In [4]:
```
!pip install cassandra-driver
```

```
Collecting cassandra-driver
  Downloading cassandra_driver-3.29.2-cp310-cp310-manylinux_2_17_x86_64.ma
nylinux2014_x86_64.whl.metadata (6.2 kB)
Collecting geomet<0.3,>=0.1 (from cassandra-driver)
  Downloading geomet-0.2.1.post1-py3-none-any.whl.metadata (1.0 kB)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-pac
kages (from geomet<0.3,>=0.1->cassandra-driver) (8.1.7)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packa
ges (from geomet<0.3,>=0.1->cassandra-driver) (1.16.0)
Downloading cassandra_driver-3.29.2-cp310-cp310-manylinux_2_17_x86_64.many
linux2014_x86_64.whl (3.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.9/3.9 MB 33.3 MB/s eta 0:00:
00
Downloading geomet-0.2.1.post1-py3-none-any.whl (18 kB)
Installing collected packages: geomet, cassandra-driver
Successfully installed cassandra-driver-3.29.2 geomet-0.2.1.post1
```

In [7]:
```
from cassandra.cluster import Cluster
from cassandra.auth import PlainTextAuthProvider
import pandas as pd
import json

# Path to the secure connect bundle
SECURE_CONNECT_BUNDLE = "secure-connect-1st-db.zip"

with open("1st_DB-token.json") as f:
  secrets = json.load(f)

# Authenticate and connect
auth_provider = PlainTextAuthProvider(
    'aSZgROdjEjGZNMrpwbnrulud',
    'eQjM3OCWe2UX,N4qrM+0Ek.U-,YdYpsN+3RpJpZReYjwcGWsIs4rGXixyl_ggpCBKIHZ
)
cluster = Cluster(cloud={'secure_connect_bundle': SECURE_CONNECT_BUNDLE},
session = cluster.connect()

# Set the keyspace
session.set_keyspace('sales')

if session:
    print('Connected to database')
else:
    print('Connection failed')
```

```
WARNING:cassandra.cluster:Downgrading core protocol version from 66 to 65
for 006603c6-e580-4575-b9d9-f1b5e69e7bd9-eu-west-1.db.astra.datastax.com:2
9042:9e3a5bee-3d95-3bf7-90f5-09bd2177324b. To avoid this, it is best pract
ice to explicitly set Cluster(protocol_version) to the version supported b
y your cluster. http://datastax.github.io/python-driver/api/cassandra/clus
ter.html#cassandra.cluster.Cluster.protocol_version
WARNING:cassandra.cluster:Downgrading core protocol version from 65 to 5 f
or 006603c6-e580-4575-b9d9-f1b5e69e7bd9-eu-west-1.db.astra.datastax.com:29
042:9e3a5bee-3d95-3bf7-90f5-09bd2177324b. To avoid this, it is best practi
ce to explicitly set Cluster(protocol_version) to the version supported by
your cluster. http://datastax.github.io/python-driver/api/cassandra/cluste
r.html#cassandra.cluster.Cluster.protocol_version
WARNING:cassandra.cluster:Downgrading core protocol version from 5 to 4 fo
r 006603c6-e580-4575-b9d9-f1b5e69e7bd9-eu-west-1.db.astra.datastax.com:290
42:9e3a5bee-3d95-3bf7-90f5-09bd2177324b. To avoid this, it is best practic
e to explicitly set Cluster(protocol_version) to the version supported by
your cluster. http://datastax.github.io/python-driver/api/cassandra/cluste
r.html#cassandra.cluster.Cluster.protocol_version
Connected to database
```

In [10]:
```
# Download sales_100.csv from GitHub
url = "https://raw.githubusercontent.com/gchandra10/filestorage/main/sale
sales_data = pd.read_csv(url)
print(sales_data.head())
```

```
                       Region           Country  Item Type Sales Channel
\
0           Sub-Saharan Africa      South Africa     Fruits       Offline
1  Middle East and North Africa           Morocco    Clothes        Online
2        Australia and Oceania  Papua New Guinea       Meat       Offline
3           Sub-Saharan Africa          Djibouti    Clothes       Offline
4                       Europe          Slovakia  Beverages       Offline

  Order Priority  Order Date   Order ID   Ship Date  UnitsSold  UnitPrice
\
0              M   7/27/2012  443368995   7/28/2012       1593       9.33
1              M   9/14/2013  667593514  10/19/2013       4611     109.28
2              M   5/15/2015  940995585    6/4/2015        360     421.89
3              H   5/17/2017  880811536    7/2/2017        562     109.28
4              L  10/26/2016  174590194   12/4/2016       3973      47.45

   UnitCost  TotalRevenue  TotalCost  TotalProfit
0      6.92      14862.69   11023.56      3839.13
1     35.84     503890.08  165258.24    338631.84
2    364.69     151880.40  131288.40     20592.00
3     35.84      61415.36   20142.08     41273.28
4     31.79     188518.85  126301.67     62217.18
```

In [11]:
```
session.execute("""
CREATE TABLE IF NOT EXISTS bronze_sales (
    id UUID PRIMARY KEY,
    raw_data TEXT
);
""")
```

Out[11]:
```
<cassandra.cluster.ResultSet at 0x7b08e490b490>
```

In [12]:
```
import uuid

for _, row in sales_data.iterrows():
```

```
    raw_data = row.to_json()
    session.execute(
        "INSERT INTO bronze_sales (id, raw_data) VALUES (%s, %s)",
        (uuid.uuid4(), raw_data)
    )
print("Bronze table populated with raw data.")
```

```
Bronze table populated with raw data.
```

In [13]:
```
session.execute("""
CREATE TABLE IF NOT EXISTS silver_sales (
    order_id BIGINT PRIMARY KEY,
    region TEXT,
    country TEXT,
    item_type TEXT,
    sales_channel TEXT,
    order_priority TEXT,
    order_date DATE,
    ship_date DATE,
    units_sold INT,
    unit_price FLOAT,
    unit_cost FLOAT,
    total_revenue FLOAT,
    total_cost FLOAT,
    total_profit FLOAT
);
""")
print("Recreated the silver_sales table.")
```

```
Recreated the silver_sales table.
```

In [14]:
```
sales_data.rename(columns={
    'Order ID': 'order_id',
    'Region': 'region',
    'Country': 'country',
    'Item Type': 'item_type',
    'Sales Channel': 'sales_channel',
    'Order Priority': 'order_priority',
    'Order Date': 'order_date',
    'Ship Date': 'ship_date',
    'UnitsSold': 'units_sold',
    'UnitPrice': 'unit_price',
    'UnitCost': 'unit_cost',
    'TotalRevenue': 'total_revenue',
    'TotalCost': 'total_cost',
    'TotalProfit': 'total_profit'
}, inplace=True)
```

In [15]:
```
sales_data['order_id'] = sales_data['order_id'].astype(int)
sales_data['units_sold'] = sales_data['units_sold'].astype(int)
sales_data['unit_price'] = sales_data['unit_price'].astype(float)
sales_data['unit_cost'] = sales_data['unit_cost'].astype(float)
sales_data['total_revenue'] = sales_data['total_revenue'].astype(float)
sales_data['total_cost'] = sales_data['total_cost'].astype(float)
sales_data['total_profit'] = sales_data['total_profit'].astype(float)

# Convert date columns
sales_data['order_date'] = pd.to_datetime(sales_data['order_date']).dt.da
sales_data['ship_date'] = pd.to_datetime(sales_data['ship_date']).dt.date
```

In [16]:
```
for _, row in sales_data.iterrows():
    session.execute("""
    INSERT INTO silver_sales (order_id, region, country, item_type, sales
                              order_priority, order_date, ship_date, unit
                              unit_price, unit_cost, total_revenue, total
                              total_profit)
    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
    """, (
        row['order_id'], row['region'], row['country'], row['item_type'],
        row['sales_channel'], row['order_priority'], row['order_date'],
        row['ship_date'], row['units_sold'], row['unit_price'],
        row['unit_cost'], row['total_revenue'], row['total_cost'], row['t
    ))
print("Silver table populated with cleaned data.")
```

```
Silver table populated with cleaned data.
```

In [17]:
```
rows = session.execute("SELECT * FROM silver_sales LIMIT 5")
for row in rows:
    print(row)
```

```
Row(order_id=294530856, country='Italy', item_type='Cereal', order_date=Da
te(15293), order_priority='M', region='Europe', sales_channel='Online', sh
ip_date=Date(15336), total_cost=829138.8125, total_profit=627217.1875, tot
al_revenue=1456356.0, unit_cost=117.11000061035156, unit_price=205.6999969
482422, units_sold=7080)
Row(order_id=274930989, country='Dominica', item_type='Household', order_d
ate=Date(15297), order_priority='C', region='Central America and the Carib
bean', sales_channel='Offline', ship_date=Date(15321), total_cost=3539891.
75, total_profit=1167402.125, total_revenue=4707294.0, unit_cost=502.54000
85449219, unit_price=668.27001953125, units_sold=7044)
Row(order_id=498071897, country='Taiwan', item_type='Cereal', order_date=D
ate(14710), order_priority='H', region='Asia', sales_channel='Online', shi
p_date=Date(14755), total_cost=1100482.625, total_profit=832480.25, total_
revenue=1932962.875, unit_cost=117.11000061035156, unit_price=205.69999694
82422, units_sold=9397)
Row(order_id=940980136, country='New Zealand', item_type='Beverages', orde
r_date=Date(15624), order_priority='M', region='Australia and Oceania', sa
les_channel='Online', ship_date=Date(15648), total_cost=184000.515625, tot
al_profit=90640.078125, total_revenue=274640.59375, unit_cost=31.790000915
527344, unit_price=47.45000076293945, units_sold=5788)
Row(order_id=324669444, country='France', item_type='Cosmetics', order_dat
e=Date(16776), order_priority='M', region='Europe', sales_channel='Onlin
e', ship_date=Date(16818), total_cost=1516254.125, total_profit=1001143.43
75, total_revenue=2517397.5, unit_cost=263.3299865722656, unit_price=437.2
0001220703125, units_sold=5758)
```

1st Gold Table

In [18]:
```
session.execute("""
CREATE TABLE IF NOT EXISTS gold_sales_by_region (
    region TEXT PRIMARY KEY,
    total_sales FLOAT
);
""")
```

Out[18]:
```
<cassandra.cluster.ResultSet at 0x7b08ac3bcf40>
```

In [19]:
```
region_sales = sales_data.groupby('region')['total_revenue'].sum().reset_

for _, row in region_sales.iterrows():
    session.execute("""
    INSERT INTO gold_sales_by_region (region, total_sales) VALUES (%s, %s
    """, (row['region'], float(row['total_revenue'])))
print("Gold table 1 populated with total sales by region.")
```

```
Gold table 1 populated with total sales by region.
```

In [20]:
```
# Query the table and fetch data
rows = session.execute("SELECT * FROM gold_sales_by_region")

# Convert the result set to a list of dictionaries
data = [{'Region': row.region, 'Total Sales': row.total_sales} for row in

# Create a DataFrame
df = pd.DataFrame(data)

# Display the DataFrame
print("Gold Table 1: Total Sales by Region")
print(df)
```

```
Gold Table 1: Total Sales by Region
                              Region  Total Sales
0              Australia and Oceania   10711258.0
1                             Europe   34964748.0
2       Middle East and North Africa   24765128.0
3  Central America and the Caribbean   17570836.0
4                               Asia   28840812.0
5                 Sub-Saharan Africa   24225438.0
6                      North America    3611757.5
```

2nd Gold Table

> Top Selling Items

In [21]:
```
session.execute("""
CREATE TABLE IF NOT EXISTS gold_top_items (
    item_type TEXT PRIMARY KEY,
    total_units_sold INT
);
""")
```

Out[21]:
```
<cassandra.cluster.ResultSet at 0x7b08ac3bf940>
```

In [24]:
```
item_sales = sales_data.groupby('item_type')['units_sold'].sum().reset_in

for _, row in item_sales.iterrows():
    session.execute("""
    INSERT INTO gold_top_items (item_type, total_units_sold) VALUES (%s,
    """, (row['item_type'], int(row['units_sold'])))
print("Gold table 2 populated with top-selling items.")
```

```
Gold table 2 populated with top-selling items.
```

In [25]:
```
import pandas as pd

# Query the table and fetch data
rows = session.execute("SELECT * FROM gold_top_items")

# Convert the result set to a list of dictionaries
data = [{'Item Type': row.item_type, 'Total Units Sold': row.total_units_

# Create a DataFrame
df = pd.DataFrame(data)

# Display the DataFrame
print("Gold Table 2: Top-Selling Items")
print(df)
```

```
Gold Table 2: Top-Selling Items
          Item Type  Total Units Sold
0         Household             57640
1   Office Supplies             42814
2        Vegetables              7368
3            Snacks             14377
4     Personal Care             39045
5              Meat             50437
6            Fruits             65920
7         Beverages             45206
8            Cereal             45776
9         Cosmetics             65707
10        Baby Food             20372
11          Clothes             40148
```

3rd Gold Table

> Most Profitable Item

In [26]:
```
session.execute("""
CREATE TABLE IF NOT EXISTS gold_profit_by_priority (
    order_priority TEXT PRIMARY KEY,
    total_profit FLOAT
);
""")
```

Out[26]:
```
<cassandra.cluster.ResultSet at 0x7b08a96df070>
```

In [28]:
```
priority_profit = sales_data.groupby('order_priority')['total_profit'].su

for _, row in priority_profit.iterrows():
    session.execute("""
    INSERT INTO gold_profit_by_priority (order_priority, total_profit) VA
    """, (row['order_priority'], float(row['total_profit'])))
print("Gold table 3 populated with profit by priority.")
```

```
Gold table 3 populated with profit by priority.
```

In [29]:
```
import pandas as pd

# Query the table and fetch data
rows = session.execute("SELECT * FROM gold_profit_by_priority")
```

```
# Convert the result set to a list of dictionaries
data = [{'Order Priority': row.order_priority, 'Total Profit': row.total_

# Create a DataFrame
df = pd.DataFrame(data)

# Display the DataFrame
print("Gold Table 3: Profit by Order Priority")
print(df)
```

```
Gold Table 3: Profit by Order Priority
  Order Priority  Total Profit
0              C     7649023.0
1              M    14607042.0
2              H    11910944.0
3              L     7160182.5
```

In [30]:
```
rows = session.execute("SELECT * FROM gold_sales_by_region")
for row in rows:
    print(row)

rows = session.execute("SELECT * FROM gold_top_items")
for row in rows:
    print(row)

rows = session.execute("SELECT * FROM gold_profit_by_priority")
for row in rows:
    print(row)
```

```
Row(region='Australia and Oceania', total_sales=10711258.0)
Row(region='Europe', total_sales=34964748.0)
Row(region='Middle East and North Africa', total_sales=24765128.0)
Row(region='Central America and the Caribbean', total_sales=17570836.0)
Row(region='Asia', total_sales=28840812.0)
Row(region='Sub-Saharan Africa', total_sales=24225438.0)
Row(region='North America', total_sales=3611757.5)
Row(item_type='Household', total_units_sold=57640)
Row(item_type='Office Supplies', total_units_sold=42814)
Row(item_type='Vegetables', total_units_sold=7368)
Row(item_type='Snacks', total_units_sold=14377)
Row(item_type='Personal Care', total_units_sold=39045)
Row(item_type='Meat', total_units_sold=50437)
Row(item_type='Fruits', total_units_sold=65920)
Row(item_type='Beverages', total_units_sold=45206)
Row(item_type='Cereal', total_units_sold=45776)
Row(item_type='Cosmetics', total_units_sold=65707)
Row(item_type='Baby Food', total_units_sold=20372)
Row(item_type='Clothes', total_units_sold=40148)
Row(order_priority='C', total_profit=7649023.0)
Row(order_priority='M', total_profit=14607042.0)
Row(order_priority='H', total_profit=11910944.0)
Row(order_priority='L', total_profit=7160182.5)
```

In [31]:
```
# Function to display rows in a cleaner format
def display_table(title, rows, columns):
    print(f"\n{title}")
    print("-" * len(title))
    for row in rows:
        print(", ".join([f"{col}: {getattr(row, col)}" for col in columns
    print("\n")
```

```
# Query and display Gold Table 1: Total Sales by Region
rows = session.execute("SELECT * FROM gold_sales_by_region")
display_table("Gold Table 1: Total Sales by Region", rows, ["region", "to

# Query and display Gold Table 2: Top-Selling Items
rows = session.execute("SELECT * FROM gold_top_items")
display_table("Gold Table 2: Top-Selling Items", rows, ["item_type", "tot

# Query and display Gold Table 3: Profit by Order Priority
rows = session.execute("SELECT * FROM gold_profit_by_priority")
display_table("Gold Table 3: Profit by Order Priority", rows, ["order_pri
```

```
Gold Table 1: Total Sales by Region
-----------------------------------
region: Australia and Oceania, total_sales: 10711258.0
region: Europe, total_sales: 34964748.0
region: Middle East and North Africa, total_sales: 24765128.0
region: Central America and the Caribbean, total_sales: 17570836.0
region: Asia, total_sales: 28840812.0
region: Sub-Saharan Africa, total_sales: 24225438.0
region: North America, total_sales: 3611757.5



Gold Table 2: Top-Selling Items
-------------------------------
item_type: Household, total_units_sold: 57640
item_type: Office Supplies, total_units_sold: 42814
item_type: Vegetables, total_units_sold: 7368
item_type: Snacks, total_units_sold: 14377
item_type: Personal Care, total_units_sold: 39045
item_type: Meat, total_units_sold: 50437
item_type: Fruits, total_units_sold: 65920
item_type: Beverages, total_units_sold: 45206
item_type: Cereal, total_units_sold: 45776
item_type: Cosmetics, total_units_sold: 65707
item_type: Baby Food, total_units_sold: 20372
item_type: Clothes, total_units_sold: 40148



Gold Table 3: Profit by Order Priority
--------------------------------------
order_priority: C, total_profit: 7649023.0
order_priority: M, total_profit: 14607042.0
order_priority: H, total_profit: 11910944.0
order_priority: L, total_profit: 7160182.5
```