can be difficult to narrow down the search. The objective of this project was to consider 3 choices of datasets that were purposed for music and determine if there were any similarities or relationships that could be developed to unify or sync the separate datasets. Of the 3 choices given, Musicbrainz and Discogs were chosen.

A set of Amazon Web Services were utilized to aid in development of this unified database. S3 was purposed for raw data storage while Amazon Redshift was the data warehouse. SQL scripts were created to form the data dictionary and language. GitHub was fundamental for version control and access. LucidChart was used for designing and conceptualizing the schemas. Lastly, QuickSight served as a visualization aspect for the database that allowed for a more proper and meaningful representation of the database and its findings.

### ***Environment Setup***

The Redshift cluster was initiated and a test connection was conducted to ensure that the project would start smoothly. To consider outside users viewing in on the database designed, proper IAM security roles were created and assigned. The datasets were pre-loaded onto pre-assigned S3 buckets purposed for data copying. QuickSight was set up in a later portion of the project after queries had been successfully tested and views created.

The appendix provided below shows everything that was developed through the project. Although not everything in the report may be linked to an entry in the appendix, the appendix will consist of everything that has been considered throughout the project.

### ***Database Design***

To begin the conceptualization of the database, the root of the design was centered on the types of questions that the queries are designed to answer. When trying to relate the data, it was necessary to see the context in which the data would be related. To understand the data, Musicbrainz and Discogs's public database schema was reviewed. The method of relations used was entity-relation-diagrams (ERD). Each of the entities and their attributes had to be understood and filtered to know what types of queries would be possible. Once the relations start to appear and the keys can be distributed or established, the conceptual diagrams for each subset was generated on LucidChart. The conceptual diagram was primarily utilized to map out the flow of the data and see where the attributes are similar and thus which tables can be related with which columns. Once the general map has been drawn out, all the relationships between tables should be understood within each dataset. Following the concept diagrams, the physical diagram for each concept was produced. The "physical" sense of these diagrams are in that it gives you a better look at each column of the tables, where the primary and potential foreign keys, and the *type* of the data is clearly defined. An additional adjustment that occurs in the physical diagram is the normalization of the data. One-to-one relationships are generally equally related so they can be put into the same column. As for many-to-many relationships, a junction table was created in-between to relate the two tables. Fortunately, the Discogs datasets were already normalized and was therefore very data dictionary language friendly. As for the Musicbrainz data, there was a particularly convenient trend. The convenience serves in all the descriptions of

types are id's in the primary artist and release tables that link to the table that describes what that attribute is whether it be genre, area of release, or type of release. Although these types of relationships were somewhat also prevalent in the Discogs database, they were practically used for every other entity in Musicbrainz that were used to describe the artist or the release.

The majority of the relationships that were developed and linked for the physical diagrams were one-to-many with some of them linking to a junction table, which can be abstracted to be representative of a many-to-many relationship. Because the physical diagrams offered an extension to the extent of expressing the keys (primary and foreign) and data types, it allows for the development of the data dictionary. This spreadsheet file\* logs each entities as a separate sheet with each of its attributes that explain their purpose in the database or schema. With these clear definitions and distributions, the schemas become clearer. Once the physical diagrams for each respective dataset were created, the unification dealt with finding similar links and description of the attributes of the entities. The similarities found were in the releases and artists. Considering that the artist would the same logged name – as it should be –, the same was said about the releases. The only difference is that Musicbrainz is a bit more descriptive and accommodates for much larger and holistic storage of data representation. To link the tables across the datasets in the unified physical diagram, one-to-one relationships were demonstrated.

### ***Data Loading***

As mentioned in the proposal, S3 was the software utilized for data storage. Using redshift's copy commands, the data was able to be loaded while simultaneously being compressed for convenient storage. One of many advantages of using redshift is its incredibly efficient and effective copy commands.

To start this process, the tables that were mapped out in the conceptual and physical diagrams must be generated. Because the original Musicbrainz dataset developed by the native company is a bit more complex, the generate\_DDL script was obtained for their site to accommodate for all their descriptions of the data types and entity attributes. The second script used for the Discogs dataset was provided by the instructor. Because of the convenient format of the data dictionary created in the design milestone, the script was able to smoothly run through and generate create table statements with the defined data types and potential constraints. Similar to the Postgres, when loading the data onto the tables, every column expressed in the dataset must be accommodated for in the table. This was learned through the loading phase as some of the tables in the DDL generation script provided by Musicbrainz were actually missing columns that were present in the dataset. Although it took a few debugging attempts to fix a trivial issue, it demonstrated the importance of consistency between the data and your database design.

Before loading the data, the schemas were created on the cluster and during loading, the path to the target schema was set. The data was already pre-loaded onto S3 buckets, providing us with a quicker data loading process. Most of the technical problems that were encountered was due to the data type that was described in the DDL generation scripts. The Discogs dataset was a lot more consistent with the DDL that was created. Musicbrainz encountered more errors because some columns had either long or characters that were accommodated by the defined data type. Because the queries that were designed did not require all of the columns, most of the data loading errors were accommodated by altering the DDL and changing the data type to simply accommodate max amount of vartchars. After a few attempts, the data was able to be successfully loaded onto the tables that were generated for each respective schema.

The testing of the loaded data involved running select \* statements that show 10 random entries of the table. One for each of the tables in each of the schemas were done and run. The tests are provided below.

### ***Data Integration***

Now that the database design had been developed and the data was clearly defined and described, the transformation of the data and tables to accommodate the queries the unified database was purposed to answer is possible.

To begin, it was necessary to brainstorm and consider what problems could occur in the descriptions of the attributes. A particularly common issue would be the naming of the artists or releases. These discrepancies were addressed primarily by using redshift's BTRIM and SPLIT\_PART methods. In addition, the first letter of each word that begins after a space was capitalized using the initcap method to ensure that all the artists or releases were similar throughout both the Discog and Musicbrainz datasets. For the transformations to be possible, a new column for each attribute that was going through change was created to store the new data, while the older column would later be dropped. An user-defined-function (UDF) was created for each column to determine the data type length necessary to accommodate for the new type of data or entry.

The queries that were developed in the design milestone was further expressed to be SQL friendly. Each query was described with the type of aggregate, join, or filters that it would have or need to come to its answers or analysis. Once that was done, the mission for transforming the data became clearer. Punctuation was a big potential differing factor between the two descriptions of similar data across the two datasets. So the BTRIM method mentioned above was used by specifying a trim to get rid of everything that occurs from ' - ' and onward. This big cut

allowed for the data descriptions to be more similar which should allow for more relations. This was done on both the names of the artists and the releases. Because time and year was going to be of use in the queries developed, a cleaner format of the date type was necessary. This was accomplished by taking the entire column, creating a new for 'YYYY', 'MM', and 'DD'. The SPLIT\_PART method was what allowed for the splitting and separation of the time part of the date type and each of the 3 components for the cleaner format of the date type. Once the data and their respective types were either trimmed or corrected, the queries were assessed once again to determine the columns that were needed. Based on the queries that were developed, only required columns were the date for releases and artists, and each id that links to another table. It was a particularly simple database link and design, but it was adequate for developing the queries.

After transforming and determining the columns required for each query, the SQL scripts for running the select-from-where statements were generated and tested. When developing these scripts, it became apparent that the year was required to be separate from the date type. This required for backtracking to the data dictionary and keeping the temporary year column created to for the SPLIT\_PART methods. Once this was fixed, the queries were further developed and tested for error. The queries are provided below. These queries were all mostly aggregates to show the count difference between genre popularity, amount of releases in which format, artists that have the most releases, or the most common artist name in the industry of music. These relationships were what was most clearly seen to be tangible in obtaining during the database designing.

Once the data for each of the respective schemas had been transformed, cleaned, and filtered, the unified schema DDL was generated. This unified schema followed the convention

that was provided by the unified physical diagram that was produced during the database design milestone. As shown in the physical diagram provided, it can be seen that not all column/attributes for each entity were used. The as-select statements were created to transfer the data from separate dataset schemas to load the unified schema.

### ***QuickSight Visualization***

This step was more of a polish on top of the queries that were developed for the designed unified database. What would be the use of data and queries if there was no meaningful representation of the analysis to share?

Fundamental to using the QuickSight software that AWS provides is creating views. These views are a convenient expression of the query scripts that were developed in the previous milestone. These views were named based on the question or answer that it was purposed to express. This step can be seen analogous to creating a python query interface that allows for connection to the database, clean structure, and operational inputs that allow for quick viewing of the developed queries. The view script generation was simple as it mostly just required a slight addition to each of the query scripts. To test for efficiency of the views and to consider whether or not the query is an effective enough of a design to use and answer, the timing module was used. The timing module was utilized by testing for the run time for each view to generate each view. To follow a convenient convention, the times for each view was documented in the SQL scripts as comments. Once the proper views were generated in the cluster, QuickSight was able to take the view and generate a colorful and meaningful representation of the data. There were a few tweaks that were made to the queries after the QuickSight visualizations were generated as some showed to be pretty weakly expressed as a visual. Another perk of using QuickSight was that it was possible to choose which types of axis would be used and what data it would

accommodate. It also offered a wide range of visuals to demonstrate a collective of queries.

Going through this step revealed more about the data and the database design and demonstrated that query development must be thorough in order to produce meaningful representations without going through trial and error inefficiency.

### ***Improvements for Initial Demo***

The first few fixes pertain to the visuals of a few queries. Because some of the queries resulted in a single line or a single attribute such as the most common name or most common word used. These queries generally just yield the name and the count of the amount of times this attribute appears in desc order and limiting it to one to consider the maximum. It was considerably difficult to make these more visually appealing with the limited amount of resources and visuals that QuickSight provides. One solution, albeit a simple one, would be to exclude the limit and allow for more comparisons to be displayed. This can allow for a deeper understanding by allowing for interpreters to view the industry holistically from what the datasets and combined unified dataset has to offer. When considering for a finding such as a max, it could be important to include the next few entities down the line to show the scale and relativeness of the data. Also, one of the queries that was initially developed had the wrong implementation in SQL. Instead of considering the most group that had the most releases, it ended up answering the question of what is the most common word used in releases? The query started off by joining the unified table of artists and the unified table of releases. It was first intended to display the number of times an artist appears in the table of releases since artists could have multiple releases. Rather than joining the tables based on the artist names, the join was centered on equating the artist name and the release's artist's credit id which linked to the artist. The effort was grouped by the name and based on the count of the appearance of each



‘name.’ As it turns out, the table that this join created was not one of the artists but one of words in either a track release or a general release. This was concluded by glancing further at the data that the script was joining and the results that the joins results in. Looking further down the table they are single words that were involved in the name of a release. This was only one of many of the fixes.

Although this was more of a mistake than an error, the visual for the languages was badly represented as one of the axis only displayed every other country. This was primarily in the 2D bar graph using the x-axis to display the countries. While this seemed like a trivial issue and one that could resolved with a few slight adjustments, the issue was worked around by switching the axis to accommodate for the independent value, the language. Taking from this error correction, I was able to successfully apply it to the other graphs that were dealing with similar types parameters issues such as an independent value being on the wrong axis, skipping out on a few of the important values.

The core to adjusting the visuals to make them not only more appealing but to make them into a generally more purposeful analysis lies within the queries that the database was designed around. Although it is definitely possible to change the entire database adjust the types and relations to accommodate more or new queries, the process is inefficient. The next best step is to consider the database design that was already implemented and worked from there to adjust queries to match a preferred portfolio such as a much more meaningful representation or analysis. As such, 3 of the queries that particularly dealt with a single aggregate or max value was altered to further accommodate for a wider range of entries within the table join. This allows for a much better comparison that also shows contrastable things such as magnitudes in difference and scale.

## ***Future Notes***

One of the many benefits of completion of the project is achieving a deeper understanding of how data is described and related. Even further, attempting to understand the data and successfully relating them together is an achievement that awards the developer with a better grasp of database design and theory.

All in all, the road to completion was a success, albeit a bumpy one. The datasets were studied and their purposes in their representation and answers provided by their data were understood and transformed into a unified dataset that allowed for queries to be run on either the single sets of data or the unified set. These accomplishments demonstrated a basic foundation in the developer's ability to define, create, load, transform, and analyze.

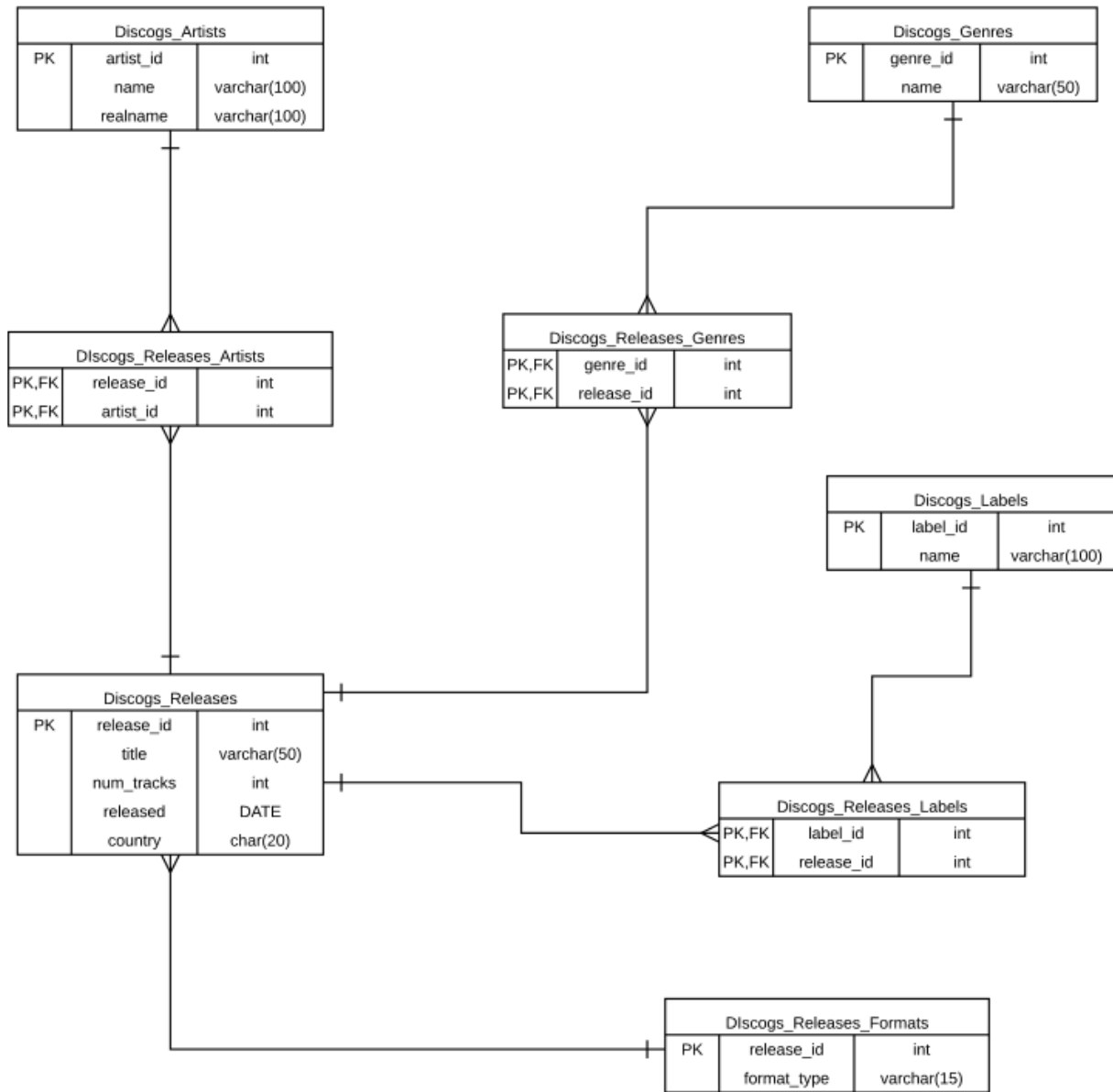
This project has given insight towards the realm of data science and has allowed for the developer to dip their feet into the vast pool of data analytics. Future projects and the related database implementation are now within grasp and the concepts and theory of data design and databases are now a mystery of the past. The skills harnessed through this course and project is a particularly useful addition to one's skillset.

In hindsight, this project didn't involve particularly complicated joins as the data itself was already fairly organized and neat when given to the developers. As a result, the queries did not end up answering incredibly difficult or resource-demanding types of queries. This project served as an example of what database design and implementation involved. For future projects, more meaningful and data driven queries will be the goal for accomplishing. The theory, design, and logic behind what it takes to build and manage a database has been demonstrated successfully.

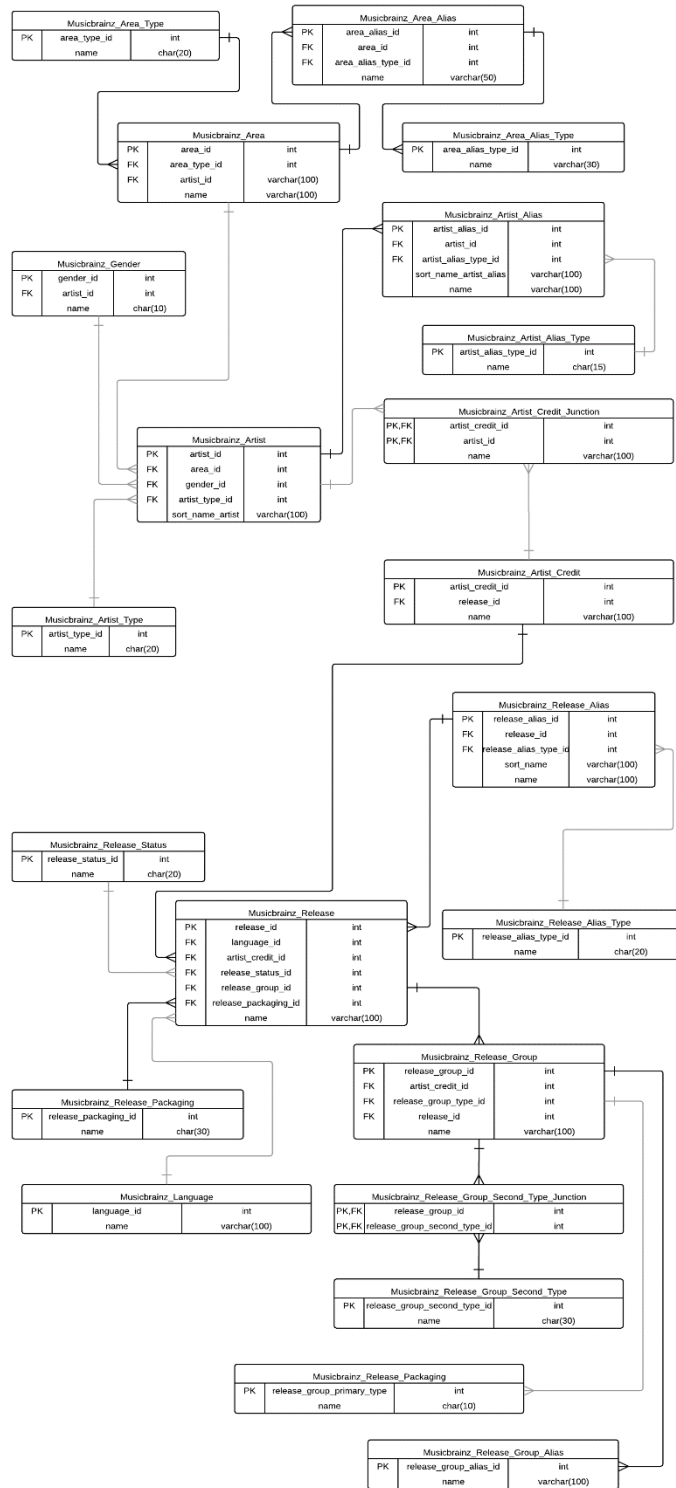
## Appendix

Conceptual Diagrams for each Dataset:

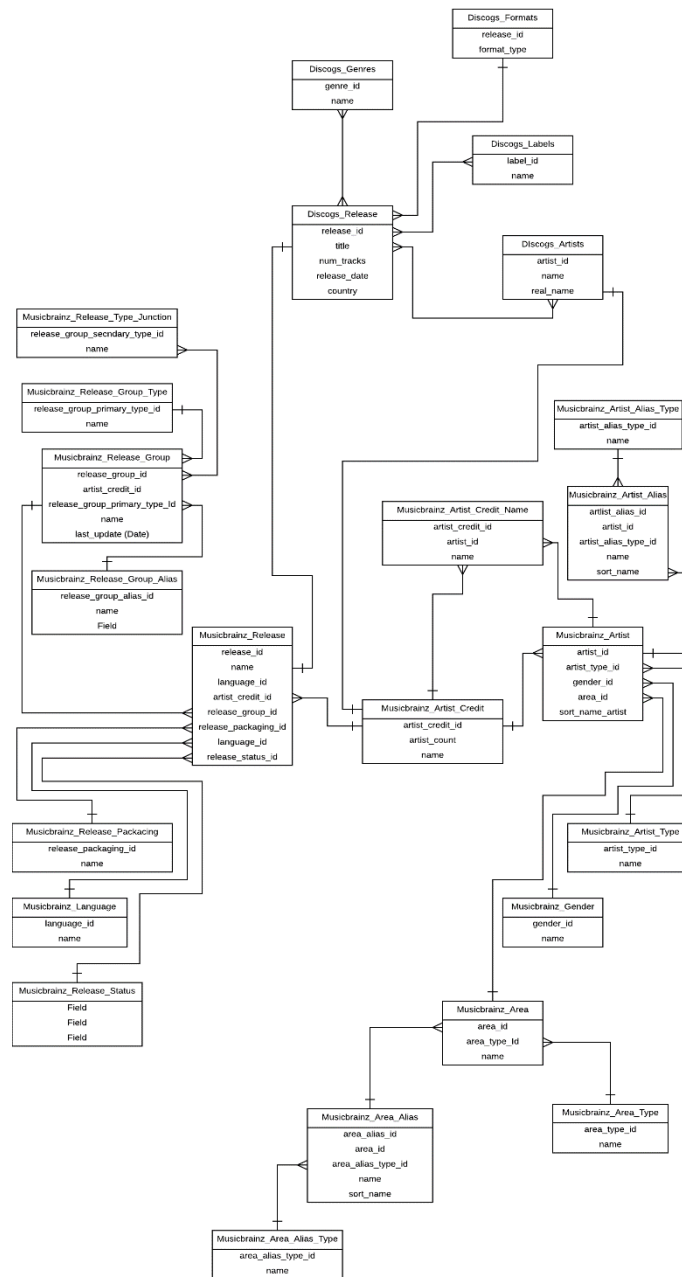
### Discogs



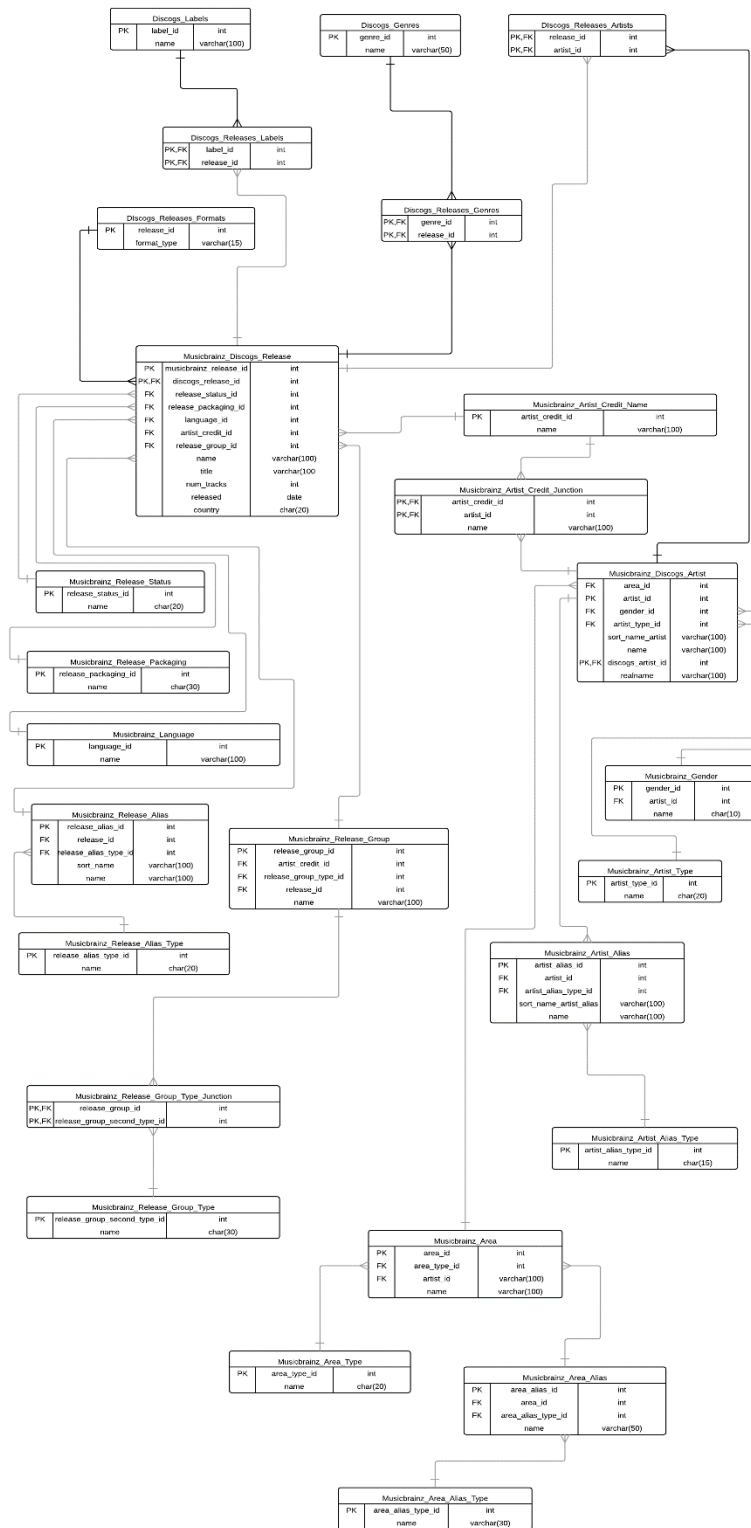
*Musicbrainz*



## Unified Conceptual Diagram:



*Unified Physical Diagram:*




## Queries:

Queries:

- 1) Which artist/group produced the most number of "Album" releases? (Aggregate, group by where filter) -entire dataset
- 2) Most common artist name in the music industry? (Aggregate, group by, order by) -entire dataset
- 3) Average group size of artists? (Average aggregate) -musicbrainz dataset
- 4) Most common medium of music in 2015. (count, inner join, group by, order by) -discogs
- 5) Artist that made the biggest contributions/involved in more albums? (aggregate, groupby, having) -entire dataset
- 6) What area has the most artists? (count, right outer, order by) - entire dataset
- 7) Most common type of artist. (count, right outer join) -musicbrainz dataset
- 8) What genre is most popular in the year 2013? (count, join, left outer join, having, group by, order by) -entire dataset
- 9) What is the biggest group of musicians? (max, join) -musicbrainz dataset
- 10) What language is the most common in the industry on music? (count, full join) - entire dataset

## Musicbrainz Site:



Log In Create Account

Search Artist Q

About Us Products Search Documentation English

### Welcome to MusicBrainz!

MusicBrainz is an open music encyclopedia that collects music metadata and makes it available to the public.

MusicBrainz aims to be:

1. **The ultimate source of music information** by allowing anyone to contribute and releasing the [data](#) under [open licenses](#).
2. **The universal lingua franca for music** by providing a reliable and unambiguous form of [music identification](#), enabling both people and machines to have meaningful conversations about music.

Like Wikipedia, MusicBrainz is maintained by a global community of users and we want everyone — including you — to [participate and contribute](#).

[More Information](#) — [FAQs](#) — [Contact Us](#)

MusicBrainz is operated by the [MetaBrainz Foundation](#), a California based 501(c)(3) tax-exempt non-profit corporation dedicated to keeping MusicBrainz free and open source.


### MusicBrainz Blog

**Latest posts:**

- [Server update, 2017-04-10](#)
- [Simplification of the featured artist guideline](#)
- [Picard 1.4.1 released](#)
- [Server update, 2017-03-27](#)
- [Server update, 2017-03-13](#)
- [May 2017 Schema Change](#)  
Release: May 15, 2017


[Read more »](#)

### Tag Your Music




- [MusicBrainz Picard](#)
- [Magic MP3 Tagger](#)
- [Yate Music Tagger](#)

### Quick Start




- [Beginners guide](#)
- [Editing introduction](#)
- [Style guidelines](#)
- [FAQs](#)

### Community




- [How to Contribute](#)
- [Bug Tracker](#)
- [Forums](#)

### MusicBrainz Database



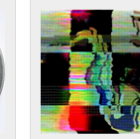

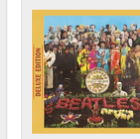


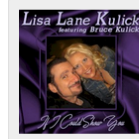

The majority of the data in the **MusicBrainz Database** is released into the **Public Domain** and can be downloaded and used **for free**.

### Developers



Use our **XML web service** or **development libraries** to create your own MusicBrainz-enabled applications.

### Recent Additions



Donate | Wiki | Forums | Bug Tracker | Blog | Twitter | Use beta site

Brought to you by [MetaBrainz Foundation](#) and our [sponsors](#) and [supporters](#). Cover Art provided by the [Cover Art Archive](#).

