Data Cartel

CIS 5500

EGOTdb

Team Members

Brendan Brett - bbt@seas.upenn.edu - @brendanbrett
Edoardo Palazzi - palazzi@seas.upenn.edu - @EdoardoPalazzi
Eitan Jacob - eitanj@seas.upenn.edu - @EitanJacob
GitHub Repo: https://github.com/brendanbrett/cis5500-project | **Demo Video**: Google Drive - Direct Link
REQUIREMENTS: 1) Put first .env in root folder. 2) Put second .env in client\app folder

1. Introduction & Project Goals

An *EGOT* refers to an individual who has won each of the four major American performing art awards: '**E**mmy, **G**rammy, **O**scar, **T**ony'. Each of these four awards represent outstanding achievement in television, recording, film, and Broadway theater[1]. Being nominated for just *one* of these awards is a major honor for an artist. Winning any one of these awards cements you into the annals of entertainment. Winning all four of these awards, however, elevates an artist into a legend. We wanted to honor the 19 individuals who have accomplished this feat.

The motivation for the application idea was to create an authoritative source database to track and identify those legendary artists. The internet contains many popular websites that act as a database used to track and identify exceptionally talented individuals across various fields and sectors: an example includes the *Pro Football Reference Hall of Fame Monitor*[2], a website that can be used to identify American Football players in the Hall of Fame, in addition to highlighting active players who are on track to be inducted in the future. With our application **EGOT**db, the primary goal is to similarly allow users to quickly identify designated EGOTs and to find out which artists might be on track to perform this feat next. Through our review of existing sources online, we identified a gap in the market: we did not find any websites that served as a source of award information for all four major awards, and while we found many static news articles from the past that list out the EGOT winners, we have not found any websites which aggregate this information in a dynamic and systematic way. Therefore, we identified an opportunity to be the first to do this.

2. Data Sources & Architecture

In order for a website to serve as an authoritative source, the underlying data must be perfect - or at least, as close as possible. Our target from the onset of the project was to accurately identify the EGOTs by querying the individual award databases; we had an internal mandate for ourselves to identify the 19 individuals without error. This goal was only achievable with accurate and reliable data, since the site's functionality and information was only as good as the data used for its foundation. Identifying data sources that enabled us to link the datasets together was extremely challenging, and we go deeper into this topic in the *Technical Challenges* section at the end of this document. In the end, we were able to create our own source datasets for 3 of the 4 major award datasets and link them together using the InternetMovieDB ID (IMDB ID). We are extremely proud of the quality of the data which we were able to produce. Using IMDB_ID also allowed us to utilize APIs such as themoviedb.org which uses IMDB_ID as an external key to find artists. *See* *Appendix A. Data Sources* for an exhaustive list of data sources we used for this project.

For the one dataset that we used from Kaggle (Grammy's data), each nomination record contained *n* artists, and so we had to disaggregate/unwind the artists and create one record per artist. For the datasets we scraped on our

[1] Wikipedia. "List of EGOT winners." *Wikipedia*, https://en.wikipedia.org/wiki/List_of_EGOT_winners.
[2] ProFootballReference. "Pro Football P Hall of Fame Monitors". https://www.pro-football-reference.com/hof/hofm_P.htm

second attempt from IMDB Events, we had control of the data we extracted and the quality and accuracy was extremely high.

We used Jupyter Notebook (via Google Colab) to perform extensive EDA on our datasets to confirm their accuracy. We used the Python pandas library to produce source files with one file per table in the database. We used the Python SQLAlchemy library to connect to the MySQL database hosted on AWS to insert the records from the files. This data ingestion pipeline worked very efficiently; it allowed us to iteratively and quickly update the Jupyter Notebook to produce a new file if we noticed any warnings or errors produced by the database upon insertion of the data.

We are using React for the front-end and Node.JS for the server-side development and we leveraged the code from Homework 2 to serve as a template for our web application. We utilized the open-source React component library Material UI to provide a polished theme and consistent styling across our web app. Our data is stored in one MySQL database hosted on Amazon AWS RDS. We have tables which represent the award organizations (which present the awards), the awards, the nominations for each award and the artists. We used GitHub as a version control system (VCS) to allow for easy collaboration amongst the team.

Additionally, we integrated our web application with TheMovieDB.org[3] using their free REST-based API to allow for dynamic streaming retrieval of images and biographical information for each artist.

Relational Schema & Normalization

Our database tables are in BCNF (and therefore, 3NF)

organization (<u>name</u>, year_founded) PRIMARY KEY (name)
 Functional Dependencies: name->year_founded
 Normalization: Normalized.

award (<u>name</u>,official_name ,awarded_for, presenting_organization, PRIMARY KEY (name), FOREIGN KEY (presenting_organization) REFERENCES organization(name))
 Functional Dependencies

1. name ->official_name,awarded_for, presenting_organization
2. official_name->name, awarded_for, presenting_organization - not a violation of BCNF/3NF is official-name is a superkey.

nominee (<u>id</u>, name, imdb_id, tony_id, birth_year, death_year PRIMARY KEY (id))
 Functional Dependencies: id->name, imdb_id, tony_id, birth_year, death_year

emmy_nomination (<u>year</u>, <u>category</u>, <u>title</u>, <u>nominee_id</u>, winner, award_type, <u>episode</u>, PRIMARY KEY (category, nominee_id, title, year, episode), FOREIGN KEY (award_type) REFERENCES award(name), FOREIGN KEY (nominee_id) REFERENCES nominee(id))
 Functional Dependencies: nominee_id, year, category, title, episode -> winner, company, producer, role

grammy_nomination (<u>year</u>, <u>category</u>, <u>title</u>, <u>nominee_id</u>, winner, PRIMARY KEY (category, nominee_id, title, year), FOREIGN KEY (nominee_id) REFERENCES nominee(id))
 Functional Dependencies: nominee_id, year, category, title -> winner, award-type

oscar_nomination (<u>year</u>, <u>category</u>, <u>title</u>, <u>nominee_id</u>, winner, <u>song_title</u>, PRIMARY KEY (category, nominee_id, title, year, song_title), FOREIGN KEY (nominee_id) REFERENCES nominee(id))
 Functional Dependencies: No Functional dependencies - this is confirmed in EDA Little Mermaid Example

[3]https://developer.themoviedb.org/

tony_nomination (year, category, title, nominee_id, winner, PRIMARY KEY (category, nominee_id, title, year), FOREIGN KEY (nominee_id) REFERENCES nominee(id))

Functional Dependencies: nominee_id, year, category, title, award_type -> winner

For optimization purposes, we subsequently added in an egot table to improve query performance:

egot_winner(id, imdb_id, age, years_to_egot, path,birth_year,death_year)

Functional Dependencies

1) Id -> imdb_id, age, years_to_egot, path,birth_year,death_year
2) imdb -> id, age, years_to_egot, path,birth_year,death_year- note that this does not violate 3NF or BCNF, since imdb is a candidate key.

3. Web App Description

Below we briefly describe the pages and functionality of our web application:

Home page:
This explains what an EGOT is to our end users.

EGOT page:
This page provides users with quick access to the most important functionality of the web app: displaying an image list of the legendary EGOT winners. This page shows all artists who have won each of the four awards, sorted chronologically by the year in which they were designated an EGOT. Each of the individual artist images is a link to the *Artist Details* page.

Artist Detail page:
This page provides details on the artist, including an image fetched from themoviedb.org on the fly, what the artist is known for, their ‘popularity’ according to themoviedb.org, their nominations for each award, and their wins.

Award List page:
This page will provide the entire history of awards for a particular major award for a given year. There are four award list pages for the four major awards: Emmy, Grammy, Oscar and Tonys. The user can change the year they are viewing using the slider at the top of the page. Users can tick ‘Winners only’ if they would like to filter the table to only those nominees who won an award.

Award Winner Analysis & Trivia page:
This page allows users to view the Analysis & Trivia answers based on the EGOT award winner data. These queries are the most challenging in terms of performance, and an example of 5 of the queries can be found in *Appendix C*. We’ve created queries that will produce answers for the following questions:

- Who accomplished the feat of winning an EGOT at the earliest age?
- Who accomplished the feat of winning an EGOT in the shortest duration?
- What award nominees have never won an award?
- Who has been nominated the most times without any wins?
- Which artist has the highest number of distinct nominations?
- Who is the closest to being the next EGOT (i.e., who has 3 awards?)
- Which artists have the highest average amount of losses before a win?
- What are the most common combinations for artists who have won two awards?
- When was the first time each EGOT won an award?
- Who has been nominated for all 4 awards but never won any of them?6. API Specification (Routes)

4. API Specification (Routes)

Below we've identified a list of the routes, along with a description of the functionality and parameters.

GET /author/:type

- **Functionality:** Retrieves the authors of the application
- **Request Parameters:**
    - **type:** type of response to be retrieved i.e., name

GET /egots

- **Functionality:** Retrieves artists who have won all four awards
- **Request Parameters:**
    - **type:** type of response to be retrieved i.e., name

Gets /egots/youngest_fastest

- **Functionality:** Retrieves EGOTs, sorted in ascending order by age and shortest time period to designation

GET /nominee/:id

- **Functionality:** a list of all award nominations for a nominee
- **Request Parameters:**
    - **id:** the unique identifier from the artists table

GET /nominations/:award/:year

- **Functionality:** Retrieves a list of all nominations for a year and aggregates individual nominees into one record. Response is returned in ascending order by category and by winners.
- **Request Parameters:**
    - award (str): either grammy, oscar, tony, or emmy
    - year (int): the year of specified award
- **Query Parameters:**
    - onlyWinners (boolean): True to see winners only, False to see all nominees.

GET /recent_nominees/year?onlyWinners=

- **Functionality:** Retrieves a list of recent nominees (nominees since a given year).
- **Request Parameters:**
    - year (int): the year of specified award
- **Query Parameters:**
    - onlyWinners (boolean): True to see winners only, False to see all nominees.

GET /losses_before_first_win

- **Functionality**: Retrieves a list of nominees with the highest number of nominations before their first win.

GET /nominee_most_categories?onlyWinners=

- **Functionality**: Retrieves a list of nominees with the most amount of distinct nominations
- **Query Parameters:**
    - onlyWinners (boolean): True to see winners only, False to see all nominees.

GET /nominee_longest_year_span?onlyWinners=

- **Functionality**: Retrieves a list of nominees with the longest nomination year span
- **Query Parameters:**
    - onlyWinners (boolean): True to see winners only, False to see all nominees

GET /losses_with_no_win

- **Functionality:** Retrieves a list of nominees that never won and orders them in descending order of total nominations

GET /missing_one_award

- **Functionality:** Retrieves a list of nominees that are only missing one award win to complete their EGOT status

GET /hardest_categories

- **Functionality:** Retrieves a list of award categories who have the highest average amount of losses before a win.

GET /common_combos

- **Functionality:** Retrieves a list of the most common combinations of two award winners

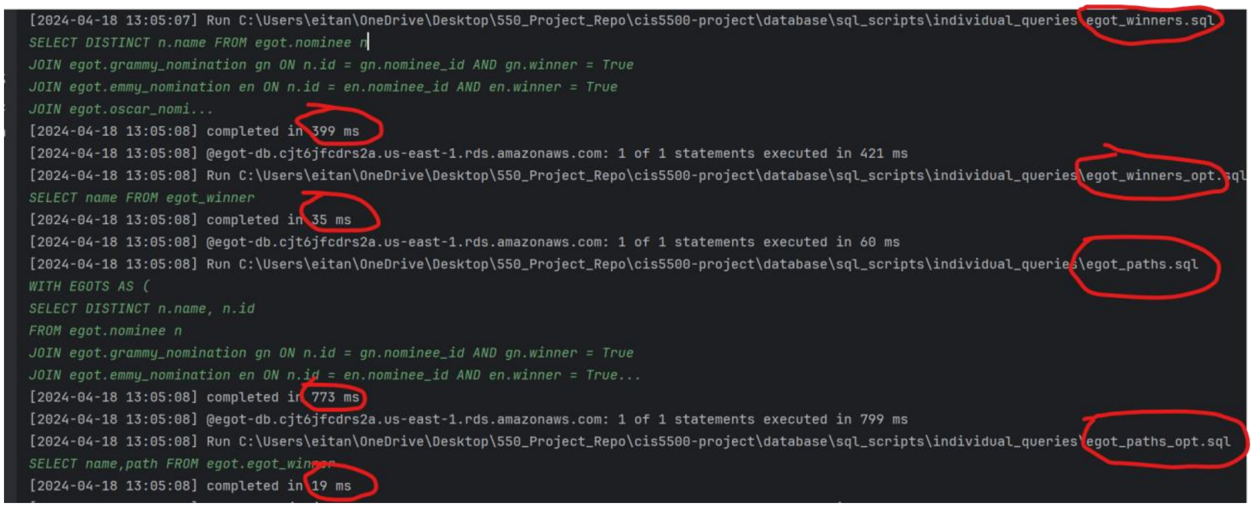
GET /egot_paths

- **Functionality:** Retrieves a list, in order, of the first time each EGOT winner won each award.

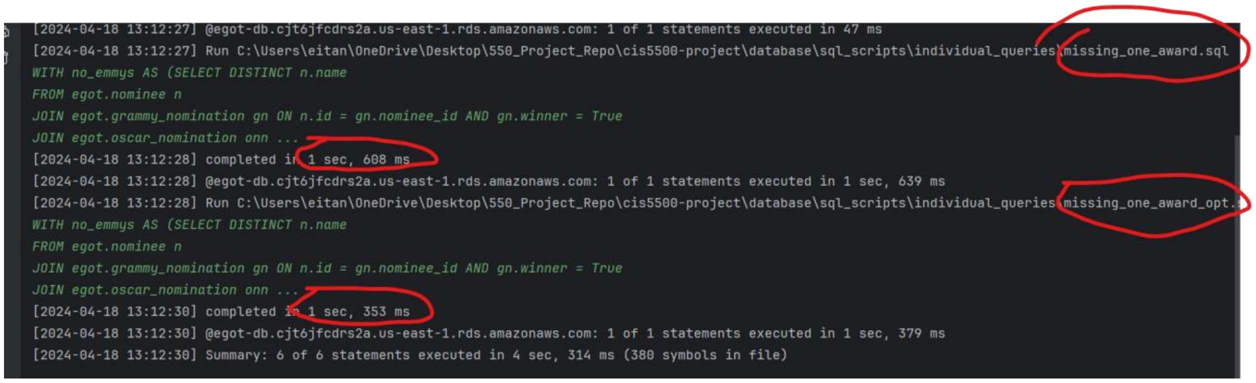
5.Performance evaluation

Method #1: Storing A New Table of Egot Winners in the Database.

Our first method consisted of storing EGOT information in a new table called egot_winners. By adding this table, several queries that involved EGOT winners were significantly optimized.



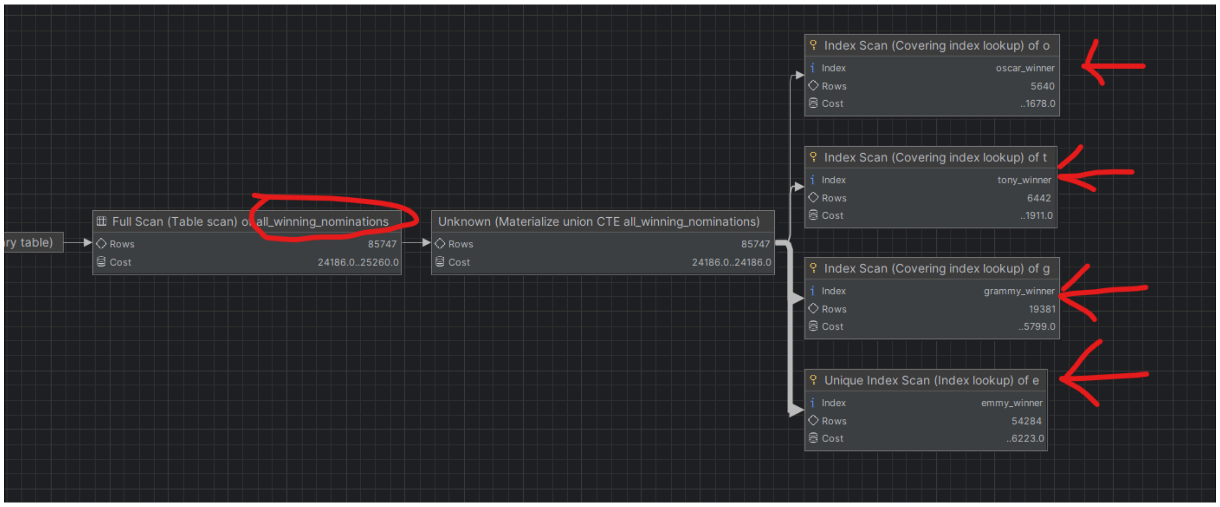
```
[2024-04-18 13:05:07] Run C:\Users\eitan\OneDrive\Desktop\550_Project_Repo\cis5500-project\database\sql_scripts\individual_queries\egot_winners.sql
SELECT DISTINCT n.name FROM egot.nominee n
JOIN egot.grammy_nomination gn ON n.id = gn.nominee_id AND gn.winner = True
JOIN egot.emmy_nomination en ON n.id = en.nominee_id AND en.winner = True
JOIN egot.oscar_nomi...
[2024-04-18 13:05:08] completed in 399 ms
[2024-04-18 13:05:08] @egot-db.cjt6jfcdrs2a.us-east-1.rds.amazonaws.com: 1 of 1 statements executed in 421 ms
[2024-04-18 13:05:08] Run C:\Users\eitan\OneDrive\Desktop\550_Project_Repo\cis5500-project\database\sql_scripts\individual_queries\egot_winners_opt.sql
SELECT name FROM egot_winner
[2024-04-18 13:05:08] completed in 35 ms
[2024-04-18 13:05:08] @egot-db.cjt6jfcdrs2a.us-east-1.rds.amazonaws.com: 1 of 1 statements executed in 60 ms
[2024-04-18 13:05:08] Run C:\Users\eitan\OneDrive\Desktop\550_Project_Repo\cis5500-project\database\sql_scripts\individual_queries\egot_paths.sql
WITH EGOTS AS (
SELECT DISTINCT n.name, n.id
FROM egot.nominee n
JOIN egot.grammy_nomination gn ON n.id = gn.nominee_id AND gn.winner = True
JOIN egot.emmy_nomination en ON n.id = en.nominee_id AND en.winner = True...
[2024-04-18 13:05:08] completed in 773 ms
[2024-04-18 13:05:08] @egot-db.cjt6jfcdrs2a.us-east-1.rds.amazonaws.com: 1 of 1 statements executed in 799 ms
[2024-04-18 13:05:08] Run C:\Users\eitan\OneDrive\Desktop\550_Project_Repo\cis5500-project\database\sql_scripts\individual_queries\egot_paths_opt.sql
SELECT name,path FROM egot.egot_winner
[2024-04-18 13:05:08] completed in 19 ms
```



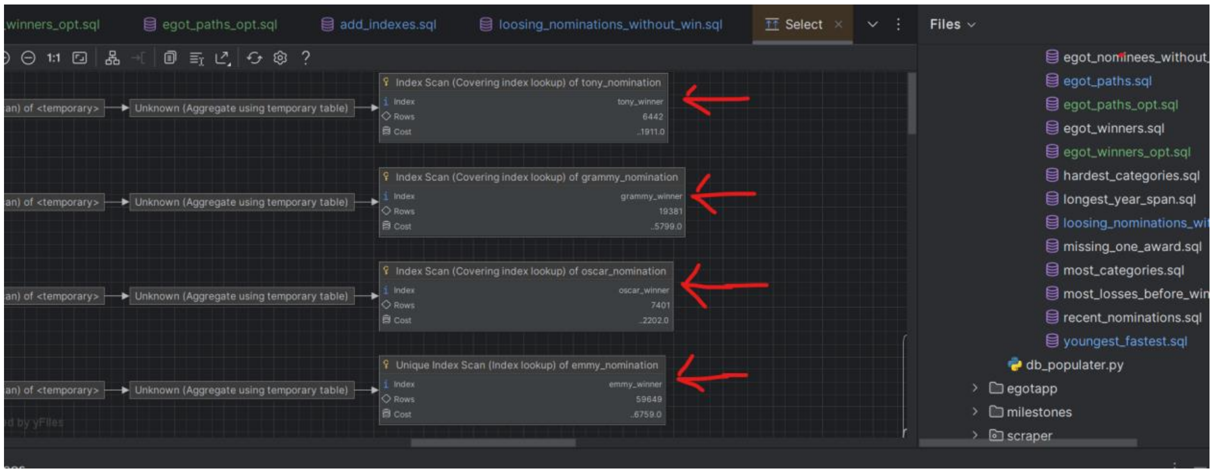
```
[2024-04-18 13:12:27] @egot-db.cjt6jfcdrs2a.us-east-1.rds.amazonaws.com: 1 of 1 statements executed in 47 ms
[2024-04-18 13:12:27] Run C:\Users\eitan\OneDrive\Desktop\550_Project_Repo\cis5500-project\database\sql_scripts\individual_queries\missing_one_award.sql
WITH no_emmys AS (SELECT DISTINCT n.name
FROM egot.nominee n
JOIN egot.grammy_nomination gn ON n.id = gn.nominee_id AND gn.winner = True
JOIN egot.oscar_nomination onn ...
[2024-04-18 13:12:28] completed in 1 sec, 608 ms
[2024-04-18 13:12:28] @egot-db.cjt6jfcdrs2a.us-east-1.rds.amazonaws.com: 1 of 1 statements executed in 1 sec, 639 ms
[2024-04-18 13:12:28] Run C:\Users\eitan\OneDrive\Desktop\550_Project_Repo\cis5500-project\database\sql_scripts\individual_queries\missing_one_award_opt.
WITH no_emmys AS (SELECT DISTINCT n.name
FROM egot.nominee n
JOIN egot.grammy_nomination gn ON n.id = gn.nominee_id AND gn.winner = True
JOIN egot.oscar_nomination onn ...
[2024-04-18 13:12:30] completed in 1 sec, 353 ms
[2024-04-18 13:12:30] @egot-db.cjt6jfcdrs2a.us-east-1.rds.amazonaws.com: 1 of 1 statements executed in 1 sec, 379 ms
[2024-04-18 13:12:30] Summary: 6 of 6 statements executed in 4 sec, 314 ms (380 symbols in file)
```

Method #2: Creating Indexes

In all nomination schemas, the attribute winner was not included in the PK due to normalization considerations. However, since our queries frequently make use of where conditions that depend on the winner, we added indexes on all of the nomination tables. It can be seen in the “Explain Plan” diagrams that queries are making use of this index instead of performing expensive full scans, particularly when using the CTE ‘all_winning_nominations.’



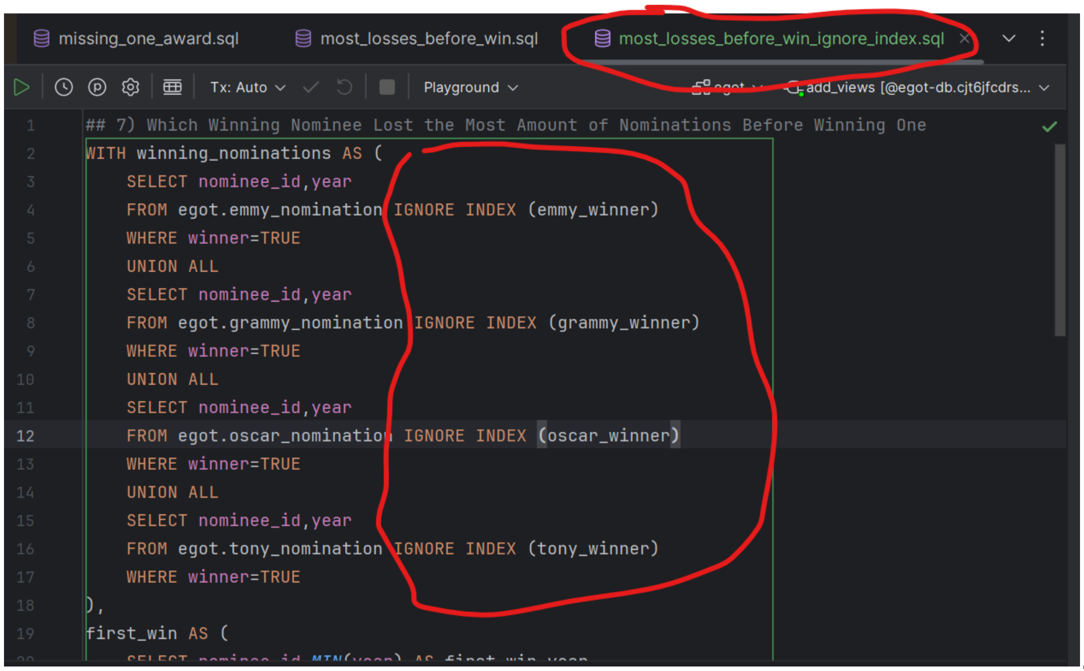
```
ry table) → Full Scan (Table scan) of all_winning_nominations
            Rows   85747
            Cost   24186.0..25260.0
→ Unknown (Materialize union CTE all_winning_nominations)
            Rows   85747
            Cost   24186.0..24186.0
→ Index Scan (Covering index lookup) of o
            Index  oscar_winner
            Rows   5640
            Cost   ..1678.0
→ Index Scan (Covering index lookup) of t
            Index  tony_winner
            Rows   6442
            Cost   ..1911.0
→ Index Scan (Covering index lookup) of g
            Index  grammy_winner
            Rows   19381
            Cost   ..5799.0
→ Unique Index Scan (Index lookup) of e
            Index  emmy_winner
            Rows   54284
            Cost   ..6223.0
```



```
an) of <temporary> → Unknown (Aggregate using temporary table) → Index Scan (Covering index lookup) of tony_nomination
            Index  tony_winner
            Rows   6442
            Cost   ..1911.0
an) of <temporary> → Unknown (Aggregate using temporary table) → Index Scan (Covering index lookup) of grammy_nomination
            Index  grammy_winner
            Rows   19381
            Cost   ..5799.0
an) of <temporary> → Unknown (Aggregate using temporary table) → Index Scan (Covering index lookup) of oscar_nomination
            Index  oscar_winner
            Rows   7401
            Cost   ..2202.0
an) of <temporary> → Unknown (Aggregate using temporary table) → Unique Index Scan (Index lookup) of emmy_nomination
            Index  emmy_winner
            Rows   59649
            Cost   ..6759.0
```

```
egot_nominees_without
egot_paths.sql
egot_paths_opt.sql
egot_winners.sql
egot_winners_opt.sql
hardest_categories.sql
longest_year_span.sql
loosing_nominations_wi
missing_one_award.sql
most_categories.sql
most_losses_before_win
recent_nominations.sql
youngest_fastest.sql
db_populater.py
egotapp
milestones
scraper
```

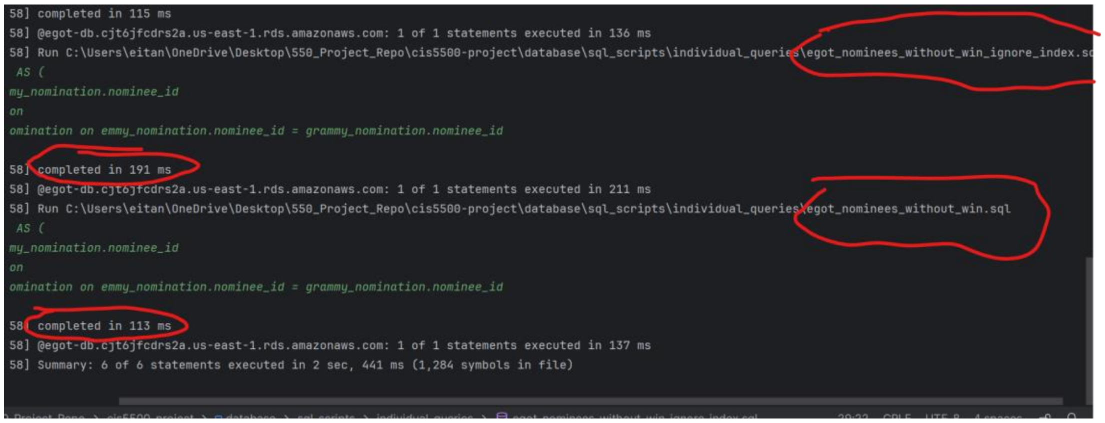
We used ‘IGNORE index’ to check the runtime before and after.



missing_one_award.sql
most_losses_before_win.sql
most_losses_before_win_ignore_index.sql
Tx: Auto
Playground
egot
add_views [@egot-db.cjt6jfcdrs...

```sql
## 7) Which Winning Nominee Lost the Most Amount of Nominations Before Winning One
WITH winning_nominations AS (
    SELECT nominee_id,year
    FROM egot.emmy_nomination IGNORE INDEX (emmy_winner)
    WHERE winner=TRUE
    UNION ALL
    SELECT nominee_id,year
    FROM egot.grammy_nomination IGNORE INDEX (grammy_winner)
    WHERE winner=TRUE
    UNION ALL
    SELECT nominee_id,year
    FROM egot.oscar_nomination IGNORE INDEX (oscar_winner)
    WHERE winner=TRUE
    UNION ALL
    SELECT nominee_id,year
    FROM egot.tony_nomination IGNORE INDEX (tony_winner)
    WHERE winner=TRUE
),
first_win AS (
    SELECT nominee_id,MIN(year) AS first_win_year
```

.



```
58] completed in 115 ms
58] @egot-db.cjt6jfcdrs2a.us-east-1.rds.amazonaws.com: 1 of 1 statements executed in 136 ms
58] Run C:\Users\eitan\OneDrive\Desktop\550_Project_Repo\cis5500-project\database\sql_scripts\individual_queries\egot_nominees_without_win_ignore_index.sc
 AS (
my_nomination.nominee_id
on
omination on emmy_nomination.nominee_id = grammy_nomination.nominee_id

58] completed in 191 ms
58] @egot-db.cjt6jfcdrs2a.us-east-1.rds.amazonaws.com: 1 of 1 statements executed in 211 ms
58] Run C:\Users\eitan\OneDrive\Desktop\550_Project_Repo\cis5500-project\database\sql_scripts\individual_queries\egot_nominees_without_win.sql
 AS (
my_nomination.nominee_id
on
omination on emmy_nomination.nominee_id = grammy_nomination.nominee_id

58] completed in 113 ms
58] @egot-db.cjt6jfcdrs2a.us-east-1.rds.amazonaws.com: 1 of 1 statements executed in 137 ms
58] Summary: 6 of 6 statements executed in 2 sec, 441 ms (1,284 symbols in file)
```

The run configurations for both of the above optimization methods are stored in .xml files in our project directory.

Query Performance Challenges

We came across two primary challenges in query performance.

1. Query Time Starting Point: All queries started off fairly quickly despite their complexity, making the optimization results more nuanced.
2. Materialized Views In MySQL: Since MySQL does not allow for storing views that persist, we needed to add in another table in the database which complicates the database's overall structure.

6. Technical challenges

Working as a group of 3 members (instead of 4, as planned) was challenging. Finding good data sources that allowed us to create this web app proved especially challenging. We initially searched Kaggle and the internet extensively, and found a few potential datasets that could be used. Unfortunately, after a preliminary exploratory data analysis (EDA), we quickly dismissed all but one of them as they were not current, accurate, or comprehensive. We also learned we needed an extra dataset as Emmy awards are broken out into two types: Primetime and Daytime. Despite these major challenges, we were determined to find a way to bring this app to life as we knew it had real value. We therefore embarked on the difficult task of crawling and scraping this data directly from the official sources: Grammy.com, oscars.org, and ibdb.com (*See Appendix A. Data Sources* for Links*)*.

The task of retrieving this data ourselves proved especially difficult and time consuming as we had to account for three completely different websites, which had three very different URL request path formats and very different HTML page structures for the data. Utilizing Python and the Beautiful Soup library, we created individual *scrapers* to crawl and retrieve the data from each of the sites and *processors* which we used to extract the relevant data from the HTML to create new datasets. We were confident that we now had the most accurate data possible, and our EDA proved this to be correct.

Unfortunately, we again encountered a large obstacle as we began working on Entity Resolution. Since each of the datasets we scraped were from different data sources, we had no common identifier which could be used to uniquely identify individuals. We attempted to use artist names, but of course, quickly learned it would not be sufficient since records which contained artists with common names would inaccurately be merged together. We had to start over.

Determined, we finally discovered three of the data sources existed on imdb.com *events,* and since each page provided a URL to the Artist page on imdb.com, we had found our master key for entity resolution, which was the IMDB_ID of the artist. Again utilizing Python, we created new web scrapers and new data processors. Fortunately, the site structure was similar for each page, which meant we could accomplish this task in a reasonable time frame. Our fourth dataset (Tony's data) was not available and we still had to identify a method to perform entity resolution on this dataset. Using the new data from IMDB allowed us to link to the IMDBs Artist dataset, which proved useful as we then used this information to link the Tony's dataset using artist name, birth year, and death year. All data sets were now linked together.

Appendix A - Data Sources

- Emmys Data:
    - First dataset:
        - https://www.kaggle.com/datasets/unanimad/emmy-awards
    - Final source #1 (Daytime Emmys) scraped and processed by the team. https://www.imdb.com/event/ev0000206/2023/1/
    - Final source #2 (Primetime Emmys) scraped and processed by the team:
        -
- Grammys Data:
    - First original dataset scraped and processed by the team:
        - https://www.grammy.com/awards
    - Final source scraped and processed by the team:
        - https://www.imdb.com/event/ev0000223/2023/1/
- Oscars Data:
    - First original dataset scraped and processed by the team:
        - https://awardsdatabase.oscars.org
    - Final source scraped and processed by the team. https://www.imdb.com/event/ev0000003/2024/1/
- Tony's Data:
    - Dataset scraped and processed by the team:
        - https://www.ibdb.com/awards/
- Artists Data:
    - Sourced via IMDb Non-Commercial Datasets:
        - https://datasets.imdbws.com/name.basics.tsv.gz
- Organization and Award Information:
    - Sourced manually via wikipedia.com
        - https://en.wikipedia.org/wiki/Emmy
        - https://en.wikipedia.org/wiki/Grammy_Awards
        - https://en.wikipedia.org/wiki/Academy_Awards
        - https://en.wikipedia.org/wiki/Tony_Awards

Data Sources

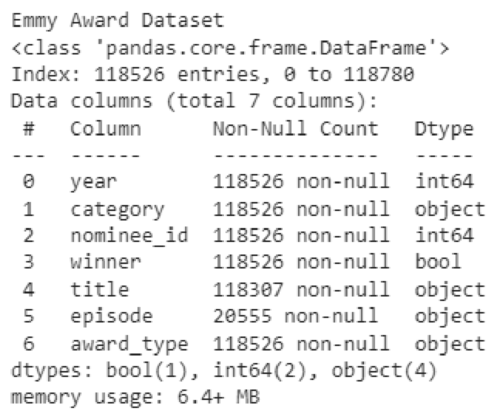
Dataset	Source	Description	Repository Location
Emmy Awards	https://www.imdb.com/event - scraped daytime and primetime awards by project team (code in repo)	Comprehensive history of the Emmy Awards (both primetime and daytime). Includes the year, award category, all nominees for the award, the winner of the award, and the title for which they won. We scraped Primetime and Daytime sources separately and merged them together, using an award_type attribute to distinguish the two.	database/cleaned_datasets/emmy_award_history.csv
Grammy Awards	https://www.imdb.com/event - scraped by project team (code in repo)	Comprehensive history of the Grammy Awards. Includes the year, award category, all nominees for the award, the winner of the award, and the title for which they won.	database/cleaned_datasets/grammy_award_history.csv
Oscar Awards	https://www.imdb.com/event - scraped by project team (code in repo)	Comprehensive history of the Oscar Awards. Includes the year, award category, all nominees for the award, the winner of the award, the title of the award, the song_title for music awards for which they won.	database/cleaned_datasets/oscar_award_history.csv
Tony Awards	ibdb.com/awards/ - scraped by project team (code in repo)	Comprehensive history of the Tony Awards. Includes the year, award category, all nominees for the award, the winner of the award.	database/cleaned_datasets/tony_award_winners.tsv
Award	Manually built by project team	Contains name, official_name, presenting_organization, and awarded_for. Sourced data from Wikipedia.	database/cleaned_datasets/award.csv
Nominee	Merge of nominees from Awards datasets + https://datasets.imdbws.com/name.basics.tsv.gz	Merged nominees from award datasets and merged in birth year and death year from name.basics.tsv.gz using imdb_id as key.	database/cleaned_datasets/nominee.csv
Organization	Manually built by project team	Contain name and year_founded. Sourced data from Wikipedia.	database/cleaned_datasets/organization.csv

Summary & Statistics

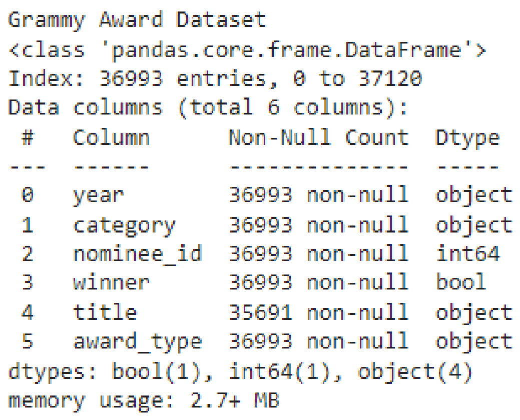
Detailed EDA and summary statistics can be found in our Colab notebook.

Emmy Award Dataset:	118,526 rows
Grammy Award Dataset:	36,993 rows
Oscar Award Dataset:	14,817 rows

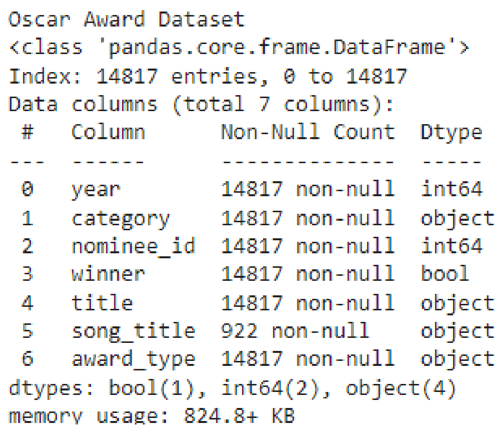
Tony Award Dataset:	12,338 rows
Award Dataset:	5 rows
Nominee Dataset:	57,375 rows
Organization Dataset:	6 rows



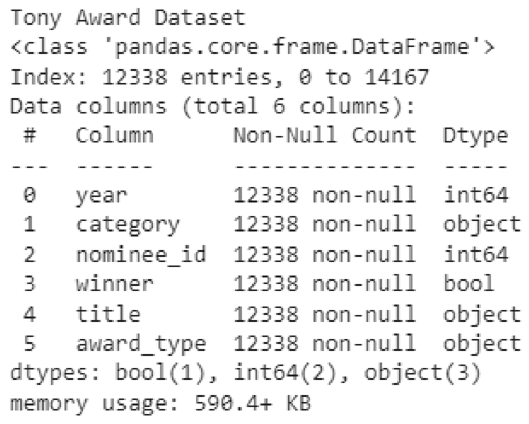
```
Emmy Award Dataset
<class 'pandas.core.frame.DataFrame'>
Index: 118526 entries, 0 to 118780
Data columns (total 7 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   year        118526 non-null  int64
 1   category    118526 non-null  object
 2   nominee_id  118526 non-null  int64
 3   winner      118526 non-null  bool
 4   title       118307 non-null  object
 5   episode     20555 non-null   object
 6   award_type  118526 non-null  object
dtypes: bool(1), int64(2), object(4)
memory usage: 6.4+ MB
```



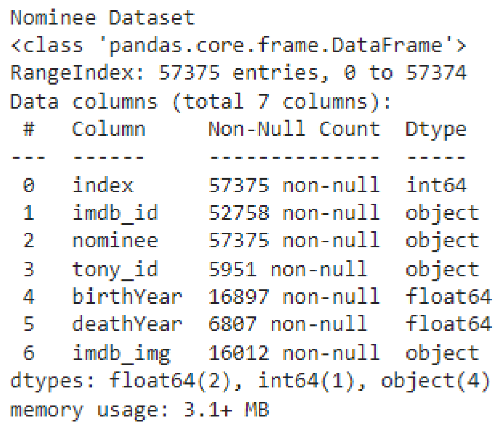
```
Grammy Award Dataset
<class 'pandas.core.frame.DataFrame'>
Index: 36993 entries, 0 to 37120
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   year        36993 non-null  object
 1   category    36993 non-null  object
 2   nominee_id  36993 non-null  int64
 3   winner      36993 non-null  bool
 4   title       35691 non-null  object
 5   award_type  36993 non-null  object
dtypes: bool(1), int64(1), object(4)
memory usage: 2.7+ MB
```



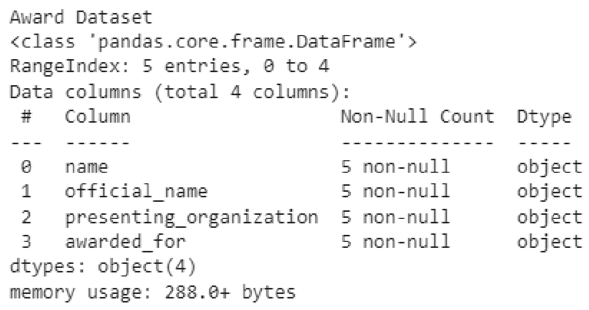
```
Oscar Award Dataset
<class 'pandas.core.frame.DataFrame'>
Index: 14817 entries, 0 to 14817
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   year        14817 non-null  int64
 1   category    14817 non-null  object
 2   nominee_id  14817 non-null  int64
 3   winner      14817 non-null  bool
 4   title       14817 non-null  object
 5   song_title  922 non-null    object
 6   award_type  14817 non-null  object
dtypes: bool(1), int64(2), object(4)
memory usage: 824.8+ KB
```



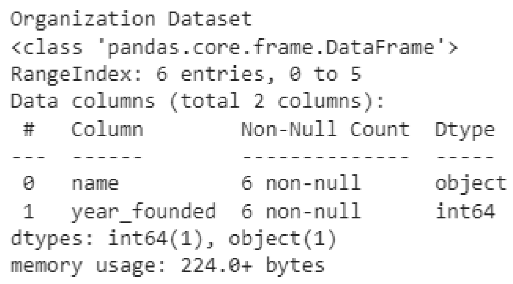
```
Tony Award Dataset
<class 'pandas.core.frame.DataFrame'>
Index: 12338 entries, 0 to 14167
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   year        12338 non-null  int64
 1   category    12338 non-null  object
 2   nominee_id  12338 non-null  int64
 3   winner      12338 non-null  bool
 4   title       12338 non-null  object
 5   award_type  12338 non-null  object
dtypes: bool(1), int64(2), object(3)
memory usage: 590.4+ KB
```



```
Nominee Dataset
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 57375 entries, 0 to 57374
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   index      57375 non-null  int64
 1   imdb_id    52758 non-null  object
 2   nominee    57375 non-null  object
 3   tony_id    5951 non-null   object
 4   birthYear  16897 non-null  float64
 5   deathYear  6807 non-null   float64
 6   imdb_img   16012 non-null  object
dtypes: float64(2), int64(1), object(4)
memory usage: 3.1+ MB
```

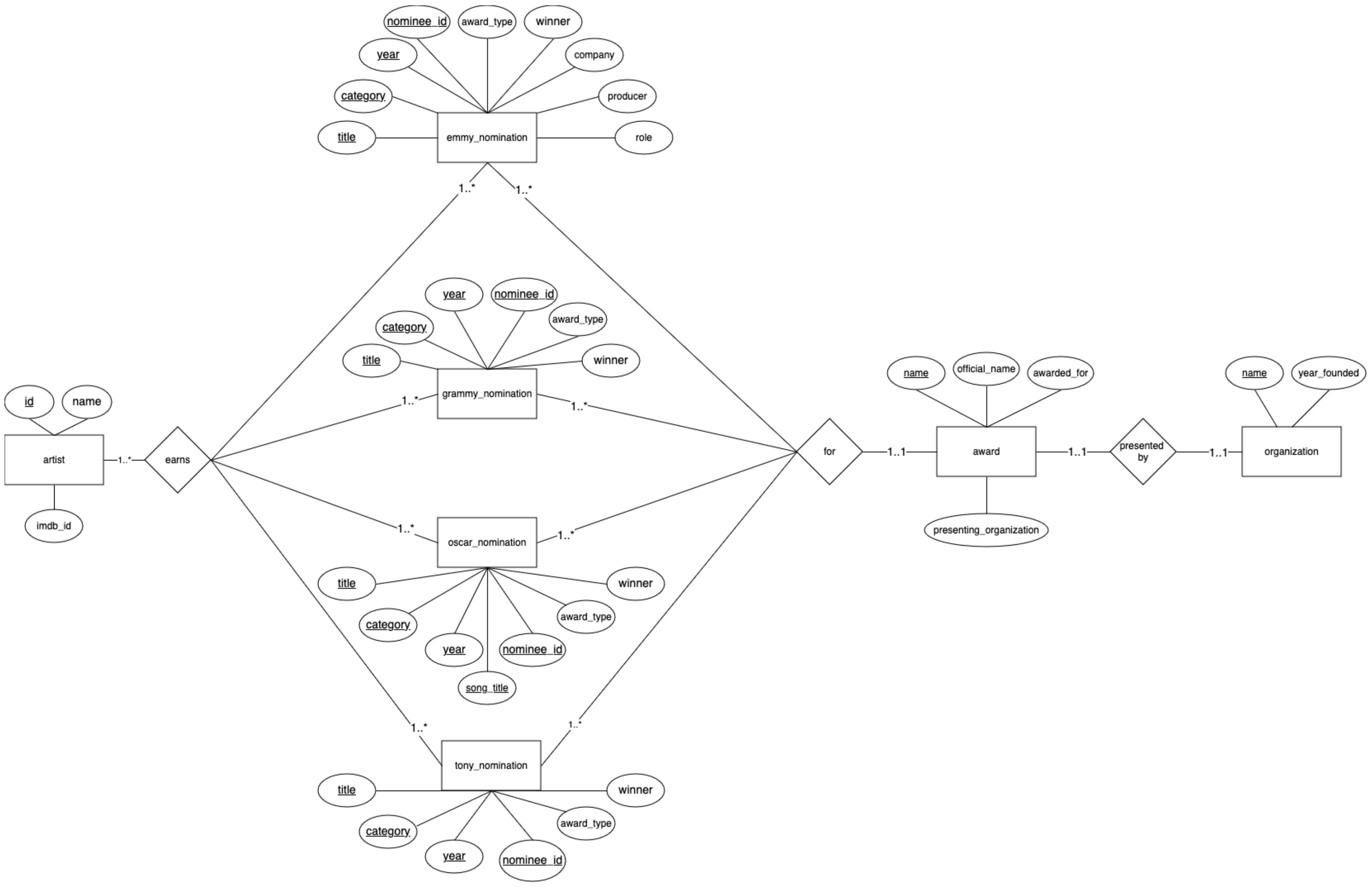


```
Award Dataset
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 4 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   name                     5 non-null      object
 1   official_name            5 non-null      object
 2   presenting_organization  5 non-null      object
 3   awarded_for              5 non-null      object
dtypes: object(4)
memory usage: 288.0+ bytes
```



```
Organization Dataset
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   name           6 non-null      object
 1   year_founded   6 non-null      int64
dtypes: int64(1), object(1)
memory usage: 224.0+ bytes
```

Appendix B - Entity Relationship Diagram



Appendix C - SQL Queries

Below we've identified some of our most complex queries used in our web app, identifying where the query is utilized along with its function.

1. Missing One Award

Used in: Trivia

Purpose: Retrieves a list of nominees that are only missing one award win to complete their EGOT status

```
WITH no_emmys AS (SELECT DISTINCT n.name, id
                 FROM egot.nominee n
                          JOIN egot.grammy_nomination gn ON n.id = gn.nominee_id AND gn.winner = True
                          JOIN egot.oscar_nomination onn ON n.id = onn.nominee_id AND onn.winner = True
                          JOIN egot.tony_nomination tn ON n.id = tn.nominee_id AND tn.winner = True),
    no_grammys AS (SELECT DISTINCT n.name, id
                   FROM egot.nominee n
                            JOIN egot.oscar_nomination onn ON n.id = onn.nominee_id AND onn.winner = True
                            JOIN egot.emmy_nomination en ON n.id = en.nominee_id AND en.winner = True
                            JOIN egot.tony_nomination tn ON n.id = tn.nominee_id AND tn.winner = True),
    no_oscars AS (SELECT DISTINCT n.name, id
                  FROM egot.nominee n
                           JOIN egot.grammy_nomination gn ON n.id = gn.nominee_id AND gn.winner = True
                           JOIN egot.emmy_nomination en ON n.id = en.nominee_id AND en.winner = True
                           JOIN egot.tony_nomination tn ON n.id = tn.nominee_id AND tn.winner = True),
    no_tonys AS (SELECT DISTINCT n.name, id
                 FROM egot.nominee n
                          JOIN egot.grammy_nomination gn ON n.id = gn.nominee_id AND gn.winner = True
                          JOIN egot.oscar_nomination onn ON n.id = onn.nominee_id AND onn.winner = True
                          JOIN egot.emmy_nomination en ON n.id = en.nominee_id AND en.winner = True),
    egot_winners AS (SELECT DISTINCT n.name, id
                     FROM egot.nominee n
                              JOIN egot.grammy_nomination gn ON n.id = gn.nominee_id AND gn.winner = True
                              JOIN egot.oscar_nomination onn ON n.id = onn.nominee_id AND onn.winner = True
                              JOIN egot.emmy_nomination en ON n.id = en.nominee_id AND en.winner = True
                              JOIN egot.tony_nomination tn ON n.id = tn.nominee_id AND tn.winner = True),
    all_missing_one AS (SELECT id, name, 'Emmy' AS missing_award
                        FROM no_emmys
                        WHERE id NOT IN (SELECT id from egot_winners)
                        UNION
                        SELECT id, name, 'Grammy' AS missing_award
                        FROM no_grammys
                        WHERE id NOT IN (SELECT id from egot_winners)
                        UNION
                        SELECT id, name, 'Oscar' AS missing_award
                        FROM no_oscars
                        WHERE id NOT IN (SELECT id from egot_winners)
                        UNION
                        SELECT id, name, 'Tony' AS missing_award
                        FROM no_tonys
                        WHERE id NOT IN (SELECT id from egot_winners)),

    all_winning_nominations AS (SELECT e.nominee_id, year, award_type
                                FROM egot.emmy_nomination e
                                WHERE winner = TRUE
                                UNION ALL
                                SELECT g.nominee_id, year, 'Grammy' AS award_type
                                FROM egot.grammy_nomination g
                                WHERE winner = TRUE
                                UNION ALL
                                SELECT o.nominee_id, year, 'Oscar' AS award_type
                                FROM egot.oscar_nomination o
                                WHERE winner = TRUE
                                UNION ALL
                                SELECT t.nominee_id, year, 'Tony' AS award_type
                                FROM egot.tony_nomination t
                                WHERE winner = TRUE),
    first_win AS (SELECT all_missing_one.name                     AS name,
                         CONCAT(award_type, ' (', MIN(year), ')') as first_win,
                         MIN(year)                                  as year,
                         missing_award
                  FROM all_missing_one
                           LEFT JOIN all_winning_nominations ON all_missing_one.id = all_winning_nominations.nominee_id
                  GROUP BY name, award_type
                  ORDER BY name, MIN(year))
SELECT name, missing_award, GROUP_CONCAT(first_win ORDER BY (year)) AS path
FROM first_win
GROUP BY name;
```

2. Losing Nominations

Used in: Trivia

Purpose: Retrieves a list of nominees with the highest number of nominations before their first win

```
WITH loosing_nominations AS (SELECT nominee_id, COUNT(*) as nominations
                            FROM egot.emmy_nomination
                            WHERE winner = FALSE
                            GROUP BY nominee_id
                            UNION ALL
                            SELECT nominee_id, COUNT(*) as nominations
                            FROM egot.grammy_nomination
                            WHERE winner = FALSE
                            GROUP BY nominee_id
                            UNION ALL
                            SELECT nominee_id, COUNT(*) as nominations
                            FROM egot.oscar_nomination
                            WHERE winner = FALSE
                            GROUP BY nominee_id
                            UNION ALL
                            SELECT nominee_id, COUNT(*) as nominations
                            FROM egot.tony_nomination
                            WHERE winner = FALSE
                            GROUP BY nominee_id),
    all_winning_nominations AS (SELECT e.nominee_id
                                FROM egot.emmy_nomination e
                                WHERE winner = TRUE
                                UNION ALL
                                SELECT g.nominee_id
                                FROM egot.grammy_nomination g
                                WHERE winner = TRUE
                                UNION ALL
                                SELECT o.nominee_id
                                FROM egot.oscar_nomination o
                                WHERE winner = TRUE
                                UNION ALL
                                SELECT t.nominee_id
                                FROM egot.tony_nomination t
                                WHERE winner = TRUE)
SELECT DISTINCT name, loosing_nominations.nominations AS amount
FROM loosing_nominations
        JOIN nominee ON loosing_nominations.nominee_id = nominee.id
WHERE nominee_id NOT IN (SELECT nominee_id FROM all_winning_nominations)
ORDER BY loosing_nominations.nominations DESC
LIMIT 5;
```

3. Longest Year Span

Used in: Trivia

Purpose: What is the longest span (in years) that an artist has been nominated?

```
WITH all_winning_nominations AS (
       SELECT e.nominee_id, year
        FROM egot.emmy_nomination e
        ${winnerCondition}
        UNION ALL
        SELECT g.nominee_id, year
        FROM egot.grammy_nomination g
        ${winnerCondition}
        UNION ALL
        SELECT o.nominee_id, year
        FROM egot.oscar_nomination o
        ${winnerCondition}
        UNION ALL
        SELECT t.nominee_id, year
        FROM egot.tony_nomination t
        ${winnerCondition}
    ),
    max_min_years AS (
        SELECT nominee_id, MIN(YEAR) as min ,MAX(YEAR) as max
        FROM all_winning_nominations
        GROUP BY nominee_id
    )
    SELECT name, max-min AS years
    FROM nominee
        JOIN max_min_years ON max_min_years.nominee_id= nominee.id
    WHERE nominee.birth_year IS NOT NULL  ##### this rules out companies
    ORDER BY years DESC
    LIMIT 5;
```

4. Most Losses Before Win

Used in: Trivia

Purpose: Which artists have the highest average amount of losses before a win?

```
WITH no_emmys AS (SELECT DISTINCT n.name, id
                  FROM egot.nominee n
                           JOIN egot.grammy_nomination gn ON n.id = gn.nominee_id AND gn.winner = True
                           JOIN egot.oscar_nomination onn ON n.id = onn.nominee_id AND onn.winner = True
                           JOIN egot.tony_nomination tn ON n.id = tn.nominee_id AND tn.winner = True),
    no_grammys AS (SELECT DISTINCT n.name, id
                   FROM egot.nominee n
                            JOIN egot.oscar_nomination onn ON n.id = onn.nominee_id AND onn.winner = True
                            JOIN egot.emmy_nomination en ON n.id = en.nominee_id AND en.winner = True
                            JOIN egot.tony_nomination tn ON n.id = tn.nominee_id AND tn.winner = True),
    no_oscars AS (SELECT DISTINCT n.name, id
                  FROM egot.nominee n
                           JOIN egot.grammy_nomination gn ON n.id = gn.nominee_id AND gn.winner = True
                           JOIN egot.emmy_nomination en ON n.id = en.nominee_id AND en.winner = True
                           JOIN egot.tony_nomination tn ON n.id = tn.nominee_id AND tn.winner = True),
    no_tonys AS (SELECT DISTINCT n.name, id
                 FROM egot.nominee n
                          JOIN egot.grammy_nomination gn ON n.id = gn.nominee_id AND gn.winner = True
                          JOIN egot.oscar_nomination onn ON n.id = onn.nominee_id AND onn.winner = True
                          JOIN egot.emmy_nomination en ON n.id = en.nominee_id AND en.winner = True),
    egot_winners AS (SELECT DISTINCT n.name, id
                     FROM egot.nominee n
                              JOIN egot.grammy_nomination gn ON n.id = gn.nominee_id AND gn.winner = True
                              JOIN egot.oscar_nomination onn ON n.id = onn.nominee_id AND onn.winner = True
                              JOIN egot.emmy_nomination en ON n.id = en.nominee_id AND en.winner = True
                              JOIN egot.tony_nomination tn ON n.id = tn.nominee_id AND tn.winner = True),
    all_missing_one AS (SELECT id, name, 'Emmy' AS missing_award
                        FROM no_emmys
                        WHERE id NOT IN (SELECT id from egot_winners)
                        UNION
                        SELECT id, name, 'Grammy' AS missing_award
                        FROM no_grammys
                        WHERE id NOT IN (SELECT id from egot_winners)
                        UNION
                        SELECT id, name, 'Oscar' AS missing_award
                        FROM no_oscars
                        WHERE id NOT IN (SELECT id from egot_winners)
                        UNION
                        SELECT id, name, 'Tony' AS missing_award
                        FROM no_tonys
                        WHERE id NOT IN (SELECT id from egot_winners)),

    all_winning_nominations AS (SELECT e.nominee_id, year, award_type
                                FROM egot.emmy_nomination e
                                WHERE winner = TRUE
                                UNION ALL
                                SELECT g.nominee_id, year, 'Grammy' AS award_type
                                FROM egot.grammy_nomination g
                                WHERE winner = TRUE
                                UNION ALL
                                SELECT o.nominee_id, year, 'Oscar' AS award_type
                                FROM egot.oscar_nomination o
                                WHERE winner = TRUE
                                UNION ALL
                                SELECT t.nominee_id, year, 'Tony' AS award_type
                                FROM egot.tony_nomination t
                                WHERE winner = TRUE),
    first_win AS (SELECT all_missing_one.name                     AS name,
                         CONCAT(award_type, ' (', MIN(year), ')') as first_win,
                         MIN(year)                                as year,
                         missing_award
                  FROM all_missing_one
                           LEFT JOIN all_winning_nominations ON all_missing_one.id = all_winning_nominations.nominee_id
                  GROUP BY name, award_type
                  ORDER BY name, MIN(year))
SELECT name, missing_award, GROUP_CONCAT(first_win ORDER BY (year)) AS path
FROM first_win
GROUP BY name;
```

5. Hardest Categories

Used in: Trivia

Purpose: Retrieves a list of award categories who have the highest average amount of losses before a win.

```
With all_categories AS (SELECT 'Grammy' AS award, category, nominee_id, MIN(DISTINCT year) as first_win
                        FROM grammy_nomination
                        WHERE winner = TRUE
                        GROUP BY category, nominee_id
                        UNION ALL
                        SELECT 'Oscar' AS award, category, nominee_id, MIN(DISTINCT year) as first_win
                        FROM oscar_nomination
                        WHERE winner = TRUE
                        GROUP BY category, nominee_id
                        UNION ALL
                        SELECT 'Tony' AS award, category, nominee_id, MIN(DISTINCT year) as first_win
                        FROM tony_nomination
                        WHERE winner = TRUE
                        GROUP BY category, nominee_id
                        UNION ALL
                        SELECT award_type, category, nominee_id, MIN(DISTINCT year) as first_win
                        FROM emmy_nomination
                        WHERE winner = TRUE
                        GROUP BY category, nominee_id),
    previous_losses AS (SELECT all_categories.award,
                               first_win,
                               all_categories.nominee_id,
                               all_categories.category,
                               COUNT(*) as previous_losses
                         FROM grammy_nomination
                                  JOIN all_categories ON grammy_nomination.nominee_id = all_categories.nominee_id
                             AND grammy_nomination.category = all_categories.category
                         WHERE winner = FALSE
                           AND year < first_win
                         GROUP BY all_categories.category, all_categories.nominee_id
                         UNION ALL
                         SELECT all_categories.award,
                               first_win,
                               all_categories.nominee_id,
                               all_categories.category,
                               COUNT(*) as previous_losses
                         FROM oscar_nomination
                                  JOIN all_categories ON oscar_nomination.nominee_id = all_categories.nominee_id
                             AND oscar_nomination.category = all_categories.category
```

```sql
                        WHERE winner = FALSE
                          AND year < first_win
                        GROUP BY all_categories.category, all_categories.nominee_id
                        UNION ALL
                        SELECT all_categories.award,
                               first_win,
                               all_categories.nominee_id,
                               all_categories.category,
                               COUNT(*) as previous_losses
                        FROM tony_nomination
                                 JOIN all_categories ON tony_nomination.nominee_id = all_categories.nominee_id
                            AND tony_nomination.category = all_categories.category
                        WHERE winner = FALSE
                          AND year < first_win
                        GROUP BY all_categories.category, all_categories.nominee_id
                        UNION ALL
                        SELECT all_categories.award,
                               first_win,
                               all_categories.nominee_id,
                               all_categories.category,
                               COUNT(*) as previous_losses
                        FROM emmy_nomination
                                 JOIN all_categories ON emmy_nomination.nominee_id = all_categories.nominee_id
                            AND emmy_nomination.category = all_categories.category
                        WHERE winner = FALSE
                          AND year < first_win
                        GROUP BY all_categories.category, all_categories.nominee_id)
SELECT previous_losses.award, previous_losses.category, ROUND(AVG(previous_losses), 2) AS avg_previous_losses
FROM previous_losses
GROUP BY category
ORDER BY avg_previous_losses DESC
LIMIT 5;
```