Librerias a instalar

pip install psycopg2-binary

pip install SQLAlchemy

Para ejecutar este código en Python con PostgreSQL, puedes utilizar la biblioteca psycopg2 que proporciona una interfaz de base de datos para PostgreSQL. Aquí te muestro cómo puedes hacerlo y de esta manera se cargarían datos prediseñados:

=============================================================================

import psycopg2

from psycopg2 import sql

# Conexión a la base de datos

“””

Asegúrate de reemplazar "tu\_host", "tu\_base\_de\_datos", "tu\_usuario", y "tu\_contraseña" con la información adecuada para tu base de datos PostgreSQL. Además, ten en cuenta que este código asume que ya tienes la biblioteca psycopg2 instalada. Puedes instalarla utilizando el siguiente comando:

”””

conn = psycopg2.connect(

host="tu\_host",

database="tu\_base\_de\_datos",

user="tu\_usuario",

password="tu\_contraseña"

)

# Creación de tablas

with conn.cursor() as cursor:

cursor.execute("""

CREATE TABLE customers(

customerid INT primary key,

name VARCHAR(50),

occupation VARCHAR(50),

email VARCHAR(50),

company VARCHAR(50),

phonenumber VARCHAR(20),

age INT

);

CREATE TABLE agents(

agentid INT primary key,

name VARCHAR(50)

);

CREATE TABLE calls(

callid INT primary key,

agentid INT,

customerid INT,

pickedup SMALLINT,

duration INT,

productsold SMALLINT

);

""")

conn.commit()

# Inserción de registros

with conn.cursor() as cursor:

cursor.execute("""

INSERT INTO calls VALUES (10000, 4,6, 1, 130, 1);

INSERT INTO calls VALUES (10001, 5,7, 1, 131, 0);

INSERT INTO calls VALUES (10002, 10,260, 0, 0, 0);

INSERT INTO calls VALUES (10003, 3,5, 1, 60, 1);

INSERT INTO calls VALUES (10004, 10,731, 1, 90, 0);

INSERT INTO calls VALUES (10005, 4,415, 0, 0, 0);

""")

conn.commit()

# Consulta analoga a OLAP

with conn.cursor() as cursor:

cursor.execute("""

SELECT a.name AS AgentName, cu.name AS CustomerName, x.duration

FROM

(

SELECT ca.agentid, ca.duration, max(customerid) AS cid

FROM

(

SELECT agentid, min(duration) as fastestcall

FROM calls

WHERE productsold = 1

GROUP BY agentid

) min

JOIN calls ca ON ca.agentid = min.agentid AND ca.duration = min.fastestcall

WHERE productsold = 1

GROUP BY ca.agentid, ca.duration

) x

JOIN agents a ON x.agentid = a.agentid

JOIN customers cu ON cu.customerid = x.cid

""")

result = cursor.fetchall()

for row in result:

print(row)

# Cierre de la conexión

conn.close()

===================================================================

Si deseas cargar datos desde archivos CSV en las mismas tablas en PostgreSQL, puedes utilizar la herramienta COPY de PostgreSQL o la biblioteca pandas en Python junto con psycopg2 para realizar la carga de datos.

Aquí tienes un ejemplo utilizando COPY directamente desde PostgreSQL:

import psycopg2

from psycopg2 import sql

# Conexión a la base de datos

conn = psycopg2.connect(

host="tu\_host",

database="tu\_base\_de\_datos",

user="tu\_usuario",

password="tu\_contraseña"

)

# Cargar datos desde archivos CSV usando COPY

with conn.cursor() as cursor:

# Ruta a tus archivos CSV

csv\_files = ["1.csv", "2.csv"]

for csv\_file in csv\_files:

table\_name = csv\_file.split(".")[0] # Tomar el nombre del archivo sin la extensión

copy\_query = sql.SQL("COPY {} FROM %s WITH CSV HEADER;").format(sql.Identifier(table\_name))

with open(csv\_file, 'r') as f:

cursor.copy\_expert(copy\_query, f)

conn.commit()

# Cierre de la conexión

conn.close()

Este código asume que los archivos CSV tienen una fila de encabezado (HEADER). Si no es así, elimina WITH CSV HEADER de la consulta COPY.

Si prefieres utilizar pandas, puedes hacer lo siguiente:

import pandas as pd

import psycopg2

# Conexión a la base de datos

conn = psycopg2.connect(

host="tu\_host",

database="tu\_base\_de\_datos",

user="tu\_usuario",

password="tu\_contraseña"

)

# Cargar datos desde archivos CSV usando pandas

csv\_files = ["1.csv", "2.csv"]

for csv\_file in csv\_files:

table\_name = csv\_file.split(".")[0] # Tomar el nombre del archivo sin la extensión

df = pd.read\_csv(csv\_file)

df.to\_sql(table\_name, conn, if\_exists='replace', index=False)

# Cierre de la conexión

conn.close()

En este segundo ejemplo, to\_sql de pandas se utiliza para cargar los datos en las tablas correspondientes. Ten en cuenta que if\_exists='replace' significa que la tabla se reemplazará si ya existe. Ajusta esta opción según tus necesidades.

=============================================================================

Para encontrar la información de conexión necesaria para PostgreSQL, puedes seguir estos pasos:

1. Localiza la información de conexión:
   * Host: Generalmente, esto es la dirección IP o el nombre del host donde se encuentra tu servidor PostgreSQL. Si estás trabajando localmente, puede ser "localhost" o "127.0.0.1".
   * Puerto: Por defecto, el puerto de PostgreSQL es 5432, pero podría ser diferente según la configuración de tu sistema.
   * Base de datos: Este es el nombre de la base de datos a la que deseas conectarte.
   * Usuario: Nombre de usuario con permisos para acceder a la base de datos.
   * Contraseña: La contraseña asociada al usuario.
2. Cómo obtener la información:
   * Local: Si PostgreSQL está instalado localmente en tu máquina, la información de conexión puede encontrarse en el archivo de configuración postgresql.conf o pg\_hba.conf dentro del directorio de instalación de PostgreSQL.
   * Servidor remoto: Si estás conectándote a un servidor remoto, es posible que necesites contactar al administrador del servidor o al proveedor de servicios para obtener la información de conexión.
3. Herramientas de administración:
   * Si estás utilizando una herramienta de administración de bases de datos como pgAdmin, puedes encontrar la información de conexión en la configuración de la conexión dentro de la interfaz de usuario.
4. Comando desde la línea de comandos:
   * Si PostgreSQL está instalado localmente, puedes ejecutar el siguiente comando en la línea de comandos para encontrar la información:

bashCopy code

psql -U tu\_usuario -h localhost -p 5432 -d tu\_base\_de\_datos

Sustituye "tu\_usuario", "localhost", "5432" y "tu\_base\_de\_datos" con tus propias configuraciones.

Asegúrate de tener los privilegios necesarios y que el servidor PostgreSQL esté en ejecución antes de intentar la conexión. La información de conexión también puede variar según la configuración específica de tu entorno y de la instalación de PostgreSQL.

El error indica que estás intentando ejecutar una consulta SQL que es específica de SQLite en lugar de PostgreSQL. La función to\_sql de pandas utiliza SQLAlchemy para interactuar con la base de datos, y por defecto, asume que estás utilizando SQLite. Para PostgreSQL, necesitas especificar un motor de SQLAlchemy compatible con PostgreSQL.

Modifica el código que carga los datos usando pandas para asegurarte de que estás utilizando el motor correcto para PostgreSQL. Aquí hay una versión corregida del código:

import pandas as pd

from sqlalchemy import create\_engine

# Conexión a la base de datos PostgreSQL

engine = create\_engine("postgresql://tu\_usuario:tu\_contraseña@tu\_host:5432/tu\_base\_de\_datos")

# Cargar datos desde archivos CSV usando pandas

csv\_files = ["1.csv", "2.csv"]

for csv\_file in csv\_files:

table\_name = csv\_file.split(".")[0] # Tomar el nombre del archivo sin la extensión

df = pd.read\_csv(csv\_file)

# Utilizar el motor de SQLAlchemy para escribir en PostgreSQL

df.to\_sql(table\_name, engine, if\_exists='replace', index=False)

# Cierra la conexión del motor de SQLAlchemy

engine.dispose()

Asegúrate de reemplazar "tu\_usuario", "tu\_contraseña", "tu\_host" y "tu\_base\_de\_datos" con tus propias configuraciones de conexión PostgreSQL.

Este código utiliza el objeto create\_engine de SQLAlchemy para conectarse a PostgreSQL y luego utiliza este motor en el método to\_sql. Esto debería resolver el problema que estás experimentando.

Claro, parece que estás tratando de cargar datos desde archivos CSV a PostgreSQL. Aquí tienes el código mejorado:

import pandas as pd

from sqlalchemy import create\_engine

# Conexión a la base de datos PostgreSQL

engine = create\_engine("postgresql://postgres:12345@localhost/postgres")

# Cargar datos desde archivos CSV usando pandas

csv\_files = ["agents.csv", "calls.csv", "customers.csv"]

for csv\_file in csv\_files:

table\_name = csv\_file.split(".")[0] # Tomar el nombre del archivo sin la extensión

df = pd.read\_csv(csv\_file)

# Utilizar el motor de SQLAlchemy para escribir en PostgreSQL

df.to\_sql(table\_name, engine, if\_exists='replace', index=False)

# Cierra la conexión del motor de SQLAlchemy

engine.dispose()

Este código utiliza el objeto create\_engine de SQLAlchemy para establecer la conexión con PostgreSQL. Asegúrate de reemplazar "12345" con la contraseña correcta de tu usuario PostgreSQL.

Para instalar SQLAlchemy, puedes utilizar el gestor de paquetes de Python, pip. Abre tu terminal o línea de comandos y ejecuta el siguiente comando:

pip install SQLAlchemy

Este comando descargará e instalará la biblioteca SQLAlchemy y sus dependencias. Asegúrate de tener una conexión a Internet estable para que pip pueda descargar los archivos necesarios desde el repositorio de Python Package Index (PyPI).

Después de la instalación, puedes verificar que todo esté correctamente configurado ejecutando el siguiente comando:

pip show SQLAlchemy

Este comando mostrará información sobre la versión instalada de SQLAlchemy y la ubicación del paquete en tu sistema.

Con SQLAlchemy instalado, podrás utilizarlo en tus scripts de Python para interactuar con bases de datos relacionales como PostgreSQL.

Si ya has cargado tus datos en PostgreSQL utilizando psycopg2 y ahora deseas utilizar SQLAlchemy para interactuar con la base de datos, puedes hacerlo de la siguiente manera. Asumiré que ya tienes tus tablas cargadas con los datos en PostgreSQL.

Luego, puedes modificar tu script para usar SQLAlchemy de la siguiente manera:

import pandas as pd

from sqlalchemy import create\_engine

# Conexión a la base de datos PostgreSQL

engine = create\_engine("postgresql://postgres:12345@localhost/postgres")

# Cargar datos desde archivos CSV usando pandas

csv\_files = ["agents.csv", "calls.csv", "customers.csv"]

for csv\_file in csv\_files:

table\_name = csv\_file.split(".")[0] # Tomar el nombre del archivo sin la extensión

df = pd.read\_csv(csv\_file)

# Utilizar el motor de SQLAlchemy para escribir en PostgreSQL

df.to\_sql(table\_name, engine, if\_exists='replace', index=False)

# Cierra la conexión del motor de SQLAlchemy

engine.dispose()

# Ejemplo de consulta utilizando SQLAlchemy

from sqlalchemy import text

with engine.connect() as connection:

query = text("SELECT \* FROM agents")

result = connection.execute(query)

for row in result:

print(row)

# Cierra la conexión del motor de SQLAlchemy

engine.dispose()

En este ejemplo, después de cargar los datos con pandas, he agregado un ejemplo simple de cómo puedes ejecutar una consulta utilizando SQLAlchemy. En este caso, la consulta selecciona todos los datos de la tabla "agents". Puedes ajustar la consulta según tus necesidades.

Recuerda cambiar la contraseña y otros detalles de conexión según tu configuración específica de PostgreSQL.

como hago una consulta para que me muestre todos los datos que cargue recien

Para realizar una consulta que te muestre todos los datos que has cargado recientemente en tus tablas, puedes hacer algo así en SQLAlchemy:

import pandas as pd

from sqlalchemy import create\_engine

# Conexión a la base de datos PostgreSQL

engine = create\_engine("postgresql://postgres:12345@localhost/postgres")

# Ejemplo de consulta para mostrar todos los datos en las tablas

with engine.connect() as connection:

# Puedes personalizar esta consulta según tus necesidades y las tablas que has creado

query = "SELECT \* FROM agents;"

# Ejecutar la consulta

result = connection.execute(query)

# Obtener y mostrar los resultados

rows = result.fetchall()

# Convertir los resultados a un DataFrame de pandas para visualización

df = pd.DataFrame(rows, columns=result.keys())

print(df)

# Cierra la conexión del motor de SQLAlchemy

engine.dispose()

Asegúrate de ajustar la consulta según las tablas que has creado y la estructura de tus datos. En este ejemplo, he usado la tabla "agents" como referencia, pero puedes personalizar la consulta para incluir todas las tablas que has creado. Si necesitas ayuda específica con la consulta y la estructura de tus datos, por favor, proporcióname más detalles.

El error indica que estás intentando ejecutar una cadena directamente como una consulta SQL, pero SQLAlchemy espera un objeto de tipo text o sqlalchemy.sql.expression.Executable. Para corregir esto, necesitas usar el objeto text de SQLAlchemy para envolver tu consulta SQL. Aquí está el código corregido:

import pandas as pd

from sqlalchemy import create\_engine, text

# Conexión a la base de datos PostgreSQL

engine = create\_engine("postgresql://postgres:12345@localhost/postgres")

# Ejemplo de consulta para mostrar todos los datos en las tablas

with engine.connect() as connection:

# Utilizar text de SQLAlchemy para envolver la consulta SQL

query = text("SELECT \* FROM agents;")

# Ejecutar la consulta

result = connection.execute(query)

# Obtener y mostrar los resultados

rows = result.fetchall()

# Convertir los resultados a un DataFrame de pandas para visualización

df = pd.DataFrame(rows, columns=result.keys())

print(df)

# Cierra la conexión del motor de SQLAlchemy

engine.dispose()

Aquí, he envuelto la cadena de la consulta SQL con text("SELECT \* FROM agents;") para asegurarme de que sea un objeto ejecutable para SQLAlchemy.

Momento 2 OLTP: Almacenar transacciones que se dieron en un día particular en la empresa de Call Center. Esto traducido a sentencias SQL -> quiero hacer un script en python para poder realizar esa consulta, pero quiero almacenarla en una nueva base de datos.

Como podria mosrtar por consola la respuesta de la consulta

Entiendo que deseas realizar una consulta SQL que recupere las transacciones ocurridas en un día específico en la empresa de Call Center y luego almacenar los resultados en una nueva base de datos. A continuación, te proporciono un ejemplo de cómo podrías hacer esto en Python utilizando psycopg2 para la conexión con PostgreSQL y pandas para la manipulación de datos y su visualización en la consola:

import pandas as pd

import psycopg2

from sqlalchemy import create\_engine

# Conexión a la base de datos original (donde se encuentran tus datos)

source\_conn = psycopg2.connect(

host="localhost",

database="tu\_base\_de\_datos\_original",

user="tu\_usuario",

password="tu\_contraseña"

)

# Fecha específica que deseas consultar (ajústala según tu necesidad)

fecha\_consulta = '2024-03-06'

# Consulta SQL para obtener transacciones de un día particular

sql\_query = f"""

SELECT \*

FROM calls

WHERE fecha\_columna = '{fecha\_consulta}';

"""

# Utilizando pandas para ejecutar la consulta y obtener un DataFrame

df = pd.read\_sql\_query(sql\_query, source\_conn)

# Muestra los resultados en la consola

print("Resultados de la consulta:")

print(df)

# Conexión a la nueva base de datos donde deseas almacenar los resultados

target\_engine = create\_engine("postgresql://tu\_usuario:tu\_contraseña@localhost/tu\_nueva\_base\_de\_datos")

# Almacenar los resultados en una nueva tabla en la base de datos de destino

df.to\_sql('nueva\_tabla', target\_engine, if\_exists='replace', index=False)

# Cierra las conexiones

source\_conn.close()

target\_engine.dispose()

Asegúrate de reemplazar "tu\_base\_de\_datos\_original", "tu\_usuario", "tu\_contraseña", "tu\_nueva\_base\_de\_datos" y "fecha\_columna" con los valores específicos de tu entorno.

Este script realiza la consulta SQL para obtener las transacciones de un día particular y muestra los resultados en la consola utilizando print(df). Además, almacena los resultados en una nueva base de datos y tabla usando to\_sql. Ten en cuenta que debes ajustar la fecha y la columna específicas en la consulta según la estructura real de tu base de datos.

Momento 3 OLAP: extraer información para cada agente de: duración de venta más rápida, a qué cliente fue y el nombre del agente. Todo esto para poder otorgar bonos a los agentes. me ayudarias con armando la consulta sql siguiendo la linea de programacion anterior no quiero usar psycopg2

Entiendo que deseas realizar una consulta SQL para el Momento 3 OLAP, que consiste en extraer información para cada agente sobre la duración de la venta más rápida, a qué cliente fue, y el nombre del agente. Además, prefieres utilizar pandas en lugar de psycopg2. A continuación, te proporciono un ejemplo de cómo podrías estructurar esta consulta SQL y manejar los resultados con pandas:

import pandas as pd

from sqlalchemy import create\_engine

# Conexión a la base de datos PostgreSQL

engine = create\_engine("postgresql://tu\_usuario:tu\_contraseña@localhost/tu\_base\_de\_datos")

# Consulta SQL para obtener la duración de la venta más rápida para cada agente

sql\_query = """

SELECT

a.name AS agent\_name,

c.name AS customer\_name,

c.duration AS fastest\_sale\_duration

FROM (

SELECT

agentid,

MIN(duration) AS fastest\_duration

FROM calls

WHERE productsold = 1

GROUP BY agentid

) AS min\_duration

JOIN calls c ON c.agentid = min\_duration.agentid AND c.duration = min\_duration.fastest\_duration

JOIN agents a ON a.agentid = c.agentid

WHERE c.productsold = 1;

"""

# Utilizando pandas para ejecutar la consulta y obtener un DataFrame

df = pd.read\_sql\_query(sql\_query, engine)

# Muestra los resultados en la consola

print("Resultados de la consulta OLAP:")

print(df)

# Cierra la conexión del motor de SQLAlchemy

engine.dispose()

Asegúrate de reemplazar "tu\_usuario", "tu\_contraseña" y "tu\_base\_de\_datos" con los valores específicos de tu entorno.

Este script ejecuta la consulta SQL para obtener la duración de la venta más rápida para cada agente, el nombre del cliente y el nombre del agente. Los resultados se almacenan en un DataFrame de pandas, que puedes utilizar para realizar análisis adicionales o aplicar lógica de bonificación según tus criterios.

El error que estás enfrentando se debe a que la columna c.name no existe en la tabla calls. Parece que estás intentando acceder a una columna que no está presente en la tabla utilizada en la consulta.

Supongo que estás intentando obtener el nombre del cliente (customer\_name). Ajustemos la consulta para que funcione correctamente:

import pandas as pd

from sqlalchemy import create\_engine

# Conexión a la base de datos PostgreSQL

engine = create\_engine("postgresql://tu\_usuario:tu\_contraseña@localhost/tu\_base\_de\_datos")

# Consulta SQL para obtener la duración de la venta más rápida para cada agente

sql\_query = """

SELECT

a.name AS agent\_name,

cu.name AS customer\_name,

c.duration AS fastest\_sale\_duration

FROM (

SELECT

agentid,

MIN(duration) AS fastest\_duration

FROM calls

WHERE productsold = 1

GROUP BY agentid

) AS min\_duration

JOIN calls c ON c.agentid = min\_duration.agentid AND c.duration = min\_duration.fastest\_duration

JOIN agents a ON a.agentid = c.agentid

JOIN customers cu ON cu.customerid = c.customerid

WHERE c.productsold = 1;

"""

# Utilizando pandas para ejecutar la consulta y obtener un DataFrame

df = pd.read\_sql\_query(sql\_query, engine)

# Muestra los resultados en la consola

print("Resultados de la consulta OLAP:")

print(df)

# Cierra la conexión del motor de SQLAlchemy

engine.dispose()

He agregado una nueva unión (JOIN) con la tabla customers para obtener el nombre del cliente (cu.name). Asegúrate de reemplazar "tu\_usuario", "tu\_contraseña" y "tu\_base\_de\_datos" con los valores específicos de tu entorno.

Con estos cambios, deberías obtener los resultados deseados sin el error de columna indefinida.

Momento 4 Interpretación de resultados en términos de tiempo en cada consulta. Luego de efectuar los 4 momentos como puedo interpretar

En el contexto del procesamiento analítico de datos (OLAP) y las consultas que has realizado en los cuatro momentos (OLTP, carga de datos; OLAP, consulta para otorgar bonos a los agentes; y momentos adicionales), la interpretación de resultados en términos de tiempo puede proporcionar información valiosa sobre el rendimiento de tu sistema y la eficiencia de las consultas. Aquí hay algunas pautas generales sobre cómo puedes interpretar los resultados:

Momento 1 (OLTP - Carga de datos):

Tiempo de ejecución: Evalúa cuánto tiempo lleva cargar los datos en el sistema. Un tiempo de carga rápido es esencial para asegurar que los datos estén disponibles para consultas rápidamente.

Momento 2 (OLTP - Consulta para bonos a los agentes):

Tiempo de ejecución: Evalúa cuánto tiempo lleva ejecutar la consulta para determinar los bonos de los agentes. Un tiempo más corto es preferible, ya que permite tomar decisiones rápidas y eficientes sobre los bonos.

Momento 3 (OLAP - Duración de la venta más rápida por agente):

Tiempo de ejecución: Evalúa cuánto tiempo lleva ejecutar la consulta OLAP para determinar la duración de la venta más rápida para cada agente. Este tiempo es crítico para proporcionar información analítica a los agentes de manera oportuna.

Momento 4 (Interpretación de resultados en términos de tiempo):

Comparación de tiempos: Compara los tiempos de ejecución entre los diferentes momentos y consultas. Identifica patrones de rendimiento y busca oportunidades de optimización.

Identificación de cuellos de botella: Analiza si hay consultas específicas o momentos que consumen más tiempo que otros. Identifica cualquier cuello de botella que pueda estar afectando el rendimiento general del sistema.

Ajustes y optimizaciones: Considera realizar ajustes en el diseño de la base de datos, índices o consultas para mejorar el rendimiento. Puedes utilizar herramientas de monitoreo y perfiles de consultas para identificar áreas de mejora.

Recuerda que la interpretación de resultados en términos de tiempo puede depender de varios factores, incluyendo la cantidad de datos, la complejidad de las consultas y la capacidad del sistema. Además, es recomendable realizar pruebas en un entorno de desarrollo antes de implementar cambios en un entorno de producción para evitar impactos no deseados.