Snowflake_Case

September 14, 2025

[1]:
```
%pip install --upgrade pip
%pip install "snowflake-connector-python>=3,<4" python-dotenv
```

```
Requirement already satisfied: pip in /opt/conda/lib/python3.12/site-packages
(24.2)
Collecting pip
  Downloading pip-25.2-py3-none-any.whl.metadata (4.7 kB)
Downloading pip-25.2-py3-none-any.whl (1.8 MB)
                          1.8/1.8 MB
1.2 MB/s eta 0:00:00a 0:00:010m
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 24.2
    Uninstalling pip-24.2:
      Successfully uninstalled pip-24.2
Successfully installed pip-25.2
Note: you may need to restart the kernel to use updated packages.
Collecting snowflake-connector-python<4,>=3
  Using cached snowflake_connector_python-3.17.3-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (74 kB)
Requirement already satisfied: python-dotenv in /opt/conda/lib/python3.12/site-
packages (1.0.1)
Collecting asn1crypto<2.0.0,>0.24.0 (from snowflake-connector-python<4,>=3)
  Using cached asn1crypto-1.5.1-py2.py3-none-any.whl.metadata (13 kB)
Collecting boto3>=1.24 (from snowflake-connector-python<4,>=3)
  Using cached boto3-1.40.30-py3-none-any.whl.metadata (6.7 kB)
Requirement already satisfied: botocore>=1.24 in /opt/conda/lib/python3.12/site-
packages (from snowflake-connector-python<4,>=3) (1.35.90)
Requirement already satisfied: cffi<2.0.0,>=1.9 in
/opt/conda/lib/python3.12/site-packages (from snowflake-connector-python<4,>=3)
(1.17.1)
Requirement already satisfied: cryptography>=3.1.0 in
/opt/conda/lib/python3.12/site-packages (from snowflake-connector-python<4,>=3)
(43.0.3)
Collecting pyOpenSSL<26.0.0,>=22.0.0 (from snowflake-connector-python<4,>=3)
  Downloading pyopenssl-25.2.0-py3-none-any.whl.metadata (17 kB)
Requirement already satisfied: pyjwt<3.0.0 in /opt/conda/lib/python3.12/site-
packages (from snowflake-connector-python<4,>=3) (2.9.0)
```

```
Requirement already satisfied: pytz in /opt/conda/lib/python3.12/site-packages
(from snowflake-connector-python<4,>=3) (2024.1)
Requirement already satisfied: requests<3.0.0 in /opt/conda/lib/python3.12/site-
packages (from snowflake-connector-python<4,>=3) (2.32.3)
Requirement already satisfied: packaging in /opt/conda/lib/python3.12/site-
packages (from snowflake-connector-python<4,>=3) (24.1)
Requirement already satisfied: charset_normalizer<4,>=2 in
/opt/conda/lib/python3.12/site-packages (from snowflake-connector-python<4,>=3)
(3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.12/site-
packages (from snowflake-connector-python<4,>=3) (3.10)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.12/site-packages (from snowflake-connector-python<4,>=3)
(2024.12.14)
Requirement already satisfied: typing_extensions<5,>=4.3 in
/opt/conda/lib/python3.12/site-packages (from snowflake-connector-python<4,>=3)
(4.12.2)
Requirement already satisfied: filelock<4,>=3.5 in
/opt/conda/lib/python3.12/site-packages (from snowflake-connector-python<4,>=3)
(3.16.1)
Requirement already satisfied: sortedcontainers>=2.4.0 in
/opt/conda/lib/python3.12/site-packages (from snowflake-connector-python<4,>=3)
(2.4.0)
Requirement already satisfied: platformdirs<5.0.0,>=2.6.0 in
/opt/conda/lib/python3.12/site-packages (from snowflake-connector-python<4,>=3)
(4.3.6)
Requirement already satisfied: tomlkit in /opt/conda/lib/python3.12/site-
packages (from snowflake-connector-python<4,>=3) (0.13.2)
Requirement already satisfied: pycparser in /opt/conda/lib/python3.12/site-
packages (from cffi<2.0.0,>=1.9->snowflake-connector-python<4,>=3) (2.22)
Collecting cryptography>=3.1.0 (from snowflake-connector-python<4,>=3)
  Downloading cryptography-45.0.7-cp311-abi3-manylinux_2_34_x86_64.whl.metadata
(5.7 kB)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/opt/conda/lib/python3.12/site-packages (from requests<3.0.0->snowflake-
connector-python<4,>=3) (2.2.3)
Collecting botocore>=1.24 (from snowflake-connector-python<4,>=3)
  Using cached botocore-1.40.30-py3-none-any.whl.metadata (5.7 kB)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in
/opt/conda/lib/python3.12/site-packages (from boto3>=1.24->snowflake-connector-
python<4,>=3) (1.0.1)
Collecting s3transfer<0.15.0,>=0.14.0 (from boto3>=1.24->snowflake-connector-
python<4,>=3)
  Using cached s3transfer-0.14.0-py3-none-any.whl.metadata (1.7 kB)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in
/opt/conda/lib/python3.12/site-packages (from botocore>=1.24->snowflake-
connector-python<4,>=3) (2.9.0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
```

```
packages (from python-dateutil<3.0.0,>=2.1->botocore>=1.24->snowflake-connector-
python<4,>=3) (1.16.0)
Using cached snowflake_connector_python-3.17.3-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.7 MB)
Using cached asn1crypto-1.5.1-py2.py3-none-any.whl (105 kB)
Downloading pyopenssl-25.2.0-py3-none-any.whl (57 kB)
Downloading cryptography-45.0.7-cp311-abi3-manylinux_2_34_x86_64.whl (4.5 MB)
                          4.5/4.5 MB
784.9 kB/s  0:00:05 eta 0:00:01
Using cached boto3-1.40.30-py3-none-any.whl (139 kB)
Using cached botocore-1.40.30-py3-none-any.whl (14.0 MB)
Using cached s3transfer-0.14.0-py3-none-any.whl (85 kB)
Installing collected packages: asn1crypto, cryptography, botocore, s3transfer,
pyOpenSSL, boto3, snowflake-connector-python
  Attempting uninstall: cryptography
    Found existing installation: cryptography 43.0.3
    Uninstalling cryptography-43.0.3:
      Successfully uninstalled cryptography-43.0.3
1/7 [cryptography]
  Attempting uninstall: botocore
1/7 [cryptography]
    Found existing installation: botocore 1.35.90
1/7 [cryptography]
    Uninstalling botocore-1.35.90:
1/7 [cryptography]
      Successfully uninstalled botocore-1.35.90
2/7 [botocore]
  Attempting uninstall: s3transfer[90m
2/7 [botocore]
    Found existing installation: s3transfer 0.10.4
2/7 [botocore]
    Uninstalling s3transfer-0.10.4:
3/7 [s3transfer]
      Successfully uninstalled s3transfer-0.10.4
3/7 [s3transfer]
                          7/7
[snowflake-connector-python]-connector-python]
```

```
ERROR: pip's dependency resolver does not currently take into
account all the packages that are installed. This behaviour is the source of the
following dependency conflicts.
awscli 1.36.31 requires botocore==1.35.90, but you have botocore 1.40.30 which
is incompatible.
awscli 1.36.31 requires s3transfer<0.11.0,>=0.10.0, but you have s3transfer
0.14.0 which is incompatible.
Successfully installed asn1crypto-1.5.1 boto3-1.40.30 botocore-1.40.30
cryptography-45.0.7 pyOpenSSL-25.2.0 s3transfer-0.14.0 snowflake-connector-
python-3.17.3
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
import snowflake.connector
```

[3]:
```
pip install python-dotenv
```

```
Requirement already satisfied: python-dotenv in /opt/conda/lib/python3.12/site-
packages (1.0.1)
Note: you may need to restart the kernel to use updated packages.
```

[ ]:
```
##Setting up Snowflake Credientials from .env

from dotenv import load_dotenv, find_dotenv
import os

# Find and load the nearest .env (project root)
load_dotenv(find_dotenv(), override=True)

def need(name):
    v = os.getenv(name)
    if not v:
        raise RuntimeError(f"Missing required env var: {name}")
    # remove accidental quotes/spaces
    return v.strip().strip('"').strip("'")

SNOWFLAKE_USER     = need("SNOWFLAKE_USER")
SNOWFLAKE_PASSWORD = need("SNOWFLAKE_PASSWORD")
SNOWFLAKE_ACCOUNT  = need("SNOWFLAKE_ACCOUNT")  # e.g., afaypkh-rjb48354
```

[ ]:
```
##Snowflake Connector

import snowflake.connector

conn = snowflake.connector.connect(
    user=SNOWFLAKE_USER,
    password=SNOWFLAKE_PASSWORD,
    account=SNOWFLAKE_ACCOUNT,
)
```

[6]:
```
cs = conn.cursor()
```

[ ]:
```
##Creating Warehouse
cs.execute("CREATE WAREHOUSE IF NOT EXISTS my_first_warehouse")
```

[ ]:
```
<snowflake.connector.cursor.SnowflakeCursor at 0x73cd1002fa70>
```

[ ]:
```
##Creating Testdb

cs.execute("CREATE DATABASE IF NOT EXISTS testdb")
```

[ ]:
```
<snowflake.connector.cursor.SnowflakeCursor at 0x73cd1002fa70>
```

[ ]:
```
##Creating Suupplier_Case info

from pathlib import Path
import re

# ---- 0) connect  ----
conn = snowflake.connector.connect(
    host="qbmhuza-bnb86629.snowflakecomputing.com",
    account="qbmhuza-bnb86629",
    user="second2",
    password="gyczeg6kaHqywownor",
    warehouse="COMPUTE_WH",
)
cs = conn.cursor()

# make sure the warehouse will wake itself
cs.execute("ALTER WAREHOUSE COMPUTE_WH SET AUTO_SUSPEND=60 AUTO_RESUME=TRUE")

# ---- 1) set DB/Schema (use your existing TESTDB) ----
DB, SCHEMA = "TESTDB", "PUBLIC"
cs.execute(f"CREATE DATABASE IF NOT EXISTS {DB}")
cs.execute(f"CREATE SCHEMA   IF NOT EXISTS {DB}.{SCHEMA}")
cs.execute(f"USE DATABASE {DB}")
cs.execute(f"USE SCHEMA {SCHEMA}")

# ---- 2) read and run your .pgsql file from the repo ----
sql_path = Path("Data/supplier_case.pgsql")
assert sql_path.exists(), f"Not found: {sql_path.resolve()}"
txt = sql_path.read_text(encoding="utf-8")
```

```
# tiny Postgres -> Snowflake cleanups
txt = "\n".join(l for l in txt.splitlines() if not l.strip().startswith("\\")) ␣
↪          # drop psql meta commands
txt = re.sub(r"\bNUMERIC\b", "NUMBER", txt, flags=re.I)                        ␣
↪          # NUMERIC -> NUMBER (safe)
txt = re.sub(r"\bsupplier_case\b", f"{DB}.{SCHEMA}.SUPPLIER_CASE", txt,␣
↪flags=re.I)       # fully-qualify table

# split on semicolons and execute
stmts = [s.strip() for s in re.split(r";\s*(?=\n|$)", txt) if s.strip()]
for s in stmts:
    cs.execute(s)

# ---- 3) visualize (still only cs.execute) ----
print("Rows:", cs.execute(f"SELECT COUNT(*) FROM {DB}.{SCHEMA}.SUPPLIER_CASE").
↪fetchone()[0])

print("\nSample rows:")
for r in cs.execute(f"""
    SELECT SupplierID, SupplierName, PhoneNumber, WebsiteURL,
           TRY_TO_DATE(ValidFrom) AS ValidFrom, TRY_TO_DATE(ValidTo) AS ValidTo
    FROM {DB}.{SCHEMA}.SUPPLIER_CASE
    ORDER BY SupplierID
    LIMIT 10
""").fetchall():
    print(r)

print("\nSchema:")
for r in cs.execute(f"DESCRIBE TABLE {DB}.{SCHEMA}.SUPPLIER_CASE").fetchall():
    print(r[0], r[1])
```

```
Rows: 13

Sample rows:
(1, 'A Datum Corporation', '(847) 555-0100', 'http://www.adatum.com', None,
None)
(2, 'Contoso, Ltd.', '(360) 555-0100', 'http://www.contoso.com', None, None)
(3, 'Consolidated Messenger', '(415) 555-0100',
'http://www.consolidatedmessenger.com', None, None)
(4, 'Fabrikam, Inc.', '(203) 555-0104', 'http://www.fabrikam.com', None, None)
(5, 'Graphic Design Institute', '(406) 555-0105',
'http://www.graphicdesigninstitute.com', None, None)
(6, 'Humongous Insurance', '(423) 555-0105',
'http://www.humongousinsurance.com', None, None)
(7, 'Litware, Inc.', '(209) 555-0108', 'http://www.litwareinc.com', None, None)
(8, 'Lucerne Publishing', '(423) 555-0103', 'http://www.lucernepublishing.com',
None, None)
(9, 'Nod Publishers', '(252) 555-0100', 'http://www.nodpublishers.com', None,
None)
(10, 'Northwind Electric Cars', '(201) 555-0105',
'http://www.northwindelectriccars.com', None, None)

Schema:
SUPPLIERID NUMBER(38,0)
SUPPLIERNAME VARCHAR(16777216)
SUPPLIERCATEGORYID NUMBER(38,0)
PRIMARYCONTACTPERSONID NUMBER(38,0)
ALTERNATECONTACTPERSONID NUMBER(38,0)
DELIVERYMETHODID NUMBER(38,0)
POSTALCITYID NUMBER(38,0)
SUPPLIERREFERENCE VARCHAR(16777216)
BANKACCOUNTNAME VARCHAR(16777216)
BANKACCOUNTBRANCH VARCHAR(16777216)
BANKACCOUNTCODE NUMBER(38,0)
BANKACCOUNTNUMBER NUMBER(38,0)
BANKINTERNATIONALCODE NUMBER(38,0)
PAYMENTDAYS NUMBER(38,0)
INTERNALCOMMENTS VARCHAR(16777216)
PHONENUMBER VARCHAR(16777216)
FAXNUMBER VARCHAR(16777216)
WEBSITEURL VARCHAR(16777216)
DELIVERYADDRESSLINE1 VARCHAR(16777216)
DELIVERYADDRESSLINE2 VARCHAR(16777216)
DELIVERYPOSTALCODE NUMBER(38,0)
DELIVERYLOCATION VARCHAR(16777216)
POSTALADDRESSLINE1 VARCHAR(16777216)
POSTALADDRESSLINE2 VARCHAR(16777216)
POSTALPOSTALCODE NUMBER(38,0)
LASTEDITEDBY NUMBER(38,0)
VALIDFROM VARCHAR(16777216)
VALIDTO VARCHAR(16777216)
```

[ ]:
```
##Setting actua db/schema and making table

cs.execute("USE DATABASE TESTDB")
cs.execute("USE SCHEMA PUBLIC")

cs.execute("""
CREATE OR REPLACE TABLE TESTDB.PUBLIC.SUPPLIER_CASE_CLEAN AS
SELECT
  CAST(SUPPLIERID              AS INT)       AS SUPPLIERID,
  SUPPLIERNAME                                AS SUPPLIERNAME,
  CAST(SUPPLIERCATEGORYID      AS INT)       AS SUPPLIERCATEGORYID,
  CAST(PRIMARYCONTACTPERSONID   AS INT)        AS PRIMARYCONTACTPERSONID,
  CAST(ALTERNATECONTACTPERSONID AS INT)        AS ALTERNATECONTACTPERSONID,
  CAST(DELIVERYMETHODID         AS INT)        AS DELIVERYMETHODID,
  CAST(POSTALCITYID             AS INT)        AS POSTALCITYID,
  SUPPLIERREFERENCE                              AS SUPPLIERREFERENCE,
  PHONENUMBER                                   AS PHONENUMBER,
  WEBSITEURL                                    AS WEBSITEURL,
  DELIVERYADDRESSLINE1                           AS DELIVERYADDRESSLINE1,
  CAST(DELIVERYPOSTALCODE       AS INT)        AS DELIVERYPOSTALCODE,
  POSTALADDRESSLINE1                             AS POSTALADDRESSLINE1,
  CAST(POSTALPOSTALCODE         AS INT)        AS POSTALPOSTALCODE,
  CAST(LASTEDITEDBY             AS INT)        AS LASTEDITEDBY,
  TRY_TO_DATE(VALIDFROM)                        AS VALIDFROM,
  TRY_TO_DATE(VALIDTO)                          AS VALIDTO
FROM TESTDB.PUBLIC.SUPPLIER_CASE;
""")
```

[ ]: `<snowflake.connector.cursor.SnowflakeCursor at 0x73cd1002ff80>`

[ ]:
```
##Testing that it works

print(cs.execute("SELECT COUNT(*) FROM TESTDB.PUBLIC.SUPPLIER_CASE_CLEAN").
↪fetchone()[0])
for r in cs.execute("""
  SELECT SUPPLIERID, SUPPLIERNAME, PHONENUMBER, WEBSITEURL, VALIDFROM, VALIDTO
  FROM TESTDB.PUBLIC.SUPPLIER_CASE_CLEAN
  ORDER BY SUPPLIERID
  LIMIT 10
""").fetchall():
    print(r)
```

```
13
(1, 'A Datum Corporation', '(847) 555-0100', 'http://www.adatum.com', None,
None)
(2, 'Contoso, Ltd.', '(360) 555-0100', 'http://www.contoso.com', None, None)
(3, 'Consolidated Messenger', '(415) 555-0100',
'http://www.consolidatedmessenger.com', None, None)
(4, 'Fabrikam, Inc.', '(203) 555-0104', 'http://www.fabrikam.com', None, None)
(5, 'Graphic Design Institute', '(406) 555-0105',
'http://www.graphicdesigninstitute.com', None, None)
(6, 'Humongous Insurance', '(423) 555-0105',
'http://www.humongousinsurance.com', None, None)
(7, 'Litware, Inc.', '(209) 555-0108', 'http://www.litwareinc.com', None, None)
(8, 'Lucerne Publishing', '(423) 555-0103', 'http://www.lucernepublishing.com',
None, None)
(9, 'Nod Publishers', '(252) 555-0100', 'http://www.nodpublishers.com', None,
None)
(10, 'Northwind Electric Cars', '(201) 555-0105',
'http://www.northwindelectriccars.com', None, None)
```

```
[ ]: ##Getting weather data from NOAA... Searching first for those specific tables␣
      ↪then putting into Snowflake Env
     import snowflake.connector

     def find_weather_db(cs):
         # Get all db names
         names = [r[1] for r in cs.execute("SHOW DATABASES").fetchall()]
         # Try exact names from the brief
         for cand in ("WEATHER__ENVIRONMENT", "WEATHER_ENVIRONMENT"):
             if cand in names:
                 return cand
         # Fuzzy fallback (handles custom names)
         for n in names:
             if "WEATHER" in n and "ENVIRONMENT" in n:
                 return n
         return None

     def print_table_sample(cs, fqtn, sample_rows=5):
         print(f"\n=== {fqtn} ===")
         cs.execute(f"SELECT * FROM {fqtn} LIMIT {sample_rows}")
         rows = cs.fetchall()
         cols = [d[0] for d in cs.description]
         print("Columns:", ", ".join(cols))
         for i, r in enumerate(rows, 1):
             print(f"{i:>2}: {r}")
         cs.execute(f"SELECT COUNT(*) FROM {fqtn}")
         print("Total rows:", cs.fetchone()[0])

     def print_cybersyn_weather_tables(conn, warehouse="COMPUTE_WH"):
         with conn.cursor() as cs:
             cs.execute(f"USE WAREHOUSE {warehouse}")

             db_name = find_weather_db(cs)
             if not db_name:
                 print("   Skipping Cybersyn weather: no WEATHER…ENVIRONMENT␣
      ↪database found in this account.")
                 have = [r[1] for r in cs.execute("SHOW DATABASES").fetchall()]
                 print("Databases you have:", have)
                 return

             cs.execute(f"USE DATABASE {db_name}")

             # Prefer CYBERSYN schema if present; otherwise fall back to PUBLIC
             schemas = {r[1] for r in cs.execute(f"SHOW SCHEMAS IN DATABASE␣
      ↪{db_name}").fetchall()}
```

```
        schema = "CYBERSYN" if "CYBERSYN" in schemas else "PUBLIC"
        cs.execute(f"USE SCHEMA {schema}")

        # If the exact NOAA table names differ, list what's there and pick two␣
↪NOAA* tables
        all_tables = [r[1] for r in cs.execute(f"SHOW TABLES IN SCHEMA␣
↪{db_name}.{schema}").fetchall()]
        candidates = [t for t in all_tables if t.startswith("NOAA_")]
        if not candidates:
            print(f"No NOAA_* tables in {db_name}.{schema}. Available tables:",␣
↪all_tables)
            return

        for t in candidates[:2]:
            print_table_sample(cs, f"{db_name}.{schema}.{t}", sample_rows=5)
```

[13]:
```
import snowflake.connector

conn = snowflake.connector.connect(
    user=SNOWFLAKE_USER,
    password=SNOWFLAKE_PASSWORD,
    account=SNOWFLAKE_ACCOUNT,
)
cs = conn.cursor()
```

[ ]:
```
#Pre-creating the PO Table and setting datatypes

import os, glob
from pathlib import Path

## Creating the PO_Table with Datatypes
cs.execute(f"USE DATABASE {DB}")
cs.execute(f"USE SCHEMA {SCHEMA}")
cs.execute(
"CREATE OR REPLACE TABLE PO_Data("
"purchaseorderid NUMBER(38,0), "
"supplierid NUMBER(38,0), "
"orderdate DATE, "
"deliverymethodid NUMBER(38,0), "
"contactpersonid NUMBER(38,0), "
"expecteddeliverydate DATE, "
"supplierreference VARCHAR, "
"isorderfinalized NUMBER(1,0), "
"comments VARCHAR, "
"internalcomments VARCHAR, "
"lasteditedby NUMBER(38,0), "
"purchaseorderlineid NUMBER(38,0), "
"stockitemid NUMBER(38,0), "
"orderedouters NUMBER(38,0), "
"description VARCHAR, "
"receivedouters NUMBER(38,0), "
"packagetypeid NUMBER(38,0), "
"expectedunitpriceperouter NUMBER(18,4), "
"lastreceiptdate DATE, "
"isorderlinefinalized NUMBER(1,0), "
"right_lasteditedby NUMBER(38,0), "
"right_lasteditedwhen TIMESTAMP_NTZ"
")")

# ---------- Resolve repo-relative data folder ----------
def find_monthly_po_dir() -> Path:
    """
    Locate the 'Data/Monthly PO Data' folder relative to the repository.
    Works from notebooks or scripts, on Windows/macOS/Linux.
    """
    candidates = [
        Path.cwd() / "Data" / "Monthly PO Data",
        Path.cwd() / "data" / "Monthly PO Data",
    ]

    # If running from a subfolder, search upward then rglob for the directory
    # 1) Walk up to (at most) 5 levels to find a '.git' folder (repo root)
    here = Path.cwd()
    ups = [here] + list(here.parents)[:5]
    repo_roots = [p for p in ups if (p / ".git").exists()]
    roots_to_search = repo_roots[:1] or [here]

    for root in roots_to_search:
        candidates.append(root / "Data" / "Monthly PO Data")
        candidates.append(root / "data" / "Monthly PO Data")
        # fallback: recursive search for the exact folder name
        for p in root.rglob("Monthly PO Data"):
            candidates.append(p)

    for p in candidates:
        if p.exists() and p.is_dir():
            # Must contain CSVs to be considered valid
            if any(p.glob("*.csv")):
                return p

    raise SystemExit("Could not find 'Data/Monthly PO Data' in this repo. "
                     "Make sure the data folder exists and contains .csv files.
↪")
```

```
local_dir_path = find_monthly_po_dir()
local_dir = str(local_dir_path)  # keep your existing code style
print("Using data folder:", local_dir)

# ---------- Stage + file format ----------


cs.execute("CREATE OR REPLACE STAGE po_data_stage")
cs.execute("""
CREATE OR REPLACE FILE FORMAT po_csv_ff
  TYPE=CSV
  FIELD_DELIMITER=','
  FIELD_OPTIONALLY_ENCLOSED_BY='"'
  SKIP_HEADER=1
  TRIM_SPACE=TRUE
  EMPTY_FIELD_AS_NULL=TRUE
  NULL_IF=('','NULL','null','00:00.0','0:00.0','00:00','0:00')
  DATE_FORMAT='AUTO'
  TIME_FORMAT='AUTO'
  TIMESTAMP_FORMAT='AUTO'
""")

# ---------- Local files to stage (repo-relative) ----------

pattern = os.path.join(local_dir, "*.csv")
files = glob.glob(pattern)
print("Matched CSVs:", len(files))
if not files:
    raise SystemExit(f"No CSVs matched at: {pattern}")

# ---------- PUT files into stage (auto-compress -> .gz) ----------
for filepath in files:
    base = os.path.basename(filepath)
    if ":" in base:   # skip Windows ADS like ':Zone.Identifier'
        continue
    abs_path = os.path.abspath(filepath).replace("\\", "/")   # ensure forward␣
 ↪slashes
    file_uri = "file:///" + abs_path.lstrip("/")              # exactly 3␣
 ↪slashes, no URL-encoding
    print("PUT ->", file_uri)
    cs.execute(f"PUT '{file_uri}' @po_data_stage AUTO_COMPRESS=TRUE␣
 ↪OVERWRITE=TRUE")


# --- sanity check what's in the stage ---
cs.execute("LIST @po_data_stage")
print("Staged objects (top 10):", cs.fetchall()[:10])
```

```
# --- load into the table (skipping $12 = lasteditedwhen) ---
cs.execute("""
COPY INTO PO_Data
  FROM (
    SELECT
      $1  ::NUMBER(38,0)  AS purchaseorderid,
      $2  ::NUMBER(38,0)  AS supplierid,
      TRY_TO_DATE($3)     AS orderdate,
      $4  ::NUMBER(38,0)  AS deliverymethodid,
      $5  ::NUMBER(38,0)  AS contactpersonid,
      TRY_TO_DATE($6)     AS expecteddeliverydate,
      $7                  AS supplierreference,
      $8  ::NUMBER(1,0)   AS isorderfinalized,
      $9                  AS comments,
      $10                 AS internalcomments,
      $11 ::NUMBER(38,0)  AS lasteditedby,
      /* skip $12 */
      $13 ::NUMBER(38,0)  AS purchaseorderlineid,
      $14 ::NUMBER(38,0)  AS stockitemid,
      $15 ::NUMBER(38,0)  AS orderedouters,
      $16                 AS description,
      $17 ::NUMBER(38,0)  AS receivedouters,
      $18 ::NUMBER(38,0)  AS packagetypeid,
      $19 ::NUMBER(18,4)  AS expectedunitpriceperouter,
      TRY_TO_DATE($20)    AS lastreceiptdate,
      $21 ::NUMBER(1,0)   AS isorderlinefinalized,
      $22 ::NUMBER(38,0)  AS right_lasteditedby,
      TRY_TO_TIMESTAMP_NTZ($23) AS right_lasteditedwhen
    FROM @po_data_stage (FILE_FORMAT => 'po_csv_ff')
  )
  ON_ERROR = ABORT_STATEMENT
""")
```

```
Using data folder: /home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data
Matched CSVs: 41
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2019-6.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2021-10.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2020-1.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2022-3.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2021-1.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
```

```
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2020-8.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2021-7.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2021-5.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2020-7.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2020-9.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2021-12.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2019-2.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2020-6.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2019-10.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2020-12.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2019-3.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2019-5.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2019-11.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2022-5.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2021-3.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2020-2.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2019-4.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2020-4.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2022-2.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2021-11.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2022-1.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2019-8.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2021-6.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2021-8.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
```

```
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2021-9.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2019-7.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2020-11.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2022-4.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2020-5.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2019-12.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2021-4.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2020-10.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2019-9.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2021-2.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2020-3.csv
PUT -> file:///home/jovyan/UCSD CLASSES/MGTA 464-
SQL/MGTA_464_Snowflake_Project/Data/Monthly PO Data/2019-1.csv
Staged objects (top 10): [('po_data_stage/2019-1.csv.gz', 6784,
'a15f2fa77d98a431c564917c46a8f54f', 'Sun, 14 Sep 2025 21:57:56 GMT'),
('po_data_stage/2019-10.csv.gz', 3328, '30fddf0b054c15abdf58d9008cfd507d', 'Sun,
14 Sep 2025 21:57:42 GMT'), ('po_data_stage/2019-11.csv.gz', 3056,
'cea00f90062fda98ac3b9cf93003f446', 'Sun, 14 Sep 2025 21:57:44 GMT'),
('po_data_stage/2019-12.csv.gz', 2992, 'f7f2bf307ed9bc4844dac53cbd45e6e6', 'Sun,
14 Sep 2025 21:57:53 GMT'), ('po_data_stage/2019-2.csv.gz', 2384,
'c63aedaf830d7dba4e7080fcd8b2fd8a', 'Sun, 14 Sep 2025 21:57:40 GMT'),
('po_data_stage/2019-3.csv.gz', 2848, 'dc2ccfb5cad11094f2139c3bd04bfb5e', 'Sun,
14 Sep 2025 21:57:43 GMT'), ('po_data_stage/2019-4.csv.gz', 3088,
'8c7471de208492d2a57559e77a1b148d', 'Sun, 14 Sep 2025 21:57:46 GMT'),
('po_data_stage/2019-5.csv.gz', 3200, '8c01985520ab970c77732bd8a2d88a9d', 'Sun,
14 Sep 2025 21:57:44 GMT'), ('po_data_stage/2019-6.csv.gz', 3024,
'f926b36f2e78f5ec00e1b688260f5d9e', 'Sun, 14 Sep 2025 21:57:31 GMT'),
('po_data_stage/2019-7.csv.gz', 3264, 'ec2a86136f0ff8bf13eeeac1ded84faa', 'Sun,
14 Sep 2025 21:57:51 GMT')]
```

[ ]:
```
<snowflake.connector.cursor.SnowflakeCursor at 0x73cd100688f0>
```

[ ]:
```
##Testing that data was created completely

cs.execute("SELECT COUNT(*) FROM PO_Data")
print("Row count:", cs.fetchone()[0])
print("Row count should equal 8367")
```

```
Row count: 8367
Row count should equal 8367
```

[16]:
```
cs.execute("SELECT * FROM PO_Data LIMIT 10")
for row in cs.fetchall():
    print(row)
```

```
(558, 4, datetime.date(2019, 12, 2), 7, 2, datetime.date(2019, 12, 22),
'293092', 1, None, None, 8, 2160, 77, 767, '"The Gu" red shirt XML tag t-shirt
(White) XXS', 767, 6, Decimal('84.0000'), datetime.date(2019, 12, 3), 1, 8,
None)
(558, 4, datetime.date(2019, 12, 2), 7, 2, datetime.date(2019, 12, 22),
'293092', 1, None, None, 8, 2161, 78, 981, '"The Gu" red shirt XML tag t-shirt
(White) XS', 981, 6, Decimal('84.0000'), datetime.date(2019, 12, 3), 1, 8, None)
(558, 4, datetime.date(2019, 12, 2), 7, 2, datetime.date(2019, 12, 22),
'293092', 1, None, None, 8, 2162, 80, 397, '"The Gu" red shirt XML tag t-shirt
(White) M', 397, 6, Decimal('84.0000'), datetime.date(2019, 12, 3), 1, 8, None)
(558, 4, datetime.date(2019, 12, 2), 7, 2, datetime.date(2019, 12, 22),
'293092', 1, None, None, 8, 2163, 86, 816, '"The Gu" red shirt XML tag t-shirt
(White) 5XL', 816, 6, Decimal('96.0000'), datetime.date(2019, 12, 3), 1, 8,
None)
(558, 4, datetime.date(2019, 12, 2), 7, 2, datetime.date(2019, 12, 22),
'293092', 1, None, None, 8, 2164, 95, 521, '"The Gu" red shirt XML tag t-shirt
(Black) XL', 521, 6, Decimal('90.0000'), datetime.date(2019, 12, 3), 1, 8, None)
(558, 4, datetime.date(2019, 12, 2), 7, 2, datetime.date(2019, 12, 22),
'293092', 1, None, None, 8, 2165, 98, 978, '"The Gu" red shirt XML tag t-shirt
(Black) 4XL', 978, 6, Decimal('96.0000'), datetime.date(2019, 12, 3), 1, 8,
None)
(559, 7, datetime.date(2019, 12, 2), 2, 2, datetime.date(2019, 12, 22),
'BC0280982', 1, None, None, 8, 2166, 193, 344, 'Black and orange glass with care
despatch tape 48mmx75m', 344, 7, Decimal('38.4000'), datetime.date(2019, 12, 3),
1, 8, None)
(559, 7, datetime.date(2019, 12, 2), 2, 2, datetime.date(2019, 12, 22),
'BC0280982', 1, None, None, 8, 2167, 204, 467, 'Tape dispenser (Red)', 467, 7,
Decimal('170.0000'), datetime.date(2019, 12, 3), 1, 8, None)
(560, 4, datetime.date(2019, 12, 3), 7, 2, datetime.date(2019, 12, 23),
'293092', 1, None, None, 17, 2168, 77, 769, '"The Gu" red shirt XML tag t-shirt
(White) XXS', 769, 6, Decimal('84.0000'), datetime.date(2019, 12, 4), 1, 17,
None)
(560, 4, datetime.date(2019, 12, 3), 7, 2, datetime.date(2019, 12, 23),
'293092', 1, None, None, 17, 2169, 78, 979, '"The Gu" red shirt XML tag t-shirt
(White) XS', 979, 6, Decimal('84.0000'), datetime.date(2019, 12, 4), 1, 17,
None)
```

1) PO totals (POAmount) and a tidy PO header table

[17]:
```
# One row per purchase order with the required total
cs.execute("""
CREATE OR REPLACE VIEW PO_Header AS
SELECT
  purchaseorderid,
  MIN(orderdate)                  AS orderdate,
  MIN(supplierid)                 AS supplierid,
  SUM(receivedouters * expectedunitpriceperouter) AS POAmount
FROM PO_Data
GROUP BY purchaseorderid;
""")
```

```
[17]: <snowflake.connector.cursor.SnowflakeCursor at 0x73cd100688f0>
```

```
[ ]: ##Getting supplier transaction data via XML extraction

from pathlib import Path
import shutil

# Path to the XML in your repo (adjust this if your folder name differs)
repo_xml = Path("Data") / "Supplier Transactions XML.xml"

# Make a simple, safe upload path that definitely exists *inside the container*
safe_dir = Path.home() / "sf_uploads"
safe_dir.mkdir(parents=True, exist_ok=True)

safe = safe_dir / "supplier_transactions.xml"   # normalize name
shutil.copy2(repo_xml, safe)                      # copy into place

uri = safe.as_uri()  # e.g., file:///home/jovyan/sf_uploads/
 ↪supplier_transactions.xml
print("Local file URI:", uri)

# Make sure the stage exists and is an *internal* stage
cs.execute("CREATE STAGE IF NOT EXISTS invoice_xml_stage")

# Do the PUT
cs.execute(f"PUT '{uri}' @invoice_xml_stage OVERWRITE=TRUE AUTO_COMPRESS=TRUE")

# Confirm
print(cs.execute("LIST @invoice_xml_stage").fetchall())
```

```
Local file URI: file:///home/jovyan/sf_uploads/supplier_transactions.xml
[('invoice_xml_stage/supplier_transactions.xml.gz', 72528,
'15c0f8f4276bf24ac457fb4e00e4107a', 'Sun, 14 Sep 2025 22:40:45 GMT')]
```

```
[ ]: # 1) Recreate the XML file format WITH strip_outer_element
cs.execute("""
CREATE OR REPLACE FILE FORMAT xml_ff
  TYPE=XML
  STRIP_OUTER_ELEMENT=TRUE
""")
print("xml_ff (with STRIP_OUTER_ELEMENT=TRUE) ready.")
```

```
xml_ff (with STRIP_OUTER_ELEMENT=TRUE) ready.
```

[52]:
```
# 2) Reload the XML so each <row> is its own table row
cs.execute("CREATE OR REPLACE TABLE INVOICE_RAW (doc VARIANT)")
cs.execute("""
COPY INTO INVOICE_RAW
FROM @invoice_xml_stage
FILE_FORMAT = xml_ff
ON_ERROR = ABORT_STATEMENT
""")
print("INVOICE_RAW rows (should be ~2438):", cs.execute("SELECT COUNT(*) FROM␣
  ↪INVOICE_RAW").fetchone()[0])
```

```
INVOICE_RAW rows (should be ~2438): 2438
```

[ ]:
```
##Taking Invoice_raw staging table and putting it into SUPPLIER_INVOICES in␣
  ↪actual Schema

cs.execute("""
CREATE OR REPLACE TABLE SUPPLIER_INVOICES AS
SELECT
  TRY_TO_NUMBER(XMLGET(doc,'SupplierTransactionID'):"$"::STRING)      AS␣
  ↪SupplierTransactionID,
  TRY_TO_NUMBER(XMLGET(doc,'SupplierID'):"$"::STRING)                  AS␣
  ↪SupplierID,
  TRY_TO_NUMBER(XMLGET(doc,'TransactionTypeID'):"$"::STRING)           AS␣
  ↪TransactionTypeID,
  NULLIF(XMLGET(doc,'PurchaseOrderID'):"$"::STRING,'')::NUMBER         AS␣
  ↪PurchaseOrderID,
  NULLIF(XMLGET(doc,'SupplierInvoiceNumber'):"$"::STRING,'')           AS␣
  ↪SupplierInvoiceNumber,
  TRY_TO_DATE(XMLGET(doc,'TransactionDate'):"$"::STRING)               AS␣
  ↪TransactionDate,   -- safer
  TRY_TO_DECIMAL(XMLGET(doc,'AmountExcludingTax'):"$"::STRING,18,2)    AS␣
  ↪AmountExcludingTax,
  TRY_TO_DECIMAL(XMLGET(doc,'TaxAmount'):"$"::STRING,18,2)             AS␣
  ↪TaxAmount,
  TRY_TO_DECIMAL(XMLGET(doc,'TransactionAmount'):"$"::STRING,18,2)     AS␣
  ↪TransactionAmount,
  TRY_TO_DECIMAL(XMLGET(doc,'OutstandingBalance'):"$"::STRING,18,2)    AS␣
  ↪OutstandingBalance,
  TRY_TO_DATE(XMLGET(doc,'FinalizationDate'):"$"::STRING)              AS␣
  ↪FinalizationDate,
  TRY_TO_BOOLEAN(XMLGET(doc,'IsFinalized'):"$"::STRING)                AS␣
  ↪IsFinalized
FROM INVOICE_RAW
""")

print("Rows in SUPPLIER_INVOICES:", cs.execute("SELECT COUNT(*) FROM␣
  ↪SUPPLIER_INVOICES").fetchone()[0])
print(cs.execute("SELECT * FROM SUPPLIER_INVOICES LIMIT 5").fetchall())
```

```
Rows in SUPPLIER_INVOICES: 2438
[(134, 2, 5, 1, '7290', datetime.date(2019, 1, 2), Decimal('313.50'),
Decimal('47.03'), Decimal('360.53'), Decimal('0.00'), datetime.date(2019, 1, 7),
True), (169, 4, 5, 2, '3898', datetime.date(2019, 1, 2), Decimal('21732.00'),
Decimal('3259.80'), Decimal('24991.80'), Decimal('0.00'), datetime.date(2019, 1,
7), True), (186, 5, 5, 3, '616', datetime.date(2019, 1, 2), Decimal('2740.50'),
Decimal('411.11'), Decimal('3151.61'), Decimal('0.00'), datetime.date(2019, 1,
7), True), (215, 7, 5, 4, '3869', datetime.date(2019, 1, 2),
Decimal('42481.20'), Decimal('6372.19'), Decimal('48853.39'), Decimal('0.00'),
datetime.date(2019, 1, 7), True), (224, 10, 5, 5, '4697', datetime.date(2019, 1,
2), Decimal('35067.50'), Decimal('5260.14'), Decimal('40327.64'),
Decimal('0.00'), datetime.date(2019, 1, 7), True)]
```

[55]:
```
# 1) Find a likely PO table and the columns we need
#    (searches current database across common schemas; tweak the schema list if␣
  ↪needed)
schemas_to_check = ["STAGE_AND_RAW", "PUBLIC", "RAW", "DATA", "MARTS"]
like_filters = ["%PO%", "%PURCHASE%"]

# helper: run a query and return rows
def q(sql):
    return cs.execute(sql).fetchall()

# find candidate tables
candidates = []
for sch in schemas_to_check:
    rows = q(f"""
        SELECT table_schema, table_name
        FROM information_schema.tables
        WHERE table_type='BASE TABLE'
          AND table_schema = '{sch}'
          AND (UPPER(table_name) LIKE '{like_filters[0]}' OR UPPER(table_name)␣
  ↪LIKE '{like_filters[1]}')
    """)
    candidates.extend(rows)

print("Candidate PO tables:", candidates)
```

```
# choose the best candidate that has PurchaseOrderID and a price*qty pair
chosen = None
qty_col = None
price_col = None

pairs = [
    ("RECEIVEDOUTERS", "EXPECTEDUNITPRICEPEROUTER"),
    ("QUANTITY", "UNITPRICE")
]

for sch, tbl in candidates:
    cols = {r[0] for r in q(f"""
        SELECT UPPER(column_name)
        FROM information_schema.columns
        WHERE table_schema = '{sch}' AND table_name = '{tbl}'
    """)}
    if "PURCHASEORDERID" in cols:
        for qcol, pcol in pairs:
            if qcol in cols and pcol in cols:
                chosen = (sch, tbl)
                qty_col, price_col = qcol, pcol
                break
    if chosen:
        break

if not chosen:
    raise RuntimeError("Couldn't auto-detect a PO detail table with the needed␣
 ↪columns. "
                       "If you already know it, set␣
 ↪`chosen=('YOUR_SCHEMA','YOUR_TABLE')` "
                       "and `qty_col, price_col` accordingly and re-run.")

print("Chosen PO table:", chosen, "with columns:", qty_col, "x", price_col)

# for convenience
PO_SCHEMA, PO_TABLE = chosen
```

```
Candidate PO tables: [('PUBLIC', 'PO_DATA')]
Chosen PO table: ('PUBLIC', 'PO_DATA') with columns: RECEIVEDOUTERS x
EXPECTEDUNITPRICEPEROUTER
```

[ ]:
```
##Creating PO_totals table

PO_SCHEMA, PO_TABLE = "PUBLIC", "PO_DATA"   # from your auto-detect result
qty_col, price_col = "RECEIVEDOUTERS", "EXPECTEDUNITPRICEPEROUTER"

cs.execute(f"""
CREATE OR REPLACE TABLE {PO_SCHEMA}.PO_TOTALS AS
SELECT
  PurchaseOrderID,
  SUM(COALESCE({qty_col},0) * COALESCE({price_col},0)) AS POAmount
FROM {PO_SCHEMA}.{PO_TABLE}
GROUP BY 1
""")
print("po_totals sample:", cs.execute(f"SELECT * FROM {PO_SCHEMA}.PO_TOTALS␣
 ↪LIMIT 5").fetchall())
```

```
po_totals sample: [(562, Decimal('402192.0000')), (576, Decimal('407994.0000')),
(814, Decimal('186559.6000')), (577, Decimal('98336.4000')), (581,
Decimal('413298.0000'))]
```

[ ]:
```
##Creating the view where we merged Purchase Orders and Invoices are:␣
 ↪PURCHASE_ORDERS_AND_INVOICES as TABLE.


# Make sure we're in the right place (safe to re-run)
cs.execute("USE SCHEMA PUBLIC")

# Try MATERIALIZED VIEW first; fall back to TABLE if not supported
built = "MATERIALIZED VIEW"
try:
    cs.execute("""
    CREATE OR REPLACE MATERIALIZED VIEW PUBLIC.PURCHASE_ORDERS_AND_INVOICES AS
    SELECT
      si.SupplierTransactionID,
      si.SupplierID,
      si.PurchaseOrderID,
      si.TransactionDate                 AS InvoiceDate,
      si.AmountExcludingTax,
      pt.POAmount,
      (si.AmountExcludingTax - pt.POAmount) AS invoiced_vs_quoted
    FROM PUBLIC.SUPPLIER_INVOICES si
    JOIN PUBLIC.PO_TOTALS pt USING (PurchaseOrderID)
    """)
except Exception:
    built = "TABLE"
    cs.execute("""
    CREATE OR REPLACE TABLE PUBLIC.PURCHASE_ORDERS_AND_INVOICES AS
    SELECT
      si.SupplierTransactionID,
      si.SupplierID,
      si.PurchaseOrderID,
      si.TransactionDate                 AS InvoiceDate,
      si.AmountExcludingTax,
      pt.POAmount,
      (si.AmountExcludingTax - pt.POAmount) AS invoiced_vs_quoted
    FROM PUBLIC.SUPPLIER_INVOICES si
    JOIN PUBLIC.PO_TOTALS pt USING (PurchaseOrderID)
    """)

print(f"Built PUBLIC.PURCHASE_ORDERS_AND_INVOICES as {built}.")
print("Row count:", cs.execute("SELECT COUNT(*) FROM PUBLIC.
  ↪PURCHASE_ORDERS_AND_INVOICES").fetchone()[0])
print(cs.execute("SELECT * FROM PUBLIC.PURCHASE_ORDERS_AND_INVOICES LIMIT 10").
  ↪fetchall())
```

```
Built PUBLIC.PURCHASE_ORDERS_AND_INVOICES as TABLE.
Row count: 2072
[(83481, 4, 562, datetime.date(2019, 12, 5), Decimal('402192.00'),
Decimal('402192.0000'), Decimal('0.0000')), (85554, 4, 576, datetime.date(2019,
12, 16), Decimal('407994.00'), Decimal('407994.0000'), Decimal('0.0000')),
(121887, 7, 814, datetime.date(2020, 5, 7), Decimal('186559.60'),
Decimal('186559.6000'), Decimal('0.0000')), (85557, 7, 577, datetime.date(2019,
12, 16), Decimal('98336.40'), Decimal('98336.4000'), Decimal('0.0000')), (86182,
4, 581, datetime.date(2019, 12, 18), Decimal('413298.00'),
Decimal('413298.0000'), Decimal('0.0000')), (86385, 4, 583, datetime.date(2019,
12, 19), Decimal('413682.00'), Decimal('413682.0000'), Decimal('0.0000')),
(87795, 7, 594, datetime.date(2019, 12, 25), Decimal('102054.80'),
Decimal('102054.8000'), Decimal('0.0000')), (123490, 4, 821, datetime.date(2020,
5, 13), Decimal('569754.00'), Decimal('569754.0000'), Decimal('0.0000')),
(123675, 4, 823, datetime.date(2020, 5, 14), Decimal('569790.00'),
Decimal('569790.0000'), Decimal('0.0000')), (124573, 7, 828, datetime.date(2020,
5, 16), Decimal('189055.20'), Decimal('189055.2000'), Decimal('0.0000'))]
```

[ ]:
```
##Testing connections to Postgres to get our Vendor Info

# If host is macOS/Windows this works out of the box:
PG_HOST = "host.docker.internal"

# If you're on Linux and host.docker.internal doesn't resolve, uncomment this␣
  ↪instead:
# import subprocess
# PG_HOST = subprocess.check_output("ip route | awk '/default/ {print $3}'",␣
  ↪shell=True).decode().strip()

PG_PORT = 8765              # <-- from your VS Code connection
PG_DB   = "rsm-docker"      # <-- from your VS Code connection
PG_USER = "jovyan"          # <-- from your VS Code connection
PG_PWD  = "postgres"        # <-- from your VS Code connection
```

```
[ ]: ##Connecting to postgres, and copying the table to create supplier_Case table␣
     ↪(One time)

     import psycopg2

     try:
         conn = psycopg2.connect(host=PG_HOST, port=PG_PORT, dbname=PG_DB,␣
     ↪user=PG_USER, password=PG_PWD)
         conn.autocommit = True
         cur = conn.cursor()
         print("  Connected to Postgres at", PG_HOST, PG_PORT)

         # Find the table (unquoted identifiers in Postgres become lowercase)
         cur.execute("""
           SELECT table_schema, table_name
           FROM information_schema.tables
           WHERE table_name ILIKE 'supplier_case'
         """)
         print("Tables named supplier_case:", cur.fetchall())

         # Show columns (you'll likely see supplierid, suppliername,␣
     ↪postalpostalcode)
         cur.execute("""
           SELECT column_name
           FROM information_schema.columns
           WHERE table_name ILIKE 'supplier_case'
           ORDER BY ordinal_position
         """)
         cols = [r[0] for r in cur.fetchall()]
         print("Columns:", cols)

         # Quick row count
         cur.execute('SELECT COUNT(*) FROM public.supplier_case')
         print("Row count:", cur.fetchone()[0])

         cur.close()
         conn.close()
     except Exception as e:
         print("  Postgres connection failed:", e)
```

```
 Connected to Postgres at host.docker.internal 8765
Tables named supplier_case: [('public', 'supplier_case')]
Columns: ['supplierid', 'suppliername', 'suppliercategoryid',
'primarycontactpersonid', 'alternatecontactpersonid', 'deliverymethodid',
'postalcityid', 'supplierreference', 'bankaccountname', 'bankaccountbranch',
'bankaccountcode', 'bankaccountnumber', 'bankinternationalcode', 'paymentdays',
'internalcomments', 'phonenumber', 'faxnumber', 'websiteurl',
'deliveryaddressline1', 'deliveryaddressline2', 'deliverypostalcode',
'deliverylocation', 'postaladdressline1', 'postaladdressline2',
'postalpostalcode', 'lasteditedby', 'validfrom', 'validto']
Row count: 13
```

```
[ ]: ##Exporting the previous step as a csv so that we can now refer to it within␣
      ↪the repo
     import psycopg2
     from pathlib import Path

     conn = psycopg2.connect(host=PG_HOST, port=PG_PORT, dbname=PG_DB, user=PG_USER,␣
      ↪password=PG_PWD)
     conn.autocommit = True
     cur = conn.cursor()

     safe_dir = Path.home() / "sf_uploads"
     safe_dir.mkdir(parents=True, exist_ok=True)
     export_csv = safe_dir / "supplier_case_export.csv"

     # Use lowercase column names per Postgres rules
     with open(export_csv, "w", encoding="utf-8", newline="") as f:
         cur.copy_expert(
             """
             COPY (
                 SELECT supplierid, suppliername, postalpostalcode
                 FROM public.supplier_case
             ) TO STDOUT WITH CSV HEADER
             """,
             f
         )

     cur.close()
     conn.close()
     print("Exported CSV:", export_csv)
```

```
Exported CSV: /home/jovyan/sf_uploads/supplier_case_export.csv
```

```
[ ]: ##Actually pushing it to snowflake

     # Use your existing Snowflake cursor `cs`
     uri = export_csv.resolve().as_uri()

     cs.execute("CREATE STAGE IF NOT EXISTS PUBLIC.refdata_stage")
     cs.execute(f"PUT '{uri}' @PUBLIC.refdata_stage OVERWRITE=TRUE␣
      ↪AUTO_COMPRESS=TRUE")
     print("Staged:", cs.execute("LIST @PUBLIC.refdata_stage").fetchall())

     # Keep STRING types to preserve leading zeros in ZIPs
```

```
cs.execute("""
CREATE OR REPLACE TABLE PUBLIC.SUPPLIER_CASE (
  SupplierID STRING,
  SupplierName STRING,
  PostalPostalCode STRING
)
""")

cs.execute("""
COPY INTO PUBLIC.SUPPLIER_CASE
FROM (SELECT $1, $2, $3 FROM @PUBLIC.refdata_stage)
FILE_FORMAT=(TYPE=CSV FIELD_OPTIONALLY_ENCLOSED_BY='"' SKIP_HEADER=1)
ON_ERROR=ABORT_STATEMENT
""")

print("Rows in PUBLIC.SUPPLIER_CASE:",
      cs.execute("SELECT COUNT(*) FROM PUBLIC.SUPPLIER_CASE").fetchone()[0])
print(cs.execute("SELECT * FROM PUBLIC.SUPPLIER_CASE LIMIT 5").fetchall())
```

```
Staged: [('refdata_stage/supplier_case_export.csv.gz', 320,
'afd5053a5689ecf4cc02688db7c733da', 'Sun, 14 Sep 2025 23:46:14 GMT')]
Rows in PUBLIC.SUPPLIER_CASE: 13
[('1', 'A Datum Corporation', '22202'), ('2', 'Contoso, Ltd.', '80125'), ('3',
'Consolidated Messenger', '60523'), ('4', 'Fabrikam, Inc.', '95642'), ('5',
'Graphic Design Institute', '80125')]
```

[ ]:
```
##Intermittent check step to make sure that all relevant tables are pulling␣
 ↪correctly

print(cs.execute("SELECT COUNT(*) FROM PUBLIC.SUPPLIER_INVOICES").fetchone()[0])
print(cs.execute("SELECT COUNT(*) FROM PUBLIC.PO_TOTALS").fetchone()[0])
print(cs.execute("SELECT COUNT(*) FROM PUBLIC.PURCHASE_ORDERS_AND_INVOICES").
 ↪fetchone()[0])
print(cs.execute("SELECT COUNT(*) FROM PUBLIC.SUPPLIER_CASE").fetchone()[0])

# Join suppliers onto the PO/invoice view (zip normalized)
cs.execute("""
CREATE OR REPLACE VIEW PUBLIC.JOINED_WITH_SUPPLIERS AS
SELECT
  p.*,
  s.SupplierName,
  LPAD(REGEXP_REPLACE(s.PostalPostalCode,'\\D',''),5,'0') AS PostalPostalCode
FROM PUBLIC.PURCHASE_ORDERS_AND_INVOICES p
LEFT JOIN PUBLIC.SUPPLIER_CASE s
  ON TO_VARCHAR(p.SupplierID) = TO_VARCHAR(s.SupplierID)
""")
```

```
print(cs.execute("SELECT * FROM PUBLIC.JOINED_WITH_SUPPLIERS LIMIT 10").
 ↪fetchall())
```

```
2438
2074
2072
13
[(83481, 4, 562, datetime.date(2019, 12, 5), Decimal('402192.00'),
Decimal('402192.0000'), Decimal('0.0000'), 'Fabrikam, Inc.', '95642'), (85554,
4, 576, datetime.date(2019, 12, 16), Decimal('407994.00'),
Decimal('407994.0000'), Decimal('0.0000'), 'Fabrikam, Inc.', '95642'), (121887,
7, 814, datetime.date(2020, 5, 7), Decimal('186559.60'), Decimal('186559.6000'),
Decimal('0.0000'), 'Litware, Inc.', '95642'), (85557, 7, 577,
datetime.date(2019, 12, 16), Decimal('98336.40'), Decimal('98336.4000'),
Decimal('0.0000'), 'Litware, Inc.', '95642'), (86182, 4, 581,
datetime.date(2019, 12, 18), Decimal('413298.00'), Decimal('413298.0000'),
Decimal('0.0000'), 'Fabrikam, Inc.', '95642'), (86385, 4, 583,
datetime.date(2019, 12, 19), Decimal('413682.00'), Decimal('413682.0000'),
Decimal('0.0000'), 'Fabrikam, Inc.', '95642'), (87795, 7, 594,
datetime.date(2019, 12, 25), Decimal('102054.80'), Decimal('102054.8000'),
Decimal('0.0000'), 'Litware, Inc.', '95642'), (123490, 4, 821,
datetime.date(2020, 5, 13), Decimal('569754.00'), Decimal('569754.0000'),
Decimal('0.0000'), 'Fabrikam, Inc.', '95642'), (123675, 4, 823,
datetime.date(2020, 5, 14), Decimal('569790.00'), Decimal('569790.0000'),
Decimal('0.0000'), 'Fabrikam, Inc.', '95642'), (124573, 7, 828,
datetime.date(2020, 5, 16), Decimal('189055.20'), Decimal('189055.2000'),
Decimal('0.0000'), 'Litware, Inc.', '95642')]
```

[ ]:
```
##Getting the US Census Data from the zip file within the repo

from pathlib import Path
import zipfile

# Unzip the file
zcta_zip = Path("Data") / "2021_Gaz_zcta_national.zip"
safe_dir = Path.home() / "sf_uploads"
safe_dir.mkdir(parents=True, exist_ok=True)

with zipfile.ZipFile(zcta_zip, 'r') as zf:
    member = [m for m in zf.namelist() if m.lower().endswith(".txt")][0]
    zf.extract(member, safe_dir)
    zcta_txt = safe_dir / member

print("ZCTA extracted:", zcta_txt)

# Stage the file
uri = zcta_txt.resolve().as_uri()
```

```python
cs.execute("CREATE STAGE IF NOT EXISTS PUBLIC.zcta_stage")
cs.execute(f"PUT '{uri}' @PUBLIC.zcta_stage OVERWRITE=TRUE AUTO_COMPRESS=TRUE")
print("Staged:", cs.execute("LIST @PUBLIC.zcta_stage").fetchall())

# Create table for ZIP, lat, lon
cs.execute("""
CREATE OR REPLACE TABLE PUBLIC.ZCTA_2021 (
  GEOID     STRING,
  INTPTLAT  FLOAT,
  INTPTLONG FLOAT
)
""")

# Define TSV file format (skip header, tab delimiter)
# File format: tab-delimited, header row, values optionally quoted, trim spaces
# Use the TSV format we already created
cs.execute("""
CREATE OR REPLACE FILE FORMAT PUBLIC.TSV_FF
  TYPE = CSV
  FIELD_DELIMITER = '\t'
  FIELD_OPTIONALLY_ENCLOSED_BY = '"'
  TRIM_SPACE = TRUE
  SKIP_HEADER = 1
  NULL_IF = ('','NULL')
""")

# Recreate the target table
cs.execute("""
CREATE OR REPLACE TABLE PUBLIC.ZCTA_2021 (
  GEOID     STRING,
  INTPTLAT  DOUBLE,
  INTPTLONG DOUBLE
)
""")

# COPY using correct column positions: $1 (GEOID), $6 (LAT), $7 (LON)
cs.execute("""
COPY INTO PUBLIC.ZCTA_2021 (GEOID, INTPTLAT, INTPTLONG)
FROM (
  SELECT
    $1,
    TRY_TO_DOUBLE(REPLACE($6,'+','')),
    TRY_TO_DOUBLE(REPLACE($7,'+',''))
  FROM @PUBLIC.zcta_stage (FILE_FORMAT => 'PUBLIC.TSV_FF')
)
ON_ERROR = ABORT_STATEMENT
""")
```

```
print("ZCTA rows:", cs.execute("SELECT COUNT(*) FROM PUBLIC.ZCTA_2021").
  ↪fetchone()[0])
print(cs.execute("SELECT GEOID, INTPTLAT, INTPTLONG FROM PUBLIC.ZCTA_2021 ORDER␣
  ↪BY GEOID LIMIT 5").fetchall())
```

```
ZCTA extracted: /home/jovyan/sf_uploads/2021_Gaz_zcta_national.txt
Staged: [('zcta_stage/2021_Gaz_zcta_national.txt.gz', 966848,
'c040089a8850ac7d8e74238beacb9315', 'Mon, 15 Sep 2025 00:09:49 GMT')]
ZCTA rows: 33791
[('00601', 18.180555, -66.749961), ('00602', 18.361945, -67.175597), ('00603',
18.458497, -67.123906), ('00606', 18.158327, -66.932928), ('00610', 18.294032,
-67.127156)]
```

[ ]:
```
# Recreate clean table
cs.execute("""
CREATE OR REPLACE TABLE PUBLIC.ZCTA_2021 (
  GEOID     STRING,
  INTPTLAT  DOUBLE,
  INTPTLONG DOUBLE
)
""")

# COPY: $1=GEOID, $8=INTPTLAT, $9=INTPTLONG; remove leading '+' before casting
cs.execute("""
COPY INTO PUBLIC.ZCTA_2021 (GEOID, INTPTLAT, INTPTLONG)
FROM (
  SELECT
    $1,
    TRY_TO_DOUBLE(REPLACE($8,'+','')),
    TRY_TO_DOUBLE(REPLACE($9,'+',''))
  FROM @PUBLIC.zcta_stage (FILE_FORMAT => 'PUBLIC.TSV_FF')
)
ON_ERROR = ABORT_STATEMENT
""")

print("ZCTA rows:", cs.execute("SELECT COUNT(*) FROM PUBLIC.ZCTA_2021").
  ↪fetchone()[0])
print(cs.execute("SELECT GEOID, INTPTLAT, INTPTLONG FROM PUBLIC.ZCTA_2021 ORDER␣
  ↪BY GEOID LIMIT 5").fetchall())
```

[84]:
```
# Unique supplier ZIPs (normalize to 5 digits)
cs.execute("""
CREATE OR REPLACE TEMP VIEW PUBLIC.SUPPLIER_ZIPS AS
SELECT DISTINCT LPAD(REGEXP_REPLACE(PostalPostalCode,'\\D',''),5,'0') AS ZIP5
FROM PUBLIC.SUPPLIER_CASE
WHERE PostalPostalCode IS NOT NULL
""")

# Join ZIPs to ZCTA lat/lon
cs.execute("""
CREATE OR REPLACE TEMP VIEW PUBLIC.SUPPLIER_ZIPS_WITH_GEO AS
SELECT sz.ZIP5, z.INTPTLAT AS LAT, z.INTPTLONG AS LON
FROM PUBLIC.SUPPLIER_ZIPS sz
JOIN PUBLIC.ZCTA_2021 z
  ON z.GEOID = sz.ZIP5
""")
print("Supplier ZIPs with geo:",
      cs.execute("SELECT COUNT(*) FROM PUBLIC.SUPPLIER_ZIPS_WITH_GEO").
 ↪fetchone()[0])
```

```
Supplier ZIPs with geo: 8
```

[90]:

```
# --- Find weather DB/schema (reuse your function if already defined) ---
def find_weather_db(cs):
    names = [r[1] for r in cs.execute("SHOW DATABASES").fetchall()]
    for cand in ("WEATHER__ENVIRONMENT","WEATHER_ENVIRONMENT"):
        if cand in names: return cand
    for n in names:
        if "WEATHER" in n and "ENVIRONMENT" in n: return n
    return None

db = find_weather_db(cs)
schemas = {r[1] for r in cs.execute(f"SHOW SCHEMAS IN DATABASE {db}").
 ↪fetchall()}
wx_schema = "CYBERSYN" if "CYBERSYN" in schemas else "PUBLIC"

station_idx = f"{db}.{wx_schema}.NOAA_WEATHER_STATION_INDEX"
metrics_ts  = f"{db}.{wx_schema}.NOAA_WEATHER_METRICS_TIMESERIES"

# --- Columns present in each table ---
st_cols = [r[0] for r in cs.execute(f"""
  SELECT column_name
  FROM {db}.INFORMATION_SCHEMA.COLUMNS
  WHERE table_schema='{wx_schema}' AND table_name='NOAA_WEATHER_STATION_INDEX'
""").fetchall()]
mt_cols = [r[0] for r in cs.execute(f"""
  SELECT column_name
  FROM {db}.INFORMATION_SCHEMA.COLUMNS
  WHERE table_schema='{wx_schema}' AND␣
 ↪table_name='NOAA_WEATHER_METRICS_TIMESERIES'
""").fetchall()]
STU, MTU = [c.upper() for c in st_cols], [c.upper() for c in mt_cols]
print("Station index columns:", st_cols)
print("Metrics columns      :", mt_cols)

# Hard-set from your printed columns
lat_col      = "LATITUDE"
lon_col      = "LONGITUDE"
stn_idx_col  = "NOAA_WEATHER_STATION_ID"  # in station index
stn_met_col  = "NOAA_WEATHER_STATION_ID"  # in metrics

# Build ZIP -> nearest station (CROSS JOIN + Haversine)
cs.execute(f"""
CREATE OR REPLACE TEMP VIEW PUBLIC.ZIP_TO_NEAREST_STATION AS
WITH pairs AS (
  SELECT
    z.ZIP5,
    s.{stn_idx_col} AS STATION_KEY,
    s.{lat_col}     AS s_lat,
    s.{lon_col}     AS s_lon,
    z.LAT           AS z_lat,
    z.LON           AS z_lon,
    2*6371*ASIN(SQRT(
        POWER(SIN(RADIANS(z.LAT - s.{lat_col})/2),2)
      + COS(RADIANS(z.LAT))*COS(RADIANS(s.{lat_col}))
      * POWER(SIN(RADIANS(z.LON - s.{lon_col})/2),2)
    )) AS km_distance
  FROM PUBLIC.SUPPLIER_ZIPS_WITH_GEO z
  CROSS JOIN {station_idx} s
)
SELECT ZIP5, STATION_KEY, km_distance
FROM (
  SELECT p.*, ROW_NUMBER() OVER (PARTITION BY ZIP5 ORDER BY km_distance) AS rn
  FROM pairs p
)
WHERE rn = 1
""")

print("ZIP→station rows:",
      cs.execute("SELECT COUNT(*) FROM PUBLIC.ZIP_TO_NEAREST_STATION").
 ↪fetchone()[0])
print(cs.execute("SELECT * FROM PUBLIC.ZIP_TO_NEAREST_STATION LIMIT 5").
 ↪fetchall())
```

```
Station index columns: ['COUNTRY_GEO_ID', 'WEATHER_STATION_NETWORK',
'STATE_GEO_ID', 'LONGITUDE', 'ASSOCIATED_NETWORKS', 'ZIP_NAME',
'WORLD_METEOROLOGICAL_ORGANIZATION_ID', 'LATITUDE', 'ZIP_GEO_ID',
'COUNTRY_NAME', 'STATE_NAME', 'NOAA_WEATHER_STATION_NAME',
'NOAA_WEATHER_STATION_ID', 'ELEVATION', 'SOURCE_DATA']
Metrics columns      : ['UNIT', 'VARIABLE', 'DATETIME', 'VALUE',
'NOAA_WEATHER_STATION_ID', 'VARIABLE_NAME', 'DATE']
```

```
ZIP→station rows: 8
[('06331', 'US1CTWN0011', 8.101088905729062), ('34269', 'US1FLCH0054',
8.065228870939682), ('95642', 'US1CAAM0011', 0.6813252694400262), ('42437',
'USC00155569', 8.955611989180575), ('80125', 'US1CODG0159', 1.7581248900134214)]
```

[92]:

```
# Compose table names
metrics_ts = f"{db}.{wx_schema}.NOAA_WEATHER_METRICS_TIMESERIES"

# Build as MV if allowed; otherwise VIEW
ddl = f"""
SELECT
  z.ZIP5                   AS zip_code,
  m.DATE                   AS date,              -- daily grain
  CAST(m.VALUE AS FLOAT)  AS high_temperature  -- VALUE already numeric
FROM PUBLIC.ZIP_TO_NEAREST_STATION z
JOIN {metrics_ts} m
  ON m.{stn_met_col} = z.STATION_KEY
WHERE m.VALUE IS NOT NULL
  AND (
        UPPER(m.VARIABLE) = 'TMAX'
     OR (UPPER(m.VARIABLE_NAME) LIKE '%MAX%' AND UPPER(m.VARIABLE_NAME) LIKE␣
 ↪'%TEMP%')
  )
"""

built = "MATERIALIZED VIEW"
try:
    cs.execute(f"CREATE OR REPLACE MATERIALIZED VIEW PUBLIC.
 ↪SUPPLIER_ZIP_CODE_WEATHER AS {ddl}")
except Exception:
    built = "VIEW"
    cs.execute(f"CREATE OR REPLACE VIEW PUBLIC.SUPPLIER_ZIP_CODE_WEATHER AS␣
 ↪{ddl}")

print(f"Built PUBLIC.SUPPLIER_ZIP_CODE_WEATHER as {built}.")
print(cs.execute("SELECT * FROM PUBLIC.SUPPLIER_ZIP_CODE_WEATHER LIMIT 10").
 ↪fetchall())
print("Distinct zips covered:",
      cs.execute("SELECT COUNT(DISTINCT zip_code) FROM PUBLIC.
 ↪SUPPLIER_ZIP_CODE_WEATHER").fetchone()[0])
```

```
Built PUBLIC.SUPPLIER_ZIP_CODE_WEATHER as VIEW.
[('42437', datetime.date(2007, 10, 24), 17.2), ('42437', datetime.date(2006, 4,
24), 23.9), ('42437', datetime.date(2006, 6, 28), 26.7), ('42437',
datetime.date(2008, 11, 27), 14.4), ('42437', datetime.date(2008, 6, 4), 32.2),
('42437', datetime.date(2006, 4, 23), 27.8), ('42437', datetime.date(2007, 10,
27), 13.3), ('42437', datetime.date(2008, 2, 28), 0.0), ('42437',
datetime.date(2007, 3, 16), 10.0), ('42437', datetime.date(2005, 11, 26), 6.7)]
```

```
Distinct zips covered: 1
```

[94]:
```
# Distinct supplier ZIPs × invoice dates you actually care about
cs.execute("""
CREATE OR REPLACE TEMP VIEW PUBLIC.NEEDED_ZIP_DATES AS
SELECT
  LPAD(REGEXP_REPLACE(s.PostalPostalCode,'\\D',''),5,'0') AS ZIP5,
  p.InvoiceDate                                           AS DATE
FROM PUBLIC.PURCHASE_ORDERS_AND_INVOICES p
JOIN PUBLIC.SUPPLIER_CASE s
  ON TO_VARCHAR(p.SupplierID) = TO_VARCHAR(s.SupplierID)
GROUP BY 1,2
""")
print("ZIP×date pairs:",
      cs.execute("SELECT COUNT(*) FROM PUBLIC.NEEDED_ZIP_DATES").fetchone()[0])
```

```
ZIP×date pairs: 914
```

[95]:
```
# From your schema: station index & metrics columns
db_schemas = [r[1] for r in cs.execute("SHOW DATABASES").fetchall()]
def find_weather_db():
    for cand in ("WEATHER__ENVIRONMENT","WEATHER_ENVIRONMENT"):
        if cand in db_schemas: return cand
    for n in db_schemas:
        if "WEATHER" in n and "ENVIRONMENT" in n: return n
    return None

db = find_weather_db()
schemas = {r[1] for r in cs.execute(f"SHOW SCHEMAS IN DATABASE {db}").
 ↪fetchall()}
wx_schema = "CYBERSYN" if "CYBERSYN" in schemas else "PUBLIC"

station_idx = f"{db}.{wx_schema}.NOAA_WEATHER_STATION_INDEX"
metrics_ts  = f"{db}.{wx_schema}.NOAA_WEATHER_METRICS_TIMESERIES"

lat_col     = "LATITUDE"
lon_col     = "LONGITUDE"
stn_idx_col = "NOAA_WEATHER_STATION_ID"
stn_met_col = "NOAA_WEATHER_STATION_ID"
```

[96]:
```
# We'll filter metrics to "daily high" rows and the needed dates only
# Then pick nearest station among those with data for that (ZIP, DATE)
ddl = f"""
WITH
needed AS (
  SELECT ZIP5, DATE FROM PUBLIC.NEEDED_ZIP_DATES
),
daily_max AS (
  SELECT
    m.{stn_met_col}   AS STATION_KEY,
    m.DATE            AS DATE,
    CAST(m.VALUE AS FLOAT) AS HIGH_TEMPERATURE
  FROM {metrics_ts} m
  WHERE m.VALUE IS NOT NULL
    AND (
         UPPER(m.VARIABLE) = 'TMAX'
      OR (UPPER(m.VARIABLE_NAME) LIKE '%MAX%' AND UPPER(m.VARIABLE_NAME) LIKE␣
 ↪'%TEMP%')
    )
),
pairs AS (
  SELECT
    n.ZIP5,
    n.DATE,
    d.HIGH_TEMPERATURE,
    s.{stn_idx_col} AS STATION_KEY,
    2*6371*ASIN(SQRT(
        POWER(SIN(RADIANS(z.INTPTLAT - s.{lat_col})/2),2)
      + COS(RADIANS(z.INTPTLAT))*COS(RADIANS(s.{lat_col}))
      * POWER(SIN(RADIANS(z.INTPTLONG - s.{lon_col})/2),2)
    )) AS km_distance
  FROM needed n
  JOIN PUBLIC.ZCTA_2021 z
    ON z.GEOID = n.ZIP5
  JOIN daily_max d
    ON d.DATE = n.DATE
  JOIN {station_idx} s
    ON s.{stn_idx_col} = d.STATION_KEY
)
SELECT ZIP5 AS zip_code, DATE, HIGH_TEMPERATURE
FROM (
  SELECT
    p.*,
    ROW_NUMBER() OVER (PARTITION BY ZIP5, DATE ORDER BY km_distance) AS rn
  FROM pairs p
)
WHERE rn = 1
"""

# Create the final per-zip per-day weather view
try:
    cs.execute(f"CREATE OR REPLACE MATERIALIZED VIEW PUBLIC.
 ↪SUPPLIER_ZIP_CODE_WEATHER AS {ddl}")
    built = "MATERIALIZED VIEW"
except Exception:
    cs.execute(f"CREATE OR REPLACE VIEW PUBLIC.SUPPLIER_ZIP_CODE_WEATHER AS␣
↪{ddl}")
    built = "VIEW"

print(f"Built PUBLIC.SUPPLIER_ZIP_CODE_WEATHER as {built}.")
print(cs.execute("SELECT * FROM PUBLIC.SUPPLIER_ZIP_CODE_WEATHER LIMIT 10").
↪fetchall())
print("Distinct zips covered:",
      cs.execute("SELECT COUNT(DISTINCT zip_code) FROM PUBLIC.
↪SUPPLIER_ZIP_CODE_WEATHER").fetchone()[0])
print("Dates covered (min,max):",
      cs.execute("SELECT MIN(date), MAX(date) FROM PUBLIC.
↪SUPPLIER_ZIP_CODE_WEATHER").fetchone())
```

```
Built PUBLIC.SUPPLIER_ZIP_CODE_WEATHER as VIEW.
[('95642', datetime.date(2022, 5, 20), 29.4), ('95642', datetime.date(2019, 7,
24), 36.1), ('95642', datetime.date(2020, 3, 27), 11.1), ('95642',
datetime.date(2021, 2, 25), 21.7), ('95642', datetime.date(2019, 6, 11), 35.0),
('95642', datetime.date(2020, 6, 24), 36.1), ('95642', datetime.date(2021, 5,
18), 24.4), ('95642', datetime.date(2021, 9, 24), 33.9), ('95642',
datetime.date(2020, 8, 14), 38.9), ('95642', datetime.date(2022, 1, 12), 18.9)]
Distinct zips covered: 3
Dates covered (min,max): (datetime.date(2019, 1, 2), datetime.date(2022, 5, 31))
```

[97]:
```
cs.execute("""
CREATE OR REPLACE VIEW PUBLIC.FINAL_SUPPLIER_PO_INVOICE_WEATHER AS
SELECT
  p.*,
  s.SupplierName,
  LPAD(REGEXP_REPLACE(s.PostalPostalCode,'\\D',''),5,'0') AS SupplierZIP,
  w.HIGH_TEMPERATURE
FROM PUBLIC.PURCHASE_ORDERS_AND_INVOICES p
LEFT JOIN PUBLIC.SUPPLIER_CASE s
  ON TO_VARCHAR(p.SupplierID) = TO_VARCHAR(s.SupplierID)
LEFT JOIN PUBLIC.SUPPLIER_ZIP_CODE_WEATHER w
  ON w.zip_code = LPAD(REGEXP_REPLACE(s.PostalPostalCode,'\\D',''),5,'0')
 AND w.date = p.InvoiceDate
""")

print("Final rows:",
      cs.execute("SELECT COUNT(*) FROM PUBLIC.
↪FINAL_SUPPLIER_PO_INVOICE_WEATHER").fetchone()[0])
print("Rows with weather:",
      cs.execute("SELECT COUNT(*) FROM PUBLIC.FINAL_SUPPLIER_PO_INVOICE_WEATHER␣
↪WHERE HIGH_TEMPERATURE IS NOT NULL").fetchone()[0])

print(cs.execute("""
  SELECT SupplierID, PurchaseOrderID, InvoiceDate, SupplierZIP, HIGH_TEMPERATURE
  FROM PUBLIC.FINAL_SUPPLIER_PO_INVOICE_WEATHER
  WHERE HIGH_TEMPERATURE IS NOT NULL
  ORDER BY InvoiceDate
  LIMIT 10
""").fetchall())
```

```
Final rows: 2072
Rows with weather: 2069
[(4, 2, datetime.date(2019, 1, 2), '95642', 12.2), (7, 4, datetime.date(2019, 1,
2), '95642', 12.2), (10, 5, datetime.date(2019, 1, 2), '22202', 8.9), (5, 3,
datetime.date(2019, 1, 2), '80125', -7.2), (12, 6, datetime.date(2019, 1, 2),
'80125', -7.2), (2, 1, datetime.date(2019, 1, 2), '80125', -7.2), (7, 9,
datetime.date(2019, 1, 3), '95642', 11.1), (4, 7, datetime.date(2019, 1, 3),
'95642', 11.1), (12, 11, datetime.date(2019, 1, 3), '80125', 3.9), (5, 8,
datetime.date(2019, 1, 3), '80125', 3.9)]
```