# IMDB Clone Project - SQL Query Documentation

Submitted by:

• Yashvi TANK  
• Jesica KARDOS  
• Jay PATEL

Course: Database Systems

Project: IMDB Clone - Backend and Query Implementation

We were on the goal of creating the whole clone for the project but later on we didn’t have much time to do so. Hence, we stopped at a point and started working on the main thing which was important for submission which is database basically. So, the project structure might show frontend and backend part which is not fully complete yet and which will be worked upon later on. But here is our submission for the whole project of Relational Databases.

Additionally, we have other queries mentioned and explained inside our queries folder in the project. Do not hesitate to contact us if something’s off the guard.

## 1. Load Data into 'title' Table

We extracted relevant fields from the staging\_title\_basics table, including primary titles, start and end years, and runtime. Non-numeric values were filtered and cleaned before casting to integers.

SQL Query:

INSERT INTO title (title\_id, tconst, primary\_title, original\_title, start\_year, end\_year, runtime\_minutes)  
SELECT sb.stb\_id::int, sb.tconst, sb.primary\_title, sb.original\_title,  
 CASE WHEN sb.start\_year ~ '^[0-9]+$' THEN sb.start\_year::int END AS start\_year,  
 CASE WHEN sb.end\_year ~ '^[0-9]+$' THEN sb.end\_year::int END AS end\_year,  
 CASE WHEN sb.runtime\_minutes ~ '^[0-9]+$' THEN sb.runtime\_minutes::int END AS runtime\_minutes  
FROM staging\_title\_basics sb  
WHERE sb.start\_year <> 'startYear' AND sb.end\_year <> 'endYear' AND sb.runtime\_minutes <> 'runtimeMinutes'  
ON CONFLICT DO NOTHING;

## 2. Load Data into 'genre' and 'title\_genre' Tables

Genres were split from a pipe-separated list in the staging table and joined with normalized genre names to ensure consistency.

SQL Query:

DO $$  
DECLARE  
 start\_id INT := 1;  
 end\_id INT := 100000;  
BEGIN  
 LOOP  
 INSERT INTO title\_genre (title\_id, genre\_id)  
 SELECT t.title\_id, g.genre\_id  
 FROM staging\_title\_basics sb  
 JOIN title t ON t.tconst = sb.tconst  
 CROSS JOIN LATERAL unnest(string\_to\_array(sb.genres, '|')) AS genre\_name  
 JOIN genre g ON g.name = genre\_name  
 WHERE sb.stb\_id BETWEEN start\_id AND end\_id  
 AND sb.genres IS NOT NULL AND sb.genres <> '\N'  
 ON CONFLICT DO NOTHING;  
  
 RAISE NOTICE 'Inserted batch % - %', start\_id, end\_id;  
 EXIT WHEN end\_id > 11700000;  
 start\_id := end\_id + 1;  
 end\_id := end\_id + 100000;  
 END LOOP;  
END  
$$ LANGUAGE plpgsql;

## 3. Load Data into 'person' Table

Basic name data was pulled from the name staging table. Only rows with proper birth and death years were included. Header rows were filtered.

SQL Query:

INSERT INTO person (person\_id, nconst, primary\_name, birth\_year, death\_year)  
SELECT sb.sp\_id::int, sb.nconst, sb.primary\_name,  
 CASE WHEN sb.birth\_year ~ '^[0-9]+$' THEN sb.birth\_year::int END,  
 CASE WHEN sb.death\_year ~ '^[0-9]+$' THEN sb.death\_year::int END  
FROM staging\_name\_basics sb  
WHERE sb.nconst <> 'nconst'  
ON CONFLICT DO NOTHING;

## 4. Load Data into 'title\_person' Table

This table links people to titles based on the principal category (e.g., actor, director). We also generated a serial ID and used safe casting for ordering.

SQL Query:

INSERT INTO title\_person (title\_id, person\_id, category, job, characters, ordering)  
SELECT t.title\_id, p.person\_id, st.category, st.job, st.characters,  
 CASE WHEN st.ordering ~ '^[0-9]+$' THEN st.ordering::int END  
FROM staging\_title\_principals st  
JOIN title t ON t.tconst = st.tconst  
JOIN person p ON p.nconst = st.nconst  
WHERE st.nconst <> 'nconst'  
ON CONFLICT DO NOTHING;

## 5. Analysis Query: Ratings by Decade

Groups average ratings by decade and returns total count of titles in each group.

SQL Query:

SELECT (t.start\_year/10)\*10 AS decade,  
 ROUND(AVG(r.average\_rating)::numeric,2) AS avg\_rating,  
 COUNT(\*) AS titles\_count  
FROM title t  
JOIN rating r USING (title\_id)  
GROUP BY decade  
ORDER BY decade;

## 6. Analysis Query: Top Directors by Average Rating

Finds directors with at least 5 rated movies and calculates their average rating.

SQL Query:

SELECT p.primary\_name,  
 ROUND(AVG(r.average\_rating)::numeric,2) AS avg\_director\_rating,  
 COUNT(\*) AS movie\_count  
FROM title\_person tp  
JOIN person p USING (person\_id)  
JOIN rating r ON tp.title\_id = r.title\_id  
WHERE tp.category = 'director'  
GROUP BY p.primary\_name  
HAVING COUNT(\*) >= 5  
ORDER BY avg\_director\_rating DESC  
LIMIT 20;

## 7. Analysis Query: Genre Popularity Over Time

This query calculates the number of titles per genre for each year.

SQL Query:

SELECT g.name AS genre, t.start\_year AS year, COUNT(\*) AS titles\_count  
FROM title t  
JOIN title\_genre tg USING (title\_id)  
JOIN genre g USING (genre\_id)  
GROUP BY g.name, year  
ORDER BY g.name, year;

## 8. Analysis Query: Writers vs. Actors Role Count

Counts how many titles each actor and writer has contributed to.

SQL Query:

SELECT tp.category, p.primary\_name, COUNT(\*) AS roles\_count  
FROM title\_person tp  
JOIN person p USING (person\_id)  
WHERE tp.category IN ('writer','actor')  
GROUP BY tp.category, p.primary\_name  
ORDER BY tp.category, roles\_count DESC  
LIMIT 20;

Completed by:

Yashvi TANK, Jesica KARDOS, Jay PATEL