In [1]:
```
import datetime
from datetime import timedelta
import urllib.request
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.pandas.utils import spark_column_equals
from pyspark.sql import SparkSession, Window
import sys
from collections import namedtuple
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark.sql.functions import *
import builtins
import random
import urllib.request
import os

python_path = sys.executable
os.environ['PYSPARK_PYTHON'] = python_path
os.environ['JAVA_HOME'] = r'C:\Users\LENOVO\.jdks\corretto-1.8.0_422'
os.environ["HADOOP_HOME"] = "C:\Program Files\Hadoop"

conf = (SparkConf()
                .setAppName("pyspark")
                .setMaster("local[*]")
                .set("spark.driver.host", "localhost")
                .set("spark.default.parallelism", "1")
                )

# Check if a SparkContext already exists
if 'sc' in globals():
    sc.stop()  # Stop the existing SparkContext

sc = SparkContext(conf=conf)
spark = SparkSession.builder.getOrCreate()
```

```
c:\Users\LENOVO\AppData\Local\Programs\Python\Python310\lib\site-packages\pyspark\pandas\__init__.py:50: UserWarning: 'PYARROW_IGNORE_TIMEZONE' environment variable was not set. It is required to set this environment variable to '1' in both driver and executor sides if you use pyarrow>=2.0.0. pandas-on-Spark will set it for you but it does not work if there is a Spark context already launched.
  warnings.warn(
```

1. Assume you're given a table Twitter tweet data, write a query to obtain a histogram of tweets posted per user in 2022. Output the tweet count per user as the bucket and the number of Twitter users who fall into that bucket. In other words, group the users by the number of tweets they posted in 2022 and count the number of users in each group.

In [2]:
```
x = [
("214252",      "111","Am considering taking Tesla private at $420. Funding secured.",      "12/30/2021 00:00:00"),
("739252",      "111","Despite the constant negative press covfefe",                        "01/01/2022 00:00:00"),
("846402",      "111","Following @NickSinghTech on Twitter changed my life!",               "02/14/2022 00:00:00"),
("241425",      "254","If the salary is so competitive why won't you tell me what it is?",  "03/01/2022 00:00:00"),
("231574",      "148","I no longer have a manager. I can't be managed",                     "03/23/2022 00:00:00"),
]
df_daily_sales = spark.createDataFrame(x, ['tweet_id', 'user_id', 'msg', 'tweet_date'])

df_daily_sales.show()
df_daily_sales.printSchema()
df_with_timestamp = (df_daily_sales
                     .withColumn(
                    "year",
                        to_timestamp(df_daily_sales["tweet_date"], "MM/dd/yyyy HH:mm:ss"))
                    )
# filtering the  tweets in the year 2022
df = (df_with_timestamp
        .filter(year(df_with_timestamp["year"]) == 2022)
        .groupby("user_id")
        .agg(count("tweet_id").alias("tweet_count_per_user")) )
df.show()

# grouping by the count of the tweets
res = df.groupby("tweet_count_per_user").agg(count("user_id").alias("tweet_bucket"))
res.show()
```

```
+--------+-------+--------------------+--------------------+
|tweet_id|user_id|                 msg|         tweet_date|
+--------+-------+--------------------+--------------------+
|  214252|    111|Am considering ta...|12/30/2021 00:00:00|
|  739252|    111|Despite the const...|01/01/2022 00:00:00|
|  846402|    111|Following @NickSi...|02/14/2022 00:00:00|
|  241425|    254|If the salary is ...|03/01/2022 00:00:00|
|  231574|    148|I no longer have ...|03/23/2022 00:00:00|
+--------+-------+--------------------+--------------------+

root
 |-- tweet_id: string (nullable = true)
 |-- user_id: string (nullable = true)
 |-- msg: string (nullable = true)
 |-- tweet_date: string (nullable = true)

+-------+--------------------+
|user_id|tweet_count_per_user|
+-------+--------------------+
|    111|                   2|
|    148|                   1|
|    254|                   1|
+-------+--------------------+

+--------------------+------------+
|tweet_count_per_user|tweet_bucket|
+--------------------+------------+
|                   1|           2|
|                   2|           1|
+--------------------+------------+
```

2. Given a table of candidates and their skills, you're tasked with finding the candidates best suited for an open Data Science job. You want to find candidates who are proficient in Python, Tableau, and PostgreSQL. Write a query to list the candidates who possess all of the required skills for the job. Sort the output by candidate ID in ascending order.

In [3]:
```
columns = ["candidate_id",      "skill"]
data = [
    ("123",     "Python"),
    ("123",     "Tableau"),
    ("123",     "PostgreSQL"),
    ("234",     "R"),
```

```
    ("234",     "PowerBI"),
    ("234",     "SQL Server"),
    ("345",     "Python"),
    ("345",     "Tableau"),
]
df = spark.createDataFrame(data, columns)
df.show()
df.printSchema()
# using dictionary to check for the skills needed
required_skills_dict = {"Python", "Tableau", "PostgreSQL"}
filtered_candidates = df.filter(df.skill.isin(required_skills_dict))
filtered_candidates.show()
candidates_with_skills = (filtered_candidates
                          .groupBy("candidate_id")
                          .agg(countDistinct("skill").alias("skill_count"))
                          .filter(col("skill_count") == len(required_skills_dict))
                          .orderBy("candidate_id"))
candidates_with_skills.show()

list_df=df.groupby("candidate_id").agg(sort_array(collect_list("skill")).alias("skills"))
# when you use the array to compare there will be a datatype difference this below is the python list
# not a java array list so we need to convert it into a datatype that spark can read using lit() method
required_skills = sorted(["Python", "Tableau", "PostgreSQL"])
required_skills_column = array([lit(skill) for skill in required_skills])
list_df.show()
print(required_skills_column)
res = list_df.filter(list_df["skills"] == required_skills_column)
res.show()

df.createOrReplaceTempView("candidates")
spark.sql("""
SELECT
    candidate_id
FROM
    candidates
WHERE
    skill IN ('Python', 'Tableau', 'PostgreSQL')
GROUP BY
    candidate_id
HAVING
    COUNT(skill) >= 3
ORDER BY
    candidate_id;
```

```
""").show()
```

```
+------------+----------+
|candidate_id|     skill|
+------------+----------+
|         123|    Python|
|         123|   Tableau|
|         123|PostgreSQL|
|         234|         R|
|         234|   PowerBI|
|         234|SQL Server|
|         345|    Python|
|         345|   Tableau|
+------------+----------+

root
 |-- candidate_id: string (nullable = true)
 |-- skill: string (nullable = true)

+------------+----------+
|candidate_id|     skill|
+------------+----------+
|         123|    Python|
|         123|   Tableau|
|         123|PostgreSQL|
|         345|    Python|
|         345|   Tableau|
+------------+----------+

+------------+-----------+
|candidate_id|skill_count|
+------------+-----------+
|         123|          3|
+------------+-----------+

+------------+--------------------+
|candidate_id|              skills|
+------------+--------------------+
|         234|[PowerBI, R, SQL ...|
|         345|   [Python, Tableau]|
|         123|[PostgreSQL, Pyth...|
+------------+--------------------+

Column<'array(PostgreSQL, Python, Tableau)'>
+------------+--------------------+
```

```
|candidate_id|              skills|
+------------+--------------------+
|         123|[PostgreSQL, Pyth...|
+------------+--------------------+

+------------+
|candidate_id|
+------------+
|         123|
+------------+
```

3. Assume you're given two tables containing data about Facebook Pages and their respective likes (as in "Like a Facebook Page"). Write a query to return the IDs of the Facebook pages that have zero likes. The output should be sorted in ascending order based on the page IDs.

In [4]:
```
columns1 = ["page_id",  "page_name"]
data1 = [
    ("20001",   "SQL Solutions"),
    ("20045",   "Brain Exercises"),
    ("20701",   "Tips for Data Analysts")
]
columns2=["user_id",    "page_id",      "liked_date"]
data2= [
    ("111",     "20001",        "04/08/2022 00:00:00"),
    ("121",     "20045",        "03/12/2022 00:00:00"),
    ("156",     "20001",        "07/25/2022 00:00:00")
]

df1 = spark.createDataFrame(data1, columns1)
df2 = spark.createDataFrame(data2, columns2)
df1.show()
df2.show()

df1.createOrReplaceTempView("df1")
df2.createOrReplaceTempView("df2")
spark.sql("""
    select
        p.page_id,
        p.page_name
    from
        df1 p
    left anti join
```

```
        df2 l
    on
        p.page_id=l.page_id
    order by
        page_id
""").show()
# df1.join(df2, "page_id", "left_anti" ).show()
df1.join(df2, "page_id", "anti" ).orderBy("page_id").show()
```

```
+-------+--------------------+
|page_id|           page_name|
+-------+--------------------+
|  20001|       SQL Solutions|
|  20045|     Brain Exercises|
|  20701|Tips for Data Ana...|
+-------+--------------------+

+-------+-------+-------------------+
|user_id|page_id|         liked_date|
+-------+-------+-------------------+
|    111|  20001|04/08/2022 00:00:00|
|    121|  20045|03/12/2022 00:00:00|
|    156|  20001|07/25/2022 00:00:00|
+-------+-------+-------------------+

+-------+--------------------+
|page_id|           page_name|
+-------+--------------------+
|  20701|Tips for Data Ana...|
+-------+--------------------+

+-------+--------------------+
|page_id|           page_name|
+-------+--------------------+
|  20701|Tips for Data Ana...|
+-------+--------------------+
```

4. Tesla is investigating production bottlenecks and they need your help to extract the relevant data. Write a query to determine which parts have begun the assembly process but are not yet finished. Assumptions: parts_assembly table contains all parts currently in production, each at varying stages of the assembly process. An unfinished part is one that lacks a finish_date. This question is straightforward, so let's approach it with simplicity in both thinking and solution. Effective April 11th 2023, the problem statement and assumptions were updated to enhance clarity.

In [5]:
```
columns =["part",       "finish_date",  "assembly_step"]
data=[
    ("battery", "01/22/2022 00:00:00","1"),
    ("battery", "02/22/2022 00:00:00","2"),
    ("battery", "03/22/2022 00:00:00","3"),
    ("bumper",   "01/22/2022 00:00:00","1"),
    ("bumper",   "02/22/2022 00:00:00","2"),
    ("bumper",None,"3"),
    ("bumper",None,"4"),
 ]

df = spark.createDataFrame(data, columns)

df.show()

df.filter(df["finish_date"].isNull()).show()

df.createOrReplaceTempView("df")
spark.sql("""
    select
        part,
        assembly_step
    from
        df
    where
        finish_date is NULL
""").show()
```

```
+-------+-------------------+-------------+
|   part|        finish_date|assembly_step|
+-------+-------------------+-------------+
|battery|01/22/2022 00:00:00|            1|
|battery|02/22/2022 00:00:00|            2|
|battery|03/22/2022 00:00:00|            3|
| bumper|01/22/2022 00:00:00|            1|
| bumper|02/22/2022 00:00:00|            2|
| bumper|               NULL|            3|
| bumper|               NULL|            4|
+-------+-------------------+-------------+

+------+-----------+-------------+
|  part|finish_date|assembly_step|
+------+-----------+-------------+
|bumper|       NULL|            3|
|bumper|       NULL|            4|
+------+-----------+-------------+

+------+-------------+
|  part|assembly_step|
+------+-------------+
|bumper|            3|
|bumper|            4|
+------+-------------+
```

5. This is the same question as problem #3 in the SQL Chapter of Ace the Data Science Interview! Assume you're given the table on user viewership categorised by device type where the three types are laptop, tablet, and phone. Write a query that calculates the total viewership for laptops and mobile devices where mobile is defined as the sum of tablet and phone viewership. Output the total viewership for laptops as laptop_reviews and the total viewership for mobile devices as mobile_views. Effective 15 April 2023, the solution has been updated with a more concise and easy-to-understand approach.

In [6]:
```python
columns = ["user_id",   "device_type","view_time"]
data = [
    ("123",     "tablet",       "01/02/2022 00:00:00"),
    ("125",     "laptop",       "01/07/2022 00:00:0"),
    ("128",     "laptop",       "02/09/2022 00:00:0"),
    ("129",     "phone",        "02/09/2022 00:00:0"),
    ("145",     "tablet",       "02/24/2022 00:00:0"),
]
```

```
df = spark.createDataFrame(data, columns)
df.show()

df.createOrReplaceTempView("df")
spark.sql("""
    SELECT
        sum(case when device_type='laptop' then 1 else 0 end) as laptop_views,
        sum(case when device_type in ('tablet', 'phone') then 1 else 0 end) as mobile_view
    from df;
""").show()

df.agg(
    sum(when(col("device_type")=='laptop',1).otherwise(0)).alias("laptop_views"),
    sum(when(col("device_type").isin("tablet", "phone"),1).otherwise(0)).alias("mobile_views")
).show()
```

```
+-------+-----------+-------------------+
|user_id|device_type|          view_time|
+-------+-----------+-------------------+
|    123|     tablet|01/02/2022 00:00:00|
|    125|     laptop| 01/07/2022 00:00:0|
|    128|     laptop| 02/09/2022 00:00:0|
|    129|      phone| 02/09/2022 00:00:0|
|    145|     tablet| 02/24/2022 00:00:0|
+-------+-----------+-------------------+

+------------+-----------+
|laptop_views|mobile_view|
+------------+-----------+
|           2|          3|
+------------+-----------+

+------------+------------+
|laptop_views|mobile_views|
+------------+------------+
|           2|           3|
+------------+------------+
```

6. Given a table of Facebook posts, for each user who posted at least twice in 2021, write a query to find the number of days between each user's first post of the year and last post of the year in the year 2021. Output the user and number of the days between each user's first and last post.

In [7]:

```
columns = ["user_id",   "post_id",      "post_content", "post_date"]
data = [
    ("151652",  "599415",       "Need a hug",   "07/10/2021 12:00:00"),
    ("661093",  "624356",       "Bed. Class 8-12. Work 12-3. Gym 3-5 or 6. Then class 6-10. Another day that's gonna fly by. I mis
    ("004239",  "784254",       "Happy 4th of July!",   "07/04/2021 11:00:00"),
    ("661093",  "442560",       "Just going to cry myself to sleep after watching Marley and Me.",      "07/08/2021 14:00:00"),
    ("151652",  "111766",       "I'm so done with covid - need travelling ASAP!",       "07/12/2021 19:00:00")
]

df = spark.createDataFrame(data, columns)
df = df.withColumn("post_date", to_timestamp("post_date", "MM/dd/yyyy HH:mm:ss"))
df.show()
df.createOrReplaceTempView("df")
spark.sql("""
    select
        user_id,
        date_diff(max(post_date),min(post_date)) as days_different
    from
        df
    where
        year(post_date)=2021
    group by
        user_id
    having
        count(post_id)>1
    order by
        days_different desc
""").show()

( df
    .filter(year('post_date')==2021)
    .groupBy("user_id")
    .agg(date_diff(max("post_date"),min("post_date")).alias("days_difference"))
    .filter("count(post_id)>1")
    .orderBy(col("days_difference").desc())
    .show()
)
```

```
+-------+-------+--------------------+-------------------+
|user_id|post_id|        post_content|          post_date|
+-------+-------+--------------------+-------------------+
| 151652| 599415|          Need a hug|2021-07-10 12:00:00|
| 661093| 624356|Bed. Class 8-12. ...|2021-07-29 13:00:00|
| 004239| 784254|  Happy 4th of July!|2021-07-04 11:00:00|
| 661093| 442560|Just going to cry...|2021-07-08 14:00:00|
| 151652| 111766|I'm so done with ...|2021-07-12 19:00:00|
+-------+-------+--------------------+-------------------+

+-------+--------------+
|user_id|days_different|
+-------+--------------+
| 661093|            21|
| 151652|             2|
+-------+--------------+

+-------+---------------+
|user_id|days_difference|
+-------+---------------+
| 661093|             21|
| 151652|              2|
+-------+---------------+
```

7. Write a query to identify the top 2 Power Users who sent the highest number of messages on Microsoft Teams in August 2022. Display the IDs of these 2 users along with the total number of messages they sent. Output the results in descending order based on the count of the messages. Assumption: No two users have sent the same number of messages in August 2022. find the top 2 senders in the month of august 2022

In [8]:
```
columns=["message_id",  "sender_id",    "receiver_id",  "content",      "sent_date"]
data = [
    ("901",     "3601", "4500", "You up?",      "08/03/2022 00:00:00"),
    ("902",     "4500", "3601", "Only if you're buying",        "08/03/2022 00:00:00"),
    ("743",     "3601", "8752", "Let's take this offline",      "06/14/2022 00:00:00"),
    ("922",     "3601", "4500", "Get on the call",      "08/10/2022 00:00:00"),
]

rdd = spark.sparkContext.parallelize(data)
df = rdd.toDF(columns)
# df = spark.createDataFrame(data, columns)
df = df.withColumn("sent_date", to_timestamp("sent_date", "MM/dd/yyyy HH:mm:ss"))
df.createOrReplaceTempView("df")
```

```
df.show()
# df.printSchema()

# Perform filtering, grouping, counting, ordering, and limiting
( df
    .filter((year('sent_date')==2022)&(month('sent_date')==8))
    .groupBy("sender_id")
    .count()
    .orderBy(col("count").desc())
    .limit(2)
    .show()
)

# df.filter(year('sent_date')==2022) & (month('sent_date')==8)).groupBy("sender_id").count().orderBy(col("count").desc()).limit(2)
spark.sql("""
    select
        sender_id,
        count(*) as count
    from
        df
    where
       year(sent_date)=2022
       and
       month(sent_date)=8
    group by
        sender_id
    order by
        count desc
""").show()
```

```
+----------+---------+-----------+--------------------+-------------------+
|message_id|sender_id|receiver_id|             content|          sent_date|
+----------+---------+-----------+--------------------+-------------------+
|       901|     3601|       4500|             You up?|2022-08-03 00:00:00|
|       902|     4500|       3601|Only if you're bu...|2022-08-03 00:00:00|
|       743|     3601|       8752|Let's take this o...|2022-06-14 00:00:00|
|       922|     3601|       4500|     Get on the call|2022-08-10 00:00:00|
+----------+---------+-----------+--------------------+-------------------+

+---------+-----+
|sender_id|count|
+---------+-----+
|     3601|    2|
|     4500|    1|
+---------+-----+

+---------+-----+
|sender_id|count|
+---------+-----+
|     3601|    2|
|     4500|    1|
+---------+-----+
```

8. This is the same question as problem #8 in the SQL Chapter of Ace the Data Science Interview! Assume you're given a table containing job postings from various companies on the LinkedIn platform. Write a query to retrieve the count of companies that have posted duplicate job listings. Definition: Duplicate job listings are defined as two job listings within the same company that share identical titles and descriptions.

In [9]:
```
columns = ["job_id",    "company_id",   "title",        "description"]

data = [
(
    "248",
    "827",
    "Business Analyst",
    """Business analyst evaluates past and current business data with the primary
    goal of improving decision-making processes within organizations."""
),
(
    "149",
    "845",
    "Business Analyst",
```

```
    """Business analyst evaluates past and current business data with the primary
    goal of improving decision-making processes within organizations."""
),
(
    "945",
    "345",
    "Data Analyst",
    """Data analyst reviews data to identify key insights into a business's customers
    and ways the data can be used to solve problems."""
),
(
    "164",
    "345",
    "Data Analyst",
    """Data analyst reviews data to identify key insights into a business's customers
    and ways the data can be used to solve problems."""
),
(
    "172",
    "244",
    "Data Engineer",
    """Data engineer works in a variety of settings to build systems that collect,
    manage, and convert raw data into usable information for data scientists and
    business analysts to interpret."""
),
]

rdd = sc.parallelize(data)
df = rdd.toDF(columns)
df.show()
df.createOrReplaceTempView("df")
spark.sql("""
    WITH job_count_cte AS (
          SELECT
                company_id,
                title,
                description,
                COUNT(job_id) AS job_count
          FROM df
          GROUP BY company_id, title, description
    )
    SELECT COUNT(DISTINCT company_id) AS duplicate_companies
    FROM job_count_cte
```

```
    WHERE job_count > 1;
""").show()


spark.sql("""
SELECT COUNT(DISTINCT company_id) AS duplicate_company_count
FROM (
    SELECT company_id, COUNT(*) AS cnt
    FROM df
    GROUP BY company_id, title, description
    HAVING cnt > 1
)
""").show()

( df
 .groupby("company_id", "title", "description")
 .agg(count("*").alias("count"))
 .filter("count>1")
 .select("company_id")
 .distinct()
 .show()
)
```

```
+------+----------+---------------+--------------------+
|job_id|company_id|          title|         description|
+------+----------+---------------+--------------------+
|   248|       827|Business Analyst|Business analyst ...|
|   149|       845|Business Analyst|Business analyst ...|
|   945|       345|    Data Analyst|Data analyst revi...|
|   164|       345|    Data Analyst|Data analyst revi...|
|   172|       244|   Data Engineer|Data engineer wor...|
+------+----------+---------------+--------------------+

+-------------------+
|duplicate_companies|
+-------------------+
|                  1|
+-------------------+

+-----------------------+
|duplicate_company_count|
+-----------------------+
|                      1|
+-----------------------+

+----------+
|company_id|
+----------+
|       345|
+----------+
```

9. This is the same question as problem #2 in the SQL Chapter of Ace the Data Science Interview! Assume you're given the tables containing completed trade orders and user details in a Robinhood trading system. Write a query to retrieve the top three cities that have the highest number of completed trade orders listed in descending order. Output the city name and the corresponding number of completed trade orders.

In [10]:
```
columns1=["order_id",   "user_id",      "quantity",     "status",       "date", "price"]
data1=[
        ("100101",      "111",  "10",   "Cancelled",    "08/17/2022 12:00:00",  "9.80"),
        ("100102",      "111",  "10","Completed",       "08/17/2022 12:00:00",  "10.00"),
        ("100259",      "148",  "35","Completed","08/25/2022 12:00:00", "5.10"),
        ("100264",      "148",  "40","Completed","08/26/2022 12:00:00","4.80"),
        ("100305",      "300",  "15","Completed","09/05/2022 12:00:00","10.00"),
        ("100400",      "178",  "32","Completed","09/17/2022 12:00:00","12.00"),
        ("100565",      "265",  "2", "Completed","09/27/2022 12:00:00","8.70"),
```

```
]


columns2 = ["user_id",  "city", "email",        "signup_date"]
data2 = [
        ("111", "San Francisco",        "rrok10@gmail.com",     "08/03/2021 12:00:00"),
        ("148", "Boston",       "sailor9820@gmail.com", "08/20/2021 12:00:00"),
        ("178", "San Francisco",        "harrypotterfan182@gmail.com",  "01/05/2022 12:00:00"),
        ("265", "Denver",       "shadower_@hotmail.com",        "02/26/2022 12:00:00"),
        ("300", "San Francisco",        "houstoncowboy1122@hotmail.com",        "06/30/2022 12:00:00"),
]
rdd1 = sc.parallelize(data1)
rdd2 = sc.parallelize(data2)
df1 = rdd1.toDF(columns1)
df2 = rdd2.toDF(columns2)

df1.show()
df2.show()

df1.createOrReplaceTempView("df1")
df2.createOrReplaceTempView("df2")
spark.sql("""
        select
                city,
                        count(*) as count_orders
        from
            df1
        join
            df2
        on
            df1.user_id = df2.user_id
        where
                df1.status='Completed'
        group by
                city
        order by
                count_orders desc
        limit
                3
""").show()

result_df = df1.join(df2, "user_id" ) \
        .filter(df1["status"] == 'Completed') \
```

```
        .groupBy("city") \
        .agg(count("*").alias("count_orders")) \
        .orderBy(col("count_orders").desc()) \
        .limit(3)

# Show the result
result_df.show()
```

```
+--------+-------+--------+---------+-------------------+-----+
|order_id|user_id|quantity|   status|               date|price|
+--------+-------+--------+---------+-------------------+-----+
|  100101|    111|      10|Cancelled|08/17/2022 12:00:00| 9.80|
|  100102|    111|      10|Completed|08/17/2022 12:00:00|10.00|
|  100259|    148|      35|Completed|08/25/2022 12:00:00| 5.10|
|  100264|    148|      40|Completed|08/26/2022 12:00:00| 4.80|
|  100305|    300|      15|Completed|09/05/2022 12:00:00|10.00|
|  100400|    178|      32|Completed|09/17/2022 12:00:00|12.00|
|  100565|    265|       2|Completed|09/27/2022 12:00:00| 8.70|
+--------+-------+--------+---------+-------------------+-----+

+-------+-------------+--------------------+-------------------+
|user_id|         city|               email|        signup_date|
+-------+-------------+--------------------+-------------------+
|    111|San Francisco|    rrok10@gmail.com|08/03/2021 12:00:00|
|    148|       Boston|sailor9820@gmail.com|08/20/2021 12:00:00|
|    178|San Francisco|harrypotterfan182...|01/05/2022 12:00:00|
|    265|       Denver|shadower_@hotmail...|02/26/2022 12:00:00|
|    300|San Francisco|houstoncowboy1122...|06/30/2022 12:00:00|
+-------+-------------+--------------------+-------------------+

+-------------+------------+
|         city|count_orders|
+-------------+------------+
|San Francisco|           3|
|       Boston|           2|
|       Denver|           1|
+-------------+------------+

+-------------+------------+
|         city|count_orders|
+-------------+------------+
|San Francisco|           3|
|       Boston|           2|
|       Denver|           1|
+-------------+------------+
```

10. Given the reviews table, write a query to retrieve the average star rating for each product, grouped by month. The output should display the month as a numerical value, product ID, and average star rating rounded to two decimal places. Sort the output first by month and then by product ID.

In [11]:

```
columns= [ "review_id", "user_id",      "submit_date",  "product_id",   "stars" ]
data = [
        ( "6171",       "123",  "06/08/2022 00:00:00",  "50001",        "4" ),
        ( "7802",       "265",  "06/10/2022 00:00:00",  "69852","4" ),
        ( "5293",       "362",  "06/18/2022 00:00:00",  "50001","3" ),
        ( "6352",       "192",  "07/26/2022 00:00:00",  "69852","3"),
        ( '4517',       "981",  "07/05/2022 00:00:00",  "69852","2" ),
]

rdd = sc.parallelize(data)
df = rdd.toDF(columns)
df = df.withColumn("submit_date", to_timestamp("submit_date", "MM/dd/yyyy HH:mm:ss"))
df.show()
df.printSchema()
df.createOrReplaceTempView("df")
spark.sql("""
        select
                month(submit_date) as month,
                product_id,
                avg(stars) as avg
        from
                df
        group by
                month(submit_date),
                product_id
        order by
                month,
                product_id
""").show()

x = (df
         .withColumn("month", month("submit_date"))
         .groupBy("month", "product_id")
         .agg(avg("stars").alias("avg"))
        .withColumn("avg", round("avg",2))
         .orderBy("month", "product_id")
         )
x.show()
```

```
+---------+-------+-------------------+----------+-----+
|review_id|user_id|        submit_date|product_id|stars|
+---------+-------+-------------------+----------+-----+
|     6171|    123|2022-06-08 00:00:00|     50001|    4|
|     7802|    265|2022-06-10 00:00:00|     69852|    4|
|     5293|    362|2022-06-18 00:00:00|     50001|    3|
|     6352|    192|2022-07-26 00:00:00|     69852|    3|
|     4517|    981|2022-07-05 00:00:00|     69852|    2|
+---------+-------+-------------------+----------+-----+

root
 |-- review_id: string (nullable = true)
 |-- user_id: string (nullable = true)
 |-- submit_date: timestamp (nullable = true)
 |-- product_id: string (nullable = true)
 |-- stars: string (nullable = true)

+-----+----------+---+
|month|product_id|avg|
+-----+----------+---+
|    6|     50001|3.5|
|    6|     69852|4.0|
|    7|     69852|2.5|
+-----+----------+---+

+-----+----------+---+
|month|product_id|avg|
+-----+----------+---+
|    6|     50001|3.5|
|    6|     69852|4.0|
|    7|     69852|2.5|
+-----+----------+---+
```

11. This is the same question as problem #1 in the SQL Chapter of Ace the Data Science Interview! Assume you have an events table on Facebook app analytics. Write a query to calculate the click-through rate (CTR) for the app in 2022 and round the results to 2 decimal places. Definition and note:Percentage of click-through rate (CTR) = 100.0 * Number of clicks / Number of impressions To avoid integer division, multiply the CTR by 100.0, not 100.

In [12]:
```
columns = [ "app_id",   "event_type",   "timestamp" ]
data = [
```

```
        ( "123",        "impression",   "07/18/2022 11:36:12" ),
        ( "123",        "impression",   "07/18/2022 11:37:12" ),
        ( "123",        "click",        "07/18/2022 11:37:42" ),
        ( "234",        "impression",   "07/18/2022 14:15:12" ),
        ( "234",        "click",        "07/18/2022 14:16:12" ),
]

rdd = sc.parallelize(data)
df = rdd.toDF(columns)
df = df.withColumn("timestamp", to_timestamp("timestamp", "MM/dd/yyyy HH:mm:ss"))
df.printSchema()
df.show()

df.groupBy("app_id").agg(
    (
        100.0
        * sum(when(col("event_type") == "click", 1).otherwise(0))
        / sum(when(col("event_type") == "impression", 1).otherwise(0))
    ).alias("click_through_rate")
).show()
```

```
root
 |-- app_id: string (nullable = true)
 |-- event_type: string (nullable = true)
 |-- timestamp: timestamp (nullable = true)

+------+----------+-------------------+
|app_id|event_type|          timestamp|
+------+----------+-------------------+
|   123|impression|2022-07-18 11:36:12|
|   123|impression|2022-07-18 11:37:12|
|   123|     click|2022-07-18 11:37:42|
|   234|impression|2022-07-18 14:15:12|
|   234|     click|2022-07-18 14:16:12|
+------+----------+-------------------+

+------+------------------+
|app_id|click_through_rate|
+------+------------------+
|   234|             100.0|
|   123|              50.0|
+------+------------------+
```

12. Assume you're given tables with information about TikTok user sign-ups and confirmations through email and text. New users on TikTok sign up using their email addresses, and upon sign-up, each user receives a text message confirmation to activate their account. Write a query to display the user IDs of those who did not confirm their sign-up on the first day, but confirmed on the second day. Definition: action_date refers to the date when users activated their accounts and confirmed their sign-up through text messages.

In [13]:
```
columns1 = [ "email_id",        "user_id",      "signup_date"]
data = [
        ( "125",        "7771", "06/14/2022 00:00:00" ),
        ( "433",        "1052", "07/09/2022 00:00:00" ),
]

cols2=[ "text_id",      "email_id",     "signup_action",        "action_date" ]
data2 = [
        ( "6878",       "125",  "Confirmed",    "06/14/2022 00:00:00" ),
        ( "6997",       "433","Not Confirmed","07/09/2022 00:00:00" ),
        ( "7000",       "433","Confirmed","07/10/2022 00:00:00" ),
]
rdd1= sc.parallelize( data )
rdd2= sc.parallelize( data2 )
df1 = rdd1.toDF(columns1)
df2 = rdd2.toDF(cols2)

df1 =df1.withColumn("signup_date", to_timestamp("signup_date", "MM/dd/yyyy HH:mm:ss"))
df2=df2.withColumn("action_date", to_timestamp("action_date", "MM/dd/yyyy HH:mm:ss"))
df1.createOrReplaceTempView("emails")
df2.createOrReplaceTempView("texts")
df1.show()
df1.printSchema()
df2.show()
df2.printSchema()
spark.sql("""
        SELECT
                DISTINCT emails.user_id
        FROM
                emails
        INNER JOIN
                texts
        ON
                emails.email_id = texts.email_id
        WHERE
                DATE(texts.action_date) = DATE_ADD(DATE(emails.signup_date), 1)
```

```
                AND
                texts.signup_action = 'Confirmed';
""").show()

res = (df1
           .join(df2, "email_id", "inner")
           .filter(
                       (df2.action_date == date_add(df1.signup_date, 1))
                               &
                       (df2.signup_action=='Confirmed')
                )
           .select("user_id")
           .distinct())
res.show()
```

```
+--------+-------+-------------------+
|email_id|user_id|        signup_date|
+--------+-------+-------------------+
|     125|   7771|2022-06-14 00:00:00|
|     433|   1052|2022-07-09 00:00:00|
+--------+-------+-------------------+

root
 |-- email_id: string (nullable = true)
 |-- user_id: string (nullable = true)
 |-- signup_date: timestamp (nullable = true)

+-------+--------+-------------+-------------------+
|text_id|email_id|signup_action|        action_date|
+-------+--------+-------------+-------------------+
|   6878|     125|    Confirmed|2022-06-14 00:00:00|
|   6997|     433|Not Confirmed|2022-07-09 00:00:00|
|   7000|     433|    Confirmed|2022-07-10 00:00:00|
+-------+--------+-------------+-------------------+

root
 |-- text_id: string (nullable = true)
 |-- email_id: string (nullable = true)
 |-- signup_action: string (nullable = true)
 |-- action_date: timestamp (nullable = true)

+-------+
|user_id|
+-------+
|   1052|
+-------+

+-------+
|user_id|
+-------+
|   1052|
+-------+
```

13. IBM is analyzing how their employees are utilizing the Db2 database by tracking the SQL queries executed by their employees. The objective is to generate data to populate a histogram that shows the number of unique queries run by employees during the third quarter of 2023 (July to September). Additionally, it should count the number of employees who did not run any queries during this period. Display the number of unique queries as

histogram categories, along with the count of employees who executed that number of unique queries. need some iterations on the query need to figure out for the dsl and the spark sql

In [14]:
```
cols1 = [ "employee_id",        "query_id",     "query_starttime",      "execution_time" ]
data1 = [
        ( "3",  "856987",       "07/01/2023 03:25:12",  "2698" ),
        ( "3",  "286115",       "07/01/2023 04:34:38",  "2705" ),
        ( "3",  "33683","07/02/2023 10:55:14",  "91" ),
        ( "1",  "413477","07/15/2023 11:35:09", "470" ),
        ( "1", "421983", "07/01/2023 14:33:47", "3020" ),
        ( "2",  "17745","07/01/2023 14:33:47",  "2093" ),
        ( "2", "958745", "07/02/2023 08:11:45", "512" ),
        ( "2", "684293", "07/22/2023 18:42:31", "1630" ),
        ( "2", "385739", "07/25/2023 14:25:17", "240" ),
        ( "2", "123456", "07/26/2023 16:12:18", "950" ),
]


cols2 = [ "employee_id",        "full_name",    "gender" ]
data2 = [
        ( "1",  "Judas Beardon",        "Male" ),
        ( "2",  "Lainey Franciotti",    "Female" ),
        ( "3",  "Ashbey Strahan",       "Male" ),
]
rdd1 = sc.parallelize(data1)
rdd2 = sc.parallelize(data2)
df1 = rdd1.toDF(cols1)
df2 = rdd2.toDF(cols2)
df1 = df1.withColumn("query_starttime", to_timestamp("query_starttime","MM/dd/yyyy HH:mm:ss"))
df1.show()
df2.show()
df2.createOrReplaceTempView("employees")
df1.createOrReplaceTempView("queries")
spark.sql("""
WITH employee_queries AS (
  SELECT
    e.employee_id,
    COALESCE(COUNT(DISTINCT q.query_id), 0) AS unique_queries
  FROM employees AS e
  LEFT JOIN queries AS q
    ON e.employee_id = q.employee_id
      AND q.query_starttime >= '2023-07-01T00:00:00Z'
```

```
      AND q.query_starttime < '2023-10-01T00:00:00Z'
  GROUP BY e.employee_id
)

SELECT
  unique_queries,
  COUNT(employee_id) AS employee_count
FROM employee_queries
GROUP BY unique_queries
ORDER BY unique_queries;
""").show()


filtered_queries = df1.filter(
        (col("query_starttime") >= "2023-07-01") &
        (col("query_starttime") < "2023-10-01")
)

# Step 2: Perform the left join between employees and filtered queries
joined_df = df2.join(
        filtered_queries,
        df2["employee_id"] == filtered_queries["employee_id"],
        how="left"
)

# Step 3: Group by employee_id and count distinct query_id
employee_queries = joined_df.groupBy(df2["employee_id"]).agg(
        coalesce(countDistinct("query_id"), lit(0)).alias("unique_queries")
)

# Step 4: Group by unique_queries and count the number of employees in each category
result = employee_queries.groupBy("unique_queries").agg(
        count("employee_id").alias("employee_count")
).orderBy("unique_queries")

# Show the result
result.show()
```

```
+-----------+--------+-------------------+--------------+
|employee_id|query_id|    query_starttime|execution_time|
+-----------+--------+-------------------+--------------+
|          3|  856987|2023-07-01 03:25:12|          2698|
|          3|  286115|2023-07-01 04:34:38|          2705|
|          3|   33683|2023-07-02 10:55:14|            91|
|          1|  413477|2023-07-15 11:35:09|           470|
|          1|  421983|2023-07-01 14:33:47|          3020|
|          2|   17745|2023-07-01 14:33:47|          2093|
|          2|  958745|2023-07-02 08:11:45|           512|
|          2|  684293|2023-07-22 18:42:31|          1630|
|          2|  385739|2023-07-25 14:25:17|           240|
|          2|  123456|2023-07-26 16:12:18|           950|
+-----------+--------+-------------------+--------------+

+-----------+-----------------+------+
|employee_id|        full_name|gender|
+-----------+-----------------+------+
|          1|    Judas Beardon|  Male|
|          2|Lainey Franciotti|Female|
|          3|   Ashbey Strahan|  Male|
+-----------+-----------------+------+

+--------------+--------------+
|unique_queries|employee_count|
+--------------+--------------+
|             1|             1|
|             2|             1|
|             5|             1|
+--------------+--------------+

+--------------+--------------+
|unique_queries|employee_count|
+--------------+--------------+
|             2|             1|
|             3|             1|
|             5|             1|
+--------------+--------------+
```

14. Your team at JPMorgan Chase is preparing to launch a new credit card, and to gain some insights, you're analyzing how many credit cards were issued each month. Write a query that outputs the name of each credit card and the difference in the number of issued cards between the month with the

highest issuance cards and the lowest issuance. Arrange the results based on the largest disparity.

In [15]:
```
cols = ["card_name",    "issued_amount",        "issue_month",  "issue_year" ]
data = [
         ( "Chase Freedom Flex",        "55000",        "1",    "2021" ),
         ( "Chase Freedom Flex","60000",        "2",    "2021" ),
         ( "Chase Freedom Flex","65000",        "3",    "2021" ),
         ( "Chase Freedom Flex","70000","4",    "2021" ),
         ( "Chase Sapphire Reserve",    "170000",       "1",    "2021" ),
         ( "Chase Sapphire Reserve","175000",   "2",    "2021" ),
         ( "Chase Sapphire Reserve","180000",   "3",    "2021" ),
]


rdd = sc.parallelize(data)
df = rdd.toDF(cols)
df.show()
df.createOrReplaceTempView("df")

window_spec = Window.partitionBy("card_name").orderBy("issue_month")

res = (df
           .groupBy("card_name")
           .agg( (max("issued_amount")-min("issued_amount")).alias("difference"))
           .withColumn("difference", floor("difference"))
        )
res.show(truncate=False)


spark.sql("""
        select
                card_name,
                floor(max(issued_amount)-min(issued_amount)) as diff
        from
                df
        group by
                card_name
""").show(truncate=False)
```

```
+-------------------+------------+----------+----------+
|          card_name|issued_amount|issue_month|issue_year|
+-------------------+------------+----------+----------+
|  Chase Freedom Flex|       55000|         1|      2021|
|  Chase Freedom Flex|       60000|         2|      2021|
|  Chase Freedom Flex|       65000|         3|      2021|
|  Chase Freedom Flex|       70000|         4|      2021|
|Chase Sapphire Re...|      170000|         1|      2021|
|Chase Sapphire Re...|      175000|         2|      2021|
|Chase Sapphire Re...|      180000|         3|      2021|
+-------------------+------------+----------+----------+

+---------------------+----------+
|card_name            |difference|
+---------------------+----------+
|Chase Freedom Flex   |15000     |
|Chase Sapphire Reserve|10000     |
+---------------------+----------+

+---------------------+-----+
|card_name            |diff |
+---------------------+-----+
|Chase Freedom Flex   |15000|
|Chase Sapphire Reserve|10000|
+---------------------+-----+
```

15. You('re trying to find the mean number of items per order on Alibaba, rounded to 1 decimal place using 'tables which includes information on the count of items in each order (item_count table) and the ' 'corresponding number of orders for each item count (order_occurrences table).)

In [16]:
```
cols = [ "item_count",  "order_occurrences" ]
data = [
        ( "1",  "500" ),
        ( "2",  "1000") ,
        ( "3",  "800" ),
        ( "4", "1000" ),
]
rdd = sc.parallelize(data)
df = rdd.toDF(cols)
df.show()
df.createOrReplaceTempView("df")
spark.sql("""
```

```
        select
                round(sum(item_count * order_occurrences) / sum(order_occurrences),2) as mean
        from df
""").show()


(
        df
        .agg(
                round(
                        sum(df.item_count * df.order_occurrences) / sum(df.order_occurrences),2)
                                .alias("mean"))
        .show()
)
```

```
+----------+-----------------+
|item_count|order_occurrences|
+----------+-----------------+
|         1|              500|
|         2|             1000|
|         3|              800|
|         4|             1000|
+----------+-----------------+

+----+
|mean|
+----+
| 2.7|
+----+

+----+
|mean|
+----+
| 2.7|
+----+
```

16. CVS Health is trying to better understand its pharmacy sales, and how well different products are selling. Each drug can only be produced by one manufacturer. Write a query to find the top 3 most profitable drugs sold, and how much profit they made. Assume that there are no ties in the profits. Display the result from the highest to the lowest total profit. Definition: cogs stands for Cost of Goods Sold which is the direct cost associated with producing the drug. Total Profit = Total Sales - Cost of Goods Sold

In [17]:

```
cols = [ "product_id",  "units_sold",   "total_sales",  "cogs", "manufacturer", "drug" ]
data = [
        ( "9",  "37410",        "293452.54",    "208876.01",    "Eli Lilly",    "Zyprexa" ),
        ( "34", "94698",        "600997.19",    "521182.16",    "AstraZeneca",  "Surmontil" ),
        ( "61", "77023",        "500101.61",    "419174.97",    "Biogen",       "Varicose Relief" ),
        ( "136",        "144814",       "1084258",      "1006447.73","Biogen",  "Burkhart" ),
]


rdd = sc.parallelize(data)
df = rdd.toDF(cols)
df.show()
df.createOrReplaceTempView("df")
spark.sql("""
        select
                drug,
                total_sales - cogs as profits
        from
                df
        order by
                total_sales desc
        limit
                3
""").show()


df.withColumn("profits", df.total_sales.cast("double") - df.cogs.cast("double")) \
        .select("drug", "profits") \
        .orderBy(col("profits").desc()) \
        .limit(3) \
        .show()
```

```
+----------+----------+-----------+----------+------------+---------------+
|product_id|units_sold|total_sales|      cogs|manufacturer|           drug|
+----------+----------+-----------+----------+------------+---------------+
|         9|     37410|  293452.54| 208876.01|   Eli Lilly|        Zyprexa|
|        34|     94698|  600997.19| 521182.16| AstraZeneca|      Surmontil|
|        61|     77023|  500101.61| 419174.97|      Biogen|Varicose Relief|
|       136|    144814|    1084258|1006447.73|      Biogen|       Burkhart|
+----------+----------+-----------+----------+------------+---------------+

+---------------+-----------------+
|           drug|          profits|
+---------------+-----------------+
|      Surmontil|79815.02999999997|
|Varicose Relief|80926.64000000001|
|        Zyprexa|84576.52999999997|
+---------------+-----------------+

+---------------+-----------------+
|           drug|          profits|
+---------------+-----------------+
|        Zyprexa|84576.52999999997|
|Varicose Relief|80926.64000000001|
|      Surmontil|79815.02999999997|
+---------------+-----------------+
```

17. CVS Health is analyzing its pharmacy sales data, and how well different products are selling in the market. Each drug is exclusively manufactured by a single manufacturer. Write a query to identify the manufacturers associated with the drugs that resulted in losses for CVS Health and calculate the total amount of losses incurred. Output the manufacturer's name, the number of drugs associated with losses, and the total losses in absolute value. Display the results sorted in descending order with the highest losses displayed at the top.

In [18]:
```
cols = ["product_id", "units_sold", "total_sales", "cogs", "manufacturer", "drug"]
data = [
        (156, 89514, 3130097.00, 3427421.73, "Biogen", "Acyclovir"),
        (25, 222331, 2753546.00, 2974975.36, "AbbVie", "Lamivudine and Zidovudine"),
        (50, 90484, 2521023.73, 2742445.90, "Eli Lilly", "Dermasorb TA Complete Kit"),
        (98, 110746, 813188.82, 140422.87, "Biogen", "Medi-Chord")
]
rdd = sc.parallelize(data)
df = rdd.toDF(cols)
df.show()
```

```
df.createOrReplaceTempView("df")
spark.sql("""
        select
                manufacturer,
                count(drug) as drug_count,
                sum(cogs-total_sales) as total_loss
        from
                df
        where
                cogs > total_sales
        group by
                manufacturer
""").show()


( df
        .filter("cogs>total_sales")
        .groupby("manufacturer")
        .agg(
                sum(df.cogs-df.total_sales).alias("total_loss"),
                count("drug").alias("count")
        )
        .show()
)
```

```
+----------+----------+-----------+----------+------------+--------------------+
|product_id|units_sold|total_sales|      cogs|manufacturer|                drug|
+----------+----------+-----------+----------+------------+--------------------+
|       156|     89514|  3130097.0|3427421.73|      Biogen|           Acyclovir|
|        25|    222331|  2753546.0|2974975.36|      AbbVie|Lamivudine and Zi...|
|        50|     90484| 2521023.73| 2742445.9|   Eli Lilly|Dermasorb TA Comp...|
|        98|    110746|  813188.82| 140422.87|      Biogen|          Medi-Chord|
+----------+----------+-----------+----------+------------+--------------------+

+------------+----------+------------------+
|manufacturer|drug_count|        total_loss|
+------------+----------+------------------+
|      AbbVie|         1|221429.35999999987|
|      Biogen|         1|         297324.73|
|   Eli Lilly|         1|221422.16999999993|
+------------+----------+------------------+

+------------+------------------+-----+
|manufacturer|        total_loss|count|
+------------+------------------+-----+
|      AbbVie|221429.35999999987|    1|
|      Biogen|         297324.73|    1|
|   Eli Lilly|221422.16999999993|    1|
+------------+------------------+-----+
```

18. CVS Health wants to gain a clearer understanding of its pharmacy sales and the performance of various products. Write a query to calculate the total drug sales for each manufacturer. Round the answer to the nearest million and report your results in descending order of total sales. In case of any duplicates, sort them alphabetically by the manufacturer name. Since this data will be displayed on a dashboard viewed by business stakeholders, please format your results as follows: "$36 million".

In [19]:
```python
cols = ["product_id", "units_sold", "total_sales", "cogs", "manufacturer", "drug"]
data = [
        (94, 132362, 2041758.41, 1373721.70, "Biogen", "UP and UP"),
        (9, 37410, 293452.54, 208876.01, "Eli Lilly", "Zyprexa"),
        (50, 90484, 2521023.73, 2742445.90, "Eli Lilly", "Dermasorb"),
        (61, 77023, 500101.61, 419174.97, "Biogen", "Varicose Relief"),
        (136, 144814, 1084258.00, 1006447.73, "Biogen", "Burkhart")
]

df = spark.createDataFrame(data, cols)
```

```
df.show()
df.createOrReplaceTempView("df")
spark.sql("""
        select
                manufacturer,
                concat('$', floor(sum(total_sales)/1000000), ' million') as sales_mill
        from
                df
        group by
                manufacturer
        order by
                sum(total_sales) desc,
                manufacturer
""").show()


result_df = (
        df.groupBy("manufacturer")
        .agg(floor(sum("total_sales") / 1000000).alias("sales_mil"))  # Sum and convert to millions
        .select(
                col("manufacturer"),
                concat(lit("$"), col("sales_mil"), lit(" million")).alias("sales_mil")  # Format as required
        )
        .orderBy(col("sales_mil").desc(), col("manufacturer"))  # Order by sales_mil and manufacturer
)
result_df.show()
```

```
+----------+----------+-----------+----------+------------+---------------+
|product_id|units_sold|total_sales|      cogs|manufacturer|           drug|
+----------+----------+-----------+----------+------------+---------------+
|        94|    132362| 2041758.41| 1373721.7|      Biogen|      UP and UP|
|         9|     37410|  293452.54| 208876.01|   Eli Lilly|        Zyprexa|
|        50|     90484| 2521023.73| 2742445.9|   Eli Lilly|      Dermasorb|
|        61|     77023|  500101.61| 419174.97|      Biogen|Varicose Relief|
|       136|    144814|  1084258.0|1006447.73|      Biogen|       Burkhart|
+----------+----------+-----------+----------+------------+---------------+

+------------+----------+
|manufacturer|sales_mill|
+------------+----------+
|      Biogen|$3 million|
|   Eli Lilly|$2 million|
+------------+----------+

+------------+----------+
|manufacturer| sales_mil|
+------------+----------+
|      Biogen|$3 million|
|   Eli Lilly|$2 million|
+------------+----------+
```

19. UnitedHealth Group (UHG) has a program called Advocate4Me, which allows policy holders (or, members) to call an advocate and receive support for their health care needs – whether that's claims and benefits support, drug coverage, pre- and post-authorisation, medical records, emergency assistance, or member portal services. Write a query to find how many UHG policy holders made three, or more calls, assuming each call is identified by the case_id column.

In [20]:
```python
columns = ["policy_holder_id", "case_id", "call_category", "call_date", "call_duration_secs"]

# Create sample data
data = [
        (1, "f1d012f9-9d02-4966-a968-bf6c5bc9a9fe", "emergency assistance", "2023-04-13T19:16:53Z", 144),
        (1, "41ce8fb6-1ddd-4f50-ac31-07bfcce6aaab", "authorisation", "2023-05-25T09:09:30Z", 815),
        (2, "9b1af84b-eedb-4c21-9730-6f099cc2cc5e", "claims assistance", "2023-01-26T01:21:27Z", 992),
        (2, "8471a3d4-6fc7-4bb2-9fc7-4583e3638a9e", "emergency assistance", "2023-03-09T10:58:54Z", 128),
        (2, "38208fae-bad0-49bf-99aa-7842ba2e37bc", "benefits", "2023-06-05T07:35:43Z", 619)
]
```

```python
# Create DataFrame
df = spark.createDataFrame(data, schema=columns)
df.createOrReplaceTempView("df")
df.show()
spark.sql("""
        with call_records as (
                select
                        policy_holder_id,
                        count(case_id) as call_count
                from
                        df
                group by
                        policy_holder_id
                having
                        count(case_id)>=3
        )
        select
                count(policy_holder_id) as policy_holder_count
        from
                call_records
""").show()

res =df.groupby("policy_holder_id").agg(count("case_id").alias('call_count'))
print("the number of UHG policy holders with more than 3 or more calls is: ",res.filter(res.call_count >= 3).count())
```

```
+----------------+--------------------+--------------------+--------------------+------------------+
|policy_holder_id|             case_id|       call_category|           call_date|call_duration_secs|
+----------------+--------------------+--------------------+--------------------+------------------+
|               1|f1d012f9-9d02-496...|emergency assistance|2023-04-13T19:16:53Z|               144|
|               1|41ce8fb6-1ddd-4f5...|       authorisation|2023-05-25T09:09:30Z|               815|
|               2|9b1af84b-eedb-4c2...|   claims assistance|2023-01-26T01:21:27Z|               992|
|               2|8471a3d4-6fc7-4bb...|emergency assistance|2023-03-09T10:58:54Z|               128|
|               2|38208fae-bad0-49b...|            benefits|2023-06-05T07:35:43Z|               619|
+----------------+--------------------+--------------------+--------------------+------------------+

+--------------------+
|policy_holder_count|
+--------------------+
|                   1|
+--------------------+

the number of UHG policy holders with more than 3 or more calls is:  1
```

20. As a data analyst on the Oracle Sales Operations team, you are given a list of salespeople's deals, and the annual quota they need to hit. Write a query that outputs each employee id and whether they hit the quota or not ('yes' or 'no'). Order the results by employee id in ascending order. Definitions: deal_size: Deals acquired by a salesperson in the year. Each salesperson may have more than 1 deal. quota: Total annual quota for each salesperson.

In [21]:
```
data_quota = [
        (101, 500000),
        (201, 400000),
        (301, 600000)
]
columns_quota = ["employee_id", "quota"]
quotas = spark.createDataFrame(data_quota, columns_quota)

data_deal = [
        (101, 400000),
        (101, 300000),
        (201, 500000),
        (301, 500000)
]
columns_deal = ["employee_id", "deal_size"]

deals= spark.createDataFrame(data_deal, columns_deal)
quotas.createOrReplaceTempView("quotas")
deals.createOrReplaceTempView("deals")

spark.sql("""
        select
                deals.employee_id,
                case
                        when sum(deals.deal_size) > q.quota then 'yes'
                        else 'no'
                end as made_quota
        from
                deals
        join
                quotas q
        on
                deals.employee_id=q.employee_id
        group by
                deals.employee_id, q.quota
        order by
                deals.employee_id
```

```
""").show()


(deals
        .join(quotas,  "employee_id","inner")
        .groupBy("employee_id", "quota")
        .agg(when(
                        sum(deals.deal_size)> quotas.quota, 'yes')
                                                .otherwise('no').alias("made_quota")
        ).select("employee_id", "made_quota").show()
 )
```

```
+-----------+----------+
|employee_id|made_quota|
+-----------+----------+
|        101|       yes|
|        201|       yes|
|        301|        no|
+-----------+----------+

+-----------+----------+
|employee_id|made_quota|
+-----------+----------+
|        101|       yes|
|        201|       yes|
|        301|        no|
+-----------+----------+
```

21. Companies often perform salary analyses to ensure fair compensation practices. One useful analysis is to check if there are any employees earning more than their direct managers. As a HR Analyst, you're asked to identify all employees who earn more than their direct managers. The result should include the employee's ID and name.

In [22]:
```
data = [
        (1, "Emma Thompson", 3800, 1, 6),
        (2, "Daniel Rodriguez", 2230, 1, 7),
        (3, "Olivia Smith", 7000, 1, 8),
        (4, "Noah Johnson", 6800, 2, 9),
        (5, "Sophia Martinez", 1750, 1, 11),
        (6, "Liam Brown", 13000, 3, None),
        (7, "Ava Garcia", 12500, 3, None),
        (8, "William Davis", 6800, 2, None)
```

```
]

# Define the schema (column names)
columns = ["employee_id", "name", "salary", "department_id", "manager_id"]

# Create the DataFrame
df = spark.createDataFrame(data, columns)
df.createOrReplaceTempView("df")
df.show()

spark.sql("""
SELECT
  emp.employee_id,
  emp.name as employee_name
from
        df mgr
INNER JOIN
        df emp
ON
        mgr.employee_id=emp.manager_id
where
        emp.salary>mgr.salary
""").show()


df.alias("mgr") \
        .join(df.alias("emp"), col("mgr.employee_id") == col("emp.manager_id")) \
        .filter(col("emp.salary") > col("mgr.salary")) \
        .select(col("emp.employee_id"), col("emp.name").alias("employee_name")) \
        .show()
```

```
+-----------+----------------+------+-------------+----------+
|employee_id|            name|salary|department_id|manager_id|
+-----------+----------------+------+-------------+----------+
|          1|   Emma Thompson|  3800|            1|         6|
|          2|Daniel Rodriguez|  2230|            1|         7|
|          3|    Olivia Smith|  7000|            1|         8|
|          4|    Noah Johnson|  6800|            2|         9|
|          5| Sophia Martinez|  1750|            1|        11|
|          6|      Liam Brown| 13000|            3|      NULL|
|          7|      Ava Garcia| 12500|            3|      NULL|
|          8|   William Davis|  6800|            2|      NULL|
+-----------+----------------+------+-------------+----------+

+-----------+-------------+
|employee_id|employee_name|
+-----------+-------------+
|          3| Olivia Smith|
+-----------+-------------+

+-----------+-------------+
|employee_id|employee_name|
+-----------+-------------+
|          3| Olivia Smith|
+-----------+-------------+
```

22.Assume you are given the table below on Uber transactions made by users. Write a query to obtain the third transaction of every user. Output the user id, spend and transaction date.

In [23]:
```
data = [
        (111, 100.50, "01/08/2022 12:00:00"),
        (111, 55.00, "01/10/2022 12:00:00"),
        (121, 36.00, "01/18/2022 12:00:00"),
        (145, 24.99, "01/26/2022 12:00:00"),
        (111, 89.60, "02/05/2022 12:00:00")
]
cols = ["user_id", "spend", "transaction_date"]
df = spark.createDataFrame(data, cols)
df.show()
df = df.withColumn("transaction_date", to_timestamp(df.transaction_date , "MM/dd/yyyy HH:mm:ss"))
df.createOrReplaceTempView("df")
spark.sql("""
        with cte as(
```

```
                select
                        user_id,
                        spend,
                        transaction_date,
                        row_number()over(partition by user_id order by transaction_date) as row_num
                from df
        )
        select
                user_id,
                spend,
                transaction_date
        from cte
        where row_num=3
""").show()

window_spec = Window.partitionBy("user_id").orderBy("transaction_date")
(
        df
        .withColumn("row_num", row_number().over(window_spec))
        .filter("row_num=3")
        .select("user_id", "spend", "transaction_date")
        .show()
)
```

```
+-------+-----+-------------------+
|user_id|spend|   transaction_date|
+-------+-----+-------------------+
|    111|100.5|01/08/2022 12:00:00|
|    111| 55.0|01/10/2022 12:00:00|
|    121| 36.0|01/18/2022 12:00:00|
|    145|24.99|01/26/2022 12:00:00|
|    111| 89.6|02/05/2022 12:00:00|
+-------+-----+-------------------+

+-------+-----+-------------------+
|user_id|spend|   transaction_date|
+-------+-----+-------------------+
|    111| 89.6|2022-02-05 12:00:00|
+-------+-----+-------------------+

+-------+-----+-------------------+
|user_id|spend|   transaction_date|
+-------+-----+-------------------+
|    111| 89.6|2022-02-05 12:00:00|
+-------+-----+-------------------+
```

23. Imagine you're an HR analyst at a tech company tasked with analyzing employee salaries. Your manager is keen on understanding the pay distribution and asks you to determine the second highest salary among all employees. It's possible that multiple employees may share the same second highest salary. In case of duplicate, display the salary only once.

In [24]:
```
data = [
        (1, "Emma Thompson", 3800, 1, 6),
        (2, "Daniel Rodriguez", 2230, 1, 7),
        (3, "Olivia Smith", 2000, 1, 8)
]

# Define the schema (column names)
columns = ["employee_id", "name", "salary", "department_id", "manager_id"]
df = spark.createDataFrame(data, columns)
df.createOrReplaceTempView("df")
df.show()
window_spec = Window.orderBy(col("salary").desc())

df.withColumn("rank", dense_rank().over(window_spec)).filter("rank=2").select("salary").show()
```

```
spark.sql("""
        with rank as(
                select
                        employee_id, salary,
                        dense_rank() over (order by salary desc) as rank
                from df
        )
        select
                salary
        from rank
        where rank=2
""").show()
```

```
+-----------+----------------+------+-------------+----------+
|employee_id|            name|salary|department_id|manager_id|
+-----------+----------------+------+-------------+----------+
|          1|   Emma Thompson|  3800|            1|         6|
|          2|Daniel Rodriguez|  2230|            1|         7|
|          3|    Olivia Smith|  2000|            1|         8|
+-----------+----------------+------+-------------+----------+

+------+
|salary|
+------+
|  2230|
+------+

+------+
|salary|
+------+
|  2230|
+------+
```

24. Assume you're given tables with information on Snapchat users, including their ages and time spent sending and opening snaps. Write a query to obtain a breakdown of the time spent sending vs. opening snaps as a percentage of total time spent on these activities grouped by age group. Round the percentage to 2 decimal places in the output. Notes: Calculate the following percentages: time spent sending / (Time spent sending + Time spent opening) Time spent opening / (Time spent sending + Time spent opening) To avoid integer division in percentages, multiply by 100.0 and not 100.

In [25]:
```
cols = [ "activity_id", "user_id", "activity_type", "time_spent", "activity_date"]
cols2 = ["user_id", "age_bucket"]
activity_data = [
```

```
        (7274, 123, "open", 4.50, "06/22/2022 12:00:00"),
        (2425, 123, "send", 3.50, "06/22/2022 12:00:00"),
        (1413, 456, "send", 5.67, "06/23/2022 12:00:00"),
        (1414, 789, "chat", 11.00, "06/25/2022 12:00:00"),
        (2536, 456, "open", 3.00, "06/25/2022 12:00:00")
]

age_data = [
        (123, "31-35"),
        (456, "26-30"),
        (789, "21-25")
]

df1 = spark.createDataFrame(activity_data, cols)
df2 = spark.createDataFrame(age_data, cols2)
df1.show()
df2.show()


df = df2.join(df1, "user_id","left")
df.createOrReplaceTempView("df")

res = (
df
.groupBy("age_bucket")
.agg(
    sum(
        when(col("activity_type").isin("chat","open", "send"), col("time_spent"))
        .otherwise(0)).alias("total_time"),
    sum(
        when(col("activity_type").isin("chat","open"), col("time_spent"))
        .otherwise(0)).alias("time_open"),
    sum(
        when(col("activity_type").isin("send"), col("time_spent"))
        .otherwise(0)).alias("time_send")
))
res.show()
dff = (res
          .withColumn("open_perc", (res.time_open/res.total_time) * 100.0)
          .withColumn("send_perc", (res.time_send/res.total_time) * 100.0)
)
dff.show()
spark.sql("""
```

```
        with time as (
                select
                        age_bucket,
                        sum(time_spent) as total_time,
                        sum( case
                                        when activity_type in ( 'chat', 'open' ) then time_spent
                                        else 0
                                        end
                        ) as time_open,
                        sum( case
                                        when activity_type in ( 'send' ) then time_spent
                                        else 0
                                        end
                        ) as time_send
                from
                        df
                group by
                        age_bucket
        )
        select
                age_bucket,
                (time_open/total_time) * 100.0 as open_perc,
                (time_send/total_time) * 100.0 as spend_per
        from time
""").show()
```

```
+-----------+-------+-------------+----------+-------------------+
|activity_id|user_id|activity_type|time_spent|      activity_date|
+-----------+-------+-------------+----------+-------------------+
|       7274|    123|         open|       4.5|06/22/2022 12:00:00|
|       2425|    123|         send|       3.5|06/22/2022 12:00:00|
|       1413|    456|         send|      5.67|06/23/2022 12:00:00|
|       1414|    789|         chat|      11.0|06/25/2022 12:00:00|
|       2536|    456|         open|       3.0|06/25/2022 12:00:00|
+-----------+-------+-------------+----------+-------------------+

+-------+----------+
|user_id|age_bucket|
+-------+----------+
|    123|     31-35|
|    456|     26-30|
|    789|     21-25|
+-------+----------+

+----------+----------+---------+---------+
|age_bucket|total_time|time_open|time_send|
+----------+----------+---------+---------+
|     21-25|      11.0|     11.0|      0.0|
|     31-35|       8.0|      4.5|      3.5|
|     26-30|      8.67|      3.0|     5.67|
+----------+----------+---------+---------+

+----------+----------+---------+---------+------------------+-----------------+
|age_bucket|total_time|time_open|time_send|         open_perc|        send_perc|
+----------+----------+---------+---------+------------------+-----------------+
|     21-25|      11.0|     11.0|      0.0|             100.0|              0.0|
|     31-35|       8.0|      4.5|      3.5|             56.25|            43.75|
|     26-30|      8.67|      3.0|     5.67|34.602076124567475|65.39792387543253|
+----------+----------+---------+---------+------------------+-----------------+

+----------+------------------+-----------------+
|age_bucket|         open_perc|        spend_per|
+----------+------------------+-----------------+
|     21-25|             100.0|              0.0|
|     31-35|             56.25|            43.75|
|     26-30|34.602076124567475|65.39792387543253|
+----------+------------------+-----------------+
```

25. Given a table of tweet data over a specified time period, calculate the 3-day rolling average of tweets for each user. Output the user ID, tweet date, and rolling averages rounded to 2 decimal places. Notes: A rolling average, also known as a moving average or running mean is a time-series technique that examines trends in data over a specified period of time. In this case, we want to determine how the tweet count for each user changes over a 3-day period.

In [26]:
```
data = [
        (111, "06/01/2022 00:00:00", 2),
        (111, "06/02/2022 00:00:00", 1),
        (111, "06/03/2022 00:00:00", 3),
        (111, "06/04/2022 00:00:00", 4),
        (111, "06/05/2022 00:00:00", 5),
        (114, "06/03/2022 00:00:00", 3),
        (114, "06/04/2022 00:00:00", 4),
        (114, "06/05/2022 00:00:00", 5)
]
cols = ["user_id", "tweet_date", "tweet_count"]
df = spark.createDataFrame(data, cols)

df.show(truncate=False)
df.createOrReplaceTempView("df")
spark.sql("""
        select
                user_id,
                tweet_date,
                round(avg(tweet_count) over(partition by user_id order by tweet_date rows between 2 preceding and current row),2)
        from
                df
""").show()
print("with spark DSL")
window_spec = Window.partitionBy("user_id").orderBy("tweet_date").rowsBetween(-2,0)
df.withColumn("rolling_avg_3_days", avg("tweet_count").over(window_spec)).show()
```

```
+-------+-------------------+-----------+
|user_id|tweet_date         |tweet_count|
+-------+-------------------+-----------+
|111    |06/01/2022 00:00:00|2          |
|111    |06/02/2022 00:00:00|1          |
|111    |06/03/2022 00:00:00|3          |
|111    |06/04/2022 00:00:00|4          |
|111    |06/05/2022 00:00:00|5          |
|114    |06/03/2022 00:00:00|3          |
|114    |06/04/2022 00:00:00|4          |
|114    |06/05/2022 00:00:00|5          |
+-------+-------------------+-----------+

+-------+-------------------+------------------+
|user_id|         tweet_date|rolling_avg_3_days|
+-------+-------------------+------------------+
|    111|06/01/2022 00:00:00|               2.0|
|    111|06/02/2022 00:00:00|               1.5|
|    111|06/03/2022 00:00:00|               2.0|
|    111|06/04/2022 00:00:00|              2.67|
|    111|06/05/2022 00:00:00|               4.0|
|    114|06/03/2022 00:00:00|               3.0|
|    114|06/04/2022 00:00:00|               3.5|
|    114|06/05/2022 00:00:00|               4.0|
+-------+-------------------+------------------+

with spark DSL
+-------+-------------------+-----------+------------------+
|user_id|         tweet_date|tweet_count|rolling_avg_3_days|
+-------+-------------------+-----------+------------------+
|    111|06/01/2022 00:00:00|          2|               2.0|
|    111|06/02/2022 00:00:00|          1|               1.5|
|    111|06/03/2022 00:00:00|          3|               2.0|
|    111|06/04/2022 00:00:00|          4|2.6666666666666665|
|    111|06/05/2022 00:00:00|          5|               4.0|
|    114|06/03/2022 00:00:00|          3|               3.0|
|    114|06/04/2022 00:00:00|          4|               3.5|
|    114|06/05/2022 00:00:00|          5|               4.0|
+-------+-------------------+-----------+------------------+
```

26. Assume you're given a table containing data on Amazon customers and their spending on products in different category, write a query to identify the top two highest-grossing products within each category in the year 2022. The output should include the category, product, and total spend.

In [27]:
```
data = [
        ("appliance", "refrigerator", 165, 246.00, "2021-12-26 12:00:00"),
        ("appliance", "refrigerator", 123, 299.99, "2022-03-02 12:00:00"),
        ("appliance", "washing machine", 123, 219.80, "2022-03-02 12:00:00"),
        ("electronics", "vacuum", 178, 152.00, "2022-04-05 12:00:00"),
        ("electronics", "wireless headset", 156, 249.90, "2022-07-08 12:00:00"),
        ("electronics", "vacuum", 145, 189.00, "2022-07-15 12:00:00")
]
cols = ["category",     "product",      "user_id",      "spend",        "transaction_date"]
# Create DataFrame
df = spark.createDataFrame(data, cols)
df=df.withColumn("transaction_date", to_timestamp("transaction_date", "yyyy-MM-dd HH:mm:ss"))
df.show()

df.createOrReplaceTempView("df")
spark.sql("""
        with rank as(
                select
                        category,
                        product,
                        sum(spend) as total_spend,
                        rank() over(partition by category order by sum(spend) desc) as rank
                from
                        df
                where
                        year(transaction_date)=2022
                group by
                        category,
                        product
        )
        select
                category,
                product,
                total_spend
        from rank
        where rank <=2
        order by category, rank
""").show()
```

```
window_spec = Window.partitionBy("category").orderBy(col("total_spend").desc())

m1 = df.withColumn("year", year("transaction_date"))
m2 = (m1
          .filter(col("year")==2022)
          .groupby("category", "product")
          .agg(sum("spend").alias("total_spend"))
          .withColumn("rank", rank().over(window_spec))
          .filter(col("rank")<=2)
          .orderBy("category", "rank")
          .select("category", "product", "total_spend")
        )
m2.show()
```

```
+-----------+---------------+-------+------+-------------------+
|   category|        product|user_id| spend|   transaction_date|
+-----------+---------------+-------+------+-------------------+
|  appliance|   refrigerator|    165| 246.0|2021-12-26 12:00:00|
|  appliance|   refrigerator|    123|299.99|2022-03-02 12:00:00|
|  appliance|washing machine|    123| 219.8|2022-03-02 12:00:00|
|electronics|         vacuum|    178| 152.0|2022-04-05 12:00:00|
|electronics|wireless headset|    156| 249.9|2022-07-08 12:00:00|
|electronics|         vacuum|    145| 189.0|2022-07-15 12:00:00|
+-----------+---------------+-------+------+-------------------+

+-----------+---------------+-----------+
|   category|        product|total_spend|
+-----------+---------------+-----------+
|  appliance|   refrigerator|     299.99|
|  appliance|washing machine|      219.8|
|electronics|         vacuum|      341.0|
|electronics|wireless headset|      249.9|
+-----------+---------------+-----------+

+-----------+---------------+-----------+
|   category|        product|total_spend|
+-----------+---------------+-----------+
|  appliance|   refrigerator|     299.99|
|  appliance|washing machine|      219.8|
|electronics|         vacuum|      341.0|
|electronics|wireless headset|      249.9|
+-----------+---------------+-----------+
```

27. As part of an ongoing analysis of salary distribution within the company, your manager has requested a report identifying high earners in each department. A 'high earner' within a department is defined as an employee with a salary ranking among the top three salaries within that department. You're tasked with identifying these high earners across all departments. Write a query to display the employee's name along with their department name and salary. In case of duplicates, sort the results of department name in ascending order, then by salary in descending order. If multiple employees have the same salary, then order them alphabetically. Note: Ensure to utilize the appropriate ranking window function to handle duplicate salaries effectively.

In [28]:
```
employee_data = [
        (1, "Emma Thompson", 3800, 1, 6),
        (2, "Daniel Rodriguez", 2230, 1, 7),
        (3, "Olivia Smith", 2000, 1, 8),
        (4, "Noah Johnson", 6800, 2, 9),
```

```
        (5, "Sophia Martinez", 1750, 1, 11),
        (6, "Liam Brown", 13000, 3, None),
        (7, "Ava Garcia", 12500, 3, None),
        (8, "William Davis", 6800, 2, None),
        (9, "Isabella Wilson", 11000, 3, None),
        (10, "James Anderson", 4000, 1, 11)
]
cols = ["employee_id","employee_name","salary", "department_id", "manager_id"]
cols2 = ["department_id", "department_name"]

# Department data
department_data = [
        (1, "Data Analytics"),
        (2, "Data Science")
]

df1 = spark.createDataFrame(employee_data, cols)
df2 = spark.createDataFrame(department_data, cols2)
df1.show()
df2.show()
df1.createOrReplaceTempView("df1")
df2.createOrReplaceTempView("df2")


res = df1.join(df2, "department_id", 'left')
window_spec = Window.partitionBy("department_id").orderBy(col("salary").desc(), col("employee_name"))
res = (
        res
        .withColumn("rank", dense_rank().over(window_spec))
        .filter("department_id!=3")
        .select("department_name", "employee_name", "salary")
        .filter("rank<=3")
        )
res.show()

spark.sql("""
        with rank as(
                select
                        df1.department_id,
                        df1.salary,
                        department_name,
                        employee_name,
                        dense_rank()over(partition by df1.department_id order by df1.salary desc, df1.employee_name) as rank
```

```
                from
                        df1
                left join
                        df2
                on
                        df1.department_id=df2.department_id
                where
                        df1.department_id!=3
        )
        select
                department_name,
                employee_name,
                salary
        from
                rank
        where
                rank<=3
""").show()
```

```
+-----------+---------------+------+-------------+----------+
|employee_id|   employee_name|salary|department_id|manager_id|
+-----------+---------------+------+-------------+----------+
|          1|   Emma Thompson|  3800|            1|         6|
|          2|Daniel Rodriguez|  2230|            1|         7|
|          3|    Olivia Smith|  2000|            1|         8|
|          4|    Noah Johnson|  6800|            2|         9|
|          5| Sophia Martinez|  1750|            1|        11|
|          6|      Liam Brown| 13000|            3|      NULL|
|          7|      Ava Garcia| 12500|            3|      NULL|
|          8|   William Davis|  6800|            2|      NULL|
|          9| Isabella Wilson| 11000|            3|      NULL|
|         10|  James Anderson|  4000|            1|        11|
+-----------+---------------+------+-------------+----------+

+-------------+---------------+
|department_id|department_name|
+-------------+---------------+
|            1| Data Analytics|
|            2|   Data Science|
+-------------+---------------+

+---------------+----------------+------+
|department_name|   employee_name|salary|
+---------------+----------------+------+
| Data Analytics|  James Anderson|  4000|
| Data Analytics|   Emma Thompson|  3800|
| Data Analytics|Daniel Rodriguez|  2230|
|   Data Science|    Noah Johnson|  6800|
|   Data Science|   William Davis|  6800|
+---------------+----------------+------+

+---------------+----------------+------+
|department_name|   employee_name|salary|
+---------------+----------------+------+
| Data Analytics|  James Anderson|  4000|
| Data Analytics|   Emma Thompson|  3800|
| Data Analytics|Daniel Rodriguez|  2230|
|   Data Science|    Noah Johnson|  6800|
|   Data Science|   William Davis|  6800|
+---------------+----------------+------+
```

28. Assume there are three Spotify tables: artists, songs, and global_song_rank, which contain information about the artists, songs, and music charts, respectively. Write a query to find the top 5 artists whose songs appear most frequently in the Top 10 of the global_song_rank table. Display the top 5 artist names in ascending order, along with their song appearance ranking. If two or more artists have the same number of song appearances, they should be assigned the same ranking, and the rank numbers should be continuous (i.e. 1, 2, 2, 3, 4, 5).

In [29]:
```
artists_data = [
        (101, "Ed Sheeran", "Warner Music Group"),
        (120, "Drake", "Warner Music Group"),
        (125, "Bad Bunny", "Rimas Entertainment")
]

# Songs data
songs_data = [
        (55511, 101, "Perfect"),
        (45202, 101, "Shape of You"),
        (22222, 120, "One Dance"),
        (19960, 120, "Hotline Bling")
]

# Global song rank data
global_song_rank_data = [
        (1, 45202, 5),
        (3, 45202, 2),
        (1, 19960, 3),
        (9, 19960, 15)
]

# Create DataFrames without specifying schema
artists_df = spark.createDataFrame(artists_data, ["artist_id", "artist_name", "label_owner"])
songs_df = spark.createDataFrame(songs_data, ["song_id", "artist_id", "name"])
global_song_rank_df = spark.createDataFrame(global_song_rank_data, ["day", "song_id", "rank"])

# Show DataFrames
artists_df.show()
songs_df.show()
global_song_rank_df.show()
artists_df.createOrReplaceTempView("artists")
songs_df.createOrReplaceTempView("songs")
global_song_rank_df.createOrReplaceTempView("ranking")

window_spec = Window.orderBy(col("song_count").desc())
```

```python
# Perform the join and aggregation
top_10_df = (artists_df
                        .join(songs_df, "artist_id", "inner")
                        .join(global_song_rank_df, "song_id", "inner")
                        .filter(global_song_rank_df.rank <= 10)
                        .groupBy(artists_df.artist_name)
                        .agg(count("song_id").alias("song_count"))
                        .withColumn("artist_rank", dense_rank().over(window_spec))
                        )
top_10_df.show()
final_result = top_10_df.filter(top_10_df.artist_rank <= 5).select("artist_name", "artist_rank")
final_result.show()

spark.sql("""
        WITH top_10_cte as (
          SELECT
                artists.artist_name,
                dense_rank()over(ORDER BY count(songs.song_id) desc) as artist_rank
          from
                artists
          inner JOIN
                songs
          ON
                artists.artist_id = songs.artist_id
          inner JOIN
                ranking
          ON
                songs.song_id=ranking.song_id
          WHERE
                ranking.rank <=10
          GROUP BY
                artists.artist_name
        )

        SELECT
                artist_name,
                artist_rank
        from
                top_10_cte
        WHERE
                artist_rank<=5;
""").show()
```

```
+---------+-----------+------------------+
|artist_id|artist_name|       label_owner|
+---------+-----------+------------------+
|      101| Ed Sheeran| Warner Music Group|
|      120|      Drake| Warner Music Group|
|      125|  Bad Bunny|Rimas Entertainment|
+---------+-----------+------------------+

+-------+---------+-------------+
|song_id|artist_id|         name|
+-------+---------+-------------+
|  55511|      101|      Perfect|
|  45202|      101| Shape of You|
|  22222|      120|    One Dance|
|  19960|      120|Hotline Bling|
+-------+---------+-------------+

+---+-------+----+
|day|song_id|rank|
+---+-------+----+
|  1|  45202|   5|
|  3|  45202|   2|
|  1|  19960|   3|
|  9|  19960|  15|
+---+-------+----+

+-----------+----------+-----------+
|artist_name|song_count|artist_rank|
+-----------+----------+-----------+
| Ed Sheeran|         2|          1|
|      Drake|         1|          2|
+-----------+----------+-----------+

+-----------+-----------+
|artist_name|artist_rank|
+-----------+-----------+
| Ed Sheeran|          1|
|      Drake|          2|
+-----------+-----------+

+-----------+-----------+
|artist_name|artist_rank|
+-----------+-----------+
```

```
| Ed Sheeran|          1|
|      Drake|          2|
+-----------+-----------+
```

29. New TikTok users sign up with their emails. They confirmed their signup by replying to the text confirmation to activate their accounts. Users may receive multiple text messages for account confirmation until they have confirmed their new account. A senior analyst is interested to know the activation rate of specified users in the emails table. Write a query to find the activation rate. Round the percentage to 2 decimal places. Definitions: emails table contain the information of user signup details. texts table contains the users' activation information. Assumptions: The analyst is interested in the activation rate of specific users in the emails table, which may not include all users that could potentially be found in the texts table. For example, user 123 in the emails table may not be in the texts table and vice versa.

In [30]:
```
emails_data = [
        (125, 7771, "2022-06-14 00:00:00"),
        (236, 6950, "2022-07-01 00:00:00"),
        (433, 1052, "2022-07-09 00:00:00")
]

# Create emails DataFrame
emails_df = spark.createDataFrame(emails_data, ["email_id", "user_id", "signup_date"])

# Sample data for texts
texts_data = [
        (6878, 125, "Confirmed"),
        (6920, 236, "Not Confirmed"),
        (6994, 236, "Confirmed")
]

# Create texts DataFrame
texts_df = spark.createDataFrame(texts_data, ["text_id", "email_id", "signup_action"])

# Show DataFrames
emails_df.show()
texts_df.show()

emails_df.createOrReplaceTempView("emails")
texts_df.createOrReplaceTempView("texts")

spark.sql("""
SELECT
        ROUND(COUNT(texts.email_id)/COUNT(DISTINCT emails.email_id),2) AS activation_rate
```

```
FROM emails
LEFT JOIN texts
ON emails.email_id = texts.email_id
AND texts.signup_action = 'Confirmed';
""").show()
#


result_df = (
        emails_df
        .join(texts_df, emails_df.email_id == texts_df.email_id, "left")
        .agg(
                round(
                        count(when(texts_df.signup_action == 'Confirmed', texts_df.email_id)) / countDistinct(emails_df.email_id)
                ).alias("activation_rate")
        )
)

result_df.show()
```

```
+--------+-------+-------------------+
|email_id|user_id|        signup_date|
+--------+-------+-------------------+
|     125|   7771|2022-06-14 00:00:00|
|     236|   6950|2022-07-01 00:00:00|
|     433|   1052|2022-07-09 00:00:00|
+--------+-------+-------------------+

+-------+--------+-------------+
|text_id|email_id|signup_action|
+-------+--------+-------------+
|   6878|     125|    Confirmed|
|   6920|     236|Not Confirmed|
|   6994|     236|    Confirmed|
+-------+--------+-------------+

+---------------+
|activation_rate|
+---------------+
|           0.67|
+---------------+

+---------------+
|activation_rate|
+---------------+
|           0.67|
+---------------+
```

30. A Microsoft Azure Supercloud customer is defined as a customer who has purchased at least one product from every product category listed in the products table. Write a query that identifies the customer IDs of these Supercloud customers.

In [31]:
```python
customer_contracts_data = [
        (1, 1, 1000),
        (1, 6, 2000),
        (1, 5, 1500),
        (2, 2, 3000),
        (2, 6, 2000)
]

# Sample data for products
products_data = [
```

```
        (1, "Analytics", "Azure Databricks"),
        (2, "Analytics", "Azure Stream Analytics"),
        (4, "Containers", "Azure Kubernetes Service"),
        (5, "Containers", "Azure Service Fabric"),
        (6, "Compute", "Virtual Machines"),
        (7, "Compute", "Azure Functions")
]

# Create DataFrames
customer_contracts_df = spark.createDataFrame(customer_contracts_data, ["customer_id", "product_id", "amount"])
products_df = spark.createDataFrame(products_data, ["product_id", "product_category", "product_name"])

# Show DataFrames
customer_contracts_df.show()
products_df.show()
customer_contracts_df.createOrReplaceTempView("customer_contracts")
products_df.createOrReplaceTempView("products")
spark.sql("""
        WITH supercloud_cust AS (
                SELECT
                        customers.customer_id,
                        COUNT(DISTINCT products.product_category) AS product_count
                FROM
                        customer_contracts AS customers
                INNER JOIN
                        products
                ON
                        customers.product_id = products.product_id
                GROUP BY
                        customers.customer_id
        )
        SELECT
                customer_id
        FROM
                supercloud_cust
        WHERE
                product_count = (
                        SELECT
                                COUNT(DISTINCT product_category) FROM products
        );
""").show()
```

```
supercloud_cust_df = ( customer_contracts_df
  .join(products_df, "product_id", "inner")
  .groupBy("customer_id")
  .agg(count_distinct(products_df.product_category).alias("product_count"))
)


total_product_category_count = (products_df
                                                                .select(countDistinct("product_category").alias("total_pr
                                                                .collect()[0][0])

result_df = (
        supercloud_cust_df
        .filter(supercloud_cust_df.product_count == total_product_category_count)
        .select("customer_id")
)

result_df.show()
```

```
+-----------+----------+------+
|customer_id|product_id|amount|
+-----------+----------+------+
|          1|         1|  1000|
|          1|         6|  2000|
|          1|         5|  1500|
|          2|         2|  3000|
|          2|         6|  2000|
+-----------+----------+------+
```

```
+----------+---------------+--------------------+
|product_id|product_category|        product_name|
+----------+---------------+--------------------+
|         1|      Analytics|    Azure Databricks|
|         2|      Analytics|Azure Stream Anal...|
|         4|     Containers|Azure Kubernetes ...|
|         5|     Containers|Azure Service Fabric|
|         6|        Compute|    Virtual Machines|
|         7|        Compute|     Azure Functions|
+----------+---------------+--------------------+
```

```
+-----------+
|customer_id|
+-----------+
|          1|
+-----------+
```

```
+-----------+
|customer_id|
+-----------+
|          1|
+-----------+
```

31. This is the same question as problem #28 in the SQL Chapter of Ace the Data Science Interview! Assume you're given a table with measurement values obtained from a Google sensor over multiple days with measurements taken multiple times within each day. Write a query to calculate the sum of odd-numbered and even-numbered measurements separately for a particular day and display the results in two different columns. Refer to the Example Output below for the desired format. Definition: Within a day, measurements taken at 1st, 3rd, and 5th times are considered odd-numbered measurements, and measurements taken at 2nd, 4th, and 6th times are considered even-numbered measurements.

info you would want to assign a row number to the date then you can just %2 the results as odd sum and even sum

In [32]:

```
measurement_data = [
        (131233, 1109.51, "07/10/2022 09:00:00"),
        (135211, 1662.74, "07/10/2022 11:00:00"),
        (523542, 1246.24, "07/10/2022 13:15:00"),
        (143562, 1124.50, "07/11/2022 15:00:00"),
        (346462, 1234.14, "07/11/2022 16:45:00")
]

# Define the column names
columns = ["measurement_id", "measurement_value", "measurement_time"]
df = spark.createDataFrame(measurement_data, columns)
df = df.withColumn("measurement_time", to_date(df["measurement_time"], "MM/dd/yyyy HH:mm:ss"))
df.show()
df.createOrReplaceTempView("df")

window_spec = Window.partitionBy("measurement_time").orderBy(col("measurement_time").desc())
row_df = df.withColumn("row_num", row_number().over(window_spec))

row_df.groupBy("measurement_time").agg(
        sum(when(col("row_num") % 2 != 0, col("measurement_value")).otherwise(0)).alias("odd_sum"),
        sum(when(col("row_num") % 2 == 0, col("measurement_value")).otherwise(0)).alias("even_sum")
).show()

# info the key is to assign a row number and then % 2 that row number you will get the odd and even numbers sum
spark.sql("""
        with view as (
                select
                        measurement_time,
                        measurement_value,
                        row_number() over(partition by measurement_time order by measurement_time desc) as row_num
                from
                        df
        )
        select
                measurement_time,
                sum(case when row_num%2!=0 then measurement_value else 0 end) as odd_sum,
                sum(case when row_num%2=0 then measurement_value else 0 end) as even_sum
        from
                view
        group by
                measurement_time
""").show()
```

```
+--------------+-----------------+----------------+
|measurement_id|measurement_value|measurement_time|
+--------------+-----------------+----------------+
|        131233|          1109.51|      2022-07-10|
|        135211|          1662.74|      2022-07-10|
|        523542|          1246.24|      2022-07-10|
|        143562|           1124.5|      2022-07-11|
|        346462|          1234.14|      2022-07-11|
+--------------+-----------------+----------------+

+----------------+-------+--------+
|measurement_time|odd_sum|even_sum|
+----------------+-------+--------+
|      2022-07-10|2355.75| 1662.74|
|      2022-07-11| 1124.5| 1234.14|
+----------------+-------+--------+

+----------------+-------+--------+
|measurement_time|odd_sum|even_sum|
+----------------+-------+--------+
|      2022-07-10|2355.75| 1662.74|
|      2022-07-11| 1124.5| 1234.14|
+----------------+-------+--------+
```

32. Assume you're given a table on Walmart user transactions. Based on their most recent transaction date, write a query that retrieve the users along with the number of products they bought. Output the user's most recent transaction date, user ID, and the number of products, sorted in chronological order by the transaction date. Starting from November 10th, 2022, the official solution was updated, and the expected output of transaction date, number of users, and number of products was changed to the current expected output.

In [33]:
```python
data = [
        (3673, 123, 68.90, "07/08/2022 12:00:00"),
        (9623, 123, 274.10, "07/08/2022 12:00:00"),
        (1467, 115, 19.90, "07/08/2022 12:00:00"),
        (2513, 159, 25.00, "07/08/2022 12:00:00"),
        (1452, 159, 74.50, "07/10/2022 12:00:00")
]

# Define the column names
columns = ["product_id", "user_id", "spend", "transaction_date"]
```

```python
# Create DataFrame
df = spark.createDataFrame(data, columns)

# Show the DataFrame
df.show(truncate=False)
df.createOrReplaceTempView("df")
# we need to get the most latest transaction and show how many products bought on that transaction
# we can rank all the transactions then filter out the rank=1 and then do the extra remaining logic
spark.sql("""
        with latest as (
                select
                        transaction_date,
                        user_id,
                        product_id,
                        rank() over(partition by user_id order by transaction_date desc) as trans_rank
                from
                        df
        )
        select
                transaction_date,
                user_id,
                count(product_id) as purchase_count
        from
                latest
        where
                trans_rank=1
        group by
                transaction_date,
                user_id
        order by
                transaction_date
""").show()


window_spec = Window.partitionBy("user_id").orderBy(col("transaction_date").desc())
df = df.withColumn("rank", rank().over(window_spec))
res = df.filter("rank=1").groupBy("transaction_date", "user_id").agg(count("product_id").alias("count"))
res.show()
```

```
+----------+-------+-----+-------------------+
|product_id|user_id|spend|transaction_date   |
+----------+-------+-----+-------------------+
|3673      |123    |68.9 |07/08/2022 12:00:00|
|9623      |123    |274.1|07/08/2022 12:00:00|
|1467      |115    |19.9 |07/08/2022 12:00:00|
|2513      |159    |25.0 |07/08/2022 12:00:00|
|1452      |159    |74.5 |07/10/2022 12:00:00|
+----------+-------+-----+-------------------+

+-------------------+-------+--------------+
|   transaction_date|user_id|purchase_count|
+-------------------+-------+--------------+
|07/08/2022 12:00:00|    115|             1|
|07/08/2022 12:00:00|    123|             2|
|07/10/2022 12:00:00|    159|             1|
+-------------------+-------+--------------+

+-------------------+-------+-----+
|   transaction_date|user_id|count|
+-------------------+-------+-----+
|07/08/2022 12:00:00|    115|    1|
|07/08/2022 12:00:00|    123|    2|
|07/10/2022 12:00:00|    159|    1|
+-------------------+-------+-----+
```

33. You're given a table containing the item count for each order on Alibaba, along with the frequency of orders that have the same item count. Write a query to retrieve the mode of the order occurrences. Additionally, if there are multiple item counts with the same mode, the results should be sorted in ascending order. Clarifications: item_count: Represents the number of items sold in each order. order_occurrences: Represents the frequency of orders with the corresponding number of items sold per order. For example, if there are 800 orders with 3 items sold in each order, the record would have an item_count of 3 and an order_occurrences of 800.

you want to return the item_count for which the maximum order_occurrences are there if there are multiple order by item_count in asc order

In [34]:
```
data = [
        (1, 500),
        (2, 1000),
        (3, 800)
]
```

```
# Define the column names
columns = ["item_count", "order_occurrences"]

# Create DataFrame
df = spark.createDataFrame(data, columns)

# Show the DataFrame
df.show(truncate=False)
df.createOrReplaceTempView("df")
spark.sql("""
        SELECT item_count AS mode
FROM df
WHERE order_occurrences = (
  SELECT MAX(order_occurrences)
  FROM df
)
ORDER BY item_count;
""").show()
max_order_occurrences = df.orderBy(col("order_occurrences").desc()).first()
max_value = max_order_occurrences["order_occurrences"]
print(max_value)
res = df.filter(df.order_occurrences==max_value).select(col("item_count").alias("mode"))
res.show()
```

```
+----------+-----------------+
|item_count|order_occurrences|
+----------+-----------------+
|1         |500              |
|2         |1000             |
|3         |800              |
+----------+-----------------+

+----+
|mode|
+----+
|   2|
+----+

1000
+----+
|mode|
+----+
|   2|
+----+
```

34. Your team at JPMorgan Chase is soon launching a new credit card. You are asked to estimate how many cards you'll issue in the first month. Before you can answer this question, you want to first get some perspective on how well new credit card launches typically do in their first month. Write a query that outputs the name of the credit card, and how many cards were issued in its launch month. The launch month is the earliest record in the monthly_cards_issued table for a given card. Order the results starting from the biggest issued amount.

In [35]:
```
data = [
        (1, 2021, "Chase Sapphire Reserve", 170000),
        (2, 2021, "Chase Sapphire Reserve", 175000),
        (3, 2021, "Chase Sapphire Reserve", 180000),
        (3, 2021, "Chase Freedom Flex", 65000),
        (4, 2021, "Chase Freedom Flex", 70000)
]

# Define the column names
columns = ["issue_month", "issue_year", "card_name", "issued_amount"]

# Create DataFrame
df = spark.createDataFrame(data, columns)
```

```
# Show the DataFrame
df.show(truncate=False)
df.createOrReplaceTempView("df")
spark.sql("""
        with ranked as (
                select
                        card_name,
                        row_number()over(partition by card_name order by issue_year, issue_month) as row_number,
                        issued_amount
                from df
        )
        select
                card_name,
                issued_amount
        from
                ranked
        where
                row_number=1
        order by
                issued_amount desc
""").show()


window_spec = Window.partitionBy("card_name").orderBy("issue_year", "issue_month")
res = (df
           .withColumn("rank", row_number().over(window_spec))
           .filter("rank=1")
           .select("card_name","issued_amount")
           .orderBy(col("issued_amount").desc())
           )
res.show()
```

```
+-----------+----------+----------------------+-------------+
|issue_month|issue_year|card_name             |issued_amount|
+-----------+----------+----------------------+-------------+
|1          |2021      |Chase Sapphire Reserve|170000       |
|2          |2021      |Chase Sapphire Reserve|175000       |
|3          |2021      |Chase Sapphire Reserve|180000       |
|3          |2021      |Chase Freedom Flex    |65000        |
|4          |2021      |Chase Freedom Flex    |70000        |
+-----------+----------+----------------------+-------------+

+--------------------+-------------+
|           card_name|issued_amount|
+--------------------+-------------+
|Chase Sapphire Re...|       170000|
|  Chase Freedom Flex|        65000|
+--------------------+-------------+

+--------------------+-------------+
|           card_name|issued_amount|
+--------------------+-------------+
|Chase Sapphire Re...|       170000|
|  Chase Freedom Flex|        65000|
+--------------------+-------------+
```

35. A phone call is considered an international call when the person calling is in a different country than the person receiving the call. What percentage of phone calls are international? Round the result to 1 decimal. Assumption: The caller_id in phone_info table refers to both the caller and receiver. phone_calls Table:

Data for phone_calls

In [36]:
```
phone_calls_data = [
        (1, 2, "2022-07-04 10:13:49"),
        (1, 5, "2022-08-21 23:54:56"),
        (5, 1, "2022-05-13 17:24:06"),
        (5, 6, "2022-03-18 12:11:49"),
        (5, 6, "2022-03-18 12:11:56"),
        (5, 6, "2022-03-18 12:11:59")
]
phone_calls_columns = ["caller_id", "receiver_id", "call_time"]
phone_calls_df = spark.createDataFrame(phone_calls_data, phone_calls_columns)
phone_calls_df.show(truncate=False)
```

```
phone_info_data = [
        (1, "US", "Verizon", "+1-212-897-1964"),
        (2, "US", "Verizon", "+1-703-346-9529"),
        (3, "US", "Verizon", "+1-650-828-4774"),
        (4, "US", "Verizon", "+1-415-224-6663"),
        (5, "IN", "Vodafone", "+91 7503-907302"),
        (6, "IN", "Vodafone", "+91 2287-664895")
]
phone_info_columns = ["caller_id", "country_id", "network", "phone_number"]
phone_info_df = spark.createDataFrame(phone_info_data, phone_info_columns)
phone_info_df.show(truncate=False)

phone_calls_df.createOrReplaceTempView("phone_calls")
phone_info_df.createOrReplaceTempView("phone_info")
res = spark.sql("""
        WITH international_calls AS (
                SELECT
                  caller.caller_id,
                  caller.country_id,
                  receiver.caller_id,
                  receiver.country_id
                FROM
                        phone_calls AS calls
                LEFT JOIN
                        phone_info AS caller
                ON
                        calls.caller_id = caller.caller_id
                LEFT JOIN
                        phone_info AS receiver
                ON
                        calls.receiver_id = receiver.caller_id
                WHERE
                        caller.country_id <> receiver.country_id
        )

        SELECT
          ROUND(100.0 * COUNT(*)/ (SELECT COUNT(*) FROM phone_calls),1) AS international_call_pct
        FROM international_calls;
""")
res.show()

result_df = (
```

```
        phone_calls_df
        .join(
                        phone_info_df.alias("caller_info"),
                        phone_calls_df.caller_id == col("caller_info.caller_id"),
                        "inner"
        )
        .join(
                phone_info_df.alias("receiver_info"),
                phone_calls_df.receiver_id == col("receiver_info.caller_id"),
                "inner"
        )
        .select(
                phone_calls_df.caller_id,
                phone_calls_df.receiver_id,
                col("caller_info.country_id").alias("caller_country_id"),
                col("receiver_info.country_id").alias("receiver_country_id")
        )
)
result_df.show()

counts_df = (result_df
        .agg(
                count("*").alias("total_calls"),
                count(
                        when(col("caller_country_id") != col("receiver_country_id"), 1)
                ).alias("international_calls")
        )
)

percentage_df = counts_df.select(
        (round(100.0 * col("international_calls") / col("total_calls"), 1))
        .alias("international_call_pct")
)

# Show the percentage DataFrame
percentage_df.show()
```

```
+---------+-----------+-------------------+
|caller_id|receiver_id|call_time          |
+---------+-----------+-------------------+
|1        |2          |2022-07-04 10:13:49|
|1        |5          |2022-08-21 23:54:56|
|5        |1          |2022-05-13 17:24:06|
|5        |6          |2022-03-18 12:11:49|
|5        |6          |2022-03-18 12:11:56|
|5        |6          |2022-03-18 12:11:59|
+---------+-----------+-------------------+

+---------+----------+--------+---------------+
|caller_id|country_id|network |phone_number   |
+---------+----------+--------+---------------+
|1        |US        |Verizon |+1-212-897-1964|
|2        |US        |Verizon |+1-703-346-9529|
|3        |US        |Verizon |+1-650-828-4774|
|4        |US        |Verizon |+1-415-224-6663|
|5        |IN        |Vodafone|+91 7503-907302|
|6        |IN        |Vodafone|+91 2287-664895|
+---------+----------+--------+---------------+

+----------------------+
|international_call_pct|
+----------------------+
|                  33.3|
+----------------------+

+---------+-----------+-----------------+-------------------+
|caller_id|receiver_id|caller_country_id|receiver_country_id|
+---------+-----------+-----------------+-------------------+
|        5|          6|               IN|                 IN|
|        5|          6|               IN|                 IN|
|        5|          6|               IN|                 IN|
|        1|          5|               US|                 IN|
|        5|          1|               IN|                 US|
|        1|          2|               US|                 US|
+---------+-----------+-----------------+-------------------+

+----------------------+
|international_call_pct|
+----------------------+
|                  33.3|
```

```
+----------------------+
```

36. The Bloomberg terminal is the go-to resource for financial professionals, offering convenient access to a wide array of financial datasets. As a Data Analyst at Bloomberg, you have access to historical data on stock performance. Currently, you're analyzing the highest and lowest open prices for each FAANG stock by month over the years. For each FAANG stock, display the ticker symbol, the month and year ('Mon-YYYY') with the corresponding highest and lowest open prices (refer to the Example Output format). Ensure that the results are sorted by ticker symbol.

In [37]:
```
schema = ["date", "ticker", "open", "high", "low", "close"]

data = [
    ("01/31/2023 00:00:00", "AAPL", 142.28, 144.34, 142.70, 144.29),
    ("02/28/2023 00:00:00", "AAPL", 146.83, 149.08, 147.05, 147.41),
    ("03/31/2023 00:00:00", "AAPL", 161.91, 165.00, 162.44, 164.90),
    ("04/30/2023 00:00:00", "AAPL", 167.88, 169.85, 168.49, 169.68),
    ("05/31/2023 00:00:00", "AAPL", 176.76, 179.35, 177.33, 177.25),
]
# Create the DataFrame
df = spark.createDataFrame(data, schema)
df.createOrReplaceTempView("df")
# Show the DataFrame


df = df.withColumn(
    "formatted_date", date_format(to_date("date", "MM/dd/yyyy HH:mm:ss"), "MMMM-yyyy")
)
df.show()

window_spec_max = Window.partitionBy("ticker").orderBy(col("open").desc())
window_spec_min = Window.partitionBy("ticker").orderBy("open")

# the point here is that we create our wanted date format and then we select the rows with min and max as
# separate df and join them as the solution
df_with_max = (
    df.withColumn("highest_open", max("open").over(Window.partitionBy("ticker")))
    .filter(col("open") == col("highest_open"))
    .select("ticker", "formatted_date", "highest_open")
)

df_with_min = (
    df.withColumn("lowest_open", min("open").over(Window.partitionBy("ticker")))
```

```
    .filter(col("open") == col("lowest_open"))
    .select("ticker", "formatted_date", "lowest_open")
)

df_with_max.show()
df_with_min.show()

# Join the DataFrames to include the dates for max and min open
final_df = df_with_max.join(df_with_min, "ticker", "inner")

# Show the final result
final_df.show(truncate=False)


df.groupBy("ticker").agg(
    max("open").alias("highest_open"), min("open").alias("lowest_close")
).show()

spark.sql(
    """
    WITH highest_prices AS (
        SELECT
            ticker,
            date_format(to_date(date, 'MM/dd/yyyy HH:mm:ss'), 'MMM-yyyy') AS highest_mth,
            MAX(open) AS highest_open,
            ROW_NUMBER() OVER (PARTITION BY ticker ORDER BY open DESC) AS row_num
        FROM
            df
        GROUP BY
            ticker,
            date_format(to_date(date, 'MM/dd/yyyy HH:mm:ss'), 'MMM-yyyy'),
            open
    ),
    lowest_prices AS (
        SELECT
            ticker,
            date_format(to_date(date, 'MM/dd/yyyy HH:mm:ss'), 'MMM-yyyy') AS lowest_mth,
            MIN(open) AS lowest_open,
            ROW_NUMBER() OVER (PARTITION BY ticker ORDER BY open) AS row_num
        FROM
            df
        GROUP BY
            ticker,
```

```
            date_format(to_date(date, 'MM/dd/yyyy HH:mm:ss'), 'MMM-yyyy'),
            open
    )
    SELECT
        highest.ticker,
        highest.highest_mth,
        highest.highest_open,
        lowest.lowest_mth,
        lowest.lowest_open
    FROM
        highest_prices AS highest
    INNER JOIN
        lowest_prices AS lowest
    ON
        highest.ticker = lowest.ticker
        AND
        highest.row_num = 1 -- Highest open price
        AND
        lowest.row_num = 1 -- Lowest open price
    ORDER BY
        highest.ticker;
"""
).show()
```

```
+-------------------+------+------+------+------+------+--------------+
|               date|ticker|  open|  high|   low| close|formatted_date|
+-------------------+------+------+------+------+------+--------------+
|01/31/2023 00:00:00|  AAPL|142.28|144.34| 142.7|144.29|  January-2023|
|02/28/2023 00:00:00|  AAPL|146.83|149.08|147.05|147.41| February-2023|
|03/31/2023 00:00:00|  AAPL|161.91| 165.0|162.44| 164.9|    March-2023|
|04/30/2023 00:00:00|  AAPL|167.88|169.85|168.49|169.68|    April-2023|
|05/31/2023 00:00:00|  AAPL|176.76|179.35|177.33|177.25|      May-2023|
+-------------------+------+------+------+------+------+--------------+

+------+--------------+------------+
|ticker|formatted_date|highest_open|
+------+--------------+------------+
|  AAPL|      May-2023|      176.76|
+------+--------------+------------+

+------+--------------+-----------+
|ticker|formatted_date|lowest_open|
+------+--------------+-----------+
|  AAPL|  January-2023|     142.28|
+------+--------------+-----------+

+------+--------------+------------+--------------+-----------+
|ticker|formatted_date|highest_open|formatted_date|lowest_open|
+------+--------------+------------+--------------+-----------+
|AAPL  |May-2023      |176.76      |January-2023  |142.28     |
+------+--------------+------------+--------------+-----------+

+------+------------+------------+
|ticker|highest_open|lowest_close|
+------+------------+------------+
|  AAPL|      176.76|      142.28|
+------+------------+------------+

+------+-----------+------------+----------+-----------+
|ticker|highest_mth|highest_open|lowest_mth|lowest_open|
+------+-----------+------------+----------+-----------+
|  AAPL|   May-2023|      176.76|  Jan-2023|     142.28|
+------+-----------+------------+----------+-----------+
```

37. UnitedHealth Group (UHG) has a program called Advocate4Me, which allows policy holders (or, members) to call an advocate and receive support for their health care needs – whether that's claims and benefits support, drug coverage, pre- and post-authorisation, medical records, emergency assistance, or member portal services. Calls to the Advocate4Me call centre are classified into various categories, but some calls cannot be neatly categorised. These uncategorised calls are labeled as "n/a", or are left empty when the support agent does not enter anything into the call category field. Write a query to calculate the percentage of calls that cannot be categorised. Round your answer to 1 decimal place. For example, 45.0, 48.5, 57.7.

In [38]:
```
schema = ["policy_holder_id", "case_id", "call_category", "call_date", "call_duration_secs"]

# Input data
data = [
    (
        1,
        "f1d012f9-9d02-4966-a968-bf6c5bc9a9fe",
        "emergency assistance",
        "2023-04-13T19:16:53Z",
        144,
    ),
    (
        1,
        "41ce8fb6-1ddd-4f50-ac31-07bfcce6aaab",
        "authorisation",
        "2023-05-25T09:09:30Z",
        815,
    ),
    (2, "9b1af84b-eedb-4c21-9730-6f099cc2cc5e", "n/a", "2023-01-26T01:21:27Z", 992),
    (
        2,
        "8471a3d4-6fc7-4bb2-9fc7-4583e3638a9e",
        "emergency assistance",
        "2023-03-09T10:58:54Z",
        128,
    ),
    (
        2,
        "38208fae-bad0-49bf-99aa-7842ba2e37bc",
        None,
        "2023-06-05T07:35:43Z",
        619,
    ),
]
```

```
# Create the DataFrame
df = spark.createDataFrame(data, schema)

# Show the DataFrame
df.show(truncate=False)
df.createOrReplaceTempView("df")

spark.sql(
    """
    with uncategorised_callers as(
        SELECT
            COUNT(*) AS count
        FROM
            df
        WHERE
            call_category IS NULL
            OR call_category = 'n/a'
            OR call_category = ''
    )
    SELECT
        ROUND(100.0 * count
            / (SELECT COUNT(*) FROM df), 1) AS uncategorised_call_pct
    FROM
        uncategorised_callers;
"""
).show()

total_count_df = df.select(count("*").alias("total_count"))

# Step 2: Calculate count of uncategorised calls
uncategorised_count_df = df.filter(
    (col("call_category").isNull())
    | (col("call_category") == "n/a")
    | (col("call_category") == "")
).select(count("*").alias("uncategorised_count"))

# Step 3: Join the two counts and compute percentage
percentage_df = uncategorised_count_df.crossJoin(total_count_df).select(
    round(100.0 * col("uncategorised_count") / col("total_count"), 1).alias(
        "uncategorised_call_pct"
    )
)
```

```
# Step 4: Show the result
percentage_df.show()
```

```
+----------------+------------------------------------+--------------------+--------------------+------------------+
|policy_holder_id|case_id                             |call_category       |call_date           |call_duration_secs|
+----------------+------------------------------------+--------------------+--------------------+------------------+
|1               |f1d012f9-9d02-4966-a968-bf6c5bc9a9fe|emergency assistance|2023-04-13T19:16:53Z|144               |
|1               |41ce8fb6-1ddd-4f50-ac31-07bfcce6aaab|authorisation       |2023-05-25T09:09:30Z|815               |
|2               |9b1af84b-eedb-4c21-9730-6f099cc2cc5e|n/a                 |2023-01-26T01:21:27Z|992               |
|2               |8471a3d4-6fc7-4bb2-9fc7-4583e3638a9e|emergency assistance|2023-03-09T10:58:54Z|128               |
|2               |38208fae-bad0-49bf-99aa-7842ba2e37bc|NULL                |2023-06-05T07:35:43Z|619               |
+----------------+------------------------------------+--------------------+--------------------+------------------+

+----------------------+
|uncategorised_call_pct|
+----------------------+
|                  40.0|
+----------------------+

+----------------------+
|uncategorised_call_pct|
+----------------------+
|                  40.0|
+----------------------+
```

39. Zomato is a leading online food delivery service that connects users with various restaurants and cuisines, allowing them to browse menus, place orders, and get meals delivered to their doorsteps. Recently, Zomato encountered an issue with their delivery system. Due to an error in the delivery driver instructions, each item's order was swapped with the item in the subsequent row. As a data analyst, you're asked to correct this swapping error and return the proper pairing of order ID and item. If the last item has an odd order ID, it should remain as the last item in the corrected data. For example, if the last item is Order ID 7 Tandoori Chicken, then it should remain as Order ID 7 in the corrected data. In the results, return the correct pairs of order IDs and items.

In [39]:
```
data = [
    (1, "Chow Mein"),
    (2, "Pizza"),
    (3, "Pad Thai"),
    (4, "Butter Chicken"),
    (5, "Eggrolls"),
    (6, "Burger"),
```

```
    (7, "Tandoori Chicken"),
]

columns = ["order_id", "item"]

df = spark.createDataFrame(data, schema=columns)
df.createOrReplaceTempView("df")

df.show()
window_spec = Window.orderBy("order_id")
lead_lag_df = (
    df.withColumn("leading", lead("item", 1).over(window_spec))
    .withColumn("lagging", lag("item", 1).over(window_spec))
    .withColumn("is_last", lead("order_id", 1).over(window_spec).isNull())
)
res = lead_lag_df.withColumn(
    "item",
    when(
        col("is_last") & (col("order_id") % 2 != 0), col("item")
    ).otherwise(
        when(
            col("order_id") % 2 == 0, col("lagging")
        ).otherwise(
            col("leading")
        )
    ),
).select("order_id", "item")
res.show()


spark.sql("""
    select
        order_id,
        case
            when lead(order_id,1) over(order by order_id) is null and order_id % 2 !=0 then item
            when order_id % 2=0 then lag(item, 1) over( order by order_id)
            else lead(item ,1)over(order by order_id)
        end as item
    from
        df
""").show()
```

```
+--------+---------------+
|order_id|           item|
+--------+---------------+
|       1|       Chow Mein|
|       2|           Pizza|
|       3|        Pad Thai|
|       4|  Butter Chicken|
|       5|        Eggrolls|
|       6|          Burger|
|       7|Tandoori Chicken|
+--------+---------------+

+--------+---------------+
|order_id|           item|
+--------+---------------+
|       1|           Pizza|
|       2|       Chow Mein|
|       3|  Butter Chicken|
|       4|        Pad Thai|
|       5|          Burger|
|       6|        Eggrolls|
|       7|Tandoori Chicken|
+--------+---------------+

+--------+---------------+
|order_id|           item|
+--------+---------------+
|       1|           Pizza|
|       2|       Chow Mein|
|       3|  Butter Chicken|
|       4|        Pad Thai|
|       5|          Burger|
|       6|        Eggrolls|
|       7|Tandoori Chicken|
+--------+---------------+
```

In [60]:
```
employee_data = [
    (1, "Emma Thompson", 3800, 1, 6),
    (2, "Daniel Rodriguez", 2230, 1, 7),
    (3, "Olivia Smith", 7000, 2, 8),
    (4, "James Johnson", 5000, 2, 8),
    (5, "Sophia Martinez", 1750, 3, 11),
```

```
    (6, "Michael Brown", 4500, 3, 11),
    (7, "Isabella Garcia", 6000, 4, 12),
    (8, "Ethan Davis", 3000, 4, 12),
    (9, "Ava Wilson", 7500, 5, 13),
    (10, "Alexander Lee", 2900, 5, 13),
]

employee_columns = ["employee_id", "name", "salary", "department_id", "manager_id"]

# Expanded salary data
salary_data = [
    (1, 1, 3800, "03/31/2024 00:00:00"),
    (2, 2, 2230, "03/31/2024 00:00:00"),
    (3, 3, 7000, "03/31/2024 00:00:00"),
    (4, 4, 6800, "03/31/2024 00:00:00"),
    (5, 5, 1750, "03/31/2024 00:00:00"),
    (6, 6, 4500, "03/31/2024 00:00:00"),
    (7, 7, 6000, "03/31/2024 00:00:00"),
    (8, 8, 3000, "03/31/2024 00:00:00"),
    (9, 9, 7500, "03/31/2024 00:00:00"),
    (10, 10, 2900, "03/31/2024 00:00:00"),
    (11, 1, 3800, "04/30/2024 00:00:00"),
    (12, 2, 2230, "04/30/2024 00:00:00"),
    (13, 3, 7000, "04/30/2024 00:00:00"),
    (14, 4, 6800, "04/30/2024 00:00:00"),
    (15, 5, 1750, "04/30/2024 00:00:00"),
    (16, 6, 4500, "04/30/2024 00:00:00"),
    (17, 7, 6000, "04/30/2024 00:00:00"),
    (18, 8, 3000, "04/30/2024 00:00:00"),
    (19, 9, 7500, "04/30/2024 00:00:00"),
    (20, 10, 2900, "04/30/2024 00:00:00"),
]
salary_columns = ["salary_id", "employee_id", "amount", "payment_date"]

# Create DataFrames
employee_df = spark.createDataFrame(employee_data, schema=employee_columns)
salary_df = spark.createDataFrame(salary_data, schema=salary_columns)

salary_df = salary_df.withColumn(
    "payment_date",
    to_timestamp(
        "payment_date", "MM/dd/yyyy HH:mm:ss"
    ),
```

```
)
employee_df.show()
salary_df.show()

employee_df.createOrReplaceTempView("employee")
salary_df.createOrReplaceTempView("salary")


spark.sql(
    """
    WITH company_avg AS (
        SELECT
            payment_date,
            AVG(amount) AS co_avg_salary
        FROM
            salary
        WHERE
            payment_date = to_timestamp('03/31/2024 00:00:00', 'MM/dd/yyyy HH:mm:ss')
        GROUP BY
            payment_date
    ),
    dept_avg AS (
        SELECT
            e.department_id,
            s.payment_date,
            AVG(s.amount) AS dept_avg_salary
        FROM
            salary AS s
        INNER JOIN
            employee AS e
        ON
            s.employee_id = e.employee_id
        WHERE
            s.payment_date = to_timestamp('03/31/2024 00:00:00', 'MM/dd/yyyy HH:mm:ss')
        GROUP BY
            e.department_id, s.payment_date
    )
    SELECT
        d.department_id,
        date_format(d.payment_date, 'MM/yyyy') AS payment_date, -- Update the pattern
        CASE
            WHEN d.dept_avg_salary > c.co_avg_salary+100 THEN 'higher'
            WHEN d.dept_avg_salary < c.co_avg_salary-100 THEN 'lower'
```

```
            ELSE 'same'
        END AS comparison
    FROM
        dept_avg AS d
    INNER JOIN
        company_avg AS c
    ON
        d.payment_date = c.payment_date;

"""
).show()


company_avg_salary_df = (
    salary_df
    .filter(col("payment_date")=='2024-03-31 00:00:00')
    .groupBy("payment_date")
    .agg(avg("amount").alias("company_avg_salary"))
)
company_avg_salary_df.show()
dept_avg_salary = (
    salary_df
    .filter(col("payment_date")=='2024-03-31 00:00:00')
    .join(employee_df, "employee_id", "inner")
    .groupBy(employee_df.department_id, salary_df.payment_date)
    .agg(avg(col("amount")).alias("avg_salary_by_dept"))
    .select(employee_df.department_id, salary_df.payment_date, "avg_salary_by_dept")
)
dept_avg_salary.show()
res_df =  (
    dept_avg_salary
    .join(company_avg_salary_df, "payment_date", "inner")
    .withColumn(
        "comparison",
        when(
            col("avg_salary_by_dept") > col("company_avg_salary")+100,
            'higher'
        ).otherwise(
            when(
                col("avg_salary_by_dept") < col("company_avg_salary")-100, 'lower'
            ).otherwise("same")
        )
    ).select(
```

```
        "department_id",
        date_format("payment_date", "MM/yyyy").alias("payment_date"),
        "comparison"
    )
)
res_df.show()
```

```
+-----------+---------------+------+-------------+----------+
|employee_id|           name|salary|department_id|manager_id|
+-----------+---------------+------+-------------+----------+
|          1|   Emma Thompson|  3800|            1|         6|
|          2|Daniel Rodriguez|  2230|            1|         7|
|          3|    Olivia Smith|  7000|            2|         8|
|          4|   James Johnson|  5000|            2|         8|
|          5| Sophia Martinez|  1750|            3|        11|
|          6|   Michael Brown|  4500|            3|        11|
|          7| Isabella Garcia|  6000|            4|        12|
|          8|     Ethan Davis|  3000|            4|        12|
|          9|      Ava Wilson|  7500|            5|        13|
|         10|   Alexander Lee|  2900|            5|        13|
+-----------+---------------+------+-------------+----------+

+---------+-----------+------+-------------------+
|salary_id|employee_id|amount|       payment_date|
+---------+-----------+------+-------------------+
|        1|          1|  3800|2024-03-31 00:00:00|
|        2|          2|  2230|2024-03-31 00:00:00|
|        3|          3|  7000|2024-03-31 00:00:00|
|        4|          4|  6800|2024-03-31 00:00:00|
|        5|          5|  1750|2024-03-31 00:00:00|
|        6|          6|  4500|2024-03-31 00:00:00|
|        7|          7|  6000|2024-03-31 00:00:00|
|        8|          8|  3000|2024-03-31 00:00:00|
|        9|          9|  7500|2024-03-31 00:00:00|
|       10|         10|  2900|2024-03-31 00:00:00|
|       11|          1|  3800|2024-04-30 00:00:00|
|       12|          2|  2230|2024-04-30 00:00:00|
|       13|          3|  7000|2024-04-30 00:00:00|
|       14|          4|  6800|2024-04-30 00:00:00|
|       15|          5|  1750|2024-04-30 00:00:00|
|       16|          6|  4500|2024-04-30 00:00:00|
|       17|          7|  6000|2024-04-30 00:00:00|
|       18|          8|  3000|2024-04-30 00:00:00|
|       19|          9|  7500|2024-04-30 00:00:00|
|       20|         10|  2900|2024-04-30 00:00:00|
+---------+-----------+------+-------------------+

+-------------+------------+----------+
|department_id|payment_date|comparison|
+-------------+------------+----------+
```

```
|            1|     03/2024|     lower|
|            3|     03/2024|     lower|
|            2|     03/2024|    higher|
|            5|     03/2024|    higher|
|            4|     03/2024|      same|
+-------------+------------+----------+

+-------------------+------------------+
|       payment_date|company_avg_salary|
+-------------------+------------------+
|2024-03-31 00:00:00|            4548.0|
+-------------------+------------------+

+-------------+-------------------+------------------+
|department_id|       payment_date|avg_salary_by_dept|
+-------------+-------------------+------------------+
|            1|2024-03-31 00:00:00|            3015.0|
|            3|2024-03-31 00:00:00|            3125.0|
|            2|2024-03-31 00:00:00|            6900.0|
|            5|2024-03-31 00:00:00|            5200.0|
|            4|2024-03-31 00:00:00|            4500.0|
+-------------+-------------------+------------------+

+-------------+------------+----------+
|department_id|payment_date|comparison|
+-------------+------------+----------+
|            1|     03/2024|     lower|
|            3|     03/2024|     lower|
|            2|     03/2024|    higher|
|            5|     03/2024|    higher|
|            4|     03/2024|      same|
+-------------+------------+----------+
```

In [ ]:

In [ ]: