Chinook Project – Data Analysis Final Project

Name: Ran Haim

Files & Execution Order:

Execution Order:

1. Python Pandas.ipynb
2. SQL/data_warehouse.sql
3. SQL/data_analysis.sql
4. Analysis.ipynb – Last task is in here at the end of the file.
5. albums_revenue_analysis.sql – Last task in SQL (my analysis)
    a. There is a word file called "Chinook Database Project SQL Analysis", this file contains the explanation of this SQL script.
    b. The same explanation provided in the word is also provided at the SQL file.

Dump file of database:

A dump file of my local database can be found at:

- SQL/chinook_dump_complete.sql

Python:

Each notebook is well documented with both comments and markdown.

Considering the information previously stated, I won't elaborate on each task over here. The notebook allows us to write explanations about each step we are taking throughout the analysis, so it's better to put the explanation in there.

API:

In the API I used the Bank Of Israel API which is free to use and require no signups or api key. The problem is that this API is slow, therefor in "Python Pandas.ipynb" it might take a bit to execute the entire file.

My Analysis - Assessing playlist revenue contribution at the country level

My analysis can found on "Analysis.ipynb" at the end of the page.

The analysis is focused on assessing playlist revenue contribution at the country level to find out which type of tracks the vast majority of the audience prefer.

In my analysis I looked for the countries which bring the most revenue and based the analysis on them. Those countries have dozens if not hundreds of customers unlike the other countries which have less than 10-20 customers.

In my analysis I found out that the audience will usually buy:

1. Music
2. 90's Music
3. TV Show's

Which lead the graph by a significant amount.

Therefor if there is a track which is uploaded and categorized to one of those types, the company should announce and advertise it so customers will be notified that a track has been added. They will most likely buy the track.

SQL:

Data Warehouse

In the data warehouse part of the SQL, I started by analyzing the data in all tables just so I can understand how can I "connect" between them for later JOIN operations and see which data is not needed for the warehouse.

Some data that I found out which is not needed:

- Last_update – data warehouse is more focused for analysis. It doesn't add to the analysis and hits the performance by adding a lot of cells to the table.
- Bytes – bytes isn't a data which can help the company improve upon tracks revenue, which playlist should we add or anything similar.
- Composer – it can be helpful in general, **BUT** there is way too much missing data over there, with a count function of all the null values I found out there are 900 tracks with no composer.
- Postal code, City, Address – Can't group this data to retrieve good information about customers.

Therefor I didn't extract the data to the warehouse.

Warehouse creation

The first step is to create the schema of the warehouse. You can do it by ethier using DBeaver UI or by executing the query **CREATE SCHEMA dwh;**

Then you can start add tables to the schema.

Before creation of each table I have looked over the required tables to merge. After I got a general idea on what to merge then I started to write the full script to see if it works.

At end of the process I would wrap the query with the following:

```
CREATE TABLE IF NOT EXISTS dwh.table_name AS ( QUERY )
```

- dwh.table_name – the table name is replaced by the required table name
- QUERY – just a placeholder in here but in the script I have provided a full query with comments. Each comment explain the next CTE, line or a difficult part to understand.

Warehouse Analysis SQL

Most / Least total tracks sold per playlist and avg tracks

At first we need to find the playlists with the most and least amount of tracks which can be done using the **count()** function on **trackid**.

Then we can use **order by** ordering them by the amount of tracks in each playlist and using **LIMIT 1** to get:

- The most amount by ordering the total tracks amount by descending order.
- The least amount by ordering the total tracks amount by ascending order.

With **UNION ALL** we can gather the information to only one table.

The average amount of tracks can be calculated in a separate CTE by using the **avg()** function on the total_tracks list.

Joining the two tables can be done using **left join** with **ON 1 = 1** which will join the tables on all rows.

Total tracks sold from each group

We need to **join** two tables, fact_invoiceline and dim_track in order to get the all of the sales of each track, achieved by using the **count()** function on trackid in fact_invoiceline.

Afterwards we need to group them with a **CASE** statement with the following conditions:

- Total tracks are 0
- Total tracks between 1 and 5

- Total tracks between 5 and 10
- Greater than 10

At the end of the grouping I make the table look better by ordering the conditions above using **CASE** statement giving them numbers 1 to 4 and then print it to the user.

Top and Bottom profits from countries

This question I divided to two queries.

The first query: get the top 5 and bottom 5 countries by revenue.

I did it by using the **sum()** function to sum the revenue of each country, and afterwards I selected the top 5 and bottom 5 countries using the same method of union as in Most / Least total tracks sold per playlist and avg tracks (with union all).

The second query: get the percentage revenue of each genre and rank them.

I copied the first part which shows the top 5 and bottom 5, and summed the line total with a grouping on billing country and genre name.

The above statement show only the sum revenue. In order to get the percentage I used the formula of (sum genre line / sum billingcountry) * 100.

At the end I display the percentage with a ranking using the **dense_rank()** function to get the ranks.

Average data of countries with total customers

At first we need to get the overall information:

- Total customers
- Total orders
- Revenue

Then we can start calculating the average data.

First, we start with the countries with more than one customer:

We can get the country and total customers for the first query, we only need to calculate the average data using (data / total customers).

Afterwards we do the same for the countries which have only one customer but we change the billingcountry to be 'Other' so group all those countries together. It does give the results need but to different rows therefore we will need to change it in the union later.

Union all at the end to get a big table with the results.

In the second query of the union we take the results of the other group avg data and run it through group by with both **sum** and **avg** functions to get the results needed for the question.

Employee Analysis

We can gather the data of seniority, id and full name of employee from the "dim_employee" table.

Then we have to gather the "stats" of each employee by merging the table with "dim_customer" and "fact_invoice" to get the following information:

- Which customers have been served by the employee.
- The year which it happen by looking at the invoice date.
- The total revenue for each year

After gathering all the information, we can get the growth percentage of each year by, getting the revenue from last year using **lag(total_revenue,1)** and use it to do

(current year revenue / last year revenue) * 100.

At the end with make the table presentation a little bit more nice with:

1. If there is a growth than represent it with greater than 100%
2. If there is a decline than represent it with minus and how much decline there is.
3. And place 0 where it's the beginning of income because there is nothing to compare data with.

My SQL Analysis

My SQL analysis can be found in the SQL folder at albums_revenue_analysis.

In python we have done an analysis about general information of albums but we never dive into how much revenue does each album make or how many copies sold from it.

In my analysis I wanted to see how each album is preforming in terms of revenue and copies sold, to see if there is an album which need some price adjustment or need's more help of advertising.

Also this analysis can bring some light on albums which doesn't perform at all and have 0 purchases.

So I started by looking at the "dim_tracks" and "fact_invoiceline" to see how can I achieve my analysis.

Both tables can be connected to one another by using the trackid column.

I **LEFT JOIN** the fact_invoiceline into dim_tracks to not lose any data about albums which haven't been purchased.

I calculated how many times an album have been sold and how much revenue each one of them generated using the **count** and **sum** functions.

Then I went on to calculate the percentage of sales on each album, to have better assessments of each album.

At the end I printed it all out and looked at the data to have a conclusion about it.

Conclusion

There are 42 albums which haven't been sold even once. I came up with two approaches to this:

1. The company can have some advertisements for those albums and try to get people to buy them. Or the company can sell a **bundle** of some of them in a low price so it will seem a good deal to buy it.
2. The company can decide to give those albums a deadline to meet to sell at least X number of copies, otherwise the albums will be removed from the platform. This approach can help to manage the database and get rid of unwanted albums