Netflix Data Analysis Project Using PostgreSQL

Motivation

This project explores the comprehensive analysis of Netflix content by building a normalized PostgreSQL database and performing complex SQL queries. It demonstrates skills in data cleaning, relational modeling, SQL querying, and extracting business insights.

Objectives

- Design a relational schema from raw CSV data
- Use SQL to solve analytical and business queries
- Visualize database schema for better clarity
- Discover insights from the Netflix content library

Tools Used

- **Python (Pandas)** – Data cleaning and preprocessing
- **PostgreSQL** – Database design and querying
- **pgAdmin 4** – PostgreSQL management tool
- **dbdiagram.io** – ER diagram creation
- **Git & GitHub** – Version control and collaboration

ER Diagram For My Database

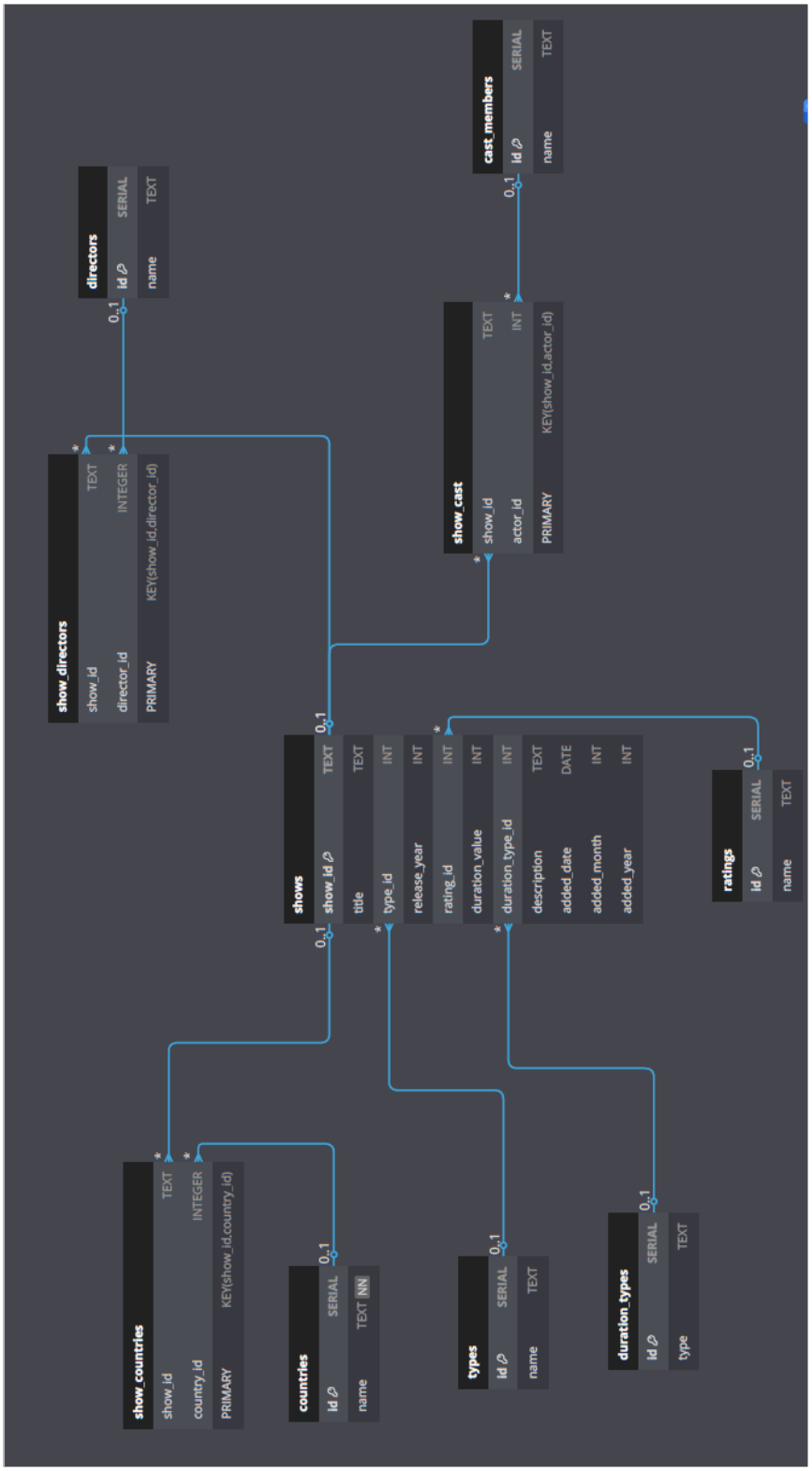

Figure 1: Figure: Entity-Relationship Mode For My Database

Query Analysis & Solutions

1. How many shows are there in total?

Problem Statement: Identify the total number of Netflix shows for basic dataset understanding and to establish a foundational content volume metric.

Approach:

- Use the `COUNT(*)` aggregate function on the `shows` table.
- Provide an alias `no_of_shows` for clarity in output.

SQL Solution:

```sql
SELECT COUNT(*) AS NO_OF_SHOWS
FROM SHOWS;
```

Listing 1: Total Number of Shows

---

2. List all movies released in the year 2020

Problem Statement: Identify all Netflix movies (not TV shows) that were released in the year 2020 for content acquisition analysis.

Approach:

- Perform an INNER JOIN between the `shows` and `types` tables.
- Filter results to only include rows where `type = 'movie'`.
- Apply a condition to restrict `release_year = 2020`.

SQL Solution:

```sql
SELECT S.TITLE
FROM SHOWS S
INNER JOIN TYPES T ON S.TYPE_ID = T.ID
WHERE T.NAME = 'MOVIE'
  AND S.RELEASE_YEAR = 2020;
```

Listing 2: Movies Released in 2020

---

3. Find the total number of shows for each content type

Problem Statement: Determine how many shows exist in each content category such as Movies and TV Shows to analyze the distribution of content types.

Approach:

- Perform an INNER JOIN between the `shows` and `types` tables.
- Group the results by type and use COUNT to determine total shows per type.
- Also consider alternative version without JOIN for comparison.

SQL Solution:

```sql
SELECT T.NAME AS TYPE_OF_SHOW, COUNT(*) AS NO_OF_SHOWS
FROM SHOWS S
INNER JOIN TYPES T ON S.TYPE_ID = T.ID
GROUP BY T.NAME;

-- ALTERNATE VERSION WITHOUT JOIN
SELECT TYPE_ID AS SHOW_TYPE, COUNT(*) AS NO_OF_SHOWS
FROM SHOWS
GROUP BY TYPE_ID;
```

Listing 3: Total Shows per Content Type

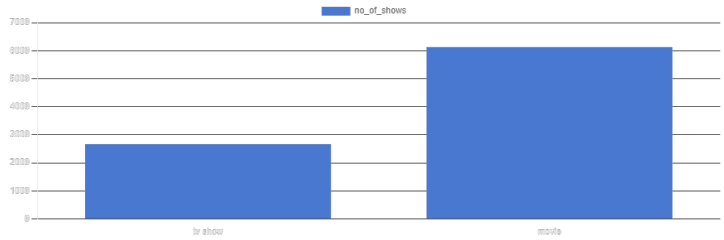

Figure 2: Above Bar Chart shows count of shows for each content type.

---

4. List distinct ratings available in the dataset

Problem Statement: Retrieve all unique content ratings to understand the classification of shows available on Netflix.

Approach:

- Query the `ratings` table directly.
- Use `DISTINCT` to return only unique values.

SQL Solution:

```sql
SELECT DISTINCT NAME
FROM RATINGS;
```

Listing 4: Distinct Ratings in Dataset

---

5. Number of shows per country

Problem Statement: Determine content production by country to assess international contributions to Netflix's library.

Approach:

- Join the `shows`, `show_countries`, and `countries` tables.
- Group by `country name` and count associated shows.
- Order by show count in descending order.

SQL Solution:

```sql
SELECT C.NAME AS COUNTRY_NAME, COUNT(*) AS NO_OF_SHOWS
FROM SHOWS S
INNER JOIN SHOW_COUNTRIES SC ON S.SHOW_ID = SC.SHOW_ID
INNER JOIN COUNTRIES C ON SC.COUNTRY_ID = C.ID
GROUP BY C.NAME
ORDER BY NO_OF_SHOWS DESC;
```

Listing 5: Shows by Country

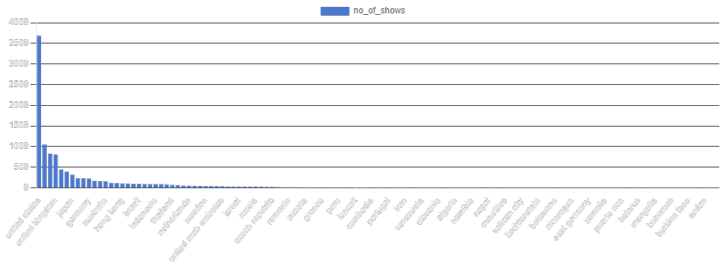

Figure 3: Above Bar Chart Represents No of Shows per Country

6. Directors with the most shows

Problem Statement: Identify directors who have contributed the most content to Netflix.

Approach:

- Join `directors` and `show_directors` tables.
- Group by `director name`.
- Count shows and sort in descending order, excluding unknowns.

SQL Solution:

```sql
SELECT NAME AS DIRECTOR_NAME, COUNT(*) AS NO_OF_SHOWS
FROM DIRECTORS
INNER JOIN SHOW_DIRECTORS ON ID = DIRECTOR_ID
WHERE NAME NOT ILIKE 'UNKNOWN'
GROUP BY NAME
ORDER BY NO_OF_SHOWS DESC;
```

Listing 6: Top Directors

---

7. Top 5 most-featured actors

Problem Statement: List the top five actors based on the number of Netflix shows they appear in.

Approach:

- Join `cast_members` and `show_cast` tables.
- Count appearances for each actor and group by name.
- Sort by count and limit results to top 5.

SQL Solution:

```sql
SELECT NAME AS ACTOR_NAME, COUNT(*) AS NO_OF_SHOWS
FROM CAST_MEMBERS
INNER JOIN SHOW_CAST ON ID = ACTOR_ID
WHERE NAME NOT ILIKE 'UNKNOWN'
GROUP BY NAME
ORDER BY NO_OF_SHOWS DESC
LIMIT 5;
```

Listing 7: Top 5 Actors by Appearances

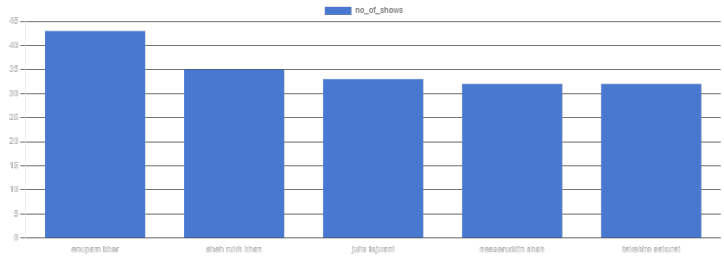

Figure 4: Top 5 popular Actors in Netflix

---

8. Countries producing most TV Shows

Problem Statement: Determine which countries have produced the most TV shows on Netflix.

Approach:

- Join `countries`, `show_countries`, `shows`, and `types` tables.
- Filter for type 'TV Show' and valid country names.
- Group by country and sort by show count.

SQL Solution:

```sql
SELECT C.NAME AS COUNTRY_NAME, COUNT(*) AS NO_OF_TV_SHOWS
FROM COUNTRIES C
INNER JOIN SHOW_COUNTRIES SC ON C.ID = SC.COUNTRY_ID
INNER JOIN SHOWS S ON S.SHOW_ID = SC.SHOW_ID
INNER JOIN TYPES T ON S.TYPE_ID = T.ID
WHERE C.NAME NOT ILIKE 'UNKNOWN' AND T.NAME ILIKE 'TV SHOW'
GROUP BY C.NAME
ORDER BY NO_OF_TV_SHOWS DESC;
```

Listing 8: Top Countries by TV Shows

---

9. Average duration of movies by rating

Problem Statement: Calculate the average duration of movies grouped by their ratings.

Approach:

- Join the `shows`, `types`, `ratings`, and `duration_types` tables.

- Filter to include only movie-type shows with duration in minutes.
- Group by rating name and compute the average duration.

SQL Solution:

```sql
SELECT R.NAME AS TYPE_OF_RATING,
       (CONCAT(CAST(ROUND(AVG(S.DURATION_VALUE), 3) AS TEXT), ' min'))
   AS AVG_DURATION_OF_MOVIES_IN_THIS_RATING
FROM SHOWS S
INNER JOIN TYPES T ON S.TYPE_ID = T.ID
INNER JOIN RATINGS R ON S.RATING_ID = R.ID
INNER JOIN DURATION_TYPES DT ON S.DURATION_TYPE_ID = DT.ID
WHERE T.NAME ILIKE 'MOVIE' AND DT.TYPE ILIKE 'MIN'
GROUP BY R.NAME;
```

Listing 9: Average Duration by Rating

---

10. Number of shows added each year

Problem Statement: Track Netflix's content growth by counting the number of shows added per year.

Approach:

- Group the `shows` table by `added_year`.
- Use `COUNT(*)` to find the total number of shows added each year.

SQL Solution:

```sql
SELECT ADDED_YEAR AS YEAR_ADDED, COUNT(*) AS
   NO_OF_SHOWS_ADDED_IN_THAT_YEAR
FROM SHOWS
GROUP BY ADDED_YEAR;
```

Listing 10: Yearly Added Shows

---

11. Actors in both Movies and TV Shows

Problem Statement: Find actors who have worked in both movies and TV shows.

Approach:

- Join `cast_members`, `show_cast`, `shows`, and `types` tables.

- Group by actor name and filter those who have multiple content types using `HAVING COUNT(DISTINCT type) > 1`.

SQL Solution:

```sql
SELECT CM.NAME AS ACTOR_NAME
FROM CAST_MEMBERS CM
JOIN SHOW_CAST SC ON CM.ID = SC.ACTOR_ID
JOIN SHOWS S ON S.SHOW_ID = SC.SHOW_ID
JOIN TYPES T ON S.TYPE_ID = T.ID
WHERE CM.NAME NOT ILIKE 'UNKNOWN'
GROUP BY CM.NAME
HAVING COUNT(DISTINCT T.NAME) > 1;
```

Listing 11: Actors in Movies and TV Shows

---

12. Shows with more than 3 actors

Problem Statement: Identify shows that feature more than three distinct actors.

Approach:

- Join `shows` and `show_cast` tables.
- Group by `show_id` and use `HAVING COUNT(DISTINCT actor_id) >= 3`.

SQL Solution:

```sql
SELECT S.TITLE AS SHOW, COUNT(DISTINCT SC.ACTOR_ID) AS NO_OF_ACTORS
FROM SHOWS AS S
JOIN SHOW_CAST AS SC ON S.SHOW_ID = SC.SHOW_ID
GROUP BY S.SHOW_ID
HAVING COUNT(DISTINCT SC.ACTOR_ID) >= 3
ORDER BY NO_OF_ACTORS DESC;
```

Listing 12: Shows with More Than 3 Actors

---

13. Shows by prolific directors (more than 5 shows)

Problem Statement: List all shows directed by directors who have directed more than five Netflix shows.

Approach:

- Join `shows`, `show_directors`, and `directors` tables.

- Filter directors with more than 5 shows using a subquery in `WHERE` clause.

SQL Solution:

```sql
SELECT S.TITLE AS SHOW_NAME, D.NAME AS DIRECTOR_NAME
FROM SHOWS AS S
JOIN SHOW_DIRECTORS AS SD ON S.SHOW_ID = SD.SHOW_ID
JOIN DIRECTORS AS D ON SD.DIRECTOR_ID = D.ID
WHERE D.NAME NOT ILIKE 'UNKNOWN'
  AND D.ID IN (
    SELECT DIRECTOR_ID
    FROM SHOW_DIRECTORS
    GROUP BY DIRECTOR_ID
    HAVING COUNT(DISTINCT SHOW_ID) > 5
);
```

Listing 13: Shows by Prolific Directors

---

14. Most common release year for shows

Problem Statement: Determine the release year in which the highest number of shows were launched on Netflix. This insight helps in identifying peak years of content publication.

Approach:

- Group the `shows` table by the `release_year` column.
- Count the number of shows for each year using `COUNT(*)`.
- Use `ORDER BY` in descending order and limit the result to 1 to get the most frequent year.

SQL Solution:

```sql
SELECT RELEASE_YEAR, COUNT(*) AS NO_OF_SHOWS_RELEASED
FROM SHOWS
GROUP BY RELEASE_YEAR
ORDER BY COUNT(*) DESC
LIMIT 1;
```

Listing 14: Most Common Release Year

---

15. Shows with actors whose names start with 'A'

Problem Statement: Retrieve all shows that feature actors whose names start with the letter 'A'. Useful for studying cast patterns and naming distributions.

Approach:

- Join `shows`, `show_cast`, and `cast_members` tables.
- Filter actors using the `ILIKE 'A%'` condition to match names starting with 'A'.
- Select show titles and corresponding actor names.

SQL Solution:

```sql
SELECT S.TITLE, CM.NAME AS ACTOR_NAME
FROM SHOWS AS S
JOIN SHOW_CAST AS SC ON S.SHOW_ID = SC.SHOW_ID
JOIN CAST_MEMBERS AS CM ON SC.ACTOR_ID = CM.ID
WHERE CM.NAME ILIKE 'A%';
```

Listing 15: Actors Starting with A

---

16. Rank shows by release year within each country

Problem Statement: Generate a ranking of shows within each country based on their release years. Helps in understanding content release chronology.

Approach:

- Join `shows`, `show_countries`, and `countries` tables.
- Use the `DENSE_RANK()` window function to rank shows within each country partitioned by `country name` and ordered by `release year`.
- Exclude entries with unknown country names.

SQL Solution:

```sql
SELECT C.NAME, S.TITLE, S.RELEASE_YEAR,
       DENSE_RANK() OVER (PARTITION BY C.NAME ORDER BY RELEASE_YEAR) AS
    RANK
FROM SHOWS AS S
INNER JOIN SHOW_COUNTRIES AS SC ON S.SHOW_ID = SC.SHOW_ID
INNER JOIN COUNTRIES AS C ON SC.COUNTRY_ID = C.ID
WHERE C.NAME NOT ILIKE 'UNKNOWN'
ORDER BY C.NAME, S.RELEASE_YEAR;
```

Listing 16: Ranked Shows by Country and Year

17. Shows directed and acted by the same person

Problem Statement: Identify shows where the same individual served both as a director and actor. Useful for spotlighting multi-talented contributors.

Approach:

- Create a CTE to join `shows`, `show_directors`, `directors`, `show_cast`, and `cast_members` tables.
- Compare `director name` and `actor name` within the same row.
- Select only those rows where both names match.

SQL Solution:

```sql
WITH directors_joined AS (
    SELECT S.SHOW_ID, S.TITLE, D.NAME AS DIRECTOR_NAME, CM.NAME AS
   ACTOR_NAME
    FROM SHOWS AS S
    INNER JOIN SHOW_DIRECTORS AS SD ON S.SHOW_ID = SD.SHOW_ID
    INNER JOIN DIRECTORS AS D ON SD.DIRECTOR_ID = D.ID
    INNER JOIN SHOW_CAST AS SC ON S.SHOW_ID = SC.SHOW_ID
    INNER JOIN CAST_MEMBERS AS CM ON SC.ACTOR_ID = CM.ID
    WHERE CM.NAME NOT ILIKE 'UNKNOWN' AND D.NAME NOT ILIKE 'UNKNOWN'
)
SELECT SHOW_ID, TITLE, DIRECTOR_NAME AS DIRECTOR_AND_ACTOR
FROM directors_joined
WHERE DIRECTOR_NAME = ACTOR_NAME;
```

Listing 17: Same Person as Actor and Director

---

18. Shows added in the same month across different years

Problem Statement: List all shows that were added in the same calendar month across multiple years. This can help identify trends in release scheduling.

Approach:

- Extract the month and year from the `added_date` column using `TO_CHAR`.
- Create a CTE to structure the data with month and year split.
- Display all records sorted by month and year.

SQL Solution:

```sql
WITH dates AS (
    SELECT SHOW_ID, TITLE,
           TO_CHAR(ADDED_DATE, 'MM') AS MONTH,
           TO_CHAR(ADDED_DATE, 'YYYY') AS YEAR
    FROM SHOWS
)
SELECT *
FROM dates
ORDER BY MONTH, YEAR;
```

Listing 18: Shows Added in Same Month Over Years

---

19. Top 3 most frequently cast actors per country

Problem Statement: Find the top three actors most frequently featured in Netflix content from each country. This helps identify key actors across various regional productions.

Approach:

- Join `show_cast`, `cast_members`, `show_countries`, and `countries`.
- Use `COUNT()` to compute the number of appearances per actor-country pair.
- Apply `DENSE_RANK()` over each country partition ordered by the show count.
- Filter to only include the top 3 actors per country.

SQL Solution:

```sql
WITH cte AS (
    SELECT CM.NAME AS ACTOR_NAME, C.NAME AS COUNTRY,
           COUNT(SCO.SHOW_ID) AS NO_OF_SHOWS,
           DENSE_RANK() OVER(PARTITION BY C.NAME ORDER BY COUNT(SCO.
    SHOW_ID) DESC) AS RANK
    FROM SHOW_CAST AS SC
    INNER JOIN CAST_MEMBERS AS CM ON SC.ACTOR_ID = CM.ID
    INNER JOIN SHOW_COUNTRIES AS SCO ON SC.SHOW_ID = SCO.SHOW_ID
    INNER JOIN COUNTRIES AS C ON SCO.COUNTRY_ID = C.ID
    WHERE C.NAME NOT ILIKE 'UNKNOWN' AND CM.NAME NOT ILIKE 'UNKNOWN'
    GROUP BY COUNTRY, ACTOR_NAME
)
SELECT * FROM CTE WHERE RANK <= 3;
```

Listing 19: Top 3 Actors per Country

20. Top 10 countries by total movie duration

Problem Statement: Identify countries that have produced the longest total duration of movies. This provides insight into which countries contribute the most runtime.

Approach:

- Join `shows`, `duration_types`, `show_countries`, `countries`, and `types`.
- Filter for movie-type shows with durations measured in minutes.
- Aggregate total durations by country and order descending.
- Limit the result to the top 10.

SQL Solution:

```sql
SELECT C.NAME AS COUNTRY_NAME, SUM(S.DURATION_VALUE) || ' min' AS
    TOTAL_MOVIES_DURATION_LENGTH
FROM SHOWS S
INNER JOIN DURATION_TYPES DT ON S.DURATION_TYPE_ID = DT.ID
INNER JOIN SHOW_COUNTRIES SC ON SC.SHOW_ID = S.SHOW_ID
INNER JOIN COUNTRIES C ON C.ID = SC.COUNTRY_ID
INNER JOIN TYPES T ON S.TYPE_ID = T.ID
WHERE DT.TYPE = 'min' AND C.NAME NOT ILIKE 'unknown'
GROUP BY C.NAME
ORDER BY SUM(S.DURATION_VALUE)::NUMERIC DESC
LIMIT 10;
```

Listing 20: Top 10 Countries by Movie Duration

21. One-hit directors (only one show)

Problem Statement: Find directors who have only one show in the entire Netflix database. This helps identify rare or debut appearances.

Approach:

- Use a CTE to get `director_id`s with exactly one show by grouping and filtering via `HAVING COUNT(DISTINCT show_id) = 1`.
- Join the result back with the `directors` table to fetch names.
- Exclude unknown directors and sort alphabetically.

SQL Solution:

```sql
WITH CTE AS (
    SELECT DIRECTOR_ID
    FROM SHOW_DIRECTORS
    GROUP BY DIRECTOR_ID
    HAVING COUNT(DISTINCT SHOW_ID) = 1
)
SELECT D.NAME
FROM DIRECTORS AS D
INNER JOIN CTE AS C ON D.ID = C.DIRECTOR_ID
WHERE D.NAME NOT ILIKE 'UNKNOWN'
ORDER BY D.NAME;
```

Listing 21: One-Hit Directors

---

22. Movies with longest duration per release year

Problem Statement: Identify the movie with the longest runtime for each release year to find yearly standout content.

Approach:

- Filter `shows` for entries where `duration_type` is in minutes.
- Use a CTE and apply `DENSE_RANK()` partitioned by `release_year` and ordered by descending duration.
- Select entries ranked 1 in each year.

SQL Solution:

```sql
WITH CTE AS (
    SELECT S.TITLE, S.DURATION_VALUE, S.RELEASE_YEAR,
           DENSE_RANK() OVER(PARTITION BY S.RELEASE_YEAR ORDER BY S.
   DURATION_VALUE DESC) AS RANK
    FROM SHOWS S
    INNER JOIN DURATION_TYPES DT ON S.DURATION_TYPE_ID = DT.ID
    WHERE DT.TYPE ILIKE 'min'
)
SELECT TITLE AS MOVIE, DURATION_VALUE || ' min' AS DURATION_OF_MOVIE,
   RELEASE_YEAR
FROM CTE
WHERE RANK = 1;
```

Listing 22: Longest Movies per Year

23. Actors in multiple shows with the same director

Problem Statement: Identify actor-director pairs who have collaborated on more than one show. This can indicate strong working relationships or recurring partnerships in Netflix productions.

Approach:

- Join `shows`, `show_directors`, `directors`, `show_cast`, and `cast_members`.
- Group by both director and actor names.
- Use `HAVING COUNT(*) > 1` to filter those appearing in multiple shows together.
- Exclude unknown entries.

SQL Solution:

```sql
SELECT D.NAME AS DIRECTOR, CM.NAME AS ACTOR, COUNT(S.SHOW_ID) AS
    NO_OF_SHOWS_TOGETHER
FROM SHOWS AS S
INNER JOIN SHOW_DIRECTORS AS SD ON S.SHOW_ID = SD.SHOW_ID
INNER JOIN DIRECTORS AS D ON SD.DIRECTOR_ID = D.ID
INNER JOIN SHOW_CAST AS SC ON S.SHOW_ID = SC.SHOW_ID
INNER JOIN CAST_MEMBERS AS CM ON SC.ACTOR_ID = CM.ID
WHERE D.NAME NOT ILIKE 'UNKNOWN' AND CM.NAME NOT ILIKE 'UNKNOWN'
GROUP BY D.NAME, CM.NAME
HAVING COUNT(S.SHOW_ID) > 1
ORDER BY NO_OF_SHOWS_TOGETHER DESC;
```

Listing 23: Recurring Actor-Director Collaborations

---

24. Procedure to get shows by added year

Problem Statement: Create a stored procedure that, when given a year, returns all shows added to Netflix in that year. This is useful for reusable querying in dynamic interfaces.

Approach:

- Create a PL/pgSQL procedure that accepts a target year as an input parameter.
- Use a `FOR` loop to iterate through and print shows that match the input year.
- Filter using `TO_CHAR(added_date, 'YYYY')` for string comparison.

SQL Solution:

```sql
CREATE OR REPLACE PROCEDURE get_shows_by_year(target_year INT)
LANGUAGE plpgsql
AS $$
DECLARE
    record RECORD;
BEGIN
    RAISE NOTICE 'Shows added in %:', target_year;

    FOR record IN
        SELECT title, added_date
        FROM shows
        WHERE TO_CHAR(added_date, 'YYYY') = target_year::TEXT
    LOOP
        RAISE NOTICE 'Title: %, Added Date: %', record.title, record.
    added_date;
    END LOOP;
END;
$$;

CALL get_shows_by_year(2020);
```

Listing 24: Procedure for Shows by Year

---

25. Shows with duplicate actor entries

Problem Statement: Find shows in which the same actor appears multiple times due to data entry duplications or multiple roles. Useful for cleaning or validating cast data.

Approach:

- Join `show_cast`, `shows`, and `cast_members`.
- Group by show title and actor name.
- Use `HAVING COUNT(*) > 1` to detect repeated entries.
- Order by appearance count in descending order.

SQL Solution:

```sql
SELECT
    S.TITLE AS SHOW_TITLE,
    CM.NAME AS ACTOR_NAME,
    COUNT(*) AS APPEARANCES
FROM
    SHOW_CAST SC
JOIN SHOWS S ON SC.SHOW_ID = S.SHOW_ID
JOIN CAST_MEMBERS CM ON SC.ACTOR_ID = CM.ID
GROUP BY
    S.TITLE, CM.NAME
HAVING
    COUNT(*) > 1
ORDER BY
    APPEARANCES DESC;
```

Listing 25: Duplicate Actor Appearances

🎓 Skills Acquired

Throughout the development of this Netflix data analysis project, I gained hands-on experience with a wide range of technical and analytical skills. These are the key competencies I developed:

- **PostgreSQL:** Mastered advanced SQL techniques including aggregate functions, joins, subqueries, window functions, and stored procedures to analyze complex datasets.
- **pgAdmin 4:** Utilized pgAdmin for managing PostgreSQL databases, running and testing queries, inspecting schemas, and visualizing data structures effectively.
- **Data Cleaning using Pandas:** Cleaned and transformed raw Netflix data using Python's Pandas library within Jupyter Notebook, ensuring consistency and accuracy before loading into the database.
- **Jupyter Notebook:** Used as the primary environment for exploratory data analysis (EDA), preprocessing CSV files, and validating the logic before database integration.
- **DBMS Concepts:** Applied core database principles such as normalization, foreign key relationships, and indexing to design a scalable and consistent schema.
- **Entity-Relationship Diagrams:** Designed and interpreted ER diagrams using dbdiagram.io to map out table relationships, ensuring a normalized and query-efficient structure.
- **Analytical Thinking:** Tackled business-relevant problems by breaking them down into logical SQL queries, allowing for meaningful insights into content distribution, user trends, and production patterns.
- **Data Storytelling:** Organized insights into a clean and interpretable narrative through queries, documentation, and visual elements like ER diagrams, making technical findings more accessible.

Conclusion

This Netflix data analysis project has demonstrated the power of structured querying and relational modeling in solving real-world business problems. By designing a normalized database and writing a wide variety of SQL queries, I was able to uncover key insights such as popular genres, active contributors, content trends across countries, and patterns in viewer offerings over time.

The queries developed not only reflect proficiency in SQL but also tackle practical business questions faced by a company like Netflix—such as content planning, audience targeting, and understanding creator contributions. The use of stored procedures and window functions further illustrates how automation and advanced techniques can be leveraged for scalable analysis.

Ultimately, this project serves as a strong example of how technical skills in data engineering and analytics can be used to drive strategic decisions in the streaming industry.