CHAPTER 3

TPCH DATABASE BENCHMARK TEST [APACHE SPARK]

3.1 CONFIGURATION

3.1.1 TPCH Configuration

3.1.1.a Update Compiler, Database and Machine information

```
################
## CHANGE NAME OF ANSI COMPILER HERE
################
CC      = gcc
# Current values for DATABASE are: INFORMIX, DB2, TDAT (Teradata)
#                                  SQLSERVER, SYBASE, ORACLE, VECTORWISE
# Current values for MACHINE are:  ATT, DOS, HP, IBM, ICL, MVS,
#                                  SGI, SUN, U2200, VMS, LINUX, WIN32
# Current values for WORKLOAD are:  TPCH
DATABASE= SQLSERVER
MACHINE = LINUX
WORKLOAD = TPCH
#
CFLAGS  = -g -DDBNAME=\"dss\" -D$(MACHINE) -D$(DATABASE) -D$(WORKLOAD)
-DRNG_TEST -D_FILE_OFFSET_BITS=64
LDFLAGS = -O
# The OBJ,EXE and LIB macros will need to be changed for compilation under
#  Windows NT
OBJ     = .o
EXE     =
LIBS    = -lm
#
# NO CHANGES SHOULD BE NECESSARY BELOW THIS LINE
################
```

3.1.1.b Update values for SQLSERVER in *tpcd.h* header file.

```
#ifdef  SQLSERVER
#define GEN_QUERY_PLAN  "set showplan on\nset noexec on\ngo\n"
#define START_TRAN      "**BEGIN WORK;**"
#define END_TRAN        "**COMMIT WORK;**"
#define SET_OUTPUT      ""
#define SET_ROWCOUNT    "limit %d;\n\n"
#define SET_DBASE       "use %s;\n"
#endif
```

3.1.1.c Generate Data [100 MB].

```
$ ./dbgen -s 0.1
```

3.1.2 MongoDB Configuration

3.1.2.a Make a directory to store data

```
$ mkdir -p data/db
```

3.1.2.b Run a Mongo instance

```
$ sudo mongod --dbpath ~/data/db
```

3.1.2.c Open Mongo Shell

```
$ mongo
```

3.1.2.d Select Database *tpch*

```
> use tpch
switched to db tpch
```

3.2 POPULATE DATABASE

3.2.1 Convert Data Tables from TBL to JSON Format

Python Script:

```
#!/usr/bin/env python
# coding: utf-8

import pandas as pd
import glob
import time

tbl_files = []
for file in glob.glob("*.tbl"):
      tbl_files.append(file)


table_col_map = {}

table_col_map['part'] =
['P_PARTKEY','P_NAME','P_MFGR','P_BRAND','P_TYPE','P_SIZE','P_CONTAINER','P
_RETAILPRICE','P_COMMENT']
table_col_map['partsupp'] =
['PS_PARTKEY','PS_SUPPKEY','PS_AVAILQTY','PS_SUPPLYCOST','PS_COMMENT']
table_col_map['nation'] = ['N_NATIONKEY', 'N_NAME', 'N_REGIONKEY',
'N_COMMENT']
table_col_map['orders'] =
['O_ORDERKEY','O_CUSTKEY','O_ORDERSTATUS','O_TOTALPRICE','O_ORDERDATE','O_O
RDERPRIORITY','O_CLERK','O_SHIPPRIORITY','O_COMMENT']
table_col_map['customer'] =
['C_CUSTKEY','C_NAME','C_ADDRESS','C_NATIONKEY','C_PHONE','C_ACCTBAL','C_MK
TSEGMENT','C_COMMENT']
table_col_map['supplier'] =
['S_SUPPKEY','S_NAME','S_ADDRESS','S_NATIONKEY','S_PHONE','S_ACCTBAL','S_CO
MMENT']
table_col_map['lineitem'] =
['L_ORDERKEY','L_PARTKEY','L_SUPPKEY','L_LINENUMBER','L_QUANTITY','L_EXTEND
EDPRICE','L_DISCOUNT','L_TAX','L_RETURNFLAG','L_LINESTATUS','L_SHIPDATE','L
_COMMITDATE','L_RECEIPTDATE','L_SHIPINSTRUCT','L_SHIPMODE','L_COMMENT']
table_col_map['region'] = ['R_REGIONKEY', 'R_NAME', 'R_COMMENT']
```

```
def convert_to_json(file_name):
      file_name_without_extension = file_name.split('.')[0]
      col = table_col_map[file_name_without_extension]
      col.append('dump')
      df = pd.read_table(file_name, sep='|', header=None)
      df.columns = col
      df.drop('dump', inplace=True, axis=1)
      output_file_name = file_name_without_extension.upper() + '.json'
      df.to_json(output_file_name,orient='records')

strat_time = start_func_time =  end_time = time.time()

for file in tbl_files:
      strat_time = time.time()
      convert_to_json(file)
      end_time = time.time()
      print("Time taken to convert %s table: %.2f s" %(file, end_time -
strat_time))

end_time = time.time()

print("Total time taken: %.2f s" %(end_time - start_func_time))
```

Shell command to run script:

```
python tbl_to_json.py
```

Output:

```
(base) shekhar@pandora:/mnt/e/major_project/tpch/tpch_tool/tables$ python
tbl_to_json.py
Time taken to convert customer.tbl table: 0.09 s
Time taken to convert lineitem.tbl table: 3.45 s
Time taken to convert nation.tbl table: 0.01 s
Time taken to convert orders.tbl table: 0.65 s
Time taken to convert part.tbl table: 0.10 s
Time taken to convert partsupp.tbl table: 0.31 s
Time taken to convert region.tbl table: 0.01 s
```

```
Time taken to convert supplier.tbl table: 0.01 s
Total time taken: 4.64 s
```

3.2.2 Create and Populate Database

Python Script:

```
#!/usr/bin/env python
# coding: utf-8

import glob
import time
import findspark
findspark.init()
findspark.find()
import pyspark
findspark.find()
from pyspark.sql import SparkSession


spark = SparkSession\
      .builder\
      .appName("tpch")\
      .config("spark.mongodb.input.uri", "mongodb://127.0.0.1/tpch")\
      .config("spark.mongodb.output.uri", "mongodb://127.0.0.1/tpch")\
      .config("spark.jars.packages",
"org.mongodb.spark:mongo-spark-connector_2.12:3.0.1")\
      .getOrCreate()

db_name = "tpch"
json_files = []

for file in glob.glob("*.json"):
      json_files.append(file)

def populate_db(file_name):
      file_name_without_extension = file_name.split('.')[0]
      df = spark.read.json(file_name, multiLine = "true")
```

```
df.write.format("mongo").mode("append").option("database",db_name).option("
collection", file_name_without_extension).save()

strat_time = start_func_time =  end_time = time.time()
for file in json_files:
      strat_time = time.time()
      populate_db(file)
      end_time = time.time()
      print("Time taken to populate %s Collection: %.2f s"
%(file.split(".")[0], end_time - strat_time))
end_time = time.time()

print("Total time taken: %.2f s" %(end_time - start_func_time))
spark.stop()
```

Shell command to run script:

```
python tpch_populate.py
```

Output:

```
Time taken to populate CUSTOMER Collection: 4.04 s
Time taken to populate LINEITEM Collection: 19.93 s
Time taken to populate NATION Collection: 0.31 s
Time taken to populate ORDERS Collection: 3.23 s
Time taken to populate PART Collection: 0.65 s
Time taken to populate PARTSUPP Collection: 1.26 s
Time taken to populate REGION Collection: 0.23 s
Time taken to populate SUPPLIER Collection: 0.24 s
Total time taken: 29.89 s
```

3.2 TPCH 22 BENCHMARK QUERIES

Base Python Script

```
#!/usr/bin/env python
# coding: utf-8


import glob
import time
import sys
import findspark
findspark.init()
findspark.find()
import pyspark
findspark.find()
from pyspark.sql import SparkSession


spark = SparkSession\
      .builder\
      .appName("tpch")\
      .config("spark.mongodb.input.uri", "mongodb://127.0.0.1/tpch")\
      .config("spark.mongodb.output.uri", "mongodb://127.0.0.1/tpch")\
      .config("spark.jars.packages",
"org.mongodb.spark:mongo-spark-connector_2.12:3.0.1")\
      .config("spark.executor.memory", "2g")\
      .config("spark.driver.memory", "4g")\
      .getOrCreate()


customer_df =
spark.read.format("mongo").option("uri","mongodb://127.0.0.1/tpch.CUSTOMER"
).load()
lineitem_df =
spark.read.format("mongo").option("uri","mongodb://127.0.0.1/tpch.LINEITEM"
).load()
nation_df =
spark.read.format("mongo").option("uri","mongodb://127.0.0.1/tpch.NATION").
load()
region_df =
spark.read.format("mongo").option("uri","mongodb://127.0.0.1/tpch.REGION").
```

```
load()
orders_df =
spark.read.format("mongo").option("uri","mongodb://127.0.0.1/tpch.ORDERS").
load()
part_df =
spark.read.format("mongo").option("uri","mongodb://127.0.0.1/tpch.PART").lo
ad()
partsupp_df =
spark.read.format("mongo").option("uri","mongodb://127.0.0.1/tpch.PARTSUPP"
).load()
supplier_df =
spark.read.format("mongo").option("uri","mongodb://127.0.0.1/tpch.SUPPLIER"
).load()


customer_df.createOrReplaceTempView("customer")
lineitem_df.createOrReplaceTempView("lineitem")
nation_df.createOrReplaceTempView("nation")
region_df.createOrReplaceTempView("region")
orders_df.createOrReplaceTempView("orders")
part_df.createOrReplaceTempView("part")
partsupp_df.createOrReplaceTempView("partsupp")
supplier_df.createOrReplaceTempView("supplier")

## queries function q1 to q22 [Mentioned in below queries]

def execute_queries(q):
      start_time = end_time = time.time()
      query = 'q'+str(q)+"()"
      eval(query)
      end_time = time.time()
      print("Time taken to execute query %s : %.2f s" %(q,(end_time -
start_time)))


n = len(sys.argv)
if n > 1:
      execute_queries(sys.argv[1])
else:
      print("Mention Query No.")


spark.stop()
```

3.2.1 Pricing Summary Report Query (Q1)

Statement:
This query reports the amount of business that was billed, shipped, and returned.

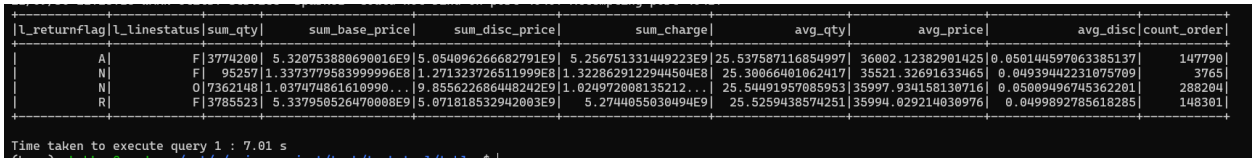
Python SQL Query function:

```
def q1():
      query = """
select
      l_returnflag,
      l_linestatus,
      sum(l_quantity) as sum_qty,
      sum(l_extendedprice) as sum_base_price,
      sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
      sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
      avg(l_quantity) as avg_qty,
      avg(l_extendedprice) as avg_price,
      avg(l_discount) as avg_disc,
      count(*) as count_order
from
      LINEITEM
where
      l_shipdate <= date '1998-12-01' - interval '108' day
group by
      l_returnflag,
      l_linestatus
order by
      l_returnflag,
      l_linestatus;
      """
      spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 1
```

Output :



```
+------------+------------+------+-----------------+-----------------+-----------------+-----------+------------+--------------------+-----------+
|l_returnflag|l_linestatus|sum_qty|   sum_base_price|  sum_disc_price|       sum_charge|    avg_qty|   avg_price|            avg_disc|count_order|
+------------+------------+------+-----------------+-----------------+-----------------+-----------+------------+--------------------+-----------+
|           A|           F|3774200| 5.320753880690016E9|5.054096266682791E9| 5.256751331449223E9|25.5375871|16854997| 36002.12382901425|0.050144597063385137|     147790|
|           N|           F|  95257|1.3373777583999996E8|1.271323726511999E8|1.3228629122944504E8| 25.30066401062417| 35521.32691633465| 0.04939442231075709|       3765|
|           N|           O|7362148|1.037474861610990...|9.855622686448242E9|1.024972008135212...| 25.54491957085953|35997.934158130716| 0.05009496745362201|     288204|
|           R|           F|3785523| 5.337950526470008E9|5.071818532942003E9|   5.27440550304944E9|  25.525943857425 1|35994.029214030976|  0.0499892785618285|     148301|
+------------+------------+------+-----------------+-----------------+-----------------+-----------+------------+--------------------+-----------+

Time taken to execute query 1 : 7.01 s
```

Execution Time: 7.01 sec

3.2.2 Minimum Cost Supplier Query (Q2)

Statement:
This query finds which supplier should be selected to place an order for a given part in a given region.

Python SQL Query function:

```
def q2():
      query = """
select
      s_acctbal,
      s_name,
      n_name,
      p_partkey,
      p_mfgr,
      s_address,
      s_phone,
      s_comment
from
      PART,
      SUPPLIER,
      PARTSUPP,
      NATION,
      REGION
where
      p_partkey = ps_partkey
      and s_suppkey = ps_suppkey
      and p_size = 30
      and p_type like '%STEEL'
      and s_nationkey = n_nationkey
      and n_regionkey = r_regionkey
      and r_name = 'ASIA'
      and ps_supplycost = (
      select
            min(ps_supplycost)
      from
```

```
            PARTSUPP,
            SUPPLIER,
            NATION,
            REGION
      where
            p_partkey = ps_partkey
            and s_suppkey = ps_suppkey
            and s_nationkey = n_nationkey
            and n_regionkey = r_regionkey
            and r_name = 'ASIA'
      )
order by
      s_acctbal desc,
      n_name,
      s_name,
      p_partkey
limit
      100;


      """
      spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 2
```

Output :

```
+---------+------------------+---------+---------+-------------+--------------------+---------------+--------------------+
|s_acctbal|           s_name|   n_name|p_partkey|        p_mfgr|           s_address|        s_phone|           s_comment|
+---------+------------------+---------+---------+-------------+--------------------+---------------+--------------------+
|  9746.01|Supplier#000000883|    CHINA|     1380|Manufacturer#1|5ppzWDz6xcMOO09Lk...|28-614-756-7513|hely final excuse...|
|  9746.01|Supplier#000000883|    CHINA|     8882|Manufacturer#1|5ppzWDz6xcMOO09Lk...|28-614-756-7513|hely final excuse...|
|  9464.26|Supplier#000000507|INDONESIA|     9988|Manufacturer#1|aF2w4JF8qV aaqApY...|19-981-569-8699|p carefully besid...|
|  9238.79|Supplier#000000550|    JAPAN|     4549|Manufacturer#4|    QQavssDXnYHbvOrg|22-648-743-9295|en, bold ideas. i...|
|  9219.26|Supplier#000000948|  VIETNAM|     4185|Manufacturer#5|LvcPHBbzYZKySxlda...|31-562-389-2753|grate slyly after...|
|   9025.9|Supplier#000000995|    CHINA|     1743|Manufacturer#3|   CgVUX8DtNbtug2M,N|28-180-818-2912|s nag. furiously ...|
|   9025.9|Supplier#000000995|    CHINA|     4232|Manufacturer#5|   CgVUX8DtNbtug2M,N|28-180-818-2912|s nag. furiously ...|
|   8684.6|Supplier#000000351|    JAPAN|     9832|Manufacturer#2|ZLWTvVCSmwskfElT7...|22-508-407-2628|ithely ironic the...|
|  8091.65|Supplier#000000106|  VIETNAM|      605|Manufacturer#1|50EV3vyfAsWJAjTbT...|31-810-990-4600|eas affix careful...|
|  8042.43|Supplier#000000324|    CHINA|    14531|Manufacturer#1|QDsgOSozg jniYR2H...|28-637-452-5085|ithely slyly spec...|
|  7916.56|Supplier#000000953|  VIETNAM|     8952|Manufacturer#4|wTTb0ilU6Nba1VLsH...|31-642-490-3022|kages are careful...|
|  7773.41|Supplier#000000043|    JAPAN|     5532|Manufacturer#4|Z5mLuAoTUEeKY5v22...|22-421-568-4862|unts. unusual, fi...|
|  7114.81|Supplier#000000293|INDONESIA|    10292|Manufacturer#3|H2JnUWv1X3s0yI7i2...|19-143-962-9484|t the instruction...|
|  6540.34|Supplier#000000388|    CHINA|     2883|Manufacturer#2|n27XQohXrXlJRLdsy...|28-306-827-7902|rate around the r...|
|  6296.15|Supplier#000000274|  VIETNAM|    14009|Manufacturer#5|usxbl9KSW41DTE6FA...|31-571-345-4549|ecial courts. exp...|
|  5926.41|Supplier#000000023|INDONESIA|     6766|Manufacturer#2|ssetugTcXc096qlD7...|19-559-422-5776|ges could have to...|
|  5672.23|Supplier#000000745|  VIETNAM|    19937|Manufacturer#2|  KBaVOy ,RKCWhLiYxW|31-469-792-6546|thely unusual ide...|
|  4332.95|Supplier#000000112|    JAPAN|    13111|Manufacturer#1|vdWe5lfgvisRCxdd8...|22-617-876-1402|es eat fluffily b...|
|  3689.14|Supplier#000000870|  VIETNAM|      119|Manufacturer#4|       QIgRinpKvCLPG|31-675-338-9417|ronic accounts. q...|
|  3689.14|Supplier#000000870|  VIETNAM|    12345|Manufacturer#3|       QIgRinpKvCLPG|31-675-338-9417|ronic accounts. q...|
+---------+------------------+---------+---------+-------------+--------------------+---------------+--------------------+
only showing top 20 rows

Time taken to execute query 2 : 10.10 s
```

Execution Time: 10.10 sec

3.2.3 Shipping Priority Query (Q3)

Statement:
This query retrieves the 10 unshipped orders with the highest value.

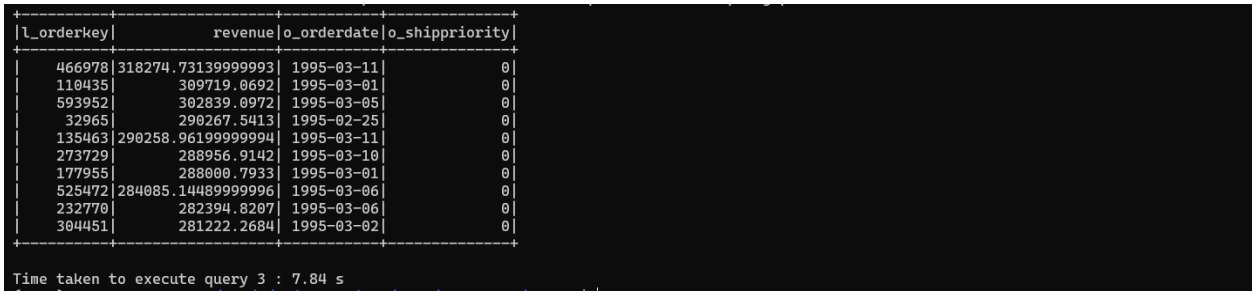
Python SQL Query function:

```
def q3():
	query = """
select
	l_orderkey,
	sum(l_extendedprice * (1 - l_discount)) as revenue,
	o_orderdate,
	o_shippriority
from
	CUSTOMER,
	ORDERS,
	LINEITEM
where
	c_mktsegment = 'AUTOMOBILE'
	and c_custkey = o_custkey
	and l_orderkey = o_orderkey
	and o_orderdate < date '1995-03-13'
	and l_shipdate > date '1995-03-13'
group by
	l_orderkey,
	o_orderdate,
	o_shippriority
order by
	revenue desc,
	o_orderdate
limit
	10;


	"""
	spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 3
```

Output :



```
+----------+-----------------+----------+--------------+
|l_orderkey|          revenue|o_orderdate|o_shippriority|
+----------+-----------------+----------+--------------+
|    466978|318274.73139999993| 1995-03-11|             0|
|    110435|       309719.0692| 1995-03-01|             0|
|    593952|       302839.0972| 1995-03-05|             0|
|     32965|       290267.5413| 1995-02-25|             0|
|    135463|290258.96199999994| 1995-03-11|             0|
|    273729|       288956.9142| 1995-03-10|             0|
|    177955|       288000.7933| 1995-03-01|             0|
|    525472|284085.14489999996| 1995-03-06|             0|
|    232770|       282394.8207| 1995-03-06|             0|
|    304451|       281222.2684| 1995-03-02|             0|
+----------+-----------------+----------+--------------+

Time taken to execute query 3 : 7.84 s
```

Execution Time: 7.84 sec

3.2.4 Order Priority Checking Query (Q4)

Statement:
This query determines how well the order priority system is working and gives an assessment of customer satisfaction.

Python SQL Query function:

```
def q4():
	query = """
select
	o_orderpriority,
	count(*) as order_count
from
	ORDERS
where
	o_orderdate >= date '1995-01-01'
	and o_orderdate < date '1995-01-01' + interval '3' month
	and exists (
	select
		*
	from
		LINEITEM
	where
		l_orderkey = o_orderkey
		and l_commitdate < l_receiptdate
```

```
        )
group by
        o_orderpriority
order by
        o_orderpriority;


        """
        spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 4
```

Output :

Execution Time: 6.92 sec

3.2.5 Local Supplier Volume Query (Q5)

Statement:
This query lists the revenue volume done through local suppliers.

Python SQL Query function:

```
def q5():
        query = """
select
        n_name,
        sum(l_extendedprice * (1 - l_discount)) as revenue
from
        CUSTOMER,
        ORDERS,
        LINEITEM,
```

```
      SUPPLIER,
      NATION,
      REGION
where
      c_custkey = o_custkey
      and l_orderkey = o_orderkey
      and l_suppkey = s_suppkey
      and c_nationkey = s_nationkey
      and s_nationkey = n_nationkey
      and n_regionkey = r_regionkey
      and r_name = 'MIDDLE EAST'
      and o_orderdate >= date '1994-01-01'
      and o_orderdate < date '1994-01-01' + interval '1' year
group by
      n_name
order by
      revenue desc;


      """
      spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 5
```

Output :

Execution Time: 12.76 sec

3.2.6 Forecasting Revenue Change Query (Q6)

Statement:
This query quantifies the amount of revenue increase that would have resulted from eliminating certain company-wide discounts in a given percentage range in a given year. Asking this type of "what if" query can be used to look for ways to increase revenues.

Python SQL Query function:

```
def q6():
      query = """
select
      sum(l_extendedprice * l_discount) as revenue
from
      LINEITEM
where
      l_shipdate >= date '1994-01-01'
      and l_shipdate < date '1994-01-01' + interval '1' year
      and l_discount between 0.06 - 0.01
      and 0.06 + 0.01
      and l_quantity < 24;


      """
      spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 6
```

Output :

Execution Time: 3.06 sec

3.2.7 Volume Shipping Query (Q7)

Statement:
This query determines the value of goods shipped between certain nations to help in the re-negotiation of shipping contracts.

Python SQL Query function:

```
def q7():
      query = """
select
      supp_nation,
      cust_nation,
      l_year,
      sum(volume) as revenue
from
      (
      select
            n1.n_name as supp_nation,
            n2.n_name as cust_nation,
            extract(
                  year
                  from
                  l_shipdate
            ) as l_year,
            l_extendedprice * (1 - l_discount) as volume
      from
            SUPPLIER,
            LINEITEM,
            ORDERS,
            CUSTOMER,
            NATION n1,
            NATION n2
      where
            s_suppkey = l_suppkey
            and o_orderkey = l_orderkey
            and c_custkey = o_custkey
            and s_nationkey = n1.n_nationkey
            and c_nationkey = n2.n_nationkey
            and (
                  (
                  n1.n_name = 'JAPAN'
                  and n2.n_name = 'INDIA'
```

```
                    )
                    or (
                    n1.n_name = 'INDIA'
                    and n2.n_name = 'JAPAN'
                    )
              )
              and l_shipdate between date '1995-01-01'
              and date '1996-12-31'
       ) as shipping
group by
       supp_nation,
       cust_nation,
       l_year
order by
       supp_nation,
       cust_nation,
       l_year;


       """
       spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 7
```

Output :

Execution Time: 14.39 sec

3.2.8 National Market Share Query (Q8)

Statement:
This query determines how the market share of a given nation within a given region has changed over two years for a given part type.

Python SQL Query function:

```
def q8():
      query = """
select
      o_year,
      sum(
      case
            when nation = 'INDIA' then volume
            else 0
      end
      ) / sum(volume) as mkt_share
from
      (
      select
            extract(
                  year
                  from
                  o_orderdate
            ) as o_year,
            l_extendedprice * (1 - l_discount) as volume,
            n2.n_name as nation
      from
            PART,
            SUPPLIER,
            LINEITEM,
            ORDERS,
            CUSTOMER,
            NATION n1,
            NATION n2,
            REGION
      where
            p_partkey = l_partkey
            and s_suppkey = l_suppkey
            and l_orderkey = o_orderkey
            and o_custkey = c_custkey
            and c_nationkey = n1.n_nationkey
            and n1.n_regionkey = r_regionkey
            and r_name = 'ASIA'
            and s_nationkey = n2.n_nationkey
            and o_orderdate between date '1995-01-01'
            and date '1996-12-31'
```

```
            and p_type = 'SMALL PLATED COPPER'
      ) as all_nations
group by
      o_year
order by
      o_year;


      """
      spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 8
```

Output :

Execution Time: 12.72 sec

3.2.9 Product Type Profit Measure Query (Q9)

Statement:
This query determines how much profit is made on a given line of parts, broken out by supplier nation and year.

Python SQL Query function:

```
def q9():
      query = """
select
      nation,
      o_year,
      sum(amount) as sum_profit
from
      (
      select
```

```
            n_name as nation,
            extract(
                  year
                  from
                  o_orderdate
            ) as o_year,
            l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity
as amount
      from
            PART,
            SUPPLIER,
            LINEITEM,
            PARTSUPP,
            ORDERS,
            NATION
      where
            s_suppkey = l_suppkey
            and ps_suppkey = l_suppkey
            and ps_partkey = l_partkey
            and p_partkey = l_partkey
            and o_orderkey = l_orderkey
            and s_nationkey = n_nationkey
            and p_name like '%dim%'
      ) as profit
group by
      nation,
      o_year
order by
      nation,
      o_year desc;


      """
      spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 9
```

Output :

```
+---------+------+-----------------+
|   nation|o_year|        sum_profit|
+---------+------+-----------------+
|  ALGERIA|  1998|1906845.3283000002|
|  ALGERIA|  1997| 3613876.263599999|
|  ALGERIA|  1996| 3288633.161600001|
|  ALGERIA|  1995|      4044611.7358|
|  ALGERIA|  1994| 3872129.156099998|
|  ALGERIA|  1993|3405270.0497999997|
|  ALGERIA|  1992|      4215274.8184|
|ARGENTINA|  1998|2038238.0456999997|
|ARGENTINA|  1997|3879101.0940000024|
|ARGENTINA|  1996|3302688.5552999987|
|ARGENTINA|  1995|4002602.1325999983|
|ARGENTINA|  1994| 3691344.314600003|
|ARGENTINA|  1993|3470808.1437000013|
|ARGENTINA|  1992| 4473494.846899998|
|   BRAZIL|  1998|2697119.3011999996|
|   BRAZIL|  1997| 4720446.985299998|
|   BRAZIL|  1996|4189215.9468000024|
|   BRAZIL|  1995|      4427615.5963|
|   BRAZIL|  1994| 4651155.958099999|
|   BRAZIL|  1993| 4910382.082600001|
+---------+------+-----------------+
only showing top 20 rows

Time taken to execute query 9 : 14.13 s
```

Execution Time: 14.13 sec

3.2.10 Returned Item Reporting Query (Q10)

Statement:
The query identifies customers who might be having problems with the parts that are shipped to them.

Python SQL Query function:

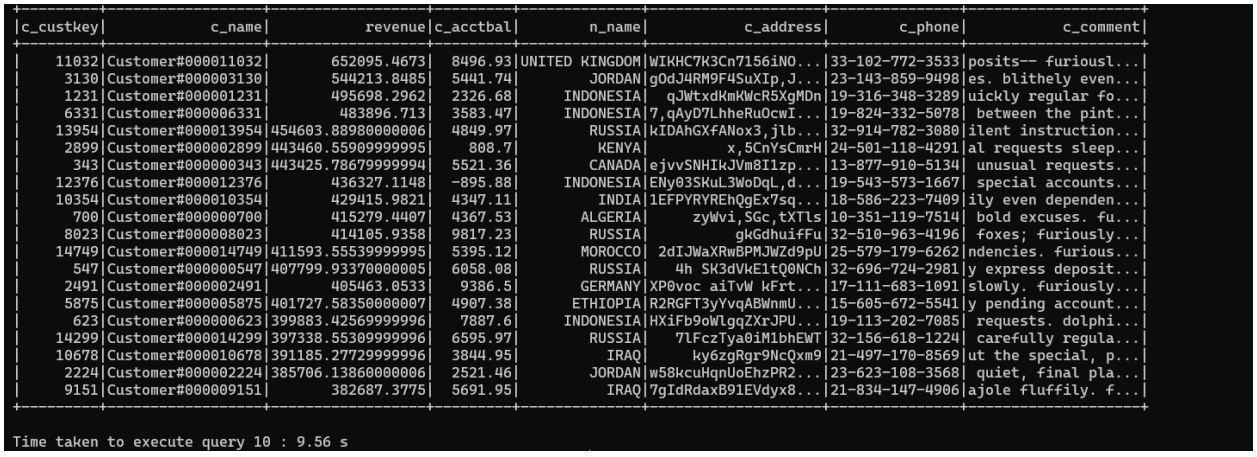
```
def q10():
	query = """
select
	c_custkey,
	c_name,
	sum(l_extendedprice * (1 - l_discount)) as revenue,
	c_acctbal,
	n_name,
	c_address,
	c_phone,
	c_comment
from
	CUSTOMER,
	ORDERS,
	LINEITEM,
	NATION
where
	c_custkey = o_custkey
	and l_orderkey = o_orderkey
```

```
      and o_orderdate >= date '1993-08-01'
      and o_orderdate < date '1993-08-01' + interval '3' month
      and l_returnflag = 'R'
      and c_nationkey = n_nationkey
group by
      c_custkey,
      c_name,
      c_acctbal,
      c_phone,
      n_name,
      c_address,
      c_comment
order by
      revenue desc
limit
      20;


      """
      spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 10
```

Output :



```
+---------+------------------+-----------+---------+-------------+--------------------+---------------+--------------------+
|c_custkey|            c_name|    revenue|c_acctbal|       n_name|           c_address|        c_phone|           c_comment|
+---------+------------------+-----------+---------+-------------+--------------------+---------------+--------------------+
|    11032|Customer#000011032|652095.4673|  8496.93|UNITED KINGDOM|WIKHC7K3Cn7156iNO...|33-102-772-3533|posits-- furiousl...|
|     3130|Customer#000003130|544213.8485|  5441.74|       JORDAN|gOdJ4RM9F4SuXIp,J...|23-143-859-9498|es. blithely even...|
|     1231|Customer#000001231|495698.2962|  2326.68|    INDONESIA|  qJWtxdKmkWcR5XgMDn|19-316-348-3289|uickly regular fo...|
|     6331|Customer#000006331| 483896.713|  3583.47|    INDONESIA|7,qAyD7LhheRuOcwI...|19-824-332-5078| between the pint...|
|    13954|Customer#000013954|454603.8898|  4849.97|       RUSSIA|kIDAhGXfANox3,jlb...|32-914-782-3080|ilent instruction...|
|     2899|Customer#000002899|443460.5590|    808.7|        KENYA|         x,5CnYsCmrH|24-501-118-4291|al requests sleep...|
|      343|Customer#000000343|443425.7867|  5521.36|       CANADA|ejvvSNHIkJVm8I1zp...|13-877-910-5134| unusual requests...|
|    12376|Customer#000012376|436327.1148|  -895.88|    INDONESIA|ENy03SKuL3WoDqL,d...|19-543-573-1667| special accounts...|
|    10354|Customer#000010354|429415.9821|  4347.11|        INDIA|1EFPYRYREhQgEx7sq...|18-586-223-7409|ily even dependen...|
|      700|Customer#000000700|415279.4407|  4367.53|      ALGERIA|     zyWvi,SGc,tXTls|10-351-119-7514| bold excuses. fu...|
|     8023|Customer#000008023|414105.9358|  9817.23|       RUSSIA|          gkGdhuifFu|32-510-963-4196| foxes; furiously...|
|    14749|Customer#000014749|411593.5553|  5395.12|      MOROCCO| 2dIJWaXRwBPMJWZd9pU|25-579-179-6262|ndencies. furious...|
|      547|Customer#000000547|407799.9337|  6058.08|       RUSSIA|   4h SK3dVkE1tQ0NCh|32-696-724-2981|y express deposit...|
|     2491|Customer#000002491|405463.0533|   9386.5|      GERMANY|XP0voc aiTvW kFrt...|17-111-683-1091|slowly. furiously...|
|     5875|Customer#000005875|401727.5835|  4907.38|     ETHIOPIA|R2RGFT3yYvqABWnmU...|15-605-672-5541|y pending account...|
|      623|Customer#000000623|399883.4256|   7887.6|    INDONESIA|HXiFb9oWlgqZXrJPU...|19-113-202-7085| requests. dolphi...|
|    14299|Customer#000014299|397338.5533|  6595.97|       RUSSIA|   7lFczTya0iM1bhEWT|32-156-618-1224| carefully regula...|
|    10678|Customer#000010678|391185.2772|  3844.95|         IRAQ|     ky6zgRgr9NcQxm9|21-497-170-8569|ut the special, p...|
|     2224|Customer#000002224|385706.1386|  2521.46|       JORDAN|w58kcuHqnUoEhzPR2...|23-623-108-3568| quiet, final pla...|
|     9151|Customer#000009151| 382687.3775|  5691.95|         IRAQ|7gIdRdaxB91EVdyx8...|21-834-147-4906|ajole fluffily. f...|
+---------+------------------+-----------+---------+-------------+--------------------+---------------+--------------------+

Time taken to execute query 10 : 9.56 s
```

Execution Time: 9.56 sec

3.2.11 Important Stock Identification Query (Q11)

Statement:
This query finds the most important subset of suppliers' stock in a given nation.

Python SQL Query function:

```
def q11():
      query = """
select
      ps_partkey,
      sum(ps_supplycost * ps_availqty) as value
from
      PARTSUPP,
      SUPPLIER,
      NATION
where
      ps_suppkey = s_suppkey
      and s_nationkey = n_nationkey
      and n_name = 'MOZAMBIQUE'
group by
      ps_partkey
having
      sum(ps_supplycost * ps_availqty) > (
      select
            sum(ps_supplycost * ps_availqty) * 0.0001000000
      from
            PARTSUPP,
            SUPPLIER,
            NATION
      where
            ps_suppkey = s_suppkey
            and s_nationkey = n_nationkey
            and n_name = 'MOZAMBIQUE'
      )
order by
      value desc;


      """
      spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 11
```

Output :

```
+----------+--------------------+
|ps_partkey|               value|
+----------+--------------------+
|     16724|       1.540221766E7|
|      9833|       1.485947034E7|
|     18069|       1.338898942E7|
|      6049|       1.329420005E7|
|      7858|1.3221883399999999E7|
|     16539|        1.24774185E7|
|     16574|        1.19410423E7|
|      1100|1.1632792399999999E7|
|     19106|       1.135809525E7|
|     12548|       1.071292425E7|
|      5177|1.0352624139999999E7|
|     15990|          9988902.43|
|     16607|   9800545.120000001|
|     13615|           9720567.6|
|      5562|          9632559.07|
|      4887|          9615857.94|
|     13595|          9611110.18|
|      6351|          9596307.95|
|     13503|          9480716.28|
|      8715|          9455820.31|
+----------+--------------------+
only showing top 20 rows

Time taken to execute query 11 : 7.62 s
```

Execution Time: 7.62 sec

3.2.12 Shipping Modes and Order Priority Query (Q12)

Statement:

This query determines whether selecting less expensive modes of shipping is negatively affecting the critical-priority orders by causing more parts to be received by customers after the committed date.

Python SQL Query function:

```
def q12():
      query = """
select l_shipmode,
sum(
      case
      when o_orderpriority = '1-URGENT'
      or o_orderpriority = '2-HIGH' then 1
      else 0
      end
) as high_line_count,
sum(
      case
      when o_orderpriority <> '1-URGENT'
      and o_orderpriority <> '2-HIGH' then 1
      else 0
      end
) as low_line_count
from
      ORDERS,
      LINEITEM
```

```
where
      o_orderkey = l_orderkey
      and l_shipmode in ('RAIL', 'FOB')
      and l_commitdate < l_receiptdate
      and l_shipdate < l_commitdate
      and l_receiptdate >= date '1997-01-01'
      and l_receiptdate < date '1997-01-01' + interval '1' year
group by
      l_shipmode
order by
      l_shipmode;


      """
      spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 12
```

Output :

Execution Time: 8.55 sec

3.2.13 Customer Distribution Query (Q13)

Statement: This query seeks relationships between customers and the size of their orders.

Python SQL Query function:

```
def q13():
      query = """
select
      c_count,
      count(*) as custdist
from
      (
      select
            c_custkey,
            count(o_orderkey) as c_count
```

```
      from
            CUSTOMER
            left outer join ORDERS on c_custkey = o_custkey
            and o_comment not like '%pending%deposits%'
      group by
            c_custkey
      ) c_orders
group by
      c_count
order by
      custdist desc,
      c_count desc;


      """
      spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 13
```

Output :

```
+-------+--------+
|c_count|custdist|
+-------+--------+
|      0|    5000|
|     10|     676|
|      9|     651|
|     11|     618|
|     12|     554|
|      8|     548|
|     13|     514|
|      7|     487|
|     19|     485|
|     18|     461|
|     14|     454|
|     20|     444|
|     16|     442|
|     17|     438|
|     15|     430|
|     21|     396|
|     22|     378|
|      6|     355|
|     23|     322|
|     24|     262|
+-------+--------+
only showing top 20 rows

Time taken to execute query 13 : 6.53 s
```

Execution Time: 6.53 sec

3.2.14 Promotion Effect Query (Q14)

Statement: This query monitors the market response to a promotion such as TV advertisements or a special campaign.

Python SQL Query function:

```
def q14():
	query = """
select
	100.00 * sum(
	case
		when p_type like 'PROMO%' then l_extendedprice * (1 -
l_discount)
		else 0
	end
	) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from
	LINEITEM,
	PART
where
	l_partkey = p_partkey
	and l_shipdate >= date '1996-12-01'
	and l_shipdate < date '1996-12-01' + interval '1' month;


	"""
	spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 14
```

Output :

Execution Time: 5.63 sec

3.2.15 Top Supplier Query (Q15)

Statement: This query determines the top supplier so it can be rewarded, given more business, or identified for special recognition.

Python SQL Query function:

```
def q15():
      query = """
with revenue0 as
 (select
 l_suppkey as supplier_no,
 sum(l_extendedprice * (1 - l_discount)) as total_revenue
 from
 lineitem
 where
 l_shipdate >= date '1997-07-01'
 and l_shipdate < date '1997-07-01' + interval '3' month
 group by
 l_suppkey)

 select
 s_suppkey,
 s_name,
 s_address,
 s_phone,
 total_revenue
 from
 supplier,
 revenue0
 where
 s_suppkey = supplier_no
 and total_revenue = (
 select
 max(total_revenue)
 from
 revenue0
 )
 order by
 s_suppkey



      """
      spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 15
```

Output :

Execution Time: 9.46 sec

3.2.16 Parts/Supplier Relationship Query (Q16)

Statement: This query finds out how many suppliers can supply parts with given attributes. It might be used, for example, to determine whether there is a sufficient number of suppliers for heavily ordered parts.

Python SQL Query function:

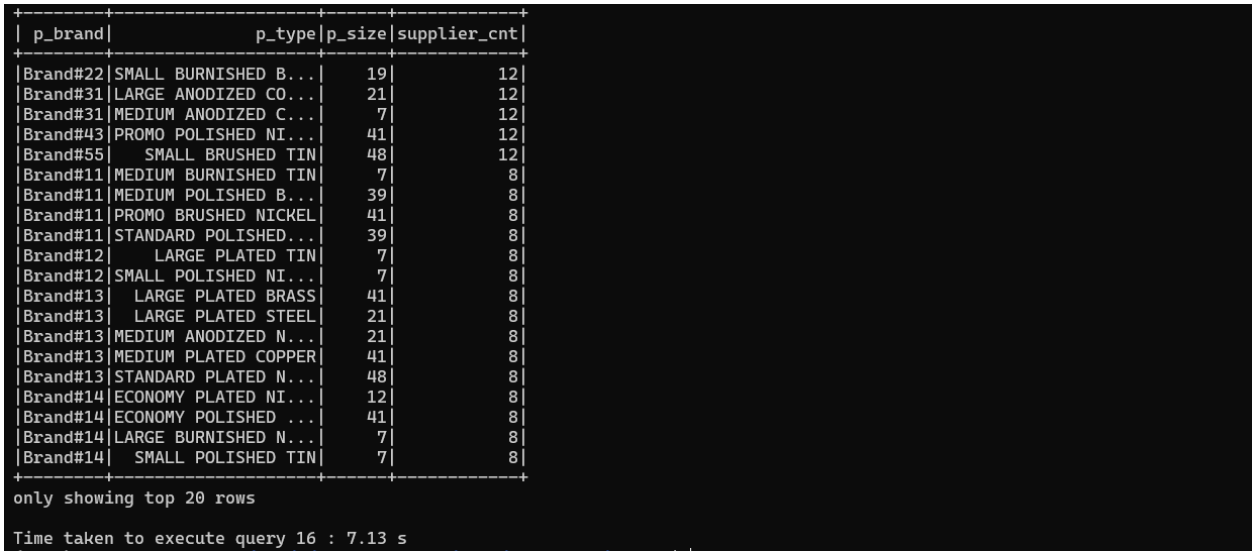
```
def q16():
      query = """
select
      p_brand,
      p_type,
      p_size,
      count(distinct ps_suppkey) as supplier_cnt
from
      PARTSUPP,
      PART
where
      p_partkey = ps_partkey
      and p_brand <> 'Brand#34'
      and p_type not like 'LARGE BRUSHED%'
      and p_size in (48, 19, 12, 4, 41, 7, 21, 39)
      and ps_suppkey not in (
      select
            s_suppkey
      from
            SUPPLIER
      where
            s_comment like '%Customer%Complaints%'
      )
group by
      p_brand,
      p_type,
```

```
      p_size
order by
      supplier_cnt desc,
      p_brand,
      p_type,
      p_size;


      """
      spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 16
```

Output :



```
+--------+--------------------+------+------------+
| p_brand|              p_type|p_size|supplier_cnt|
+--------+--------------------+------+------------+
|Brand#22|SMALL BURNISHED B...|    19|          12|
|Brand#31|LARGE ANODIZED CO...|    21|          12|
|Brand#31|MEDIUM ANODIZED C...|     7|          12|
|Brand#43|PROMO POLISHED NI...|    41|          12|
|Brand#55|   SMALL BRUSHED TIN|    48|          12|
|Brand#11|MEDIUM BURNISHED TIN|     7|           8|
|Brand#11|MEDIUM POLISHED B...|    39|           8|
|Brand#11|PROMO BRUSHED NICKEL|    41|           8|
|Brand#11|STANDARD POLISHED...|    39|           8|
|Brand#12|    LARGE PLATED TIN|     7|           8|
|Brand#12|SMALL POLISHED NI...|     7|           8|
|Brand#13|  LARGE PLATED BRASS|    41|           8|
|Brand#13|  LARGE PLATED STEEL|    21|           8|
|Brand#13|MEDIUM ANODIZED N...|    21|           8|
|Brand#13|MEDIUM PLATED COPPER|    41|           8|
|Brand#13|STANDARD PLATED N...|    48|           8|
|Brand#14|ECONOMY PLATED NI...|    12|           8|
|Brand#14|ECONOMY POLISHED ...|    41|           8|
|Brand#14|LARGE BURNISHED N...|     7|           8|
|Brand#14|  SMALL POLISHED TIN|     7|           8|
+--------+--------------------+------+------------+
only showing top 20 rows

Time taken to execute query 16 : 7.13 s
```

Execution Time: 7.13 sec

3.2.17 Small-Quantity-Order Revenue Query (Q17)

Statement: This query determines how much average yearly revenue would be lost if orders were no longer filled for small quantities of certain parts. This may reduce overhead expenses by concentrating sales on larger shipments.

Python SQL Query function:

```
def q17():
      query = """
```

```
select
      sum(l_extendedprice) / 7.0 as avg_yearly
from
      LINEITEM,
      PART
where
      p_partkey = l_partkey
      and p_brand = 'Brand#44'
      and p_container = 'WRAP PKG'
      and l_quantity < (
      select
            0.2 * avg(l_quantity)
      from
            LINEITEM
      where
            l_partkey = p_partkey
      );


      """
      spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 17
```

Output :

Execution Time: 6.90 sec

3.2.18 Large Volume Customer Query (Q18)

The Large Volume Customer Query ranks customers based on their having placed a large quantity order. Large quantity orders are defined as those orders whose total quantity is above a certain level.

Python SQL Query function:

```
def q18():
      query = """
select
```

```
      c_name,
      c_custkey,
      o_orderkey,
      o_orderdate,
      o_totalprice,
      sum(l_quantity)
from
      CUSTOMER,
      ORDERS,
      LINEITEM
where
      o_orderkey in (
      select
            l_orderkey
      from
            LINEITEM
      group by
            l_orderkey
      having
            sum(l_quantity) > 314
      )
      and c_custkey = o_custkey
      and o_orderkey = l_orderkey
group by
      c_name,
      c_custkey,
      o_orderkey,
      o_orderdate,
      o_totalprice
order by
      o_totalprice desc,
      o_orderdate
limit
      100;


      """
      spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 18
```

Output :

```
+------+---------+---------+-----------+-----------+---------------+
|c_name|c_custkey|o_orderkey|o_orderdate|o_totalprice|sum(l_quantity)|
+------+---------+---------+-----------+-----------+---------------+
+------+---------+---------+-----------+-----------+---------------+

Time taken to execute query 18 : 9.72 s
```

Execution Time: 9.72 sec

3.2.19 Discounted Revenue Query (Q19)

The Discounted Revenue Query reports the gross discounted revenue attributed to the sale of selected parts handled in a particular manner. This query is an example of code such as might be produced programmatically by a data mining tool.

Python SQL Query function:

```
def q19():
      query = """
select
      sum(l_extendedprice * (1 - l_discount)) as revenue
from
      LINEITEM,
      PART
where
      (
      p_partkey = l_partkey
      and p_brand = 'Brand#52'
      and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
      and l_quantity >= 4
      and l_quantity <= 4 + 10
      and p_size between 1
      and 5
      and l_shipmode in ('AIR', 'AIR REG')
      and l_shipinstruct = 'DELIVER IN PERSON'
      )
      or (
      p_partkey = l_partkey
      and p_brand = 'Brand#11'
      and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
      and l_quantity >= 18
      and l_quantity <= 18 + 10
      and p_size between 1
      and 10
      and l_shipmode in ('AIR', 'AIR REG')
```

```
      and l_shipinstruct = 'DELIVER IN PERSON'
      )
      or (
      p_partkey = l_partkey
      and p_brand = 'Brand#51'
      and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
      and l_quantity >= 29
      and l_quantity <= 29 + 10
      and p_size between 1
      and 15
      and l_shipmode in ('AIR', 'AIR REG')
      and l_shipinstruct = 'DELIVER IN PERSON'
      );


      """
      spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 19
```

Output :

Execution Time: 5.69 sec

3.2.20 Potential Part Promotion Query (Q20)

The Potential Part Promotion Query identifies suppliers in a particular nation having selected parts that may be candidates for a promotional offer.

Python SQL Query function:

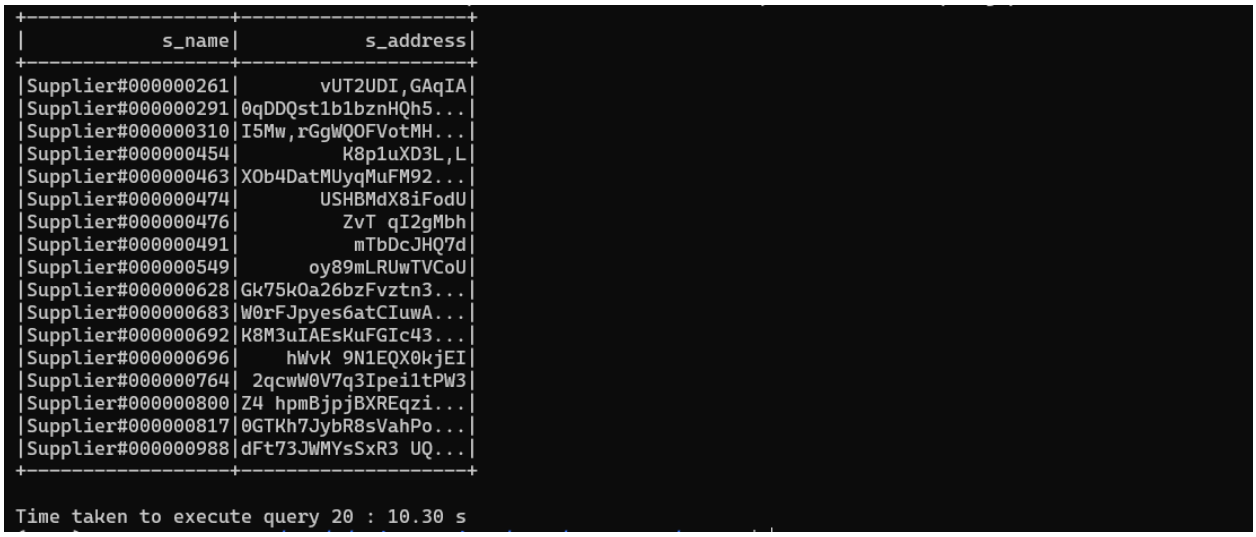
```
def q20():
      query = """
select
      s_name,
```

```
      s_address
from
      SUPPLIER,
      NATION
where
      s_suppkey in (
      select
            ps_suppkey
      from
            PARTSUPP
      where
            ps_partkey in (
                  select
                  p_partkey
                  from
                  PART
                  where
                  p_name like 'green%'
            )
            and ps_availqty > (
                  select
                  0.5 * sum(l_quantity)
                  from
                  LINEITEM
                  where
                  l_partkey = ps_partkey
                  and l_suppkey = ps_suppkey
                  and l_shipdate >= date '1993-01-01'
                  and l_shipdate < date '1993-01-01' + interval '1' year
            )
      )
      and s_nationkey = n_nationkey
      and n_name = 'ALGERIA'
order by
      s_name;



      """
      spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 20
```

Output :



```
+------------------+--------------------+
|           s_name|           s_address|
+------------------+--------------------+
|Supplier#000000261|       vUT2UDI,GAqIA|
|Supplier#000000291|0qDDQst1b1bznHQh5...|
|Supplier#000000310|I5Mw,rGgWQOFVotMH...|
|Supplier#000000454|         K8p1uXD3L,L|
|Supplier#000000463|XOb4DatMUyqMuFM92...|
|Supplier#000000474|       USHBMdX8iFodU|
|Supplier#000000476|         ZvT qI2gMbh|
|Supplier#000000491|          mTbDcJHQ7d|
|Supplier#000000549|      oy89mLRUwTVCoU|
|Supplier#000000628|Gk75kOa26bzFvztn3...|
|Supplier#000000683|W0rFJpyes6atCIuwA...|
|Supplier#000000692|K8M3uIAEsKuFGIc43...|
|Supplier#000000696|    hWvK 9N1EQX0kjEI|
|Supplier#000000764| 2qcwW0V7q3Ipei1tPW3|
|Supplier#000000800|Z4 hpmBjpjBXREqzi...|
|Supplier#000000817|0GTKh7JybR8sVahPo...|
|Supplier#000000988|dFt73JWMYsSxR3 UQ...|
+------------------+--------------------+

Time taken to execute query 20 : 10.30 s
```

Execution Time: 10.30 sec

3.2.21 Suppliers Who Kept Orders Waiting Query (Q21)

This query identifies certain suppliers who were not able to ship required parts in a timely manner.

Python SQL Query function:

```
def q21():
      query = """
select
      s_name,
      count(*) as numwait
from
      SUPPLIER,
      LINEITEM l1,
      ORDERS,
      NATION
where
      s_suppkey = l1.l_suppkey
      and o_orderkey = l1.l_orderkey
      and o_orderstatus = 'F'
      and l1.l_receiptdate > l1.l_commitdate
```

```
      and exists (
      select
            *
      from
            LINEITEM l2
      where
            l2.l_orderkey = l1.l_orderkey
            and l2.l_suppkey <> l1.l_suppkey
      )
      and not exists (
      select
            *
      from
            LINEITEM l3
      where
            l3.l_orderkey = l1.l_orderkey
            and l3.l_suppkey <> l1.l_suppkey
            and l3.l_receiptdate > l3.l_commitdate
      )
      and s_nationkey = n_nationkey
      and n_name = 'EGYPT'
group by
      s_name
order by
      numwait desc,
      s_name
limit
      100;


      """
      spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 21
```

Output :

```
+------------------+-------+
|            s_name|numwait|
+------------------+-------+
|Supplier#000000246|     15|
|Supplier#000000655|     15|
|Supplier#000000599|     14|
|Supplier#000000208|     13|
|Supplier#000000227|     13|
|Supplier#000000301|     13|
|Supplier#000000618|     13|
|Supplier#000000898|     13|
|Supplier#000000094|     12|
|Supplier#000000343|     12|
|Supplier#000000856|     12|
|Supplier#000000159|     11|
|Supplier#000000664|     11|
|Supplier#000000022|     10|
|Supplier#000000038|     10|
|Supplier#000000105|     10|
|Supplier#000000111|     10|
|Supplier#000000502|     10|
|Supplier#000000938|     10|
|Supplier#000000069|      9|
+------------------+-------+
only showing top 20 rows

Time taken to execute query 21 : 14.47 s
```

Execution Time: 14.47 sec

3.2.22 Global Sales Opportunity Query (Q22)

The Global Sales Opportunity Query identifies geographies where there are customers who may be likely to make a purchase.

Python SQL Query function:

```
def q22():
      query = """
select
      cntrycode,
      count(*) as numcust,
      sum(c_acctbal) as totacctbal
from
      (
      select
            substring(
                  c_phone
                  from
                  1 for 2
            ) as cntrycode,
            c_acctbal
      from
            CUSTOMER
```

```
      where
            substring(
                  c_phone
                  from
                  1 for 2
            ) in ('20', '40', '22', '30', '39', '42', '21')
            and c_acctbal > (
                  select
                  avg(c_acctbal)
                  from
                  CUSTOMER
                  where
                  c_acctbal > 0.00
                  and substring(
                        c_phone
                        from
                              1 for 2
                  ) in ('20', '40', '22', '30', '39', '42', '21')
            )
            and not exists (
                  select
                  *
                  from
                  ORDERS
                  where
                  o_custkey = c_custkey
            )
      ) as custsale
group by
      cntrycode
order by
      cntrycode;


      """
      spark.sql(query).show()
```

Shell command to run script:

```
$ python tpch_queries.py 22
```

Output :

```
+---------+-------+-----------------+
|cntrycode|numcust|       totacctbal|
+---------+-------+-----------------+
|       20|     91|666158.0999999997|
|       21|    101|760953.5499999999|
|       22|    106|759804.8099999996|
|       30|     87|646748.0199999999|
+---------+-------+-----------------+

Time taken to execute query 22 : 5.63 s
```

Execution Time: 5.63 sec