Database Management through Schema-Based Database Design: A Case Study on a Movie Streaming Platform Database

ABSTRACT

This project revolves around the creation of a Movie Streaming Platform database . The database contains generated subscribers information. Which includes subscription details, demographic data, and viewing statistics . Utilizing Python pandas library, Numpy and matplotlib for data manipulation, generation of randomized datasets, ensuring a diverse and representative datasets. major components of the database includes , subscription IDs,subscriber names, age groups, subscription dates, subscription plans, subscription prices, and minutes watched. The primary aim of this project is to provide a structured dataset for practicing SQL queries in a real world context. The database created offers a foundation for executing SQL queries to extract valuable insights and perform analytical tasks.

INTRODUCTION

The database comprises various aspects of subscriber data, similar to a real world scenario encountered in a streaming service like Netflix. From the subscription to the viewing habit that is the amount of time in minutes spent streaming different movies. The database offers a detailed but manageable scope for SQL exploration, through the integration of python libraries , synthetic data generation ensuring completeness and versatility. With simplicity and usefulness at its core, this database serves as an invaluable resource for sharpening SQL skills and fostering a deeper understanding of database management concepts.

NAME: **ABODUNRIN ADEYEMI**
STUDENT ID: **22098853**

I imported necessary libraries , numpy for numerical computing , pandas for data manipulation and matplotlib for visualization.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

np.loadtxt(): This is a function provided by numpy, for numerical computing. It was used to load the names generated from random name generator websites: http://random-name-generator.info/ and converted into a csv file.
The whole line of code loads the data from a csv file named "full name".The array contains strings, and each row of the CSV file corresponds to a full name.

```
# Number of subscribers
num_subs = 1000

#import Names : Nominal data
full_names =
np.loadtxt('/content/drive/MyDrive/names.csv',delimiter=',',dtype=str)
print(full_names)
```

This line of code generates a random subscription ID, such that only three characters is added to the prefect MV. in this format 'MVxxx', where 'xxx' is a three-digit integer between 001 and 1000, ensuring that each generated ID is unique and is within the specified range.

```
# create subscription IDs
# Initialization of an empty set to store generated subscription IDs
unique_ids = set()

# Generate unique subscription IDs
while len(unique_ids) < num_subs:
   # Generate a random subscription ID
   subscription_ID = f'MV{np.random.randint(1, 1001):03}'
   # Adds ID to the set
   unique_ids.add(subscription_ID)

# Converts the set to a list
subscription_ID = list(unique_ids)
print(subscription_ID)
```

This line of code randomly generates the age group of subscribers. The age group is used to represent ordinal data . The list of the age group is generated by sampling the number of samples.

```
# Ordinal data: Age groups
np.random.seed(32)
age_groups = ['18-25', '26-35', '36-45', '46-55', '56-65', '66+']
age_group_data = np.random.choice(age_groups, num_subs, p=[0.1, 0.2, 0.3,
0.2, 0.1, 0.1])
```

This line constructs an array of subscription dates by combining subscription years, months, and days into strings in the format 'YYYY-MM-DD'. It uses list creation based on the existing to iterate over the indices of the arrays' subscription year, subscription month, and subscription day, and constructs each subscription date using string formatting. The subscription date falls under the category of interval data. It allows for calculations such as finding the duration between two subscription dates, determining the frequency of subscriptions in a specific time interval.

```
# Interval data: date of subscription
np.random.seed(32)
subscription_year = np.random.randint(2020, 2023, num_subs)
subscription_month = np.random.randint(1, 13, num_subs)
subscription_day = np.random.randint(1, 32, num_subs)
subscription_date =
[f'{subscription_year[i]}-{str(subscription_month[i]).zfill(2)}-'
                    f'{str(subscription_day[i]).zfill(2)}' for i in
range(num_subs)]
```

This line of code generates different subscription plans which are basic, standard and premium users,allowing for each subscriber to be associated with one of the available subscription plans.

```
# Define subscription plans
np.random.seed(32)
subscription_plans = ['Basic', 'Standard', 'Premium']

# create subscription plan values
subscription_plan_data = np.random.choice(subscription_plans, num_subs)
```

Basic_price_range, standard_price_range, and premium_price_range: These are variables representing the price ranges for each subscription plan. Each variable is assigned a single value, indicating the price for the corresponding subscription plan. The lowest possible price is assigned to the basic plan and the highest prices are assigned to the premium.

```
# Ratio data: subscription price
# Define price ranges for each subscription plan
basic_price = (50)
standard_price = (150)
premium_price = (250)
```

```
#  subscription plan data
np.random.seed(32)
subscription_plan_data = np.random.choice(subscription_plans, num_subs)

# Assign prices based on subscription plan
price_data = []
for plan in subscription_plan_data:
   if plan == 'Basic':
       price = (basic_price)
   elif plan == 'Standard':
       price = (standard_price)
   else:  # Premium
       price = (premium_price)
   price_data.append(price)
```

This line of code was used to generate the minutes watched ranging from the minimum amount of time it takes to watch a standard movie and a whole series respectively.

```
# create random values for Minutes_watched
np.random.seed(32)
Minutes_watched = []
while len(Minutes_watched) < num_subs:
   # Generate a random value
   mins = np.random.randint(135, 150000)
   # Append it to the list
   Minutes_watched.append(mins)

# Convert the list to a NumPy array
Minutes_watched = np.array(Minutes_watched)
```

This line of code creates a DataFrame where each column of the DataFrame corresponds to the following:

- subscription_ID: Subscription IDs generated .
- full_names: Full names loaded from the CSV file.
- Age_Group: Age group data generated.
- subscription_date: Subscription dates generated.
- Subscription_Plan: Subscription plan data generated.
- subscription_Price: Subscription price data generated .
- Minutes_watched: Minutes watched data generated .

The line also sets the index of the dataframe as the subscriptionID. This is Done for easy data manipulation.

```
# Create DataFrame
df = pd.DataFrame({
   'subscription_ID': subscription_ID,
   'full_names' : full_names,
   'Age_Group': age_group_data,
   'subscription_date': subscription_date,
   'Subscription_Plan' : subscription_plan_data,
   'subscription_Price': price_data,
   'Minutes_watched': Minutes_watched
})

df.set_index('subscription_ID', inplace=True)

print(df)
```

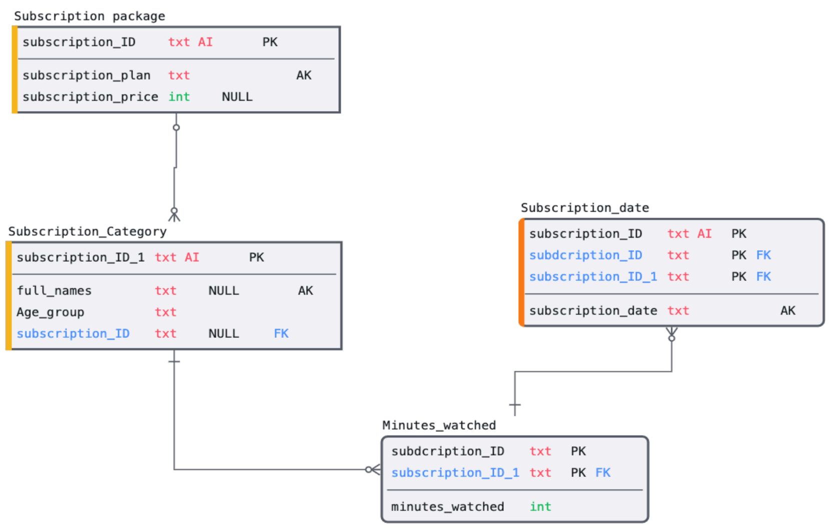
The line provides the summary of the DataFrame , including information about the index, column data types, non-null values, and memory usage.

```
df.info()
```

This last line of code is simply used to save the whole database in a csv file and also grouping into several categories and stored in csv to be used for other analysis. This grouping is also done to identify columns that share certain relationships. This leads to the explanation of primary,foreign and compound keys shown below;

```
df.to_csv('subs_data.csv')
df[['full_names','Age_Group']].to_csv('sub_category.csv')
df[['Subscription_Plan','subscription_Price']].to_csv('sub_package.csv')
df[['subscription_date']].to_csv('sub_date.csv')
df[['Minutes_watched']].to_csv('mins_watched.csv')
```

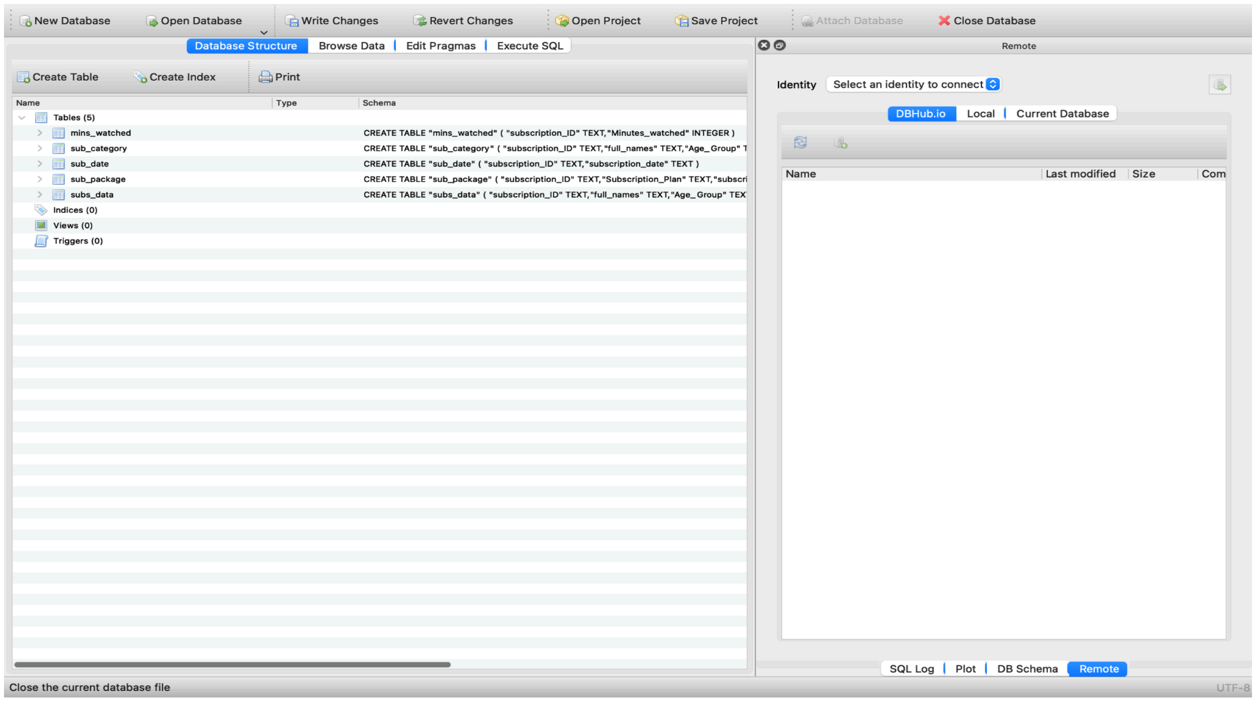
The schema shown below explains the relationship between the separate tables created . The names and age have a relationship likewise the subscription plan and price indicating the price for each plan and their prices. The date of subscription to know the day the subscription starts and when to renew. Minutes watched to show the amount of time spent streaming. While database schema has a lot of advantages, in this case it is used for;
Grouping local entities : facilitates easier management and administration of the database by organizing related objects together just as represented in the table above .
Reducing complexity: The database being splitted into separate tables enables target functionality. Easy access of certain entities in a database.


Subscription package
subscription_ID txt AI PK
subscription_plan txt AK
subscription_price int NULL
Subscription_Category
subscription_ID_1 txt AI PK
full_names txt NULL AK
Age_group txt
subscription_ID txt NULL FK
Subscription_date
subscription_ID txt AI PK
subdcription_ID txt PK FK
subscription_ID_1 txt PK FK
subscription_date txt AK
Minutes_watched
subdcription_ID txt PK
subscription_ID_1 txt PK FK
minutes_watched int

The name and age is in a separate table that protects the personal information about each user. What is common to all users is a unique Subscripcription ID which does not contain any personal information. This justifies the ethical issue of Data protection.

Data Security: Separating sensitive data into separate tables enhances data security and privacy. This is crucial for protecting user information and complying with privacy regulations.

Other tables as listed above show user plans, minutes watched , subscription date.



Name	Type	Schema
Tables (5)		
mins_watched		CREATE TABLE "mins_watched" ("subscription_ID" TEXT,"Minutes_watched" INTEGER)
sub_category		CREATE TABLE "sub_category" ("subscription_ID" TEXT,"full_names" TEXT,"Age_Group" T
sub_date		CREATE TABLE "sub_date" ("subscription_ID" TEXT,"subscription_date" TEXT)
sub_package		CREATE TABLE "sub_package" ("subscription_ID" TEXT,"Subscription_Plan" TEXT,"subscri
subs_data		CREATE TABLE "subs_data" ("subscription_ID" TEXT,"full_names" TEXT,"Age_Group" TEX
Indices (0)		
Views (0)		
Triggers (0)		

Close the current database file

Loading of the various tables into the SQL lite .

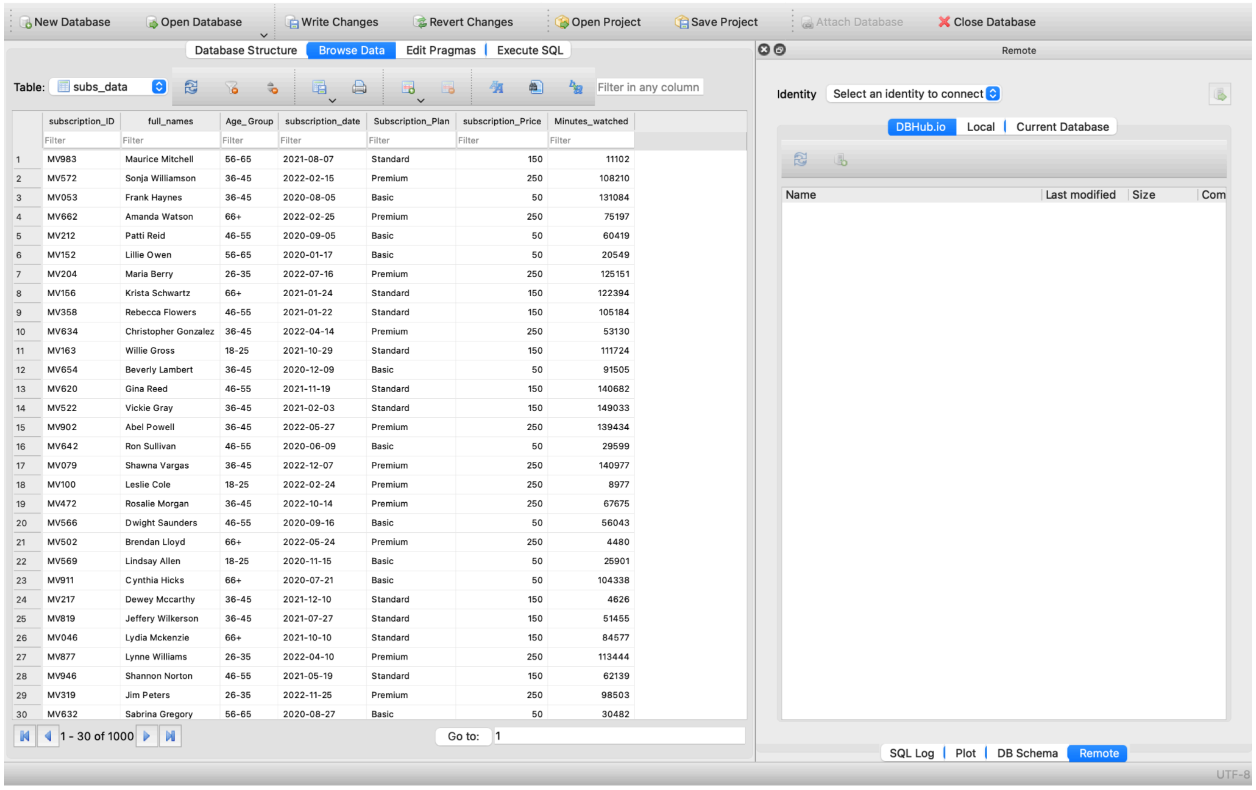


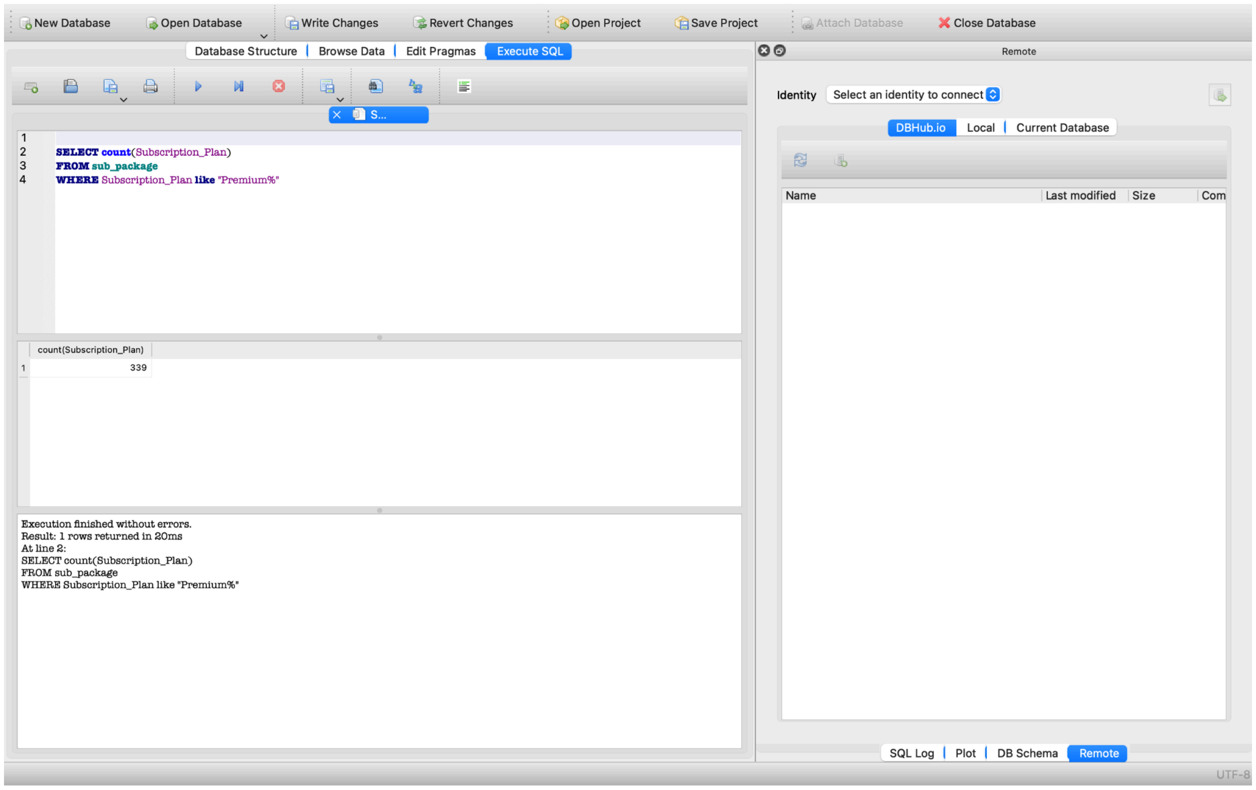
Table: subs_data

	subscription_ID	full_names	Age_Group	subscription_date	Subscription_Plan	subscription_Price	Minutes_watched
1	MV983	Maurice Mitchell	56-65	2021-08-07	Standard	150	11102
2	MV572	Sonja Williamson	36-45	2022-02-15	Premium	250	108210
3	MV053	Frank Haynes	36-45	2020-08-05	Basic	50	131084
4	MV662	Amanda Watson	66+	2022-02-25	Premium	250	75197
5	MV212	Patti Reid	46-55	2020-09-05	Basic	50	60419
6	MV152	Lillie Owen	56-65	2020-01-17	Basic	50	20549
7	MV204	Maria Berry	26-35	2022-07-16	Premium	250	125151
8	MV156	Krista Schwartz	66+	2021-01-24	Standard	150	122394
9	MV358	Rebecca Flowers	46-55	2021-01-22	Standard	150	105184
10	MV634	Christopher Gonzalez	36-45	2022-04-14	Premium	250	53130
11	MV163	Willie Gross	18-25	2021-10-29	Standard	150	111724
12	MV654	Beverly Lambert	36-45	2020-12-09	Basic	50	91505
13	MV620	Gina Reed	46-55	2021-11-19	Standard	150	140682
14	MV522	Vickie Gray	36-45	2021-02-03	Standard	150	149033
15	MV902	Abel Powell	36-45	2022-05-27	Premium	250	139434
16	MV642	Ron Sullivan	46-55	2020-06-09	Basic	50	29599
17	MV079	Shawna Vargas	36-45	2022-12-07	Premium	250	140977
18	MV100	Leslie Cole	18-25	2022-02-24	Premium	250	8977
19	MV472	Rosalie Morgan	36-45	2022-10-14	Premium	250	67675
20	MV566	Dwight Saunders	46-55	2020-09-16	Basic	50	56043
21	MV502	Brendan Lloyd	66+	2022-05-24	Premium	250	4480
22	MV569	Lindsay Allen	18-25	2020-11-15	Basic	50	25901
23	MV911	Cynthia Hicks	66+	2020-07-21	Basic	50	104338
24	MV217	Dewey Mccarthy	36-45	2021-12-10	Standard	150	4626
25	MV819	Jeffery Wilkerson	36-45	2021-07-27	Standard	150	51455
26	MV046	Lydia Mckenzie	66+	2021-10-10	Standard	150	84577
27	MV877	Lynne Williams	26-35	2022-04-10	Premium	250	113444
28	MV946	Shannon Norton	46-55	2021-05-19	Standard	150	62139
29	MV319	Jim Peters	26-35	2022-11-25	Premium	250	98503
30	MV632	Sabrina Gregory	56-65	2020-08-27	Basic	50	30482

1 - 30 of 1000

Go to: 1

Checking if all the data is completely loaded

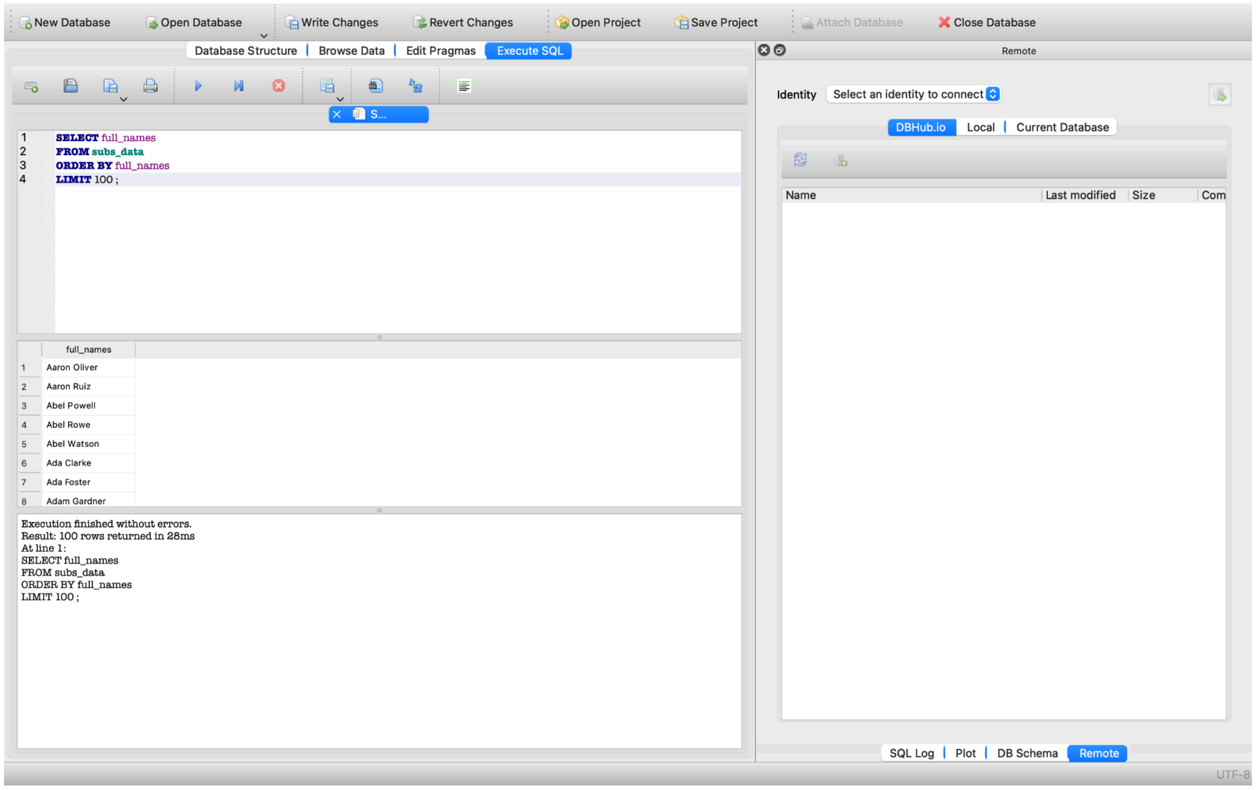


```
SELECT count(Subscription_Plan)
FROM sub_package
WHERE Subscription_Plan like "Premium%"
```

	count(Subscription_Plan)
1	339

```
Execution finished without errors.
Result: 1 rows returned in 20ms
At line 2:
SELECT count(Subscription_Plan)
FROM sub_package
WHERE Subscription_Plan like "Premium%"
```

This sql query shows the number of premium subscribers in the database which is 339 premium



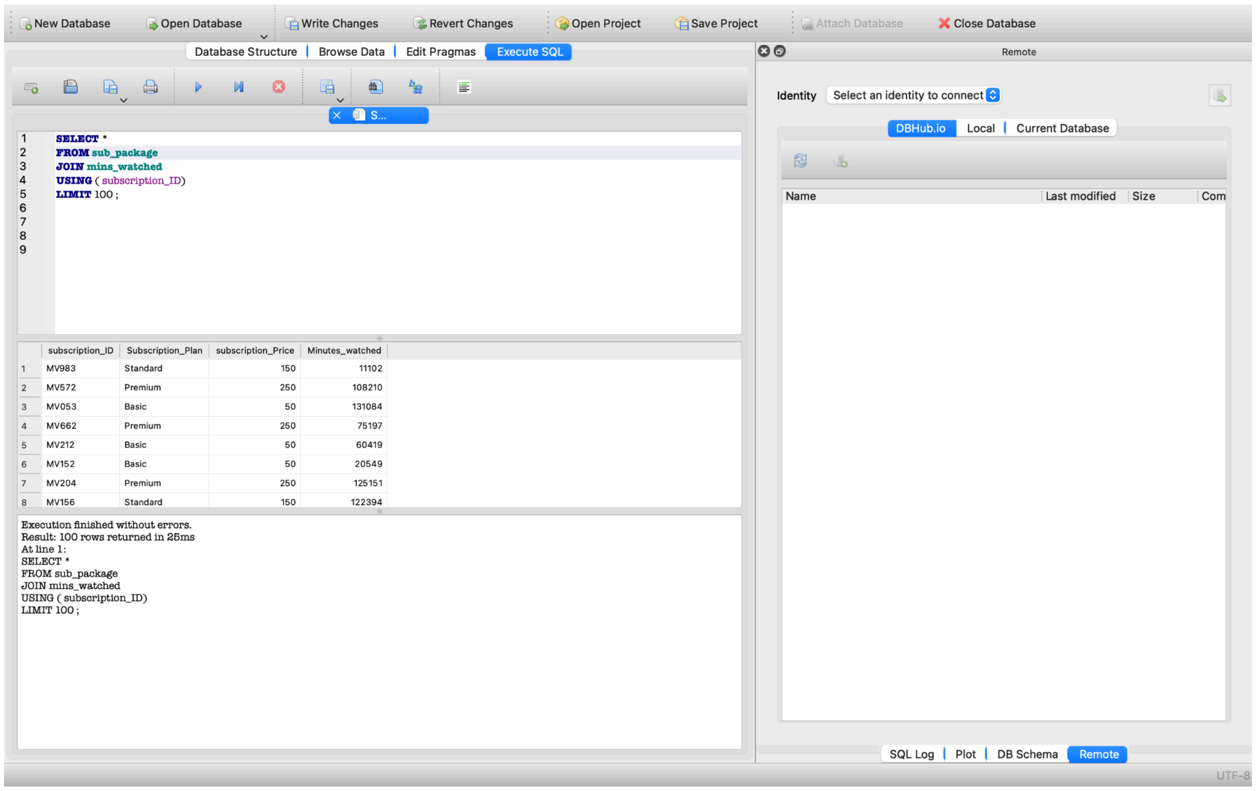
```
SELECT full_names
FROM subs_data
ORDER BY full_names
LIMIT 100 ;
```

	full_names
1	Aaron Oliver
2	Aaron Ruiz
3	Abel Powell
4	Abel Rowe
5	Abel Watson
6	Ada Clarke
7	Ada Foster
8	Adam Gardner

```
Execution finished without errors.
Result: 100 rows returned in 28ms
At line 1:
SELECT full_names
FROM subs_data
ORDER BY full_names
LIMIT 100 ;
```

This is to print all the full names

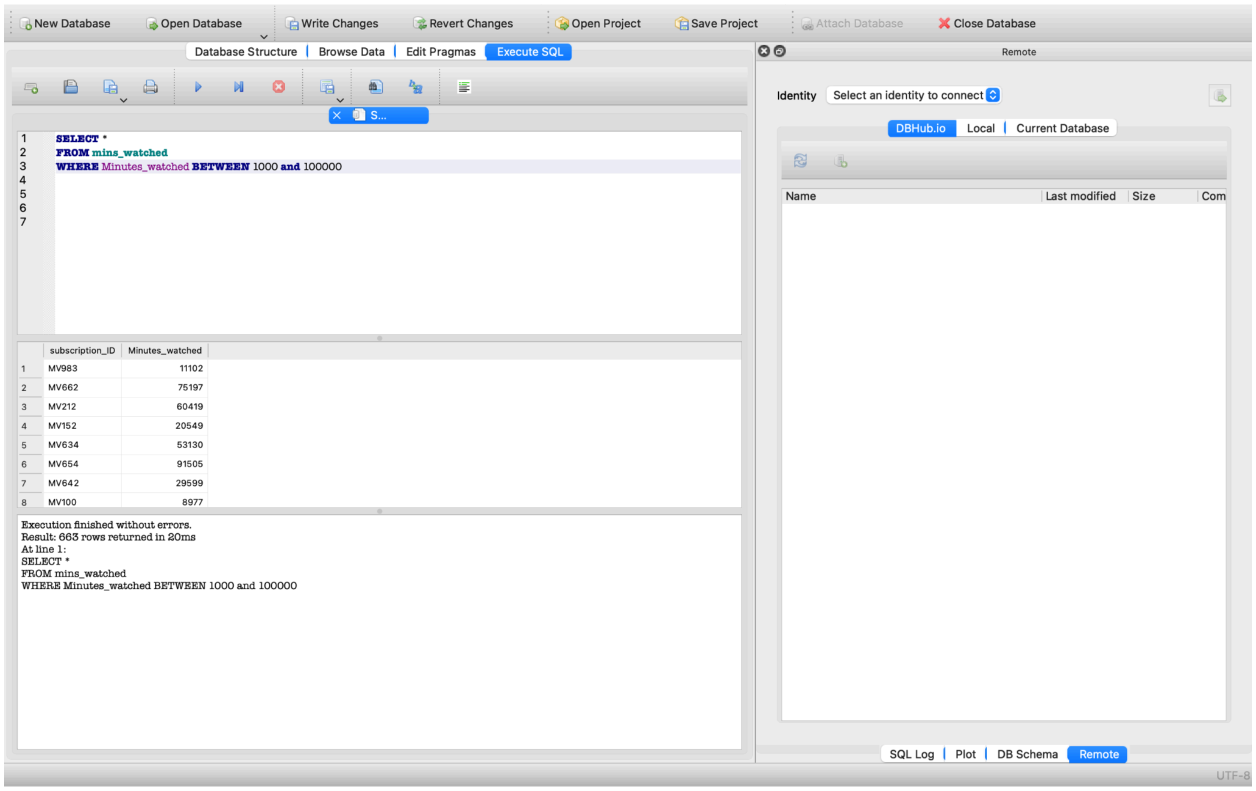
Join the minutes watch table and subscription package table using the Subscription ID which is common among all tables. Showing the first 100.



```
SELECT *
FROM sub_package
JOIN mins_watched
USING ( subscription_ID)
LIMIT 100 ;
```

	subscription_ID	Subscription_Plan	subscription_Price	Minutes_watched
1	MV983	Standard	150	11102
2	MV572	Premium	250	108210
3	MV053	Basic	50	131084
4	MV662	Premium	250	75197
5	MV212	Basic	50	60419
6	MV152	Basic	50	20549
7	MV204	Premium	250	125151
8	MV156	Standard	150	122394

```
Execution finished without errors.
Result: 100 rows returned in 25ms
At line 1:
SELECT *
FROM sub_package
JOIN mins_watched
USING ( subscription_ID)
LIMIT 100 ;
```

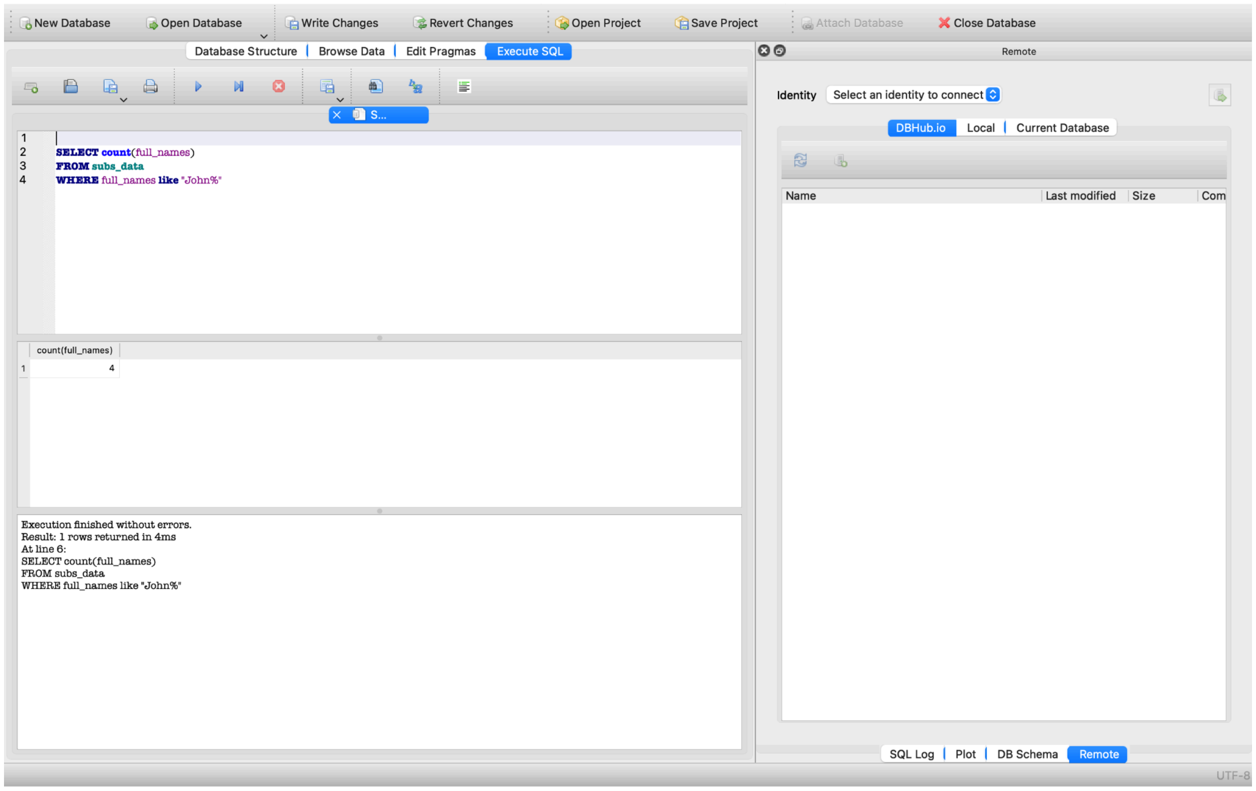


```
SELECT *
FROM mins_watched
WHERE Minutes_watched BETWEEN 1000 and 100000
```

	subscription_ID	Minutes_watched
1	MV983	11102
2	MV662	75197
3	MV212	60419
4	MV152	20549
5	MV634	53130
6	MV654	91505
7	MV642	29599
8	MV100	8977

```
Execution finished without errors.
Result: 663 rows returned in 20ms
At line 1:
SELECT *
FROM mins_watched
WHERE Minutes_watched BETWEEN 1000 and 100000
```

This query shows the Subscription ID of users who streams between 1,000 and 100,000 minutes



```

SELECT count(full_names)
FROM subs_data
WHERE full_names like "John%"
```

	count(full_names)
1	4

```
Execution finished without errors.
Result: 1 rows returned in 4ms
At line 6:
SELECT count(full_names)
FROM subs_data
WHERE full_names like "John%"
```

This shows the number of users named john in the Database which is 4 users.

Furthermore, all the tables created from the main database share a common identity which is the Subscription ID found in all the tables. The subscription ID is unique to each user and such is a **Primary Key.** The combination of two tables as shown using the Sql Join query shows that the subscription ID common between both tables serve as the **Foreign key** while a **compound key** is a combination of multiple columns that, together, uniquely identify each record in a table and this exist between the subscription ID and full_names in the same table.

In conclusion, the creation of a Netflix-like database and its organization into separate tables offers numerous benefits in terms of data management, security, and efficiency.This approach promotes better organization, enhances data security, and facilitates efficient database management practices. Overall, the project demonstrates the importance of thoughtful database design and management in optimizing data storage, access, and security.