Analysis of a Film Rental Company’s Database in SQL.

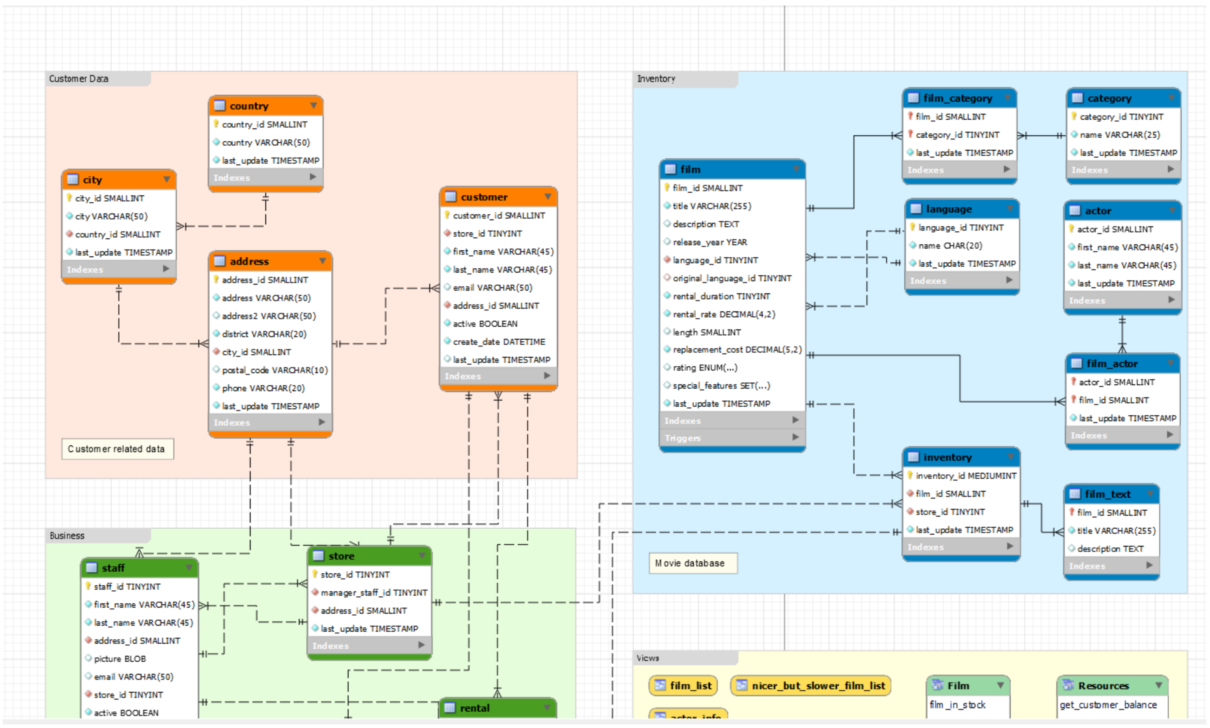
In the realm of data analysis, understanding consumer behaviour and market trends is paramount. This project delves into the intricacies of a comprehensive film rental database comprising multiple interconnected tables. Leveraging SQL queries, I've navigated through this rich dataset, uncovering valuable insights that provides information about customer preferences, rental patterns, and global distribution dynamics. Through a series of analytical tasks, I've investigated diverse aspects, including film popularity, geographical trends in customer engagement, inventory management, and financial performance metrics. These analyses not only offer a glimpse into consumer behaviour but also pave the way for strategic decision-making and optimizing business operations within the film rental domain.

Exploratory Data Analysis:

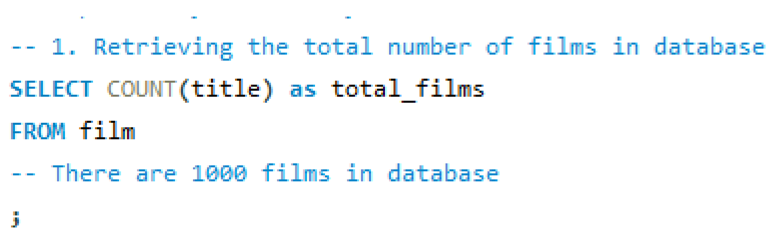
I began my work by exploring the contents of the database in the form of EER Diagram:

1) analysed content of each table,

2) tried to understand relationships between them, specifically aiming to identify the presence of foreign keys - key elements establishing connections between different tables within database.

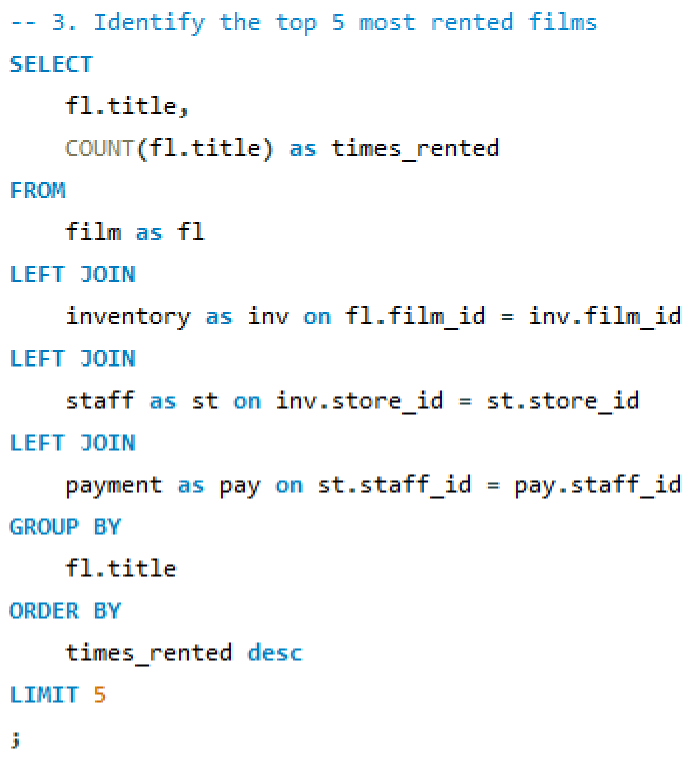


1. For reference, I’ve obtained total number of movies present in the database.



```
-- 1. Retrieving the total number of films in database
SELECT COUNT(title) as total_films
FROM film
-- There are 1000 films in database
;
```

2. Managed to identify the top 5 most rented films in the database. It will help to understand customer preferences, improve inventory management, marketing and promotional direction.

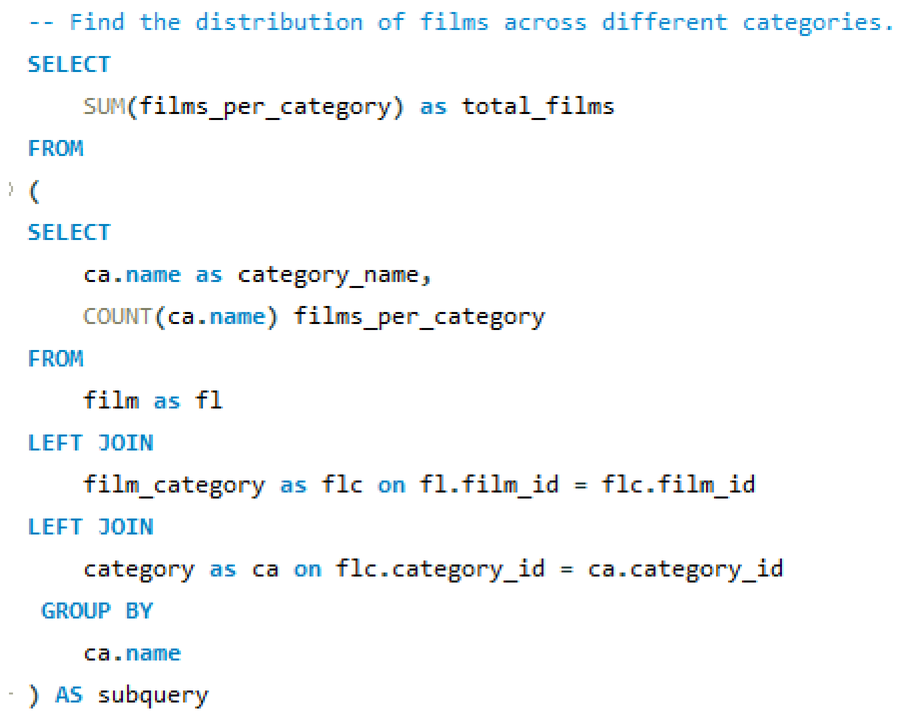


```
-- 3. Identify the top 5 most rented films
SELECT
    fl.title,
    COUNT(fl.title) as times_rented
FROM
    film as fl
LEFT JOIN
    inventory as inv on fl.film_id = inv.film_id
LEFT JOIN
    staff as st on inv.store_id = st.store_id
LEFT JOIN
    payment as pay on st.staff_id = pay.staff_id
GROUP BY
    fl.title
ORDER BY
    times_rented desc
LIMIT 5
;
```

1) Previously, I determined that the database contains a total of 1000 movies. To validate the accuracy of the count function in the second task, I verify if the returned number of rows equals 1000.
2) I've utilized foreign keys to establish connections between tables through a LEFT JOIN. Details regarding movie titles were sourced from the 'film' table, while sales information - from 'payment' table.
3) I’ve GROUPED BY title column from the 'film' table.
4) I used the 'ORDER BY' to arrange numeric results in descending order, ensuring that movies with the highest number of rentals would appear at the top rows

5) **Subsequently, I employed the LIMIT function to identify the top 5 movies with the highest rental counts.**

3. Finding the distribution of films across different categories. - It aids in making informed decisions about acquiring similar content or genres for future additions to the catalogue. It assists in selecting new films that are likely to resonate with the customer base.



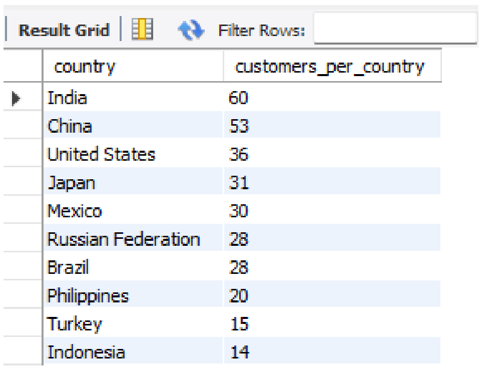
```
-- Find the distribution of films across different categories.
SELECT
    SUM(films_per_category) as total_films
FROM
(
SELECT
    ca.name as category_name,
    COUNT(ca.name) films_per_category
FROM
    film as fl
LEFT JOIN
    film_category as flc on fl.film_id = flc.film_id
LEFT JOIN
    category as ca on flc.category_id = ca.category_id
 GROUP BY
    ca.name
) AS subquery
```

1) **Once more used 'LEFT JOIN' operation to establish connection between 'film', 'film_category' and 'category' tables to get required data related to movie categories and enabling usage of 'COUNT' function, to evaluate distribution by category metrics.**
2) **Original name of the field representing movie categories in the 'category' table was 'name', which seemed misleading. To enhance clarity, I changed the field name to 'category_name' to prevent any confusion in a future.**
3) **Subsequently, I integrated it into a subquery to aggregate the total sum of movies, ensuring all of them are included. Following this, I removed the subquery, retaining the original query with two distinct columns—'category_name' and 'films_per_category'—to eliminate any potential confusion.**

4. Understanding the customer distribution by country metric enables a company to navigate cultural preferences and tendencies among its customers. This knowledge plays pivotal role in making informed decisions about providing a broader selection of movies, effectively meeting customer expectations.

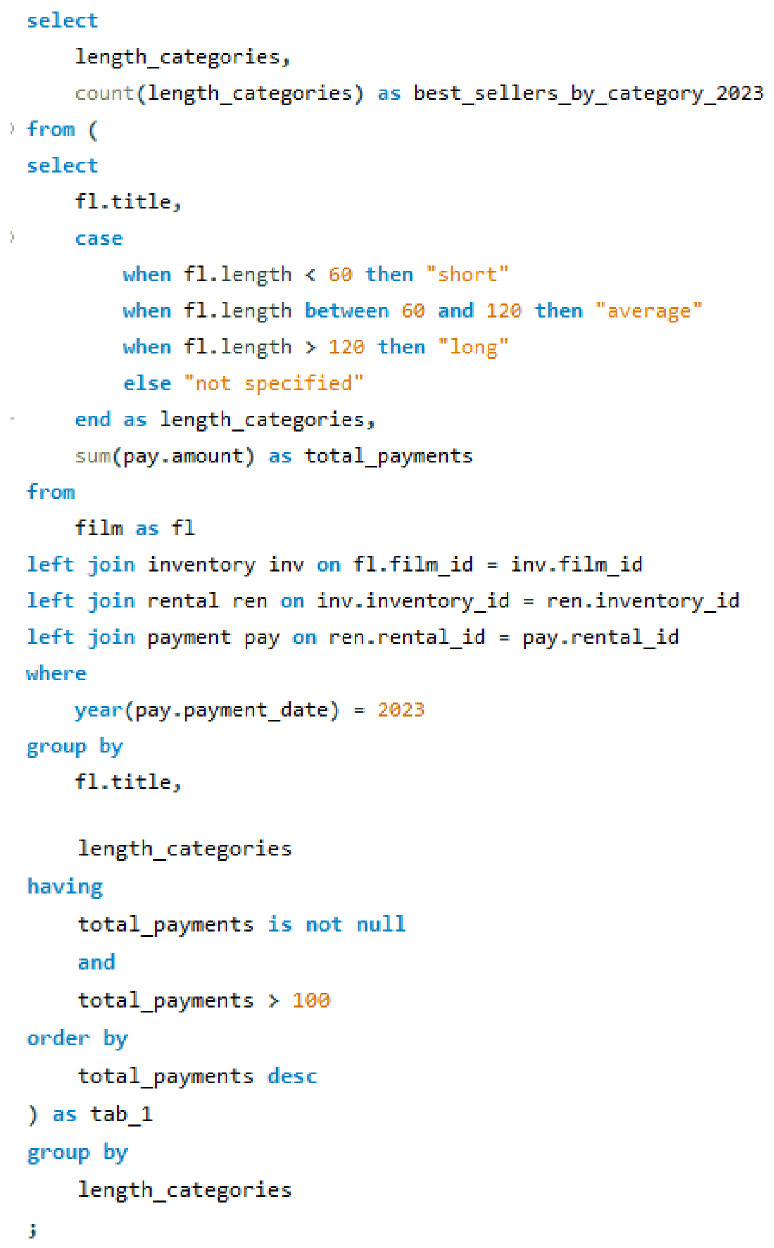
```sql
-- Analyze the distribution of customers by country. Identify TOP 10 countries

SELECT
    ctr.country,
    COUNT(cs.customer_id) as customers_per_country
FROM
    country as ctr
    LEFT JOIN city as ci on ctr.country_id = ci.country_id
    LEFT JOIN address as ad on ci.city_id = ad.city_id
    LEFT JOIN customer as cs on ad.address_id = cs.address_id
GROUP BY
    ctr.country
ORDER BY
    customers_per_country desc
LIMIT 10
;
```

1) 'LEFT JOIN' to establish connection between 'country', 'city', 'address' and 'customer' tables to get required data about customers and their respected countries. Enabling usage of 'COUNT' function, to evaluate distribution by country.
2) 'GROUP BY' – grouping our results by country.
3) 'ORDER BY' - arranging numeric results in descending order, ensuring that countries with the highest number of customers would appear at the top rows.
4) Using LIMIT function to identify the top 10 countries with the highest customer counts.



country	customers_per_country
India	60
China	53
United States	36
Japan	31
Mexico	30
Russian Federation	28
Brazil	28
Philippines	20
Turkey	15
Indonesia	14

5. Today, I received a complex task requiring a SQL query to identify the best-selling movies categorized by their length (short, average, long). The aim was to compare results by selecting movies rented out in 2023 that earned at least $100 in a year. This valuable data helps us gain a clear understanding of customer preferences regarding the length of movies they prefer to rent.



```
select
    length_categories,
    count(length_categories) as best_sellers_by_category_2023
from (
select
    fl.title,
    case
        when fl.length < 60 then "short"
        when fl.length between 60 and 120 then "average"
        when fl.length > 120 then "long"
        else "not specified"
    end as length_categories,
    sum(pay.amount) as total_payments
from
    film as fl
left join inventory inv on fl.film_id = inv.film_id
left join rental ren on inv.inventory_id = ren.inventory_id
left join payment pay on ren.rental_id = pay.rental_id
where
    year(pay.payment_date) = 2023
group by
    fl.title,

    length_categories
having
    total_payments is not null
    and
    total_payments > 100
order by
    total_payments desc
) as tab_1
group by
    length_categories
;
```

1) Initially, I utilized 'JOIN' statements to connect the 'film' and 'payment' tables, allowing me to pair movie titles with their respective payment information.

2) Subsequently, I categorized movies by duration using the 'CASE' statement to create four categories: 'Short' for movies under 60 minutes, 'Average' for those lasting between 60 and 120 minutes, 'Long' for movies exceeding 120 minutes and ‘not specified’ group in case there is an error .

3) **I applied the 'SUM' aggregation function to calculate the total income generated by each movie, narrowing down the search to the year 2023 with a 'WHERE' statement.**

4) **I excluded movies with 'NULL' values in the 'total_payment' field and refined the selection to movies earning more than $100 in 2023.**

5) **To count the best-selling movies by 'length_category', I constructed another select function, utilizing the previous query's results as a table within the 'FROM' statement to finalize my findings.**

6) **The analysis revealed that movies of 'Average' and 'Long' duration categories were the most popular based on rental patterns.**

6. Temporary tables in SQL enable us to create them within our temporary query operations, offering benefits through the 'DROP' statement. This statement frees computer memory space, ensuring smoother SQL operation without overwhelming the system. Temporary tables allow us to work on distinct query content, making it easier and faster to identify issues in case of errors compared to complex and longer queries.

   In a specific example, I demonstrate the creation and management of three temporary tables. This demonstration showcases not only their utilization but also how effectively the 'DROP' function releases operational memory space the computer (Query on the next page).

```
create temporary table FILMFILTR (

select
    film_id,
    title,
    description,
    rating,
    case
        when rating = "PG" or rating = "G" then "PG_G"
        when rating = "NC-17" or rating = "PG-13" then "NC-17-PG-13"
        else "Nesvarbu"
    end as group_of_ratings
from
    film
where
    rating = "PG" or rating = "G"
)

;

create temporary table AKTORSK (

select
    fl.title,
    count(a.actor_id) total_actors
from
    film fl
left join film_actor fla on fl.film_id = fla.film_id
left join actor a on fla.actor_id = a.actor_id
group by
    fl.title
)
;

create temporary table REZULTATAS (
select
    f.film_id,
    f.title,
    f.description,
    f.group_of_ratings,
    a.total_actors
from
    FILMFILTR f
    left join AKTORSK a on f.title = a.title
 )
 ;
 drop table FILMFILTR;
 drop table AKTORSK;
```