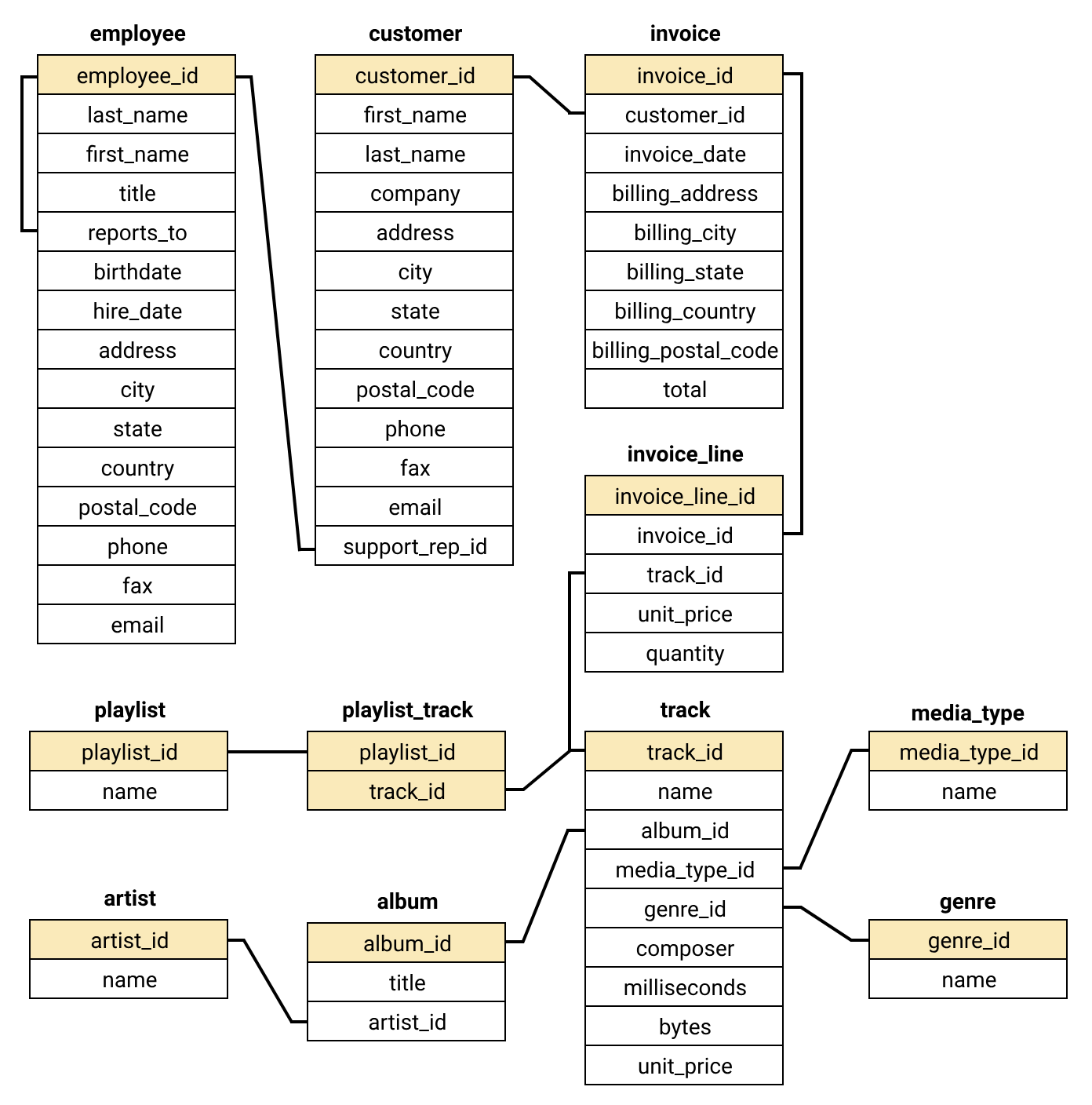
**Evaluación Técnica**

**SQL**

A continuación, se comparte el Diagrama de la base de datos de un cliente dedicado a la venta online de música. El esquema posee en total 11 tablas y cada tabla cuenta con el detalle de los campos que incluye y la relación que existe entre cada una de ellas. En cada tabla se han identificado con color las Claves Principales (o Primary Keys) de cada una de ellas.



1. Suponiendo que la columna invoice\_date tiene la fecha de venta de cada factura en formato YYYY-MM-DD, armar una query que permita saber las ventas de cada día (en una columna llamada sales\_day) y las ventas del día anterior (por ej en una columna llamada sales\_yesterday). Por ejemplo, para el día 10 de septiembre de 2020, la columna date debería tener el valor 2020-09-10, la columna sales\_day debería tener el total de las ventas del 10-Sept y la columna sales\_yesterday debería tener el total de las ventas del 09-Sept
2. Armar una query que encuentre las ventas totales por cada representante de ventas de la compañía.
3. Armar una query que nos permita saber quien es el artista que ha sido incluido en la mayor cantidad de playlists.
4. Armar una query que nos permita saber las ventas totales por género. Además, la tabla final debería tener una columna llamada ranking, que tenga el ranking de cada género musical según sus ventas.
5. Suponiendo que la columna invoice\_date tiene la fecha de venta de cada factura en formato YYYY-MM-DD, armar una query que permita saber las ventas de cada día (en una columna llamada sales\_day) y las ventas del día anterior (por ej en una columna llamada sales\_yesterday). Por ejemplo, para el día 10 de septiembre de 2020, la columna date debería tener el valor 2020-09-10, la columna sales\_day debería tener el total de las ventas del 10-Sept y la columna sales\_yesterday debería tener el total de las ventas del 09-Sept

**Forecasting**

El archivo peajes\_caba.csv tiene información diaria de la cantidad de autos que pasan por la estación Alberdi de peajes de la Ciudad de Buenos Aires.

1. Compartir insights o detalles valiosos descubiertos dentro de los datos.
2. Si tuviera que hacer un pronóstico para julio-diciembre 2019, ¿cómo podría armar un pronóstico que venga de una query? Es decir, cómo podría usar los datos pasados, sin pasar por un modelo, para generar un pronóstico
3. Generar un pronóstico para el período julio-diciembre 2019. Ahora sí, se puede usar cualquier modelo (sea de Machine Learning, series de tiempo, etc.) para generar este pronóstico

**Data Science / Datamining**

Construir un modelo en base a los datos del ***“train.csv”*** que permita clasificar correctamente los datos del ***“test.csv”***. Los datos pertenecen a un cliente que se encuentra interesado en anticipar que empleados se podrían llegar a ir de la compañía en base a datos históricos para tomar acciones de RRHH que le permitan mejorar la retención de sus empleados.

1. Entregar la predicción del modelo en formato CSV. Se puede utilizar como ejemplo de entrega el file ***“submission.csv”***.
2. Explicar el paso a paso y la metodología utilizada para construir el modelo.
3. Compartir insights o detalles valiosos descubiertos dentro de los datos.

**Algoritmos - Programación**

La idea de este ejercicio es realizar una heurística simple para conseguir una solución al Traveling Salesman Problem. Para hacer esto, habrá que trabajar sobre el archivo ***solver.py***. El mismo ya tiene una función, llamada solve\_it que lee el archivo input (tsp\_70\_1) y crea una lista, llamada points. Cada elemento de la lista points es un punto, definido por una coordenada X y una coordenada Y. Si se observa el archivo tsp\_70\_1 se puede ver que la primera fila indica la cantidad de puntos que quedarán en points y el resto de las filas muestran las coordenadas X y Y de cada uno.

Supongamos que arrancamos en el punto 0 (es decir, el primer elemento de la lista points) y tenemos que recorrer todos los otros puntos una sola vez, pero volviendo al final al punto inicial. ¿Cómo sabemos en qué orden visitar estos puntos? Una solución bien simple es la propuesta en la línea 26 del código -> visitamos primero el punto 0, luego el 1, luego el 2, y así sucesivamente hasta llegar al punto 69 (luego del punto 69, volveríamos a 0). Sin embargo, esto podría ser una solución ineficiente ya que podríamos estar haciendo potencialmente muchas “idas y vueltas”.

Por lo tanto, la idea del código es intentar mejorar esta solución, llegando a un recorrido que genere una distancia menor al actual. De esta forma, el output debería ser una lista, llamada solution, que marque el orden del recorrido (importante, el primer punto de esa lista siempre debería ser el cero).

En la solución ejemplo, solution es igual a [0, 1, 2, ….], indicando que se arranca por el punto 0, luego se va a 1, luego a 2, etc. Otra solución posible es [0, 10, 11, …]: es decir, luego de arrancar en 0 se visita al punto 10, luego al 11, y así.

IMPORTANTE: el objetivo de este ejercicio no es llegar a la MEJOR solución, sino a una solución que tenga sentido y mejore lo que exista. Para esto, les pedimos que la solución que generen siempre arranque por el punto 0 y que, además, no utilicen librerías de optimización para resolverlo, sino armar una solución que nazca de operaciones lógicas.